User's Guide

BayES™

Bayesian Econometrics Software

Grigorios Emvalomatis

February 21, 2020

Trademarks

Contents

Chapter 1

Getting Started

1.1 Overview

BayES is a software designed for performing Bayesian inference in some popular econometric models using Markov Chain Monte Carlo (MCMC) techniques. Bayesian inference traditionally requires skills and a lot of effort from the part of the researcher, both in terms of mathematical derivations and computer programming. BayES provides canned procedures for Bayesian inference for a set of models, thus avoiding the time-consuming process of deriving complete/full conditionals and coding the samplers.

BayES is primarily menu driven, providing an intuitive user interface that enables first-time users to run a model in a matter of minutes. However, it also features a rich scripting language that allows fast and efficient communication with the program, while facilitating reproducibility of the analysis. Users familiar with other popular matrix languages will find the transition to BayES' scripting language a breeze.

Modern Bayesian inference is based on very computationally intensive techniques. However, MCMC methods are "embarrassingly parallel". BayES has built-in multi-threading support, thus making estimation much faster: when estimating a model, BayES will start as many threads as the number of chains requested by the user. No additional coding or knowledge of multi-threading processing is required.[1]

BayES is not simply another econometrics software package. Its emphasis is in models that are hard or impossible to estimate using classical (frequentist) inference and, thus, it is not a substitute for mainstream and well-established econometric software packages. On the other side of the spectrum, BayES is not competing with general or statistical scripting languages either. BayES is designed for the user who wants to perform Bayesian inference in a computationally involved problem, but who does not want go through the process of learning a new programming language for doing so.

The current version of BayES (version 2.5) supports the following models:

- linear models, including models for cross-sectional and panel data (random effects, random coefficients)
- stochastic frontier models for cross-sectional and panel data, using an array of possible specifications
- discrete-choice models, including models for cross-sectional and panel data and for binary or multi-response dependent variables
- simple models for ordinal data (ordered Probit and Logit)
- simple models for count data (Poisson and negative Binomial)
- type-I and type-II Tobit models
- linear seemingly unrelated regressions
- vector autoregressive models for time-series data

Version 2.5 of BayES runs on both 32-bit and 64-bit Microsoft® Windows® (Windows 7, 8, and 10) and Linux systems[2] and on 64-bit macOS systems. Appendix A provides installation instructions under these systems.

1.2 Running BayES

Once successfully installed, BayES can be used in two modes: (i) interactive mode using its GUI facilities, and (ii) batch mode by calling BayES from the command line and providing a script file for execution. Instructions on using BayES in each of these modes are presented separately in the following two sections. See also the video tutorials on BayES' website.

[1] Future development plans include taking advantage of Graphical Processing Units (GPUs), which are present in most personal computers. Such a feature could increase speed by orders of magnitude.

[2] BayES has been tested on CentOS 7, Fedora 28, Debian 9.4, Ubuntu 18.04, openSUSE 15, and Linux Mint 19. It may be able to run on other distributions as well, but use at your own risk.

1.2.1 Running BayES in interactive mode

BayES can be initiated in interactive mode by (i) clicking on the BayES icon on the system's 'start' menu or on the Launchpad (under macOS), (ii) double-clicking on the BayES icon on the system's desktop (iii) typing `BayES` and hitting the return key on the system's console window, or (iv) double-clicking on the BayES link file in the installation directory, as it was provided during installation. Note that the first three options may be unavailable, depending on the options selected during installation. However, the fourth option will always be available.

When BayES starts in interactive mode the main window will appear. This window contains:

- the BayES console, where output will be printed once instructions are submitted by the user
- the datasets pane, which will contain a list of datasets held in memory, once any datasets are imported or created
- the models pane, which will contain a list of models held in memory, once any models are estimated and their results stored
- the matrices pane, which will contain a list of matrices held in memory, once any matrices are defined by the user
- the main menu on the upper-left corner of the window, which provides the main way of interacting with BayES when using the GUI (importing data, estimating models, plotting, etc.)
- the main window toolbar, located right below the main menu, which provides quick access to some frequently used functions, such as importing data, saving and loading workspaces or interrupting the execution of a procedure
- the status bar at the bottom of the main window, which displays the current working directory as well as a progress bar that will display information on the progress of computationally intensive procedures

An additional window, the BayES script editor, which does not appear automatically when BayES starts, can be shown from the main menu: `View→Script Editor`. This window can be used to type code in BayES' scripting language and submit instructions in bulk.

Once the user submits instructions, either via the GUI or the script editor window, these instructions are interpreted and the appropriate actions executed. During this time most GUI functions will not be accessible. Upon completion of execution control returns to the user, who can continue with submitting further instructions.

The first time that BayES starts in interactive mode the working directory is set to the installation directory, as provided during installation. This directory will be used as the starting point for importing data and saving output (datasets, matrices, plots, etc.) and looking for user-defined functions. Depending on the options chosen by the user (from `Help→Preferences` under Microsoft® Windows® and Linux systems and from `Preferences` under macOS systems in the main menu), this directory may change when BayES is subsequently invoked in interactive mode.

1.2.2 Running BayES from the command shell

If BayES is made globally available from the command line[3] then BayES can also be run in batch mode. This is achieved by submitting a command of the following form on the system's command console (`cmd` under Microsoft® Windows® or `bash` under Linux and macOS):

```
BayES <input file> [<output file>]
```

where:

[3] This can be achieved by checking the relevant checkbox during installation.

- <input file> is a BayES script file that contains statements written in the BayES language (see Chapter 2 for more details). <input file> should be provided as the name of the script file (including its extension), prepended by the path to the file, either in absolute terms or relative to the command console's current directory.

- <*output file*> is an optional argument that defines a file to which any output from the BayES console is redirected. As before, <*output file*> should be provided as the name of the file (including its extension), prepended by the path to the file, either in absolute terms or relative to the command console's current directory. If such a file does not exist then BayES will create it. If the file exists its contents will be replaced.

Once a command of this form is submitted to the system's command console, BayES will fire-up without displaying its GUI and it will start executing the commands contained in <input file>. If an <*output file*> is provided, output from the BayES console is directed to this file; otherwise output is printed on the system's console window. Note that BayES does not distinguish between normal and error output and everything is printed on the normal channel. Upon completion of the execution of the statements contained in <input file> or upon encountering an error, BayES terminates and control is passed back to the system's command console.

1.3 Quick start for the impatient: video tutorials

Video tutorials that can be found on the BayES website illustrate how to use BayES, either in interactive or batch mode. These tutorials will have a first-time user running models in a matter of minutes and they cover basic, as well as more advanced topics:

1. An Overview of the BayES™ Desktop
2. Importing, Summarizing and Plotting Data
3. Running Models from the GUI
4. Working with Code: The BayES™ Script Editor
5. Working with BayES™ from the Command Shell
6. Writing and Executing BayES™ Functions

1.4 How to use this document

The following chapter provides a quick overview of the BayES language and it is recommended that all users go through it. Chapter 3 documents the way external programs can be accessed from BayES and is recommended to users who plan to use these interfaces. From that point onwards the chapters cover specific models and need not be studied in the order presented in this document: every chapter and, to a large extend, every section within a chapter is self contained. Documentation of every model starts with a mathematical representation, continues with the analysis of syntax of the relevant command, describes the results presented and stored from the model and concludes with code examples. All these models can also be accessed from the GUI, but this is not documented here. Appendix B documents all the functions available in BayES, except the model estimation functions, which are documented in the relevant chapters.

This document is not a substitute for a textbook on Bayesian econometrics. It is assumed that the user is familiar with the basics of Bayesian inference at the level presented, for example, in Greenberg (2013), Koop (2003), or Lancaster (2004). "Bayesian Econometrics using BayES™" is an early draft of a book on Bayesian econometrics, which is associated with BayES and uses examples that employ BayES for estimation. This is made available under a Creative Commons license and can be used as an introduction to Bayesian econometrics, as well as a "soft" introduction to the software itself.

Chapter 2

The BayES Language

2.1 Introduction

Although full functionality is available via the Graphical User Interface (GUI), oftentimes it may become cumbersome and time consuming to submit commands via this route. For example, it happens very often in practice that one needs to add one extra independent variable to a model previously estimated, leaving all other model parameters unchanged. In such a case, including the additional variable in the model can be done by a few keystrokes rather than a series of mouse clicks. Furthermore, creating and deleting variables or otherwise transforming data may involve repetitive tasks that can be accomplished much faster through writing code, rather than via the menu.

To address these issues, BayES features a programming language that allows making adjustments to previously submitted commands and data processing much more efficient. That said, someone fluent in the BayES language can dispense of the graphical environment altogether and work only using code.

The BayES language is very intuitive and users of software packages that work with matrices or arrays will find the transition to programming in BayES' language very easy. Furthermore, to facilitate use of the language, every time a command is submitted through the GUI, the first thing that is printed on the BayES console is the code that accomplishes the tasks requested by the user.[1]

2.1.1 Writing and submitting code

The easiest way to write and submit code for execution to BayES is by invoking the script editor from the menu: `View→Script Editor`.[2] The window that pops up is a simple text editor designed to highlight code that is written in the BayES language. There are three equivalent ways of submitting code for execution via the graphical user interface:

1. via the script editor's menu: `Run→Run`
2. by clicking the `Run` button on the script editor's toolbar (below the window's menu)
3. by hitting the `Ctrl`/`Cmd` and `R` keys simultaneously (`Ctrl+R`)

Any of these procedures will submit for interpretation and execution all the contents of the active script tab in the editor window. Alternatively, if you wish to submit only part of the code in a script file, highlight the part of the code you want to execute before submitting it, using any of the three ways mentioned above. Script files can be saved on the computer's hard disk, typically using a `.bsf` (BayES script file) extension.

BayES can also be invoked from the command line without starting up the graphical environment.[3] In this context, BayES can execute the commands contained in a script file and either print output on the system console or redirect output to another text file.

2.1.2 Using the sample files that come with BayES

BayES ships with a number of sample files. During installation these files are extracted in a folder named `Samples` under the BayES installation directory. These files contain sample code that is typically much more complete than the examples found in this guide. They contain many comments and are self-explanatory. To run the sample code open them from the script editor window and click on the `Run` button or hit `Ctrl+R`.

Note that many of these files contain statements like `setwd("$BayESHOME");`. This statement sets the working directory to the BayES installation directory, so that it can call functions and load data from files that are located in the folders created during installation.

[1] In fact, the only role for the GUI in BayES is to prepare code based on user input. Once a command is submitted via the GUI (typically by clicking on an "OK" or "RUN" button), the associated code is printed on the console and then interpreted, performing the requested task.

[2] See also the video tutorial "Working with Code: The BayES™ Script Editor".

[3] See section 1.2.2 and the video tutorial "Working with BayES™ from the Command Shell".

2.2 Arithmetic operations

In its simplest form BayES can be used as a desk calculator, which has the ability of storing values to user-defined variables. In this context, BayES supports:

1. the basic arithmetic operations (addition, subtraction, multiplication, division)
2. unary minus
3. exponentiation
4. functions for taking logarithms, square roots, etc.
5. parentheses to group expressions

An example of using BayES as a calculator is given in Example 2.1. This example also illustrates that any line starting with a double slash is treated by BayES as a comment. Multi-line comments are enclosed in matching "`/*`" and "`*/`", as shown in Example 2.4.

■ **Example 2.1**

▼ Input

```
// Define an item with id value A and
// print it on screen
A = 3;
print(A);

// Define an item with id value B and
// print it on screen
B = 2*A - 4;
print(B);

// Define an item with id value C and
// print it on screen
C = 2*(A+3)^2 - 5/2*log(B);
print(C);
```

▼ Output

```
A =
  3


B =
  2


C =
  70.2671
```

The term "id value" is used in this guide to denote the name of an item held in memory. In the example above three values were defined with names `A`, `B` and `C`. These names (id values) are then used to refer to the numerical values stored in the respective items.

2.3 Matrices and matrix calculations

2.3.1 Defining and using matrices

Matrices can be defined using square brackets. Example 2.2 shows how this is accomplished. It also shows how to define matrices using other matrices as blocks and how to perform basic arithmetic operations on matrices.

2.3.2 Indexing matrices and the range operator

The entries of matrices can be accessed using parentheses after the matrix id value. For example, if `A` is a matrix currently in memory, the statement:

```
b = A(3,4);
```

will take the element of `A` located in the 3rd row, 4th column and store it in an item with id value `b`.

When you want to access consecutive entries in a matrix you have to use the range operator, ":". When the range operator is preceded and followed by positive integers, `i` and `j`, respectively, with `i` < `j`, it is taken to mean "*from* `i` *to* `j`". For example, the statement:

```
C = A(2:3,4);
```

■ **Example 2.2**

▼ Input

```
// A is a 2x3 matrix
A = [1, 2, 3;
     4, 5, 6];
print(A);

// White space is irrelevant. All
// that matters is that columns are
// separated by "," and rows by ";".
B = [1,2,3; 4,5,6];
print(B);

// B is equal to A
print(A-B);

// Arithmetic operations on matrices
C = 3*A - 2*(B+A);
print(C);

// Take the transpose of C
D = C';
print(D);

// Create E by defining its blocks:
// 1 -> a 3x1 vector
// 2 -> a 3x2 matrix
E = [ [1;2;3], 2*D ];
print(E);
```

▼ Output

```
A =
  1  2  3
  4  5  6


B =
  1  2  3
  4  5  6


A-B =
  0  0  0
  0  0  0


C =
 -1  -2  -3
 -4  -5  -6


D =
 -1  -4
 -2  -5
 -3  -6


E =
    1   -2   -8
    2   -4  -10
    3   -6  -12
```

will take the entries of A located in rows 2 to 3, column 4 and store them in a 2×1 matrix C.

When the range operator is not preceded and not followed by integers, it is taken to mean "*all items in the respective dimension*". For example, the statement:

```
D = A(:,4);
```

will take the entire 4^{th} column of A and store it in a matrix (column vector) D.

Finally, when a single range operator is used to index the elements of a vector or matrix, A, then all entries of A are requested. That is, all three statements:

```
D = A;
D = A(:,:);
D = A(:);
```

are equivalent when A is a matrix, and they create a copy of A into a matrix with id value D.

Example 2.3 puts all these together, while Example 2.4 shows how indexing is performed and how the range operator can be used in the left-hand side of assignment statements to alter a block of a matrix already in memory. It also shows how to copy the entries of a matrix A into another matrix B, respecting the dimensions of the left-hand-side matrix.

2.3.3 Element-wise operators

Matrix addition and subtraction work, as expected, in an element-wise fashion: if $\mathbf{A}$ and $\mathbf{B}$ are two $M\times N$ matrices and $\mathbf{C} = \mathbf{A} + \mathbf{B}$, then $\mathbf{C}$ is defined such that $c_{ij} = a_{ij} + b_{ij}$, $\forall i = 1,\dots M, \quad j = 1,\dots N$. Matrix-matrix multiplication, again as expected, works in a non-element-wise fashion: if $\mathbf{A}$ is an $M\times K$ matrix, $\mathbf{B}$ is a $K\times N$ matrix and $\mathbf{C} = \mathbf{A}\cdot\mathbf{B}$, then $\mathbf{C}$ is an $M\times N$ matrix such that:

$$c_{ij} = \sum_{k=1}^{K} a_{ik}\cdot b_{kj} \quad \forall i = 1,\dots M, \quad j = 1,\dots N$$

■ **Example 2.3**

▼ Input

```
// Define A as a 4x4 matrix
A = [ 1, 2, 3, 4;
     -1,-2,-3,-4;
      8, 7, 6, 5;
     -4,-5,-6,-8
    ];

// b is the entry of A in row 3,
// column 4
b = A(3,4);
print(b);

// Print the elements in rows 2 to 3,
// column 4
print(A(2:3,4));

// Print the 4th column of A
print(A(:,4));

// Print the 1st row of A
print(A(1,:));
```

▼ Output

```
b =
  5


A(2:3,4) =
  -4
  5


A(:,4) =
  4
  -4
  5
  -8


A(1,:) =
  1  2  3  4
```

■ **Example 2.4**

▼ Input

```
/* Define A as a 3x2 matrix of random
draws from an exponential distribution
with rate 2.5 and print it */
A = exprnd(2.5,3,2);
print(A);

// Make the entry in row 2, column 1
// of A equal to one and print A
A(2,1) = 1;
print(A);

// Make the first row of A equal to a
// vector of zeros
A(1,:) = zeros(1, cols(A));
print(A);

// Define B as a 3x2 matrix of zeros
// and assign the entries of A to it
B = zeros(2,3);
B(:) = A;
print(A);
print(B);

/* A and B have different dimensions
but the same number of entries. The
entries of A are stored in B in a
column-major fashion */
```

▼ Output

```
A =
   0.20604   1.10341
  0.170939  0.142362
  0.535967  0.360443


A =
   0.20604   1.10341
         1  0.142362
  0.535967  0.360443


A =
         0         0
         1  0.142362
  0.535967  0.360443


A =
         0         0
         1  0.142362
  0.535967  0.360443


B =
         0  0.535967  0.142362
         1         0  0.360443
```

However, very frequently in practice, one needs to do element-wise multiplication, division or exponentiation of matrices. As it is common in other matrix languages, BayES defines the following element-wise operators:

- ".*" for element-wise multiplication
- "./" for element-wise division
- ".^" for element-wise exponentiation

Example 2.5 shows how these operators can be used.

Example 2.5

▼ Input

```
// Define matrices
A = [ 1, 2;
      3, 4];
B = [ 5, 4;
      2, 1];

// Element-wise multiplication
print(A.*B);

// Element-wise division
print(A./B);

// Element-wise exponentiation
print(A.^B);
```

▼ Output

```
A.*B =
  5  8
  6  4


A./B =
  0.2  0.5
  1.5    4


A.^B =
   1  16
   9   4
```

2.3.4 Operator precedence

Arithmetic operations contained in an expression are evaluated in the following order:

1. expressions inside parentheses – `(...)`
2. transposition (′), scalar and element-wise exponentiation (`^`, `.^`)
3. unary minus (-)
4. matrix and element-wise multiplication (`*`, `.*`), scalar and element-wise division (`/`, `./`)
5. addition (`+`) and subtraction (-)

Operators with equal precedence are evaluated from left to right.

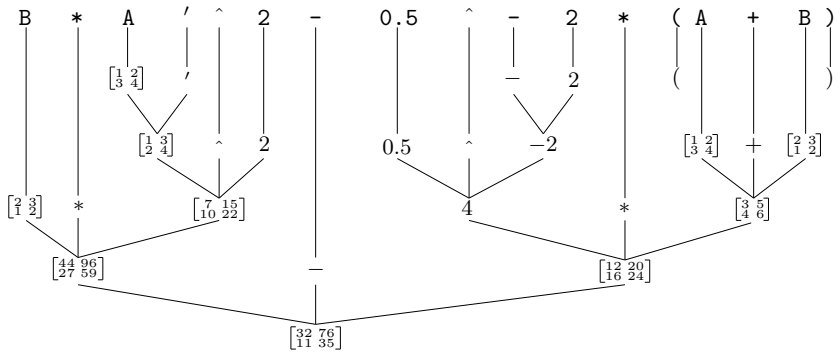
As an example, consider the expression:

```
B*A'^2 - 0.5^-2*(A+B)
```

where `A` and `B` are 2×2 matrices:

$$\texttt{A} = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \qquad \text{and} \qquad \texttt{B} = \begin{bmatrix} 2 & 3 \\ 1 & 2 \end{bmatrix}$$

The expression above is evaluated in the following way:



2.3.5 Functions operating on matrices

BayES provides an array of functions that take matrices as arguments and return matrices. These include:

- special matrix functions like `inv()`, `trace()`, `diag()`, etc.
- functions that provide generalizations of scalar functions like `log()`, `exp()`, `sqrt()`, etc. to matrices and work in an element-wise fashion
- summing, rounding and descriptive-statistics functions like `sum()`, `floor()`, `mean()`, etc.
- decompositions of matrices like `chol()` and `eig()`

Detailed descriptions of these functions are given in Appendix B.

2.4 Datasets and data transformations

The fundamental data type in BayES is the matrix and BayES' language is designed to work primarily with matrices. However, to facilitate statistical analysis and reporting of results BayES uses an additional type that can store data: the dataset. For practical purposes, a dataset in BayES is a special type of a matrix with each one of its columns storing data on a particular variable and having an associated name; the variable's name.

Typically datasets are defined by importing data into BayES using a statement like:

```
myData = import("c:/myFiles/mydata/dataset1.csv", ",");
```

This statement reads the data from the csv file `dataset1.csv`, located at `"c:/myFiles/mydata"`, into a dataset with id value `myData` and using comma as the field separator. Each column of `dataset1.csv` must contain the data on a variable, with the first row specifying the variable names.

Datasets can also be constructed in BayES by turning a matrix into a dataset. For example, if `X` is a matrix in the current workspace, then the statement:

```
myData = dataset(X);
```

will create a dataset with id value `myData` by treating each column of `X` as a variable. Because variables in datasets must have an associated name, this statement will name the variables in `myData` as `_V1`, `_V2`, etc. Alternatively, variable names can be provided as an optional argument to the `dataset()` function or the variables in `myData` can be renamed using the `rename()` function after the dataset has been constructed (see section B.13 for details).

Once a dataset item is constructed then the functions operating on datasets can be used. These functions include printing summary statistics, creating, deleting and renaming variables, sorting data, etc. and are documented in B.13. Sample `"2-Working with datasets.bsf"`, located at `"$BayESHOME/Samples/1-GettingStarted"`, demonstrates many of BayES' capabilities in working with datasets.

Because datasets in BayES are simply special matrices, all operations and functions defined for matrices work also on datasets. Importantly, the indexing methods described in section 2.3.2 can be used to extract or alter the values stored in a dataset. For example, if `myData` is a dataset with three variables named `var1`, `var2` and `var3`, then the statement:

```
X = myData(:,3);
```

will construct a matrix `X` that stores the values of `var3` (third column of the dataset). Variables in a dataset can also be referred to by their name by using the '.' operator. For example, an equivalent way of constructing `X` from the values of `var3` would be:

```
X = myData.var3;
```

The '.' operator can also be used to define new variables for a dataset. Continuing working with the `myData` dataset, the following statement creates a new variable named `logvar2` with values equal to the natural logartihm of `var2`:

```
myData.logvar2 = log(myData.var2);
```

Indexing and range operations can be used in the left-hand side of assignment statements involving datasets as well. For example, the statement:

```
myData(1:3,1) = zeros(3,1);
```

makes the first three rows of the first variable in `myData` equal to zero. As long as `var1` is the first variable in `myData`, this statement is equivalent to:

```
myData.var1(1:3) = zeros(3,1);
```

where, instead of using a column index in the left-hand side of the statement, the column of the dataset is referenced by the variable name.

When mathematical operations or operations that involve indexing are performed repeatedly on a dataset (for example within a loop), it is usually faster to turn the dataset into a matrix, perform the operations on the matrix and then turn the final matrix back into a dataset. For example, the following statements:[4]

```
X = myData;
for (i=1:rows(myData))
    X(i,1) = i;
end
myData = dataset(X, {var1,var2,var3});
```

store the data in `myData` into a matrix `X`, alter the values in the first column of `X` using a loop and replace the original dataset with a new one, constructed from the values in `X`. The same task could be accomplished by:

```
for (i=1:rows(myData))
    myData(i,1) = i;
end
```

but the second set of statements could be slower, especially if `myData` contains many observations.

2.5 Models

The model data type is used to store the results obtained from estimating the parameters of any model available in BayES. For example the statement:

```
myModel = lm( y ~ x1 x2 x3 );
```

runs a simple linear model, where `y` is the dependent variable and `x1`, `x2` and `x3` the independent variables, and prints a summary of the results on the BayES console. It also stores the results into a model item with id value `myModel` and these results become available for further processing (model comparison, testing restrictions on parameters, plotting or exporting the draws from the posterior, etc.). If the statement above is submitted without providing a left-hand-side value, i.e. if it were submitted simply as:

```
lm( y ~ x1 x2 x3 );
```

then BayES would still run the model and print the results, but these results would not be available for further processing.

If `myModel` is a model in memory in the current workspace then the statement:

```
print(myModel);
```

prints a summary of the results from the model on the BayES console. The statement:

```
who(myModel);
```

prints a list of the elements stored in `myModel`. These elements always include the values of the Gibbs-sampler parameters (number of chains, number of burn-in draws per chain, total number of retained draws from the posterior, value of the thinning parameter and the seed

[4] See section 2.9 for a general discussion on program flow and subsection 2.9.3 for a description of **for** loops.

for the random-number generator), the value of the log-marginal likelihood using the Laplace approximation (Lewis & Raftery, 1997) and the draws from the posterior distribution for each parameter. All these elements can be accessed using the '.' operator. For example, the statement:

```
print(myModel.logML);
```

will print the value of the log-marginal likelihood of `myModel` on the BayES console, while the statement:

```
samples_beta1 = myModel.x1;
```

will create a vector with id value `samples_beta1` that contains the values of the draws for the parameter associated with variable `x1`.

Many different types of models can be estimated in BayES by using different model-creating functions. For example, the `lm()` function estimates a linear model, the `sf_re()` function estimates a stochastic-frontier model with random effects and the `probit()` function estimates a Probit model. All model-creating functions take the model specification as their (required) first argument. The model specification is typically given in the form:

```
<dependent variable> ~ <independent variables>
```

and an example of was this given at the beginning of this section. The model-creating functions also take a list of optional arguments, some of which are common to all functions and some being available only to specific models. These optional arguments always come in option-value pairs (eg. `"chains"=3`, `"draws"=10000`, etc.) and the order in which they are provided to the function does not play a role. That is, the following two statements are equivalent:

```
myModel = probit( y ~ x1 x2 x3, "chains"=3, "draws"=10000 );
myModel = probit( y ~ x1 x2 x3, "draws"=10000, "chains"=3 );
```

The meaning of both common and model-specific optional arguments and their default values is documented analytically in the following chapters, under each specific model. It is worth noting here, however, that the common optional arguments for all functions are:

- the values of the Gibbs sampler parameters (number of chains, number of burn-in and retained draws per chain, and the values of the thinning parameter and the random-number generator seed)
- a logical argument (its value could be either **true** or **false**) that indicates whether the Chib (1995) or Chib & Jeliazkov (2001) approximation to the log-marginal likelihood should be calculated; depending on the type of model, these calculations may be as demanding as the original estimation of the model

Because a model item stores the draws from the posterior, it may use much of the machine's memory. It is, therefore, advisable to delete models from memory (using the **clear()** function) if the results from the model are no longer needed.

Sample `"3-First models.bsf"`, located at `"$BayESHOME/Samples/1-GettingStarted"` provides a first exposure to estimating models in BayES. The samples located at `"$BayESHOME/Samples/2-Models"` illustrate how all types of supported models can be estimated and their results analyzed.

2.6 Structures

A structure in BayES is a data type used to group together different data types into a single element. For example, it may be convenient to bundle together a set of a matrices and strings, assign an id value to the bundle and use this id value to refer to it in subsequent statements. As an example, suppose that `A` is a matrix, `myModel` a model and `myData` a dataset, all of them being in memory in the current workspace. The statement:

```
S = { e1 = A, m = myModel, d = myData };
```

defines a structure `S` with three elements, the id values of which are `e1`, `m` and `d`. The elements of `S` can be accessed using the '.' operator. For example, the statement:

```
print(S.m);
```

will print on the BayES console the results from the model that has an id value `m` within structure `S`. The list of elements contained in a structure is printed on the BayES console using the **who()** function:

```
who(S);
```

The elements of `S` are copies of the original elements and have their own life: altering the entries of `S.e1` in the example above will not affect matrix `A` in the current workspace. Structures can have other structures as elements and can be infinitely nested.

Structures can be used within a script to make code more succinct. Simple examples of using structures can be found in sample `"5-Working with structures.bsf"`, which is located at `"$BayESHOME/Samples/1-GettingStarted"`. Although optional in most cases, structures must be used to communicate with programs external to BayES. Initial values are passed to JAGS and OpenBUGS (see Sections 3.2 and 3.3) in the form of structures and the samples located at `"$BayESHOME/Samples/3-JAGS-OpenBUGS"` demonstrate their use in this context. Other interface functions provided by BayES (see Sections 3.5 to 3.8) return structures after completion and their use is demonstrated in the samples located at `"$BayESHOME/Samples/5-Interfaces"`.

2.7 Strings

Strings of characters constitute another data type in BayES. Strings are defined as words or phrases enclosed in double quote marks and they can be assigned id values, which can be subsequently used to refer to them. These can be concatenated or substrings extracted from them. Strings are used primarily for printing information on the BayES console or when working with directory statements. Example 2.6 demonstrates many of BayES' capabilities in working with strings. The full list of functions that operate on strings is given in section B.15.

■ **Example 2.6**

▼ Input

```
// Define two strings, s1 and s2
s1 = "Hello ";
s2 = "World!";

// Concatenate s1 and s2 and print the
// result on the console
s = strcat(s1,s2);
print(s);

// Print characters 1 to 5 from s
print(substr(s,1:5));

// Compare two strings
r1 = "BayES strings are ";
if ( strcmp(s2,"WORLD!") )
    r2 = "not case sensitive ";
else
    r2 = "case sensitive ";
end
print(strcat(r1,r2)) ;
```

▼ Output

```
Hello World!
Hello
BayES strings are case sensitive
```

2.8 Data types and assignments

The previous five sections describe the five data types used by BayES:

1. matrices
2. datasets
3. models
4. structures
5. strings

When using assignment statements, BayES infers the type of the item defined from the right-hand side expression. For example, the statement:

```
s = "Hello from BayES!";
```

defines a string with id value `s` and assigns the value `"Hello from BayES!"` to it. The user does not have to declare `s` as a string before assigning a value to it: BayES infers that `s` is a string because the right-hand side expression, `"Hello from BayES!"`, is a string. All functions that operate on strings can be used on `s`.

Suppose that a subsequent statement in the same script is:

```
s = [3;2];
```

This statement redefines `s` as a 2×1 vector and all functions that operate on matrices can be used with `s`. Furthermore, BayES deletes the value that `s` had when it was a string from memory without issuing any error or warning. This means that the same id value can be used in a script to represent different data types at different points. At every point, however, an id value can be associated with a single data type and previous definitions are cleared from memory.

2.9 Program flow

Until now this document covered code where the program control flows sequentially through the statements in a script file: statements are interpreted and executed one-by-one in the order they are defined in the script and the program ends when the last statement finishes executing. Oftentimes in practice it is necessary to execute a statement repeatedly or if a condition is satisfied. In these cases program control needs to jump from one point to another and execute statements in a non-linear fashion.

BayES provides three ways of controlling the program flow:

1. **if-else** blocks where a block of statements are executed if a condition is true
2. **for** and **while** loops where a block of statements are executed repeatedly for a given number of times or while a condition is true
3. user-defined functions

User-defined functions are covered in detail in section 2.10. The rest of this section covers conditional execution and loops, after first defining boolean operators.

2.9.1 Boolean expressions and operators

A boolean expression is a logical statement that could be either true or false. For example:

- `1<3` is true
- `1>=3` is false
- `x==3` is true or false, depending on whether `x` was defined to be equal to 3 or not[5]

Single boolean expressions, like the ones mentioned above, can be combined to form larger expressions using boolean operators. Boolean operators are nothing more than symbols that are used to convey the meaning of truth functionals in formal logic. Suppose that we have two boolean expressions `A` and `B`. Table 2.1 shows how these two expressions can be combined using boolean operators and the truth values of the resulting composite expression, depending on the truth values of the two original expressions.

[5]Notice that when testing for equality between two values, a double equality, '`==`', is used. This is because the single equality, '`=`', is reserved for assignments and a statement like `x=3` would assign the value 3 to `x`, instead of testing whether `x` is equal to 3.

Truth value of `A`	Truth value of `B`	Logical AND `A & B`	Logical OR `A | B`	Logical NOT `~A`
TRUE	TRUE	TRUE	TRUE	FALSE
TRUE	FALSE	FALSE	TRUE	FALSE
FALSE	TRUE	FALSE	TRUE	TRUE
FALSE	FALSE	FALSE	FALSE	TRUE

Table 2.1: Logical operators and their syntax in BayES

Logical operators in BayES are evaluated from left to right, but parentheses can be used to group composite boolean expressions and alter the order of evaluation. For example if `A` is true and `B` false:

- `~ A & B` evaluates to false
- `~ (A & B)` evaluates to true

2.9.2 if-else statements

if-else statements are used to execute a block of code conditional on the truth value of a boolean expression. An **if-else** statement has the following general form:

```
if ( <boolean expression> )
    <list of statements>
    // these will be executed only if the
    // boolean expression evaluates to true
else
    <list of statements>
    // these will be executed only if the
    // boolean expression evaluates to false
end
```

Note that if there is nothing to be executed if the boolean expression evaluates to false, then the **else** part can be omitted altogether:

```
if ( <boolean expression> )
    <list of statements>
end
```

Example 2.7 demonstrates the use of the **if-else** statement. Sample `"6-Program flow.bsf"`, located at `"$BayESHOME/Samples/1-GettingStarted"` contains some more complete examples of the statement.

■ **Example 2.7**

▼ Input

```
// Define x as a scalar equal to 3
x = 3;

// Print a message to the console
// conditional on the value of x
if (x==3)
    print("x is equal to 3");
else
    print("x is equal not to 3");
end
```

▼ Output

```
x is equal to 3
```

2.9.3 for loops

for loops are used to repeatedly execute a block of statements for a given number of times. A **for** statement has the following general form:

```
for ( <loop index> = <lower value>:<upper value> )
    <list of statements>
    // these will be executed repeatedly
end
```

In this statement the loop index is an integer which, upon execution, will be set equal to the lower value specified in the first line of the statement. During execution the list of statements will be executed and the loop index value increased by one, until it reaches the upper value specified in the first line of the statement.

Example 2.8 demonstrates the use of the **for** statement. Sample "6-Program flow.bsf", located at "$BayESHOME/Samples/1-GettingStarted" contains some more complete examples of the statement.

■ **Example 2.8**

▼ Input

```
// Print the value of the loop
// index, i, within the loop
for (i=1:3)
    print(i);
end
```

▼ Output

```
i =
  1


i =
  2


i =
  3
```

2.9.4 while loops

while loops are used to repeatedly execute a block of statements while a condition is true. A **while** statement has the following general form:

```
while ( <boolean expression> )
    <list of statements>
    // these will be executed repeatedly
end
```

During execution the boolean expression will be evaluated and, if the result is true, the list of statements inside the block will be executed repeatedly. The block of statements should make changes to items appearing in the boolean expression in every iteration, otherwise the expression will never evaluate to false and the program control will never jump out of the loop.

Example 2.9 demonstrates the use of the **while** statement. Sample "6-Program flow.bsf", located at "$BayESHOME/Samples/1-GettingStarted" contains some more complete examples of the statement.

■ **Example 2.9**

▼ Input

```
// Print the value of i within a while
// loop
i = 4;
while (i>0)
    print(i);
    i = i - 1;
end

// It is important to change the value
// of i inside the loop, otherwise the
// program will never leave it
```

▼ Output

```
i =
  4


i =
  3


i =
  2


i =
  1
```

2.10 User-defined functions

BayES supports user-defined functions. These functions are defined in BayES script files and can take multiple arguments and return multiple values. User-defined functions are called from script files or other functions and can be used to perform a particular task that may require many lines of code, helping to keep the code in the main script tidy. Also, in cases where a task needs to be performed repeatedly, it is easier to describe the task in a user-defined function and to call this function at the appropriate places in the script file.

Each user-defined function must be declared and implemented in its own BayES script file. These special files have a specific form. For example, suppose that we want to define a function named `myFunction` that takes a list of arguments, `ar1, ar2, ...`, uses these argument to do some calculations and defines a list of return values, `rv1, rv2, ...`, which will be available after the function completes its job. The following code should be put in a BayES script file called `myFunction.bsf` and this file saved in the current working directory:

```
function [rv1, rv2, ... ] = @myFunction(ar1, ar2, ...)
    <list of statements>
    // these statements will use ar1, ar2, ..., and define
    // rv1, rv2, ..., which will be available in the calling
    // workspace once the function completes its job
end
```

Functions are not meant to be executed independently, but to be called from other scripts (or other functions). A statement like:

```
[y1, y2, ... ] = @myFunction(x1, x2, ...);
```

placed within a BayES script represents a call to the user-defined function `myFunction`. When such a calling statement is encountered the following process takes place:

1. the arguments `x1, x2, ...` passed to `myFunction` in the calling statement are copied and made available in the function's workspace as `ar1, ar2, ...`[6]
2. program control jumps to the user-defined function `myFunction`, where the listed statements are executed and the return values, `rv1, rv2, ...`, are assigned
3. program control jumps back to the calling script and the function's return values are made available to the workspace that contains the calling statement as `y1, y2, ...`

The directory `"$BayESHOME/Samples/4-Functions"` contains samples of simple and not-so-simple functions. Here we will describe how to define and use a simple function called `SimpleFunction`, which takes a single argument `x` and returns `2*x`. Suppose the file `SimpleFunction.bsf` is located in the current working directory and it contains the following code:

```
function [y] = @SimpleFunction(x)
    // Print a message on the console
    print("program control is now in @SimpleFunction");

    // Print the value of x on the console
    print(x);

    // Assign the value 2*x to y
    y = 2*x;

    // Print a message on the console
    print("program control is about to leave @SimpleFunction");
end
```

The first statement in the function simply notifies the user that program control has passed to `@SimpleFunction` by printing a message on the BayES console. The next statement prints the value of `x`, as it is defined in the workspace of `@SimpleFunction`. The third statement assigns

[6] Each function has its own workspace and it cannot access any elements defined in the workspace of the calling script (see also Section 2.10.1).

the value `2*x` to `y` and the last statement, again, prints information on the BayES console that program control leaves `@SimpleFunction`.

The code in `SimpleFunction.bsf` cannot be executed simply by hitting Ctrl+R on the script editor window: the function needs to be called from another script. This is demonstrated in Example 2.10.

■ **Example 2.10**

▼ Input

```
// Define a 1x2 matrix z using draws from an exponential distribution
z = exprnd(2.5, 1, 2);

// Call "@SimpleFunction" and assign its return value to w
w = @SimpleFunction(z);

// Print a message on the console to indicate that @SimpleFunction finished
print("\n\nControl is back to the calling script");

// Print the value of w from the calling script
print(w);
```

▼ Output

```
Program control is now in @SimpleFunction
x =
   0.5597008   0.23834727

Program control is about to leave @SimpleFunction

Control is back to the calling script
w =
   1.1194016   0.47669454
```

2.10.1 Rules for defining and calling functions

The following rules apply when defining and calling functions:

1. each function must be defined in its own BayES script file
2. the keyword **function** must be the first word in the script file that defines the function
3. the function name must be the same as the file name of the BayES script file that contains its definition, prepended by '`@`'
4. the function definition ends with the **end** keyword; everything that follows this keyword in the script file that defines the function is ignored
5. when calling a function the BayES script file that contains its definition must be in the current working directory
6. functions can have zero, one, or multiple arguments of different types (matrices, datasets, strings, etc.)
7. the order in which arguments are passed to a function matters
8. functions can have zero, one, or multiple return values of different types (matrices, datasets, strings, etc.); if the function returns only one value then the square brackets in the definition and calling statements can be omitted
9. the order in which return values are assigned to elements in the workspace of the calling script matters
10. functions have their own workspace and arguments are passed by making copies: functions are aware only of items that have been passed to them as arguments and cannot access items in the workspace of the calling script[7]

[7] If the workspace of the calling script has an item with id value `x` and a function is called, which defines

2.11 Plotting

BayES provides functions for producing elementary graphics. The following types of plots are supported:

1. histograms using the `hist()` function
2. scatter plots (y versus x) using the `scatter()` function
3. correlograms (acf plots) using the `acf()` function
4. line plots (y versus x or the values of y versus their row index) using the `plot()` function
5. kernel density estimates using the `kden()` function

Section B.16 provides extensive documentation on the plotting functions. The remainder of this section gives a general overview of how plots are handled in BayES and presents some simple examples.

When a plotting function is called in BayES in its simple form, a new figure window is created and the corresponding plot is drawn within this window. Figure windows are named consecutively as `"Figure 1"`, `"Figure 2"`, etc., and these names can be used to interact with them, for example to close them programmatically or save their plots in any of the supported graphics formats. For example, the statement:

```
close("Figure 2");
```

will close the figure window with `"Figure 2"` appearing on its title bar and release the memory occupied by this figure. To prevent extreme use of memory resources for presenting plots, BayES restricts the maximum number of figure windows that are open simultaneously to 20. This number can be changed using the **maxfigures()** function.

Once a figure window is closed, its name will be reused. For example, if there are currently two figure windows open with titles `"Figure 1"` and `"Figure 3"` (the user closed the figure window with title `"Figure 2"`), the next time a plotting function is used, the title of the new figure window will be `"Figure 2"`. The statement:

```
close(all);
```

closes all currently open figure windows and releases resources.

The titles of figure windows can also be used to export the associated plots to the following graphics formats:

1. encapsulated postscript (`.eps`)
2. portable network format (`.png`)
3. joint photographic experts group (`.jpeg`)

This is achieved using the **export()** function, by passing the name of the figure window that contains the plot to be exported as the first argument of the **export()** function. Section B.4 provides extensive documentation on the **export()** function.

The five basic plotting functions mentioned above differ in the number of numerical arguments they take, but all of them have the following optional arguments:

- `"title"`
- `"xlabel"`
- `"ylabel"`
- `"grid"`
- `"colors"`

These arguments, if provided, must be given after the numerical arguments of the respective plotting function, separated by commas.[8] The values of the first four options should be strings and of the `"colors"` option a matrix. The three first options, as presented above, specify the title of the plot and the labels on the 'x' and 'y' axes, respectively. The values of the `"grid"`

another item with id value `x`, these two items are distinct: altering the value of `x` defined inside the function will not affect the value of `x` in the calling script.

[8] Optional arguments passed to the plotting functions can be provided in any order, but always come in pairs (eg. `"title"="my title"`).

option must equal to either "on" or "off", with the first value requesting that a grid is plotted on the graph. The last option specifies the colors to be used in the graph and its value must a matrix with three columns and, possibly, multiple rows. The values in each row represent a color in RGB (red-green-blue) format and should be between zero and one. The first row specifies the background color of the graph[9] and the second the color of the axes and text labels. The remaining rows specify the colors to be used when plotting the data.

BayES provides support for figure windows which can contain multiple plots. A call to the multiplot() function will initialize a figure window which can store multiple plots. Subsequent calls to the five basic plotting functions, accompanied by calls to the subplot() function, can be used to populate the spaces of this window with actual plots. See section B.16 for more details.

Example 2.11 demonstrates how to plot a histogram of a set of values contained in a vector, while example 2.12 shows how to overlay kernel density estimates of the values contained in two vectors. Finally, 2.13 demonstrates how to plot a set of functions. Sample "4-Plotting data.bsf", located at "$BayESHOME/Samples/1-GettingStarted" contains some more complete examples of using the plotting functions.

■ Example 2.11

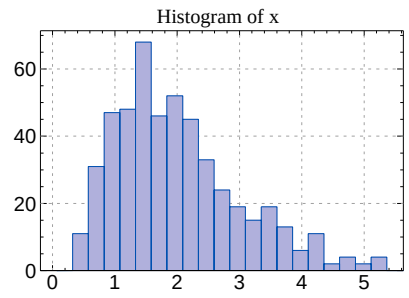
▼ Input

```
// Set the seed for the random-number
// generator and draw 500 numbers from
// a Gamma(4,2) distribution
seed(42);
x = gamrnd(4,2,500,1);

// Plot a histogram of the values in x
hist(x, 20,
     "title"="Histogram of x",
     "grid"="on" );

// Export the graph as eps
export( "Figure 1", "./Hist.eps",
     "width"=420, "height"=280 );
```

▼ Output



■ Example 2.12

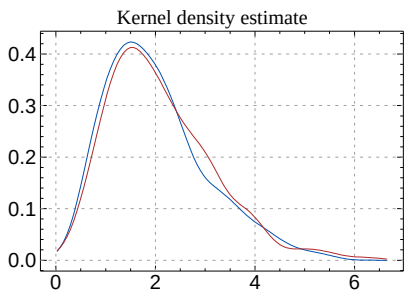
▼ Input

```
// Draw two sets of 500 numbers from
// a Gamma(4,2) distribution
seed(42);
x = gamrnd(4,2,500,1);
y = gamrnd(4,2,500,1);

// Overlay the kernel density estima-
// tes for the values in x and y
kden( [ x, y ],
     "title"="Kernel density estimate",
     "grid"="on" );

// Export the graph as eps
export( "Figure 1", "./Kden.eps",
     "width"=420, "height"=280 );
```

▼ Output



[9]Subplots within graphs must have the same background color. The overall background color in graphs that contain multiple subplots is the background color specified for the subplot at the upper left corner of the graph.

■ **Example 2.13**

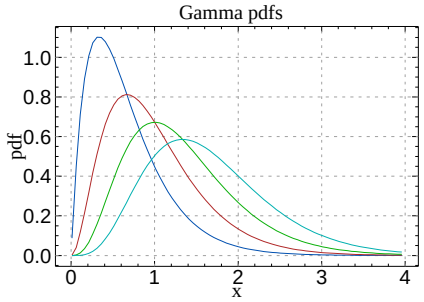
▼ Input

```
// Define x-axis values
x = range(0.01, 4, 0.05);

// Plot the Gamma pdf with varying
// shape parameter
y1 = gampdf(x, 2, 3);
y2 = gampdf(x, 3, 3);
y3 = gampdf(x, 4, 3);
y4 = gampdf(x, 5, 3);
myPlot = plot( [ y1, y2, y3, y4 ], x,
    "title" = "Gamma pdfs",
    "xlabel" = "x", "ylabel" = "pdf",
    "grid"="on" );

// Export the graph as eps
export( myPlot, "./Gampdf.eps",
    "width"=420, "height"=280 );
```

▼ Output



Chapter 3

Interfaces to external programs

3.1 Overview

BayES is designed to perform some very specific tasks and it recognizes that there could be other software packages available that are more appropriate for other types of tasks. Instead of trying to do everything by itself, BayES provides interfaces to these packages to facilitate communication. This is achieved by means of software-specific interface functions, which allow the user to pass data and native code to, execute scripts in and retrieve results from each package, without ever leaving the BayES environment.

When an interface function is called in BayES, control passes to the respective software package, which starts executing the code provided by the user in the package's native language. Once execution completes, control passes back to BayES. Depending on the particular software package, any output produced by it will be printed on the BayES console, either in real time or after execution of the external program completes. External programs to which BayES provides interfaces must be separately installed on the machine on which BayES is currently used. Furthermore, the locations of executable/binary files of these external software packages must be known to BayES.[1]

BayES interface functions pass data to and retrieve results from external processes by writing temporary files in the current working directory. Upon successful completion of the external program, these files are deleted. Therefore, the user should have write access to the current working directory for the interface functions to work.

3.2 Interface to JAGS

Just Another Gibbs Sampler (JAGS) is an open-source program which takes as inputs data and a model specification file (written in JAGS' own language) and draws samples from the posterior distribution of the model's parameters or latent variables, picking the most appropriate sampling method automatically.

BayES' `jags()` function provides a convenient interface to JAGS, which allows the user to:

- pass BayES matrices as input data to JAGS
- request JAGS to draw samples from the posterior distribution of parameters specified by the user, given a model specification file
- retrieve the draws from JAGS, summarize them and print summary statistics on the BayES console
- store the draws from JAGS in a BayES model item, making them available for post estimation analysis

The general syntax of the `jags()` function is the following:[2]

```
[<model name> = ] jags( <model specification file>
        [, "data"=<list of matrices to pass to JAGS> ]
        [, "monitor"=<list of parameters to monitor> ]
        [, "inits"=<structure of initial values> ]
        [, "chains"=<positive integer> ]
        [, "burnin"=<positive integer> ]
        [, "draws"=<positive integer> ]
        [, "thin"=<positive integer> ]
        [, "seed"=<positive integer> ]
        );
```

[1] BayES can be made aware of the location of a package's binary/executable either using the **setbinary()** function (see section B.17 for more information) or using the main menu, via `Help→External Binaries`.

[2] Arguments inside square brackets are optional. Optional arguments passed to the `jags()` function can be provided in any order, but always after the mandatory argument (model specification file). Optional arguments always come in pairs (eg. `"chains"=1`).

where:

- *<model name>* is a BayES id value which will be associated with the model resulting from executing the `jags()` function. If no model name is provided the results from JAGS will still be returned to BayES and summarized, but they will not be stored for further analysis. `jags()`, `openbugs()` and `stan()` are the three interface functions that provide the highest level of integration with BayES: the results from these functions are stored in BayES model items, on which all BayES functions which operate on models can be used.

- *<model specification file>* is a string pointing to the file which contains the specification of the JAGS model. If the specification file is not in the current directory then the file name must be prepended by the path to the file, either in absolute terms (eg. `"C:/MyFiles/myModel.txt"`) or relative to the current working directory (eg. `"../myModel.txt"`). This is the only mandatory argument of the `jags()` function.

- `"data"` specifies the data matrices that will be passed as input to JAGS. *<list of matrices>* is a list of the id values of matrices (comma-separated names inside curly brackets), as they appear in the JAGS model specification file. These matrices must be defined in the current workspace.

- `"monitor"` specifies the parameters or latent variables for which JAGS will store the draws obtained from their posterior distributions. *<list of parameters to monitor>* is a list of strings (comma-separated strings inside curly brackets) that specify the names of the parameters/latent data that should be monitored, as they appear in the JAGS model specification file. If no monitors are set JAGS will still draw samples from the posterior but these will not be saved and, subsequently, no BayES model item will be defined upon return of the `jags()` function.

- `"inits"` specifies the initial values per chain used by JAGS. *<structure of initial values>* is a BayES structure, the elements of which could be structures themselves. Each element of the chain-specific structure corresponds to a parameter or latent variable, using the same id values as the ones used in the JAGS model specification file. It is possible to provide initial values for all parameters/latent variables or only a subset of them. It is also possible to leave entire chains uninitialized. In such cases JAGS will generate initial values for the chains/parameters/latent variables which are not initialized by the user.

- `"chains"` specifies the number of chains that JAGS will run in parallel. The right-hand side must be a positive integer and the default value is 1.

- `"burnin"` specifies the number of draws from the posterior that will be discarded (per chain) to avoid dependence of the results on initial values. The right-hand side must be a positive integer and the default value is 10,000.

- `"draws"` specifies the number of draws from the posterior that will be retained, per chain. The right-hand side must be a positive integer and the default value is 20,000.

- `"thin"` specifies the number of draws from the posterior that will be skipped (after the burn-in phase) per retained draw, to avoid high autocorrelation of the retained draws. For example, if the thinning parameter is set to 3, then only one in three consecutive draws will be retained and become available for inference and post-estimation analysis. The right-hand side must be a positive integer and the default value is 1.

- `"seed"` specifies the seed for the random-number generator used by JAGS. The right-hand side must be a positive integer and the default value is 42.

Under Linux and macOS systems, the path to the JAGS model specification file must not contain any spaces.

As the jags() function executes, JAGS attempts to print output on the system's command console. BayES grabs this output and redirects it to the BayES main console in real time. This output is entirely determined by JAGS and it includes information on the model specification file used in the current run, any errors or warnings and, most importantly, information on the progress of the sampler relative to the total number of requested draws from the posterior.

Many of the sample script files in "$BayESHOME/Samples/3-JAGS-OpenBUGS-Stan" contain examples of using the jags() function, along with JAGS model specification files for simple models. The JAGS interface is also accessible from the BayES main menu via Interfaces→JAGS.

3.3 Interface to OpenBUGS

OpenBUGS is another open-source program that is very similar to JAGS. It too takes as inputs data and a model specification file (written in OpenBUGS' own language) and draws samples from the posterior distribution of the model's parameters or latent variables, using an expert system to pick the most appropriate method for sampling from the posterior.

BayES' openbugs() function provides a convenient interface to OpenBUGS, which allows the user to:

- pass BayES matrices as input data to OpenBUGS
- request OpenBUGS to draw samples from the posterior distribution of parameters specified by the user, given a model specification file
- retrieve the draws from OpenBUGS, summarize them and print summary statistics on the BayES console
- store the draws from OpenBUGS in a BayES model item, making them available for post estimation analysis

The general syntax of the openbugs() function is the following:[3]

```
[<model name> = ] openbugs( <model specification file>
        [, "data"=<list of matrices to pass to OpenBUGS> ]
        [, "monitor"=<list of parameters to monitor> ]
        [, "summarize"=<list of parameters to summarize> ]
        [, "inits"=<structure of initial values> ]
        [, "chains"=<positive integer> ]
        [, "burnin"=<positive integer> ]
        [, "draws"=<positive integer> ]
        [, "thin"=<positive integer> ]
        [, "seed"=<positive integer> ]
        [, "window"=true|false ]
        [, "debug"=true|false ]
        );
```

where:

- <model name> is a BayES id value which will be associated with the model resulting from executing the openbugs() function. If no model name is provided the results from OpenBUGS will still be returned to BayES and summarized, but they will not be stored for further analysis. jags(), openbugs() and stan() are the three interface functions that provide the highest level of integration with BayES: the results from these functions are stored in BayES model items, on which all BayES functions which operate on models can be used.

- <model specification file> is a string pointing to the file which contains the specification of the OpenBUGS model. If the specification file is not in the current working directory then the file name must be prepended by the path to the file, either in absolute terms (eg. "C:/MyFiles/myModel.txt") or relative to the current working directory (eg. "../myModel.txt"). This is the only mandatory argument of the openbugs() function.

[3] Arguments inside square brackets are optional. Optional arguments passed to the openbugs() function can be provided in any order, but always after the mandatory argument (model specification file). Optional arguments always come in pairs (eg. "chains"=1).

- "data" specifies the data matrices that will be passed as input to OpenBUGS. *<list of matrices>* is a list of the id values of matrices (comma-separated names inside curly brackets), as they appear in the OpenBUGS model specification file. These matrices must be defined in the current workspace.

- "monitor" specifies the parameters or latent variables for which OpenBUGS will store the draws obtained from their posterior distributions. *<list of parameters to monitor>* is a list of strings (comma-separated strings inside curly brackets) that specify the names of the parameters/latent data that should be monitored, as they appear in the OpenBUGS model specification file. If no monitors are set OpenBUGS will still draw samples from the posterior but these will not be saved and, subsequently, no BayES model item will be defined upon return of the openbugs() function.

- "summarize" specifies the parameters or latent variables for which OpenBUGS will set "summary monitors". OpenBUGS will sample from the posterior of these parameters/latent variables but, instead of storing these draws, it will only provide summary statistics for them (calculated inline). This option can lead to a large reduction in memory usage compared to the case where a large number of parameters/latent variables are monitored using the "monitor" option. *The "summarize" option is currently not implemented and will be ignored.*

- "inits" specifies the initial values per chain used by OpenBUGS. *<structure of initial values>* is a BayES structure, the elements of which could be structures themselves. Each element of the chain-specific structure corresponds to a parameter or latent variable, using the same id values as the ones used in the OpenBUGS model specification file. It is possible to provide initial values for all parameters/latent variables or only a subset of them. It is also possible to leave entire chains uninitialized. In such cases OpenBUGS will generate initial values for the chains/parameters/latent variables which are not initialized by the user.

- "chains" specifies the number of chains that OpenBUGS will run in parallel. The right-hand side must be a positive integer and the default value is 1. Note that OpenBUGS can use up to 14 different values for the seed number of its random-number generators. If more than 14 chains are requested by the user these seed values will be recycled, effectively leading to chains with exactly the same draws from the posterior.

- "burnin" specifies the number of draws from the posterior that will be discarded (per chain) to avoid dependence of the results on initial values. The right-hand side must be a positive integer and the default value is 10,000.

- "draws" specifies the number of draws from the posterior that will be retained, per chain. The right-hand side must be a positive integer and the default value is 20,000.

- "thin" specifies the number of draws from the posterior that will be skipped (after the burn-in phase) per retained draw, to avoid high autocorrelation of the retained draws. For example, if the thinning parameter is set to 3 then only one in three consecutive draws will be retained and become available for inference and post-estimation analysis. The right-hand side must be a positive integer and the default value is 1.

- "seed" specifies the seed for the random-number generator used by OpenBUGS. The right-hand side must be a positive integer and the default value is 42.

- "window" specifies whether OpenBUGS should run in its own graphical environment (under Microsoft® Windows® only) or print any output on the BayES console. The default value value is **true**, which requests OpenBUGS to run using its own GUI. Under Linux systems OpenBUGS does not provide a GUI and, therefore, this option is ignored.

- `"debug"` specifies whether the OpenBUGS window will remain open after completion of the Gibbs sampler (under Microsoft® Windows® only). The default value is `"false"`, in which case the OpenBUGS window closes after completion of the Gibbs sampler and control is returned to BayES automatically. Because the `"debug"` option requires an OpenBUGS window, the value of the `"window"` option is overwritten to **`true`** whenever the user sets `"debug"` to **`true`**. This option is ignored under Linux systems.

When OpenBUGS runs using its GUI (under Microsoft® Windows® only) it prints any output on its own output window. When OpenBUGS runs without its GUI then BayES grabs any OpenBUGS output intended for the system's command console and redirects it to the BayES console in real time. This output is entirely determined by OpenBUGS and it includes information on the stage of the estimation process (model check, loading data, etc.) and very limited information on the progress of the sampler (only whether the burn-in phase is complete).

Many of the sample script files in `"$BayESHOME/Samples/3-JAGS-OpenBUGS-Stan"` contain examples of using the `openbugs()` function, along with OpenBUGS model specification files for simple models. The OpenBUGS interface is also accessible from the BayES main menu via `Interfaces` →`OpenBUGS`.

3.4 Interface to Stan

Stan is another open-source program similar to JAGS and OpenBUGS, but with two important differences (i) the language it uses is slightly more complex, but also more flexible, and (ii) it uses specialized sampling algorithms designed to work efficiently on hierarchical models. No matter these differences, Stan also takes as inputs data and a model specification file (written in Stan's own language) and draws samples from the posterior distribution of the model's parameters or latent variables, or maximizes the respective likelihood function.

BayES' `stan()` function provides a convenient interface to Stan, which allows the user to:

- pass BayES matrices as input data to Stan
- request Stan to draw samples from the posterior distribution of the model's parameters and latent data or maximize the likelihood function, given a model specification file
- retrieve the draws or other results from Stan, summarize them and print summary statistics on the BayES console
- store the draws or maximum-likelihood estimates from Stan in a BayES model item, making them available for post estimation analysis

The general syntax of the `stan()` function is the following:[4]

```
[<model name> = ] stan( <model specification file>
        [, "method"="sample"|"variational"|"optimize" ]
        [, "data"=<list of matrices to pass to Stan> ]
        [, "inits"=<structure of initial values> ]
        [, "output"=<file where Stan should store its results> ]
        [, "diagnostic"=<file where Stan should store results for diagnostics> ]
        [, "options"=<additional command-line options> ]
        [, "summarize"=true|false ]
        [, "diagnose"=true|false ]
        [, "chains"=<positive integer> ]
        [, "burnin"=<positive integer> ]
        [, "draws"=<positive integer> ]
        [, "thin"=<positive integer> ]
        [, "seed"=<positive integer> ]
        [, "refresh"=<positive integer> ]
        );
```

[4] Arguments inside square brackets are optional. Optional arguments passed to the `stan()` function can be provided in any order, but always after the mandatory argument (model specification file). Optional arguments always come in pairs (eg. `"chains"=1`).

where:

- *<model name>* is a BayES id value which will be associated with the model resulting from executing the `stan()` function. If no model name is provided the results from Stan will still be returned to BayES and summarized, but they will not be stored for further analysis. `jags()`, `openbugs()` and `stan()` are the three interface functions that provide the highest level of integration with BayES: the results from these functions are stored in BayES model items, on which all BayES functions which operate on models can be used.

- *<model specification file>* is a string pointing to the file which contains the specification of the Stan model. If the specification file is not in the current working directory then the file name must be prepended by the path to the file, either in absolute terms (eg. `"C:/MyFiles/myModel.stan"`) or relative to the current directory (eg. `"./myModel.stan"`). This is the only mandatory argument of the `stan()` function.

- `"method"` specifies the Stan method to be used. This can be one of the strings `"sample"`, `"variational"` or `"optimize"`, each one of them invoking the respective Stan method. Note that Stan's "diagnose" method cannot be accessed in BayES directly, but diagnostic tests can be performed within Stan by setting the `"diagnose"` option to **`true`** in the `stan()` function. The default value of the `"method"` argument is `"sample"`, in which case Stan samples from the posterior distribution of the model's parameters using a Hamiltonian Monte Carlo (HMC) algorithm of fixed-parameter sampling (depending on other options).

- `"data"` specifies the data matrices that will be passed as input to Stan. *<list of matrices>* is a list of the id values of matrices (comma-separated names inside curly brackets), as they appear in the Stan model specification file. These matrices must be defined in the current workspace.

- `"inits"` specifies the initial values per chain used by Stan. *<structure of initial values>* is a BayES structure, the elements of which could be structures themselves. Each element of the chain-specific structure corresponds to a parameter or latent variable, using the same id values as the ones used in the Stan model specification file. It is possible to provide initial values for all parameters/latent variables or only a subset of them. It is also possible to leave entire chains uninitialized. In such cases Stan will generate initial values for the chains/parameters/latent variables which are not initialized by the user, using its default options.

- `"output"` specifies the file to which Stan should store its results. If the output file is not in the current directory then the file name must be prepended by the path to the file, either in absolute terms (eg. `"C:/MyFiles/myResults.csv"`) or relative to the current working directory (eg. `"./myResults.csv"`). If the output file is not specified then BayES will create temporary files in the current working directory. If, however, the user provides a name for the output file(s), these will persist even after exiting BayES.

- `"diagnostic"` specifies the file to which Stan should store results that can be used for post-estimation diagnostics. If the diagnostic file is not in the current directory then the file name must be prepended by the path to the file, either in absolute terms (eg. `"C:/MyFiles/myDgnstcs.csv"`) or relative to the current directory (eg. `"./myDgnstcs.csv"`). If the user provides a name for the diagnostic file(s), these will persist even after exiting BayES.

- `"summarize"` indicates whether the results produced by Stan (draws or values at which the likelihood function is maximized) should be summarized withing Stan, before returning control to BayES. The default value of `"summarize"` is **`true`**.

- `"diagnose"` indicates whether Stan should run diagnostic tests on the results it produced (draws or values at which the likelihood function is maximized), before returning control to BayES. The default value of `"summarize"` is **`true`**. Note that this optional argument effectively replaces Stan's "diagnose" method.

- `"options"` can be used to pass additional options to Stan, using its extensive argument tree. These options should be provided to BayES' `stan()` function as a string, which is then passed verbatim to Stan. For example, setting the right-hand side of the `"options"` argument to `"algorihtm=fixed_param"` requests Stan to use the fixed-parameter sampler under its "sample" method.

- `"chains"` specifies the number of chains that Stan will run in parallel. If the `"method"` argument of the `stan()` function is set to `"sample"` (default), BayES spawns as many Stan processes as the number of chains, which run in parallel and also mutes Stan's output on the console. If, however, the `"method"` argument is set to either `"variational"` or `"optimize"`, the `"chains"` argument is ignored.

- `"burnin"` performs different functions under different Stan methods. If the `"method"` argument of the `stan()` function is set to `"sample"` (default), `"burnin"` specifies the number of draws from the posterior that will be discarded (per chain) to avoid dependence of the results on initial values. If the `"method"` argument is set to `"variational"`, `"burnin"` specifies the maximum number of ADVI iterations. This argument is ignored when the `"method"` argument is set to `"optimize"`. The right-hand side must be a positive integer and the default value is 10,000.

- `"draws"` performs different functions under different Stan methods. If the `"method"` argument of the `stan()` function is set to `"sample"` (default), or `"variational"`, `"draws"` specifies the number of draws from the posterior that will be retained, per chain. If the `"method"` argument is set to `"optimize"`, `"draws"` specifies the maximum number of iterations of the algorithm that is used to maximize the likelihood. The right-hand side must be a positive integer and the default value is 20,000.

- `"thin"` specifies, when the `"method"` argument of the `stan()` function is set to `"sample"` (default), the number of draws from the posterior that will be skipped (after the burn-in phase) per retained draw, to avoid high autocorrelation of the retained draws. For example, if the thinning parameter is set to 3, then only one in three consecutive draws will be retained and become available for inference and post-estimation analysis. The `"thin"` argument is ignored when the `"method"` argument is set to `"variational"` or `"optimize"`. The right-hand side must be a positive integer and the default value is 1.

- `"seed"` specifies the seed for the random-number generator used by Stan. The right-hand side must be a positive integer and the default value is 42.

- `"refresh"` specifies the rate at which Stan prints information about its progress on the console. For example, if `"refresh"` is set equal to 100, Stan will print information every 100 iterations of the respective algorithm. The right-hand side must be a positive integer and the default value is 1000.

The path to the Stan model specification file must not contain any spaces.

As the `stan()` function executes, Stan attempts to print output on the system's command console. BayES grabs this output and redirects it to the BayES main console in real time. This output is entirely determined by Stan and it includes information on the model specification file used in the current run, all the options used, any errors or warnings and, most importantly, information on the progress of the algorithm being used. Note that when multiple chains are run in parallel (under Stan's "sample" method), BayES mutes Stan's output on the console.

Many of the sample script files in `"$BayESHOME/Samples/3-JAGS-OpenBUGS-Stan"` contain examples of using the `stan()` function, along with Stan model specification files for simple models. The Stan interface is also accessible from the BayES main menu via `Interfaces→Stan`.

3.5 Interface to R

R is a free software environment designed for statistical computing and graphics. Compared to JAGS and OpenBUGS, R has a much more complete programming language and encompasses a vast array of computational and statistical techniques. Nevertheless, it too operates using script files written in its native language, which makes running it in batch mode feasible.

BayES' `rproject()` function provides a convenient interface to R, which allows the user to:

- pass BayES matrices and datasets as input to R
- request R to execute code written in its native language
- retrieve output from R and store it in BayES dataset and matrix items; all data to be returned from R are stored inside a BayES structure item

The general syntax of the `rproject()` function is the following:[5]

```
[<structure name> = ] rproject ( <R script file>
        [, "data"=<list of matrices/datasets to pass to R> ]
        [, "return"=<list of matrices/dataframes to retrieve from R> ]
        );
```

where:

- *<structure name>* is a BayES id value which will be associated with the BayES structure that the `rproject()` function returns. This structure will contain any R matrices or data frames that the user requests to be returned to BayES (using the `"return"` option) after execution of the R script completes.
- *<R script file>* is a string pointing to the file which contains the code (written in R's language) that R will be requested to execute. If this file is not in the current directory then the file name must be prepended by the path to the file, either in absolute terms (eg. `"C:/MyFiles/myScript.R"`) or relative to the current working directory (eg. `"../myScript.R"`). This is the only mandatory argument of the `rproject()` function.
- `"data"` specifies the data that will be passed as input to R. These can be either BayES datasets or matrices. *<list of matrices/datasets to pass to R>* is a list of the id values of matrices or datasets (comma-separated names inside curly brackets), as they appear in the R script file. These matrices or datasets must be defined in the current workspace. When a BayES matrix is passed as input to R then this becomes available as an R matrix in R. When a BayES dataset is passed as input to R then this becomes available as an R data frame in R.
- `"return"` specifies the R objects that will be returned to BayES when execution of the R script completes. *<list of matrices/dataframes to retrieve from R>* is a list of id values (comma-separated id values inside curly brackets) that specify the names of the matrices/datasets that should be returned from R, as they appear in the R script file. Any R matrix that is returned will be stored in BayES as a matrix, while any R data frame will be stored as a BayES dataset. These matrices/datasets are grouped together into a BayES structure. Passing other data types (structures, lists, strings, etc.) between BayES and R is not supported.

As the `rproject()` function executes, R attempts to print output on the system's command console. BayES grabs this output and redirects it to the BayES main console in real time. This output is entirely determined by R and the commands contained in the R script file provided to `rproject()`.

The sample script file in `"$BayESHOME/Samples/5-Interfaces/rproject"` contains an example of using the `rproject()` function, along with a simple R script file. The R interface is also accessible from the BayES main menu via `Interfaces→R project`.

[5] Arguments inside square brackets are optional. Optional arguments passed to the `rproject()` function can be provided in any order, but always after the mandatory argument (R script file). Optional arguments always come in pairs (eg. `"data"={myDataset,myMatrix}`).

3.6 Interface to Stata®

Stata is a proprietary software package designed for data analysis, data management, and graphics. Stata provides canned procedures for estimating many popular econometric models, most of them being frequentist. However, recent versions of Stata also include Bayesian procedures. Stata has a complete GUI but it also features a simple scripting language, which makes running it in batch mode feasible.

BayES' `stata()` function provides a convenient interface to Stata, which allows the user to:

- pass BayES matrices and a single dataset as input to Stata
- request Stata to execute code written in its native language
- retrieve output from Stata and store it in BayES dataset and matrix items; all data to be returned from Stata are stored inside a BayES structure item

The general syntax of the `stata()` function is the following:[6]

```
[<structure name> = ] stata( <Stata .do file>
        [, "data"=<list of matrices/dataset to pass to Stata> ]
        [, "return"=<list of matrices to retrieve from Stata> ]
        );
```

where:

- *<structure name>* is a BayES id value which will be associated with the BayES structure that the `stata()` function returns. This structure will contain any Stata matrices or dataset that the user requests to be returned to BayES (using the `"return"` option) after execution of the Stata `.do` file completes.

- *<Stata .do file>* is a string pointing to the file which contains the code (written in Stata's language) that Stata will be requested to execute. If this file is not in the current directory then the file name must be prepended by the path to the file, either in absolute terms (eg. `"C:/MyFiles/myFile.do"`) or relative to the current working directory (eg. `"../myFile.do"`). This is the only mandatory argument of the `stata()` function.

- `"data"` specifies the data that will be passed as input to Stata. *<list of matrices/dataset to pass to Stata>* is a list of the id values of matrices or a dataset (comma-separated names inside curly brackets), as they appear in the Stata `.do` file. The list can contain any number of BayES matrices and at most one BayES dataset. When a BayES matrix is passed as input to Stata then this becomes available as a Stata matrix in Stata. When a BayES dataset is passed as input to Stata then this becomes available in Stata as its dataset.[7] These matrices/dataset must be defined in the current workspace.

- `"return"` specifies the Stata objects that will be returned to BayES when execution of the `.do` file completes. *<list of matrices to retrieve from Stata>* is a list of id values (comma-separated id values inside curly brackets) that specify the names of the matrices that should be returned from Stata, as they appear in the Stata `.do` file. Any Stata matrix that is returned will be stored in BayES as a matrix. If a dataset is available in Stata when execution of the code contained in the provided `.do` file completes, then `stata()` retrieves this Stata dataset and stores it as a BayES dataset, even is not requested by the user.[8] These retrieved matrices/dataset are grouped together into a BayES structure. Passing other data types (structures, models, strings, etc.) between BayES and Stata is not supported.

[6] Arguments inside square brackets are optional. Optional arguments passed to the `stata()` function can be provided in any order, but always after the mandatory argument (Stata `.do` file). Optional arguments always come in pairs (eg. `"data"={myDataset,myMatrix}`).

[7] Stata can handle only a single dataset at a time. If the user attempts to pass more than one datasets as data to Stata, `stata()` will issue an error.

[8] If you do not want the Stata dataset to be returned to BayES, simply include `clear` as the last line of code in the provided Stata `.do` file.

When the stata() function calls Stata, Stata starts only part of its GUI and executes the code contained in the provided Stata .do file. BayES automatically directs any Stata output to a log file. When execution of the code contained in the .do file completes BayES reads the Stata output and prints it on the BayES console. Therefore, BayES may appear frozen during the time Stata executes the commands in the .do file. The output printed on the BayES console is entirely determined by Stata and the commands contained in the Stata .do file provided to stata().

The sample script file in "$BayESHOME/Samples/5-Interfaces/stata" contains an example of using the stata() function, along with a simple Stata .do file. The Stata interface is also accessible from the BayES main menu via Interfaces→Stata®.

3.7 Interface to MATLAB®

MATLAB is a proprietary software platform designed for solving engineering and scientific problems. Although it provides some toolboxes that facilitate statistical analysis, in general, and econometric modeling, in particular (mostly time-series models), MATLAB is primarily known for its extensive and easy to use language. This is largely a matrix-based language, specifically designed for solving computational problems.

BayES' matlab() function provides a convenient interface to MATLAB, which allows the user to:

- pass BayES matrices and datasets as input to MATLAB
- request MATLAB to execute code written in its native language
- retrieve output from MATLAB and store it in BayES matrix items; all data to be returned from MATLAB are stored inside a BayES structure item

The general syntax of the matlab() function is the following:[9]

```
[<structure name> = ] matlab( <MATLAB .m file>
        [, "data"=<list of matrices/datasets to pass to MATLAB> ]
        [, "return"=<list of matrices to retrieve from MATLAB> ]
        [, "options"=<string of options when starting MATLAB> ]
        );
```

where:

- <*structure name*> is a BayES id value which will be associated with the BayES structure that the matlab() function returns. This structure will contain any MATLAB matrices that the user requests to be returned to BayES (using the "return" option) after execution of the MATLAB .m file completes.
- <*MATLAB .m file*> is a string pointing to the file which contains the code (written in MATLAB's language) that MATLAB will be requested to execute. If this file is not in the current directory then the file name must be prepended by the path to the file, either in absolute terms (eg. "C:/MyFiles/myFile.m") or relative to the current working directory (eg. "../myFile.m"). This is the only mandatory argument of the matlab() function.
- "data" specifies the data that will be passed as input to MATLAB. These can be either BayES datasets or matrices. <*list of matrices/datasets to pass to MATLAB*> is a list of the id values of matrices or datasets (comma-separated names inside curly brackets), as they appear in the MATLAB .m file. These matrices or datasets must be defined in the current workspace. When either a BayES matrix or dataset is passed as input to MATLAB then it becomes available in MATLAB as a MATLAB matrix. That is, BayES datasets are stripped of their additional information (most importantly, variable names and dataset structure) and only the raw data are passed to MATLAB.

[9] Arguments inside square brackets are optional. Optional arguments passed to the matlab() function can be provided in any order, but always after the mandatory argument (MATLAB .m file). Optional arguments always come in pairs (eg. "data"={myDataset,myMatrix}).

- `"return"` specifies the MATLAB matrices that will be returned to BayES when execution of the MATLAB `.m` file completes. *<list of matrices to retrieve from MATLAB>* is a list of id values (comma-separated id values inside curly brackets) that specify the names of the matrices that should be returned from MATLAB, as they appear in the MATLAB `.m` file. Any MATLAB matrix that is returned will be stored in BayES as a matrix and all returned matrices are grouped together into a BayES structure. Passing other data types (structures, cells, strings, etc.) between BayES and MATLAB is not supported.

- `"options"` specifies the MATLAB options to be passed to the command-line argument when calling MATLAB. All these options should be provided as a single string. The default value is:
  `"-nojvm -nodesktop -nodisplay -nosplash -minimize -wait -r"`
  and has the following effects:

  - `nojvm`: starts MATLAB without the JVM software. Features that require Java software (such as the desktop tools and graphics) are not supported.
  - `nodesktop`: runs the JVM software without opening the MATLAB desktop
  - `nodisplay`: starts the JVM software without starting the MATLAB desktop
  - `nosplash`: does not display the splash screen during startup
  - `minimize`: minimizes the MATLAB window
  - `wait`: blocks the script from continuing until the results from MATLAB are generated.
  - `r`: executes the MATLAB statement, specified as a string or as the name of a MATLAB script or function. This should always be used as the last option because BayES appends to this option the location of the provided MATLAB `.m` file.

As the `matlab()` function executes with the above options, MATLAB attempts to print output on the system's command console. BayES grabs this output and redirects it to the BayES main console in real time. This output is entirely determined by MATLAB and the commands contained in the MATLAB `.m` file provided to `matlab()`.

The sample script file in `"$BayESHOME/Samples/5-Interfaces/matlab"` contains an example of using the `matlab()` function, along with a simple MATLAB `.m` file. The MATLAB interface is also accessible from the BayES main menu via `Interfaces`→`MATLAB`®.

3.8 Interface to GNU Octave

GNU Octave is free software designed as a matrix programming language, which is largely compatible with MATLAB. Later versions of GNU Octave feature a simple GUI, but the vast majority of functions can still be accessed only using code. GNU Octave is intended for numerical computations, in general, and graphics, but it can be also used for performing Bayesian inference, especially when using some of its statistics-oriented loadable modules.

BayES' `octave()` function provides a convenient interface to GNU Octave, which allows the user to:

- pass BayES matrices and datasets as input to GNU Octave
- request GNU Octave to execute code written in its native language
- retrieve output from GNU Octave and store it in BayES matrix items; all data to be returned from GNU Octave are stored inside a BayES structure item

The general syntax of the `octave()` function is the following:[10]

[10] Arguments inside square brackets are optional. Optional arguments passed to the `octave()` function can be provided in any order, but always after the mandatory argument (GNU Octave `.m` file). Optional arguments always come in pairs (eg. `"data"={myDataset,myMatrix}`).

```
[<structure name> = ] octave( <GNU Octave .m file>
        [, "data"=<list of matrices/datasets to pass to GNU Octave> ]
        [, "return"=<list of matrices to retrieve from GNU Octave> ]
        );
```

where:

- *<structure name>* is a BayES id value which will be associated with the BayES structure that the `octave()` function returns. This structure will contain any GNU Octave matrices that the user requests to be returned to BayES (using the `"return"` option) after execution of the GNU Octave `.m` file completes.
- *<GNU Octave .m file>* is a string pointing to the file which contains the code (written in GNU Octave's language) that GNU Octave will be requested to execute. If this file is not in the current directory then the file name must be prepended by the path to the file, either in absolute terms (eg. `"C:/MyFiles/myFile.m"`) or relative to the current working directory (eg. `"../myFile.m"`). This is the only mandatory argument of the `octave()` function.
- `"data"` specifies the data that will be passed as input to GNU Octave. These can be either BayES datasets or matrices. *<list of matrices/datasets to pass to GNU Octave>* is a list of the id values of matrices or datasets (comma-separated names inside curly brackets), as they appear in the GNU Octave `.m` file. These matrices or datasets must be defined in the current workspace. When either a BayES matrix or dataset is passed as input to GNU Octave then it becomes available as a GNU Octave matrix. That is, BayES datasets are stripped of their additional information (most importantly, variable names and dataset structure) and only the raw data are passed to GNU Octave.
- `"return"` specifies the GNU Octave matrices that will be returned to BayES when execution of the GNU Octave `.m` file completes. *<list of matrices to retrieve from GNU Octave>* is a list of id values (comma-separated id values inside curly brackets) that specify the names of the matrices that should be returned from GNU Octave, as they appear in the GNU Octave `.m` file. Any GNU Octave matrix that is returned will be stored in BayES as a matrix and all returned matrices are grouped together into a BayES structure. Passing other data types (structures, cells, strings, etc.) between BayES and GNU Octave is not supported.

As the `octave()` function executes, GNU Octave attempts to print output on the system's command console. BayES grabs this output and redirects it to the BayES main console in real time. This output is entirely determined by GNU Octave and the commands contained in the GNU Octave `.m` file provided to `octave()`.

The sample script file in `"$BayESHOME/Samples/5-Interfaces/octave"` contains an example of using the `octave()` function, along with a simple GNU Octave `.m` file. The GNU Octave interface is also accessible from the BayES main menu via `Interfaces→GNU Octave`.

3.9 The system function

Apart from the functions that BayES provides to facilitate communication with other scientific software, the **`system()`** function can be used to give access to the operating system's command line. This function works slightly differently on Microsoft® Windows® and Linux/macOS systems:

1. on **Linux and macOS systems** the **`system()`** function executes the command passed to it as a string. For example, the statement:

   ```
   system("ls");
   ```

   will list all folders and files located in the current working directory, while the statement:

   ```
   system("bash myScript.sh");
   ```

will execute the commands contained in the shell script file `myScript.sh` (assuming that such a file exists in the current directory). In both cases, the output of the command passed to **system()** will be directed to the BayES console.

2. on **Microsoft® Windows® systems** the **system()** function executes the command passed to it as a string, after prepending it with the string `"cmd /Q /C "`. For example, when the user runs:

```
system("dir");
```

the command submitted to the Microsoft® Windows® shell is actually `"cmd /Q /C dir"`. This is done so that the **system()** function can call both Microsoft® Windows® applications, with a statement like:

```
system("notepad");
```

as well as Microsoft® Windows® DOS commands, such as the `dir` command used in the example above.

`cmd` requests Microsoft® Windows® to start a new shell to execute the command or start the application and the two option specifiers have the following effect:

- `/C`: terminates the shell after the command finishes execution
- `/Q`: turns echo off

A statement like:

```
system("myScript.bat");
```

will execute the DOS commands contained in the `myScript.bat` batch file (assuming that such a file exists in the current directory). If the command to be executed contains spaces, then it can be enclosed in double-quote marks. For example:

```
system("\"copy myFile.txt .\AnotherFolder\"");
```

In all cases, any output sent to the Microsoft® Windows® console by the program or command will be redirected to the BayES console.

The script file `"1-System control statements.bsf"`, located at `"$BayESHOME/Samples/6-AdvancedUsage"` contains an extensive example of using the **system()** function. Note that this file differs between Microsoft® Windows® and Linux/macOS systems and the BayES installer saves only the relevant file for the respective host system.

Chapter 4

Linear Models

4.1 Basic linear model

Mathematical representation

$$y_i = \mathbf{x}_i'\boldsymbol{\beta} + \varepsilon_i, \qquad \varepsilon_i \sim \mathrm{N}\left(0, \tfrac{1}{\tau}\right) \tag{4.1}$$

- the model is estimated using N observations
- y_i is the value of the dependent variable for observation i
- $\mathbf{x}_i$ is a $K \times 1$ vector that stores the values of the K independent variables for observation i
- $\boldsymbol{\beta}$ is a $K \times 1$ vector of parameters
- τ is the precision of the error term: $\sigma_\varepsilon^2 = \frac{1}{\tau}$

Priors

Parameter	Probability density function	Default hyperparameters
$\boldsymbol{\beta}$	$\mathrm{p}(\boldsymbol{\beta}) = \frac{\lvert\mathbf{P}\rvert^{1/2}}{(2\pi)^{K/2}} \exp\left\{-\frac{1}{2}(\boldsymbol{\beta}-\mathbf{m})'\mathbf{P}(\boldsymbol{\beta}-\mathbf{m})\right\}$	$\mathbf{m} = \mathbf{0}_K$, $\mathbf{P} = 0.001 \cdot \mathbf{I}_K$
τ	$\mathrm{p}(\tau) = \frac{b_\tau^{a_\tau}}{\Gamma(a_\tau)}\tau^{a_\tau - 1}e^{-\tau b_\tau}$	$a_\tau = 0.001$, $b_\tau = 0.001$

Syntax

```
[<model name> = ] lm( y ~ x1 x2 ... xK [, <options> ] );
```

where:

- `y` is the dependent variable name, as it appears in the dataset used for estimation
- `x1 x2 ... xK` is a list of the K independent variable names, as they appear in the dataset used for estimation; when a constant term is to be included in the model, this must be requested explicitly

The optional arguments for the simple linear model are:[1]

Gibbs parameters	
`"chains"`	number of chains to run in parallel (positive integer); the default value is 1
`"burnin"`	number of burn-in draws per chain (positive integer); the default value is 10000
`"draws"`	number of retained draws per chain (positive integer); the default value is 20000
`"thin"`	value of the thinning parameter (positive integer); the default value is 1
`"seed"`	value of the seed for the random-number generator (positive integer); the default value is 42
Hyperparameters	
`"m"`	mean vector of the prior for $\boldsymbol{\beta}$ ($K \times 1$ vector); the default value is $\mathbf{0}_K$
`"P"`	precision matrix of the prior for $\boldsymbol{\beta}$ ($K \times K$ symmetric and positive-definite matrix); the default value is $0.001 \cdot \mathbf{I}_K$
`"a_tau"`	shape parameter of the prior for τ (positive number); the default value is 0.001
`"b_tau"`	rate parameter of the prior for τ (positive number); the default value is 0.001
Dataset and log-marginal likelihood	
`"dataset"`	the id value of the dataset that will be used for estimation; the default value is the first dataset in memory (in alphabetical order)

[1] Optional arguments are always given in option-value pairs (eg. `"chains"=3`).

`"logML_CJ"`	boolean indicating whether the Chib (1995)/Chib & Jeliazkov (2001) approximation to the log-marginal likelihood should be calculated (**true**|**false**); the default value is **false**

Reported Parameters

$\boldsymbol{\beta}$	`variable_name`	vector of parameters associated with the independent variables
τ	`tau`	precision parameter of the error term, ε_i
σ_ε	`sigma_e`	standard deviation of the error term: $\sigma_\varepsilon = 1/\tau^{1/2}$

Stored values and post-estimation analysis

If a left-hand-side id value is provided when a simple linear model is created, then the following results are saved in the model item and are accessible via the '.' operator:

`Samples`	a matrix containing the draws from the posterior of $\boldsymbol{\beta}$ and τ
`x1,...,xK`	vectors containing the draws from the posterior of the parameters associated with variables `x1,...,xK` (the names of these vectors are the names of the variables that were included in the right-hand side of the model)
`tau`	vector containing the draws from the posterior of τ
`logML`	the Lewis & Raftery (1997) approximation of the log-marginal likelihood
`logML_CJ`	the Chib (1995)/Chib & Jeliazkov (2001) approximation to the log-marginal likelihood; this is available only if the model was estimated with the `"logML_CJ"=true` option
`nchains`	the number of chains that were used to estimate the model
`nburnin`	the number of burn-in draws per chain that were used when estimating the model
`ndraws`	the total number of retained draws from the posterior (=`chains` · `draws`)
`nthin`	value of the thinning parameter that was used when estimating the model
`nseed`	value of the seed for the random-number generator that was used when estimating the model

Additionally, the following functions are available for post-estimation analysis (see section B.14):

- `diagnostics()`
- `pmp()`
- `test()`

Examples

Example 1

```
myData = import("$BayESHOME/Datasets/dataset1.csv");
myData.constant = ones(rows(myData), 1);

lm( y ~ constant x1 x2 x3);
```

Example 2

```
myData = import("$BayESHOME/Datasets/dataset1.csv");
myData.constant = ones(rows(myData), 1);

myModel = lm(y ~ constant x1 x2 x3,
    "m"=ones(4,1), "P" = 0.1*eye(4,4),
    "a_tau"=0.01, "b_tau"=0.01,
    "burnin"=10000, "draws"=40000, "thin"=4, "chains"=2,
    "logML_CJ" = true, "dataset"=myData);

diagnostics("model"=myModel);

plot(myModel.tau,
    "title"="draws from the posterior of tau",
    "grid"="on");
```

4.2 Heteroskedastic linear model

Mathematical representation

$$y_i = \mathbf{x}_i'\boldsymbol{\beta} + \varepsilon_i \qquad \varepsilon_i \sim \mathrm{N}\left(0, \tfrac{1}{\tau_i}\right) \tag{4.2}$$

$$\log \tau_i = \mathbf{w}_i'\boldsymbol{\delta} + v_i \qquad v_i \sim \mathrm{N}\left(0, \tfrac{1}{\phi}\right) \tag{4.3}$$

- the model is estimated using N observations
- y_i is the value of the dependent variable for observation i
- $\mathbf{x}_i$ is a $K \times 1$ vector that stores the values of the K independent variables for observation i
- $\boldsymbol{\beta}$ is a $K \times 1$ vector of parameters
- τ_i is the precision of the error term for observation i: $\sigma^2_{\varepsilon_i} = \frac{1}{\tau_i}$
- $\mathbf{w}_i$ is an $L \times 1$ vector that stores the values of the L variables that determine the precision of the error term for observation i
- $\boldsymbol{\delta}$ is an $L \times 1$ vector of parameters
- ϕ is the precision of the error term in the equation for $\log \tau_i$: $\sigma^2_v = \frac{1}{\phi}$

Priors

Parameter	Probability density function	Default hyperparameters
$\boldsymbol{\beta}$	$\mathrm{p}(\boldsymbol{\beta}) = \frac{\lvert\mathbf{P}_\beta\rvert^{1/2}}{(2\pi)^{K/2}} \exp\left\{-\frac{1}{2}(\boldsymbol{\beta} - \mathbf{m}_\beta)' \mathbf{P}_\beta (\boldsymbol{\beta} - \mathbf{m}_\beta)\right\}$	$\mathbf{m}_\beta = \mathbf{0}_K$, $\mathbf{P}_\beta = 0.001 \cdot \mathbf{I}_K$
$\boldsymbol{\delta}$	$\mathrm{p}(\boldsymbol{\delta}) = \frac{\lvert\mathbf{P}_\delta\rvert^{1/2}}{(2\pi)^{L/2}} \exp\left\{-\frac{1}{2}(\boldsymbol{\delta} - \mathbf{m}_\delta)' \mathbf{P}_\delta (\boldsymbol{\delta} - \mathbf{m}_\delta)\right\}$	$\mathbf{m}_\delta = \mathbf{0}_L$, $\mathbf{P}_\delta = 0.01 \cdot \mathbf{I}_L$
ϕ	$\mathrm{p}(\phi) = \frac{b_\phi^{a_\phi}}{\Gamma(a_\phi)} \phi^{a_\phi - 1} e^{-\phi b_\phi}$	$a_\phi = 0.001$, $b_\phi = 0.001$

Syntax

```
[<model name> = ] lm( y ~ x1 x2 ... xK | w1 w2 ... wL [,<options>] );
```

where:

- `y` is the dependent variable name, as it appears in the dataset used for estimation
- `x1 x2 ... xK` is a list of the K independent variable names, as they appear in the dataset used for estimation; when a constant term is to be included in the model, this must be requested explicitly
- `w1 w2 ... wK` is a list of the names of the L variables which determine the precision of ε_i, as they appear in the dataset used for estimation; when a constant term is to be included in the precision equation, this must be requested explicitly

The optional arguments for the heteroskedastic linear model are:[2]

Gibbs parameters	
`"chains"`	number of chains to run in parallel (positive integer); the default value is 1
`"burnin"`	number of burn-in draws per chain (positive integer); the default value is 10000
`"draws"`	number of retained draws per chain (positive integer); the default value is 20000
`"thin"`	value of the thinning parameter (positive integer); the default value is 1

[2]Optional arguments are always given in option-value pairs (eg. `"chains"=3`).

`"seed"`	value of the seed for the random-number generator (positive integer); the default value is 42
`Hyperparameters`	
`"m_beta"`	mean vector of the prior for $\boldsymbol{\beta}$ ($K \times 1$ vector); the default value is $\mathbf{0}_K$
`"P_beta"`	precision matrix of the prior for $\boldsymbol{\beta}$ ($K \times K$ symmetric and positive-definite matrix); the default value is $0.001 \cdot \mathbf{I}_K$
`"m_delta"`	mean vector of the prior for $\boldsymbol{\delta}$ ($L \times 1$ vector); the default value is $\mathbf{0}_L$
`"P_delta"`	precision matrix of the prior for $\boldsymbol{\delta}$ ($L \times L$ symmetric and positive-definite matrix); the default value is $0.01 \cdot \mathbf{I}_L$
`"a_phi"`	shape parameter of the prior for ϕ (positive number); the default value is 0.001
`"b_phi"`	rate parameter of the prior for ϕ (positive number); the default value is 0.001
`Dataset and log-marginal likelihood`	
`"dataset"`	the id value of the dataset that will be used for estimation; the default value is the first dataset in memory (in alphabetical order)
`"logML_CJ"`	boolean indicating whether the Chib (1995)/Chib & Jeliazkov (2001) approximation to the log-marginal likelihood should be calculated (**true**|**false**); the default value is **false**

Reported Parameters

$\boldsymbol{\beta}$	`variable_name`	vector of parameters associated with the independent variables in the observed equation
$\boldsymbol{\delta}$	`variable_name`	vector of parameters associated with the independent variables in the precision equation
ϕ	`phi`	precision parameter of the error term in the precision equation, v_i
σ_v	`sigma_v`	standard deviation of the error term in the precision equation: $\sigma_v = 1/\phi^{1/2}$

Stored values and post-estimation analysis

If a left-hand-side id value is provided when a heteroskedastic linear model is created, then the following results are saved in the model item and are accessible via the '.' operator:

`Samples`	a matrix containing the draws from the posterior of $\boldsymbol{\beta}$, $\boldsymbol{\delta}$ and ϕ
`y$x1,...,y$xK`	vectors containing the draws from the posterior of the parameters associated with variables `x1,...,xK` (the names of these vectors are the names of the variables that were included in the right-hand side of the model, prepended by `y$`, where `y` is the name of the dependent variable; this is done so that the samples on the parameters associated with a variable that appears in both `x` and `w` lists can be distinguished)
`logtau$z1,..., logtau$zL`	vectors containing the draws from the posterior of the parameters associated with variables `w1,...,wL` (the names of these vectors are the names of the variables that were included in the `w` list, in the right-hand side of the model, prepended by `logtau$`; this is done so that the samples on the parameters associated with a variable that appears in both `x` and `w` lists can be distinguished)
`phi`	vector containing the draws from the posterior of ϕ
`logML`	the Lewis & Raftery (1997) approximation of the log-marginal likelihood
`logML_CJ`	the Chib (1995)/Chib & Jeliazkov (2001) approximation to the log-marginal likelihood; this is available only if the model was estimated with the `"logML_CJ"=true` option
`nchains`	the number of chains that were used to estimate the model

`nburnin`	the number of burn-in draws per chain that were used when estimating the model
`ndraws`	the total number of retained draws from the posterior (=`chains` · `draws`)
`nthin`	value of the thinning parameter that was used when estimating the model
`nseed`	value of the seed for the random-number generator that was used when estimating the model

Additionally, the following functions are available for post-estimation analysis (see section B.14):

- `diagnostics()`
- `pmp()`
- `test()`

Examples

Example 1

```
myData = import("$BayESHOME/Datasets/dataset3.csv");
myData.constant = ones(rows(myData), 1);

lm( y ~ constant x1 x2 x3 | constant z1 z2);
```

Example 2

```
myData = import("$BayESHOME/Datasets/dataset3.csv");
myData.constant = ones(rows(myData), 1);

myModel = lm(y ~ constant x1 x2 x3 | constant z1 z2,
    "m_beta"=ones(4,1), "P_beta" = 0.01*eye(4,4),
    "m_delta"=ones(3,1), "P_delta" = 0.1*eye(3,3),
    "a_phi"=0.01, "b_phi"=0.001,
    "burnin"=10000, "draws"=40000, "thin"=4, "chains"=2,
    "logML_CJ" = true, "dataset"=myData);

diagnostics("model"=myModel);

plotdraws(phi, "model"=myModel);
plotdraws(logtau$z2, "model"=myModel);
```

4.3 Random-effects linear model

Mathematical representation

$$y_{it} = \alpha_i + \mathbf{x}'_{it}\boldsymbol{\beta} + \varepsilon_{it}, \qquad \varepsilon_{it} \sim \mathrm{N}\left(0, \tfrac{1}{\tau}\right), \quad \alpha_i \sim \mathrm{N}\left(0, \tfrac{1}{\omega}\right) \tag{4.4}$$

- the model is estimated using observations from N groups, each group observed for T_i periods (balanced or unbalanced panels); the total number of observations is $\sum_{i=1}^{N} T_i$
- y_{it} is the value of the dependent variable for group i, observed in period t
- $\mathbf{x}_{it}$ is a $K \times 1$ vector that stores the values of the K independent variables for group i, observed in period t
- $\boldsymbol{\beta}$ is a $K \times 1$ vector of parameters
- τ is the precision of the observation-specific error term: $\sigma_\varepsilon^2 = \frac{1}{\tau}$
- α_i is the group-specific error term for group i
- ω is the precision of the group-specific error term: $\sigma_\alpha^2 = \frac{1}{\omega}$

The mean of the distribution of the α_is is restricted to zero and, therefore, these are simply group-specific errors terms. However, including a constant term in the set of independent variables is valid and leads to a specification equivalent to one where the group effects are draws from a normal distribution with mean equal to the parameter associated with the constant term and precision ω.

Priors

Parameter	Probability density function	Default hyperparameters
$\boldsymbol{\beta}$	$\mathrm{p}(\boldsymbol{\beta}) = \frac{\lvert\mathbf{P}\rvert^{1/2}}{(2\pi)^{K/2}} \exp\left\{-\frac{1}{2}(\boldsymbol{\beta}-\mathbf{m})' \mathbf{P} (\boldsymbol{\beta}-\mathbf{m})\right\}$	$\mathbf{m} = \mathbf{0}_K$, $\mathbf{P} = 0.001 \cdot \mathbf{I}_K$
τ	$\mathrm{p}(\tau) = \frac{b_\tau^{a_\tau}}{\Gamma(a_\tau)} \tau^{a_\tau - 1} e^{-\tau b_\tau}$	$a_\tau = 0.001$, $b_\tau = 0.001$
ω	$\mathrm{p}(\omega) = \frac{b_\omega^{a_\omega}}{\Gamma(a_\omega)} \omega^{a_\omega - 1} e^{-\omega b_\omega}$	$a_\omega = 0.01$, $b_\omega = 0.001$

Syntax

```
[<model name> = ] lm_re( y ~ x1 x2 ... xK [, <options> ] );
```

where:

- `y` is the dependent variable name, as it appears in the dataset used for estimation
- `x1 x2 ... xK` is a list of the K independent variable names, as they appear in the dataset used for estimation; when a constant term is to be included in the model, this must be requested explicitly

Before using the `lm_re()` function the dataset used for estimation must be declared as a panel dataset using the `set_pd()` function (see section B.13).

BayES automatically drops from the sample used for estimation groups which are observed only once. This is because for these groups the group effect (α_i) cannot be distinguished from the error term (ε_{it}).

The optional arguments for the random-effects linear model are:[3]

Gibbs parameters	
`"chains"`	number of chains to run in parallel (positive integer); the default value is 1
`"burnin"`	number of burn-in draws per chain (positive integer); the default value is 10000
`"draws"`	number of retained draws per chain (positive integer); the default value is 20000
`"thin"`	value of the thinning parameter (positive integer); the default value is 1
`"seed"`	value of the seed for the random-number generator (positive integer); the default value is 42
Hyperparameters	
`"m"`	mean vector of the prior for $\boldsymbol{\beta}$ ($K \times 1$ vector); the default value is $\mathbf{0}_K$
`"P"`	precision matrix of the prior for $\boldsymbol{\beta}$ ($K \times K$ symmetric and positive-definite matrix); the default value is $0.001 \cdot \mathbf{I}_K$
`"a_tau"`	shape parameter of the prior for τ (positive number); the default value is 0.001
`"b_tau"`	rate parameter of the prior for τ (positive number); the default value is 0.001
`"a_omega"`	shape parameter of the prior for ω (positive number); the default value is 0.01
`"b_omega"`	rate parameter of the prior for ω (positive number); the default value is 0.001
Dataset and log-marginal likelihood	
`"dataset"`	the id value of the dataset that will be used for estimation; the default value is the first dataset in memory (in alphabetical order)
`"logML_CJ"`	boolean indicating whether the Chib (1995)/Chib & Jeliazkov (2001) approximation to the log-marginal likelihood should be calculated (**true**|**false**); the default value is **false**

Reported Parameters

$\boldsymbol{\beta}$	`variable_name`	vector of parameters associated with the independent variables
τ	`tau`	precision parameter of the observation-specific error term, ε_{it}
ω	`omega`	precision parameter of the group-specific error term, α_i
σ_ε	`sigma_e`	standard deviation of the observation-specific error term: $\sigma_\varepsilon = 1/\tau^{1/2}$
σ_α	`sigma_alpha`	standard deviation of the group-specific error term: $\sigma_\alpha = 1/\omega^{1/2}$

Stored values and post-estimation analysis

If a left-hand-side id value is provided when a random-effects linear model is created, then the following results are saved in the model item and are accessible via the '.' operator:

`Samples`	a matrix containing the draws from the posterior of $\boldsymbol{\beta}$, τ and ω
`x1,...,xK`	vectors containing the draws from the posterior of the parameters associated with variables `x1,...,xK` (the names of these vectors are the names of the variables that were included in the right-hand side of the model)
`tau`	vector containing the draws from the posterior of τ
`omega`	vector containing the draws from the posterior of ω
`logML`	the Lewis & Raftery (1997) approximation of the log-marginal likelihood
`logML_CJ`	the Chib (1995)/Chib & Jeliazkov (2001) approximation to the log-marginal likelihood; this is available only if the model was estimated with the `"logML_CJ"=true` option

[3] Optional arguments are always given in option-value pairs (eg. `"chains"=3`).

`alpha_i`	$N \times 1$ vector that stores the group-specific errors; the values in this vector are not guaranteed to be in the same order as the order in which the groups appear in the dataset used for estimation; use the `store()` function to associate the values in `alpha_i` with the observations in the dataset
`nchains`	the number of chains that were used to estimate the model
`nburnin`	the number of burn-in draws per chain that were used when estimating the model
`ndraws`	the total number of retained draws from the posterior (=`chains` · `draws`)
`nthin`	value of the thinning parameter that was used when estimating the model
`nseed`	value of the seed for the random-number generator that was used when estimating the model

Additionally, the following functions are available for post-estimation analysis (see section B.14):

- `diagnostics()`
- `pmp()`
- `test()`
- `store()`

The random-effects linear model uses the `store()` function to associate the group effects (`alpha_i`) with specific observations and store their values in the dataset used for estimation. The generic syntax for a statement involving the `store()` function after estimation of a random-effects linear model is:

```
store( alpha_i , <new variable name>, ["model"=<model name>] );
```

Examples

Example 1

```
myData = import("$BayESHOME/Datasets/dataset2.csv", ",");
myData.constant = ones(rows(myData), 1);
set_pd( year, id, "dataset" = myData);

lm_re( y ~ constant x1 x2 x3 );
```

Example 2

```
myData = import("$BayESHOME/Datasets/dataset2.csv", ",");
myData.constant = ones(rows(myData), 1);
set_pd( year, id, "dataset" = myData);

myModel = lm_re(y ~ constant x1 x2 x3,
    "m"=ones(4,1), "P" = 0.1*eye(4,4),
    "a_tau"=0.01, "b_tau"=0.01,
    "a_omega"=0.1, "b_omega"=0.01,
    "burnin"=10000, "draws"=40000, "thin"=4, "chains"=2,
    "logML_CJ" = true, "dataset"=myData);

diagnostics("model"=myModel);

store( alpha_i, re, "model" = myModel );

test( myModel.omega > 8 );
```

4.4 Random-coefficients linear model

Mathematical representation

$$y_{it} = \mathbf{z}_{it}'\boldsymbol{\gamma}_i + \mathbf{x}_{it}'\boldsymbol{\beta} + \varepsilon_{it}, \qquad \varepsilon_{it} \sim \mathrm{N}\left(0, \tfrac{1}{\tau}\right), \quad \boldsymbol{\gamma}_i \sim \mathrm{N}\left(\bar{\boldsymbol{\gamma}}, \boldsymbol{\Omega}^{-1}\right) \tag{4.5}$$

- the model is estimated using observations from N groups, each group observed for T_i periods (balanced or unbalanced panels); the total number of observations is $\sum_{i=1}^{N} T_i$ and T_i could be equal to one for all i (cross-sectional data)
- y_{it} is the value of the dependent variable for group i, observed in period t
- $\mathbf{z}_{it}$ is a $K \times 1$ vector that stores the values of the K independent variables which are associated with group-specific coefficients, for group i, observed in period t
- $\mathbf{x}_{it}$ is an $L \times 1$ vector that stores the values of the L independent variables which are associated with coefficients common to all groups, for group i, observed in period t (L could be zero)
- $\boldsymbol{\gamma}_i$ is a $K \times 1$ vector of parameters associated with group i
- $\bar{\boldsymbol{\gamma}}$ is a $K \times 1$ vector of parameters that represents the mean of the $\boldsymbol{\gamma}_i$s
- $\boldsymbol{\Omega}$ is a $K \times K$ precision matrix for the distribution of the $\boldsymbol{\gamma}_i$s
- $\boldsymbol{\beta}$ is an $L \times 1$ vector of parameters
- τ is the precision of the error term: $\sigma_\varepsilon^2 = \frac{1}{\tau}$

Priors

Parameter	Probability density function	Default hyperparameters
$\bar{\boldsymbol{\gamma}}$	$\mathrm{p}\left(\bar{\boldsymbol{\gamma}}\right) = \frac{\lvert\mathbf{P}_\gamma\rvert^{1/2}}{(2\pi)^{K/2}} \exp\left\{-\frac{1}{2}\left(\bar{\boldsymbol{\gamma}} - \mathbf{m}_\gamma\right)' \mathbf{P}_\gamma \left(\bar{\boldsymbol{\gamma}} - \mathbf{m}_\gamma\right)\right\}$	$\mathbf{m}_\gamma = \mathbf{0}_K$, $\mathbf{P}_\gamma = 0.001 \cdot \mathbf{I}_K$
$\boldsymbol{\Omega}$	$\mathrm{p}\left(\boldsymbol{\Omega}\right) = \frac{\lvert\boldsymbol{\Omega}\rvert^{\frac{n-K-1}{2}} \lvert\mathbf{V}^{-1}\rvert^{n/2}}{2^{nK/2}\Gamma_K\left(\frac{n}{2}\right)} \exp\left\{-\frac{1}{2}\operatorname{tr}\left(\mathbf{V}^{-1}\boldsymbol{\Omega}\right)\right\}$	$n = K^2$, $\mathbf{V} = \frac{100}{K} \cdot \mathbf{I}_K$
$\boldsymbol{\beta}$	$\mathrm{p}\left(\boldsymbol{\beta}\right) = \frac{\lvert\mathbf{P}_\beta\rvert^{1/2}}{(2\pi)^{L/2}} \exp\left\{-\frac{1}{2}\left(\boldsymbol{\beta} - \mathbf{m}_\beta\right)' \mathbf{P}_\beta \left(\boldsymbol{\beta} - \mathbf{m}_\beta\right)\right\}$	$\mathbf{m}_\beta = \mathbf{0}_L$, $\mathbf{P}_\beta = 0.001 \cdot \mathbf{I}_L$
τ	$\mathrm{p}\left(\tau\right) = \frac{b_\tau^{a_\tau}}{\Gamma(a_\tau)} \tau^{a_\tau - 1} e^{-\tau b_\tau}$	$a_\tau = 0.001$, $b_\tau = 0.001$

Syntax

```
[<model name> = ] lm_rc( y ~ z1 z2 ... zK [| x1 x2 ... xL ] [, <options> ] );
```

where:

- `y` is the dependent variable name, as it appears in the dataset used for estimation
- `z1 z2 ... zK` is a list of the names, as they appear in the dataset used for estimation, of the independent variables which are associated with group-specific coefficients; when a constant term is to be included in the set of group-specific coefficients this must be requested explicitly
- `x1 x2 ... xL` is is a list of the names, as they appear in the dataset used for estimation, of the independent variables which are associated with coefficients common to all groups; when a constant term is to be included in the set of common coefficients, this must be requested explicitly

An independent variable could be included in either the `x` or the `z` variable list, depending on whether the parameter associated with this variable is common to all groups or not. However, including a variable in both lists would lead to exact multicollinearity and, in this case, BayES will issue an error.

Before using the `lm_rc()` function the dataset used for estimation must be declared as a panel dataset using the `set_pd()` function (see section B.13). In the case of cross-sectional data, the dataset still needs to be declared as a panel, but the group-id variable could be constructed as a list of unique integers using, for example, the `range()` function.

For groups observed only once, a group-specific parameter associated with a constant term cannot be distinguished from the error term (ε_{it}). Thus, a warning is produced when a constant term is included in the `z` list and the dataset contains at least one group which is observed only once.

The optional arguments for the random-coefficients linear model are:[4]

`Gibbs parameters`	
`"chains"`	number of chains to run in parallel (positive integer); the default value is 1
`"burnin"`	number of burn-in draws per chain (positive integer); the default value is 10000
`"draws"`	number of retained draws per chain (positive integer); the default value is 20000
`"thin"`	value of the thinning parameter (positive integer); the default value is 1
`"seed"`	value of the seed for the random-number generator (positive integer); the default value is 42
`Hyperparameters`	
`"m_gamma"`	mean vector of the prior for $\bar{\gamma}$ ($K \times 1$ vector); the default value is $\mathbf{0}_K$
`"P_gamma"`	precision matrix of the prior for $\bar{\gamma}$ ($K \times K$ symmetric and positive-definite matrix); the default value is $0.001 \cdot \mathbf{I}_K$
`"V"`	scale matrix of the prior for Ω ($K \times K$ symmetric and positive-definite matrix); the default value is $\frac{100}{K} \cdot \mathbf{I}_K$
`"n"`	degrees-of-freedom parameter of the prior for $\mathbf{\Omega}$ (real number greater than or equal to K); the default value is K^2
`"m_beta"`	mean vector of the prior for $\boldsymbol{\beta}$ ($L \times 1$ vector); the default value is $\mathbf{0}_L$
`"P_beta"`	precision matrix of the prior for $\boldsymbol{\beta}$ ($L \times L$ symmetric and positive-definite matrix); the default value is $0.001 \cdot \mathbf{I}_L$
`"a_tau"`	shape parameter of the prior for τ (positive number); the default value is 0.001
`"b_tau"`	rate parameter of the prior for τ (positive number); the default value is 0.001
`Dataset and log-marginal likelihood`	
`"dataset"`	the id value of the dataset that will be used for estimation; the default value is the first dataset in memory (in alphabetical order)
`"logML_CJ"`	boolean indicating whether the Chib (1995)/Chib & Jeliazkov (2001) approximation to the log-marginal likelihood should be calculated (**true**|**false**); the default value is **false**

Reported Parameters

$\bar{\gamma}$	`variable_name`	vector of parameters associated with the independent variables in the `z` list; these are the means of the group-specific parameters
$\boldsymbol{\beta}$	`variable_name`	vector of parameters associated with the independent variables in the `x` list
τ	`tau`	precision parameter of the error term, ε_i
σ_ε	`sigma_e`	standard deviation of the error term: $\sigma_\varepsilon = 1/\tau^{1/2}$

[4] Optional arguments are always given in option-value pairs (eg. `"chains"=3`).

Stored values and post-estimation analysis

If a left-hand-side id value is provided when a random-coefficients linear model is created, then the following results are saved in the model item and are accessible via the '.' operator:

`Samples`	a matrix containing the draws from the posterior of $\bar{\gamma}$, $\boldsymbol{\beta}$, τ and the unique elements of $\boldsymbol{\Omega}$
`z1,...,zK`	vectors containing the draws from the posterior of the mean of the group-specific coefficients ($\bar{\gamma}$s) associated with variables `z1,...,zK` (the names of these vectors are the names of the variables that were included in the right-hand side of the model)
`x1,...,xL`	vectors containing the draws from the posterior of the parameters associated with variables `x1,...,xL` (the names of these vectors are the names of the variables that were included in the right-hand side of the model)
`tau`	vector containing the draws from the posterior of τ
`Omega_i_j`	vectors containing the draws from the posterior of the unique elements of $\boldsymbol{\Omega}$; because $\boldsymbol{\Omega}$ is symmetric, only $\frac{(K-1)K}{2}+K$ of its elements are stored (instead of all K^2 elements); `i` and `j` index the row and column of $\boldsymbol{\Omega}$, respectively, at which the corresponding element is located
`Omega`	$K \times K$ matrix that stores the posterior mean of $\boldsymbol{\Omega}$
`logML`	the Lewis & Raftery (1997) approximation of the log-marginal likelihood
`logML_CJ`	the Chib (1995)/Chib & Jeliazkov (2001) approximation to the log-marginal likelihood; this is available only if the model was estimated with the `"logML_CJ"=true` option
`gamma_i`	$N \times K$ matrix that stores the group-specific coefficients for the variables in the `z` list; the values in this matrix are not guaranteed to be in the same order as the order in which the groups appear in the dataset used for estimation; use the `store()` function to associate the values in `gamma_i` with the observations in the dataset
`nchains`	the number of chains that were used to estimate the model
`nburnin`	the number of burn-in draws per chain that were used when estimating the model
`ndraws`	the total number of retained draws from the posterior (=`chains` · `draws`)
`nthin`	value of the thinning parameter that was used when estimating the model
`nseed`	value of the seed for the random-number generator that was used when estimating the model

Additionally, the following functions are available for post-estimation analysis (see section B.14):

- `diagnostics()`
- `pmp()`
- `test()`
- `store()`

The random-coefficients linear model uses the `store()` function to associate the group-specific parameters (`gamma_i`) with specific observations and store their values in the dataset used for estimation. The generic syntax for a statement involving the `store()` function after estimation of a random-coefficients linear model is:

```
store( gamma_i , <new variable name prefix>, ["model"=<model name>] );
```

This statement will generate K additional variables in the dataset used for estimation of the random-coefficients model, with names constructed by prepending the prefix provided as the second argument to `store()` to the names of the variables which are associated with group-specific coefficients.

Examples

Example 1

```
myData = import("$BayESHOME/Datasets/dataset2.csv", ",");
myData.constant = ones(rows(myData), 1);
set_pd( year, id, "dataset" = myData);

// all rhs variables are associated with group-specific coefficients
lm_rc( y ~ constant x1 x2 x3);
```

Example 2

```
myData = import("$BayESHOME/Datasets/dataset2.csv", ",");
myData.constant = ones(rows(myData), 1);
set_pd( year, id, "dataset" = myData);

// only the constant term and the coefficient associated with x1 are
// group-specific; the coefficients on x2 and x3 are common to all groups
lm_rc( y ~ constant x1 | x2 x3);
```

Example 3

```
myData = import("$BayESHOME/Datasets/dataset2.csv", ",");
myData.constant = ones(rows(myData), 1);
set_pd( year, id, "dataset" = myData);

// all rhs variables are associated with group-specific coefficients
model1 = lm_rc(y ~ constant x1 x2 x3,
    "burnin"=10000, "draws"=40000, "thin"=4, "chains"=2,
    "logML_CJ" = true, "dataset"=myData);
store( gamma_i, rc1_, "model" = model1 );

// only the constant term and the coefficient associated with x1 are
// group-specific; the coefficients on x2 and x3 are common to all groups
model2 = lm_rc( y ~ constant x1 | x2 x3,
    "burnin"=10000, "draws"=40000, "thin"=4, "chains"=2,
    "logML_CJ" = true, "dataset"=myData);
store( gamma_i, rc2_, "model" = model2 );

pmp( { model1, model2 } );
```

4.5 Latent-class linear model

Mathematical representation

$$y_{i|c} = \mathbf{x}_i'\boldsymbol{\beta}_c + \varepsilon_{i|c}, \qquad \varepsilon_{i|c} \sim \mathrm{N}\left(0, \tfrac{1}{\tau_c}\right), \qquad c = 0, 1, \ldots, C-1 \tag{4.6}$$

- the model is estimated using N observations and involves C classes (counting starts at zero)
- y_i is the value of the dependent variable for observation i
- $\mathbf{x}_i$ is a $K \times 1$ vector that stores the values of the K independent variables for observation i
- $\boldsymbol{\beta}_c$ is a $K \times 1$ vector of parameters for class c
- τ_c is the precision of the error term for class c: $\sigma^2_{\varepsilon,c} = \frac{1}{\tau_c}$
- each observation, i, belongs to class c with prior probability (before seeing the data $\{y_i, \mathbf{x}_i\}$) $\pi_{i,c}$. BayES supports two types of models:

  1. unconditional prior class membership probabilities, in which case:

     $$\pi_{i,c} = \pi_c \quad \forall i, c$$

     With this specification $\boldsymbol{\pi} \equiv \begin{bmatrix}\pi_0 & \pi_1 & \cdots & \pi_{C-1}\end{bmatrix}'$ is a vector of parameters to be estimated, with $\pi_c > 0\ \forall c$ and $\sum_{c=0}^{C-1} \pi_c = 1$.

  2. conditional prior class membership probabilities, in which case:

     $$\pi_{i,c} = \frac{e^{\mathbf{z}_i'\boldsymbol{\delta}_c}}{\sum_{\ell=0}^{C-1} e^{\mathbf{z}_i'\boldsymbol{\delta}_\ell}} \quad \forall i, c$$

     where:

     - $\mathbf{z}_i$ is an $L \times 1$ vector that stores the values of the L determinants of class-membership for observation i
     - $\boldsymbol{\delta} \equiv \begin{bmatrix}\boldsymbol{\delta}_1' & \boldsymbol{\delta}_2' & \cdots \boldsymbol{\delta}_{C-1}'\end{bmatrix}'$ is an $L \cdot (C-1) \times 1$ vector of parameters to be estimated

     In this specification class-membership probabilities are determined by a multinomial Logit model, where, for identification purposes, $\boldsymbol{\delta}_0$ is normalized to an $L \times 1$ vector of zeros.

Priors

Parameter	Probability density function	Default hyperparameters
`Common to both model types`		
$\boldsymbol{\beta}_c$	$\mathrm{p}(\boldsymbol{\beta}_c) = \frac{\lvert\mathbf{P}_c\rvert^{1/2}}{(2\pi)^{K/2}} \exp\left\{-\frac{1}{2}(\boldsymbol{\beta}_c - \mathbf{m}_c)'\mathbf{P}_c(\boldsymbol{\beta}_c - \mathbf{m}_c)\right\}$	$\mathbf{m}_c = \mathbf{0}_K$, $\mathbf{P}_c = 0.001 \cdot \mathbf{I}_K$
τ_c	$\mathrm{p}(\tau_c) = \frac{b_{\tau_c}^{a_{\tau_c}}}{\Gamma(a_{\tau_c})}\tau_c^{a_{\tau_c}-1}e^{-\tau_c b_{\tau_c}}$	$a_{\tau_c} = 0.001$, $b_{\tau_c} = 0.001$
`Model with unconditional class-membership probabilities`		
$\boldsymbol{\pi}$	$\mathrm{p}(\boldsymbol{\pi}) = \frac{1}{B(\mathbf{a})}\prod_{c=0}^{C-1}\pi_c^{a_c-1}$	$a_0 = a_1 = \cdots = a_{C-1} = 1$
`Model with conditional class-membership probabilities`		
$\boldsymbol{\delta}$	$\mathrm{p}(\boldsymbol{\delta}) = \frac{\lvert\mathbf{P}_\delta\rvert^{1/2}}{(2\pi)^{\frac{L(C-1)}{2}}} \exp\left\{-\frac{1}{2}(\boldsymbol{\delta} - \mathbf{m}_\delta)'\mathbf{P}_\delta(\boldsymbol{\delta} - \mathbf{m}_\delta)\right\}$	$\mathbf{m}_\delta = \mathbf{0}_{L(C-1)}$, $\mathbf{P}_\delta = 0.001 \cdot \mathbf{I}_{L(C-1)}$

Syntax

```
[<model name> = ] lm_lc( y ~ x1 x2 ... xK [ | z1 z2 ... zL] [, <options> ] );
```

where:

- `y` is the dependent variable name, as it appears in the dataset used for estimation
- `x1 x2 ... xK` is a list of the K independent variable names, as they appear in the dataset used for estimation; when a constant term is to be included in the model, this must be requested explicitly
- `z1 z2 ... zL` is a list of the L variable names that enter the specification of the class-membership probabilities (determinants of class membership), as they appear in the dataset used for estimation; when a constant term is to be included in the model, this must be requested explicitly; this list is optional and when provided the conditional latent-class model is estimated; if not provided the unconditional model is estimated

If the dataset used for estimation has been previously declared as a panel dataset (typically, by a call to the `set_pd()` function) then the model estimated is the one documented in the following section. Each group in that model is restricted to belong to the same class for the entire period for which it is observed.

The optional arguments for the latent-class linear model are:[5]

Gibbs parameters	
`"chains"`	number of chains to run in parallel (positive integer); the default value is 1
`"burnin"`	number of burn-in draws per chain (positive integer); the default value is 10000
`"draws"`	number of retained draws per chain (positive integer); the default value is 20000
`"thin"`	value of the thinning parameter (positive integer); the default value is 1
`"seed"`	value of the seed for the random-number generator (positive integer); the default value is 42
Model specification	
`"classes"`	specification of the number of classes to be used in the model (positive integer); the default value is 2
Hyperparameters	
Common to both model types	
`"m"`	mean vector of the prior for each $\boldsymbol{\beta}_c$ ($K \times 1$ vector); the default value is $\mathbf{0}_K$
`"P"`	precision matrix of the prior for each $\boldsymbol{\beta}_c$ ($K \times K$ symmetric and positive-definite matrix); the default value is $0.001 \cdot \mathbf{I}_K$
`"mj"`	mean vector of the prior for $\boldsymbol{\beta}_j$, $j = 0, 1, \ldots, C-1$ ($K \times 1$ vector); this mean overwrites the generic mean (`"m"`) for class j only
`"Pj"`	precision matrix of the prior for $\boldsymbol{\beta}_j$, $j = 0, 1, \ldots, C-1$ ($K \times K$ symmetric and positive-definite matrix); this precision matrix overwrites the generic precision matrix (`"P"`) for class j only
`"a_tau"`	shape parameter of the prior for each τ_c (positive number); the default value is 0.001
`"b_tau"`	rate parameter of the prior for each τ_c (positive number); the default value is 0.001
`"a_tauj"`	shape parameter of the prior for τ_j, $j = 0, 1, \ldots, C-1$ (positive number); this shape parameter overwrites the generic shape parameter (`"a_tau"`) for class j only

[5] Optional arguments are always given in option-value pairs (eg. `"chains"=3`).

`"b_tauj"`	rate parameter of the prior for τ_j, $j = 0, 1, \ldots, C-1$ (positive number); this rate parameter overwrites the generic rate parameter (`"b_tau"`) for class j only
`Model with unconditional class-membership probabilities`	
`"a"`	vector of concentration parameters for the Dirichlet prior on $\boldsymbol{\pi}$ ($C \times 1$ vector with positive entries); the default value is a $C \times 1$ vector of ones
`Model with conditional class-membership probabilities`	
`"m_delta"`	mean vector of the prior for $\boldsymbol{\delta}$ ($L\,(C-1) \times 1$ vector); the default value is $\mathbf{0}_{L(C-1)}$
`"P_delta"`	precision matrix of the prior for $\boldsymbol{\delta}$ ($L\,(C-1) \times 1$ symmetric and positive-definite matrix); the default value is $0.001 \cdot \mathbf{I}_{L(C-1)}$
`Dataset and log-marginal likelihood`	
`"dataset"`	the id value of the dataset that will be used for estimation; the default value is the first dataset in memory (in alphabetical order)
`"logML_CJ"`	boolean indicating whether the Chib (1995)/Chib & Jeliazkov (2001) approximation to the log-marginal likelihood should be calculated (**true**|**false**); the default value is **false**

Reported Parameters

`Common to both model types`		
$\boldsymbol{\beta}_c$	`variable_name`	vector of parameters associated with the independent variables for class c
τ_c	`tau`	precision parameter of the error term for class c, $\varepsilon_{i\|c}$
$\sigma_{\varepsilon,c}$	`sigma_e`	standard deviation of the error term for class c: $\sigma_{\varepsilon,c} = 1/\tau_c^{1/2}$
`Model with unconditional class-membership probabilities`		
π_c	`pi`	prior class-membership probability for class c
`Model with conditional class-membership probabilities`		
$\boldsymbol{\delta}_c$	`variable_name`	vector of parameters associated with the determinants of class membership for class c; for identification purposes, these parameters for class 0 are normalized to zero

Stored values and post-estimation analysis

If a left-hand-side id value is provided when a latent-class linear model is created, then the following results are saved in the model item and are accessible via the '.' operator:

`Samples`	a matrix containing the draws from the posterior of $\boldsymbol{\beta}_c$ and τ_c for $c = 0, 1, \ldots, C-1$, and, depending on the estimated model, $\boldsymbol{\pi}$ or $\boldsymbol{\delta}$.
`cj$x1,...,cj$xK`	vectors containing the draws from the posterior of the parameters associated with variables `x1,...,xK`, for $j = 0, 1, \ldots, C-1$ (the names of these vectors are the names of the variables that were included in the right-hand side of the model, prepended by '`c`', the class index and a dollar sign; in this way '`cj`' can be used to distinguish among parameters across different classes)
`cj$tau`	vectors containing the draws from the posterior of each τ_c, for $j = 0, 1, \ldots, C-1$ ('`tau`' is prepended by '`c`', the class index and the dollar sign; in this way '`cj`' can be used to distinguish among precision parameters in different classes)
`pi_j`	vectors containing the draws from the posterior of each π_c, for $j = 0, 1, \ldots, C-1$ (these vectors are available only after the estimation of the model with unconditional class-membership probabilities)

`pi_j$z1,...,` `pi_j$zL`	vectors containing the draws from the posterior of the parameters associated with variables `z1,...,zL`, for $j = 1, \ldots, C-1$[6] (the names of these vectors are the names of the variables that were included the `z` list, in the right-hand side of the model, prepended by '`pi_j`' and the dollar sign; in this way '`pi_j`' can be used to distinguish among parameters associated with variables with different roles in the model, for example the same variable appearing in both `x` and `z` lists, as well as among parameters associated with a variable in the `z` list, but corresponding to different classes; these vectors are available only after the estimation of the model with conditional class-membership probabilities)
`logML`	the Lewis & Raftery (1997) approximation of the log-marginal likelihood
`logML_CJ`	the Chib (1995)/Chib & Jeliazkov (2001) approximation to the log-marginal likelihood; this is available only if the model was estimated with the `"logML_CJ"=true` option
`pi_i`	$N \times C$ matrix that stores the expected values of the posterior class-membership probabilities for each observation and for each of the C classes
`nchains`	the number of chains that were used to estimate the model
`nburnin`	the number of burn-in draws per chain that were used when estimating the model
`ndraws`	the total number of retained draws from the posterior (=`chains` · `draws`)
`nthin`	value of the thinning parameter that was used when estimating the model
`nseed`	value of the seed for the random-number generator that was used when estimating the model
`nclasses`	number of classes used during the estimation of the model

Additionally, the following functions are available for post-estimation analysis (see section B.14):

- `diagnostics()`
- `test()`
- `pmp()`
- `store()`
- `mfx()`

The latent-class linear model uses the `store()` function to associate the estimates of the posterior class-membership probabilities (`pi_i`) with specific observations and store their values in the dataset used for estimation. The generic syntax for a statement involving the `store()` function after estimation of a latent-class linear model is:

```
store( pi_i, <new variable name prefix> [, "model"=<model name>] );
```

This statement will generate C additional variables in the dataset used for estimation of the model, with names constructed by appending the class index $(0, 1, \ldots, C-1)$ to the prefix provided as the second argument to `store()`.

The latent-class linear model with conditional class-membership probabilities uses the `mfx()` function to calculate and report the marginal effects of the variables in the `z` list on the prior class-membership probabilities that come from the multinomial-Logit part of the model. The generic syntax for a statement involving the `mfx()` function after estimation of a latent-class linear model with conditional class-membership probabilities is:

```
mfx( ["type"=1] [, "point"=<point of calculation>] [, "model"=<model name>] );
```

See the general documentation of the `mfx()` function (section B.14) for details on the optional arguments.

[6] Indexing starts at 1 because the parameters of the multinomial-Logit part of the model associated with class 0 are normalized to zero for identification purposes.

Examples

Example 1

```
myData = import("$BayESHOME/Datasets/dataset2.csv");
myData.constant = 1;

lm_lc(y ~ constant x1 x2);
```

Example 2

```
myData = import("$BayESHOME/Datasets/dataset2.csv");
myData.constant = 1;

myModel = lm_lc(y ~ constant x1 x2,
    "m0"=[2;0.6;0.3], "P" = 10*eye(3,3),
    "burnin"=10000, "draws"=30000, "thin"=2, "chains"=2, "classes"=2,
    "logML_CJ" = true, "dataset"=myData);

diagnostics("model"=myModel);

plot([myModel.c0$x1, myModel.c1$x1],
    "title"="\beta2 for the two classes");
plot([myModel.pi_0, myModel.pi_1],
    "title"="Prior class-membership probabilities");
```

Example 3

```
myData = import("$BayESHOME/Datasets/dataset2.csv");
myData.constant = 1;

myModel = lm_lc(y ~ constant x1 x2 | constant x3,
    "m0"=[2;0.6;0.3], "P" = 10*eye(3,3),
    "burnin"=10000, "draws"=30000, "thin"=2, "chains"=2, "classes"=2,
    "logML_CJ" = true, "dataset"=myData);

diagnostics("model"=myModel);
mfx("model"=myModel);

plot([myModel.c0$x1, myModel.c1$x1],
    "title"="\beta2 for the two classes");
plot(myModel.pi_1$x3,
    "title"="\delta2 for class 1");
```

4.6 Latent-class linear model with panel data

Mathematical representation

$$y_{it|c} = \mathbf{x}'_{it}\boldsymbol{\beta}_c + \varepsilon_{it|c}, \qquad \varepsilon_{it|c} \sim \mathrm{N}\left(0, \tfrac{1}{\tau_c}\right), \qquad c = 0, 1, \ldots, C-1 \tag{4.7}$$

- the model is estimated using observations from N groups, each group observed for T_i periods (balanced or unbalanced panels); the total number of observations is $\sum_{i=1}^{N} T_i$
- the model involves C classes (counting starts at zero) and each group, i, is restricted to belong to the same class for all T_i observations
- y_{it} is the value of the dependent variable for group i, observed in period t
- $\mathbf{x}_{it}$ is a $K \times 1$ vector that stores the values of the K independent variables for group i, observed in period t
- $\boldsymbol{\beta}_c$ is a $K \times 1$ vector of parameters for class c
- τ_c is the precision of the error term for class c: $\sigma^2_{\varepsilon,c} = \frac{1}{\tau_c}$
- each group, i, belongs to class c with prior probability (before seeing the data $\{y_{it}, \mathbf{x}_{it}\}_{t=1}^{T_i}$) $\pi_{i,c}$. BayES supports two types of models:

  1. unconditional prior class membership probabilities, in which case:

     $$\pi_{i,c} = \pi_c \quad \forall i, c$$

     With this specification $\boldsymbol{\pi} \equiv \begin{bmatrix}\pi_0 & \pi_1 & \cdots & \pi_{C-1}\end{bmatrix}'$ is a vector of parameters to be estimated, with $\pi_c > 0\ \forall c$ and $\sum_{c=0}^{C-1} \pi_c = 1$.

  2. conditional prior class membership probabilities, in which case:

     $$\pi_{i,c} = \frac{e^{\mathbf{z}'_i \boldsymbol{\delta}_c}}{\sum_{\ell=0}^{C-1} e^{\mathbf{z}'_i \boldsymbol{\delta}_\ell}} \quad \forall i, c$$

     where:

     - $\mathbf{z}_i$ is an $L \times 1$ vector that stores the values of the L determinants of class-membership for group i; these variables vary by group only (not within group)
     - $\boldsymbol{\delta} \equiv \begin{bmatrix}\boldsymbol{\delta}'_1 & \boldsymbol{\delta}'_2 & \cdots \boldsymbol{\delta}'_{C-1}\end{bmatrix}'$ is an $L \cdot (C-1) \times 1$ vector of parameters to be estimated

     In this specification class-membership probabilities are determined by a multinomial Logit model, where, for identification purposes, $\boldsymbol{\delta}_0$ is normalized to an $L \times 1$ vector of zeros.

Priors

Parameter	Probability density function	Default hyperparameters
`Common to both model types`		
$\boldsymbol{\beta}_c$	$\mathrm{p}(\boldsymbol{\beta}_c) = \frac{\lvert\mathbf{P}_c\rvert^{1/2}}{(2\pi)^{K/2}} \exp\left\{-\frac{1}{2}(\boldsymbol{\beta}_c - \mathbf{m}_c)' \mathbf{P}_c (\boldsymbol{\beta}_c - \mathbf{m}_c)\right\}$	$\mathbf{m}_c = \mathbf{0}_K$, $\mathbf{P}_c = 0.001 \cdot \mathbf{I}_K$
τ_c	$\mathrm{p}(\tau_c) = \frac{b_{\tau_c}^{a_{\tau_c}}}{\Gamma(a_{\tau_c})} \tau_c^{a_{\tau_c}-1} e^{-\tau_c b_{\tau_c}}$	$a_{\tau_c} = 0.001$, $b_{\tau_c} = 0.001$
`Model with unconditional class-membership probabilities`		
$\boldsymbol{\pi}$	$\mathrm{p}(\boldsymbol{\pi}) = \frac{1}{B(\mathbf{a})} \prod_{c=0}^{C-1} \pi_c^{a_c - 1}$	$a_0 = a_1 = \cdots = a_{C-1} = 1$
`Model with conditional class-membership probabilities`		
$\boldsymbol{\delta}$	$\mathrm{p}(\boldsymbol{\delta}) = \frac{\lvert\mathbf{P}_\delta\rvert^{1/2}}{(2\pi)^{\frac{L(C-1)}{2}}} \exp\left\{-\frac{1}{2}(\boldsymbol{\delta} - \mathbf{m}_\delta)' \mathbf{P}_\delta (\boldsymbol{\delta} - \mathbf{m}_\delta)\right\}$	$\mathbf{m}_\delta = \mathbf{0}_{L(C-1)}$, $\mathbf{P}_\delta = 0.001 \cdot \mathbf{I}_{L(C-1)}$

Syntax

```
[<model name> = ] lm_lc( y ~ x1 x2 ... xK [ | z1 z2 ... zL] [, <options> ] );
```

where:

- `y` is the dependent variable name, as it appears in the dataset used for estimation
- `x1 x2 ... xK` is a list of the K independent variable names, as they appear in the dataset used for estimation; when a constant term is to be included in the model, this must be requested explicitly
- `z1 z2 ... zL` is a list of the L variable names that enter the specification of the class-membership probabilities (determinants of class membership), as they appear in the dataset used for estimation; when a constant term is to be included in the model, this must be requested explicitly; this list is optional and when provided the conditional latent-class model is estimated; if not provided the unconditional model is estimated; the values of the variables in this list must be constant within each group

If the dataset used for estimation has not been previously declared as a panel dataset or if this structure has been removed (by a call to the `set_cs()` function) then the model estimated is the one documented in the preceding section. Different time observations from the same group in that model are allowed to belong to different classes.

The optional arguments for the latent-class linear model with panel data are:[7]

Gibbs parameters	
`"chains"`	number of chains to run in parallel (positive integer); the default value is 1
`"burnin"`	number of burn-in draws per chain (positive integer); the default value is 10000
`"draws"`	number of retained draws per chain (positive integer); the default value is 20000
`"thin"`	value of the thinning parameter (positive integer); the default value is 1
`"seed"`	value of the seed for the random-number generator (positive integer); the default value is 42
Model specification	
`"classes"`	specification of the number of classes to be used in the model (positive integer); the default value is 2
Hyperparameters	
Common to both model types	
`"m"`	mean vector of the prior for each $\boldsymbol{\beta}_c$ ($K\times 1$ vector); the default value is $\mathbf{0}_K$
`"P"`	precision matrix of the prior for each $\boldsymbol{\beta}_c$ ($K\times K$ symmetric and positive-definite matrix); the default value is $0.001\cdot\mathbf{I}_K$
`"mj"`	mean vector of the prior for $\boldsymbol{\beta}_j$, $j=0,1,\ldots,C-1$ ($K\times 1$ vector); this mean overwrites the generic mean (`"m"`) for class j only
`"Pj"`	precision matrix of the prior for $\boldsymbol{\beta}_j$, $j=0,1,\ldots,C-1$ ($K\times K$ symmetric and positive-definite matrix); this precision matrix overwrites the generic precision matrix (`"P"`) for class j only
`"a_tau"`	shape parameter of the prior for each τ_c (positive number); the default value is 0.001
`"b_tau"`	rate parameter of the prior for each τ_c (positive number); the default value is 0.001
`"a_tauj"`	shape parameter of the prior for τ_j, $j=0,1,\ldots,C-1$ (positive number); this shape parameter overwrites the generic shape parameter (`"a_tau"`) for class j only

[7] Optional arguments are always given in option-value pairs (eg. `"chains"=3`).

`"b_tauj"`	rate parameter of the prior for τ_j, $j = 0, 1, \ldots, C-1$ (positive number); this rate parameter overwrites the generic rate parameter (`"b_tau"`) for class j only
`Model with unconditional class-membership probabilities`	
`"a"`	vector of concentration parameters for the Dirichlet prior on $\boldsymbol{\pi}$ ($C \times 1$ vector with positive entries); the default value is a $C \times 1$ vector of ones
`Model with conditional class-membership probabilities`	
`"m_delta"`	mean vector of the prior for $\boldsymbol{\delta}$ ($L\,(C-1) \times 1$ vector); the default value is $\mathbf{0}_{L(C-1)}$
`"P_delta"`	precision matrix of the prior for $\boldsymbol{\delta}$ ($L\,(C-1) \times 1$ symmetric and positive-definite matrix); the default value is $0.001 \cdot \mathbf{I}_{L(C-1)}$
`Dataset and log-marginal likelihood`	
`"dataset"`	the id value of the dataset that will be used for estimation; the default value is the first dataset in memory (in alphabetical order)
`"logML_CJ"`	boolean indicating whether the Chib (1995)/Chib & Jeliazkov (2001) approximation to the log-marginal likelihood should be calculated (**true**|**false**); the default value is **false**

Reported Parameters

`Common to both model types`		
$\boldsymbol{\beta}_c$	`variable_name`	vector of parameters associated with the independent variables for class c
τ_c	`tau`	precision parameter of the error term for class c, $\varepsilon_{it\|c}$
$\sigma_{\varepsilon,c}$	`sigma_e`	standard deviation of the error term for class c: $\sigma_{\varepsilon,c} = 1/\tau_c^{1/2}$
`Model with unconditional class-membership probabilities`		
π_c	`pi`	prior class-membership probability for class c
`Model with conditional class-membership probabilities`		
$\boldsymbol{\delta}_c$	`variable_name`	vector of parameters associated with the determinants of class membership for class c; for identification purposes, these parameters for class 0 are normalized to zero

Stored values and post-estimation analysis

If a left-hand-side id value is provided when a latent-class linear model with panel data is created, then the following results are saved in the model item and are accessible via the '.' operator:

`Samples`	a matrix containing the draws from the posterior of $\boldsymbol{\beta}_c$ and τ_c for $c = 0, 1, \ldots, C-1$, and, depending on the estimated model, $\boldsymbol{\pi}$ or $\boldsymbol{\delta}$.
`cj$x1,...,cj$xK`	vectors containing the draws from the posterior of the parameters associated with variables `x1,...,xK`, for $j = 0, 1, \ldots, C-1$ (the names of these vectors are the names of the variables that were included in the right-hand side of the model, prepended by '`c`', the class index and a dollar sign; in this way '`cj`' can be used to distinguish among parameters across different classes)
`cj$tau`	vectors containing the draws from the posterior of each τ_c, for $j = 0, 1, \ldots, C-1$ ('`tau`' is prepended by '`c`', the class index and the dollar sign; in this way '`cj`' can be used to distinguish among precision parameters in different classes)
`pi_j`	vectors containing the draws from the posterior of each π_c, for $j = 0, 1, \ldots, C-1$ (these vectors are available only after the estimation of the model with unconditional class-membership probabilities)

`pi_j$z1,...,` `pi_j$zL`	vectors containing the draws from the posterior of the parameters associated with variables `z1,...,zL`, for $j = 1, \ldots, C-1$[8] (the names of these vectors are the names of the variables that were included the `z` list, in the right-hand side of the model, prepended by '`pi_j`' and the dollar sign; in this way '`pi_j`' can be used to distinguish among parameters associated with variables with different roles in the model, for example the same variable appearing in both `x` and `z` lists, as well as among parameters associated with a variable in the `z` list, but corresponding to different classes; these vectors are available only after the estimation of the model with conditional class-membership probabilities)
`logML`	the Lewis & Raftery (1997) approximation of the log-marginal likelihood
`logML_CJ`	the Chib (1995)/Chib & Jeliazkov (2001) approximation to the log-marginal likelihood; this is available only if the model was estimated with the `"logML_CJ"=true` option
`pi_i`	$N \times C$ matrix that stores the expected values of the posterior class-membership probabilities for each group and for each of the C classes
`nchains`	the number of chains that were used to estimate the model
`nburnin`	the number of burn-in draws per chain that were used when estimating the model
`ndraws`	the total number of retained draws from the posterior (=`chains` · `draws`)
`nthin`	value of the thinning parameter that was used when estimating the model
`nseed`	value of the seed for the random-number generator that was used when estimating the model
`nclasses`	number of classes used during the estimation of the model

Additionally, the following functions are available for post-estimation analysis (see section B.14):

- `diagnostics()`
- `test()`
- `pmp()`
- `store()`
- `mfx()`

The latent-class linear model with panel data uses the `store()` function to associate the estimates of the posterior class-membership probabilities (`pi_i`) with specific observations and store their values in the dataset used for estimation. The generic syntax for a statement involving the `store()` function after estimation of a latent-class linear model with panel data is:

```
store( pi_i , <new variable name prefix> [, "model"=<model name>] );
```

This statement will generate C additional variables in the dataset used for estimation of the model, with names constructed by appending the class index $(0, 1, \ldots, C-1)$ to the prefix provided as the second argument to `store()`.

The latent-class linear model with panel data and conditional class-membership probabilities uses the `mfx()` function to calculate and report the marginal effects of the variables in the `z` list on the prior class-membership probabilities that come from the multinomial-Logit part of the model. The generic syntax for a statement involving the `mfx()` function after estimation of a latent-class linear model with panel data and conditional class-membership probabilities is:

```
mfx( ["type"=1] [, "point"=<point of calculation>] [, "model"=<model name>] );
```

See the general documentation of the `mfx()` function (section B.14) for details on the optional arguments.

[8] Indexing starts at 1 because the parameters of the multinomial-Logit part of the model associated with class 0 are normalized to zero for identification purposes.

Examples

Example 1

```
myData = import("$BayESHOME/Datasets/dataset2.csv");
myData.constant = 1;

set_pd( year, id, "dataset" = myData);
lm_lc(y ~ constant x1 x2);
```

Example 2

```
myData = import("$BayESHOME/Datasets/dataset2.csv");
myData.constant = 1;

set_pd( year, id, "dataset" = myData);
myModel = lm_lc(y ~ constant x1 x2,
    "m0"=[2;0.6;0.3], "P" = 10*eye(3,3),
    "burnin"=10000, "draws"=30000, "thin"=2, "chains"=2, "classes"=2,
    "logML_CJ" = true, "dataset"=myData);

diagnostics("model"=myModel);

plot([myModel.c0$x1, myModel.c1$x1],
    "title"="\beta2 for the two classes");
plot([myModel.pi_0, myModel.pi_1],
    "title"="Prior class-membership probabilities");
```

Chapter 5

Stochastic Frontier Models

5.1 Simple stochastic frontier

Mathematical representation

$$y_i = \mathbf{x}_i'\boldsymbol{\beta} + v_i \pm u_i, \qquad v_i \sim \mathrm{N}\left(0, \tfrac{1}{\tau}\right), \quad u_i \sim \mathrm{D}\left(\boldsymbol{\theta}\right) \tag{5.1}$$

- the model is estimated using N observations
- y_i is the value of the dependent variable for observation i
- $\mathbf{x}_i$ is a $K \times 1$ vector that stores the values of the K independent variables for observation i
- $\boldsymbol{\beta}$ is a $K \times 1$ vector of parameters
- τ is the precision of the noise component of the error term: $\sigma_v^2 = \frac{1}{\tau}$
- u_i is the inefficiency component of the error term and it can have any non-negative distribution, represented in the equation above by $\mathrm{D}\left(\boldsymbol{\theta}\right)$; BayES supports the following distributions for u_i:
  - exponential: $\mathrm{p}\left(u_i\right) = \lambda e^{-\lambda u_i}$
  - half normal: $\mathrm{p}\left(u_i\right) = \frac{2\phi^{1/2}}{(2\pi)^{1/2}} \exp\left\{-\frac{\phi}{2} u_i^2\right\}$
  - truncated normal: $\mathrm{p}\left(u_i\right) = \frac{\phi^{1/2} \exp\left\{-\frac{\phi}{2}(u_i-\mu)^2\right\}}{(2\pi)^{1/2}\Phi^{1/2}\left(\phi^{1/2}\mu\right)}$
  - gamma: $\mathrm{p}\left(u_i\right) = \frac{\lambda^\kappa}{\Gamma(\kappa)} u_i^{\kappa-1} e^{-\lambda u_i}$
  - log-Normal: $\mathrm{p}\left(u_i\right) = \frac{\phi^{1/2}}{(2\pi)^{1/2} u_i} \exp\left\{-\frac{\phi}{2}\left(\log u_i - \mu\right)^2\right\}$

When u_i enters the specification with a plus sign then the model represents a cost frontier, while when u_i enters with a minus sign the model represents a production frontier. For the efficiency scores generated by a stochastic frontier model to be meaningful, the dependent variable in both cases must be in logarithms.

Priors

Parameter	Probability density function	Default hyperparameters
`Common to all models`		
$\boldsymbol{\beta}$	$\mathrm{p}\left(\boldsymbol{\beta}\right) = \frac{\lvert\mathbf{P}\rvert^{1/2}}{(2\pi)^{K/2}} \exp\left\{-\frac{1}{2}\left(\boldsymbol{\beta}-\mathbf{m}\right)'\mathbf{P}\left(\boldsymbol{\beta}-\mathbf{m}\right)\right\}$	$\mathbf{m} = \mathbf{0}_K$, $\mathbf{P} = 0.001 \cdot \mathbf{I}_K$
τ	$\mathrm{p}\left(\tau\right) = \frac{b_\tau^{a_\tau}}{\Gamma(a_\tau)} \tau^{a_\tau - 1} e^{-\tau b_\tau}$	$a_\tau = 0.001$, $b_\tau = 0.001$
`Exponential model`		
λ	$\mathrm{p}\left(\lambda\right) = \frac{b_\lambda^{a_\lambda}}{\Gamma(a_\lambda)} \lambda^{a_\lambda - 1} e^{-\lambda b_\lambda}$	$a_\lambda = 1$, $b_\lambda = 0.15$
`Half normal model`		
ϕ	$\mathrm{p}\left(\phi\right) = \frac{b_\phi^{a_\phi}}{\Gamma(a_\phi)} \phi^{a_\phi - 1} e^{-\phi b_\phi}$	$a_\phi = 7$, $b_\phi = 0.5$
`Truncated normal model`		
μ	$\mathrm{p}\left(\mu\right) = \frac{t_\mu^{1/2}}{(2\pi)^{1/2}} \exp\left\{-\frac{t_\mu}{2}\left(\mu - m_\mu\right)^2\right\}$	$m_\mu = 0$, $t_\mu = 1$
ϕ	$\mathrm{p}\left(\phi\right) = \frac{b_\phi^{a_\phi}}{\Gamma(a_\phi)} \phi^{a_\phi - 1} e^{-\phi b_\phi}$	$a_\phi = 5$, $b_\phi = 0.5$
`Gamma model`		
κ	$\mathrm{p}\left(\kappa\right) = \frac{b_\kappa^{a_\kappa}}{\Gamma(a_\kappa)} \kappa^{a_\kappa - 1} e^{-\kappa b_\kappa}$	$a_\kappa = 3$, $b_\kappa = 2$
λ	$\mathrm{p}\left(\lambda\right) = \frac{b_\lambda^{a_\lambda}}{\Gamma(a_\lambda)} \lambda^{a_\lambda - 1} e^{-\lambda b_\lambda}$	$a_\lambda = \kappa$, $b_\lambda = 0.2$
`Log-normal model`		
μ	$\mathrm{p}\left(\mu\right) = \frac{t_\mu^{1/2}}{(2\pi)^{1/2}} \exp\left\{-\frac{t_\mu}{2}\left(\mu - m_\mu\right)^2\right\}$	$m_\mu = -1.5$, $t_\mu = 1$
ϕ	$\mathrm{p}\left(\phi\right) = \frac{b_\phi^{a_\phi}}{\Gamma(a_\phi)} \phi^{a_\phi - 1} e^{-\phi b_\phi}$	$a_\phi = 2$, $b_\phi = 1$

Syntax

```
[<model name> = ] sf( y ~ x1 x2 ... xK [, <options> ] );
```

where:

- `y` is the dependent variable name, as it appears in the dataset used for estimation
- `x1 x2 ... xK` is a list of the K independent variable names, as they appear in the dataset used for estimation; when a constant term is to be included in the model, this must be requested explicitly

The optional arguments for the simple stochastic frontier model are:[1]

Gibbs parameters	
`"chains"`	number of chains to run in parallel (positive integer); the default value is 1
`"burnin"`	number of burn-in draws per chain (positive integer); the default value is 10000
`"draws"`	number of retained draws per chain (positive integer); the default value is 20000
`"thin"`	value of the thinning parameter (positive integer); the default value is 1
`"seed"`	value of the seed for the random-number generator (positive integer); the default value is 42
Model specification	
`"udist"`	specification of the distribution of the inefficiency component of the error term; the following options are available, corresponding to the distributions presented at the beginning of this section: • `"exp"` • `"tnorm"` • `"lognorm"` • `"hnorm"` • `"gamma"` the default value is `"exp"`
`"production"`	boolean specifying the type of frontier (production/cost); it could be set to either **true** (production) or **false** (cost); the default value is **true**
Hyperparameters	
Common to all models	
`"m"`	mean vector of the prior for $\boldsymbol{\beta}$ ($K \times 1$ vector); the default value is $\mathbf{0}_K$
`"P"`	precision matrix of the prior for $\boldsymbol{\beta}$ ($K \times K$ symmetric and positive-definite matrix); the default value is $0.001 \cdot \mathbf{I}_K$
`"a_tau"`	shape parameter of the prior for τ (positive number); the default value is 0.001
`"b_tau"`	rate parameter of the prior for τ (positive number); the default value is 0.001
Exponential model	
`"a_lambda"`	shape parameter of the prior for λ (positive number); the default value is 1
`"b_lambda"`	rate parameter of the prior for λ (positive number); the default value is 0.15
Half normal model	
`"a_phi"`	shape parameter of the prior for ϕ (positive number); the default value is 7
`"b_phi"`	rate parameter of the prior for ϕ (positive number); the default value is 0.5
Truncated normal model	
`"m_mu"`	location parameter of the prior for μ (real number); the default value is 0
`"t_mu"`	precision parameter of the prior for μ (positive number); the default value is 1
`"a_phi"`	shape parameter of the prior for ϕ (positive number); the default value is 5
`"b_phi"`	rate parameter of the prior for ϕ (positive number); the default value is 0.5
Gamma model	
`"a_kappa"`	shape parameter of the prior for κ (positive number); the default value is 3

[1] Optional arguments are always given in option-value pairs (eg. `"chains"=3`).

`"b_kappa"`	rate parameter of the prior for κ (positive number); the default value is 2
`"b_lambda"`	rate parameter of the prior for λ (positive number); the default value is 0.2
`Log-normal model`	
`"m_mu"`	location parameter of the prior for μ (real number); the default value is -1.5
`"t_mu"`	precision parameter of the prior for μ (positive number); the default value is 1
`"a_phi"`	shape parameter of the prior for ϕ (positive number); the default value is 2
`"b_phi"`	rate parameter of the prior for ϕ (positive number); the default value is 1
`Dataset and log-marginal likelihood`	
`"dataset"`	the id value of the dataset that will be used for estimation; the default value is the first dataset in memory (in alphabetical order)
`"logML_CJ"`	boolean indicating whether the Chib (1995)/Chib & Jeliazkov (2001) approximation to the log-marginal likelihood should be calculated (**true**|**false**); the default value is **false**

Reported Parameters

`Common to all models`		
$\boldsymbol{\beta}$	`variable_name`	vector of parameters associated with the independent variables
τ	`tau`	precision parameter of the noise component of the error term, v_i
σ_v	`sigma_v`	standard deviation of the noise component of the error term, $\sigma_v = 1/\tau^{1/2}$
`Exponential model`		
λ	`lambda`	rate parameter of the distribution of the inefficiency component of the error term, u_i
σ_u	`sigma_u`	scale parameter of the inefficiency component of the error term: $\sigma_u = 1/\lambda$. For the exponential model the standard deviation of u_i is equal to the scale parameter.
`Half normal model`		
ϕ	`phi`	precision parameter of the distribution of the inefficiency component of the error term, u_i
σ_u	`sigma_u`	scale parameter of the inefficiency component of the error term: $\sigma_u = 1/\phi^{1/2}$. The standard deviation of u_i for the half-normal model can be obtained as $\sigma_u\sqrt{1-\frac{2}{\pi}}$.
`Truncated normal model`		
μ	`mu`	location parameter of the distribution of the inefficiency component of the error term, u_i
ϕ	`phi`	precision parameter of the distribution of the inefficiency component of the error term, u_i
σ_u	`sigma_u`	scale parameter of the inefficiency component of the error term: $\sigma_u = 1/\phi^{1/2}$. The standard deviation of u_i for the truncated-normal model can be obtained as $\sigma_u\sqrt{1-2\frac{\mu}{\sigma_u}\phi\left(-\frac{\mu}{\sigma_u}\right)-4\phi^2\left(-\frac{\mu}{\sigma_u}\right)}$.
`Gamma model`		
κ	`kappa`	shape parameter of the distribution of the inefficiency component of the error term, u_i
λ	`lambda`	rate parameter of the distribution of the inefficiency component of the error term, u_i
θ	`theta`	scale parameter of the inefficiency component of the error term: $\theta = 1/\lambda$. The standard deviation of u_i for the Gamma model can be obtained as $\theta\sqrt{\kappa}$.

table continues on next page

table continued from previous page

Log-normal model		
μ	`mu`	location parameter of the distribution of the inefficiency component of the error term, u_i
ϕ	`phi`	precision parameter of the distribution of the inefficiency component of the error term, u_i
σ_u	`sigma_u`	scale parameter of the inefficiency component of the error term: $\sigma_u = 1/\phi^{1/2}$. The standard deviation of u_i for the log-normal model can be obtained as $\sqrt{\left(e^{\sigma_u^2} - 1\right) e^{2\mu+\sigma_u^2}}$.

Stored values and post-estimation analysis

If a left-hand-side id value is provided when a simple stochastic frontier model is created, then the following results are saved in the model item and are accessible via the '.' operator:

`Samples`	a matrix containing the draws from the posterior of $\boldsymbol{\beta}$ and τ, and, depending on the estimated model, λ, μ, ϕ, κ
`x1,...,xK`	vectors containing the draws from the posterior of the parameters associated with variables `x1,...,xK` (the names of these vectors are the names of the variables that were included in the right-hand side of the model)
`tau`	vector containing the draws from the posterior of τ
`lambda`	vector containing the draws from the posterior of λ (available after the estimation of the exponential and Gamma models)
`mu`	vector containing the draws from the posterior of μ (available after the estimation of the truncated-normal and log-normal models)
`phi`	vector containing the draws from the posterior of ϕ (available after the estimation of the half-normal, truncated-normal and log-normal models)
`kappa`	vector containing the draws from the posterior of κ (available after the estimation of the Gamma model)
`logML`	the Lewis & Raftery (1997) approximation of the log-marginal likelihood
`logML_CJ`	the Chib (1995)/Chib & Jeliazkov (2001) approximation to the log-marginal likelihood; this is available only if the model was estimated with the `"logML_CJ"=true` option
`eff_i`	$N \times 1$ vector that stores the expected values of the observation-specific efficiency scores, $\mathrm{E}\left(e^{-u_i}\right)$; the values in this vector are not guaranteed to be in the same order as the order in which the observations appear in the dataset used for estimation; use the `store()` function to associate the values in `eff_i` with the observations in the dataset
`nchains`	the number of chains that were used to estimate the model
`nburnin`	the number of burn-in draws per chain that were used when estimating the model
`ndraws`	the total number of retained draws from the posterior (=`chains` · `draws`)
`nthin`	value of the thinning parameter that was used when estimating the model
`nseed`	value of the seed for the random-number generator that was used when estimating the model

Additionally, the following functions are available for post-estimation analysis (see section B.14):

- `diagnostics()`
- `pmp()`
- `test()`
- `store()`

The simple stochastic frontier model uses the `store()` function to associate the estimates of the efficiency scores (`eff_i`) with specific observations and store their values in the dataset used for estimation. The generic syntax for a statement involving the `store()` function after estimation of a simple stochastic frontier model is:

```
store( eff_i, <new variable name> [, "model"=<model name>] );
```

Examples

Example 1

```
myData = import("$BayESHOME/Datasets/dataset1.csv");
myData.constant = ones(rows(myData), 1);

sf( y ~ constant x1 x2 x3, "logML_CJ" = true );
```

Example 2

```
myData = import("$BayESHOME/Datasets/dataset1.csv");
myData.constant = ones(rows(myData), 1);

expSF = sf( y ~ constant x1 x2 x3,
    "a_lambda"=-log(0.8), "b_lambda"=1.0,
    "logML_CJ"=true );

hnSF = sf( y ~ constant x1 x2 x3,
    "udist" = "hnorm",
    "a_phi"=7.0, "b_phi"=0.5,
    "logML_CJ"=true );

pmp( { expSF, hnSF } );
pmp( { expSF, hnSF }, "logML_CJ"=true);

store( eff_i, eff_exp, "model"=expSF );
store( eff_i, eff_hn, "model"=hnSF );

hist(myData.eff_exp,
    "title"="Efficiency scores from the exponential model",
    "grid"="on");
hist(myData.eff_hn,
    "title"="Efficiency scores from the half-normal model",
    "grid"="on");
```

5.2 Inefficiency-effects stochastic frontier

Mathematical representation

$$y_i = \mathbf{x}_i'\boldsymbol{\beta} + v_i \pm u_i, \qquad v_i \sim \mathrm{N}\left(0, \tfrac{1}{\tau}\right), \quad u_i \sim \mathrm{D}\left(\boldsymbol{\theta}, \mathbf{z}_i\right) \tag{5.2}$$

- the model is estimated using N observations
- y_i is the value of the dependent variable for observation i
- $\mathbf{x}_i$ is a $K\times 1$ vector that stores the values of the K independent variables for observation i
- $\boldsymbol{\beta}$ is a $K\times 1$ vector of parameters
- $\mathbf{z}_i$ is an $L\times 1$ vector that stores the values of the L determinants of inefficiency for observation i
- τ is the precision of the noise component of the error term: $\sigma_v^2 = \frac{1}{\tau}$
- u_i is the inefficiency component of the error term and it can have any non-negative distribution, represented in the equation above by $\mathrm{D}\left(\boldsymbol{\theta}, \mathbf{z}_i\right)$; BayES supports the following distributions for u_i:
  - exponential: $\mathrm{p}\left(u_i\right) = \lambda_i e^{-\lambda_i u_i}$, with $\lambda_i = e^{\mathbf{z}_i'\boldsymbol{\delta}}$ and $\boldsymbol{\delta}$ being an $L\times 1$ vector of parameters to be estimated
  - truncated normal: $\mathrm{p}\left(u_i\right) = \frac{\phi^{1/2}\exp\left\{-\frac{\phi}{2}(u_i-\mu_i)^2\right\}}{(2\pi)^{1/2}\Phi^{1/2}\left(\phi^{1/2}\mu_i\right)}$, with $\mu_i = \mathbf{z}_i'\boldsymbol{\delta}$ and $\boldsymbol{\delta}$ being an $L\times 1$ vector of parameters and ϕ a scalar parameter to be estimated

When u_i enters the specification with a plus sign then the model represents a cost frontier, while when u_i enters with a minus sign the model represents a production frontier. For the efficiency scores generated by a stochastic frontier model to be meaningful, the dependent variable in both cases must be in logarithms.

Priors

Parameter	Probability density function	Default hyperparameters
`Common to all models`		
$\boldsymbol{\beta}$	$\mathrm{p}\left(\boldsymbol{\beta}\right) = \frac{\lvert\mathbf{P}_\beta\rvert^{1/2}}{(2\pi)^{K/2}}\exp\left\{-\frac{1}{2}\left(\boldsymbol{\beta}-\mathbf{m}_\beta\right)'\mathbf{P}_\beta\left(\boldsymbol{\beta}-\mathbf{m}_\beta\right)\right\}$	$\mathbf{m}_\beta = \mathbf{0}_K$, $\mathbf{P}_\beta = 0.001\cdot\mathbf{I}_K$
τ	$\mathrm{p}\left(\tau\right) = \frac{b_\tau^{a_\tau}}{\Gamma(a_\tau)}\tau^{a_\tau-1}e^{-\tau b_\tau}$	$a_\tau = 0.001$, $b_\tau = 0.001$
`Exponential model`		
$\boldsymbol{\delta}$	$\mathrm{p}\left(\boldsymbol{\delta}\right) = \frac{\lvert\mathbf{P}_\delta\rvert^{1/2}}{(2\pi)^{L/2}}\exp\left\{-\frac{1}{2}\left(\boldsymbol{\delta}-\mathbf{m}_\delta\right)'\mathbf{P}_\delta\left(\boldsymbol{\delta}-\mathbf{m}_\delta\right)\right\}$	$\mathbf{m}_\delta = \mathbf{0}_L$, $\mathbf{P}_\delta = 0.01\cdot\mathbf{I}_L$
`Truncated normal model`		
$\boldsymbol{\delta}$	$\mathrm{p}\left(\boldsymbol{\delta}\right) = \frac{\lvert\mathbf{P}_\delta\rvert^{1/2}}{(2\pi)^{L/2}}\exp\left\{-\frac{1}{2}\left(\boldsymbol{\delta}-\mathbf{m}_\delta\right)'\mathbf{P}_\delta\left(\boldsymbol{\delta}-\mathbf{m}_\delta\right)\right\}$	$\mathbf{m}_\delta = \mathbf{0}_L$, $\mathbf{P}_\delta = 0.01\cdot\mathbf{I}_L$
ϕ	$\mathrm{p}\left(\phi\right) = \frac{b_\phi^{a_\phi}}{\Gamma(a_\phi)}\phi^{a_\phi-1}e^{-\phi b_\phi}$	$a_\phi = 4$, $b_\phi = 0.5$

Syntax

```
[<model name> = ] sf( y ~ x1 x2 ... xK | z1 z2 ... zL [, <options>] );
```

where:

- `y` is the dependent variable name, as it appears in the dataset used for estimation
- `x1 x2 ... xK` is a list of the K independent variable names, as they appear in the dataset used for estimation; when a constant term is to be included in the model, this must be requested explicitly

- `z1 z2 ... zL` is a list of the L variable names that affect u_i (determinants of inefficiency), as they appear in the dataset used for estimation; when a constant term is to be included in the model, this must be requested explicitly

The optional arguments for the inefficiency-effects stochastic frontier model are:[2]

Gibbs parameters	
`"chains"`	number of chains to run in parallel (positive integer); the default value is 1
`"burnin"`	number of burn-in draws per chain (positive integer); the default value is 10000
`"draws"`	number of retained draws per chain (positive integer); the default value is 20000
`"thin"`	value of the thinning parameter (positive integer); the default value is 1
`"seed"`	value of the seed for the random-number generator (positive integer); the default value is 42
Model specification	
`"udist"`	specification of the distribution of the inefficiency component of the error term; the following options are available, corresponding to the distributions presented at the beginning of this section: • `"exp"` • `"tnorm"` the default value is `"exp"`
`"production"`	boolean specifying the type of frontier (production/cost); it could be set to either **true** (production) or **false** (cost); the default value is **true**
Hyperparameters	
Common to all models	
`"m_beta"`	mean vector of the prior for $\boldsymbol{\beta}$ ($K\times 1$ vector); the default value is $\mathbf{0}_K$
`"P_beta"`	precision matrix of the prior for $\boldsymbol{\beta}$ ($K\times K$ symmetric and positive-definite matrix); the default value is $0.001\cdot\mathbf{I}_K$
`"a_tau"`	shape parameter of the prior for τ (positive number); the default value is 0.001
`"b_tau"`	rate parameter of the prior for τ (positive number); the default value is 0.001
Exponential model	
`"m_delta"`	mean vector of the prior for $\boldsymbol{\delta}$ ($L\times 1$ vector); the default value is $\mathbf{0}_L$
`"P_delta"`	precision matrix of the prior for $\boldsymbol{\delta}$ ($L\times L$ symmetric and positive-definite matrix); the default value is $0.01\cdot\mathbf{I}_L$
Truncated normal model	
`"m_delta"`	mean vector of the prior for $\boldsymbol{\delta}$ ($L\times 1$ vector); the default value is $\mathbf{0}_L$
`"P_delta"`	precision matrix of the prior for $\boldsymbol{\delta}$ ($L\times L$ symmetric and positive-definite matrix); the default value is $0.01\cdot\mathbf{I}_L$
`"a_phi"`	shape parameter of the prior for ϕ (positive number); the default value is 4
`"b_phi"`	rate parameter of the prior for ϕ (positive number); the default value is 0.5
Dataset and log-marginal likelihood	
`"dataset"`	the id value of the dataset that will be used for estimation; the default value is the first dataset in memory (in alphabetical order)
`"logML_CJ"`	boolean indicating whether the Chib (1995)/Chib & Jeliazkov (2001) approximation to the log-marginal likelihood should be calculated (**true**|**false**); the default value is **false**

[2] Optional arguments are always given in option-value pairs (eg. `"chains"=3`).

Reported Parameters

Common to all models		
$\boldsymbol{\beta}$	variable_name	vector of parameters associated with the independent variables in the x list
τ	tau	precision parameter of the noise component of the error term, v_i
σ_v	sigma_v	standard deviation of the noise component of the error term, $\sigma_v = 1/\tau^{1/2}$
Exponential model		
$\boldsymbol{\delta}$	variable_name	vector of parameters associated with the independent variables in the z list
Truncated normal model		
$\boldsymbol{\delta}$	variable_name	vector of parameters associated with the independent variables in the z list
ϕ	phi	precision parameter of the distribution of the inefficiency component of the error term, u_i
σ_u	sigma_u	scale parameter of the inefficiency component of the error term: $\sigma_u = 1/\phi^{1/2}$.

Stored values and post-estimation analysis

If a left-hand-side id value is provided when an inefficiency-effects stochastic frontier model is created, then the following results are saved in the model item and are accessible via the '.' operator:

Samples	a matrix containing the draws from the posterior of $\boldsymbol{\beta}$ and τ, and, depending on the estimated model, $\boldsymbol{\delta}$, or $\boldsymbol{\delta}$ and ϕ
y$x1,...,y$xK	vectors containing the draws from the posterior of the parameters associated with variables x1,...,xK (the names of these vectors are the names of the variables that were included in the right-hand side of the model, prepended by y$, where y is the name of the dependent variable; this is done so that the samples on the parameters associated with a variable that appears in both x and z lists can be distinguished)
tau	vector containing the draws from the posterior of τ
u$z1,...,u$zL	vectors containing the draws from the posterior of the parameters associated with variables z1,...,zL (the names of these vectors are the names of the variables that were included in the z list, in the right-hand side of the model, prepended by u$; this is done so that the samples on the parameters associated with a variable that appears in both x and z lists can be distinguished)
phi	vector containing the draws from the posterior of ϕ (available after the estimation of the truncated-normal model)
logML	the Lewis & Raftery (1997) approximation of the log-marginal likelihood
logML_CJ	the Chib (1995)/Chib & Jeliazkov (2001) approximation to the log-marginal likelihood; this is available only if the model was estimated with the "logML_CJ"=true option
eff_i	$N \times 1$ vector that stores the expected values of the observation-specific efficiency scores, $\mathrm{E}(e^{-u_i})$; the values in this vector are not guaranteed to be in the same order as the order in which the observations appear in the dataset used for estimation; use the store() function to associate the values in eff_i with the observations in the dataset
nchains	the number of chains that were used to estimate the model
nburnin	the number of burn-in draws per chain that were used when estimating the model
ndraws	the total number of retained draws from the posterior (=chains · draws)

`nthin`	value of the thinning parameter that was used when estimating the model
`nseed`	value of the seed for the random-number generator that was used when estimating the model

Additionally, the following functions are available for post-estimation analysis (see section B.14):

- `diagnostics()`
- `test()`
- `pmp()`
- `store()`
- `mfx()`

The inefficiency-effects stochastic frontier model uses the `store()` function to associate the estimates of the efficiency scores (`eff_i`) with specific observations and store their values in the dataset used for estimation. The generic syntax for a statement involving the `store()` function after estimation of an inefficiency-effects stochastic frontier model is:

```
store( eff_i , <new variable name> [, "model"=<model name>] );
```

The inefficiency-effects stochastic frontier model uses the `mfx()` function to calculate and report the marginal effects of the variables in the `z` list on the expected value of u and on the expected value of the efficiency score ($= e^{-u}$). The two types of marginal effects can be requested by setting the `"type"` argument of the `mfx()` function equal to `1` or `2`. The generic syntax for a statement involving the `mfx()` function after estimation of an inefficiency-effects stochastic frontier model is:

```
mfx( ["type"=1] [, "point"=<point of calculation>] [, "model"=<model name>] );
```

and:

```
mfx( "type"=2 [, "point"=<point of calculation>] [, "model"=<model name>] );
```

for calculation of the marginal effects on $\mathrm{E}(u)$ and on $\mathrm{E}(e^{-u})$, respectively. The default value of the `"type"` option is `1`. See the general documentation of the `mfx()` function (section B.14) for details on the other optional arguments.

Examples

Example 1

```
myData = import("$BayESHOME/Datasets/dataset3.csv", ",");
myData.constant = ones(rows(myData), 1);

sf( y ~ constant x2 x3 x4 | constant z2 z3, "production"=false );
```

Example 2

```
myData = import("$BayESHOME/Datasets/dataset3.csv", ",");
myData.constant = ones(rows(myData), 1);

expSF = sf( y ~ constant x2 x3 x4 | constant z2 z3,
    "udist"="exp", "production"=false );

tnormSF = sf( y ~ constant x2 x3 x4 | constant z2 z3,
    "udist"="tnorm", "production"=false );

store( eff_i, eff_exp, "model" = expSF );
store( eff_i, eff_tnorm, "model" = tnormSF );

mfx( "point"="mean", "model"=expSF, "type"=1 );
mfx( "point"="mean", "model"=tnormSF, "type"=1 );

mfx( "point"="mean", "model"=expSF, "type"=2 );
mfx( "point"="mean", "model"=tnormSF, "type"=2 );

pmp( { expSF, tnormSF } );
```

5.3 Random-effects stochastic frontier

Mathematical representation

$$y_{it} = \alpha_i + \mathbf{x}'_{it}\boldsymbol{\beta} + v_{it} \pm u_{it}, \qquad v_{it} \sim \mathrm{N}\left(0, \tfrac{1}{\tau}\right), \quad \alpha_i \sim \mathrm{N}\left(0, \tfrac{1}{\omega}\right), \quad u_{it} \sim \mathrm{D}\left(\boldsymbol{\theta}\right) \tag{5.3}$$

- the model is estimated using observations from N groups, each group observed for T_i periods (balanced or unbalanced panels); the total number of observations is $\sum_{i=1}^{N} T_i$
- y_{it} is the value of the dependent variable for group i, observed in period t
- $\mathbf{x}_{it}$ is a $K \times 1$ vector that stores the values of the K independent variables for group i, observed in period t
- $\boldsymbol{\beta}$ is a $K \times 1$ vector of parameters
- τ is the precision of the noise component of the error term: $\sigma_v^2 = \frac{1}{\tau}$
- α_i is the group-specific error term for group i
- ω is precision of the group-specific error term: $\sigma_\alpha^2 = \frac{1}{\omega}$
- u_{it} is the inefficiency component of the error term for group i in period t and it can have any non-negative distribution, represented in the equation above by $\mathrm{D}\left(\boldsymbol{\theta}\right)$; BayES supports the following distributions for u_{it}:
  - exponential: $\mathrm{p}\left(u_{it}\right) = \lambda e^{-\lambda u_{it}}$
  - half normal: $\mathrm{p}\left(u_{it}\right) = \frac{2\phi^{1/2}}{(2\pi)^{1/2}} \exp\left\{-\frac{\phi}{2} u_{it}^2\right\}$

When u_{it} enters the specification with a plus sign then the model represents a cost frontier, while when u_{it} enters with a minus sign the model represents a production frontier. For the efficiency scores generated by a stochastic frontier model to be meaningful, the dependent variable in both cases must be in logarithms.

The mean of the distribution of the α_is is restricted to zero and, therefore, these are simply group-specific errors terms. However, including a constant term in the set of independent variables is valid and leads to a specification equivalent to one where the group effects are draws from a normal distribution with mean equal to the parameter associated with the constant term and precision ω.

No time dependence is imposed on the inefficiency component of the error term: each u_{it} is treated as an independent draw from $\mathrm{D}\left(\boldsymbol{\theta}\right)$. This specification is known as the "true random effects" stochastic frontier model (Greene, 2004) .

Priors

Parameter	Probability density function	Default hyperparameters
`Common to all models`		
$\boldsymbol{\beta}$	$\mathrm{p}\left(\boldsymbol{\beta}\right) = \frac{\lvert\mathbf{P}\rvert^{1/2}}{(2\pi)^{K/2}} \exp\left\{-\frac{1}{2}\left(\boldsymbol{\beta}-\mathbf{m}\right)'\mathbf{P}\left(\boldsymbol{\beta}-\mathbf{m}\right)\right\}$	$\mathbf{m} = \mathbf{0}_K$, $\mathbf{P} = 0.001 \cdot \mathbf{I}_K$
τ	$\mathrm{p}\left(\tau\right) = \frac{b_\tau^{a_\tau}}{\Gamma(a_\tau)} \tau^{a_\tau - 1} e^{-\tau b_\tau}$	$a_\tau = 0.001$, $b_\tau = 0.001$
ω	$\mathrm{p}\left(\omega\right) = \frac{b_\omega^{a_\omega}}{\Gamma(a_\omega)} \omega^{a_\omega - 1} e^{-\omega b_\omega}$	$a_\omega = 0.01$, $b_\omega = 0.001$
`Exponential model`		
λ	$\mathrm{p}\left(\lambda\right) = \frac{b_\lambda^{a_\lambda}}{\Gamma(a_\lambda)} \lambda^{a_\lambda - 1} e^{-\lambda b_\lambda}$	$a_\lambda = 1$, $b_\lambda = 0.15$
`Half normal model`		
ϕ	$\mathrm{p}\left(\phi\right) = \frac{b_\phi^{a_\phi}}{\Gamma(a_\phi)} \phi^{a_\phi - 1} e^{-\phi b_\phi}$	$a_\phi = 7$, $b_\phi = 0.5$

Syntax

```
[<model name> = ] sf_re( y ~ x1 x2 ... xK [, <options> ] );
```

where:

- `y` is the dependent variable name, as it appears in the dataset used for estimation
- `x1 x2 ... xK` is a list of the K independent variable names, as they appear in the dataset used for estimation; when a constant term is to be included in the model, this must be requested explicitly

Before using the `sf_re()` function the dataset used for estimation must be declared as a panel dataset using the `set_pd()` function (see section B.13).

BayES automatically drops from the sample used for estimation groups which are observed only once. This is because for these groups the group effect (α_i) cannot be distinguished from the noise component of the error term (v_{it}).

The optional arguments for the random-effects stochastic frontier model are:[3]

Gibbs parameters	
`"chains"`	number of chains to run in parallel (positive integer); the default value is 1
`"burnin"`	number of burn-in draws per chain (positive integer); the default value is 10000
`"draws"`	number of retained draws per chain (positive integer); the default value is 20000
`"thin"`	value of the thinning parameter (positive integer); the default value is 1
`"seed"`	value of the seed for the random-number generator (positive integer); the default value is 42
Model specification	
`"udist"`	specification of the distribution of the inefficiency component of the error term; the following options are available, corresponding to the distributions presented at the beginning of this section: • `"exp"` • `"hnorm"` the default value is `"exp"`
`"production"`	boolean specifying the type of frontier (production/cost); it could be set to either **true** (production) or **false** (cost); the default value is **true**
Hyperparameters	
Common to all models	
`"m"`	mean vector of the prior for $\boldsymbol{\beta}$ ($K \times 1$ vector); the default value is $\mathbf{0}_K$
`"P"`	precision matrix of the prior for $\boldsymbol{\beta}$ ($K \times K$ symmetric and positive-definite matrix); the default value is $0.001 \cdot \mathbf{I}_K$
`"a_tau"`	shape parameter of the prior for τ (positive number); the default value is 0.001
`"b_tau"`	rate parameter of the prior for τ (positive number); the default value is 0.001
`"a_omega"`	shape parameter of the prior for ω (positive number); the default value is 0.01
`"b_omega"`	rate parameter of the prior for ω (positive number); the default value is 0.001

[3] Optional arguments are always given in option-value pairs (eg. `"chains"=3`).

`Exponential model`	
`"a_lambda"`	shape parameter of the prior for λ (positive number); the default value is 1
`"b_lambda"`	rate parameter of the prior for λ (positive number); the default value is 0.15
`Half normal model`	
`"a_phi"`	shape parameter of the prior for ϕ (positive number); the default value is 7
`"b_phi"`	rate parameter of the prior for ϕ (positive number); the default value is 0.5
`Dataset and log-marginal likelihood`	
`"dataset"`	the id value of the dataset that will be used for estimation; the default value is the first dataset in memory in alphabetical order
`"logML_CJ"`	boolean indicating whether the Chib (1995)/Chib & Jeliazkov (2001) approximation to the log-marginal likelihood should be calculated (**true**|**false**); the default value is **false**

Reported Parameters

`Common to all models`		
$\boldsymbol{\beta}$	`variable_name`	vector of parameters associated with the independent variables
τ	`tau`	precision parameter of the noise component of the error term, v_i
ω	`omega`	precision parameter of the group-specific error term, α_i
σ_v	`sigma_v`	standard deviation of the noise component of the error term, $\sigma_v = 1/\tau^{1/2}$
σ_α	`sigma_alpha`	standard deviation of the group-specific error term: $\sigma_\alpha = 1/\omega^{1/2}$
`Exponential model`		
λ	`lambda`	rate parameter of the distribution of the inefficiency component of the error term, u_i
σ_u	`sigma_u`	scale parameter of the inefficiency component of the error term: $\sigma_u = 1/\lambda$. For the exponential model the standard deviation of u_i is equal to the scale parameter.
`Half normal model`		
ϕ	`phi`	precision parameter of the distribution of the inefficiency component of the error term, u_i
σ_u	`sigma_u`	scale parameter of the inefficiency component of the error term: $\sigma_u = 1/\phi^{1/2}$. The standard deviation of u_i for the half-normal model can be obtained as $\sigma_u\sqrt{1-\frac{2}{\pi}}$.

Stored values and post-estimation analysis

If a left-hand-side id value is provided when a random-effects stochastic frontier model is created, then the following results are saved in the model item and are accessible via the '.' operator:

`Samples`	a matrix containing the draws from the posterior of $\boldsymbol{\beta}$, τ and either λ (exponential model) or ϕ (half-normal model)
`x1,...,xK`	vectors containing the draws from the posterior of the parameters associated with variables `x1,...,xK` (the names of these vectors are the names of the variables that were included in the right-hand side of the model)
`tau`	vector containing the draws from the posterior of τ
`omega`	vector containing the draws from the posterior of ω
`lambda`	vector containing the draws from the posterior of λ (available after the estimation of the exponential model)
`phi`	vector containing the draws from the posterior of ϕ (available after the estimation of the half-normal model)
`logML`	the Lewis & Raftery (1997) approximation of the log-marginal likelihood

`logML_CJ`	the Chib (1995)/Chib & Jeliazkov (2001) approximation to the log-marginal likelihood; this is available only if the model was estimated with the `"logML_CJ"=true` option
`alpha_i`	$N \times 1$ vector that stores the group-specific errors; the values in this vector are not guaranteed to be in the same order as the order in which the groups appear in the dataset used for estimation; use the `store()` function to associate the values in `alpha_i` with the observations in the dataset
`eff_i`	$\left(\sum_{i=1}^{N} T_i\right) \times 1$ vector that stores the expected values of the observation-specific efficiency scores, $\mathrm{E}\left(e^{-u_{it}}\right)$; the values in this vector are not guaranteed to be in the same order as the order in which the observations appear in the dataset used for estimation; use the `store()` function to associate the values in `eff_i` with the observations in the dataset
`nchains`	the number of chains that were used to estimate the model
`nburnin`	the number of burn-in draws per chain that were used when estimating the model
`ndraws`	the total number of retained draws from the posterior (=`chains` · `draws`)
`nthin`	value of the thinning parameter that was used when estimating the model
`nseed`	value of the seed for the random-number generator that was used when estimating the model

Additionally, the following functions are available for post-estimation analysis (see section B.14):

- `diagnostics()`
- `pmp()`
- `test()`
- `store()`

The random-effects stochastic frontier model uses the `store()` function to associate the group effects (`alpha_i`) or the estimates of the efficiency scores (`eff_i`) with specific observations and store their values in the dataset used for estimation. The generic syntax for a statement involving the `store()` function after estimation of a random-effects stochastic frontier model and for each of these two quantities is:

```
store( alpha_i , <new variable name> [, "model"=<model name>] );
```

and:

```
store( eff_i , <new variable name> [, "model"=<model name>] );
```

Examples

Example 1

```
myData = import("$BayESHOME/Datasets/dataset2.csv", ",");
myData.constant = ones(rows(myData), 1);
set_pd( year, id, "dataset" = myData);

myModel = sf_re( y ~ constant x1 x2 x3 );
```

Example 2

```
myData = import("$BayESHOME/Datasets/dataset2.csv", ",");
myData.constant = ones(rows(myData), 1);
set_pd( year, id, "dataset" = myData);

exp_SFRE = sf_re( y ~ constant x1 x2 x3, "logML_CJ" = true );

hnorm_SFRE = sf_re( y ~ constant x1 x2 x3, "logML_CJ" = true,
    "udist" = "hnorm" );

store( alpha_i, re_exp, "model" = exp_SFRE );
store( alpha_i, re_hnorm, "model" = hnorm_SFRE );

store( eff_i, eff_exp, "model" = exp_SFRE );
store( eff_i, eff_hnorm, "model" = hnorm_SFRE );

pmp( { exp_SFRE, hnorm_SFRE } );
pmp( { exp_SFRE, hnorm_SFRE }, "logML_CJ"=true );

hist(myData.eff_exp,
    "title"="Efficiency scores from the exponential model",
    "grid"="on");
hist(myData.eff_hn,
    "title"="Efficiency scores from the half-normal model",
    "grid"="on");
```

5.4 Random-coefficients stochastic frontier

Mathematical representation

$$y_{it} = \mathbf{z}'_{it}\boldsymbol{\gamma}_i + \mathbf{x}'_{it}\boldsymbol{\beta} + v_{it} \pm u_{it}, \qquad v_{it} \sim \mathrm{N}\left(0, \tfrac{1}{\tau}\right), \quad \boldsymbol{\gamma}_i \sim \mathrm{N}\left(\bar{\boldsymbol{\gamma}}, \boldsymbol{\Omega}^{-1}\right), \quad u_{it} \sim \mathrm{D}\left(\boldsymbol{\theta}\right) \quad (5.4)$$

- the model is estimated using observations from N groups, each group observed for T_i periods (balanced or unbalanced panels); the total number of observations is $\sum_{i=1}^{N} T_i$ and T_i could be equal to one for all i (cross-sectional data)
- y_{it} is the value of the dependent variable for group i, observed in period t
- $\mathbf{z}_{it}$ is a $K \times 1$ vector that stores the values of the K independent variables which are associated with group-specific coefficients, for group i, observed in period t
- $\mathbf{x}_{it}$ is an $L \times 1$ vector that stores the values of the L independent variables which are associated with coefficients common to all groups, for group i, observed in period t (L could be zero)
- $\boldsymbol{\gamma}_i$ is a $K \times 1$ vector of parameters associated with group i
- $\bar{\boldsymbol{\gamma}}$ is a $K \times 1$ vector of parameters that represents the mean of the $\boldsymbol{\gamma}_i$s
- $\boldsymbol{\Omega}$ is a $K \times K$ precision matrix for the distribution of the $\boldsymbol{\gamma}_i$s
- $\boldsymbol{\beta}$ is an $L \times 1$ vector of parameters
- τ is the precision of the noise component of the error term: $\sigma_v^2 = \frac{1}{\tau}$
- u_{it} is the inefficiency component of the error term for group i in period t and it can have any non-negative distribution, represented in the equation above by $\mathrm{D}\left(\boldsymbol{\theta}\right)$; BayES supports the following distributions for u_{it}:
  - exponential: $\mathrm{p}\left(u_{it}\right) = \lambda e^{-\lambda u_{it}}$
  - half normal: $\mathrm{p}\left(u_{it}\right) = \frac{2\phi^{1/2}}{(2\pi)^{1/2}} \exp\left\{-\frac{\phi}{2} u_{it}^2\right\}$

When u_{it} enters the specification with a plus sign then the model represents a cost frontier, while when u_{it} enters with a minus sign the model represents a production frontier. For the efficiency scores generated by a stochastic frontier model to be meaningful, the dependent variable in both cases must be in logarithms.

No time dependence is imposed on the inefficiency component of the error term: each u_{it} is treated as an independent draw from $\mathrm{D}\left(\boldsymbol{\theta}\right)$.

Calculation of the log-marginal likelihood for the model is performed by integrating-out the u_{it}s from the complete-data likelihood by simulation. BayES multiplies the number of draws used for the estimation of the parameters by the maximum number of time observations per group ($\max_i \{T_i\}$) to determine the number of draws to be used for this integration. However, approximation of an integral of dimension T_i for each group, i, by simulation may be imprecise if T_i is large. Therefore, using a large number of iterations is equired to reduce the Monte Carlo standard error associated with the value of the log-marginal likelihood.

Priors

Parameter	Probability density function	Default hyperparameters
`Common to all models`		
$\bar{\gamma}$	$\mathrm{p}(\bar{\gamma}) = \frac{\lvert\mathbf{P}_\gamma\rvert^{1/2}}{(2\pi)^{K/2}} \exp\left\{-\frac{1}{2}(\bar{\gamma}-\mathbf{m}_\gamma)'\mathbf{P}_\gamma(\bar{\gamma}-\mathbf{m}_\gamma)\right\}$	$\mathbf{m}_\gamma = \mathbf{0}_K$, $\mathbf{P}_\gamma = 0.001\cdot\mathbf{I}_K$
$\boldsymbol{\Omega}$	$\mathrm{p}(\boldsymbol{\Omega}) = \frac{\lvert\boldsymbol{\Omega}\rvert^{\frac{n-K-1}{2}}\lvert\mathbf{V}^{-1}\rvert^{n/2}}{2^{nK/2}\Gamma_K\left(\frac{n}{2}\right)} \exp\left\{-\frac{1}{2}\mathrm{tr}\left(\mathbf{V}^{-1}\boldsymbol{\Omega}\right)\right\}$	$n = K^2$, $\mathbf{V} = \frac{100}{K}\cdot\mathbf{I}_K$
$\boldsymbol{\beta}$	$\mathrm{p}(\boldsymbol{\beta}) = \frac{\lvert\mathbf{P}_\beta\rvert^{1/2}}{(2\pi)^{L/2}} \exp\left\{-\frac{1}{2}(\boldsymbol{\beta}-\mathbf{m}_\beta)'\mathbf{P}_\beta(\boldsymbol{\beta}-\mathbf{m}_\beta)\right\}$	$\mathbf{m}_\beta = \mathbf{0}_L$, $\mathbf{P}_\beta = 0.001\cdot\mathbf{I}_L$
τ	$\mathrm{p}(\tau) = \frac{b_\tau^{a_\tau}}{\Gamma(a_\tau)}\tau^{a_\tau-1}e^{-\tau b_\tau}$	$a_\tau = 0.001$, $b_\tau = 0.001$
`Exponential model`		
λ	$\mathrm{p}(\lambda) = \frac{b_\lambda^{a_\lambda}}{\Gamma(a_\lambda)}\lambda^{a_\lambda-1}e^{-\lambda b_\lambda}$	$a_\lambda = 1$, $b_\lambda = 0.15$
`Half normal model`		
ϕ	$\mathrm{p}(\phi) = \frac{b_\phi^{a_\phi}}{\Gamma(a_\phi)}\phi^{a_\phi-1}e^{-\phi b_\phi}$	$a_\phi = 7$, $b_\phi = 0.5$

Syntax

```
[<model name> = ] sf_rc( y ~ z1 z2 ... zK [| x1 x2 ... xL ] [, <options> ] );
```

where:

- `y` is the dependent variable name, as it appears in the dataset used for estimation
- `z1 z2 ... zK` is a list of the names, as they appear in the dataset used for estimation, of the independent variables which are associated with group-specific coefficients; when a constant term is to be included in the set of group-specific coefficients this must be requested explicitly
- `x1 x2 ... xL` is is a list of the names, as they appear in the dataset used for estimation, of the independent variables which are associated with coefficients common to all groups; when a constant term is to be included in the set of common coefficients, this must be requested explicitly

An independent variable could be included in either the `x` or the `z` variable list, depending on whether the parameter associated with this variable is common to all groups or not. However, including a variable in both lists would lead to exact multicollinearity and, in this case, BayES will issue an error.

Before using the `sf_rc()` function the dataset used for estimation must be declared as a panel dataset using the `set_pd()` function (see section B.13). In the case of cross-sectional data, the dataset still needs to be declared as a panel, but the group-id variable could be constructed as a list of unique integers using, for example, the `range()` function.

For groups observed only once, a group-specific parameter associated with a constant term cannot be distinguished from the error term (v_{it}). Thus, a warning is produced when a constant term is included in the `z` list and the dataset contains at least one group which is observed only once.

The optional arguments for the random-coefficients stochastic frontier model are:[4]

[4] Optional arguments are always given in option-value pairs (eg. `"chains"=3`).

Gibbs parameters	
`"chains"`	number of chains to run in parallel (positive integer); the default value is 1
`"burnin"`	number of burn-in draws per chain (positive integer); the default value is 10000
`"draws"`	number of retained draws per chain (positive integer); the default value is 20000
`"thin"`	value of the thinning parameter (positive integer); the default value is 1
`"seed"`	value of the seed for the random-number generator (positive integer); the default value is 42
Model specification	
`"udist"`	specification of the distribution of the inefficiency component of the error term; the following options are available, corresponding to the distributions presented at the beginning of this section: • `"exp"` • `"hnorm"` the default value is `"exp"`
`"production"`	boolean specifying the type of frontier (production/cost); it could be set to either **true** (production) or **false** (cost); the default value is **true**
Hyperparameters	
Common to all models	
`"m_gamma"`	mean vector of the prior for $\bar{\gamma}$ ($K \times 1$ vector); the default value is $\mathbf{0}_K$
`"P_gamma"`	precision matrix of the prior for $\bar{\gamma}$ ($K \times K$ symmetric and positive-definite matrix); the default value is $0.001 \cdot \mathbf{I}_K$
`"V"`	scale matrix of the prior for Ω ($K \times K$ symmetric and positive-definite matrix); the default value is $\frac{100}{K} \cdot \mathbf{I}_K$
`"n"`	degrees-of-freedom parameter of the prior for $\mathbf{\Omega}$ (real number greater than or equal to K); the default value is K^2
`"m_beta"`	mean vector of the prior for $\boldsymbol{\beta}$ ($L \times 1$ vector); the default value is $\mathbf{0}_L$
`"P_beta"`	precision matrix of the prior for $\boldsymbol{\beta}$ ($L \times L$ symmetric and positive-definite matrix); the default value is $0.001 \cdot \mathbf{I}_L$
`"a_tau"`	shape parameter of the prior for τ (positive number); the default value is 0.001
`"b_tau"`	rate parameter of the prior for τ (positive number); the default value is 0.001
Exponential model	
`"a_lambda"`	shape parameter of the prior for λ (positive number); the default value is 1
`"b_lambda"`	rate parameter of the prior for λ (positive number); the default value is 0.15
Half normal model	
`"a_phi"`	shape parameter of the prior for ϕ (positive number); the default value is 7
`"b_phi"`	rate parameter of the prior for ϕ (positive number); the default value is 0.5
Dataset and log-marginal likelihood	
`"dataset"`	the id value of the dataset that will be used for estimation; the default value is the first dataset in memory (in alphabetical order)
`"logML_CJ"`	boolean indicating whether the Chib (1995)/Chib & Jeliazkov (2001) approximation to the log-marginal likelihood should be calculated (**true**|**false**); the default value is **false**

Reported Parameters

Common to all models		
$\bar{\gamma}$	`variable_name`	vector of parameters associated with the independent variables in the `z` list; these are the means of the group-specific parameters
$\boldsymbol{\beta}$	`variable_name`	vector of parameters associated with the independent variables in the `x` list

table continues on next page

table continued from previous page

τ	`tau`	precision parameter of the noise component of the error term, v_i
σ_v	`sigma_v`	standard deviation of the noise component of the error term, $\sigma_v = 1/\tau^{1/2}$
`Exponential model`		
λ	`lambda`	rate parameter of the distribution of the inefficiency component of the error term, u_i
σ_u	`sigma_u`	scale parameter of the inefficiency component of the error term: $\sigma_u = 1/\lambda$. For the exponential model the standard deviation of u_i is equal to the scale parameter.
`Half normal model`		
ϕ	`phi`	precision parameter of the distribution of the inefficiency component of the error term, u_i
σ_u	`sigma_u`	scale parameter of the inefficiency component of the error term: $\sigma_u = 1/\phi^{1/2}$. The standard deviation of u_i for the half-normal model can be obtained as $\sigma_u\sqrt{1-\frac{2}{\pi}}$.

Stored values and post-estimation analysis

If a left-hand-side id value is provided when a random-coefficients stochastic frontier model is created, then the following results are saved in the model item and are accessible via the '.' operator:

`Samples` a matrix containing the draws from the posterior of $\bar{\gamma}$, $\boldsymbol{\beta}$, τ, the unique elements of $\boldsymbol{\Omega}$ and either λ (exponential model) or ϕ (half-normal model)

`z1,...,zK` vectors containing the draws from the posterior of the mean of the group-specific coefficients ($\bar{\gamma}$s) associated with variables `z1,...,zK` (the names of these vectors are the names of the variables that were included in the right-hand side of the model)

`x1,...,xL` vectors containing the draws from the posterior of the parameters associated with variables `x1,...,xL` (the names of these vectors are the names of the variables that were included in the right-hand side of the model)

`tau` vector containing the draws from the posterior of τ

`lambda` vector containing the draws from the posterior of λ (available after the estimation of the exponential model)

`phi` vector containing the draws from the posterior of ϕ (available after the estimation of the half-normal model)

`Omega_i_j` vectors containing the draws from the posterior of the unique elements of $\boldsymbol{\Omega}$; because $\boldsymbol{\Omega}$ is symmetric, only $\frac{(K-1)K}{2}+K$ of its elements are stored (instead of all K^2 elements); `i` and `j` index the row and column of $\boldsymbol{\Omega}$, respectively, at which the corresponding element is located

`Omega` $K \times K$ matrix that stores the posterior mean of $\boldsymbol{\Omega}$

`logML` the Lewis & Raftery (1997) approximation of the log-marginal likelihood

`logML_CJ` the Chib (1995)/Chib & Jeliazkov (2001) approximation to the log-marginal likelihood; this is available only if the model was estimated with the `"logML_CJ"=true` option

`gamma_i` $N \times K$ matrix that stores the group-specific coefficients for the variables in the `z` list; the values in this matrix are not guaranteed to be in the same order as the order in which the groups appear in the dataset used for estimation; use the `store()` function to associate the values in `gamma_i` with the observations in the dataset

`eff_i`	$\left(\sum_{i=1}^{N} T_i\right) \times 1$ vector that stores the expected values of the observation-specific efficiency scores, $\mathrm{E}\left(e^{-u_{it}}\right)$; the values in this vector are not guaranteed to be in the same order as the order in which the observations appear in the dataset used for estimation; use the `store()` function to associate the values in `eff_i` with the observations in the dataset
`nchains`	the number of chains that were used to estimate the model
`nburnin`	the number of burn-in draws per chain that were used when estimating the model
`ndraws`	the total number of retained draws from the posterior (=`chains` · `draws`)
`nthin`	value of the thinning parameter that was used when estimating the model
`nseed`	value of the seed for the random-number generator that was used when estimating the model

Additionally, the following functions are available for post-estimation analysis (see section B.14):

- `diagnostics()`
- `pmp()`
- `test()`
- `store()`

The random-coefficients stochastic frontier model uses the `store()` function to associate the group-specific parameters (`gamma_i`) or the estimates of the efficiency scores (`eff_i`) with specific observations and store their values in the dataset used for estimation. The generic syntax for a statement involving the `store()` function after estimation of a random-coefficients stochastic frontier model and for each of these two quantities is:

```
store( gamma_i , <new variable name prefix> [, "model"=<model name>] );
```

and:

```
store( eff_i , <new variable name> [, "model"=<model name>] );
```

The first statement will generate K additional variables in the dataset used for estimation of the random-coefficients model, with names constructed by prepending the prefix provided as the second argument to `store()` to the names of the variables which are associated with group-specific coefficients. The second statement will generate one additional variable in the dataset used for estimation of the model and its name will be the one provided as the second argument to `store()`.

Examples

Example 1

```
myData = import("$BayESHOME/Datasets/dataset2.csv", ",");
myData.constant = ones(rows(myData), 1);
set_pd( year, id, "dataset" = myData);

// all rhs variables are associated with group-specific coefficients
sf_rc( y ~ constant x1 x2 x3);
```

Example 2

```
myData = import("$BayESHOME/Datasets/dataset2.csv", ",");
myData.constant = ones(rows(myData), 1);
set_pd( year, id, "dataset" = myData);

// only the constant term and the coefficient associated with x1 are
// group-specific; the coefficients on x2 and x3 are common to all groups
sf_rc( y ~ constant x1 | x2 x3);
```

Example 3

```
myData = import("$BayESHOME/Datasets/dataset2.csv", ",");
myData.constant = ones(rows(myData), 1);
set_pd( year, id, "dataset" = myData);

// all rhs variables are associated with group-specific coefficients
model1 = sf_rc(y ~ constant x1 x2 x3, "udist"="hnorm",
    "burnin"=10000, "draws"=40000, "thin"=4, "chains"=2,
    "logML_CJ" = true, "dataset"=myData);
store( gamma_i, rc1_, "model" = model1 );

// only the constant term and the coefficient associated with x1 are
// group-specific; the coefficients on x2 and x3 are common to all groups
model2 = lm_rc( y ~ constant x1 | x2 x3, "udist"="hnorm",
    "burnin"=10000, "draws"=40000, "thin"=4, "chains"=2,
    "logML_CJ" = true, "dataset"=myData);
store( gamma_i, rc2_, "model" = model2 );

pmp( { model1, model2 } );
```

5.5 Latent-class stochastic frontier

Mathematical representation

$$y_{i|c} = \mathbf{x}_i'\boldsymbol{\beta}_c + v_{i|c} \pm u_{i|c}, \qquad v_{i|c} \sim \mathrm{N}\left(0, \tfrac{1}{\tau_c}\right), \quad u_{i|c} \sim \mathrm{D}\left(\boldsymbol{\theta}\right), \qquad c = 0, 1, \ldots, C-1 \tag{5.5}$$

- the model is estimated using N observations and involves C classes (counting starts at zero)
- y_i is the value of the dependent variable for observation i
- $\mathbf{x}_i$ is a $K \times 1$ vector that stores the values of the K independent variables for observation i
- $\boldsymbol{\beta}_c$ is a $K \times 1$ vector of parameters for class c
- τ_c is the precision of the noise component of the error term for class c: $\sigma^2_{v,c} = \frac{1}{\tau_c}$
- u_{it} is the inefficiency component of the error term for group i in period t and it can have any non-negative distribution, represented in the equation above by $\mathrm{D}\left(\boldsymbol{\theta}\right)$; BayES supports the following distributions for u_{it}:
  - exponential: $\mathrm{p}\left(u_{i|c}\right) = \lambda_c e^{-\lambda_c u_{i|c}}$
  - half normal: $\mathrm{p}\left(u_{i|c}\right) = \frac{2\phi_c^{1/2}}{(2\pi)^{1/2}} \exp\left\{-\frac{\phi_c}{2} u^2_{i|c}\right\}$
- each observation, i, belongs to class c with prior probability (before seeing the data $\{y_i, \mathbf{x}_i\}$) $\pi_{i,c}$. BayES supports two types of models:
  1. unconditional prior class membership probabilities, in which case:

     $$\pi_{i,c} = \pi_c \quad \forall i, c$$

     With this specification $\boldsymbol{\pi} \equiv \begin{bmatrix}\pi_0 & \pi_1 & \cdots & \pi_{C-1}\end{bmatrix}'$ is a vector of parameters to be estimated, with $\pi_c > 0\ \forall c$ and $\sum_{c=0}^{C-1} \pi_c = 1$.
  2. conditional prior class membership probabilities, in which case:

     $$\pi_{i,c} = \frac{e^{\mathbf{z}_i'\boldsymbol{\delta}_c}}{\sum_{\ell=0}^{C-1} e^{\mathbf{z}_i'\boldsymbol{\delta}_\ell}} \quad \forall i, c$$

     where:
     - $\mathbf{z}_i$ is an $L \times 1$ vector that stores the values of the L determinants of class-membership for observation i
     - $\boldsymbol{\delta} \equiv \begin{bmatrix}\boldsymbol{\delta}_1' & \boldsymbol{\delta}_2' & \cdots \boldsymbol{\delta}_{C-1}'\end{bmatrix}'$ is an $L \cdot (C-1) \times 1$ vector of parameters to be estimated

     In this specification class-membership probabilities are determined by a multinomial Logit model, where, for identification purposes, $\boldsymbol{\delta}_0$ is normalized to an $L \times 1$ vector of zeros.

When $u_{i|c}$ enters the specification with a plus sign then the model represents a cost frontier, while when $u_{i|c}$ enters with a minus sign the model represents a production frontier. For the efficiency scores generated by a stochastic frontier model to be meaningful, the dependent variable in both cases must be in logarithms.

Priors

Parameter	Probability density function	Default hyperparameters
`Common to all model types`		
$\boldsymbol{\beta}_c$	$\mathrm{p}(\boldsymbol{\beta}_c) = \frac{\lvert\mathbf{P}_c\rvert^{1/2}}{(2\pi)^{K/2}} \exp\left\{-\frac{1}{2}(\boldsymbol{\beta}_c - \mathbf{m}_c)' \mathbf{P}_c (\boldsymbol{\beta}_c - \mathbf{m}_c)\right\}$	$\mathbf{m}_c = \mathbf{0}_K$, $\mathbf{P}_c = 0.001 \cdot \mathbf{I}_K$
τ_c	$\mathrm{p}(\tau_c) = \frac{b_{\tau_c}^{a_{\tau_c}}}{\Gamma(a_{\tau_c})} \tau_c^{a_{\tau_c}-1} e^{-\tau_c b_{\tau_c}}$	$a_{\tau_c} = 0.001$, $b_{\tau_c} = 0.001$
`Exponential model`		
λ_c	$\mathrm{p}(\lambda_c) = \frac{b_{\lambda_c}^{a_{\lambda_c}}}{\Gamma(a_{\lambda_c})} \lambda_c^{a_{\lambda_c}-1} e^{-\lambda_c b_{\lambda_c}}$	$a_{\lambda_c} = 1$, $b_{\lambda_c} = 0.15$
`Half normal model`		
ϕ_c	$\mathrm{p}(\phi_c) = \frac{b_{\phi_c}^{a_{\phi_c}}}{\Gamma(a_{\phi_c})} \phi_c^{a_{\phi_c}-1} e^{-\phi_c b_{\phi_c}}$	$a_{\phi_c} = 7$, $b_{\phi_c} = 0.5$
`Model with unconditional class-membership probabilities`		
$\boldsymbol{\pi}$	$\mathrm{p}(\boldsymbol{\pi}) = \frac{1}{B(\mathbf{a})} \prod_{c=0}^{C-1} \pi_c^{a_c-1}$	$a_0 = a_1 = \cdots = a_{C-1} = 1$
`Model with conditional class-membership probabilities`		
$\boldsymbol{\delta}$	$\mathrm{p}(\boldsymbol{\delta}) = \frac{\lvert\mathbf{P}_\delta\rvert^{1/2}}{(2\pi)^{\frac{L(C-1)}{2}}} \exp\left\{-\frac{1}{2}(\boldsymbol{\delta} - \mathbf{m}_\delta)' \mathbf{P}_\delta (\boldsymbol{\delta} - \mathbf{m}_\delta)\right\}$	$\mathbf{m}_\delta = \mathbf{0}_{L(C-1)}$, $\mathbf{P}_\delta = 0.001 \cdot \mathbf{I}_{L(C-1)}$

Syntax

```
[<model name> = ] sf_lc( y ~ x1 x2 ... xK [ | z1 z2 ... zL] [, <options> ] );
```

where:

- `y` is the dependent variable name, as it appears in the dataset used for estimation
- `x1 x2 ... xK` is a list of the K independent variable names, as they appear in the dataset used for estimation; when a constant term is to be included in the model, this must be requested explicitly
- `z1 z2 ... zL` is a list of the L variable names that enter the specification of the class-membership probabilities (determinants of class membership), as they appear in the dataset used for estimation; when a constant term is to be included in the model, this must be requested explicitly; this list is optional and when provided the conditional latent-class model is estimated; if not provided the unconditional model is estimated

If the dataset used for estimation has been previously declared as a panel dataset (typically, by a call to the `set_pd()` function) then the model estimated is the one documented in the following section. Each group in that model is restricted to belong to the same class for the entire period for which it is observed.

The optional arguments for the latent-class stochastic frontier model are:[5]

`Gibbs parameters`	
`"chains"`	number of chains to run in parallel (positive integer); the default value is 1
`"burnin"`	number of burn-in draws per chain (positive integer); the default value is 10000
`"draws"`	number of retained draws per chain (positive integer); the default value is 20000
`"thin"`	value of the thinning parameter (positive integer); the default value is 1
`"seed"`	value of the seed for the random-number generator (positive integer); the default value is 42

[5] Optional arguments are always given in option-value pairs (eg. `"chains"=3`).

Model specification	
`"udist"`	specification of the distribution of the inefficiency component of the error term; the following options are available, corresponding to the distributions presented at the beginning of this section: • `"exp"` • `"hnorm"` the default value is `"exp"`
`"production"`	boolean specifying the type of frontier (production/cost); it could be set to either **true** (production) or **false** (cost); the default value is **true**
`"classes"`	specification of the number of classes to be used in the model (positive integer); the default value is 2
Hyperparameters	
Common to all model types	
`"m"`	mean vector of the prior for each $\boldsymbol{\beta}_c$ ($K \times 1$ vector); the default value is $\mathbf{0}_K$
`"P"`	precision matrix of the prior for each $\boldsymbol{\beta}_c$ ($K \times K$ symmetric and positive-definite matrix); the default value is $0.001 \cdot \mathbf{I}_K$
`"mj"`	mean vector of the prior for $\boldsymbol{\beta}_j$, $j = 0, 1, \ldots, C-1$ ($K \times 1$ vector); this mean overwrites the generic mean (`"m"`) for class j only
`"Pj"`	precision matrix of the prior for $\boldsymbol{\beta}_j$, $j = 0, 1, \ldots, C-1$ ($K \times K$ symmetric and positive-definite matrix); this precision matrix overwrites the generic precision matrix (`"P"`) for class j only
`"a_tau"`	shape parameter of the prior for each τ_c (positive number); the default value is 0.001
`"b_tau"`	rate parameter of the prior for each τ_c (positive number); the default value is 0.001
`"a_tauj"`	shape parameter of the prior for τ_j, $j = 0, 1, \ldots, C-1$ (positive number); this shape parameter overwrites the generic shape parameter (`"a_tau"`) for class j only
`"b_tauj"`	rate parameter of the prior for τ_j, $j = 0, 1, \ldots, C-1$ (positive number); this rate parameter overwrites the generic rate parameter (`"b_tau"`) for class j only
Exponential model	
`"a_lambda"`	shape parameter of the prior for each λ_c (positive number); the default value is 1
`"b_lambda"`	rate parameter of the prior for each λ_c (positive number); the default value is 0.15
`"a_lambdaj"`	shape parameter of the prior for λ_j, $j = 0, 1, \ldots, C-1$ (positive number); this shape parameter overwrites the generic shape parameter (`"a_lambda"`) for class j only
`"b_lambdaj"`	rate parameter of the prior for λ_j, $j = 0, 1, \ldots, C-1$ (positive number); this rate parameter overwrites the generic rate parameter (`"b_lambda"`) for class j only
Half normal model	
`"a_phi"`	shape parameter of the prior for each ϕ_c (positive number); the default value is 7
`"b_phi"`	rate parameter of the prior for each ϕ_c (positive number); the default value is 0.5
`"a_phij"`	shape parameter of the prior for ϕ_j, $j = 0, 1, \ldots, C-1$ (positive number); this shape parameter overwrites the generic shape parameter (`"a_phi"`) for class j only
`"b_phij"`	rate parameter of the prior for ϕ_j, $j = 0, 1, \ldots, C-1$ (positive number); this rate parameter overwrites the generic rate parameter (`"b_phi"`) for class j only

`Model with unconditional class-membership probabilities`	
`"a"`	vector of concentration parameters for the Dirichlet prior on $\boldsymbol{\pi}$ ($C\times 1$ vector with positive entries); the default value is a $C\times 1$ vector of ones
`Model with conditional class-membership probabilities`	
`"m_delta"`	mean vector of the prior for $\boldsymbol{\delta}$ ($L\,(C-1)\times 1$ vector); the default value is $\mathbf{0}_{L(C-1)}$
`"P_delta"`	precision matrix of the prior for $\boldsymbol{\delta}$ ($L\,(C-1)\times 1$ symmetric and positive-definite matrix); the default value is $0.001\cdot\mathbf{I}_{L(C-1)}$
`Dataset and log-marginal likelihood`	
`"dataset"`	the id value of the dataset that will be used for estimation; the default value is the first dataset in memory (in alphabetical order)
`"logML_CJ"`	boolean indicating whether the Chib (1995)/Chib & Jeliazkov (2001) approximation to the log-marginal likelihood should be calculated (**true**|**false**); the default value is **false**

Reported Parameters

`Common to all model types`		
$\boldsymbol{\beta}_c$	`variable_name`	vector of parameters associated with the independent variables for class c
τ_c	`tau`	precision parameter of the noise component of the error term for class c, $v_{i\|c}$
$\sigma_{v,c}$	`sigma_v`	standard deviation of the noise component of the error term for class c: $\sigma_{v,c}=1/\tau_c^{1/2}$
`Exponential model`		
λ_c	`lambda`	rate parameter of the distribution of the inefficiency component of the error term for class c: $u_{i\|c}$
$\sigma_{u,c}$	`sigma_u`	scale parameter of the inefficiency component of the error term for class c: $\sigma_{u,c}=1/\lambda_c$. For the exponential model the standard deviation of $u_{i\|c}$ is equal to the scale parameter.
`Half normal model`		
ϕ_c	`phi`	precision parameter of the distribution of the inefficiency component of the error term for class c, $u_{i\|c}$
$\sigma_{u,c}$	`sigma_u`	scale parameter of the inefficiency component of the error term for class c: $\sigma_{u,c}=1/\phi_c^{1/2}$. The standard deviation of $u_{i\|c}$ for the half-normal model can be obtained as $\sigma_{u,c}\sqrt{1-\frac{2}{\pi}}$.
`Model with unconditional class-membership probabilities`		
π_c	`pi`	prior class-membership probability for class c
`Model with conditional class-membership probabilities`		
$\boldsymbol{\delta}_c$	`variable_name`	vector of parameters associated with the determinants of class membership for class c; for identification purposes, these parameters for class 0 are normalized to zero

Stored values and post-estimation analysis

If a left-hand-side id value is provided when a latent-class stochastic frontier model is created, then the following results are saved in the model item and are accessible via the '.' operator:

`Samples`	a matrix containing the draws from the posterior of $\boldsymbol{\beta}_c$ and τ_c for $c=0,1,\ldots,C-1$, and, depending on the estimated model, either λ (exponential model) or ϕ (half-normal model) and either $\boldsymbol{\pi}$ or $\boldsymbol{\delta}$.

`cj$x1,...,cj$xK`	vectors containing the draws from the posterior of the parameters associated with variables `x1,...,xK`, for $j = 0, 1, \ldots, C-1$ (the names of these vectors are the names of the variables that were included in the right-hand side of the model, prepended by '`c`', the class index and a dollar sign; in this way '`cj`' can be used to distinguish among parameters across different classes)
`cj$tau`	vectors containing the draws from the posterior of each τ_c, for $j = 0, 1, \ldots, C-1$ ('`tau`' is prepended by '`c`', the class index and the dollar sign; in this way '`cj`' can be used to distinguish among precision parameters in different classes)
`cj$lambda`	vector containing the draws from the posterior of λ_c, for $j = 0, 1, \ldots, C-1$ ('`lambda`' is prepended by '`c`', the class index and the dollar sign; in this way '`cj`' can be used to distinguish among precision parameters in different classes; available after the estimation of the exponential model)
`cj$phi`	vector containing the draws from the posterior of ϕ_c, for $j = 0, 1, \ldots, C-1$ ('`phi`' is prepended by '`c`', the class index and the dollar sign; in this way '`cj`' can be used to distinguish among precision parameters in different classes; available after the estimation of the half-normal model)
`pi_j`	vectors containing the draws from the posterior of each π_c, for $j = 0, 1, \ldots, C-1$ (these vectors are available only after the estimation of the model with unconditional class-membership probabilities)
`pi_j$z1,...,` `pi_j$zL`	vectors containing the draws from the posterior of the parameters associated with variables `z1,...,zL`, for $j = 1, \ldots, C-1$[6] (the names of these vectors are the names of the variables that were included the `z` list, in the right-hand side of the model, prepended by '`pi_j`' and the dollar sign; in this way '`pi_j`' can be used to distinguish among parameters associated with variables with different roles in the model, for example the same variable appearing in both `x` and `z` lists, as well as among parameters associated with a variable in the `z` list, but corresponding to different classes; these vectors are available only after the estimation of the model with conditional class-membership probabilities)
`eff_i`	$N \times 1$ vector that stores the expected values of the observation-specific efficiency scores, $\mathrm{E}(e^{-u_i})$; the values in this vector are not guaranteed to be in the same order as the order in which the observations appear in the dataset used for estimation; use the `store()` function to associate the values in `eff_i` with the observations in the dataset
`pi_i`	$N \times C$ matrix that stores the expected values of the posterior class-membership probabilities for each observation and for each of the C classes
`logML`	the Lewis & Raftery (1997) approximation of the log-marginal likelihood
`logML_CJ`	the Chib (1995)/Chib & Jeliazkov (2001) approximation to the log-marginal likelihood; this is available only if the model was estimated with the `"logML_CJ"=true` option
`nchains`	the number of chains that were used to estimate the model
`nburnin`	the number of burn-in draws per chain that were used when estimating the model
`ndraws`	the total number of retained draws from the posterior (=`chains` · `draws`)
`nthin`	value of the thinning parameter that was used when estimating the model
`nseed`	value of the seed for the random-number generator that was used when estimating the model
`nclasses`	number of classes used during the estimation of the model

[6] Indexing starts at 1 because the parameters of the multinomial-Logit part of the model associated with class 0 are normalized to zero for identification purposes.

Additionally, the following functions are available for post-estimation analysis (see section B.14):

- diagnostics()
- test()
- pmp()
- store()
- mfx()

The latent-class stochastic frontier model uses the store() function to associate the estimates of the efficiency scores (eff_i) or the estimates of the posterior class-membership probabilities (pi_i) with specific observations and store their values in the dataset used for estimation. The generic syntax for a statement involving the store() function after estimation of a latent-class stochastic frontier model and for each of these two quantities is:

```
store( eff_i , <new variable name> [, "model"=<model name>] );
```

and:

```
store( pi_i , <new variable name prefix> [, "model"=<model name>] );
```

The first statement will generate one additional variable in the dataset used for estimation of the model and its name will be the one provided as the second argument to store(). This second statement will generate C additional variables in the dataset used for estimation of the model, with names constructed by appending the class index $(0, 1, \ldots, C-1)$ to the prefix provided as the second argument to store().

The latent-class stochastic frontier model with conditional class-membership probabilities uses the mfx() function to calculate and report the marginal effects of the variables in the z list on the prior class-membership probabilities that come from the multinomial-Logit part of the model. The generic syntax for a statement involving the mfx() function after estimation of a latent-class stochastic frontier model with conditional class-membership probabilities is:

```
mfx( ["type"=1] [, "point"=<point of calculation>] [, "model"=<model name>] );
```

See the general documentation of the mfx() function (section B.14) for details on the optional arguments.

Examples

Example 1

```
myData = import("$BayESHOME/Datasets/dataset2.csv");
myData.constant = ones(rows(myData), 1);

sf_lc(y ~ constant x1 x2);
```

Example 2

```
myData = import("$BayESHOME/Datasets/dataset2.csv");
myData.constant = ones(rows(myData), 1);

myModel = sf_lc(y ~ constant x1 x2, "udist"="hnorm",
    "m0"=[2;0.6;0.3], "P" = 10*eye(3,3),
    "burnin"=10000, "draws"=30000, "thin"=2, "chains"=2, "classes"=2,
    "logML_CJ" = true, "dataset"=myData);

diagnostics("model"=myModel);

plot([myModel.c0$x1, myModel.c1$x1],
    "title"="\beta2 for the two classes");
plot([myModel.pi_0, myModel.pi_1],
    "title"="Prior class-membership probabilities");
```

Example 3

```
myData = import("$BayESHOME/Datasets/dataset2.csv");
myData.constant = ones(rows(myData), 1);

myModel = sf_lc(y ~ constant x1 x2 | constant x3, "udist"="hnorm",
    "m0"=[2;0.6;0.3], "P" = 10*eye(3,3),
    "burnin"=10000, "draws"=30000, "thin"=2, "chains"=2, "classes"=2,
    "logML_CJ" = true, "dataset"=myData);

diagnostics("model"=myModel);
mfx("model"=myModel);

plot([myModel.c0$x1, myModel.c1$x1],
    "title"="\beta2 for the two classes");
plot(myModel.pi_1$x3,
    "title"="\delta2 for class 1");
```

5.6 Latent-class stochastic frontier model with panel data

Mathematical representation

$$y_{it|c} = \mathbf{x}'_{it}\boldsymbol{\beta}_c + v_{it|c} \pm u_{it|c}, \qquad v_{it|c} \sim \mathrm{N}\left(0, \tfrac{1}{\tau_c}\right), \quad u_{it|c} \sim \mathrm{D}\left(\boldsymbol{\theta}\right), \qquad c = 0, 1, \ldots, C-1 \quad (5.6)$$

- the model is estimated using observations from N groups, each group observed for T_i periods (balanced or unbalanced panels); the total number of observations is $\sum_{i=1}^{N} T_i$
- the model involves C classes (counting starts at zero) and each group, i, is restricted to belong to the same class for all T_i observations
- y_{it} is the value of the dependent variable for group i, observed in period t
- $\mathbf{x}_{it}$ is a $K \times 1$ vector that stores the values of the K independent variables for group i, observed in period t
- $\boldsymbol{\beta}_c$ is a $K \times 1$ vector of parameters for class c
- τ_c is the precision of the noise component of the error term for class c: $\sigma^2_{v,c} = \frac{1}{\tau_c}$
- u_{it} is the inefficiency component of the error term for group i in period t and it can have any non-negative distribution, represented in the equation above by $\mathrm{D}\left(\boldsymbol{\theta}\right)$; BayES supports the following distributions for u_{it}:
  - exponential: $\mathrm{p}\left(u_{it|c}\right) = \lambda_c e^{-\lambda u_{it}|c}$
  - half normal: $\mathrm{p}\left(u_{it|c}\right) = \frac{2\phi_c^{1/2}}{(2\pi)^{1/2}} \exp\left\{-\frac{\phi_c}{2} u^2_{it|c}\right\}$
- each group, i, belongs to class c with prior probability (before seeing the data $\{y_{it}, \mathbf{x}_{it}\}_{t=1}^{T_i}$) $\pi_{i,c}$. BayES supports two types of models:
  1. unconditional prior class membership probabilities, in which case:

     $$\pi_{i,c} = \pi_c \quad \forall i, c$$

     With this specification $\boldsymbol{\pi} \equiv \begin{bmatrix}\pi_0 & \pi_1 & \cdots & \pi_{C-1}\end{bmatrix}'$ is a vector of parameters to be estimated, with $\pi_c > 0 \ \forall c$ and $\sum_{c=0}^{C-1} \pi_c = 1$.
  2. conditional prior class membership probabilities, in which case:

     $$\pi_{i,c} = \frac{e^{\mathbf{z}'_i \boldsymbol{\delta}_c}}{\sum_{\ell=0}^{C-1} e^{\mathbf{z}'_i \boldsymbol{\delta}_\ell}} \quad \forall i, c$$

     where:
     - $\mathbf{z}_i$ is an $L \times 1$ vector that stores the values of the L determinants of class-membership for group i; these variables vary by group only (not within group)
     - $\boldsymbol{\delta} \equiv \begin{bmatrix}\boldsymbol{\delta}'_1 & \boldsymbol{\delta}'_2 & \cdots \boldsymbol{\delta}'_{C-1}\end{bmatrix}'$ is an $L \cdot (C-1) \times 1$ vector of parameters to be estimated

     In this specification class-membership probabilities are determined by a multinomial Logit model, where, for identification purposes, $\boldsymbol{\delta}_0$ is normalized to an $L \times 1$ vector of zeros.

When u_{it} enters the specification with a plus sign then the model represents a cost frontier, while when u_{it} enters with a minus sign the model represents a production frontier. For the efficiency scores generated by a stochastic frontier model to be meaningful, the dependent variable in both cases must be in logarithms.

No time dependence is imposed on the inefficiency component of the error term: each u_{it} is treated as an independent draw from $\mathrm{D}(\boldsymbol{\theta})$. This specification is known as the "true random effects" stochastic frontier model (Greene, 2004) .

Priors

Parameter	Probability density function	Default hyperparameters
`Common to all model types`		
$\boldsymbol{\beta}_c$	$\mathrm{p}(\boldsymbol{\beta}_c) = \frac{\lvert\mathbf{P}_c\rvert^{1/2}}{(2\pi)^{K/2}} \exp\left\{-\frac{1}{2}(\boldsymbol{\beta}_c - \mathbf{m}_c)' \mathbf{P}_c (\boldsymbol{\beta}_c - \mathbf{m}_c)\right\}$	$\mathbf{m}_c = \mathbf{0}_K$, $\mathbf{P}_c = 0.001 \cdot \mathbf{I}_K$
τ_c	$\mathrm{p}(\tau_c) = \frac{b_{\tau_c}^{a_{\tau_c}}}{\Gamma(a_{\tau_c})} \tau_c^{a_{\tau_c}-1} e^{-\tau_c b_{\tau_c}}$	$a_{\tau_c} = 0.001$, $b_{\tau_c} = 0.001$
`Exponential model`		
λ_c	$\mathrm{p}(\lambda_c) = \frac{b_{\lambda_c}^{a_{\lambda_c}}}{\Gamma(a_{\lambda_c})} \lambda_c^{a_{\lambda_c}-1} e^{-\lambda_c b_{\lambda_c}}$	$a_{\lambda_c} = 1$, $b_{\lambda_c} = 0.15$
`Half normal model`		
ϕ_c	$\mathrm{p}(\phi_c) = \frac{b_{\phi_c}^{a_{\phi_c}}}{\Gamma(a_{\phi_c})} \phi_c^{a_{\phi_c}-1} e^{-\phi_c b_{\phi_c}}$	$a_{\phi_c} = 7$, $b_{\phi_c} = 0.5$
`Model with unconditional class-membership probabilities`		
$\boldsymbol{\pi}$	$\mathrm{p}(\boldsymbol{\pi}) = \frac{1}{B(\mathbf{a})} \prod_{c=0}^{C-1} \pi_c^{a_c-1}$	$a_0 = a_1 = \cdots = a_{C-1} = 1$
`Model with conditional class-membership probabilities`		
$\boldsymbol{\delta}$	$\mathrm{p}(\boldsymbol{\delta}) = \frac{\lvert\mathbf{P}_\delta\rvert^{1/2}}{(2\pi)^{\frac{L(C-1)}{2}}} \exp\left\{-\frac{1}{2}(\boldsymbol{\delta} - \mathbf{m}_\delta)' \mathbf{P}_\delta (\boldsymbol{\delta} - \mathbf{m}_\delta)\right\}$	$\mathbf{m}_\delta = \mathbf{0}_{L(C-1)}$, $\mathbf{P}_\delta = 0.001 \cdot \mathbf{I}_{L(C-1)}$

Syntax

```
[<model name> = ] sf_lc( y ~ x1 x2 ... xK [ | z1 z2 ... zL] [, <options> ] );
```

where:

- `y` is the dependent variable name, as it appears in the dataset used for estimation
- `x1 x2 ... xK` is a list of the K independent variable names, as they appear in the dataset used for estimation; when a constant term is to be included in the model, this must be requested explicitly
- `z1 z2 ... zL` is a list of the L variable names that enter the specification of the class-membership probabilities (determinants of class membership), as they appear in the dataset used for estimation; when a constant term is to be included in the model, this must be requested explicitly; this list is optional and when provided the conditional latent-class model is estimated; if not provided the unconditional model is estimated; the values of the variables in this list must be constant within each group

If the dataset used for estimation has not been previously declared as a panel dataset or if this structure has been removed (by a call to the `set_cs()` function) then the model estimated is the one documented in the preceding section. Different time observations from the same group in that model are allowed to belong to different classes.

The optional arguments for the latent-class stochastic frontier model with panel data are:[7]

[7] Optional arguments are always given in option-value pairs (eg. `"chains"=3`).

Gibbs parameters	
`"chains"`	number of chains to run in parallel (positive integer); the default value is 1
`"burnin"`	number of burn-in draws per chain (positive integer); the default value is 10000
`"draws"`	number of retained draws per chain (positive integer); the default value is 20000
`"thin"`	value of the thinning parameter (positive integer); the default value is 1
`"seed"`	value of the seed for the random-number generator (positive integer); the default value is 42
Model specification	
`"udist"`	specification of the distribution of the inefficiency component of the error term; the following options are available, corresponding to the distributions presented at the beginning of this section: • `"exp"` • `"hnorm"` the default value is `"exp"`
`"production"`	boolean specifying the type of frontier (production/cost); it could be set to either **true** (production) or **false** (cost); the default value is **true**
`"classes"`	specification of the number of classes to be used in the model (positive integer); the default value is 2
Hyperparameters	
Common to all model types	
`"m"`	mean vector of the prior for each $\boldsymbol{\beta}_c$ ($K\times 1$ vector); the default value is $\mathbf{0}_K$
`"P"`	precision matrix of the prior for each $\boldsymbol{\beta}_c$ ($K\times K$ symmetric and positive-definite matrix); the default value is $0.001\cdot\mathbf{I}_K$
`"mj"`	mean vector of the prior for $\boldsymbol{\beta}_j$, $j=0,1,\ldots,C-1$ ($K\times 1$ vector); this mean overwrites the generic mean (`"m"`) for class j only
`"Pj"`	precision matrix of the prior for $\boldsymbol{\beta}_j$, $j=0,1,\ldots,C-1$ ($K\times K$ symmetric and positive-definite matrix); this precision matrix overwrites the generic precision matrix (`"P"`) for class j only
`"a_tau"`	shape parameter of the prior for each τ_c (positive number); the default value is 0.001
`"b_tau"`	rate parameter of the prior for each τ_c (positive number); the default value is 0.001
`"a_tauj"`	shape parameter of the prior for τ_j, $j=0,1,\ldots,C-1$ (positive number); this shape parameter overwrites the generic shape parameter (`"a_tau"`) for class j only
`"b_tauj"`	rate parameter of the prior for τ_j, $j=0,1,\ldots,C-1$ (positive number); this rate parameter overwrites the generic rate parameter (`"b_tau"`) for class j only
Exponential model	
`"a_lambda"`	shape parameter of the prior for each λ_c (positive number); the default value is 1
`"b_lambda"`	rate parameter of the prior for each λ_c (positive number); the default value is 0.15
`"a_lambdaj"`	shape parameter of the prior for λ_j, $j=0,1,\ldots,C-1$ (positive number); this shape parameter overwrites the generic shape parameter (`"a_lambda"`) for class j only
`"b_lambdaj"`	rate parameter of the prior for λ_j, $j=0,1,\ldots,C-1$ (positive number); this rate parameter overwrites the generic rate parameter (`"b_lambda"`) for class j only
Half normal model	
`"a_phi"`	shape parameter of the prior for each ϕ_c (positive number); the default value is 7

`"b_phi"`	rate parameter of the prior for each ϕ_c (positive number); the default value is 0.5
`"a_phij"`	shape parameter of the prior for ϕ_j, $j = 0, 1, \ldots, C-1$ (positive number); this shape parameter overwrites the generic shape parameter (`"a_phi"`) for class j only
`"b_phij"`	rate parameter of the prior for ϕ_j, $j = 0, 1, \ldots, C-1$ (positive number); this rate parameter overwrites the generic rate parameter (`"b_phi"`) for class j only
`Model with unconditional class-membership probabilities`	
`"a"`	vector of concentration parameters for the Dirichlet prior on $\boldsymbol{\pi}$ ($C \times 1$ vector with positive entries); the default value is a $C \times 1$ vector of ones
`Model with conditional class-membership probabilities`	
`"m_delta"`	mean vector of the prior for $\boldsymbol{\delta}$ ($L\,(C-1) \times 1$ vector); the default value is $\mathbf{0}_{L(C-1)}$
`"P_delta"`	precision matrix of the prior for $\boldsymbol{\delta}$ ($L\,(C-1) \times 1$ symmetric and positive-definite matrix); the default value is $0.001 \cdot \mathbf{I}_{L(C-1)}$
`Dataset and log-marginal likelihood`	
`"dataset"`	the id value of the dataset that will be used for estimation; the default value is the first dataset in memory (in alphabetical order)
`"logML_CJ"`	boolean indicating whether the Chib (1995)/Chib & Jeliazkov (2001) approximation to the log-marginal likelihood should be calculated (**`true`**|**`false`**); the default value is **`false`**

Reported Parameters

`Common to all model types`		
$\boldsymbol{\beta}_c$	`variable_name`	vector of parameters associated with the independent variables for class c
τ_c	`tau`	precision parameter of the noise component of the error term for class c, $v_{it\|c}$
$\sigma_{v,c}$	`sigma_v`	standard deviation of the noise component of the error term for class c: $\sigma_{v,c} = 1/\tau_c^{1/2}$
`Exponential model`		
λ_c	`lambda`	rate parameter of the distribution of the inefficiency component of the error term for class c, $u_{it\|c}$
$\sigma_{u,c}$	`sigma_u`	scale parameter of the inefficiency component of the error term for class c: $\sigma_{u,c} = 1/\lambda_c$. For the exponential model the standard deviation of $u_{it\|c}$ is equal to the scale parameter.
`Half normal model`		
ϕ_c	`phi`	precision parameter of the distribution of the inefficiency component of the error term for class c, $u_{it\|c}$
$\sigma_{u,c}$	`sigma_u`	scale parameter of the inefficiency component of the error term for class c: $\sigma_{u,c} = 1/\phi_c^{1/2}$. The standard deviation of $u_{it\|c}$ for the half-normal model can be obtained as $\sigma_{u,c}\sqrt{1 - \frac{2}{\pi}}$.
`Model with unconditional class-membership probabilities`		
π_c	`pi`	prior class-membership probability for class c
`Model with conditional class-membership probabilities`		
$\boldsymbol{\delta}_c$	`variable_name`	vector of parameters associated with the determinants of class membership for class c; for identification purposes, these parameters for class 0 are normalized to zero

Stored values and post-estimation analysis

If a left-hand-side id value is provided when a latent-class stochastic frontier model with panel data is created, then the following results are saved in the model item and are accessible via the '.' operator:

`Samples`	a matrix containing the draws from the posterior of $\boldsymbol{\beta}_c$ and τ_c for $c = 0, 1, \ldots, C-1$, and, depending on the estimated model, either λ (exponential model) or ϕ (half-normal model) and either $\boldsymbol{\pi}$ or $\boldsymbol{\delta}$.
`cj$x1,...,cj$xK`	vectors containing the draws from the posterior of the parameters associated with variables `x1,...,xK`, for $j = 0, 1, \ldots, C-1$ (the names of these vectors are the names of the variables that were included in the right-hand side of the model, prepended by '`c`', the class index and a dollar sign; in this way '`cj`' can be used to distinguish among parameters across different classes)
`cj$tau`	vectors containing the draws from the posterior of each τ_c, for $j = 0, 1, \ldots, C-1$ ('`tau`' is prepended by '`c`', the class index and the dollar sign; in this way '`cj`' can be used to distinguish among precision parameters in different classes)
`cj$tau`	vectors containing the draws from the posterior of each τ_c, for $j = 0, 1, \ldots, C-1$ ('`tau`' is prepended by '`c`', the class index and the dollar sign; in this way '`cj`' can be used to distinguish among precision parameters in different classes)
`cj$lambda`	vector containing the draws from the posterior of λ_c, for $j = 0, 1, \ldots, C-1$ ('`lambda`' is prepended by '`c`', the class index and the dollar sign; in this way '`cj`' can be used to distinguish among precision parameters in different classes; available after the estimation of the exponential model)
`cj$phi`	vector containing the draws from the posterior of ϕ_c, for $j = 0, 1, \ldots, C-1$ ('`phi`' is prepended by '`c`', the class index and the dollar sign; in this way '`cj`' can be used to distinguish among precision parameters in different classes; available after the estimation of the half-normal model)
`eff_i`	$\left(\sum_{i=1}^{N} T_i\right) \times 1$ vector that stores the expected values of the observation-specific efficiency scores, $\mathrm{E}\left(e^{-u_{it}}\right)$; the values in this vector are not guaranteed to be in the same order as the order in which the observations appear in the dataset used for estimation; use the `store()` function to associate the values in `eff_i` with the observations in the dataset
`pi_j`	vectors containing the draws from the posterior of each π_c, for $j = 0, 1, \ldots, C-1$ (these vectors are available only after the estimation of the model with unconditional class-membership probabilities)
`pi_j$z1,...,pi_j$zL`	vectors containing the draws from the posterior of the parameters associated with variables `z1,...,zL`, for $j = 1, \ldots, C-1$[8] (the names of these vectors are the names of the variables that were included the `z` list, in the right-hand side of the model, prepended by '`pi_j`' and the dollar sign; in this way '`pi_j`' can be used to distinguish among parameters associated with variables with different roles in the model, for example the same variable appearing in both `x` and `z` lists, as well as among parameters associated with a variable in the `z` list, but corresponding to different classes; these vectors are available only after the estimation of the model with conditional class-membership probabilities)
`pi_i`	$N \times C$ matrix that stores the expected values of the posterior class-membership probabilities for each group and for each of the C classes
`logML`	the Lewis & Raftery (1997) approximation of the log-marginal likelihood

[8] Indexing starts at 1 because the parameters of the multinomial-Logit part of the model associated with class 0 are normalized to zero for identification purposes.

`logML_CJ`	the Chib (1995)/Chib & Jeliazkov (2001) approximation to the log-marginal likelihood; this is available only if the model was estimated with the `"logML_CJ"=true` option
`nchains`	the number of chains that were used to estimate the model
`nburnin`	the number of burn-in draws per chain that were used when estimating the model
`ndraws`	the total number of retained draws from the posterior (=`chains` · `draws`)
`nthin`	value of the thinning parameter that was used when estimating the model
`nseed`	value of the seed for the random-number generator that was used when estimating the model
`nclasses`	number of classes used during the estimation of the model

Additionally, the following functions are available for post-estimation analysis (see section B.14):

- `diagnostics()`
- `test()`
- `pmp()`
- `store()`
- `mfx()`

The latent-class stochastic frontier model with panel data uses the `store()` function to associate the estimates of the efficiency scores (`eff_i`) or the estimates of the posterior class-membership probabilities (`pi_i`) with specific observations and store their values in the dataset used for estimation. The generic syntax for a statement involving the `store()` function after estimation of a latent-class stochastic frontier model and for each of these two quantities is:

```
store( eff_i , <new variable name> [, "model"=<model name>] );
```

and:

```
store( pi_i , <new variable name prefix> [, "model"=<model name>] );
```

The first statement will generate one additional variable in the dataset used for estimation of the model and its name will be the one provided as the second argument to `store()`. This second statement will generate C additional variables in the dataset used for estimation of the model, with names constructed by appending the class index $(0, 1, \ldots, C-1)$ to the prefix provided as the second argument to `store()`.

The latent-class stochastic frontier model with panel data and conditional class-membership probabilities uses the `mfx()` function to calculate and report the marginal effects of the variables in the `z` list on the prior class-membership probabilities that come from the multinomial-Logit part of the model. The generic syntax for a statement involving the `mfx()` function after estimation of a latent-class linear model with panel data and conditional class-membership probabilities is:

```
mfx( ["type"=1] [, "point"=<point of calculation>] [, "model"=<model name>] );
```

See the general documentation of the `mfx()` function (section B.14) for details on the optional arguments.

Examples

Example 1

```
myData = import("$BayESHOME/Datasets/dataset2.csv");
myData.constant = ones(rows(myData), 1);
set_pd( year, id, "dataset" = myData);

sf_lc(y ~ constant x1 x2);
```

Example 2

```
myData = import("$BayESHOME/Datasets/dataset2.csv");
myData.constant = ones(rows(myData), 1);
set_pd( year, id, "dataset" = myData);

myModel = sf_lc(y ~ constant x1 x2, "udist"="hnorm",
    "m0"=[2;0.6;0.3], "P" = 10*eye(3,3),
    "burnin"=10000, "draws"=30000, "thin"=2, "chains"=2, "classes"=2,
    "logML_CJ" = true, "dataset"=myData);

diagnostics("model"=myModel);

plot([myModel.c0$x1, myModel.c1$x1],
    "title"="\beta2 for the two classes");
plot([myModel.pi_0, myModel.pi_1],
    "title"="Prior class-membership probabilities");
```

5.7 Dynamic stochastic frontier

Mathematical representation

$$y_{it} = \mathbf{x}'_{it}\boldsymbol{\beta} + v_{it} \pm u_{it}, \qquad v_{it} \sim \mathrm{N}\left(0, \tfrac{1}{\tau}\right), \tag{5.7}$$

$$s_{it} = f\left(u_{it}\right) \tag{5.8}$$

$$s_{it} = \mathbf{z}'_{it}\boldsymbol{\delta} + \rho s_{i,t-1} + \xi_{it}, \qquad \xi_{it} \sim \mathrm{N}\left(0, \tfrac{1}{\phi}\right) \tag{5.9}$$

$$s_{i1} = \frac{\mathbf{z}'_{i1}\boldsymbol{\delta}}{1-\rho} + \xi_{i1}, \qquad \xi_{i1} \sim \mathrm{N}\left(0, \tfrac{1}{\phi(1-\rho)^2}\right) \tag{5.10}$$

- the model is estimated using observations from N groups, each group observed for T_i periods (balanced or unbalanced panels); the total number of observations is $\sum_{i=1}^{N} T_i$
- y_{it} is the value of the dependent variable for group i, observed in period t
- $\mathbf{x}_{it}$ is a $K \times 1$ vector that stores the values of the K independent variables for group i, observed in period t
- $\boldsymbol{\beta}$ is a $K \times 1$ vector of parameters associated with the variables in the observed equation
- τ is the precision of the noise component of the error term in the observed equation: $\sigma_v^2 = \frac{1}{\tau}$
- u_i is the inefficiency component of the error term
- s_{it} is the value of the hidden-state variable for group i, period t; this hidden state is a monotonic transformation of the inefficiency component of the error term: $s_{it} = f\left(u_{it}\right)$ and controls the autoregressive process of efficiency; BayES supports the following transformations:
  - $s_{it} = \log\left(\frac{e^{-u_{it}}}{1-e^{-u_{it}}}\right)$ as presented in Emvalomatis (2011); in this formulation the efficiency score, $e^{-u_{it}}$, follows, conditional on $s_{i,t-1}$, a logit-normal distribution
  - $s_{it} = \log\left(u_{it}\right)$ as presented in Tsionas (2006); in this formulation u_{it} follows, conditional on $s_{i,t-1}$, a log-normal distribution
- $\mathbf{z}_{it}$ is an $L \times 1$ vector that stores the values of the L determinants of inefficiency, as they enter equation (5.9), for group i, observed in period t
- $\boldsymbol{\delta}$ is an $L \times 1$ vector of parameters associated with the variables in the hidden-state equation
- ρ is a parameter that measures the persistence of inefficiency
- ϕ is the precision of the error term in the hidden-state equation: $\sigma_\xi^2 = \frac{1}{\phi}$

When u_i enters the specification with a plus sign then the model represents a cost frontier, while when u_i enters with a minus sign the model represents a production frontier. For the efficiency scores generated by a stochastic frontier model to be meaningful, the dependent variable in both cases must be in logarithms.

Due to Metropolis-Hastings updates used by BayES in the estimation of dynamic stochastic frontier models, the draws from the posterior are likely to have very large autocorrelation times. Therefore, long burn-ins are recommended (above 30,000 draws) and large thinning parameters if machine memory is an issue.

Priors

Parameter	Probability density function	Default hyperparameters
$\boldsymbol{\beta}$	$\mathrm{p}\left(\boldsymbol{\beta}\right) = \frac{\lvert\mathbf{P}_\beta\rvert^{1/2}}{(2\pi)^{K/2}} \exp\left\{-\frac{1}{2}\left(\boldsymbol{\beta}-\mathbf{m}_\beta\right)'\mathbf{P}_\beta\left(\boldsymbol{\beta}-\mathbf{m}_\beta\right)\right\}$	$\mathbf{m}_\beta = \mathbf{0}_K$, $\mathbf{P}_\beta = 0.001 \cdot \mathbf{I}_K$

table continues on next page

table continued from previous page

Parameter	Probability density function	Default hyperparameters
τ	$\mathrm{p}(\tau) = \frac{b_\tau^{a_\tau}}{\Gamma(a_\tau)} \tau^{a_\tau - 1} e^{-\tau b_\tau}$	$a_\tau = 0.001$, $b_\tau = 0.001$
$\boldsymbol{\delta}$	$\mathrm{p}(\boldsymbol{\delta}) = \frac{\lvert\mathbf{P}_\delta\rvert^{1/2}}{(2\pi)^{L/2}} \exp\left\{-\frac{1}{2}(\boldsymbol{\delta} - \mathbf{m}_\delta)' \mathbf{P}_\delta (\boldsymbol{\delta} - \mathbf{m}_\delta)\right\}$	$\mathbf{m}_\delta = \mathbf{0}_L$, $\mathbf{P}_\delta = 0.001 \cdot \mathbf{I}_L$
ρ	$\mathrm{p}(\rho) = \frac{\rho^{a_\rho - 1}(1-\rho)^{b_\rho - 1}}{B(a_\rho, b_\rho)}$	$a_\rho = 4.0$, $b_\rho = 2.0$
ϕ	$\mathrm{p}(\phi) = \frac{b_\phi^{a_\phi}}{\Gamma(a_\phi)} \phi^{a_\phi - 1} e^{-\phi b_\phi}$	$a_\phi = 0.1$, $b_\phi = 0.01$

Syntax

```
[<model name> = ] sf_dyn( y ~ x1 x2 ... xK | z1 z2 ... zL [, <options>] );
```

where:

- `y` is the dependent variable name, as it appears in the dataset used for estimation
- `x1 x2 ... xK` is a list of the K independent variable names, as they appear in the dataset used for estimation; when a constant term is to be included in the model, this must be requested explicitly
- `z1 z2 ... zL` is a list of the L variable names that enter the hidden-state equation, as they appear in the dataset used for estimation; when a constant term is to be included in the model, this must be requested explicitly; it is possible to run a model with an empty `z` list, however, it is recommended that at least a constant term is included

Groups that contain observations which are not consecutive according to the panel time variable (for example, a group is observed for two consecutive periods, not observed for the following period and observed again for another string of consecutive time periods) are split into multiple groups, with each string of consecutive observations treated as a different group. A warning is produced when the dataset used for estimation contains groups with gaps in the time dimension.

The optional arguments for the dynamic stochastic frontier model are:[9]

Gibbs parameters	
`"chains"`	number of chains to run in parallel (positive integer); the default value is 1
`"burnin"`	number of burn-in draws per chain (positive integer); the default value is 10000
`"draws"`	number of retained draws per chain (positive integer); the default value is 20000
`"thin"`	value of the thinning parameter (positive integer); the default value is 1
`"seed"`	value of the seed for the random-number generator (positive integer); the default value is 42
Model specification	
`"udist"`	specification of the distribution of the inefficiency component of the error term; the following options are available, corresponding to the distributions presented at the beginning of this section: • `"explogitn"` • `"logn"` the default value is `"explogitn"`
`"production"`	boolean specifying the type of frontier (production/cost); it could be set to either **true** (production) or **false** (cost); the default value is **true**

[9] Optional arguments are always given in option-value pairs (eg. `"chains"=3`).

Hyperparameters	
`"m_beta"`	mean vector of the prior for $\boldsymbol{\beta}$ ($K \times 1$ vector); the default value is $\mathbf{0}_K$
`"P_beta"`	precision matrix of the prior for $\boldsymbol{\beta}$ ($K \times K$ symmetric and positive-definite matrix); the default value is $0.001 \cdot \mathbf{I}_K$
`"a_tau"`	shape parameter of the prior for τ (positive number); the default value is 0.001
`"b_tau"`	rate parameter of the prior for τ (positive number); the default value is 0.001
`"m_delta"`	mean vector of the prior for $\boldsymbol{\delta}$ ($L \times 1$ vector); the default value is $\mathbf{0}_L$
`"P_delta"`	precision matrix of the prior for $\boldsymbol{\delta}$ ($L \times L$ symmetric and positive-definite matrix); the default value is $0.001 \cdot \mathbf{I}_L$
`"a_rho"`	alpha parameter of the prior for ρ (positive number); the default value is 4
`"b_rho"`	beta parameter of the prior for ρ (positive number); the default value is 2
`"a_phi"`	shape parameter of the prior for ϕ (positive number); the default value is 0.1
`"b_phi"`	rate parameter of the prior for ϕ (positive number); the default value is 0.01
`Dataset and log-marginal likelihood`	
`"dataset"`	the id value of the dataset that will be used for estimation; the default value is the first dataset in memory (in alphabetical order)
`"logML_CJ"`	boolean indicating whether the Chib (1995)/Chib & Jeliazkov (2001) approximation to the log-marginal likelihood should be calculated (**`true`**|**`false`**); the default value is **`false`**

Reported Parameters

$\boldsymbol{\beta}$	`variable_name`	vector of parameters associated with the independent variables in the `x` list
τ	`tau`	precision parameter of the noise component of the error term, v_i
$\boldsymbol{\delta}$	`variable_name`	vector of parameters associated with the independent variables in the `z` list
ϕ	`phi`	precision parameter of the error term in the hidden-state equation of the error term, u_i
σ_v	`sigma_v`	standard deviation of the noise component of the error term, $\sigma_v = 1/\tau^{1/2}$
σ_s	`sigma_s`	standard deviation of the error term in the hidden-state equation: $\sigma_\alpha = 1/\phi^{1/2}$

Stored values and post-estimation analysis

If a left-hand-side id value is provided when a dynamic stochastic frontier model is created, then the following results are saved in the model item and are accessible via the '.' operator:

`Samples`	a matrix containing the draws from the posterior of $\boldsymbol{\beta}$, τ, $\boldsymbol{\delta}$, ρ and ϕ
`y$x1,...,y$xK`	vectors containing the draws from the posterior of the parameters associated with variables `x1,...,xK` (the names of these vectors are the names of the variables that were included in the right-hand side of the model, prepended by `y$`, where `y` is the name of the dependent variable; this is done so that the samples on the parameters associated with a variable that appears in both `x` and `z` lists can be distinguished)
`tau`	vector containing the draws from the posterior of τ
`s$z1,...,s$zL`	vectors containing the draws from the posterior of the parameters associated with variables `z1,...,zL` (the names of these vectors are the names of the variables that were included in the `z` list, in the right-hand side of the model, prepended by `s$`; this is done so that the samples on the parameters associated with a variable that appears in both `x` and `z` lists can be distinguished)

`rho`	vector containing the draws from the posterior of ρ
`phi`	vector containing the draws from the posterior of ϕ (available after the estimation of the truncated-normal model)
`logML`	the Lewis & Raftery (1997) approximation of the log-marginal likelihood
`logML_CJ`	the Chib (1995)/Chib & Jeliazkov (2001) approximation to the log-marginal likelihood; this is available only if the model was estimated with the `"logML_CJ"=true` option
`eff_i`	$N \times 1$ vector that stores the expected values of the observation-specific efficiency scores, $\mathrm{E}\left(e^{-u_i}\right)$; the values in this vector are not guaranteed to be in the same order as the order in which the observations appear in the dataset used for estimation; use the `store()` function to associate the values in `eff_i` with the observations in the dataset
`nchains`	the number of chains that were used to estimate the model
`nburnin`	the number of burn-in draws per chain that were used when estimating the model
`ndraws`	the total number of retained draws from the posterior (=`chains` · `draws`)
`nthin`	value of the thinning parameter that was used when estimating the model
`nseed`	value of the seed for the random-number generator that was used when estimating the model

Additionally, the following functions are available for post-estimation analysis (see section B.14):

- `diagnostics()`
- `test()`
- `pmp()`
- `store()`
- `mfx()`

The dynamic stochastic frontier model uses the `store()` function to associate the estimates of the efficiency scores (`eff_i`) with specific observations and store their values in the dataset used for estimation. The generic syntax for a statement involving the `store()` function after estimation of a dynamic stochastic frontier model is:

```
store( eff_i , <new variable name> [, "model"=<model name>] );
```

Examples

Example 1

```
myData = import("$BayESHOME/Datasets/dataset1.csv", ",");
myData.constant = ones(rows(myData), 1);
set_pd( year, id, "dataset" = myData);

explogitnSF = sf_dyn( y ~ constant x1 x2 x3 | constant z2 );
```

Example 2

```
myData = import("$BayESHOME/Datasets/dataset1.csv", ",");
myData.constant = ones(rows(myData), 1);
set_pd( year, id, "dataset" = myData);

explogitnSF = sf_dyn( y ~ constant x1 x2 x3 | constant z2,
    "udist" = "explogitn" );

lognSF = sf_dyn( y ~ constant x1 x2 x3 | constant z2,
    "udist" = "logn" );

store( eff_i, eff_explogitn, "model" = explogitnSF );
store( eff_i, eff_logn, "model" = lognSF );

pmp( { explogitnSF, lognSF } );
```

5.8 Random-effects dynamic stochastic frontier

Mathematical representation

$$y_{it} = \alpha_i + \mathbf{x}'_{it}\boldsymbol{\beta} + v_{it} \pm u_{it}, \qquad v_{it} \sim \mathrm{N}\left(0, \tfrac{1}{\tau}\right), \quad \alpha_i \sim \mathrm{N}\left(0, \tfrac{1}{\omega}\right) \tag{5.11}$$

$$s_{it} = f\left(u_{it}\right) \tag{5.12}$$

$$s_{it} = \mathbf{z}'_{it}\boldsymbol{\delta} + \rho s_{i,t-1} + \xi_{it}, \qquad \xi_{it} \sim \mathrm{N}\left(0, \tfrac{1}{\phi}\right) \tag{5.13}$$

$$s_{i1} = \frac{\mathbf{z}'_{i1}\boldsymbol{\delta}}{1-\rho} + \xi_{i1}, \qquad \xi_{i1} \sim \mathrm{N}\left(0, \tfrac{1}{\phi(1-\rho)^2}\right) \tag{5.14}$$

- the model is estimated using observations from N groups, each group observed for T_i periods (balanced or unbalanced panels); the total number of observations is $\sum_{i=1}^{N} T_i$
- y_{it} is the value of the dependent variable for group i, observed in period t
- $\mathbf{x}_{it}$ is a $K \times 1$ vector that stores the values of the K independent variables for group i, observed in period t
- $\boldsymbol{\beta}$ is a $K \times 1$ vector of parameters associated with the variables in the observed equation
- τ is the precision of the noise component of the error term in the observed equation: $\sigma_v^2 = \frac{1}{\tau}$
- α_i is the group-specific error term for group i
- ω is precision of the group-specific error term: $\sigma_\alpha^2 = \frac{1}{\omega}$
- u_i is the inefficiency component of the error term
- s_{it} is the value of the hidden-state variable for group i, period t; this hidden state is a monotonic transformation of the inefficiency component of the error term: $s_{it} = f\left(u_{it}\right)$ and controls the autoregressive process of efficiency; BayES supports the following transformations:
  - $s_{it} = \log\left(\frac{e^{-u_{it}}}{1-e^{-u_{it}}}\right)$ as presented in Emvalomatis (2011); in this formulation the efficiency score, $e^{-u_{it}}$, follows, conditional on $s_{i,t-1}$, a logit-normal distribution
  - $s_{it} = \log\left(u_{it}\right)$ as presented in Tsionas (2006); in this formulation u_{it} follows, conditional on $s_{i,t-1}$, a log-normal distribution
- $\mathbf{z}_{it}$ is an $L \times 1$ vector that stores the values of the L determinants of inefficiency, as they enter equation (5.13), for group i, observed in period t
- $\boldsymbol{\delta}$ is an $L \times 1$ vector of parameters associated with the variables in the hidden-state equation
- ρ is a parameter that measures the persistence of inefficiency
- ϕ is the precision of the error term in the hidden-state equation: $\sigma_\xi^2 = \frac{1}{\phi}$

When u_i enters the specification with a plus sign then the model represents a cost frontier, while when u_i enters with a minus sign the model represents a production frontier. For the efficiency scores generated by a stochastic frontier model to be meaningful, the dependent variable in both cases must be in logarithms.

The mean of the distribution of the α_is is restricted to zero and, therefore, these are simply group-specific errors terms. However, including a constant term in the set of independent variables is valid and leads to a specification equivalent to one where the group effects are draws from a normal distribution with mean equal to the parameter associated with the constant term and precision ω.

Due to Metropolis-Hastings updates used by BayES in the estimation of dynamic stochastic frontier models, the draws from the posterior are likely to have very large autocorrelation times. Therefore, long burn-ins are recommended (above 30,000 draws) and large thinning parameters if machine memory is an issue.

Priors

Parameter	Probability density function	Default hyperparameters
$\boldsymbol{\beta}$	$\mathrm{p}(\boldsymbol{\beta}) = \frac{\lvert\mathbf{P}_\beta\rvert^{1/2}}{(2\pi)^{K/2}} \exp\left\{-\frac{1}{2}(\boldsymbol{\beta}-\mathbf{m}_\beta)'\mathbf{P}_\beta(\boldsymbol{\beta}-\mathbf{m}_\beta)\right\}$	$\mathbf{m}_\beta = \mathbf{0}_K$, $\mathbf{P}_\beta = 0.001 \cdot \mathbf{I}_K$
τ	$\mathrm{p}(\tau) = \frac{b_\tau^{a_\tau}}{\Gamma(a_\tau)}\tau^{a_\tau-1}e^{-\tau b_\tau}$	$a_\tau = 0.001$, $b_\tau = 0.001$
ω	$\mathrm{p}(\omega) = \frac{b_\omega^{a_\omega}}{\Gamma(a_\omega)}\omega^{a_\omega-1}e^{-\omega b_\omega}$	$a_\omega = 0.01$, $b_\omega = 0.001$
$\boldsymbol{\delta}$	$\mathrm{p}(\boldsymbol{\delta}) = \frac{\lvert\mathbf{P}_\delta\rvert^{1/2}}{(2\pi)^{L/2}} \exp\left\{-\frac{1}{2}(\boldsymbol{\delta}-\mathbf{m}_\delta)'\mathbf{P}_\delta(\boldsymbol{\delta}-\mathbf{m}_\delta)\right\}$	$\mathbf{m}_\delta = \mathbf{0}_L$, $\mathbf{P}_\delta = 0.001 \cdot \mathbf{I}_L$
ρ	$\mathrm{p}(\rho) = \frac{\rho^{a_\rho-1}(1-\rho)\rho^{b_\rho-1}}{B(a_\rho,b_\rho)}$	$a_\rho = 4.0$, $b_\rho = 2.0$
ϕ	$\mathrm{p}(\phi) = \frac{b_\phi^{a_\phi}}{\Gamma(a_\phi)}\phi^{a_\phi-1}e^{-\phi b_\phi}$	$a_\phi = 0.1$, $b_\phi = 0.01$

Syntax

```
[<model name> = ] sf_dyn_re( y ~ x1 x2 ... xK | z1 z2 ... zL [,<options> ] );
```

where:

- `y` is the dependent variable name, as it appears in the dataset used for estimation
- `x1 x2 ... xK` is a list of the K independent variable names, as they appear in the dataset used for estimation; when a constant term is to be included in the model, this must be requested explicitly
- `z1 z2 ... zL` is a list of the L variable names that enter the hidden-state equation, as they appear in the dataset used for estimation; when a constant term is to be included in the model, this must be requested explicitly; it is possible to run a model with an empty `z` list, however, it is recommended that at least a constant term is included

BayES automatically drops from the sample used for estimation groups which are observed only once. This is because for these groups the group effect (α_i) cannot be distinguished from the noise component of the error term (v_{it}).

Groups that contain observations which are not consecutive according to the panel time variable (for example, a group is observed for two consecutive periods, not observed for the following period and observed again for another string of consecutive time periods) are split into multiple groups, with each string of consecutive observations treated as a different group. A warning is produced when the dataset used for estimation contains groups with gaps in the time dimension.

The optional arguments for the random-effects dynamic stochastic frontier model are:[10]

[10] Optional arguments are always given in option-value pairs (eg. `"chains"=3`).

Gibbs parameters	
`"chains"`	number of chains to run in parallel (positive integer); the default value is 1
`"burnin"`	number of burn-in draws per chain (positive integer); the default value is 10000
`"draws"`	number of retained draws per chain (positive integer); the default value is 20000
`"thin"`	value of the thinning parameter (positive integer); the default value is 1
`"seed"`	value of the seed for the random-number generator (positive integer); the default value is 42
Model specification	
`"udist"`	specification of the distribution of the inefficiency component of the error term; the following options are available, corresponding to the distributions presented at the beginning of this section: • `"explogitn"` • `"logn"` the default value is `"explogitn"`
`"production"`	boolean specifying the type of frontier (production/cost); it could be set to either **true** (production) or **false** (cost); the default value is **true**
Hyperparameters	
`"m_beta"`	mean vector of the prior for $\boldsymbol{\beta}$ ($K \times 1$ vector); the default value is $\mathbf{0}_K$
`"P_beta"`	precision matrix of the prior for $\boldsymbol{\beta}$ ($K \times K$ symmetric and positive-definite matrix); the default value is $0.001 \cdot \mathbf{I}_K$
`"a_tau"`	shape parameter of the prior for τ (positive number); the default value is 0.001
`"b_tau"`	rate parameter of the prior for τ (positive number); the default value is 0.001
`"a_omega"`	shape parameter of the prior for ω (positive number); the default value is 0.01
`"b_omega"`	rate parameter of the prior for ω (positive number); the default value is 0.001
`"m_delta"`	mean vector of the prior for $\boldsymbol{\delta}$ ($L \times 1$ vector); the default value is $\mathbf{0}_L$
`"P_delta"`	precision matrix of the prior for $\boldsymbol{\delta}$ ($L \times L$ symmetric and positive-definite matrix); the default value is $0.001 \cdot \mathbf{I}_L$
`"a_rho"`	alpha parameter of the prior for ρ (positive number); the default value is 4
`"b_rho"`	beta parameter of the prior for ρ (positive number); the default value is 2
`"a_phi"`	shape parameter of the prior for ϕ (positive number); the default value is 0.1
`"b_phi"`	rate parameter of the prior for ϕ (positive number); the default value is 0.01
Dataset and log-marginal likelihood	
`"dataset"`	the id value of the dataset that will be used for estimation; the default value is the first dataset in memory (in alphabetical order)
`"logML_CJ"`	boolean indicating whether the Chib (1995)/Chib & Jeliazkov (2001) approximation to the log-marginal likelihood should be calculated (**true**|**false**); the default value is **false**

Reported Parameters

$\boldsymbol{\beta}$	`variable_name`	vector of parameters associated with the independent variables in the `x` list
τ	`tau`	precision parameter of the noise component of the error term, v_i
ω	`omega`	precision parameter of the group-specific error term, α_i
$\boldsymbol{\delta}$	`variable_name`	vector of parameters associated with the independent variables in the `z` list
ϕ	`phi`	precision parameter of the error term in the hidden-state equation of the error term, u_i

table continues on next page

table continued from previous page

σ_v	`sigma_v`	standard deviation of the noise component of the error term, $\sigma_v = 1/\tau^{1/2}$
σ_α	`sigma_alpha`	standard deviation of the group-specific error term: $\sigma_\alpha = 1/\omega^{1/2}$
σ_s	`sigma_s`	standard deviation of the error term in the hidden-state equation: $\sigma_\alpha = 1/\phi^{1/2}$

Stored values and post-estimation analysis

If a left-hand-side id value is provided when a random-effects dynamic stochastic frontier model is created, then the following results are saved in the model item and are accessible via the '.' operator:

`Samples`	a matrix containing the draws from the posterior of $\boldsymbol{\beta}$, τ, ω, $\boldsymbol{\delta}$, ρ and ϕ
`y$x1,...,y$xK`	vectors containing the draws from the posterior of the parameters associated with variables `x1,...,xK` (the names of these vectors are the names of the variables that were included in the right-hand side of the model, prepended by `y$`, where `y` is the name of the dependent variable; this is done so that the samples on the parameters associated with a variable that appears in both `x` and `z` lists can be distinguished)
`tau`	vector containing the draws from the posterior of τ
`omega`	vector containing the draws from the posterior of ω
`s$z1,...,s$zL`	vectors containing the draws from the posterior of the parameters associated with variables `z1,...,zL` (the names of these vectors are the names of the variables that were included in the `z` list, in the right-hand side of the model, prepended by `s$`; this is done so that the samples on the parameters associated with a variable that appears in both `x` and `z` lists can be distinguished)
`rho`	vector containing the draws from the posterior of ρ
`phi`	vector containing the draws from the posterior of ϕ (available after the estimation of the truncated-normal model)
`logML`	the Lewis & Raftery (1997) approximation of the log-marginal likelihood
`logML_CJ`	the Chib (1995)/Chib & Jeliazkov (2001) approximation to the log-marginal likelihood; this is available only if the model was estimated with the `"logML_CJ"=true` option
`alpha_i`	$N \times 1$ vector that stores the group-specific errors; the values in this vector are not guaranteed to be in the same order as the order in which the groups appear in the dataset used for estimation; use the `store()` function to associate the values in `alpha_i` with the observations in the dataset
`eff_i`	$N \times 1$ vector that stores the expected values of the observation-specific efficiency scores, $\mathrm{E}\left(e^{-u_i}\right)$; the values in this vector are not guaranteed to be in the same order as the order in which the observations appear in the dataset used for estimation; use the `store()` function to associate the values in `eff_i` with the observations in the dataset
`nchains`	the number of chains that were used to estimate the model
`nburnin`	the number of burn-in draws per chain that were used when estimating the model
`ndraws`	the total number of retained draws from the posterior (=`chains` · `draws`)
`nthin`	value of the thinning parameter that was used when estimating the model
`nseed`	value of the seed for the random-number generator that was used when estimating the model

Additionally, the following functions are available for post-estimation analysis (see section B.14):

- diagnostics()
- test()
- pmp()
- store()
- mfx()

The random-effects dynamic stochastic frontier model uses the store() function to associate the group effects (alpha_i) or the estimates of the efficiency scores (eff_i) with specific observations and store their values in the dataset used for estimation. The generic syntax for a statement involving the store() function after estimation of a random-effects dynamic stochastic frontier model and for each of these two quantities is:

```
store( alpha_i , <new variable name> [, "model"=<model name>] );
```

and:

```
store( eff_i , <new variable name> [, "model"=<model name>] );
```

Examples

Example 1

```
myData = import("$BayESHOME/Datasets/dataset1.csv", ",");
myData.constant = ones(rows(myData), 1);
set_pd( year, id, "dataset" = myData);

explogitnSF = sf_dyn_re( y ~ constant x1 x2 x3 | constant z2 );
```

Example 2

```
myData = import("$BayESHOME/Datasets/dataset1.csv", ",");
myData.constant = ones(rows(myData), 1);
set_pd( year, id, "dataset" = myData);

explogitnSF = sf_dyn_re( y ~ constant x1 x2 x3 | constant z2,
    "udist" = "explogitn" );

lognSF = sf_dyn_re( y ~ constant x1 x2 x3 | constant z2,
    "udist" = "logn" );

store( alpha_i, re_explogitn, "model" = explogitnSF );
store( alpha_i, re_logn, "model" = lognSF );

store( eff_i, eff_explogitn, "model" = explogitnSF );
store( eff_i, eff_logn, "model" = lognSF );

pmp( { explogitnSF, lognSF } );
```

Chapter 6

Discrete Choice Models

6.1 Binary Probit

Mathematical representation

$$\text{Prob}\left(y_i = 1\right) = \Phi\left(\mathbf{x}_i'\boldsymbol{\beta}\right) \tag{6.1}$$

- the model is estimated using N observations
- y_i is the value of the dependent variable for observation i and it can take two values: 0 and 1
- $\mathbf{x}_i$ is a $K \times 1$ vector that stores the values of the K independent variables for observation i
- $\boldsymbol{\beta}$ is a $K \times 1$ vector of parameters
- $\Phi\left(\cdot\right)$ is the standard-Normal cdf

An equivalent representation uses the latent variable y_i^*:

$$\begin{aligned} y_i^* &= \mathbf{x}_i'\boldsymbol{\beta} + \varepsilon_i, \qquad \varepsilon_i \sim \text{N}\left(0,1\right) \\ y_i &= \begin{cases} 1 & \text{if} \quad y_i^* > 0 \\ 0 & \text{if} \quad y_i^* \leq 0 \end{cases} \end{aligned} \tag{6.2}$$

Priors

Parameter	Probability density function	Default hyperparameters
$\boldsymbol{\beta}$	$\text{p}\left(\boldsymbol{\beta}\right) = \frac{\left\|\mathbf{P}\right\|^{1/2}}{\left(2\pi\right)^{K/2}} \exp\left\{-\frac{1}{2}\left(\boldsymbol{\beta}-\mathbf{m}\right)'\mathbf{P}\left(\boldsymbol{\beta}-\mathbf{m}\right)\right\}$	$\mathbf{m} = \mathbf{0}_K$, $\mathbf{P} = 0.001 \cdot \mathbf{I}_K$

Syntax

```
[<model name> = ] probit( y ~ x1 x2 ... xK [, <options> ] );
```

where:

- `y` is the dependent variable name, as it appears in the dataset used for estimation
- `x1 x2 ... xK` is a list of the K independent variable names, as they appear in the dataset used for estimation; when a constant term is to be included in the model, this must be requested explicitly

The dependent variable, `y`, in the dataset used for estimation must contain only two values: 0 and 1 (with 1 indicating "success"). Observations with missing values in `y` are dropped during estimation, but if a numerical value other than 0 and 1 is encountered, then an error is produced.

The optional arguments for the binary Probit model are:[1]

Gibbs parameters	
`"chains"`	number of chains to run in parallel (positive integer); the default value is 1
`"burnin"`	number of burn-in draws per chain (positive integer); the default value is 10000
`"draws"`	number of retained draws per chain (positive integer); the default value is 20000
`"thin"`	value of the thinning parameter (positive integer); the default value is 1

[1] Optional arguments are always given in option-value pairs (eg. `"chains"=3`).

`"seed"`	value of the seed for the random-number generator (positive integer); the default value is 42
`Hyperparameters`	
`"m"`	mean vector of the prior for $\boldsymbol{\beta}$ ($K \times 1$ vector); the default value is $\mathbf{0}_K$
`"P"`	precision matrix of the prior for $\boldsymbol{\beta}$ ($K \times K$ symmetric and positive-definite matrix); the default value is $0.001 \cdot \mathbf{I}_K$
`Dataset and log-marginal likelihood`	
`"dataset"`	the id value of the dataset that will be used for estimation; the default value is the first dataset in memory (in alphabetical order)
`"logML_CJ"`	boolean indicating whether the Chib (1995)/Chib & Jeliazkov (2001) approximation to the log-marginal likelihood should be calculated (**true**|**false**); the default value is **false**

Reported Parameters

$\boldsymbol{\beta}$	`variable_name`	vector of parameters associated with the independent variables

Stored values and post-estimation analysis

If a left-hand-side id value is provided when a binary Probit model is created, then the following results are saved in the model item and are accessible via the '.' operator:

`Samples`	a matrix containing the draws from the posterior of $\boldsymbol{\beta}$
`x1,...,xK`	vectors containing the draws from the posterior of the parameters associated with variables `x1,...,xK` (the names of these vectors are the names of the variables that were included in the right-hand side of the model)
`logML`	the Lewis & Raftery (1997) approximation of the log-marginal likelihood
`logML_CJ`	the Chib (1995)/Chib & Jeliazkov (2001) approximation to the log-marginal likelihood; this is available only if the model was estimated with the `"logML_CJ"`=**true** option
`nchains`	the number of chains that were used to estimate the model
`nburnin`	the number of burn-in draws per chain that were used when estimating the model
`ndraws`	the total number of retained draws from the posterior (=`chains` · `draws`)
`nthin`	value of the thinning parameter that was used when estimating the model
`nseed`	value of the seed for the random-number generator that was used when estimating the model

Additionally, the following functions are available for post-estimation analysis (see section B.14):

- `diagnostics()`
- `pmp()`
- `predict()`
- `test()`
- `mfx()`

The binary Probit model uses the `mfx()` function to calculate and report the marginal effects of the independent variables on the probability of success. Because the model calculates only one type of marginal effects, the only valid value for the `"type"` option is `1`. The generic syntax for a statement involving the `mfx()` function after estimation of a binary Probit model is:

```
mfx( ["type"=1] [, "point"=<point of calculation>] [, "model"=<model name>] );
```

See the general documentation of the `mfx()` function (section B.14) for details on the other optional arguments.

The binary Probit model uses the `predict()` function to generate predictions of the probability of success. Because the model generates only one type of predictions, the only valid value for the `"type"` option is `1`. The generic syntax for a statement involving the `predict()` function after estimation of a binary Probit model is:

```
[<id value>] = predict( ["type"=1] [, "point"=<point of calculation>] [,"model"=<model
    name>] [, "stats"=true|false] [, "prefix"=<prefix for new variable name>] );
```

See the general documentation of the `predict()` function (section B.14) for details on the other optional arguments.

Examples

Example 1

```
myData = import("$BayESHOME/Datasets/dataset4.csv");
myData.constant = ones(rows(myData), 1);

probit( y ~ constant x1 x2 x3 x4 );
```

Example 2

```
myData = import("$BayESHOME/Datasets/dataset4.csv");
myData.constant = ones(rows(myData), 1);

myModel = probit( y ~ constant x1 x2 x3 x4,
    "m"=ones(5,1), "P"=0.1*eye(5,5),
    "burnin"=10000, "draws"=40000, "thin"=4, "chains"=2,
    "logML_CJ" = true );

diagnostics("model"=myModel);

kden(myModel.x3, "title" = "\beta3 from the Probit model");

margeff_mean = mfx("point"="mean","model"=myModel);
margeff_median = mfx("point"="median","model"=myModel);
margeff_eachpoint = mfx("point"="x_i","model"=myModel);
margeff_atx = mfx("point"=[1,1,0.5,2,0],"model"=myModel);

predict();
```

6.2 Binary Logit

Mathematical representation

$$\text{Prob}\left(y_i = 1\right) = \Lambda\left(\mathbf{x}_i'\boldsymbol{\beta}\right) \tag{6.3}$$

- the model is estimated using N observations
- y_i is the value of the dependent variable for observation i and it can take two values: 0 and 1
- $\mathbf{x}_i$ is a $K \times 1$ vector that stores the values of the K independent variables for observation i
- $\boldsymbol{\beta}$ is a $K \times 1$ vector of parameters
- $\Lambda\left(\cdot\right)$ is the standard-Logistic cdf

An equivalent representation uses the latent variable y_i^*:

$$\begin{aligned} y_i^* &= \mathbf{x}_i'\boldsymbol{\beta} + \varepsilon_i, \qquad \varepsilon_i \sim \text{Logistic}\left(0,1\right) \\ y_i &= \begin{cases} 1 & \text{if} \quad y_i^* > 0 \\ 0 & \text{if} \quad y_i^* \leq 0 \end{cases} \end{aligned} \tag{6.4}$$

Priors

Parameter	Probability density function	Default hyperparameters
$\boldsymbol{\beta}$	$\text{p}\left(\boldsymbol{\beta}\right) = \frac{\left\|\mathbf{P}\right\|^{1/2}}{\left(2\pi\right)^{K/2}} \exp\left\{-\frac{1}{2}\left(\boldsymbol{\beta}-\mathbf{m}\right)'\mathbf{P}\left(\boldsymbol{\beta}-\mathbf{m}\right)\right\}$	$\mathbf{m} = \mathbf{0}_K$, $\mathbf{P} = 0.001 \cdot \mathbf{I}_K$

Syntax

```
[<model name> = ] logit( y ~ x1 x2 ... xK [, <options> ] );
```

where:

- `y` is the dependent variable name, as it appears in the dataset used for estimation
- `x1 x2 ... xK` is a list of the K independent variable names, as they appear in the dataset used for estimation; when a constant term is to be included in the model, this must be requested explicitly

The dependent variable, `y`, in the dataset used for estimation must contain only two values: 0 and 1 (with 1 indicating "success"). Observations with missing values in `y` are dropped during estimation, but if a numerical value other than 0 and 1 is encountered, then an error is produced.

The optional arguments for the binary Logit model are:[2]

Gibbs parameters	
`"chains"`	number of chains to run in parallel (positive integer); the default value is 1
`"burnin"`	number of burn-in draws per chain (positive integer); the default value is 10000
`"draws"`	number of retained draws per chain (positive integer); the default value is 20000
`"thin"`	value of the thinning parameter (positive integer); the default value is 1

[2]Optional arguments are always given in option-value pairs (eg. `"chains"=3`).

`"seed"`	value of the seed for the random-number generator (positive integer); the default value is 42
`Hyperparameters`	
`"m"`	mean vector of the prior for $\boldsymbol{\beta}$ ($K \times 1$ vector); the default value is $\mathbf{0}_K$
`"P"`	precision matrix of the prior for $\boldsymbol{\beta}$ ($K \times K$ symmetric and positive-definite matrix); the default value is $0.001 \cdot \mathbf{I}_K$
`Dataset and log-marginal likelihood`	
`"dataset"`	the id value of the dataset that will be used for estimation; the default value is the first dataset in memory (in alphabetical order)
`"logML_CJ"`	boolean indicating whether the Chib (1995)/Chib & Jeliazkov (2001) approximation to the log-marginal likelihood should be calculated (**true**|**false**); the default value is **false**

Reported Parameters

$\boldsymbol{\beta}$	`variable_name`	vector of parameters associated with the independent variables

Stored values and post-estimation analysis

If a left-hand-side id value is provided when a binary Logit model is created, then the following results are saved in the model item and are accessible via the '.' operator:

`Samples`	a matrix containing the draws from the posterior of $\boldsymbol{\beta}$
`x1,...,xK`	vectors containing the draws from the posterior of the parameters associated with variables `x1,...,xK` (the names of these vectors are the names of the variables that were included in the right-hand side of the model)
`logML`	the Lewis & Raftery (1997) approximation of the log-marginal likelihood
`logML_CJ`	the Chib (1995)/Chib & Jeliazkov (2001) approximation to the log-marginal likelihood; this is available only if the model was estimated with the `"logML_CJ"`=**true** option
`nchains`	the number of chains that were used to estimate the model
`nburnin`	the number of burn-in draws per chain that were used when estimating the model
`ndraws`	the total number of retained draws from the posterior (=`chains` · `draws`)
`nthin`	value of the thinning parameter that was used when estimating the model
`nseed`	value of the seed for the random-number generator that was used when estimating the model

Additionally, the following functions are available for post-estimation analysis (see section B.14):

- `diagnostics()`
- `pmp()`
- `predict()`
- `test()`
- `mfx()`

The binary Logit model uses the `mfx()` function to calculate and report the marginal effects of the independent variables on the probability of success. Because the model calculates only one type of marginal effects, the only valid value for the optional `"type"` is `1`. The generic syntax for a statement involving the `mfx()` function after estimation of a binary Logit model is:

```
mfx( ["type"=1] [, "point"=<point of calculation>] [, "model"=<model name>] );
```

See the general documentation of the `mfx()` function (section B.14) for details on the other optional arguments.

The binary Logit model uses the `predict()` function to generate predictions of the probability of success. Because the model generates only one type of predictions, the only valid value for the `"type"` option is `1`. The generic syntax for a statement involving the `predict()` function after estimation of a binary Logit model is:

```
[<id value>] = predict( ["type"=1] [, "point"=<point of calculation>] [,"model"=<model
    name>] [, "stats"=true|false] [, "prefix"=<prefix for new variable name>] );
```

See the general documentation of the `predict()` function (section B.14) for details on the other optional arguments.

Examples

Example 1

```
myData = import("$BayESHOME/Datasets/dataset4.csv");
myData.constant = ones(rows(myData), 1);

logit( y ~ constant x1 x2 x3 x4, "logML_CJ" = true );
```

Example 2

```
myData = import("$BayESHOME/Datasets/dataset4.csv");
myData.constant = ones(rows(myData), 1);

myModel = logit( y ~ constant x1 x2 x3 x4,
    "m"=ones(5,1), "P"=0.1*eye(5,5),
    "burnin"=10000, "draws"=40000, "thin"=4, "chains"=2,
    "logML_CJ" = true );

diagnostics("model"=myModel);

kden(myModel.x3, "title" = "beta3 from the Logit model");

margeff_mean = mfx("point"="mean","model"=myModel);
margeff_median = mfx("point"="median","model"=myModel);
margeff_eachpoint = mfx("point"="x_i","model"=myModel);
margeff_atx = mfx("point"=[1,1,0.5,2,0],"model"=myModel);

predict();
```

6.3 Random-effects binary Probit

Mathematical representation

$$\text{Prob}\left(y_{it}=1\right)=\Phi\left(\alpha_i+\mathbf{x}_{it}'\boldsymbol{\beta}\right), \qquad \alpha_i\sim \text{N}\left(0,\tfrac{1}{\omega}\right) \tag{6.5}$$

- the model is estimated using observations from N groups, each group observed for T_i periods (balanced or unbalanced panels); the total number of observations is $\sum_{i=1}^{N} T_i$
- y_{it} is the value of the dependent variable for group i, observed in period t and it can take two values: 0 and 1
- $\mathbf{x}_{it}$ is a $K\times 1$ vector that stores the values of the K independent variables for group i, observed in period t
- $\boldsymbol{\beta}$ is a $K\times 1$ vector of parameters
- α_i is the group-specific error term for group i
- ω is the precision of the group-specific error term: $\sigma_\alpha^2=\frac{1}{\omega}$
- $\Phi\left(\cdot\right)$ is the standard-Normal cdf

An equivalent representation uses the latent variable y_{it}^*:

$$\begin{aligned} y_{it}^* &= \alpha_i+\mathbf{x}_{it}'\boldsymbol{\beta}+\varepsilon_{it}, \qquad \varepsilon_{it}\sim \text{N}\left(0,1\right), \quad \alpha_i\sim \text{N}\left(0,\tfrac{1}{\omega}\right), \\ y_{it} &= \begin{cases} 1 & \text{if} \quad y_{it}^*>0 \\ 0 & \text{if} \quad y_{it}^*\leq 0 \end{cases} \end{aligned} \tag{6.6}$$

The mean of the distribution of the α_is is restricted to zero and, therefore, these are simply group-specific errors terms. However, including a constant term in the set of independent variables is valid and leads to a specification equivalent to one where the group effects are draws from a normal distribution with mean equal to the parameter associated with the constant term and precision ω.

Priors

Parameter	Probability density function	Default hyperparameters
$\boldsymbol{\beta}$	$\text{p}\left(\boldsymbol{\beta}\right)=\frac{\left\|\mathbf{P}\right\|^{1/2}}{(2\pi)^{K/2}}\exp\left\{-\frac{1}{2}\left(\boldsymbol{\beta}-\mathbf{m}\right)'\mathbf{P}\left(\boldsymbol{\beta}-\mathbf{m}\right)\right\}$	$\mathbf{m}=\mathbf{0}_K$, $\mathbf{P}=0.001\cdot\mathbf{I}_K$
ω	$\text{p}\left(\omega\right)=\frac{b_\omega^{a_\omega}}{\Gamma(a_\omega)}\omega^{a_\omega-1}e^{-\omega b_\omega}$	$a_\omega=0.01$, $b_\omega=0.001$

Syntax

```
[<model name> = ] probit_re( y ~ x1 x2 ... xK [, <options> ] );
```

where:

- `y` is the dependent variable name, as it appears in the dataset used for estimation
- `x1 x2 ... xK` is a list of the K independent variable names, as they appear in the dataset used for estimation; when a constant term is to be included in the model, this must be requested explicitly

The dependent variable, `y`, in the dataset used for estimation must contain only two values: 0 and 1 (with 1 indicating "success"). Observations with missing values in `y` are dropped during estimation, but if a numerical value other than 0 and 1 is encountered, then an error is produced.

BayES automatically drops from the sample used for estimation groups which are observed only once. This is because for these groups the group effect (α_i) cannot be distinguished from the error term (ε_{it}).

The optional arguments for the random-effects binary Probit model are:[3]

`Gibbs parameters`	
`"chains"`	number of chains to run in parallel (positive integer); the default value is 1
`"burnin"`	number of burn-in draws per chain (positive integer); the default value is 10000
`"draws"`	number of retained draws per chain (positive integer); the default value is 20000
`"thin"`	value of the thinning parameter (positive integer); the default value is 1
`"seed"`	value of the seed for the random-number generator (positive integer); the default value is 42
`Hyperparameters`	
`"m"`	mean vector of the prior for $\boldsymbol{\beta}$ ($K \times 1$ vector); the default value is $\mathbf{0}_K$
`"P"`	precision matrix of the prior for $\boldsymbol{\beta}$ ($K \times K$ symmetric and positive-definite matrix); the default value is $0.001 \cdot \mathbf{I}_K$
`"a_omega"`	shape parameter of the prior for ω (positive number); the default value is 0.01
`"b_omega"`	rate parameter of the prior for ω (positive number); the default value is 0.001
`Dataset and log-marginal likelihood`	
`"dataset"`	the id value of the dataset that will be used for estimation; the default value is the first dataset in memory (in alphabetical order)
`"logML_CJ"`	boolean indicating whether the Chib (1995)/Chib & Jeliazkov (2001) approximation to the log-marginal likelihood should be calculated (**true**|**false**); the default value is **false**

Reported Parameters

$\boldsymbol{\beta}$	`variable_name`	vector of parameters associated with the independent variables
ω	`omega`	precision parameter of the group-specific error term, α_i
σ_α	`sigma_alpha`	standard deviation of the group-specific error term: $\sigma_\alpha = 1/\omega^{1/2}$

Stored values and post-estimation analysis

If a left-hand-side id value is provided when a random-effects binary Probit model is created, then the following results are saved in the model item and are accessible via the '.' operator:

`Samples`	a matrix containing the draws from the posterior of $\boldsymbol{\beta}$ and ω
`x1,...,xK`	vectors containing the draws from the posterior of the parameters associated with variables `x1,...,xK` (the names of these vectors are the names of the variables that were included in the right-hand side of the model)
`omega`	vector containing the draws from the posterior of ω
`logML`	the Lewis & Raftery (1997) approximation of the log-marginal likelihood
`logML_CJ`	the Chib (1995)/Chib & Jeliazkov (2001) approximation to the log-marginal likelihood; this is available only if the model was estimated with the `"logML_CJ"=`**true** option

[3] Optional arguments are always given in option-value pairs (eg. `"chains"=3`).

`alpha_i`	$N \times 1$ vector that stores the group-specific errors; the values in this vector are not guaranteed to be in the same order as the order in which the groups appear in the dataset used for estimation; use the `store()` function to associate the values in `alpha_i` with the observations in the dataset
`nchains`	the number of chains that were used to estimate the model
`nburnin`	the number of burn-in draws per chain that were used when estimating the model
`ndraws`	the total number of retained draws from the posterior (=`chains` · `draws`)
`nthin`	value of the thinning parameter that was used when estimating the model
`nseed`	value of the seed for the random-number generator that was used when estimating the model

Additionally, the following functions are available for post-estimation analysis (see section B.14):

- `diagnostics()`
- `pmp()`
- `mfx()`
- `test()`
- `store()`
- `predict()`

The random-effects binary Probit model uses the `store()` function to associate the group effects (`alpha_i`) with specific observations and store their values in the dataset used for estimation. The generic syntax for a statement involving the `store()` function after estimation of a random-effects binary Probit model is:

```
store( alpha_i , <new variable name>, ["model"=<model name>] );
```

The random-effects binary Probit model uses the `mfx()` function to calculate and report the marginal effects of the independent variables on the probability of success. There are two types of marginal effects which can be requested by setting the `"type"` argument of the `mfx()` function equal to `1` or `2`:

1. when `"type"=1` the marginal effects are calculated marginally with respect to the group effects.

2. when `"type"=2` the marginal effects are calculated conditionally on the group-effects being equal to zero (the expected value of the group effects, when treated as group-specific errors).

The generic syntax for a statement involving the `mfx()` function after estimation of a random-effects binary Probit model is:

```
mfx( ["type"=1] [, "point"=<point of calculation>] [, "model"=<model name>] );
```

and:

```
mfx( "type"=2 [, "point"=<point of calculation>] [, "model"=<model name>] );
```

for calculating these two types of marginal effects. The default value of the `"type"` option is `1`. See the general documentation of the `mfx()` function (section B.14) for details on the other optional arguments.

The random-effects binary Probit model uses the `predict()` function to generate predictions of the probability of success. There are two types of predictions which can be requested by setting the `"type"` argument of the `mfx()` function equal to `1` or `2`:

1. when `"type"=1` the predictions are generated marginally with respect to the group effects.

2. when `"type"=2` the predictions are generated conditionally on the group-effects being equal to zero (the expected value of the group effects, when treated as group-specific errors).

The generic syntax for a statement involving the `predict()` function after estimation of a random-effects binary Probit model is:

```
[<id value>] = predict( ["type"=1] [, "point"=<point of calculation>] [,"model"=<model
    name>] [, "stats"=true|false] [, "prefix"=<prefix for new variable name>] );
```

and:

```
[<id value>] = predict( "type"=2 [, "point"=<point of calculation>] [,"model"=<model
    name>] [, "stats"=true|false] [, "prefix"=<prefix for new variable name>] );
```

for generating these two types of predictions effects. The default value of the `"type"` option is `1`. See the general documentation of the `predict()` function (section B.14) for details on the other optional arguments.

Examples

Example 1

```
myData = import("$BayESHOME/Datasets/dataset4.csv");
myData.constant = ones(rows(myData), 1);
set_pd( year, id, "dataset" = myData);

probit_re( y ~ constant x1 x2 x3 x4 );
```

Example 2

```
myData = import("$BayESHOME/Datasets/dataset4.csv");
myData.constant = ones(rows(myData), 1);
set_pd( year, id, "dataset" = myData);

myModel = probit_re( y ~ constant x1 x2 x3 x4,
    "m"=ones(5,1), "P"=0.1*eye(5,5), "a_omega"=0.1, "b_omega"=0.01,
    "burnin"=10000, "draws"=40000, "thin"=4, "chains"=2,
    "logML_CJ" = true );

diagnostics("model"=myModel);

kden(myModel.x3, "title" = "\beta3 from the Probit model");

margeff_mean = mfx("point"="mean","model"=myModel,"type"=1);
margeff_mean = mfx("point"="mean","model"=myModel,"type"=2);

predict("type"=1, "prefix"=marg_);
predict("type"=2, "prefix"=cond_);
```

6.4 Random-effects binary Logit

Mathematical representation

$$\text{Prob}\left(y_{it}=1\right)=\Lambda\left(\alpha_i+\mathbf{x}_{it}'\boldsymbol{\beta}\right), \qquad \alpha_i\sim \text{N}\left(0,\tfrac{1}{\omega}\right) \tag{6.7}$$

- the model is estimated using observations from N groups, each group observed for T_i periods (balanced or unbalanced panels); the total number of observations is $\sum_{i=1}^{N} T_i$
- y_{it} is the value of the dependent variable for group i, observed in period t and it can take two values: 0 and 1
- $\mathbf{x}_{it}$ is a $K\times 1$ vector that stores the values of the K independent variables for group i, observed in period t
- $\boldsymbol{\beta}$ is a $K\times 1$ vector of parameters
- α_i is the group-specific error term for group i
- ω is the precision of the group-specific error term: $\sigma_\alpha^2=\frac{1}{\omega}$
- $\Lambda\left(\cdot\right)$ is the standard-Logistic cdf

An equivalent representation uses the latent variable y_{it}^*:

$$\begin{aligned} y_{it}^* &= \alpha_i+\mathbf{x}_{it}'\boldsymbol{\beta}+\varepsilon_{it}, \qquad \varepsilon_{it}\sim \text{Logistic}\left(0,1\right), \qquad \alpha_i\sim \text{N}\left(0,\tfrac{1}{\omega}\right), \\ y_{it} &= \begin{cases} 1 & \text{if} \quad y_{it}^*>0 \\ 0 & \text{if} \quad y_{it}^*\leq 0 \end{cases} \end{aligned} \tag{6.8}$$

The mean of the distribution of the α_is is restricted to zero and, therefore, these are simply group-specific errors terms. However, including a constant term in the set of independent variables is valid and leads to a specification equivalent to one where the group effects are draws from a normal distribution with mean equal to the parameter associated with the constant term and precision ω.

Priors

Parameter	Probability density function	Default hyperparameters
$\boldsymbol{\beta}$	$\text{p}\left(\boldsymbol{\beta}\right)=\frac{\left\|\mathbf{P}\right\|^{1/2}}{(2\pi)^{K/2}}\exp\left\{-\frac{1}{2}\left(\boldsymbol{\beta}-\mathbf{m}\right)'\mathbf{P}\left(\boldsymbol{\beta}-\mathbf{m}\right)\right\}$	$\mathbf{m}=\mathbf{0}_K$, $\mathbf{P}=0.001\cdot\mathbf{I}_K$
ω	$\text{p}\left(\omega\right)=\frac{b_\omega^{a_\omega}}{\Gamma(a_\omega)}\omega^{a_\omega-1}e^{-\omega b_\omega}$	$a_\omega=0.01$, $b_\omega=0.001$

Syntax

```
[<model name> = ] logit_re( y ~ x1 x2 ... xK [, <options> ] );
```

where:

- `y` is the dependent variable name, as it appears in the dataset used for estimation
- `x1 x2 ... xK` is a list of the K independent variable names, as they appear in the dataset used for estimation; when a constant term is to be included in the model, this must be requested explicitly

The dependent variable, `y`, in the dataset used for estimation must contain only two values: 0 and 1 (with 1 indicating "success"). Observations with missing values in `y` are dropped during estimation, but if a numerical value other than 0 and 1 is encountered, then an error is produced.

BayES automatically drops from the sample used for estimation groups which are observed only once. This is because for these groups the group effect (α_i) cannot be distinguished from the error term (ε_{it}).

The optional arguments for the random-effects binary Logit model are:[4]

Gibbs parameters	
`"chains"`	number of chains to run in parallel (positive integer); the default value is 1
`"burnin"`	number of burn-in draws per chain (positive integer); the default value is 10000
`"draws"`	number of retained draws per chain (positive integer); the default value is 20000
`"thin"`	value of the thinning parameter (positive integer); the default value is 1
`"seed"`	value of the seed for the random-number generator (positive integer); the default value is 42
Hyperparameters	
`"m"`	mean vector of the prior for $\boldsymbol{\beta}$ ($K \times 1$ vector); the default value is $\mathbf{0}_K$
`"P"`	precision matrix of the prior for $\boldsymbol{\beta}$ ($K \times K$ symmetric and positive-definite matrix); the default value is $0.001 \cdot \mathbf{I}_K$
`"a_omega"`	shape parameter of the prior for ω (positive number); the default value is 0.01
`"b_omega"`	rate parameter of the prior for ω (positive number); the default value is 0.001
Dataset and log-marginal likelihood	
`"dataset"`	the id value of the dataset that will be used for estimation; the default value is the first dataset in memory (in alphabetical order)
`"logML_CJ"`	boolean indicating whether the Chib (1995)/Chib & Jeliazkov (2001) approximation to the log-marginal likelihood should be calculated (**true**|**false**); the default value is **false**

Reported Parameters

$\boldsymbol{\beta}$	`variable_name`	vector of parameters associated with the independent variables
ω	`omega`	precision parameter of the group-specific error term, α_i
σ_α	`sigma_alpha`	standard deviation of the group-specific error term: $\sigma_\alpha = 1/\omega^{1/2}$

Stored values and post-estimation analysis

If a left-hand-side id value is provided when a random-effects binary Logit model is created, then the following results are saved in the model item and are accessible via the '.' operator:

`Samples`	a matrix containing the draws from the posterior of $\boldsymbol{\beta}$ and ω
`x1,...,xK`	vectors containing the draws from the posterior of the parameters associated with variables `x1,...,xK` (the names of these vectors are the names of the variables that were included in the right-hand side of the model)
`omega`	vector containing the draws from the posterior of ω
`logML`	the Lewis & Raftery (1997) approximation of the log-marginal likelihood
`logML_CJ`	the Chib (1995)/Chib & Jeliazkov (2001) approximation to the log-marginal likelihood; this is available only if the model was estimated with the `"logML_CJ"=`**true** option

[4] Optional arguments are always given in option-value pairs (eg. `"chains"=3`).

`alpha_i`	$N \times 1$ vector that stores the group-specific errors; the values in this vector are not guaranteed to be in the same order as the order in which the groups appear in the dataset used for estimation; use the `store()` function to associate the values in `alpha_i` with the observations in the dataset
`nchains`	the number of chains that were used to estimate the model
`nburnin`	the number of burn-in draws per chain that were used when estimating the model
`ndraws`	the total number of retained draws from the posterior (=`chains` · `draws`)
`nthin`	value of the thinning parameter that was used when estimating the model
`nseed`	value of the seed for the random-number generator that was used when estimating the model

Additionally, the following functions are available for post-estimation analysis (see section B.14):

- `diagnostics()`
- `pmp()`
- `mfx()`
- `test()`
- `store()`
- `predict()`

The random-effects binary Logit model uses the `store()` function to associate the group effects (`alpha_i`) with specific observations and store their values in the dataset used for estimation. The generic syntax for a statement involving the `store()` function after estimation of a random-effects binary Logit model is:

```
store( alpha_i , <new variable name>, ["model"=<model name>] );
```

The random-effects binary Logit model uses the `mfx()` function to calculate and report the marginal effects of the independent variables on the probability of success. There are two types of marginal effects which can be requested by setting the `"type"` argument of the `mfx()` function equal to `1` or `2`:

1. when `"type"=1` the marginal effects are calculated marginally with respect to the group effects.

2. when `"type"=2` the marginal effects are calculated conditionally on the group-effects being equal to zero (the expected value of the group effects, when treated as group-specific errors).

The generic syntax for a statement involving the `mfx()` function after estimation of a random-effects binary Logit model is:

```
mfx( ["type"=1] [, "point"=<point of calculation>] [, "model"=<model name>] );
```

and:

```
mfx( "type"=2 [, "point"=<point of calculation>] [, "model"=<model name>] );
```

for calculating these two types of marginal effects. The default value of the `"type"` option is `1`. See the general documentation of the `mfx()` function (section B.14) for details on the other optional arguments.

The random-effects binary Logit model uses the `predict()` function to generate predictions of the probability of success. There are two types of predictions which can be requested by setting the `"type"` argument of the `mfx()` function equal to `1` or `2`:

1. when `"type"=1` the predictions are generated marginally with respect to the group effects.

2. when `"type"=2` the predictions are generated conditionally on the group-effects being equal to zero (the expected value of the group effects, when treated as group-specific errors).

The generic syntax for a statement involving the `predict()` function after estimation of a random-effects binary Logit model is:

```
[<id value>] = predict( ["type"=1] [, "point"=<point of calculation>] [,"model"=<model
    name>] [, "stats"=true|false] [, "prefix"=<prefix for new variable name>] );
```

and:

```
[<id value>] = predict( "type"=2 [, "point"=<point of calculation>] [,"model"=<model
    name>] [, "stats"=true|false] [, "prefix"=<prefix for new variable name>] );
```

for generating these two types of predictions effects. The default value of the `"type"` option is `1`. See the general documentation of the `predict()` function (section B.14) for details on the other optional arguments.

Examples

Example 1

```
myData = import("$BayESHOME/Datasets/dataset4.csv");
myData.constant = ones(rows(myData), 1);
set_pd( year, id, "dataset" = myData);

logit_re( y ~ constant x1 x2 x3 x4 );
```

Example 2

```
myData = import("$BayESHOME/Datasets/dataset4.csv");
myData.constant = ones(rows(myData), 1);
set_pd( year, id, "dataset" = myData);

myModel = logit_re( y ~ constant x1 x2 x3 x4,
    "m"=ones(5,1), "P"=0.1*eye(5,5), "a_omega"=0.1, "b_omega"=0.01,
    "burnin"=10000, "draws"=40000, "thin"=4, "chains"=2,
    "logML_CJ" = true );

diagnostics("model"=myModel);

kden(myModel.x3, "title" = "\beta3 from the Logit model");

margeff_mean = mfx("point"="mean","model"=myModel,"type"=1);
margeff_mean = mfx("point"="mean","model"=myModel,"type"=2);

predict("type"=1, "prefix"=marg_);
predict("type"=2, "prefix"=cond_);
```

6.5 Multinomial Probit

Mathematical representation

$$\begin{aligned} p_{mi} &= \text{Prob}\left(y_i = m|\mathbf{x}_i'\boldsymbol{\beta}_m\right), && m = 1,\ldots,M \\ p_{0i} &= 1 - \sum_{m=1}^{M} p_{mi}, && \text{with additional structure on each } p_{mi} \end{aligned} \tag{6.9}$$

- the model is estimated using N observations
- y_i is the value of the dependent variable for observation i and it can take $M+1$ values: 0, 1, ..., M
- $\mathbf{x}_i$ is a $K\times 1$ vector that stores the values of the K independent variables for observation i
- $\boldsymbol{\beta}_m$ is a $K\times 1$ vector of parameters for alternative m, $m = 1, 2, \ldots, M$

The latent-variable representation of the multinomial Probit is:

$$\begin{array}{rcl} y_{1i}^* &=& \mathbf{x}_i'\boldsymbol{\beta}_1 + \varepsilon_{1i} \\ y_{2i}^* &=& \mathbf{x}_i'\boldsymbol{\beta}_2 + \varepsilon_{2i} \\ \vdots & & \vdots \\ y_{Mi}^* &=& \mathbf{x}_i'\boldsymbol{\beta}_M + \varepsilon_{Mi} \end{array} \qquad y_i = \begin{cases} 0 & \text{if} \quad \max_j \left\{y_{ji}^*\right\} \leq 0 \\ 1 & \text{if} \quad \max_j \left\{y_{ji}^*\right\} = y_{1i}^* \quad \text{and} \quad y_{1i}^* > 0 \\ \vdots & \vdots \quad\quad\quad\quad \vdots \quad\quad\quad \vdots \quad\quad\quad \vdots \\ M & \text{if} \quad \max_j \left\{y_{ji}^*\right\} = y_{Mi}^* \quad \text{and} \quad y_{Mi}^* > 0 \end{cases} \tag{6.10}$$

Let:

$$\underset{(MK)\times 1}{\boldsymbol{\beta}} = \begin{bmatrix} \boldsymbol{\beta}_1 \\ \boldsymbol{\beta}_2 \\ \vdots \\ \boldsymbol{\beta}_M \end{bmatrix} \quad \text{and} \quad \underset{M\times 1}{\boldsymbol{\varepsilon}_i} = \begin{bmatrix} \varepsilon_{1i} \\ \varepsilon_{2i} \\ \vdots \\ \varepsilon_{Mi} \end{bmatrix}$$

$\boldsymbol{\varepsilon}_i$ is assumed to follow a multivariate-Normal distribution: $\boldsymbol{\varepsilon} \sim \text{N}\left(\mathbf{0}, \boldsymbol{\Omega}^{-1}\right)$. For identification purposes the precision matrix is restricted such that $\text{tr}\left(\boldsymbol{\Omega}\right) = M$.

Priors

Parameter	Probability density function	Default hyperparameters
$\boldsymbol{\beta}$	$\text{p}\left(\boldsymbol{\beta}\right) = \frac{\lvert\mathbf{P}\rvert^{1/2}}{(2\pi)^{MK/2}} \exp\left\{-\frac{1}{2}\left(\boldsymbol{\beta}-\mathbf{m}\right)'\mathbf{P}\left(\boldsymbol{\beta}-\mathbf{m}\right)\right\}$	$\mathbf{m}=\mathbf{0}_{MK}$, $\mathbf{P}=0.001\cdot\mathbf{I}_{MK}$
$\boldsymbol{\Omega}$	$\text{p}\left(\boldsymbol{\Omega}\right) = \frac{\lvert\boldsymbol{\Omega}\rvert^{\frac{n-M-1}{2}} \lvert\mathbf{V}^{-1}\rvert^{n/2}}{2^{nM/2}\Gamma_M\left(\frac{n}{2}\right)} \exp\left\{-\frac{1}{2}\text{tr}\left(\mathbf{V}^{-1}\boldsymbol{\Omega}\right)\right\}$	$n = M^2$, $\mathbf{V} = \frac{100}{M}\cdot\mathbf{I}_M$
Following Burgette & Nordheim (2012), the prior for $\boldsymbol{\Omega}$ is transformed such that $\text{tr}\left(\boldsymbol{\Omega}\right) = M$		

Syntax

```
[<model name> = ] mnprobit( y ~ x1 x2 ... xK [, <options> ] );
```

where:

- `y` is the dependent variable name, as it appears in the dataset used for estimation
- `x1 x2 ... xK` is a list of the K independent variable names, as they appear in the dataset used for estimation; when a constant term is to be included in the model, this must be requested explicitly

The dependent variable, `y`, in the dataset used for estimation must contain only consecutive integer values, with the numbering starting at 0 (base category). Observations with missing values in `y` are dropped during estimation, but if a non-integer numerical value is encountered or if the integer values are not consecutive (for example there are no observations for which $y_i = 1$), then an error is produced.

The optional arguments for the multinomial Probit model are:[5]

`Gibbs parameters`	
`"chains"`	number of chains to run in parallel (positive integer); the default value is 1
`"burnin"`	number of burn-in draws per chain (positive integer); the default value is 10000
`"draws"`	number of retained draws per chain (positive integer); the default value is 20000
`"thin"`	value of the thinning parameter (positive integer); the default value is 1
`"seed"`	value of the seed for the random-number generator (positive integer); the default value is 42
`Hyperparameters`	
`"m"`	mean vector of the prior for $\boldsymbol{\beta}$ ($MK \times 1$ vector); the default value is $\mathbf{0}_{MK}$
`"P"`	precision matrix of the prior for $\boldsymbol{\beta}$ ($MK \times MK$ symmetric and positive-definite matrix); the default value is $0.001 \cdot \mathbf{I}_{MK}$
`"V"`	scale matrix of the prior for Ω ($M \times M$ symmetric and positive-definite matrix); the default value is $\frac{100}{M} \cdot \mathbf{I}_M$
`"n"`	degrees-of-freedom parameter of the prior for $\boldsymbol{\Omega}$ (real number greater than or equal to M); the default value is M^2
`Dataset and log-marginal likelihood`	
`"dataset"`	the id value of the dataset that will be used for estimation; the default value is the first dataset in memory (in alphabetical order)
`"logML_CJ"`	boolean indicating whether the Chib (1995)/Chib & Jeliazkov (2001) approximation to the log-marginal likelihood should be calculated (**true**|**false**); the default value is **false**

Reported Parameters

$\boldsymbol{\beta}$	`variable_name`	vector of parameters associated with the independent variables; these are broken into groups according to the alternative, m, the parameter is associated with

Stored values and post-estimation analysis

If a left-hand-side id value is provided when a multinomial Probit model is created, then the following results are saved in the model item and are accessible via the '.' operator:

`Samples`	a matrix containing the draws from the posterior of $\boldsymbol{\beta}$ (across all alternatives, starting from the first one) and the unique elements of $\boldsymbol{\Omega}$
`y_m$x1,...,` `y_m$xK`	vectors containing the draws from the posterior of the parameters associated with variables `x1,...,xK`, for $m = 1, 2, \ldots, M$ (the names of these vectors are the names of the variables that were included in the right-hand side of the model, prepended by `y_m$`, where `y_m` is the name of the dependent variable followed by an underscore and the index of the alternative; this is done so that the samples on the parameters associated with the same independent variable but for different alternatives can be distinguished)

[5] Optional arguments are always given in option-value pairs (eg. `"chains"=3`).

`Omega_i_j`	vectors containing the draws from the posterior of the unique elements of $\boldsymbol{\Omega}$; because $\boldsymbol{\Omega}$ is symmetric, only $\frac{(M-1)M}{2}+M$ of its elements are stored (instead of all M^2 elements); `i` and `j` index the row and column of $\boldsymbol{\Omega}$, respectively, at which the corresponding element is located
`Omega`	$M \times M$ matrix that stores the posterior mean of $\boldsymbol{\Omega}$; $\boldsymbol{\Omega}$ is restricted such that $\text{tr}\left(\boldsymbol{\Omega}^{-1}\right) = M$
`logML`	the Lewis & Raftery (1997) approximation of the log-marginal likelihood
`logML_CJ`	the Chib (1995)/Chib & Jeliazkov (2001) approximation to the log-marginal likelihood; this is available only if the model was estimated with the `"logML_CJ"=true` option
`nchains`	the number of chains that were used to estimate the model
`nburnin`	the number of burn-in draws per chain that were used when estimating the model
`ndraws`	the total number of retained draws from the posterior (=`chains` · `draws`)
`nthin`	value of the thinning parameter that was used when estimating the model
`nseed`	value of the seed for the random-number generator that was used when estimating the model

Additionally, the following functions are available for post-estimation analysis (see section B.14):

- `diagnostics()`
- `pmp()`
- `predict()`
- `test()`
- `mfx()`

The multinomial Probit model uses the `mfx()` function to calculate and report the marginal effects of the independent variables on the probability of each outcome, m, occurring. Because the model calculates only one type of marginal effects, the only valid value for the `"type"` option is `1`. The generic syntax for a statement involving the `mfx()` function after estimation of a multinomial Probit model is:

```
mfx( ["type"=1] [, "point"=<point of calculation>] [, "model"=<model name>] );
```

See the general documentation of the `mfx()` function (section B.14) for details on the other optional arguments.

Although BayES can calculate marginal effects for the multinomial Probit model at each observation, the calculations may take an excessive amount of time to complete. This is because the GHK simulator needs to be invoked at every observed data point, each time using all draws from the posterior, thus leading to an immense number of computations.

The multinomial Probit model uses the `predict()` function to generate predictions of the probability each of the M+1 outcomes occuring. Because the model generates only one type of predictions, the only valid value for the `"type"` option is `1`. The generic syntax for a statement involving the `predict()` function after estimation of a multinomial Probit model is:

```
[<id value>] = predict( ["type"=1] [, "point"=<point of calculation>] [,"model"=<model
    name>] [, "stats"=true|false] [, "prefix"=<prefix for new variable name>] );
```

See the general documentation of the `predict()` function (section B.14) for details on the other optional arguments.

Although BayES can generate summary statistics of the predictions for the multinomial Probit model at each observation, the calculations may take an excessive amount of time to complete. This is because the GHK simulator needs to be invoked at every observed data point, each time using all draws from the posterior, thus leading to an immense number of computations.

Examples

Example 1

```
myData = import("$BayESHOME/Datasets/dataset7.csv");
myData.constant = ones(rows(myData), 1);

mnprobit( y ~ constant x1 x2 x3 x4 );
```

Example 2

```
myData = import("$BayESHOME/Datasets/dataset7.csv");
myData.constant = ones(rows(myData), 1);

myModel = mnprobit( y ~ constant x1 x2 x3 x4,
    "m"=ones(2*5,1), "P"=0.1*eye(2*5,2*5),
    "burnin"=20000, "draws"=40000, "thin"=4, "chains"=2,
    "logML_CJ" = true );

diagnostics("model"=myModel);

kden(myModel.y_2$x3, "title" = "\beta3 for the 2nd alternative");

margeff_mean = mfx("point"="mean","model"=myModel);
margeff_median = mfx("point"="median","model"=myModel);
margeff_atx = mfx("point"=[1,1,0.5,2,0],"model"=myModel);

predict();
```

6.6 Multinomial Logit

Mathematical representation

$$\begin{aligned} p_{mi} &= \text{Prob}\left(y_i = m | \mathbf{x}_i' \boldsymbol{\beta}_m\right), && m = 1, \ldots, M \\ p_{0i} &= 1 - \sum_{m=1}^{M} p_{mi}, && \text{with additional structure on each } p_{mi} \end{aligned} \tag{6.11}$$

- the model is estimated using N observations
- y_i is the value of the dependent variable for observation i and it can take $M+1$ values: 0, 1, ..., M
- $\mathbf{x}_i$ is a $K \times 1$ vector that stores the values of the K independent variables for observation i
- $\boldsymbol{\beta}_m$ is a $K \times 1$ vector of parameters for alternative m, $m = 1, 2, \ldots, M$

The latent-variable representation of the multinomial Logit is:

$$\begin{array}{rcl} y_{1i}^* &=& \mathbf{x}_i'\boldsymbol{\beta}_1 + \varepsilon_{1i} \\ y_{2i}^* &=& \mathbf{x}_i'\boldsymbol{\beta}_2 + \varepsilon_{2i} \\ \vdots & \vdots & \vdots \\ y_{Mi}^* &=& \mathbf{x}_i'\boldsymbol{\beta}_M + \varepsilon_{Mi} \end{array} \qquad y_i = \begin{cases} 0 & \text{if} \quad \max_j \left\{y_{ji}^*\right\} \leq 0 \\ 1 & \text{if} \quad \max_j \left\{y_{ji}^*\right\} = y_{1i}^* \quad \text{and} \quad y_{1i}^* > 0 \\ \vdots & \vdots \quad \vdots \quad \vdots \quad \vdots \quad \vdots \\ M & \text{if} \quad \max_j \left\{y_{ji}^*\right\} = y_{Mi}^* \quad \text{and} \quad y_{Mi}^* > 0 \end{cases} \tag{6.12}$$

Let:

$$\underset{(MK)\times 1}{\boldsymbol{\beta}} = \begin{bmatrix} \boldsymbol{\beta}_1 \\ \boldsymbol{\beta}_2 \\ \vdots \\ \boldsymbol{\beta}_M \end{bmatrix} \quad \text{and} \quad \underset{M\times 1}{\boldsymbol{\varepsilon}_i} = \begin{bmatrix} \varepsilon_{1i} \\ \varepsilon_{2i} \\ \vdots \\ \varepsilon_{Mi} \end{bmatrix}$$

The ε_{mi}s are assumed to follow a multivariate Logistic distribution (Malik & Abraham, 1973) with mean zero.

Priors

Parameter	Probability density function	Default hyperparameters
$\boldsymbol{\beta}$	$\text{p}(\boldsymbol{\beta}) = \frac{\lvert\mathbf{P}\rvert^{1/2}}{(2\pi)^{MK/2}} \exp\left\{-\frac{1}{2}(\boldsymbol{\beta}-\mathbf{m})'\mathbf{P}(\boldsymbol{\beta}-\mathbf{m})\right\}$	$\mathbf{m}=\mathbf{0}_{MK}$, $\mathbf{P}=0.001\cdot\mathbf{I}_{MK}$

Syntax

```
[<model name> = ] mnlogit( y ~ x1 x2 ... xK [, <options> ] );
```

where:

- `y` is the dependent variable name, as it appears in the dataset used for estimation
- `x1 x2 ... xK` is a list of the K independent variable names, as they appear in the dataset used for estimation; when a constant term is to be included in the model, this must be requested explicitly

The dependent variable, `y`, in the dataset used for estimation must contain only consecutive integer values, with the numbering starting at 0 (base category). Observations with missing values in `y` are dropped during estimation, but if a non-integer numerical value is encountered or if the integer values are not consecutive (for example there are no observations for which $y_i = 1$), then an error is produced.

The optional arguments for the multinomial Logit model are:[6]

Gibbs parameters	
"chains"	number of chains to run in parallel (positive integer); the default value is 1
"burnin"	number of burn-in draws per chain (positive integer); the default value is 10000
"draws"	number of retained draws per chain (positive integer); the default value is 20000
"thin"	value of the thinning parameter (positive integer); the default value is 1
"seed"	value of the seed for the random-number generator (positive integer); the default value is 42
Hyperparameters	
"m"	mean vector of the prior for $\boldsymbol{\beta}$ ($MK \times 1$ vector); the default value is $\mathbf{0}_{MK}$
"P"	precision matrix of the prior for $\boldsymbol{\beta}$ ($MK \times MK$ symmetric and positive-definite matrix); the default value is $0.001 \cdot \mathbf{I}_{MK}$
Dataset and log-marginal likelihood	
"dataset"	the id value of the dataset that will be used for estimation; the default value is the first dataset in memory (in alphabetical order)
"logML_CJ"	boolean indicating whether the Chib (1995)/Chib & Jeliazkov (2001) approximation to the log-marginal likelihood should be calculated (**true**|**false**); the default value is **false**

Reported Parameters

$\boldsymbol{\beta}$	variable_name	vector of parameters associated with the independent variables; these are broken into groups according to the alternative, m, the parameter is associated with

Stored values and post-estimation analysis

If a left-hand-side id value is provided when a multinomial Logit model is created, then the following results are saved in the model item and are accessible via the '.' operator:

Samples	a matrix containing the draws from the posterior of $\boldsymbol{\beta}$ (across all alternatives, starting from the first one)
y_m$x1,..., y_m$xK	vectors containing the draws from the posterior of the parameters associated with variables x1,...,xK, for $m = 1, 2, \ldots, M$ (the names of these vectors are the names of the variables that were included in the right-hand side of the model, prepended by y_m$, where y_m is the name of the dependent variable followed by an underscore and the index of the alternative; this is done so that the samples on the parameters associated with the same independent variable but for different alternatives can be distinguished)
logML	the Lewis & Raftery (1997) approximation of the log-marginal likelihood
logML_CJ	the Chib (1995)/Chib & Jeliazkov (2001) approximation to the log-marginal likelihood; this is available only if the model was estimated with the "logML_CJ"=**true** option
nchains	the number of chains that were used to estimate the model
nburnin	the number of burn-in draws per chain that were used when estimating the model
ndraws	the total number of retained draws from the posterior (=chains · draws)
nthin	value of the thinning parameter that was used when estimating the model
nseed	value of the seed for the random-number generator that was used when estimating the model

[6] Optional arguments are always given in option-value pairs (eg. "chains"=3).

Additionally, the following functions are available for post-estimation analysis (see section B.14):

- diagnostics()
- pmp()
- predict()
- test()
- mfx()

The multinomial Logit model uses the mfx() function to calculate and report the marginal effects of the independent variables on the probability of each outcome, m, occurring. Because the model calculates only one type of marginal effects, the only valid value for the "type" option is 1. The generic syntax for a statement involving the mfx() function after estimation of a multinomial Logit model is:

```
mfx( ["type"=1] [, "point"=<point of calculation>] [, "model"=<model name>] );
```

See the general documentation of the mfx() function (section B.14) for details on the other optional arguments.

The multinomial Logit model uses the predict() function to generate predictions of the probability each of the M+1 outcomes occuring. Because the model generates only one type of predictions, the only valid value for the "type" option is 1. The generic syntax for a statement involving the predict() function after estimation of a multinomial Logit model is:

```
[<id value>] = predict( ["type"=1] [, "point"=<point of calculation>] [,"model"=<model
    name>] [, "stats"=true|false] [, "prefix"=<prefix for new variable name>] );
```

See the general documentation of the predict() function (section B.14) for details on the other optional arguments.

Examples

Example 1

```
myData = import("$BayESHOME/Datasets/dataset7.csv");
myData.constant = ones(rows(myData), 1);

mnlogit( y ~ constant x1 x2 x3 x4 );
```

Example 2

```
myData = import("$BayESHOME/Datasets/dataset7.csv");
myData.constant = ones(rows(myData), 1);

myModel = mnlogit( y ~ constant x1 x2 x3 x4,
    "m"=ones(2*5,1), "P"=0.1*eye(2*5,2*5),
    "burnin"=10000, "draws"=40000, "thin"=4, "chains"=2,
    "logML_CJ" = true );

diagnostics("model"=myModel);

kden(myModel.y_2$x3, "title" = "\beta3 for the 2nd alternative");

margeff_mean = mfx("point"="mean","model"=myModel);
margeff_median = mfx("point"="median","model"=myModel);
margeff_eachpoint = mfx("point"="x_i","model"=myModel);
margeff_atx = mfx("point"=[1,1,0.5,2,0],"model"=myModel);

predict();
```

6.7 Conditional Probit

Mathematical representation

$$\begin{aligned} &p_{mi} = \text{Prob}\left(y_i = m|\mathbf{z}'_{mi}\boldsymbol{\delta} + \mathbf{x}'_i\boldsymbol{\beta}_m\right), && m = 1, \ldots, M \\ &p_{0i} = 1 - \sum_{m=1}^{M} p_{mi}, && \text{with additional structure on each } p_{mi} \end{aligned} \tag{6.13}$$

- the model is estimated using N observations
- y_i is the value of the dependent variable for observation i and it can take $M+1$ values: 0, 1, ..., M
- $\mathbf{z}_{mi}$ is a $K\times 1$ vector that stores the values of the K independent variables for observation i, which are specific to alternative m
- $\mathbf{x}_i$ is an $L\times 1$ vector that stores the values of the L independent variables for observation i, which are common to all alternatives
- $\boldsymbol{\delta}$ is a $K\times 1$ vector of parameters, common to all alternatives
- $\boldsymbol{\beta}_m$ is an $L\times 1$ vector of parameters for alternative m, $m = 1, 2, \ldots, M$
- there are $J = K + M \cdot L$ slope parameters to be estimated

The latent-variable representation of the conditional Probit is:

$$\begin{aligned} y^*_{1i} &= \mathbf{z}'_{1i}\boldsymbol{\delta} + \mathbf{x}'_i\boldsymbol{\beta}_1 + \varepsilon_{1i} \\ y^*_{2i} &= \mathbf{z}'_{2i}\boldsymbol{\delta} + \mathbf{x}'_i\boldsymbol{\beta}_2 + \varepsilon_{2i} \\ &\vdots \qquad\qquad \vdots \qquad\qquad \vdots \\ y^*_{Mi} &= \mathbf{z}'_{Mi}\boldsymbol{\delta} + \mathbf{x}'_i\boldsymbol{\beta}_M + \varepsilon_{Mi} \end{aligned} \qquad y_i = \begin{cases} 0 & \text{if} \quad \max_j \left\{y^*_{ji}\right\} \leq 0 \\ 1 & \text{if} \quad \max_j \left\{y^*_{ji}\right\} = y^*_{1i} \quad \text{and} \quad y^*_{1i} > 0 \\ \vdots & \vdots \qquad\qquad \vdots \qquad\qquad \vdots \\ M & \text{if} \quad \max_j \left\{y^*_{ji}\right\} = y^*_{Mi} \quad \text{and} \quad y^*_{Mi} > 0 \end{cases} \tag{6.14}$$

Let:

$$\underset{J\times 1}{\boldsymbol{\beta}} = \begin{bmatrix} \boldsymbol{\delta} \\ \boldsymbol{\beta}_1 \\ \boldsymbol{\beta}_2 \\ \vdots \\ \boldsymbol{\beta}_M \end{bmatrix} \quad \text{and} \quad \underset{M\times 1}{\boldsymbol{\varepsilon}_i} = \begin{bmatrix} \varepsilon_{1i} \\ \varepsilon_{2i} \\ \vdots \\ \varepsilon_{Mi} \end{bmatrix}$$

$\boldsymbol{\varepsilon}_i$ is assumed to follow a multivariate-Normal distribution: $\boldsymbol{\varepsilon} \sim \text{N}\left(\mathbf{0}, \boldsymbol{\Omega}^{-1}\right)$. For identification purposes the precision matrix is restricted such that $\text{tr}\left(\boldsymbol{\Omega}\right) = M$.

Priors

Parameter	Probability density function	Default hyperparameters
$\boldsymbol{\beta}$	$\text{p}\left(\boldsymbol{\beta}\right) = \frac{\lvert\mathbf{P}\rvert^{1/2}}{(2\pi)^{J/2}} \exp\left\{-\frac{1}{2}\left(\boldsymbol{\beta}-\mathbf{m}\right)'\mathbf{P}\left(\boldsymbol{\beta}-\mathbf{m}\right)\right\}$	$\mathbf{m}=\mathbf{0}_J$, $\mathbf{P}=0.001\cdot\mathbf{I}_J$
$\boldsymbol{\Omega}$	$\text{p}\left(\boldsymbol{\Omega}\right) = \frac{\lvert\boldsymbol{\Omega}\rvert^{\frac{n-M-1}{2}} \lvert\mathbf{V}^{-1}\rvert^{n/2}}{2^{nM/2}\Gamma_M\left(\frac{n}{2}\right)} \exp\left\{-\frac{1}{2}\text{tr}\left(\mathbf{V}^{-1}\boldsymbol{\Omega}\right)\right\}$	$n = M^2$, $\mathbf{V} = \frac{100}{M}\cdot\mathbf{I}_M$
Following Burgette & Nordheim (2012), the prior for $\boldsymbol{\Omega}$ is transformed such that $\text{tr}\left(\boldsymbol{\Omega}\right) = M$		

Syntax

```
[<model name> = ] cprobit( y ~ z1 z2 ... zK [| x1 x2 ... xL ] [, <options> ] );
```

where:

- `y` is the dependent variable name, as it appears in the dataset used for estimation
- `z1 z2 ... zK` is a list of the names, as they appear in the dataset used for estimation except for the alternative index, of the independent variables which are associated with

variables that vary by alternative; for each 'zk' variable, the dataset must contain $M+1$ variables whose names start by zk and followed by an underscore and the index of the alternative to which the variable corresponds (counting starting at zero)

- x1 x2 ... xL is a list of the names, as they appear in the dataset used for estimation, of the independent variables which are common to all alternatives; when a constant term is to be included in the model, this must be requested explicitly

The dependent variable, y, in the dataset used for estimation must contain only consecutive integer values, with the numbering starting at 0 (base category). Observations with missing values in y are dropped during estimation, but if a non-integer numerical value is encountered or if the integer values are not consecutive (for example there are no observations for which $y_i = 1$), then an error is produced.

The optional arguments for the conditional Probit model are:[7]

Gibbs parameters	
"chains"	number of chains to run in parallel (positive integer); the default value is 1
"burnin"	number of burn-in draws per chain (positive integer); the default value is 10000
"draws"	number of retained draws per chain (positive integer); the default value is 20000
"thin"	value of the thinning parameter (positive integer); the default value is 1
"seed"	value of the seed for the random-number generator (positive integer); the default value is 42
Hyperparameters	
"m"	mean vector of the prior for $\boldsymbol{\beta}$ ($J \times 1$ vector); the default value is $\mathbf{0}_J$
"P"	precision matrix of the prior for $\boldsymbol{\beta}$ ($J \times J$ symmetric and positive-definite matrix); the default value is $0.001 \cdot \mathbf{I}_J$
"V"	scale matrix of the prior for Ω ($M \times M$ symmetric and positive-definite matrix); the default value is $\frac{100}{M} \cdot \mathbf{I}_M$
"n"	degrees-of-freedom parameter of the prior for $\boldsymbol{\Omega}$ (real number greater than or equal to M); the default value is M^2
Dataset and log-marginal likelihood	
"dataset"	the id value of the dataset that will be used for estimation; the default value is the first dataset in memory (in alphabetical order)
"logML_CJ"	boolean indicating whether the Chib (1995)/Chib & Jeliazkov (2001) approximation to the log-marginal likelihood should be calculated (**true**|**false**); the default value is **false**

Reported Parameters

$\boldsymbol{\delta}$	variable_name	vector of parameters associated with the independent variables that vary by alternative
$\boldsymbol{\beta}$	variable_name	vector of parameters associated with the independent variables which are common to all alternatives; these are broken into groups according to the alternative, m, the parameter is associated with

Stored values and post-estimation analysis

If a left-hand-side id value is provided when a conditional Probit model is created, then the following results are saved in the model item and are accessible via the '.' operator:

[7] Optional arguments are always given in option-value pairs (eg. "chains"=3).

`Samples`	a matrix containing the draws from the posterior of $\boldsymbol{\beta}$ (including $\boldsymbol{\delta}$ and the $\boldsymbol{\beta}$s across all alternatives, starting from the first one) and the unique elements of $\boldsymbol{\Omega}$
`z1,...,zK`	vectors containing the draws from the posterior of the parameters associated with variables that vary by alternative, `z1,...,zK`; (the names of these vectors are the names of the variables as they were included in the right-hand side of the model, excluding the alternative index)
`y_m$x1,...,` `y_m$xK`	vectors containing the draws from the posterior of the parameters associated with variables that are common to all alternatives, `x1,...,xK`, for $m = 1, 2, \ldots, M$ (the names of these vectors are the names of the variables that were included in the right-hand side of the model, prepended by `y_m$`, where `y_m` is the name of the dependent variable followed by an underscore and the index of the alternative; this is done so that the samples on the parameters associated with the same independent variable but for different alternatives can be distinguished)
`Omega_i_j`	vectors containing the draws from the posterior of the unique elements of $\boldsymbol{\Omega}$; because $\boldsymbol{\Omega}$ is symmetric, only $\frac{(M-1)M}{2} + M$ of its elements are stored (instead of all M^2 elements); `i` and `j` index the row and column of $\boldsymbol{\Omega}$, respectively, at which the corresponding element is located
`Omega`	$M \times M$ matrix that stores the posterior mean of $\boldsymbol{\Omega}$; $\boldsymbol{\Omega}$ is restricted such that $\text{tr}\left(\boldsymbol{\Omega}^{-1}\right) = M$
`logML`	the Lewis & Raftery (1997) approximation of the log-marginal likelihood
`logML_CJ`	the Chib (1995)/Chib & Jeliazkov (2001) approximation to the log-marginal likelihood; this is available only if the model was estimated with the `"logML_CJ"=true` option
`nchains`	the number of chains that were used to estimate the model
`nburnin`	the number of burn-in draws per chain that were used when estimating the model
`ndraws`	the total number of retained draws from the posterior (=`chains` · `draws`)
`nthin`	value of the thinning parameter that was used when estimating the model
`nseed`	value of the seed for the random-number generator that was used when estimating the model

Additionally, the following functions are available for post-estimation analysis (see section B.14):

- `diagnostics()`
- `pmp()`
- `predict()`
- `test()`
- `mfx()`

The conditional Probit model uses the `mfx()` function to calculate and report the marginal effects of the independent variables on the probability of each outcome, m, occurring. Because the model calculates only one type of marginal effects, the only valid value for the `"type"` option is `1`. The generic syntax for a statement involving the `mfx()` function after estimation of a conditional Probit model is:

```
mfx( ["type"=1] [, "point"=<point of calculation>] [, "model"=<model name>] );
```

See the general documentation of the `mfx()` function (section B.14) for details on the other optional arguments.

Although BayES can calculate marginal effects for the conditional Probit model at each observation, the calculations may take an excessive amount of time to complete. This is because the GHK simulator needs to be invoked at every observed data point, each time using all draws from the posterior, thus leading to an immense number of computations.

The conditional Probit model uses the `predict()` function to generate predictions of the probability each of the $M+1$ outcomes occuring. Because the model generates only one type of predictions, the only valid value for the `"type"` option is `1`. The generic syntax for a statement involving the `predict()` function after estimation of a conditional Probit model is:

```
[<id value>] = predict( ["type"=1] [, "point"=<point of calculation>] [,"model"=<model
    name>] [, "stats"=true|false] [, "prefix"=<prefix for new variablename>] );
```

See the general documentation of the `predict()` function (section B.14) for details on the other optional arguments.

Although BayES can generate summary statistics of the predictions for the conditional Probit model at each observation, the calculations may take an excessive amount of time to complete. This is because the GHK simulator needs to be invoked at every observed data point, each time using all draws from the posterior, thus leading to an immense number of computations.

Examples

Example 1

```
myData = import("$BayESHOME/Datasets/dataset7.csv");
myData.constant = ones(rows(myData), 1);

cprobit( y ~ z w v | constant );
```

Example 2

```
myData = import("$BayESHOME/Datasets/dataset7.csv");
myData.constant = ones(rows(myData), 1);

myModel = cprobit( y ~ z w v | constant x1 x2 x3 x4,
    "m"=ones(3+2*5,1), "P"=0.01*eye(3+2*5,3+2*5),
    "n"=10, "V"=eye(2,2),
    "burnin"=20000, "draws"=40000, "thin"=4, "chains"=2,
    "logML_CJ" = true );

diagnostics("model"=myModel);

kden(myModel.z, "title" = "\delta1");
kden(myModel.y_2$x3, "title" = "\beta3 for the 2nd alternative");

margeff_mean = mfx("point"="mean","model"=myModel);
margeff_median = mfx("point"="median","model"=myModel);
x_for_mfx = [
    0.0,0.0,0.05,          // z, w, v for the base alternative
    1.0,1.1,0.16,          // z, w, v for the 1st alternative
    1.0,1.0,0.14,          // z, w, v for the 2nd alternative
    1.0,1.0,0.5,2.0,0.0    // x variables
    ];
margeff_atx = mfx("point"=x_for_mfx,"model"=myModel);

predict();
```

6.8 Conditional Logit

Mathematical representation

$$\begin{aligned} p_{mi} &= \text{Prob}\left(y_i = m | \mathbf{z}'_{mi}\boldsymbol{\delta} + \mathbf{x}'_i\boldsymbol{\beta}_m\right), && m = 1, \ldots, M \\ p_{0i} &= 1 - \sum_{m=1}^{M} p_{mi}, && \text{with additional structure on each } p_{mi} \end{aligned} \tag{6.15}$$

- the model is estimated using N observations
- y_i is the value of the dependent variable for observation i and it can take $M+1$ values: 0, 1, ..., M
- $\mathbf{z}_{mi}$ is a $K\times 1$ vector that stores the values of the K independent variables for observation i, which are specific to alternative m
- $\mathbf{x}_i$ is an $L\times 1$ vector that stores the values of the L independent variables for observation i, which are common to all alternatives
- $\boldsymbol{\delta}$ is a $K\times 1$ vector of parameters, common to all alternatives
- $\boldsymbol{\beta}_m$ is an $L\times 1$ vector of parameters for alternative m, $m = 1, 2, \ldots, M$
- there are $J = K + M \cdot L$ parameters to be estimated

The latent-variable representation of the conditional Logit is:

$$\begin{array}{l} y^*_{1i} = \mathbf{z}'_{1i}\boldsymbol{\delta} + \mathbf{x}'_i\boldsymbol{\beta}_1 + \varepsilon_{1i} \\ y^*_{2i} = \mathbf{z}'_{2i}\boldsymbol{\delta} + \mathbf{x}'_i\boldsymbol{\beta}_2 + \varepsilon_{2i} \\ \quad\vdots \qquad\qquad\quad \vdots \qquad\quad \vdots \\ y^*_{Mi} = \mathbf{z}'_{Mi}\boldsymbol{\delta} + \mathbf{x}'_i\boldsymbol{\beta}_M + \varepsilon_{Mi} \end{array} \qquad y_i = \begin{cases} 0 \text{ if } \max_j \left\{y^*_{ji}\right\} \le 0 \\ 1 \text{ if } \max_j \left\{y^*_{ji}\right\} = y^*_{1i} \text{ and } y^*_{1i} > 0 \\ \quad\vdots \quad\vdots \qquad\qquad \vdots \qquad\quad \vdots \qquad\quad \vdots \\ M \text{ if } \max_j \left\{y^*_{ji}\right\} = y^*_{Mi} \text{ and } y^*_{Mi} > 0 \end{cases} \tag{6.16}$$

Let:

$$\underset{J\times 1}{\boldsymbol{\beta}} = \begin{bmatrix} \boldsymbol{\delta} \\ \boldsymbol{\beta}_1 \\ \boldsymbol{\beta}_2 \\ \vdots \\ \boldsymbol{\beta}_M \end{bmatrix} \quad \text{and} \quad \underset{M\times 1}{\boldsymbol{\varepsilon}_i} = \begin{bmatrix} \varepsilon_{1i} \\ \varepsilon_{2i} \\ \vdots \\ \varepsilon_{Mi} \end{bmatrix}$$

The ε_{mi}s are assumed to follow a multivariate Logistic distribution (Malik & Abraham, 1973) with mean zero.

Priors

Parameter	Probability density function	Default hyperparameters
$\boldsymbol{\beta}$	$\text{p}(\boldsymbol{\beta}) = \frac{\lvert\mathbf{P}\rvert^{1/2}}{(2\pi)^{J/2}} \exp\left\{-\frac{1}{2}(\boldsymbol{\beta}-\mathbf{m})'\mathbf{P}(\boldsymbol{\beta}-\mathbf{m})\right\}$	$\mathbf{m}=\mathbf{0}_J$, $\mathbf{P}=0.001\cdot\mathbf{I}_J$

Syntax

```
[<model name> = ] clogit( y ~ z1 z2 ... zK [| x1 x2 ... xL ] [, <options> ] );
```

where:

- `y` is the dependent variable name, as it appears in the dataset used for estimation
- `z1 z2 ... zK` is a list of the names, as they appear in the dataset used for estimation except for the alternative index, of the independent variables which are associated with variables that vary by alternative; for each '`zk`' variable, the dataset must contain $M+1$ variables whose names start by `zk` and followed by an underscore and the index of the alternative to which the variable corresponds (counting starting at zero)

- `x1 x2 ... xL` is a list of the names, as they appear in the dataset used for estimation, of the independent variables which are common to all alternatives; when a constant term is to be included in the model, this must be requested explicitly

The dependent variable, `y`, in the dataset used for estimation must contain only consecutive integer values, with the numbering starting at 0 (base category). Observations with missing values in `y` are dropped during estimation, but if a non-integer numerical value is encountered or if the integer values are not consecutive (for example there are no observations for which $y_i = 1$), then an error is produced.

The optional arguments for the conditional Logit model are:[8]

`Gibbs parameters`	
`"chains"`	number of chains to run in parallel (positive integer); the default value is 1
`"burnin"`	number of burn-in draws per chain (positive integer); the default value is 10000
`"draws"`	number of retained draws per chain (positive integer); the default value is 20000
`"thin"`	value of the thinning parameter (positive integer); the default value is 1
`"seed"`	value of the seed for the random-number generator (positive integer); the default value is 42
`Hyperparameters`	
`"m"`	mean vector of the prior for $\boldsymbol{\beta}$ ($J \times 1$ vector); the default value is $\mathbf{0}_J$
`"P"`	precision matrix of the prior for $\boldsymbol{\beta}$ ($J \times J$ symmetric and positive-definite matrix); the default value is $0.001 \cdot \mathbf{I}_J$
`Dataset and log-marginal likelihood`	
`"dataset"`	the id value of the dataset that will be used for estimation; the default value is the first dataset in memory (in alphabetical order)
`"logML_CJ"`	boolean indicating whether the Chib (1995)/Chib & Jeliazkov (2001) approximation to the log-marginal likelihood should be calculated (**true**|**false**); the default value is **false**

Reported Parameters

$\boldsymbol{\delta}$	`variable_name`	vector of parameters associated with the independent variables that vary by alternative
$\boldsymbol{\beta}$	`variable_name`	vector of parameters associated with the independent variables which are common to all alternatives; these are broken into groups according to the alternative, m, the parameter is associated with

Stored values and post-estimation analysis

If a left-hand-side id value is provided when a conditional Logit model is created, then the following results are saved in the model item and are accessible via the '.' operator:

`Samples`	a matrix containing the draws from the posterior of $\boldsymbol{\beta}$ (including $\boldsymbol{\delta}$ and the $\boldsymbol{\beta}$s across all alternatives, starting from the first one)
`z1,...,zK`	vectors containing the draws from the posterior of the parameters associated with variables that vary by alternative, `z1,...,zK`; (the names of these vectors are the names of the variables as they were included in the right-hand side of the model, excluding the alternative index)

[8] Optional arguments are always given in option-value pairs (eg. `"chains"=3`).

`y_m$x1,...,` `y_m$xK`	vectors containing the draws from the posterior of the parameters associated with variables that are common to all alternatives, `x1,...,xK`, for $m = 1, 2, \ldots, M$ (the names of these vectors are the names of the variables that were included in the right-hand side of the model, prepended by `y_m$`, where `y_m` is the name of the dependent variable followed by an underscore and the index of the alternative; this is done so that the samples on the parameters associated with the same independent variable but for different alternatives can be distinguished)
`logML`	the Lewis & Raftery (1997) approximation of the log-marginal likelihood
`logML_CJ`	the Chib (1995)/Chib & Jeliazkov (2001) approximation to the log-marginal likelihood; this is available only if the model was estimated with the `"logML_CJ"=true` option
`nchains`	the number of chains that were used to estimate the model
`nburnin`	the number of burn-in draws per chain that were used when estimating the model
`ndraws`	the total number of retained draws from the posterior (=`chains` · `draws`)
`nthin`	value of the thinning parameter that was used when estimating the model
`nseed`	value of the seed for the random-number generator that was used when estimating the model

Additionally, the following functions are available for post-estimation analysis (see section B.14):

- `diagnostics()`
- `pmp()`
- `predict()`
- `test()`
- `mfx()`

The conditional Logit model uses the `mfx()` function to calculate and report the marginal effects of the independent variables on the probability of each outcome, m, occurring. Because the model calculates only one type of marginal effects, the only valid value for the `"type"` option is `1`. The generic syntax for a statement involving the `mfx()` function after estimation of a conditional Logit model is:

```
mfx( ["type"=1] [, "point"=<point of calculation>] [, "model"=<model name>] );
```

See the general documentation of the `mfx()` function (section B.14) for details on the other optional arguments.

The conditional Logit model uses the `predict()` function to generate predictions of the probability each of the M+1 outcomes occuring. Because the model generates only one type of predictions, the only valid value for the `"type"` option is `1`. The generic syntax for a statement involving the `predict()` function after estimation of a conditional Logit model is:

```
[<id value>] = predict( ["type"=1] [, "point"=<point of calculation>] [,"model"=<model
    name>] [, "stats"=true|false] [, "prefix"=<prefix for new variable name>] );
```

See the general documentation of the `predict()` function (section B.14) for details on the other optional arguments.

Examples

Example 1

```
myData = import("$BayESHOME/Datasets/dataset7.csv");
myData.constant = ones(rows(myData), 1);

clogit( y ~ z w v | constant );
```

Example 2

```
myData = import("$BayESHOME/Datasets/dataset7.csv");
myData.constant = ones(rows(myData), 1);

myModel = clogit( y ~ z w v | constant x1 x2 x3 x4,
    "m"=ones(3+2*5,1), "P"=0.01*eye(3+2*5,3+2*5),
    "burnin"=20000, "draws"=40000, "thin"=4, "chains"=2,
    "logML_CJ" = true );

diagnostics("model"=myModel);

kden(myModel.z, "title" = "\delta1");
kden(myModel.y_2$x3, "title" = "\beta3 for the 2nd alternative");

margeff_mean = mfx("point"="mean","model"=myModel);
margeff_median = mfx("point"="median","model"=myModel);
margeff_eachpoint = mfx("point"="x_i","model"=myModel);
x_for_mfx = [
    0.0,0.0,0.05,          // z, w, v for the base alternative
    1.0,1.1,0.16,          // z, w, v for the 1st alternative
    1.0,1.0,0.14,          // z, w, v for the 2nd alternative
    1.0,1.0,0.5,2.0,0.0    // x variables
    ];
margeff_atx = mfx("point"=x_for_mfx,"model"=myModel);

predict();
```

6.9 Multivariate Probit

Mathematical representation

$$\begin{array}{rcccl} y_{1i}^* & = & \mathbf{x}_{1i}'\boldsymbol{\beta}_1 & + & \varepsilon_{1i} \\ y_{2i}^* & = & \mathbf{x}_{2i}'\boldsymbol{\beta}_2 & + & \varepsilon_{2i} \\ \vdots & & \vdots & & \vdots \\ y_{Mi}^* & = & \mathbf{x}_{Mi}'\boldsymbol{\beta}_M & + & \varepsilon_{Mi} \end{array} \tag{6.17}$$

$$y_{mi} = \begin{cases} 1 & \text{if} \quad y_{mi}^* > 0 \\ 0 & \text{if} \quad y_{mi}^* \leq 0 \end{cases} \qquad \forall m = 1, 2. \ldots, M$$

where the y_{mi}^*'s are not observed, but, similarly to binary Probit and Logit models, their signs are determined by the corresponding observed y_{mi}s.

- the model is estimated using N observations
- y_{mi} is the value of equation m's dependent variable for observation i and it can take two values: 0 and 1, $\forall m = 1, 2, \ldots, M$
- $\mathbf{x}_{mi}$ is a $K_m \times 1$ vector that stores the values of the K_m independent variables for observation i, as they appear in equation m
- the same independent variable can appear in multiple equations, associated with different coefficients
- $\boldsymbol{\beta}_m$ is a $K_m \times 1$ vector of parameters associated with equation m's independent variables
- in total, there are $K = \sum_{m=1}^{M} K_m$ β parameters to be estimated
- the M error terms jointly follow a multivariate Normal distribution with mean $\mathbf{0}$ and covariance matrix $\boldsymbol{\Sigma}$

A more compact representation of the model is:

$$\mathbf{y}_i^* = \mathbf{X}_i\boldsymbol{\beta} + \boldsymbol{\varepsilon}_i, \qquad \boldsymbol{\varepsilon}_i \sim \mathrm{N}(\mathbf{0}, \boldsymbol{\Sigma})$$

where:

$$\underset{M\times 1}{\mathbf{y}_i^*} = \begin{bmatrix} y_{1i} \\ y_{2i} \\ \vdots \\ y_{Mi} \end{bmatrix}, \qquad \underset{M\times K}{\mathbf{X}_i} = \begin{bmatrix} \mathbf{x}_{1i}' & \mathbf{0} & \ldots & \mathbf{0} \\ \mathbf{0} & \mathbf{x}_{2i}' & \ldots & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{0} & \mathbf{0} & \ldots & \mathbf{x}_{Mi}' \end{bmatrix}, \qquad \underset{K\times 1}{\boldsymbol{\beta}} = \begin{bmatrix} \boldsymbol{\beta}_1 \\ \boldsymbol{\beta}_2 \\ \vdots \\ \boldsymbol{\beta}_M \end{bmatrix}, \qquad \underset{M\times 1}{\boldsymbol{\varepsilon}_i} = \begin{bmatrix} \varepsilon_{1i} \\ \varepsilon_{2i} \\ \vdots \\ \varepsilon_{Mi} \end{bmatrix}$$

and $\boldsymbol{\Sigma}$ being the covariance matrix of $\boldsymbol{\varepsilon}_i$: $\boldsymbol{\Sigma} \equiv \boldsymbol{\Omega}^{-1}$. For identification purposes, $\boldsymbol{\Sigma}$ is normalized such that its diagonal elements are equal to one. Thus, $\boldsymbol{\Sigma}$ is, in fact, a correlation matrix.

Priors

Parameter	Probability density function	Default hyperparameters
$\boldsymbol{\beta}$	$\mathrm{p}(\boldsymbol{\beta}) = \frac{\lvert\mathbf{P}\rvert^{1/2}}{(2\pi)^{K/2}} \exp\left\{-\frac{1}{2}(\boldsymbol{\beta}-\mathbf{m})' \mathbf{P} (\boldsymbol{\beta}-\mathbf{m})\right\}$	$\mathbf{m} = \mathbf{0}_K$, $\mathbf{P} = 0.001 \cdot \mathbf{I}_K$
$\boldsymbol{\Sigma}$	$\mathrm{p}\left(\breve{\boldsymbol{\Sigma}}\right) = \frac{\lvert\breve{\boldsymbol{\Sigma}}\rvert^{-\frac{n+M+1}{2}} \lvert\mathbf{S}\rvert^{n/2}}{2^{nM/2}\Gamma_M\left(\frac{n}{2}\right)} \exp\left\{-\frac{1}{2}\mathrm{tr}\left(\mathbf{S}\breve{\boldsymbol{\Sigma}}^{-1}\right)\right\}$	$n = M^2$, $\mathbf{V} = \frac{100}{M} \cdot \mathbf{I}_M$
The prior for $\breve{\boldsymbol{\Sigma}}$ is transformed such that $\mathrm{diag}(\boldsymbol{\Sigma}) = \mathbf{1}_M$ and $\boldsymbol{\Sigma}$ is a proper correlation matrix.		

An inverse-Wishart prior is used for a positive-definite, but otherwise unrestricted, matrix, $\breve{\boldsymbol{\Sigma}}$. This prior is then internally transformed to a prior for $\boldsymbol{\Sigma} = \mathbf{D}\breve{\boldsymbol{\Sigma}}\mathbf{D}$, where $\mathbf{D}$ is an $M \times M$ diagonal matrix constructed by taking the inverse of the square root of the diagonal elements of $\breve{\boldsymbol{\Sigma}}$: $\mathbf{D} = \text{diag}\left(\begin{bmatrix}\breve{\sigma}_{11}^{-1/2} & \breve{\sigma}_{22}^{-1/2} & \cdots & \breve{\sigma}_{MM}^{-1/2}\end{bmatrix}\right)$.

The magnitude of the elements of the scale matrix, $\mathbf{S}$, in the prior for $\breve{\boldsymbol{\Sigma}}$ affects the prior for $\boldsymbol{\Sigma}$ only if $\mathbf{S}$ is not diagonal. The degrees-of-freedom parameter, n, in the prior for $\breve{\boldsymbol{\Sigma}}$ can be used to control the dispersion of the implied prior density of $\boldsymbol{\Sigma}$ around the prior expected value:

- $\text{E}(\boldsymbol{\Sigma}) = \mathbf{I}_M$ if $\mathbf{S}$ is diagonal
- $\text{E}(\boldsymbol{\Sigma}) = \mathbf{DSD}$ if $\mathbf{S}$ is not diagonal

In both cases smaller values of n allow larger deviations form $\text{E}(\boldsymbol{\Sigma})$.

Due to an inverse-Wishart distribution being used as the prior for $\breve{\boldsymbol{\Sigma}}$, the prior expected value of $\breve{\boldsymbol{\Sigma}}$ is $\frac{1}{n-M-1}\mathbf{S}$. Although the prior for $\breve{\boldsymbol{\Sigma}}$ is transformed to a prior for $\boldsymbol{\Sigma}$ such that the latter is a correlation matrix, due to simulation-based integration taking place in this transformation when $\mathbf{S}$ is not diagonal, hyperparameter values that result into an expected value of $\breve{\boldsymbol{\Sigma}}$ that is far from satisfying the conditions necessary for it to be a correlation matrix may lead to numerical unstability issues when calculating the log-marginal likelihood of the model. It is, therefore, advised that n is used to control the dispersion of $\boldsymbol{\Sigma}$ around its expected value and $\mathbf{S}$ is subsequently defined such that $\frac{1}{n-M-1}\mathbf{S}$ is close to being a proper correlation matrix (positive definite with values equal to one on the main diagonal).

Syntax

```
[<model name> = ] mvprobit( {
      y1 ~ x11 x12 ... x1K1,
      y2 ~ x21 x22 ... x2K2,
      ...,
      yM ~ xM1 xM2 ... xMKM }
      [, <options> ]
);
```

where:

- `y1`, `y2`, ..., `yM` are the dependent variable names, as they appear in the dataset used for estimation
- `xm1 xm2 ... xmK`$_m$ is a list of the K_m independent variable names for equation $m = 1, 2, \ldots, M$, as they appear in the dataset used for estimation; when a constant term is to be included in an equation, this must be requested explicitly; M such lists must be provided

All dependent variables, `y1`, `y2`, ..., `yM`, in the dataset used for estimation must contain only two values: 0 and 1 (with 1 indicating "success" for the respective outcome). Observations with missing values in any dependent variable are dropped during estimation, but if a numerical value other than 0 and 1 is encountered, then an error is produced.

The optional arguments for the multivarite Probit model are:[9]

[9] Optional arguments are always given in option-value pairs (eg. `"chains"=3`).

Gibbs parameters	
`"chains"`	number of chains to run in parallel (positive integer); the default value is 1
`"burnin"`	number of burn-in draws per chain (positive integer); the default value is 10000
`"draws"`	number of retained draws per chain (positive integer); the default value is 20000
`"thin"`	value of the thinning parameter (positive integer); the default value is 1
`"seed"`	value of the seed for the random-number generator (positive integer); the default value is 42
Hyperparameters	
`"m"`	mean vector of the prior for $\boldsymbol{\beta}$ ($K \times 1$ vector); the default value is $\mathbf{0}_K$
`"P"`	precision matrix of the prior for $\boldsymbol{\beta}$ ($K \times K$ symmetric and positive-definite matrix); the default value is $0.001 \cdot \mathbf{I}_K$
`"S"`	scale matrix of the prior for $\boldsymbol{\Sigma}$ ($M \times M$ symmetric and positive-definite matrix); the default value is $\frac{100}{M} \cdot \mathbf{I}_M$
`"n"`	degrees-of-freedom parameter of the prior for $\boldsymbol{\Sigma}$ (real number greater than or equal to M); the default value is M^2
Dataset and log-marginal likelihood	
`"dataset"`	the id value of the dataset that will be used for estimation; the default value is the first dataset in memory (in alphabetical order)
`"logML_CJ"`	boolean indicating whether the Chib (1995)/Chib & Jeliazkov (2001) approximation to the log-marginal likelihood should be calculated (**true**|**false**); the default value is **false**

Reported Parameters

$\boldsymbol{\beta}$	`variable_name`	vector of parameters associated with the independent variables; these are broken into groups according to the equation in which the independent variables appear

Stored values and post-estimation analysis

If a left-hand-side id value is provided when a multivariate Probit model is created, then the following results are saved in the model item and are accessible via the '.' operator:

`Samples`	a matrix containing the draws from the posterior of $\boldsymbol{\beta}$ (across all equations, starting from the first equation) and the unique elements of $\boldsymbol{\Sigma}$
`ym$xm1,...,ym$xmK`$_m$	vectors containing the draws from the posterior of the parameters associated with variables `xm1,...,xmK`$_m$, for $m = 1, 2, \ldots, M$ (the names of these vectors are the names of the variables that were included in the right-hand side of equation m, prepended by `ym$`, where `ym` is the name of the dependent variable in equation m; this is done so that the samples on the parameters associated with a variable that appears in more than one equations can be distinguished)
`Sigma_i_j`	vectors containing the draws from the posterior of the unique elements of $\boldsymbol{\Sigma}$; because $\boldsymbol{\Sigma}$ is symmetric, only $\frac{(M-1)M}{2} + M$ of its elements are stored (instead of all M^2 elements), including the elements that are restricted to be equal to one; `i` and `j` index the row and column of $\boldsymbol{\Sigma}$, respectively, at which the corresponding element is located
`Sigma`	$M \times M$ matrix that stores the posterior mean of $\boldsymbol{\Sigma}$
`logML`	the Lewis & Raftery (1997) approximation of the log-marginal likelihood
`logML_CJ`	the Chib (1995)/Chib & Jeliazkov (2001) approximation to the log-marginal likelihood; this is available only if the model was estimated with the `"logML_CJ"=`**true** option

`nchains`	the number of chains that were used to estimate the model
`nburnin`	the number of burn-in draws per chain that were used when estimating the model
`ndraws`	the total number of retained draws from the posterior (=`chains` · `draws`)
`nthin`	value of the thinning parameter that was used when estimating the model
`nseed`	value of the seed for the random-number generator that was used when estimating the model

Additionally, the following functions are available for post-estimation analysis (see section B.14):

- `diagnostics()`
- `pmp()`
- `predict()`
- `test()`
- `mfx()`

The multivariate Probit model uses the `mfx()` function to calculate and report the marginal effects of the independent variables on the probability of success for each of M dependent variables. There are two types of marginal effects which can be requested by setting the `"type"` argument of the `mfx()` function equal to `1` or `2`:

1. when `"type"=1` the marginal effects for each of the M outcomes are calculated marginally with respect to the values of the remaining dependent variables

2. when `"type"=2` the marginal effects are calculated conditionally on the values of other dependent variables being equal to the values indicated by a vector `z`, passed to the `mfx()` function using the `"opt"` option. This vector must have dimension equal to M and values equal to either 0 or 1. A value of 0 in the m-th position of `z` indicates that the m-th dependent variable is to restricted to 0 when calculating marginal effects on the remaining variables; a value of 1 indicates that the m-th dependent variable is to be restricted to 1 when calculating these marginal effects.

The generic syntax for a statement involving the `mfx()` function after estimation of a multivariate Probit model is:

```
mfx( ["type"=1] [, "point"=<point of calculation>] [, "model"=<model name>] );
```

and:

```
mfx( "type"=2, "opt"=z [, "point"=<point of calculation>] [, "model"=<model name>] );
```

for calculating these two types of marginal effects. The default value of the `"type"` option is `1`. See the general documentation of the `mfx()` function (section B.14) for details on the other optional arguments.

Although BayES can calculate marginal effects for the multivariate Probit model at each observation, the calculations may take an excessive amount of time to complete. This is because the GHK simulator needs to be invoked at every observed data point, each time using all draws from the posterior, thus leading to an immense number of computations.

The multivariate Probit model uses the `predict()` function to generate predictions of the probability of success for each of M dependent variables. There are two types of predictions which can be requested by setting the `"type"` argument of the `predict()` function equal to `1` or `2`:

1. when `"type"=1` the predictions are generated for each of the M dependent variables are calculated marginally with respect to the values of the remaining dependent variables

2. when `"type"=2` the predictions are generated conditionally on the values of other dependent variables being equal to the values indicated by a vector `z`, passed to the `predict()` function using the `"opt"` option. This vector must have dimension equal to M and values equal to either 0 or 1. A value of 0 in the m-th position of `z` indicates that the m-th dependent variable is to restricted to 0 when generating predictions for the remaining variables; a value of 1 indicates that the m-th dependent variable is to be restricted to 1 when generating these predictions.

The generic syntax for a statement involving the `predict()` function after estimation of a random-effects binary Probit model is:

```
[<id value>] = predict( ["type"=1] [, "point"=<point of calculation>] [,"model"=<model
    name>] [, "stats"=true|false] [, "prefix"=<prefix for new variable name>] );
```

and:

```
[<id value>] = predict( "type"=2, "opt"=z [, "point"=<point of
    calculation>] [,"model"=<model name>] [, "stats"=true|false] [, "prefix"=<prefix for new
    variable name>]
);
```

for generating these two types of predictions effects. The default value of the `"type"` option is `1`. See the general documentation of the `predict()` function (section B.14) for details on the other optional arguments.

Although BayES can generate summary statistics of the predictions for the multivariate Probit model at each observation, the calculations may take an excessive amount of time to complete. This is because the GHK simulator needs to be invoked at every observed data point, each time using all draws from the posterior, thus leading to an immense number of computations.

Examples

Example 1

```
myData = import("$BayESHOME/Datasets/dataset11.csv");
myData.constant = ones(rows(myData), 1);

myModel = mvprobit( {
    y1 ~ constant x11 x12 x13 x14 x15,
    y2 ~ constant x21 x22 x23 x24,
    y3 ~ constant x21 x22 x23
} );
```

Example 2

```
myData = import("$BayESHOME/Datasets/dataset11.csv");
myData.constant = ones(rows(myData), 1);

myModel = mvprobit( {
    y1 ~ constant x11 x12 x13 x14 x15,
    y2 ~ constant x21 x22 x23 x24,
    y3 ~ constant x21 x22 x23
    },
    "m"=ones(15,1), "P"=0.01*eye(15,15), "n"=5,
    "burnin"=20000, "draws"=40000, "thin"=4, "chains"=2,
    "logML_CJ" = true
);

mfx( "point"="mean", "model"=myModel );
x_for_mfx = [
    1.0,1.0,1.0,1.0,1.0,0.0, // values for the variables in the 1st equation
    1.0,0.0,0.5,0.5,0.5,     // values for the variables in the 2nd equation
    1.0,0.0,0.5,0.5          // values for the variables in the 3rd equation
    ];
mfx( "point"=x_for_mfx, "model"=myModel );
mfx( "point"="mean", "model"=myModel, "type"=2, "opt"=[1,0,1]);

predict("prefix"=marg_);
predict("prefix"=cond_, "type"=2, "opt"=[1,0,1]);
```

Chapter 7

Models for Ordered Data

7.1 Ordered Probit model

Mathematical representation

$$y_i^* = \mathbf{x}_i'\boldsymbol{\beta} + \varepsilon_i, \qquad \varepsilon_i \sim \mathrm{N}(0,1)$$
$$y_i = \begin{cases} 1 & \text{if} \quad \gamma_0 < y_i^* \leq \gamma_1 \\ 2 & \text{if} \quad \gamma_1 < y_i^* \leq \gamma_2 \\ \vdots & \vdots \quad\quad \vdots \\ M & \text{if} \quad \gamma_{M-1} < y_i^* \leq \gamma_M \end{cases} \tag{7.1}$$

- the model is estimated using N observations
- y_i is the value of the dependent variable for observation i and it can assume integer values in the range $1, \ldots, M$
- $\mathbf{x}_i$ is a $K \times 1$ vector that stores the values of the K independent variables for observation i
- $\boldsymbol{\beta}$ is a $K \times 1$ vector of parameters
- the γs are parameters that represent the cutoff points between categories and they satisfy the relation $\gamma_0 < \gamma_1 < \cdots < \gamma_M$, with $\gamma_0 = -\infty$, $\gamma_M = \infty$ and, for identification purposes, $\gamma_1 = 0$; there are $M-2$ γs to be estimated
- following Albert & Chib (2001), to impose the inequality constraints on the γs the problem is re-parameterized using the 1-1 mapping:

$$\begin{array}{rcc} \delta_2 & = & \log(\gamma_2 - \gamma_1) \\ \delta_3 & = & \log(\gamma_3 - \gamma_2) \\ \vdots & = & \vdots \\ \delta_{M-1} & = & \log(\gamma_{M-1} - \gamma_{M-2}) \end{array}$$

  there are $M-2$ δs to be estimated and they are collected in an $(M-2) \times 1$ vector $\boldsymbol{\delta}$

Priors

Parameter	Probability density function	Default hyperparameters
$\boldsymbol{\beta}$	$\mathrm{p}(\boldsymbol{\beta}) = \frac{\lvert\mathbf{P}_\beta\rvert^{1/2}}{(2\pi)^{K/2}} \exp\left\{-\frac{1}{2}(\boldsymbol{\beta} - \mathbf{m}_\beta)'\mathbf{P}_\beta(\boldsymbol{\beta} - \mathbf{m}_\beta)\right\}$	$\mathbf{m}_\beta = \mathbf{0}_K$, $\mathbf{P}_\beta = 0.001 \cdot \mathbf{I}_K$
$\boldsymbol{\delta}$	$\mathrm{p}(\boldsymbol{\delta}) = \frac{\lvert\mathbf{P}_\delta\rvert^{1/2}}{(2\pi)^{\frac{M-2}{2}}} \exp\left\{-\frac{1}{2}(\boldsymbol{\delta} - \mathbf{m}_\delta)'\mathbf{P}_\delta(\boldsymbol{\delta} - \mathbf{m}_\delta)\right\}$	$\mathbf{m}_\delta = \mathbf{0}_{M-2}$, $\mathbf{P}_\delta = 0.001 \cdot \mathbf{I}_{M-2}$

Syntax

```
[<model name> = ] oprobit( y ~ x1 x2 ... xK [, <options> ] );
```

where:

- `y` is the dependent variable name, as it appears in the dataset used for estimation
- `x1 x2 ... xK` is a list of the K independent variable names, as they appear in the dataset used for estimation; when a constant term is to be included in the model, this must be requested explicitly

The dependent variable, `y`, in the dataset used for estimation must contain only consecutive integer values, with the numbering starting at 1. Observations with missing values in `y` are dropped during estimation, but if a non-integer numerical value is encountered or if the integer values are not consecutive (for example there are no observations for which $y_i = 2$), then an error is produced.

The optional arguments for the ordered Probit model are:[1]

`Gibbs parameters`	
`"chains"`	number of chains to run in parallel (positive integer); the default value is 1
`"burnin"`	number of burn-in draws per chain (positive integer); the default value is 10000
`"draws"`	number of retained draws per chain (positive integer); the default value is 20000
`"thin"`	value of the thinning parameter (positive integer); the default value is 1
`"seed"`	value of the seed for the random-number generator (positive integer); the default value is 42
`Hyperparameters`	
`"m_beta"`	mean vector of the prior for $\boldsymbol{\beta}$ ($K \times 1$ vector); the default value is $\mathbf{0}_K$
`"P_beta"`	precision matrix of the prior for $\boldsymbol{\beta}$ ($K \times K$ symmetric and positive-definite matrix); the default value is $0.001 \cdot \mathbf{I}_K$
`"m_delta"`	mean vector of the prior for $\boldsymbol{\delta}$ ($(M-2) \times 1$ vector); the default value is $\mathbf{0}_{M-2}$
`"P_delta"`	precision matrix of the prior for $\boldsymbol{\delta}$ ($(M-2) \times (M-2)$ symmetric and positive-definite matrix); the default value is $0.001 \cdot \mathbf{I}_{M-2}$
`Dataset and log-marginal likelihood`	
`"dataset"`	the id value of the dataset that will be used for estimation; the default value is the first dataset in memory (in alphabetical order)
`"logML_CJ"`	boolean indicating whether the Chib (1995)/Chib & Jeliazkov (2001) approximation to the log-marginal likelihood should be calculated (**true**|**false**); the default value is **false**

Reported Parameters

$\boldsymbol{\beta}$	`variable_name`	vector of parameters associated with the independent variables
$\boldsymbol{\gamma}$	`gamma_m`	vector of cutoff points ($M-2$)

Stored values and post-estimation analysis

If a left-hand-side id value is provided when an ordered Probit model is created, then the following results are saved in the model item and are accessible via the '.' operator:

`Samples`	a matrix containing the draws from the posterior of $\boldsymbol{\beta}$ and $\boldsymbol{\gamma}$
`x1,...,xK`	vectors containing the draws from the posterior of the parameters associated with variables `x1,...,xK` (the names of these vectors are the names of the variables that were included in the right-hand side of the model)
`gamma_2,..., gamma_{M-1}`	vectors containing the draws from the posterior of the cutoff parameters, for $m = 2, \ldots, M-1$
`logML`	the Lewis & Raftery (1997) approximation of the log-marginal likelihood
`logML_CJ`	the Chib (1995)/Chib & Jeliazkov (2001) approximation to the log-marginal likelihood; this is available only if the model was estimated with the `"logML_CJ"=true` option
`nchains`	the number of chains that were used to estimate the model

[1] Optional arguments are always given in option-value pairs (eg. `"chains"=3`).

`nburnin`	the number of burn-in draws per chain that were used when estimating the model
`ndraws`	the total number of retained draws from the posterior (=`chains` · `draws`)
`nthin`	value of the thinning parameter that was used when estimating the model
`nseed`	value of the seed for the random-number generator that was used when estimating the model

Additionally, the following functions are available for post-estimation analysis (see section B.14):

- `diagnostics()`
- `pmp()`
- `test()`
- `mfx()`

The ordered Probit model uses the `mfx()` function to calculate and report the marginal effects of the independent variables on the probability of the response variable being in each one of the M categories: $\text{Prob}(y = m|\mathbf{x})$, for $m = 1, 2, \ldots, M$. Because the model calculates only one type of marginal effects, the only valid value for the `"type"` option is `1`. The generic syntax for a statement involving the `mfx()` function after estimation of an ordered Probit model is:

```
mfx( ["type"=1] [, "point"=<point of calculation>] [, "model"=<model name>] );
```

See the general documentation of the `mfx()` function (section B.14) for details on the other optional arguments.

Examples

Example 1

```
myData = import("$BayESHOME/Datasets/dataset10.csv");
myData.constant = ones(rows(myData), 1);

oprobit( y ~ constant x1 x2 x3 x4 );
```

Example 2

```
myData = import("$BayESHOME/Datasets/dataset10.csv");
myData.constant = ones(rows(myData), 1);

myModel = oprobit( y ~ constant x1 x2 x3 x4,
    "m_beta"=zeros(5,1), "P_beta" = 0.01*eye(5,5),
    "m_delta"=zeros(3,1), "P_delta" = 0.1*eye(3,3),
    "burnin"=10000, "draws"=40000, "thin"=4, "chains"=2,
    "logML_CJ" = true, "dataset"=myData);

diagnostics("model"=myModel);

mfx("point"="mean","model"=myModel);
mfx("point"="median","model"=myModel);
```

7.2 Ordered Logit model

Mathematical representation

$$
\begin{aligned}
y_i^* &= \mathbf{x}_i'\boldsymbol{\beta} + \varepsilon_i, \qquad \varepsilon_i \sim \text{Logistic}\,(0,1) \\
y_i &= \begin{cases} 1 & \text{if} \quad \gamma_0 < y_i^* \leq \gamma_1 \\ 2 & \text{if} \quad \gamma_1 < y_i^* \leq \gamma_2 \\ \vdots & \vdots \quad\quad \vdots \\ M & \text{if} \quad \gamma_{M-1} < y_i^* \leq \gamma_M \end{cases}
\end{aligned} \tag{7.2}
$$

- the model is estimated using N observations
- y_i is the value of the dependent variable for observation i and it can assume integer values in the range $1, \ldots, M$
- $\mathbf{x}_i$ is a $K \times 1$ vector that stores the values of the K independent variables for observation i
- $\boldsymbol{\beta}$ is a $K \times 1$ vector of parameters
- the γs are parameters that represent the cutoff points between categories and they satisfy the relation $\gamma_0 < \gamma_1 < \cdots < \gamma_M$, with $\gamma_0 = -\infty$, $\gamma_M = \infty$ and, for identification purposes, $\gamma_1 = 0$; there are $M-2$ γs to be estimated
- following Albert & Chib (2001), to impose the inequality constraints on the γs the problem is re-parameterized using the 1-1 mapping:

$$
\begin{aligned}
\delta_2 &= \log(\gamma_2 - \gamma_1) \\
\delta_3 &= \log(\gamma_3 - \gamma_2) \\
\vdots &= \quad\quad \vdots \\
\delta_{M-1} &= \log(\gamma_{M-1} - \gamma_{M-2})
\end{aligned}
$$

  there are $M-2$ δs to be estimated and they are collected in an $(M-2) \times 1$ vector $\boldsymbol{\delta}$

Priors

Parameter	Probability density function	Default hyperparameters
$\boldsymbol{\beta}$	$\text{p}(\boldsymbol{\beta}) = \frac{\lvert\mathbf{P}_\beta\rvert^{1/2}}{(2\pi)^{K/2}} \exp\left\{-\frac{1}{2}(\boldsymbol{\beta} - \mathbf{m}_\beta)'\mathbf{P}_\beta(\boldsymbol{\beta} - \mathbf{m}_\beta)\right\}$	$\mathbf{m}_\beta = \mathbf{0}_K$, $\mathbf{P}_\beta = 0.001 \cdot \mathbf{I}_K$
$\boldsymbol{\delta}$	$\text{p}(\boldsymbol{\delta}) = \frac{\lvert\mathbf{P}_\delta\rvert^{1/2}}{(2\pi)^{\frac{M-2}{2}}} \exp\left\{-\frac{1}{2}(\boldsymbol{\delta} - \mathbf{m}_\delta)'\mathbf{P}_\delta(\boldsymbol{\delta} - \mathbf{m}_\delta)\right\}$	$\mathbf{m}_\delta = \mathbf{0}_{M-2}$, $\mathbf{P}_\delta = 0.001 \cdot \mathbf{I}_{M-2}$

Syntax

```
[<model name> = ] ologit( y ~ x1 x2 ... xK [, <options> ] );
```

where:

- `y` is the dependent variable name, as it appears in the dataset used for estimation
- `x1 x2 ... xK` is a list of the K independent variable names, as they appear in the dataset used for estimation; when a constant term is to be included in the model, this must be requested explicitly

The dependent variable, `y`, in the dataset used for estimation must contain only consecutive integer values, with the numbering starting at 1. Observations with missing values in `y` are dropped during estimation, but if a non-integer numerical value is encountered or if the integer values are not consecutive (for example there are no observations for which $y_i = 2$), then an error is produced.

The optional arguments for the ordered Logit model are:[2]

`Gibbs parameters`	
`"chains"`	number of chains to run in parallel (positive integer); the default value is 1
`"burnin"`	number of burn-in draws per chain (positive integer); the default value is 10000
`"draws"`	number of retained draws per chain (positive integer); the default value is 20000
`"thin"`	value of the thinning parameter (positive integer); the default value is 1
`"seed"`	value of the seed for the random-number generator (positive integer); the default value is 42
`Hyperparameters`	
`"m_beta"`	mean vector of the prior for $\boldsymbol{\beta}$ ($K \times 1$ vector); the default value is $\mathbf{0}_K$
`"P_beta"`	precision matrix of the prior for $\boldsymbol{\beta}$ ($K \times K$ symmetric and positive-definite matrix); the default value is $0.001 \cdot \mathbf{I}_K$
`"m_delta"`	mean vector of the prior for $\boldsymbol{\delta}$ ($(M-2) \times 1$ vector); the default value is $\mathbf{0}_{M-2}$
`"P_delta"`	precision matrix of the prior for $\boldsymbol{\delta}$ ($(M-2) \times (M-2)$ symmetric and positive-definite matrix); the default value is $0.001 \cdot \mathbf{I}_{M-2}$
`Dataset and log-marginal likelihood`	
`"dataset"`	the id value of the dataset that will be used for estimation; the default value is the first dataset in memory (in alphabetical order)
`"logML_CJ"`	boolean indicating whether the Chib (1995)/Chib & Jeliazkov (2001) approximation to the log-marginal likelihood should be calculated (**true**|**false**); the default value is **false**

Reported Parameters

$\boldsymbol{\beta}$	`variable_name`	vector of parameters associated with the independent variables
$\boldsymbol{\gamma}$	`gamma_m`	vector of cutoff points ($M-2$)

Stored values and post-estimation analysis

If a left-hand-side id value is provided when an ordered Logit model is created, then the following results are saved in the model item and are accessible via the '.' operator:

`Samples`	a matrix containing the draws from the posterior of $\boldsymbol{\beta}$ and $\boldsymbol{\gamma}$
`x1,...,xK`	vectors containing the draws from the posterior of the parameters associated with variables `x1,...,xK` (the names of these vectors are the names of the variables that were included in the right-hand side of the model)
`gamma_2,..., gamma_{M-1}`	vectors containing the draws from the posterior of the cutoff parameters, for $m = 2, \ldots, M-1$
`logML`	the Lewis & Raftery (1997) approximation of the log-marginal likelihood
`logML_CJ`	the Chib (1995)/Chib & Jeliazkov (2001) approximation to the log-marginal likelihood; this is available only if the model was estimated with the `"logML_CJ"=true` option
`nchains`	the number of chains that were used to estimate the model

[2] Optional arguments are always given in option-value pairs (eg. `"chains"=3`).

`nburnin`	the number of burn-in draws per chain that were used when estimating the model
`ndraws`	the total number of retained draws from the posterior (=`chains` · `draws`)
`nthin`	value of the thinning parameter that was used when estimating the model
`nseed`	value of the seed for the random-number generator that was used when estimating the model

Additionally, the following functions are available for post-estimation analysis (see section B.14):

- `diagnostics()`
- `pmp()`
- `test()`
- `mfx()`

The ordered Logit model uses the `mfx()` function to calculate and report the marginal effects of the independent variables on the probability of the response variable being in each one of the M categories: Prob$(y = m|\mathbf{x})$, for $m = 1, 2, \ldots, M$. Because the model calculates only one type of marginal effects, the only valid value for the `"type"` option is `1`. The generic syntax for a statement involving the `mfx()` function after estimation of an ordered Logit model is:

```
mfx( ["type"=1] [, "point"=<point of calculation>] [, "model"=<model name>] );
```

See the general documentation of the `mfx()` function (section B.14) for details on the other optional arguments.

Examples

Example 1

```
myData = import("$BayESHOME/Datasets/dataset10.csv");
myData.constant = ones(rows(myData), 1);

ologit( y ~ constant x1 x2 x3 x4 );
```

Example 2

```
myData = import("$BayESHOME/Datasets/dataset10.csv");
myData.constant = ones(rows(myData), 1);

myModel = ologit( y ~ constant x1 x2 x3 x4,
    "m_beta"=zeros(5,1), "P_beta" = 0.01*eye(5,5),
    "m_delta"=zeros(3,1), "P_delta" = 0.1*eye(3,3),
    "burnin"=10000, "draws"=40000, "thin"=4, "chains"=2,
    "logML_CJ" = true, "dataset"=myData);

diagnostics("model"=myModel);

mfx("point"="mean","model"=myModel);
mfx("point"="median","model"=myModel);
```

Chapter 8

Models for Count Data

8.1 Poisson model

Mathematical representation

$$y_i \sim \text{Poisson}\left(\lambda_i\right), \qquad \lambda_i = e^{\mathbf{x}_i'\boldsymbol{\beta}} \tag{8.1}$$

- the model is estimated using N observations
- y_i is the value of the dependent variable for observation i and it can assume non-negative integer values
- $\mathbf{x}_i$ is a $K \times 1$ vector that stores the values of the K independent variables for observation i
- $\boldsymbol{\beta}$ is a $K \times 1$ vector of parameters

Priors

Parameter	Probability density function	Default hyperparameters
$\boldsymbol{\beta}$	$\text{p}\left(\boldsymbol{\beta}\right) = \frac{\left\|\mathbf{P}\right\|^{1/2}}{(2\pi)^{K/2}} \exp\left\{-\frac{1}{2}\left(\boldsymbol{\beta}-\mathbf{m}\right)'\mathbf{P}\left(\boldsymbol{\beta}-\mathbf{m}\right)\right\}$	$\mathbf{m} = \mathbf{0}_K$, $\mathbf{P} = 0.001 \cdot \mathbf{I}_K$

Syntax

```
[<model name> = ] poisson( y ~ x1 x2 ... xK [, <options> ] );
```

where:

- `y` is the dependent variable name, as it appears in the dataset used for estimation
- `x1 x2 ... xK` is a list of the K independent variable names, as they appear in the dataset used for estimation; when a constant term is to be included in the model, this must be requested explicitly

The dependent variable, `y`, in the dataset used for estimation must contain non-negative integer values. Observations with missing values in `y` are dropped during estimation, but if a non-integer or negative numerical value is encountered, then an error is produced.

The optional arguments for the Poisson model are:[1]

Gibbs parameters	
`"chains"`	number of chains to run in parallel (positive integer); the default value is 1
`"burnin"`	number of burn-in draws per chain (positive integer); the default value is 10000
`"draws"`	number of retained draws per chain (positive integer); the default value is 20000
`"thin"`	value of the thinning parameter (positive integer); the default value is 1
`"seed"`	value of the seed for the random-number generator (positive integer); the default value is 42
Hyperparameters	
`"m"`	mean vector of the prior for $\boldsymbol{\beta}$ ($K \times 1$ vector); the default value is $\mathbf{0}_K$
`"P"`	precision matrix of the prior for $\boldsymbol{\beta}$ ($K \times K$ symmetric and positive-definite matrix); the default value is $0.001 \cdot \mathbf{I}_K$

[1] Optional arguments are always given in option-value pairs (eg. `"chains"=3`).

Dataset and log-marginal likelihood	
`"dataset"`	the id value of the dataset that will be used for estimation; the default value is the first dataset in memory (in alphabetical order)
`"logML_CJ"`	boolean indicating whether the Chib (1995)/Chib & Jeliazkov (2001) approximation to the log-marginal likelihood should be calculated (**true**|**false**); the default value is **false**

Reported Parameters

$\boldsymbol{\beta}$	`variable_name`	vector of parameters associated with the independent variables

Stored values and post-estimation analysis

If a left-hand-side id value is provided when a Poisson model is created, then the following results are saved in the model item and are accessible via the '.' operator:

`Samples`	a matrix containing the draws from the posterior of $\boldsymbol{\beta}$
`x1,...,xK`	vectors containing the draws from the posterior of the parameters associated with variables `x1,...,xK` (the names of these vectors are the names of the variables that were included in the right-hand side of the model)
`logML`	the Lewis & Raftery (1997) approximation of the log-marginal likelihood
`logML_CJ`	the Chib (1995)/Chib & Jeliazkov (2001) approximation to the log-marginal likelihood; this is available only if the model was estimated with the `"logML_CJ"`=**true** option
`nchains`	the number of chains that were used to estimate the model
`nburnin`	the number of burn-in draws per chain that were used when estimating the model
`ndraws`	the total number of retained draws from the posterior (=`chains` · `draws`)
`nthin`	value of the thinning parameter that was used when estimating the model
`nseed`	value of the seed for the random-number generator that was used when estimating the model

Additionally, the following functions are available for post-estimation analysis (see section B.14):

- `diagnostics()`
- `pmp()`
- `test()`
- `mfx()`

The Poisson model uses the `mfx()` function to calculate and report the marginal effects of the independent variables on the expected value of the dependent variable, $\mathrm{E}(y_i|\mathbf{x}_i) = \lambda_i$. Because the model calculates only one type of marginal effects, the only valid value for the `"type"` option is `1`. The generic syntax for a statement involving the `mfx()` function after estimation of a Poisson model is:

```
mfx( ["type"=1] [, "point"=<point of calculation>] [, "model"=<model name>] );
```

See the general documentation of the `mfx()` function (section B.14) for details on the other optional arguments.

Examples

Example 1

```
myData = import("$BayESHOME/Datasets/dataset9.csv");
myData.constant = ones(rows(myData), 1);

poisson( y ~ constant x1 x2 x3 x4 );
```

Example 2

```
myData = import("$BayESHOME/Datasets/dataset9.csv");
myData.constant = ones(rows(myData), 1);

myModel = poisson( y ~ constant x1 x2 x3 x4,
    "m"=zeros(5,1), "P" = 0.01*eye(5,5),
    "burnin"=10000, "draws"=40000, "thin"=4, "chains"=2,
    "logML_CJ" = true, "dataset"=myData);

diagnostics("model"=myModel);

mfx("point"="mean","model"=myModel);
mfx("point"="median","model"=myModel);
```

8.2 Negative-Binomial model

Mathematical representation

$$y_i \sim \text{NBinom}\left(p_i, \gamma\right), \qquad p_i = \frac{\mu_i}{\mu_i + \gamma} \quad \text{and} \quad \mu_i = e^{\mathbf{x}_i' \boldsymbol{\beta}} \tag{8.2}$$

With this parameterization, $\text{E}\left(y_i|\mathbf{x}_i\right) = \mu_i$ and $\text{V}\left(y_i|\mathbf{x}_i\right) = \mu_i + \frac{\mu_i^2}{\gamma}$

- the model is estimated using N observations
- y_i is the value of the dependent variable for observation i and it can assume non-negative integer values
- $\mathbf{x}_i$ is a $K \times 1$ vector that stores the values of the K independent variables for observation i
- $\boldsymbol{\beta}$ is a $K \times 1$ vector of parameters
- γ is the over-dispersion parameter and as $\gamma \to \infty$, the negative-Binomial model tends towards the Poisson model

Priors

Parameter	Probability density function	Default hyperparameters
$\boldsymbol{\beta}$	$\text{p}\left(\boldsymbol{\beta}\right) = \frac{\left\|\mathbf{P}\right\|^{1/2}}{(2\pi)^{K/2}} \exp\left\{-\frac{1}{2}\left(\boldsymbol{\beta} - \mathbf{m}\right)' \mathbf{P} \left(\boldsymbol{\beta} - \mathbf{m}\right)\right\}$	$\mathbf{m} = \mathbf{0}_K$, $\mathbf{P} = 0.001 \cdot \mathbf{I}_K$
γ	$\text{p}\left(\gamma\right) = \frac{b_\gamma^{a_\gamma}}{\Gamma(a_\gamma)} \gamma^{a_\gamma - 1} e^{-\gamma b_\gamma}$	$a_\gamma = 0.001$, $b_\gamma = 0.001$

Syntax

```
[<model name> = ] nbinom( y ~ x1 x2 ... xK [, <options> ] );
```

where:

- `y` is the dependent variable name, as it appears in the dataset used for estimation
- `x1 x2 ... xK` is a list of the K independent variable names, as they appear in the dataset used for estimation; when a constant term is to be included in the model, this must be requested explicitly

The dependent variable, `y`, in the dataset used for estimation must contain non-negative integer values. Observations with missing values in `y` are dropped during estimation, but if a non-integer or negative numerical value is encountered, then an error is produced.

The optional arguments for the negative-Binomial model are:[2]

Gibbs parameters	
`"chains"`	number of chains to run in parallel (positive integer); the default value is 1
`"burnin"`	number of burn-in draws per chain (positive integer); the default value is 10000
`"draws"`	number of retained draws per chain (positive integer); the default value is 20000
`"thin"`	value of the thinning parameter (positive integer); the default value is 1
`"seed"`	value of the seed for the random-number generator (positive integer); the default value is 42

[2]Optional arguments are always given in option-value pairs (eg. `"chains"=3`).

Hyperparameters	
"m"	mean vector of the prior for $\boldsymbol{\beta}$ ($K \times 1$ vector); the default value is $\mathbf{0}_K$
"P"	precision matrix of the prior for $\boldsymbol{\beta}$ ($K \times K$ symmetric and positive-definite matrix); the default value is $0.001 \cdot \mathbf{I}_K$
"a_gamma"	shape parameter of the prior for γ (positive number); the default value is 0.001
"b_gamma"	rate parameter of the prior for γ (positive number); the default value is 0.001
Dataset and log-marginal likelihood	
"dataset"	the id value of the dataset that will be used for estimation; the default value is the first dataset in memory (in alphabetical order)
"logML_CJ"	boolean indicating whether the Chib (1995)/Chib & Jeliazkov (2001) approximation to the log-marginal likelihood should be calculated (**true**|**false**); the default value is **false**

Reported Parameters

$\boldsymbol{\beta}$	variable_name	vector of parameters associated with the independent variables
γ	gamma	over-dispersion parameter

Stored values and post-estimation analysis

If a left-hand-side id value is provided when a negative-Binomial model is created, then the following results are saved in the model item and are accessible via the '.' operator:

Samples	a matrix containing the draws from the posterior of $\boldsymbol{\beta}$ and γ
x1,...,xK	vectors containing the draws from the posterior of the parameters associated with variables x1,...,xK (the names of these vectors are the names of the variables that were included in the right-hand side of the model)
gamma	vector containing the draws from the posterior of γ
logML	the Lewis & Raftery (1997) approximation of the log-marginal likelihood
logML_CJ	the Chib (1995)/Chib & Jeliazkov (2001) approximation to the log-marginal likelihood; this is available only if the model was estimated with the "logML_CJ"=**true** option
nchains	the number of chains that were used to estimate the model
nburnin	the number of burn-in draws per chain that were used when estimating the model
ndraws	the total number of retained draws from the posterior (=chains · draws)
nthin	value of the thinning parameter that was used when estimating the model
nseed	value of the seed for the random-number generator that was used when estimating the model

Additionally, the following functions are available for post-estimation analysis (see section B.14):

- diagnostics()
- pmp()
- test()
- mfx()

The negative-Binomial model uses the mfx() function to calculate and report the marginal effects of the variables in the x list on:

- the expected value of the dependent variable: $\frac{\partial \mathrm{E}(y_i|\mathbf{x}_i)}{\partial x_{ik}}$
- the variance of the dependent variable: $\frac{\partial \mathrm{V}(y_i|\mathbf{x}_i)}{\partial x_{ik}}$

These two types of marginal effects can be requested by setting the "type" argument of the mfx() function equal to 1 or 2, respectively. The generic syntax for a statement involving the mfx() function after estimation of a negative-Binomial model is:

```
mfx( ["type"=1] [, "point"=<point of calculation>] [, "model"=<model name>] );
```

and:

```
mfx( "type"=2 [, "point"=<point of calculation>] [, "model"=<model name>] );
```

for calculation of the marginal effects on $\mathrm{E}(y)$, and on $\mathrm{V}(y)$, respectively. The default value of the `"type"` option is `1`. See the general documentation of the `mfx()` function (section B.14) for details on the other optional arguments.

Examples

Example 1

```
myData = import("$BayESHOME/Datasets/dataset9.csv");
myData.constant = ones(rows(myData), 1);

nbinom( y ~ constant x1 x2 x3 x4 );
```

Example 2

```
myData = import("$BayESHOME/Datasets/dataset9.csv");
myData.constant = ones(rows(myData), 1);

myModel = nbinom( y ~ constant x1 x2 x3 x4,
    "m"=zeros(5,1), "P" = 0.1*eye(5,5),
    "a_gamma"=0.01, "b_gamma"=0.1,
    "burnin"=10000, "draws"=30000, "thin"=3, "chains"=2,
    "logML_CJ" = true, "dataset"=myData);

diagnostics("model"=myModel);

mfx("type"=1,"point"="median","model"=myModel);
mfx("type"=2,"point"="median","model"=myModel);
```

Chapter 9

Models for Censored and Truncated Dependent Variables

9.1 Type I Tobit

Mathematical representation

$$
\begin{aligned}
y_i^* &= \mathbf{x}_i'\boldsymbol{\beta} + \varepsilon_i, \qquad \varepsilon_i \sim \mathrm{N}\left(0, \tfrac{1}{\tau}\right) \\
y_i &= \begin{cases} \ell & \text{if} \quad y_i^* \leq \ell \\ y_i^* & \text{if} \quad \ell < y_i^* < u \\ u & \text{if} \quad y_i^* \geq u \end{cases}
\end{aligned}
\tag{9.1}
$$

- the model is estimated using N observations
- y_i is the value of the dependent variable for observation i and it can assume values in the interval $[\ell, u]$; ℓ could be $-\infty$ or u $+\infty$
- y_i^* is unobserved if $y_i^* \notin [\ell, u]$
- $\mathbf{x}_i$ is a $K \times 1$ vector that stores the values of the K independent variables for observation i
- $\boldsymbol{\beta}$ is a $K \times 1$ vector of parameters
- τ is the precision of the error term: $\sigma_\varepsilon^2 = \frac{1}{\tau}$

Priors

Parameter	Probability density function	Default hyperparameters
$\boldsymbol{\beta}$	$\mathrm{p}(\boldsymbol{\beta}) = \frac{\lvert\mathbf{P}\rvert^{1/2}}{(2\pi)^{K/2}} \exp\left\{-\frac{1}{2}(\boldsymbol{\beta}-\mathbf{m})'\mathbf{P}(\boldsymbol{\beta}-\mathbf{m})\right\}$	$\mathbf{m} = \mathbf{0}_K$, $\mathbf{P} = 0.001 \cdot \mathbf{I}_K$
τ	$\mathrm{p}(\tau) = \frac{b_\tau^{a_\tau}}{\Gamma(a_\tau)} \tau^{a_\tau - 1} e^{-\tau b_\tau}$	$a_\tau = 0.001$, $b_\tau = 0.001$

Syntax

```
[<model name> = ] tobitI( y ~ x1 x2 ... xK [, <options> ] );
```

where:

- `y` is the dependent variable name, as it appears in the dataset used for estimation
- `x1 x2 ... xK` is a list of the K independent variable names, as they appear in the dataset used for estimation; when a constant term is to be included in the model, this must be requested explicitly

The dependent variable, `y`, in the dataset used for estimation must contain values between the lower and upper censoring points. Observations with missing values in `y` are dropped during estimation, but if a numerical value beyond the provided bounds is encountered, then an error is produced.

The optional arguments for the type I Tobit model are:[1]

`Gibbs parameters`	
`"chains"`	number of chains to run in parallel (positive integer); the default value is 1
`"burnin"`	number of burn-in draws per chain (positive integer); the default value is 10000
`"draws"`	number of retained draws per chain (positive integer); the default value is 20000
`"thin"`	value of the thinning parameter (positive integer); the default value is 1

[1] Optional arguments are always given in option-value pairs (eg. `"chains"=3`).

`"seed"`	value of the seed for the random-number generator (positive integer); the default value is 42
`Model specification`	
`"lb"`	lower censoring point (it could be set equal to `-inf` if the dependent variable is not censored from below); the default value is 0
`"ub"`	upper censoring point (it could be set equal to `inf` if the dependent variable is not censored from above); the default value is $+\infty$
`Hyperparameters`	
`"m"`	mean vector of the prior for $\boldsymbol{\beta}$ ($K \times 1$ vector); the default value is $\mathbf{0}_K$
`"P"`	precision matrix of the prior for $\boldsymbol{\beta}$ ($K \times K$ symmetric and positive-definite matrix); the default value is $0.001 \cdot \mathbf{I}_K$
`"a_tau"`	shape parameter of the prior for τ (positive number); the default value is 0.001
`"b_tau"`	rate parameter of the prior for τ (positive number); the default value is 0.001
`Dataset and log-marginal likelihood`	
`"dataset"`	the id value of the dataset that will be used for estimation; the default value is the first dataset in memory (in alphabetical order)
`"logML_CJ"`	boolean indicating whether the Chib (1995)/Chib & Jeliazkov (2001) approximation to the log-marginal likelihood should be calculated (**true**|**false**); the default value is **false**

Reported Parameters

$\boldsymbol{\beta}$	`variable_name`	vector of parameters associated with the independent variables
τ	`tau`	precision parameter of the error term, ε_i
σ_ε	`sigma_e`	standard deviation of the error term: $\sigma_\varepsilon = 1/\tau^{1/2}$

Stored values and post-estimation analysis

If a left-hand-side id value is provided when a type I Tobit model is created, then the following results are saved in the model item and are accessible via the '.' operator:

`Samples`	a matrix containing the draws from the posterior of $\boldsymbol{\beta}$ and τ
`x1,...,xK`	vectors containing the draws from the posterior of the parameters associated with variables `x1,...,xK` (the names of these vectors are the names of the variables that were included in the right-hand side of the model)
`tau`	vector containing the draws from the posterior of τ
`lb`	lower censoring point
`ub`	upper censoring point
`logML`	the Lewis & Raftery (1997) approximation of the log-marginal likelihood
`logML_CJ`	the Chib (1995)/Chib & Jeliazkov (2001) approximation to the log-marginal likelihood; this is available only if the model was estimated with the `"logML_CJ"`=**true** option
`nchains`	the number of chains that were used to estimate the model
`nburnin`	the number of burn-in draws per chain that were used when estimating the model
`ndraws`	the total number of retained draws from the posterior (=`chains` · `draws`)
`nthin`	value of the thinning parameter that was used when estimating the model
`nseed`	value of the seed for the random-number generator that was used when estimating the model

Additionally, the following functions are available for post-estimation analysis (see section B.14):

- diagnostics()
- pmp()
- test()
- mfx()

The type I Tobit model uses the mfx() function to calculate and report the marginal effects of the variables in the x list on:

- the expected value of the observed dependent variable: $\frac{\partial \mathrm{E}(y_i|\mathbf{x}_i)}{\partial x_{ik}}$
- the expected value of the observed dependent variable, conditional on this being uncensored: $\frac{\partial \mathrm{E}(y_i|\mathbf{x}_i, \ell<y_i<u)}{\partial x_{ik}}$
- the probability of no censoring: $\frac{\partial \mathrm{Prob}(\ell<y_i<u|\mathbf{x}_i)}{\partial x_{ik}}$

The three types of marginal effects can be requested by setting the "type" argument of the mfx() function equal to 1, 2 or 3. The generic syntax for a statement involving the mfx() function after estimation of a type I Tobit model is:

```
mfx( ["type"=1] [, "point"=<point of calculation>] [, "model"=<model name>] );

mfx( "type"=2 [, "point"=<point of calculation>] [, "model"=<model name>] );
```

and:

```
mfx( "type"=3 [, "point"=<point of calculation>] [, "model"=<model name>] );
```

for calculation of the marginal effects on $\mathrm{E}(y)$, on $\mathrm{E}(y|\ell < y < u)$, and on $\mathrm{Prob}(\ell < y < u)$, respectively. The default value of the "type" option is 1. See the general documentation of the mfx() function (section B.14) for details on the other optional arguments.

Examples

Example 1

```
myData = import("$BayESHOME/Datasets/dataset8.csv");
myData.constant = ones(rows(myData), 1);

tobitI( y1 ~ constant x1 x2 x3 x4 );
```

Example 2

```
myData = import("$BayESHOME/Datasets/dataset8.csv");
myData.constant = ones(rows(myData), 1);
myData.y1 = -myData.y1;

myModel = tobitI( y1 ~ constant x1 x2 x3 x4,
    "lb"=-inf, "ub"=0,
    "m"=ones(5,1), "P" = 0.1*eye(5,5),
    "a_tau"=0.01, "b_tau"=0.01,
    "burnin"=10000, "draws"=40000, "thin"=4, "chains"=2,
    "logML_CJ" = true, "dataset"=myData);

diagnostics("model"=myModel);

mfx("type"=1,"point"="mean","model"=myModel);
mfx("type"=2,"point"="mean","model"=myModel);
mfx("type"=3,"point"="mean","model"=myModel);
```

9.2 Type II Tobit

Mathematical representation

$$\begin{aligned} y_i^* &= \mathbf{x}_i'\boldsymbol{\beta} + \varepsilon_i & \qquad s_i^* &= \mathbf{z}_i'\boldsymbol{\delta} + v_i \\ y_i &= \begin{cases} y_i^* & \text{if} \quad s_i = 1 \\ - & \text{if} \quad s_i = 0 \end{cases} & \qquad s_i &= \begin{cases} 1 & \text{if} \quad s_i^* > 0 \\ 0 & \text{if} \quad s_i^* \leq 0 \end{cases} \end{aligned} \tag{9.2}$$

with:

$$\begin{bmatrix} \varepsilon_i \\ v_i \end{bmatrix} \sim \mathrm{N}\left(\mathbf{0}, \boldsymbol{\Omega}^{-1}\right), \qquad \text{where} \qquad \boldsymbol{\Omega}^{-1} \equiv \boldsymbol{\Sigma} = \begin{bmatrix} \xi + \gamma^2 & \gamma \\ \gamma & 1 \end{bmatrix} \tag{9.3}$$

- the model is estimated using N observations, for N_1 of which the outcome variable is observed ($s_i = 1$) and for the remaining N_0 the outcome variable is missing ($s_i = 0$)
- y_i is the value of the dependent variable in the outcome equation for observation i and it is observed only if $s_i = 1$
- s_i^* is the value of the latent dependent variable in the selection equation for observation i
- $\mathbf{x}_i$ is a $K \times 1$ vector that stores the values of the K independent variables in the outcome equation for observation i
- $\mathbf{z}_i$ is an $L \times 1$ vector that stores the values of the L independent variables in the selection equation for observation i
- the same variable could appear in both $\mathbf{x}_i$ and $\mathbf{z}_i$
- $\boldsymbol{\beta}$ is a $K \times 1$ vector of parameters
- $\boldsymbol{\delta}$ is an $L \times 1$ vector of parameters
- $\boldsymbol{\Omega}$ is the precision matrix of the error vector, $\begin{bmatrix} \varepsilon_i & v_i \end{bmatrix}'$ and $\boldsymbol{\Sigma}$ is the covariance matrix of the error vector
- ξ is the variance of ε_i conditional on v_i
- γ is the covariance of ε_i and v_i

Priors

Parameter	Probability density function	Default hyperparameters
$\boldsymbol{\beta}$	$\mathrm{p}(\boldsymbol{\beta}) = \frac{\lvert\mathbf{P}_\beta\rvert^{1/2}}{(2\pi)^{K/2}} \exp\left\{-\frac{1}{2}(\boldsymbol{\beta} - \mathbf{m}_\beta)' \mathbf{P}_\beta (\boldsymbol{\beta} - \mathbf{m}_\beta)\right\}$	$\mathbf{m}_\beta = \mathbf{0}_K$, $\mathbf{P}_\beta = 0.001 \cdot \mathbf{I}_K$
$\boldsymbol{\delta}$	$\mathrm{p}(\boldsymbol{\delta}) = \frac{\lvert\mathbf{P}_\delta\rvert^{1/2}}{(2\pi)^{L/2}} \exp\left\{-\frac{1}{2}(\boldsymbol{\delta} - \mathbf{m}_\delta)' \mathbf{P}_\delta (\boldsymbol{\delta} - \mathbf{m}_\delta)\right\}$	$\mathbf{m}_\delta = \mathbf{0}_K$, $\mathbf{P}_\delta = 0.001 \cdot \mathbf{I}_L$
ξ	$\mathrm{p}\left(\frac{1}{\xi}\right) = \frac{b_\xi^{a_\xi}}{\Gamma(a_\xi)} \left(\frac{1}{\xi}\right)^{a_\xi - 1} e^{-b_\tau/\xi}$	$a_\xi = 0.001$, $b_\xi = 0.001$
γ	$\mathrm{p}(\gamma\lvert\xi) = \frac{\left(\frac{t_\gamma}{\xi}\right)^{1/2}}{(2\pi)^{1/2}} \left\{-\frac{t_\gamma}{2\xi}(\gamma - m_\gamma)^2\right\}$	$m_\gamma = 0$, $t_\gamma = 1$

Because ξ is a variance parameter, a Gamma prior is placed on the corresponding precision parameter, $\frac{1}{\xi}$. This is equivalent to placing and inverse-Gamma prior on ξ directly.

The prior for γ depends on the value of ξ: given ξ, γ follows a Normal distribution with mean m_γ and precision $\frac{t_\gamma}{\xi}$. This is done so that the prior uncertainty around γ scales along with the prior uncertainty around ξ.

Syntax

```
[<model name> = ] tobitII( y ~ x1 x2 ... xK | z1 z2 ... zL [,<options> ] );
```

where:

- `y` is the dependent variable name in the outcome equation, as it appears in the dataset used for estimation
- `x1 x2 ... xK` is a list of the K independent variable names in the outcome equation, as they appear in the dataset used for estimation; when a constant term is to be included in the model, this must be requested explicitly
- `z1 z2 ... zL` is a list of the L independent variable names in the selection equation, as they appear in the dataset used for estimation; when a constant term is to be included in the model, this must be requested explicitly

The dependent variable, `y`, in the dataset used for estimation must contain both numerical values and missing values ("nans"). The values of the variables in the `x` list are not used during estimation and they could be missing. However, observations with missing values in the `z` list are dropped prior to estimation.

The optional arguments for the type II Tobit model are:[2]

Gibbs parameters	
`"chains"`	number of chains to run in parallel (positive integer); the default value is 1
`"burnin"`	number of burn-in draws per chain (positive integer); the default value is 10000
`"draws"`	number of retained draws per chain (positive integer); the default value is 20000
`"thin"`	value of the thinning parameter (positive integer); the default value is 1
`"seed"`	value of the seed for the random-number generator (positive integer); the default value is 42
Hyperparameters	
`"m_beta"`	mean vector of the prior for $\boldsymbol{\beta}$ ($K\times 1$ vector); the default value is $\mathbf{0}_K$
`"P_beta"`	precision matrix of the prior for $\boldsymbol{\beta}$ ($K\times K$ symmetric and positive-definite matrix); the default value is $0.001\cdot\mathbf{I}_K$
`"m_delta"`	mean vector of the prior for $\boldsymbol{\delta}$ ($L\times 1$ vector); the default value is $\mathbf{0}_L$
`"P_delta"`	precision matrix of the prior for $\boldsymbol{\delta}$ ($L\times L$ symmetric and positive-definite matrix); the default value is $0.001\cdot\mathbf{I}_L$
`"a_xi"`	shape parameter of the prior for $\frac{1}{\xi}$ (positive number); the default value is 0.001
`"b_xi"`	rate parameter of the prior for $\frac{1}{\xi}$ (positive number); the default value is 0.001
`"m_gamma"`	mean of the prior for γ; the default value is 0
`"t_gamma"`	precision scaling parameter of the prior for γ (positive number); the default value is 1
Dataset and log-marginal likelihood	
`"dataset"`	the id value of the dataset that will be used for estimation; the default value is the first dataset in memory (in alphabetical order)
`"logML_CJ"`	boolean indicating whether the Chib (1995)/Chib & Jeliazkov (2001) approximation to the log-marginal likelihood should be calculated (**true**|**false**); the default value is **false**

[2]Optional arguments are always given in option-value pairs (eg. `"chains"=3`).

Reported Parameters

$\boldsymbol{\beta}$	`variable_name`	vector of parameters associated with the independent variables in the outcome equation
$\boldsymbol{\delta}$	`variable_name`	vector of parameters associated with the independent variables in the selection equation
ξ	`xi`	conditional variance parameter of the error term in the outcome equation, ε_i
γ	`gamma`	covariance of the error terms in the outcome and selection equations
σ_ε	`sigma_e`	standard deviation of the error term in the outcome equation: $\sigma_\varepsilon = \xi^{1/2}$
ρ	`rho`	correlation coefficient between the error terms in the outcome and selection equations: $\rho = \gamma/\xi^{1/2}$

Stored values and post-estimation analysis

If a left-hand-side id value is provided when a type II Tobit model is created, then the following results are saved in the model item and are accessible via the '.' operator:

`Samples`	a matrix containing the draws from the posterior of $\boldsymbol{\beta}$, $\boldsymbol{\delta}$, ξ and γ
`y$x1,...,y$xK`	vectors containing the draws from the posterior of the parameters associated with variables `x1,...,xK` (the names of these vectors are the names of the variables that were included in the right-hand side of the outcome equation of the model, prepended by `y$`, where `y` is the name of the dependent variable; this is done so that the samples on the parameters associated with a variable that appears in both `x` and `z` lists can be distinguished)
`s$z1,...,s$zL`	vectors containing the draws from the posterior of the parameters associated with variables `z1,...,zL` (the names of these vectors are the names of the variables that were included in the `z` list, in the right-hand side of the selection equation of the model, prepended by `s$`; this is done so that the samples on the parameters associated with a variable that appears in both `x` and `z` lists can be distinguished)
`xi`	vector containing the draws from the posterior of ξ
`gamma`	vector containing the draws from the posterior of γ
`logML`	the Lewis & Raftery (1997) approximation of the log-marginal likelihood
`logML_CJ`	the Chib (1995)/Chib & Jeliazkov (2001) approximation to the log-marginal likelihood; this is available only if the model was estimated with the `"logML_CJ"=true` option
`nchains`	the number of chains that were used to estimate the model
`nburnin`	the number of burn-in draws per chain that were used when estimating the model
`ndraws`	the total number of retained draws from the posterior (=`chains` · `draws`)
`nthin`	value of the thinning parameter that was used when estimating the model
`nseed`	value of the seed for the random-number generator that was used when estimating the model

Additionally, the following functions are available for post-estimation analysis (see section B.14):

- `diagnostics()`
- `pmp()`
- `test()`
- `mfx()`

Usually the marginal effects of primary importance in a type II Tobit model are the effects of changes in the independent variables in the outcome equation on the expected value of the dependent variable in the same equation, *for the entire population* (whether selected or not). These effects, at least for variables included linearly in the model, are the corresponding βs.

Nevertheless, there are two additional types of marginal effects that could be of interest and which are not linear functions of the model's parameters. The type II Tobit model uses the `mfx()` function to calculate and report the marginal effects of:

- the variables in the `z` list on the probability of selection: $\frac{\partial \text{Prob}(s_i=1|\mathbf{z}_i)}{\partial z_{i\ell}}$
- the variables in the `x` list on the expected value of the response variable for the part of the population that is selected[3]: $\frac{\partial \text{E}(y_i|\mathbf{x}_i,\mathbf{z}_i,s_i=1)}{\partial x_{ik}}$

The two types of marginal effects can be requested by setting the `"type"` argument of the `mfx()` function equal to `1` or `3`. The generic syntax for a statement involving the `mfx()` function after estimation of a type II Tobit model is:

```
mfx( ["type"=1] [, "point"=<point of calculation>] [, "model"=<model name>] );
```

and:

```
mfx( "type"=2 [, "point"=<point of calculation>] [, "model"=<model name>] );
```

for calculation of the marginal effects on $\text{Prob}(s=1)$ and on $\text{E}(y|s=1)$. The default value of the `"type"` option is `1`. See the general documentation of the `mfx()` function (section B.14) for details on the other optional arguments.

Examples

Example 1

```
myData = import("$BayESHOME/Datasets/dataset8.csv");
myData.constant = ones(rows(myData), 1);

tobitII( y2 ~ constant x1 x2 x3 x4 | constant x1 x2 x3 z1 z2 );
```

Example 2

```
myData = import("$BayESHOME/Datasets/dataset8.csv");
myData.constant = ones(rows(myData), 1);

myModel = tobitII( y2 ~ constant x1 x2 x3 x4 | constant x1 x2 x3 x4 z1 z2,
    "m_beta"=ones(5,1), "P_beta" = 0.1*eye(5,5),
    "m_delta"=ones(7,1), "P_delta" = 0.1*eye(7,7),
    "a_xi"=0.01, "b_xi"=0.01, "m_gamma"=0.0, "t_gamma"=0.1,
    "burnin"=10000, "draws"=40000, "thin"=4, "chains"=2,
    "logML_CJ" = true, "dataset"=myData);

diagnostics("model"=myModel);

mfx("type"=1,"point"="mean","model"=myModel);
mfx("type"=2,"point"="mean","model"=myModel);
```

[3] If the k-th independent variable in the outcome equation does not appear also as an independent variable in the selection equation then its marginal effect is simply β_k.

Chapter 10

Linear Systems of Equations

10.1 Simple Seemingly Unrelated Regressions (SUR)

Mathematical representation

$$\begin{array}{ccccc} y_{1i} & = & \mathbf{x}_{1i}'\boldsymbol{\beta}_1 & + & \varepsilon_{1i} \\ y_{2i} & = & \mathbf{x}_{2i}'\boldsymbol{\beta}_2 & + & \varepsilon_{2i} \\ \vdots & & \vdots & & \vdots \\ y_{Mi} & = & \mathbf{x}_{Mi}'\boldsymbol{\beta}_M & + & \varepsilon_{Mi} \end{array}$$

- the model consists of M equations
- the model is estimated using N observations ($i = 1, 2, \ldots, N$)
- y_{mi} is the value of equation m's dependent variable for observation i
- $\mathbf{x}_{mi}$ is a $K_m \times 1$ vector that stores the values of the K_m independent variables for observation i, as they appear in equation m
- the same independent variable can appear in multiple equations, associated with different coefficients
- $\boldsymbol{\beta}_m$ is a $K_m \times 1$ vector of parameters associated with equation m's independent variables
- in total, there are $K = \sum_{m=1}^{M} K_m$ β slope parameters to be estimated
- the M error terms jointly follow a multivariate Normal distribution with mean $\mathbf{0}$ and precision matrix $\boldsymbol{\Omega}$

An equivalent and more compact representation of the model is:

$$\mathbf{y}_i = \mathbf{X}_i\boldsymbol{\beta} + \boldsymbol{\varepsilon}_i, \qquad \boldsymbol{\varepsilon}_i \sim \mathrm{N}\left(\mathbf{0}, \boldsymbol{\Omega}^{-1}\right)$$

where:

$$\underset{M\times 1}{\mathbf{y}_i} = \begin{bmatrix} y_{1i} \\ y_{2i} \\ \vdots \\ y_{Mi} \end{bmatrix}, \quad \underset{M\times K}{\mathbf{X}_i} = \begin{bmatrix} \mathbf{x}_{1i}' & \mathbf{0} & \ldots & \mathbf{0} \\ \mathbf{0} & \mathbf{x}_{2i}' & \ldots & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{0} & \mathbf{0} & \ldots & \mathbf{x}_{Mi}' \end{bmatrix}, \quad \underset{K\times 1}{\boldsymbol{\beta}} = \begin{bmatrix} \boldsymbol{\beta}_1 \\ \boldsymbol{\beta}_2 \\ \vdots \\ \boldsymbol{\beta}_M \end{bmatrix}, \quad \underset{M\times 1}{\boldsymbol{\varepsilon}_i} = \begin{bmatrix} \varepsilon_{1i} \\ \varepsilon_{2i} \\ \vdots \\ \varepsilon_{Mi} \end{bmatrix}$$

Priors

Parameter	Probability density function	Default hyperparameters
$\boldsymbol{\beta}$	$\mathrm{p}(\boldsymbol{\beta}) = \frac{\lvert\mathbf{P}\rvert^{1/2}}{(2\pi)^{K/2}} \exp\left\{-\frac{1}{2}(\boldsymbol{\beta}-\mathbf{m})'\mathbf{P}(\boldsymbol{\beta}-\mathbf{m})\right\}$	$\mathbf{m} = \mathbf{0}_K$, $\mathbf{P} = 0.001 \cdot \mathbf{I}_K$
$\boldsymbol{\Omega}$	$\mathrm{p}(\boldsymbol{\Omega}) = \frac{\lvert\boldsymbol{\Omega}\rvert^{\frac{n-M-1}{2}} \lvert\mathbf{V}^{-1}\rvert^{n/2}}{2^{nM/2}\Gamma_M\left(\frac{n}{2}\right)} \exp\left\{-\frac{1}{2}\mathrm{tr}\left(\mathbf{V}^{-1}\boldsymbol{\Omega}\right)\right\}$	$n = M^2$, $\mathbf{V} = \frac{100}{M} \cdot \mathbf{I}_M$

Syntax

```
[<model name> = ] sur( {
      y1 ~ x11 x12 ... x1K1,
      y2 ~ x21 x22 ... x2K2,
      ...,
      yM ~ xM1 xM2 ... xMKM }
      [, <options> ]
);
```

where:

- `y1`, `y2`, ..., `yM` are the dependent variable names, as they appear in the dataset used for estimation

- `xm1 xm2 ... xmK`$_m$ is a list of the K_m independent variable names for equation $m = 1, 2, \ldots, M$, as they appear in the dataset used for estimation; when a constant term is to be included in an equation, this must be requested explicitly; M such lists must be provided

The optional arguments for the Seemingly Unrelated Regressions model are:[1]

`Gibbs parameters`	
`"chains"`	number of chains to run in parallel (positive integer); the default value is 1
`"burnin"`	number of burn-in draws per chain (positive integer); the default value is 10000
`"draws"`	number of retained draws per chain (positive integer); the default value is 20000
`"thin"`	value of the thinning parameter (positive integer); the default value is 1
`"seed"`	value of the seed for the random-number generator (positive integer); the default value is 42
`Hyperparameters`	
`"m"`	mean vector of the prior for $\boldsymbol{\beta}$ ($K \times 1$ vector); the default value is $\mathbf{0}_K$
`"P"`	precision matrix of the prior for $\boldsymbol{\beta}$ ($K \times K$ symmetric and positive-definite matrix); the default value is $0.001 \cdot \mathbf{I}_K$
`"V"`	scale matrix of the prior for Ω ($M \times M$ symmetric and positive-definite matrix); the default value is $\frac{100}{M} \cdot \mathbf{I}_M$
`"n"`	degrees-of-freedom parameter of the prior for $\boldsymbol{\Omega}$ (real number greater than or equal to M); the default value is M^2
`Dataset and log-marginal likelihood`	
`"dataset"`	the id value of the dataset that will be used for estimation; the default value is the first dataset in memory (in alphabetical order)
`"logML_CJ"`	boolean indicating whether the Chib (1995)/Chib & Jeliazkov (2001) approximation to the log-marginal likelihood should be calculated (**true**|**false**); the default value is **false**

Reported Parameters

$\boldsymbol{\beta}$	`variable_name`	vector of parameters associated with the independent variables; these are broken into groups according to the equation in which the independent variables appear

Stored values and post-estimation analysis

If a left-hand-side id value is provided when a SUR model is created, then the following results are saved in the model item and are accessible via the '.' operator:

`Samples`	a matrix containing the draws from the posterior of $\boldsymbol{\beta}$ (across all equations, starting from the first equation) and the unique elements of $\boldsymbol{\Omega}$
`ym$xm1,...,` `ym$xmK`$_m$	vectors containing the draws from the posterior of the parameters associated with variables `xm1,...,xmK`$_m$, for $m = 1, 2, \ldots, M$ (the names of these vectors are the names of the variables that were included in the right-hand side of equation m, prepended by `ym$`, where `ym` is the name of the dependent variable in equation m; this is done so that the samples on the parameters associated with a variable that appears in more than one equations can be distinguished)

[1] Optional arguments are always given in option-value pairs (eg. `"chains"=3`).

`Omega_i_j`	vectors containing the draws from the posterior of the unique elements of $\mathbf{\Omega}$; because $\mathbf{\Omega}$ is symmetric, only $\frac{(M-1)M}{2}+M$ of its elements are stored (instead of all M^2 elements); `i` and `j` index the row and column of $\mathbf{\Omega}$, respectively, at which the corresponding element is located
`Omega`	$M \times M$ matrix that stores the posterior mean of $\mathbf{\Omega}$
`logML`	the Lewis & Raftery (1997) approximation of the log-marginal likelihood
`logML_CJ`	the Chib (1995)/Chib & Jeliazkov (2001) approximation to the log-marginal likelihood; this is available only if the model was estimated with the `"logML_CJ"=true` option
`nchains`	the number of chains that were used to estimate the model
`nburnin`	the number of burn-in draws per chain that were used when estimating the model
`ndraws`	the total number of retained draws from the posterior (=`chains` · `draws`)
`nthin`	value of the thinning parameter that was used when estimating the model
`nseed`	value of the seed for the random-number generator that was used when estimating the model

Additionally, the following functions are available for post-estimation analysis (see section B.14):

- `diagnostics()`
- `pmp()`
- `test()`

Examples

Example 1

```
myData = import("$BayESHOME/Datasets/dataset5.csv");
myData.constant = ones(rows(myData), 1);

model1 = sur( {
        y1 ~ constant x1 x2 x3 x4 x5 x6 x7 x8 x9 x10,
        y2 ~ constant x1 x2 x3,
        y3 ~ constant x1 x2 x3
        } );
```

Example 2

```
myData = import("$BayESHOME/Datasets/dataset5.csv");
myData.constant = ones(rows(myData), 1);

model1 = sur( {
        y1 ~ constant x1 x2 x3 x4 x5 x6 x7 x8 x9 x10,
        y2 ~ constant x1 x2 x3,
        y3 ~ constant x1 x2 x3
        }, "logML_CJ"=true );

print(mean([model1.y1$x1-model1.y2$constant,
    model1.y1$x2-model1.y3$constant]));

model2 = sur( {
        y1 ~ constant x1 x2 x3 x4 x5 x6 x7 x8 x9 x10,
        y2 ~ constant x1 x2 x3,
        y3 ~ constant x1 x2 x3
        },
    "constraints" = {
        y1$x1-y2$constant=0, y1$x5-0.5*y2$x1=0, y1$x6-y2$x2=0,
            y1$x7-y2$x3=0,
        y1$x2-y3$constant=0, y1$x6-y3$x1=0, y1$x8-0.5*y3$x2=0,
            y1$x9-y3$x3=0,
    },
    "Xi" = 1e7*eye(8,8), "logML_CJ"=true );

print(mean([model2.y1$x1-model2.y2$constant,
    model2.y1$x2-model2.y3$constant]));

pmp( {model1, model2} );
pmp( {model2, model2}, "logML_CJ" = true);
```

Chapter 11

Vector Autoregressive and Vector Error-Correction Models

11.1 Vector Autoregressive (VAR) model for time-series data

Mathematical representation

$$\mathbf{y}_t = \mathbf{A}_0 \mathbf{w}_t + \sum_{j=1}^{p} \mathbf{A}_j \mathbf{y}_{t-j} + \boldsymbol{\varepsilon}_t, \qquad \boldsymbol{\varepsilon}_t \sim \mathrm{N}\left(\mathbf{0}, \boldsymbol{\Omega}^{-1}\right)$$

where:

- the model contains M endogenous variables (ys)
- the model is estimated using T observations ($t = 1, 2, \ldots, T$)
- $\mathbf{y}_t$ is an $M \times 1$ vector, which contains the values of the M endogenous variables at time t
- $\mathbf{w}_t$ is a $K \times 1$ vector, which contains the values of the exogenous variables (common to all equations) at time t
- $\mathbf{A}_0$ is an $M \times K$ matrix of parameters associated with the exogenous variables
- $\mathbf{A}_j$ is an $M \times M$ matrix of parameters, $j = 1, 2 \ldots, p$
- in total, there are $L = (M \cdot p + K) \cdot M$ slope parameters to be estimated (in all $\mathbf{A}$s)
- $\boldsymbol{\varepsilon}_t$ is an $M \times 1$ vector of errors at time t, which follows a multivariate Normal distribution with mean $\mathbf{0}$ and precision matrix $\boldsymbol{\Omega}$

With T time observations, an equivalent representation of the model is:

$$\mathbf{Y} = \mathbf{X}\mathbf{A} + \mathbf{E}$$

where:

$$\underset{(T-p)\times M}{\mathbf{Y}} = \begin{bmatrix} \mathbf{y}'_{p+1} \\ \mathbf{y}'_{p+2} \\ \vdots \\ \mathbf{y}'_T \end{bmatrix} \qquad \underset{(T-p)\times M}{\mathbf{E}} = \begin{bmatrix} \boldsymbol{\varepsilon}'_{p+1} \\ \boldsymbol{\varepsilon}'_{p+2} \\ \vdots \\ \boldsymbol{\varepsilon}'_T \end{bmatrix} \qquad \underset{(M\cdot p+K)\times M}{\mathbf{A}} = \begin{bmatrix} \mathbf{A}'_0 \\ \mathbf{A}'_1 \\ \vdots \\ \mathbf{A}'_p \end{bmatrix}$$

and:

$$\underset{(T-p)\times(K+M\cdot p)}{\mathbf{X}} = \begin{bmatrix} \mathbf{w}'_{p+1} & \mathbf{y}'_p & \mathbf{y}'_{p-1} & \cdots & \mathbf{y}'_1 \\ \mathbf{w}'_{p+2} & \mathbf{y}'_{p+1} & \mathbf{y}'_p & \cdots & \mathbf{y}'_2 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \mathbf{w}'_T & \mathbf{y}'_{T-1} & \mathbf{y}'_{T-2} & \cdots & \mathbf{y}'_{T-p} \end{bmatrix}$$

In this representation the endogenous variables are expanded over the columns of $\mathbf{Y}$ and the time observations are stacked one under the other.

Yet another representation is:

$$\mathbf{y} = (\mathbf{I}_M \otimes \mathbf{X})\,\boldsymbol{\alpha} + \boldsymbol{\varepsilon}, \qquad \boldsymbol{\varepsilon} \sim \mathrm{N}\left(\mathbf{0}, (\boldsymbol{\Omega} \otimes \mathbf{I}_M)^{-1}\right)$$

where:

- $\mathbf{y}$ is the $(T-p) \cdot M \times 1$ obtained by stacking the columns of $\mathbf{Y}$
- $\boldsymbol{\varepsilon}$ is the $(T-p) \cdot M \times 1$ obtained by stacking the columns of $\mathbf{E}$
- $\boldsymbol{\alpha}$ is the $(M \cdot p + K) \cdot M \times 1$ vector obtained by stacking the columns of $\mathbf{A}$

In this representation the data are first stacked over time and then by variable.

Priors

Parameter	Probability density function	Default hyperparameters
$\boldsymbol{\alpha}$	$\mathrm{p}(\boldsymbol{\alpha}) = \frac{\lvert\mathbf{P}\rvert^{1/2}}{(2\pi)^{L/2}} \exp\left\{-\frac{1}{2}(\boldsymbol{\alpha}-\mathbf{m})'\mathbf{P}(\boldsymbol{\alpha}-\mathbf{m})\right\}$	$\mathbf{m} = \mathbf{0}_L$, $\mathbf{P} = 0.001 \cdot \mathbf{I}_L$
$\boldsymbol{\Omega}$	$\mathrm{p}(\boldsymbol{\Omega}) = \frac{\lvert\boldsymbol{\Omega}\rvert^{\frac{n-M-1}{2}} \lvert\mathbf{V}^{-1}\rvert^{n/2}}{2^{nM/2}\Gamma_M\left(\frac{n}{2}\right)} \exp\left\{-\frac{1}{2}\mathrm{tr}\left(\mathbf{V}^{-1}\boldsymbol{\Omega}\right)\right\}$	$n = M^2$, $\mathbf{V} = \frac{100}{M} \cdot \mathbf{I}_M$

Syntax

```
[<model name> = ] varm( { y1, y2, ..., yM } [ ~ w1 w2 ... wK ] [, <options> ] );
```

where:

- `y1, y2, ..., yM` are the names of the endogenous variables, as they appear in the dataset used for estimation
- `w1 w2 ... wK` is a list of the K exogenous variable names, as they appear in the dataset used for estimation; these variables will be included in all equations, but if there is need to restrict the model so that of any of them do not appear in an equation, this can be achieved by restricting the associated parameters in the prior; when a constant term is to be included in the model, this must be requested explicitly

Before using the `varm()` function the dataset used for estimation must be declared as a time-series dataset using the `set_ts()` function (see section B.13).

The optional arguments for the vector autoregressive model are:[1]

`Specification of lags`	
`"lags"`	a vector of integers (either row or column vector) indicating the lags of the endogenous variables to be included in the right-hand side of the model; for example, if this vector is set equal to `[1, 4, 9]`, then the 1st, 4th and 9th lags are included; the default value is 1, in which case, only the first lag of all endogenous are included
`Gibbs parameters`	
`"chains"`	number of chains to run in parallel (positive integer); the default value is 1
`"burnin"`	number of burn-in draws per chain (positive integer); the default value is 10000
`"draws"`	number of retained draws per chain (positive integer); the default value is 20000
`"thin"`	value of the thinning parameter (positive integer); the default value is 1
`"seed"`	value of the seed for the random-number generator (positive integer); the default value is 42
`Hyperparameters`	
`"m"`	mean vector of the prior for $\boldsymbol{\alpha}$ ($L \times 1$ vector); the default value is $\mathbf{0}_L$
`"P"`	precision matrix of the prior for $\boldsymbol{\alpha}$ ($L \times L$ symmetric and positive-definite matrix); the default value is $0.001 \cdot \mathbf{I}_L$
`"V"`	scale matrix of the prior for Ω ($M \times M$ symmetric and positive-definite matrix); the default value is $\frac{100}{M} \cdot \mathbf{I}_M$
`"n"`	degrees-of-freedom parameter of the prior for $\boldsymbol{\Omega}$ (real number greater than or equal to M); the default value is M^2

[1] Optional arguments are always given in option-value pairs (eg. `"chains"=3`).

Dataset and log-marginal likelihood	
`"dataset"`	the id value of the dataset that will be used for estimation; the default value is the first dataset in memory (in alphabetical order)
`"logML_CJ"`	boolean indicating whether the Chib (1995)/Chib & Jeliazkov (2001) approximation to the log-marginal likelihood should be calculated (**true**|**false**); the default value is **false**

Reported Parameters

$\boldsymbol{\alpha}$	`variable_name`	vector of parameters associated with the exogenous variables and the lags of the endogenous variables; these are broken into groups according to the equation in which the variables appear and the parameters associated with the exogenous variables are listed first

Stored values and post-estimation analysis

If a left-hand-side id value is provided when a VAR model is created, then the following results are saved in the model item and are accessible via the '.' operator:

`Samples`	a matrix containing the draws from the posterior of $\boldsymbol{\alpha}$ (across all equations, starting from the first equation) and the unique elements of $\boldsymbol{\Omega}$
`ym$w1,...,ym$wK`	vectors containing the draws from the posterior of the parameters associated with the exogenous variables `w1,...,wK`, for $m = 1, 2, \ldots, M$ (the names of these vectors are the names of the exogenous variables that were included in the model, prepended by `ym$`, where `ym` is the name of the dependent variable in equation m)
`ym$yl_tms,...`	vectors containing the draws from the posterior of the parameters associated with the lags of the endogenous variables for $m, l = 1, 2, \ldots, M$ (the names of these vectors are the names of the exogenous variables, appended by `_tms`, where `s` is the $\texttt{s}^{\text{th}}$ lag, and prepended by `ym$`, where `ym` is the name of the dependent variable in equation m)
`Omega_i_j`	vectors containing the draws from the posterior of the unique elements of $\boldsymbol{\Omega}$; because $\boldsymbol{\Omega}$ is symmetric, only $\frac{(M-1)M}{2} + M$ of its elements are stored (instead of all M^2 elements); `i` and `j` index the row and column of $\boldsymbol{\Omega}$, respectively, at which the corresponding element is located
`Omega`	$M \times M$ matrix that stores the posterior mean of $\boldsymbol{\Omega}$
`logML`	the Lewis & Raftery (1997) approximation of the log-marginal likelihood
`logML_CJ`	the Chib (1995)/Chib & Jeliazkov (2001) approximation to the log-marginal likelihood; this is available only if the model was estimated with the `"logML_CJ"`=**true** option
`nchains`	the number of chains that were used to estimate the model
`nburnin`	the number of burn-in draws per chain that were used when estimating the model
`ndraws`	the total number of retained draws from the posterior (=`chains` · `draws`)
`nthin`	value of the thinning parameter that was used when estimating the model
`nseed`	value of the seed for the random-number generator that was used when estimating the model

Additionally, the following functions are available for post-estimation analysis (see section B.14):

- `diagnostics()`
- `pmp()`
- `irf()`
- `test()`
- `forecast()`

Examples

Example 1

```
myData = import("$BayESHOME/Datasets/dataset6.csv", ",");
myData.constant = 1;
set_ts( time, "dataset"=myData);

varm( {y1,y2,y3} ~ constant w );
```

Example 2

```
myData = import("$BayESHOME/Datasets/dataset6.csv", ",");
myData.constant = 1;
set_ts( time, "dataset"=myData);

myModel = varm( {y1,y2,y3} ~ constant w );

forecast( "horizon"=10, "model"=myModel );

irf( "horizon"=20, "model"=myModel );
```

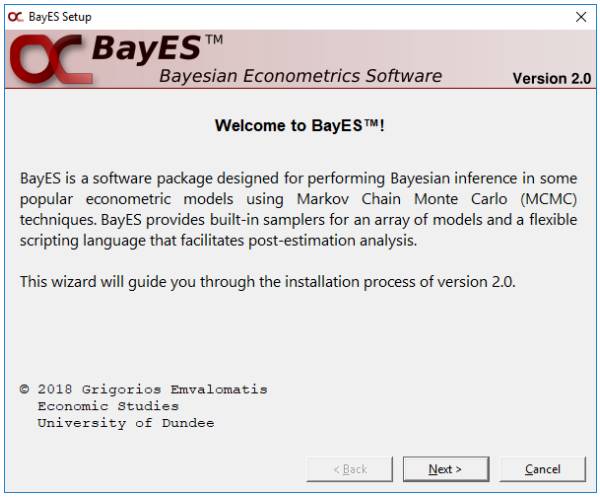
Appendix A

Installation Guide

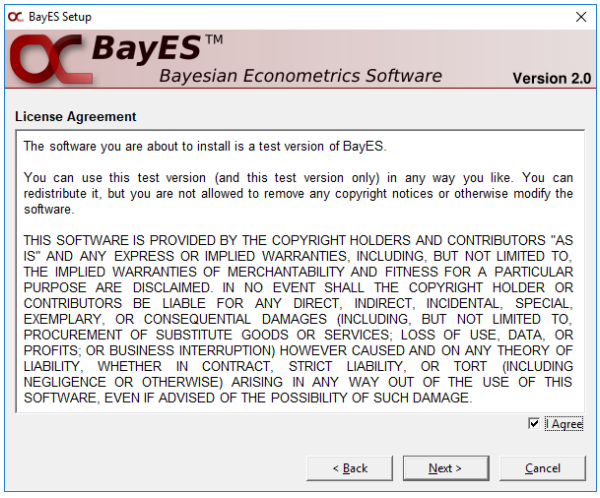
A.1 Installation under Microsoft® Windows®

The Microsoft® Windows® installer is a self extracting executable named `BayES_WinXX.exe`, where '`XX`' could be either '32' or '64', for 32-bit and 64-bit machines, respectively. To initiate the installation process double-click on the installer and accept any prompts from the system. After extracting the contents of the archive you should be able to see the first page of the setup wizard (see figure below). From now on you should follow the instructions on this wizard:

1. Click on the `Next>` button on the welcome page of the wizard to proceed:



2. Read the license agreement and either check the "`I Agree`" checkbox and click `Next>` to proceed or `Cancel` to cancel the installation process:

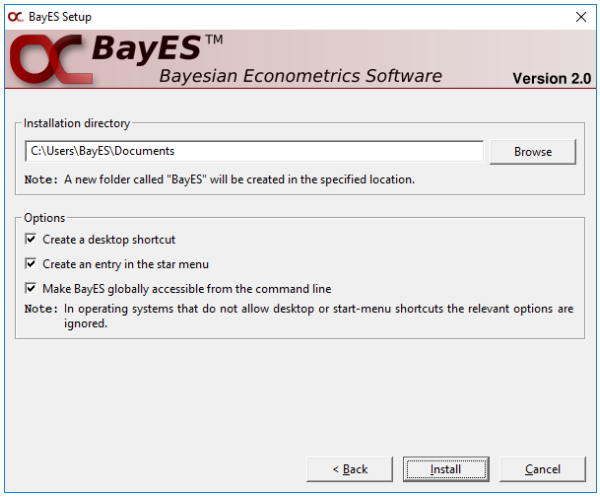


3. On the next screen you need to provide the location where BayES will place its system files (libraries, icons, initialization files, etc.), documents, datasets and sample script files. The location provided must exist on the machine's file system and the user must have write access on it. If a location is chosen for installation on which you do not have write permissions (for example `C:\Program Files`), the installer will issue a warning and a prompt to either:

   (a) pick a different installation location, or

   (b) quit the installer and rerun it with administrative rights[1]

   When BayES is installed in a system folder, users without administrative rights will not have permission to alter any of its files. This is reasonable for most purposes, as it keeps BayES' system files secure. However, a user without administrative rights will not be able to make persistent changes to the initialization files (for example, to store his/her preferences regarding prompting to save files on exit or specify the location of external binaries). If a system administrator wants to make such changes then he/she should run BayES as an administrator.



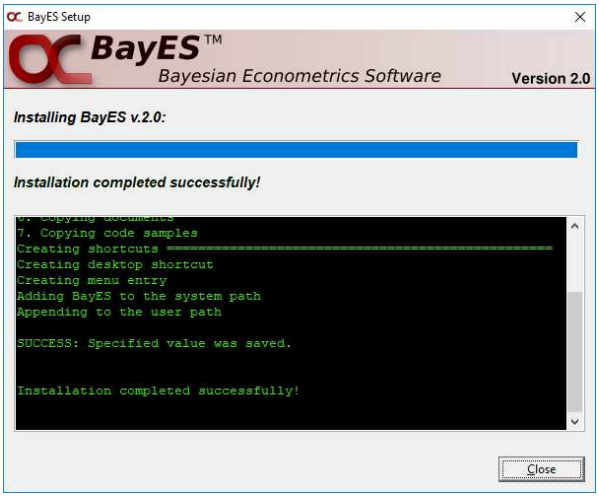
   There are three additional options available on this page:

   - if the first checkbox is checked the installer will attempt to create a shortcut to the BayES executable on the system's desktop
   - if the second checkbox is checked the installer will attempt to create a shortcut to the BayES executable on the system's "start" menu
   - if the third checkbox is checked the installer will attempt to make BayES accessible from the command line. This means that BayES will be available from the system's shell by typing `BayES`, optionally followed by a script file to be executed.

   Finally, click on `Install` to get the installer to commit changes to the system.

4. Once the installation completes you should see the following screen. Look at the install log on this page in case something went wrong during the installation and act accordingly. Click `Close` to quit the installer. If everything went fine you should now be ready to use BayES.

[1] To run the installer with administrative rights right-click on the `BayES_WinXX.exe` file and pick "`Run as administrator`" from the pop-up menu.



A.2 Installation under Linux

The Linux installer is a self-extracting binary file named `BayES_LinuxXX.bsx`, where '`XX`' could be either '32' or '64', for 32-bit and 64-bit machines, respectively. The installation process can be initiated after making this file executable using either the system's command shell or its file browser:

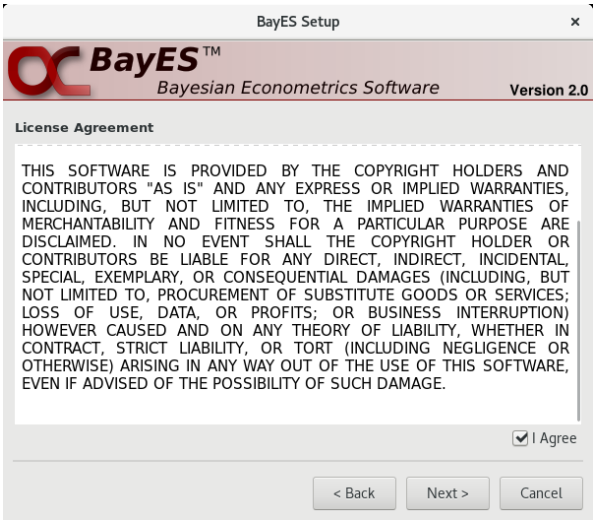
- from the system's command shell change directory to where the installer is located and execute: `chmod u+x BayES_LinuxXX.bsx` (replacing '`XX`' with either '32' or '64'), or
- from the system's file browser navigate to the location where the installer is located and right-click on it. In the pop-up menu click on `Properties` and in the pop-up window select the option to make this file executable for the current user.[2]

Next, run `./BayES_LinuxXX.bsx` from the command shell (replacing '`XX`' with either '32' or '64') This should take you to the first screen of the setup wizard (see the following figure). From now on you should follow the instructions on the setup wizard:

1. Click on the `Next>` button on the welcome page of the wizard:

[2]Depending on the Linux distribution installed on the system, the steps may differ, but most modern Linux distributions that support a GUI, allow making files executable from the GUI.

2. Read the license agreement and either check the "`I Agree`" checkbox and click `Next>` to proceed or `Cancel` to cancel the installation process:



3. On the next screen you need to provide a location where BayES will place its documents, datasets and sample script files. Under Linux the BayES system files (libraries, icons and initialization files) are always placed in the user's home directory in a folder called `.BayES` (hidden).

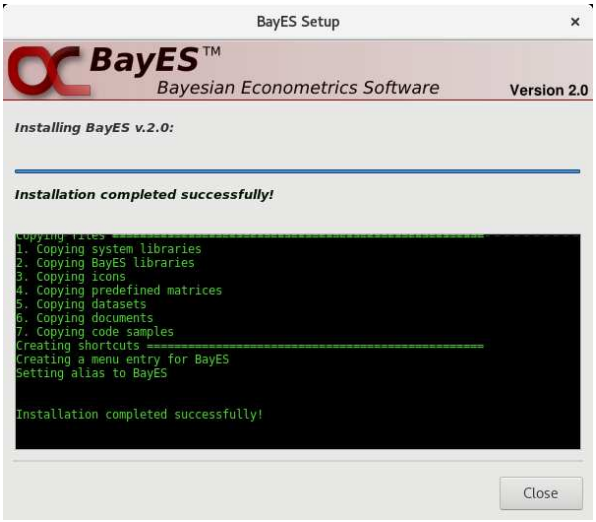
   There are three additional options available on this page:

   - if the first checkbox is checked the installer will attempt to create a shortcut to the BayES binary on the system's desktop[3]
   - if the second checkbox is checked the installer will attempt to create a shortcut to the BayES binary on the system's "start" menu
   - if the third checkbox is checked the installer will attempt to make BayES accessible from the command line. This means that BayES will be available from the system's shell by typing `BayES`, optionally followed by a script file to be executed. This is achieved by creating an entry in the user's `.bashrc` file. If such a file does not already exist, the installer will create one.

   Finally, click on `Install` to get the installer to commit changes to the system.

[3]Under some Linux distributions you may have to "trust and run" the desktop file the first time you invoke BayES through it.

4. Once the installation completes you should see the following screen. Look at the install log on this page in case something went wrong during installation and act accordingly. Click `Close` to quit the installer.



In most Linux systems the user needs to either logout and login again for all changes to take effect or issue the command:

`source ~/.bash_rc`

from the system's command shell.

A.3 Installation under macOS

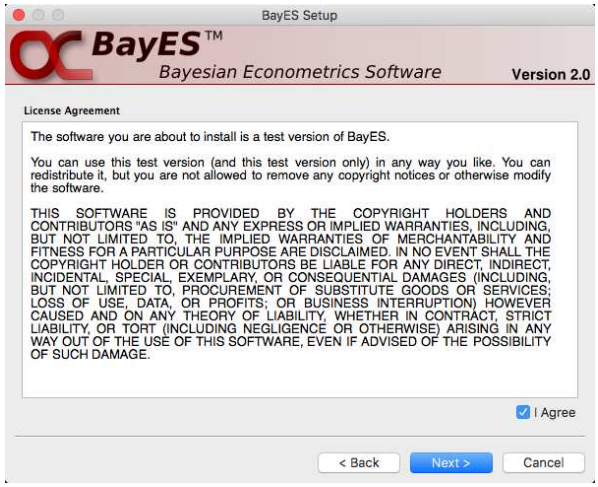
The macOS installer is a self-extracting binary file named `BayES_macOS64.bsx`. The installation process can be initiated by opening a terminal window, navigating to the location where the installer was downloaded (using the `cd ...` command) and issuing the command:

```
sh ./BayES_macOS64.bsx
```

This should fire-up the setup wizard and take you to its first screen (see the following figure). From now on you should follow the instructions on the setup wizard:

1. Click on the `Next>` button on the welcome page of the wizard:

2. Read the license agreement and either check the "`I Agree`" checkbox and click `Next>` to proceed or `Cancel` to cancel the installation process:



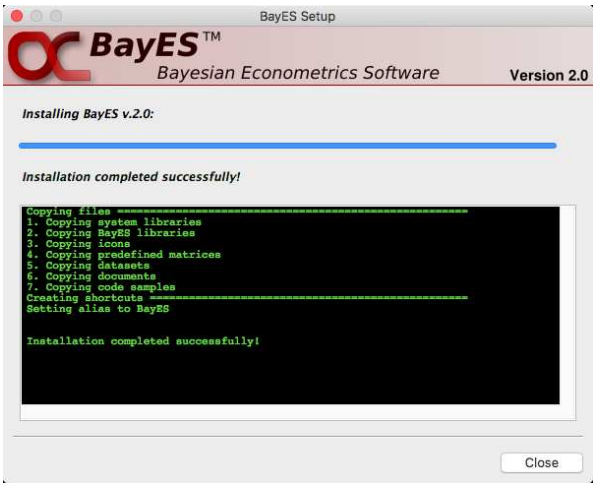
3. On the next screen you need to provide a location where BayES will place its documents, datasets and sample script files. Under macOS the BayES system files (libraries, icons and initialization files) are always placed in the user's `Applications` directory, in a folder called `BayES`. This automatically makes BayES accessible via the Launchpad.

   There are three additional options available on this page:

   - if the first checkbox is checked the installer will attempt to create a shortcut to the BayES binary on the system's desktop (this option is disabled for macOS systems and no such shortcut on the desktop is created)
   - if the second checkbox is checked the installer will create a shortcut to the BayES binary on the Launchpad (this option is disabled for macOS systems and BayES is always made accessible via the Launchpad)
   - if the third checkbox is checked the installer will attempt to make BayES accessible from the command line. This means that BayES will be available from the system's shell by typing `BayES`, optionally followed by a script file to be executed. This is achieved by creating an entry in the user's `.bash_profile` file. If such a file does not already exist, the installer will create one.

   Finally, click on `Install` to get the installer to commit changes to the system.

4. Once the installation completes you should see the following screen. Look at the install log on this page in case something went wrong during installation and act accordingly. Click `Close` to quit the installer. If everything went fine you should now be ready to use BayES.



If BayES was requested during installation to be made globally accessible through the command line the user needs to either logout and login again for this to take effect or issue the command:

```
source ~/.bash_profile
```

on the terminal window.

A.4 Unstalling BayES

To remove BayES from the system go to the directory where BayES was installed and double-click on the `Uninstall BayES` link.[4] After receiving confirmation from the user, the wizard will initiate the removal process. If the BayES installer had created an entry in the system's "start" menu during installation (under Microsoft® Windows® only), an `Uninstall BayES` option should also be available there. If BayES was installed in a system folder the uninstaller must be run with elevated rights (as an administrator).

[4] You can get the location of the installation directory by running `print("$BayESHOME");` from the BayES script editor window. The installation directory will be printed on BayES' main console.

Appendix B

List of functions and commands

Statements in this appendix are presented using the following generic syntax:

```
[A, B] = functionName(X [, Y, Z ]);
```

Function arguments given in *italics* inside thin square brackets ([...]) are optional. The right-hand side of the statement given above indicates that function `functionName` takes one mandatory argument, `X`, and two optional arguments, *Y* and *Z*. The order in which these optional arguments appear in the function call matters: to pass *Z* to `functionName` one must also pass *Y*.

Functions could have no return value (eg. **setwd**), one return value (eg. `exp`) or more than one return values (eg. `eig`). When a function has only one return value, then this value must not be enclosed in square brackets in the left-hand side of the calling statement. If a function has more than one return values, but only the first one is required, this can be requested using:

```
A = functionName(X [, Y, Z ]);
```

As with optional arguments, the order in which optional return values appear in a function call matters: to get `B` by calling `functionName`, `A` must be returned as well, as shown in the first statement above.

B.1 Directory statements

Directory statements are used to set the working directory for BayES and the installation directories of programs for which BayES provides interfaces. Under Microsoft® Windows® systems the directory separator can be either forward slash (/) or a backslash (\). However, BayES functions that return directory names will use forward slash separators, even under Microsoft® Windows® systems.

Syntax	Arguments and performed function
`setwd(d);`	Sets the current working directory to the value provided in `d`. Specification of the working directory allows calling user-defined functions from the directory and loading/saving results without needing to quote full paths. • `d` must be a string • `d` could be absolute (eg. `"C:/MyFiles/MyFolder"`) or relative to the current working directory (eg. `"./MyFolder"`)
`d = getwd();`	`d` is a string with value equal to the current working directory. • Even under Microsoft® Windows®systems the function returns a string that uses forward slash (/) to separate directories

table continues on next page

table continued from previous page

Syntax	Arguments and performed function
setbinary(<program name>, s);	This function sets the location and name of the executable/binary file of external programs for which BayES provides interfaces. • <program name> must one of the following ids: – jags – rproject – matlab – openbugs – stata – octave • s must be a string containing the full absolute path to the respective program's binary file, the name of the binary file and its extension

B.2 Console statements

The following statements are used to print on the BayES console or clear the console.

Syntax	Arguments and performed function
clc();	Clears the BayES console. • This function has no effect if BayES is run without the GUI (batch mode)
print(A [, c]);	Prints A on the BayES console. A could be any of the following data types: • matrix • model • string • dataset • structure If A is a string and c (if provided) is a 1×3 vector of values between 0 and 1, then the string is printed on the BayES console in the color defined by the values in c, in a red-green-blue (rgb) scale. If A is not a string and c is provided, then c is ignored and a warning is printed on the console.
who([A]);	When the function is called without an argument it prints on the BayES console a list of the id values of items in memory in the current workspace. If A is provided then it prints a list of the elements/variable names of A. These items/elements are reported first by data type in the following order: 1. matrices 3. models 5. strings 2. datasets 4. structures and then alphabetically, according to their id value. • A must be a dataset, model or structure • if A is a dataset then the function prints a list of the variable names containted in A

B.3 Elapsed-time statements

The following statements are used to measure and report the time that elapses between two points (defined by the user) in the execution of code.

Syntax	Arguments and performed function
tic([*s*]);	Starts a timer that can be used to measure the amount of time that elapses until a **toc**([*s*]); statement is encountered. If *s* is provided then the corresponding call to **toc**() must use the same string as identifier. • *s* must be a string
toc([*s*]);	Prints information on the time that elapsed between a call to the **tic**() function and the current call to **toc**(). If *s* is provided then it must be the same as the string used in the call of a previous **tic**. In this case **toc**(s); prints the amount of time that elapsed between the **tic**(s); and the **toc**(s); statements. If **toc** is called without an argument then it prints the time that elapsed from the previous call to **tic**() (without an argument as well). With this scheme BayES allows setting multiple timers and periodically reporting the time that elapsed when executing statements. All set tics are cleared once the BayES parser completes a job. • *s* must be a string
ctic([*s*]);	To economize on memory and CPU resources, BayES allows only a limited number of tics to be set simultaneously. When a tic that was previously set by the user is not expected to be used again, a call to **ctic**() should be made to clear it from memory and make space for other tics to be set. If the function is called without an argument, then it clears the last tic which was defined without an argument. All set tics are cleared once the BayES parser completes a job. • *s* must be a string

B.4 Import, export and memory management

The following statements are used to clear items from memory, save or load workspaces, and import or export items from/to text files.

Syntax	Arguments and performed function
clear(A);	Clears item with id value A from memory. A could be any of the following data types: • matrix • model • string • dataset • structure or the keywords: • all • datasets • structures • matrices • models • strings The statement **clear**(all); clears all items from memory, while, for example, **clear**(matrices); clears all matrices from memory.

table continues on next page

table continued from previous page

Syntax	Arguments and performed function
`save(s);`	Saves the current workspace (all items in memory) to the file specified by `s`. • `s` must be a string containing the directory, filename and extension of the file to be saved. • The path to the file specified in `s` can be absolute (eg. `"C:/MyFiles/MyFolder/MyFile.bws"`) or relative to the current working directory (eg. `"./MyFolder/MyFile.bws"`).
`load(s);`	Loads a previously saved workspace from the file specified by `s` to the current workspace. The current workspace must be empty for the operation to proceed. • `s` must be a string containing the directory, filename and extension of the file to be loaded. • The path to the file specified in `s` can be absolute (eg. `"C:/MyFiles/MyFolder/MyFile.bws"`) or relative to the current working directory (eg. `"./MyFolder/MyFile.bws"`).
`D = import(s [, d ]);` `D = webimport(s [, d ]);`	`D` is a dataset constructed by reading the contents of the file or URL specified in `s`. • `s` must be a string specifying the directory, filename and extension of (in the case of `import()`) or the URL that points to (in the case of `webimport()`) the file which contains the data to be imported. • In the case of `import()`, the path to the file can be absolute (eg. `"C:/MyFiles/MyFolder/MyFile.csv"`) or relative to the current working directory (eg. `"./MyFolder/MyFile.csv"`). • `d` is the delimiter used in the data file to separate values and must be equal to one of the following strings: – `","` – `";"` – `"\t"` The default value is comma (`","`) and is used for importing data from csv files. `";"` and `"\t"` can be specified to alter the default value separator to semicolon and tab, respectively. • The variables in the file specified by `s` must be organized in columns, with the first row containing the variable names.

table continues on next page

table continued from previous page

Syntax	Arguments and performed function
export(id, s [, *d*]); **export**(s1, s2 [, <*options*>]);	When used in the first format, **export**(id, s [, *d*]);, the function exports the item in memory with id value id to the file specified in s. • id must be the name of a dataset or matrix currently in memory (structures, models and strings cannot be exported). • s must be a string containing the directory, filename and extension of the file to which the data are to be exported • The path to the file specified in s can be absolute (eg. "C:/MyFiles/MyFolder/MyFile.csv") or relative to the current working directory (eg. "./MyFolder/MyFile.csv"). • d is the delimiter used in the data file to separate values and must be a string with one of the following values: – "," – ";" – "\t" The default value is comma (,) and is used for exporting data to the csv format. ; and \t can be specified to alter the default value separator to semicolon and tab, respectively. • When exporting a matrix, the values of the matrix are printed in the file specified in s. When exporting a dataset, the variable names are printed in the first row of the file, before printing the actual values of the variables. When used in the second format, the function exports the graph contained in the figure window with title s1 to the file specified in string s2. The type of the exported graph (eps, png or jpeg) is determined by the extension of the file provided in s2. • s1 must be a string equal to the name appearing on the title bar of the figure window, the contents of which are to be exported. • s2 must be a string containing the directory, filename and extension of the file to which the graph will be exported. • The path to the file specified in s2 can be absolute (eg. "C:/MyFiles/MyFolder/MyGraph.eps") or relative to the current working directory (eg. "./MyFolder/MyGraph.eps"). The <*options*> argument allows setting the width and height of the exported figure, in pixels. For example, **export**([...],"width"=100,"height"=100); will set the width and height of the exported figure to 100 pixels. The default values for "width" and "height" are 400 and 250, respectively.

B.5 Size information, reshaping/replicating & cleaning

The following statements are used to provide information on the dimensions of matrices or the number of observations or variables in a dataset, to reshape or replicate matrices, and to drop rows or columns with missing data from a matrix or dataset.

Syntax	Arguments and performed function
`S = size(X);`	`S` is a 2×1 matrix of integers. The first entry of `S` is the number of rows of `X` and the second entry is the number of columns of `X`. • `X` must be a matrix or dataset
`p = rows(X);`	`p` is a 1×1 matrix equal to the number of rows of `X`. • `X` must be a matrix or dataset
`p = cols(X);`	`p` is a 1×1 matrix equal to the number of columns of `X`. • `X` must be a matrix or dataset
`W = kron(X, Y);`	`W` is a matrix obtained as the Kronecker product of `X`, and `Y`. • `X` must be a matrix or dataset • `Y` must be a matrix or dataset
`W = repmat(X, M, N);`	`W` is a matrix obtained by replicating `X`, `M` times along the row dimension and `N` times along the column dimension. • `X` must be a matrix or dataset • `M` must be a positive integer • `N` must be a positive integer
`W = reshape(X, M, N);`	`W` is an `M`×`N` matrix constructed by reading the entries of `X`, in a column-major order. • `X` must be a matrix or dataset • `M` must be a positive integer • `N` must be a positive integer • If the number of entries of `X` is not `M`·`N` an error is produced
`W = dropmissing(X);`	`W` is a matrix constructed by reading the entries of `X`, row by row, but skipping any rows in `X` that contain at least one missing value. An error is produced if an empty matrix results from dropping the rows of `X` with missing values. • `X` must be a matrix or dataset When the argument provided to `dropmissing()` is a dataset, then the function returns a dataset. Therefore, this function is also documented in Section B.13. see also `dropif` and `keepif`

B.6 Special matrices

The following statements are used to construct some special matrices, like the identity or a matrix of zeros.

Syntax	Arguments and performed function
`W = eye(M, N);`	`W` is the `M`×`N` upper left block of an identity matrix of size `M`×`M` if `M`>`N` or `N`×`N` otherwise. • `M` must be a positive integer • `N` must be a positive integer
`W = zeros(M, N);`	`W` is an `M`×`N` matrix of zeros. • `M` must be a positive integer • `N` must be a positive integer
`W = ones(M, N);`	`W` is an `M`×`N` matrix of ones. • `M` must be a positive integer • `N` must be a positive integer
`W = range(f, t [, b]);`	`W` is a column vector containing values from `f` to `t`, spaced by `b`. The default value of `b` is one. • `f` must be a scalar • `t` must be a scalar • `b` must be a scalar

B.7 Simple mathematical functions

The following statements are used to transform data contained in matrices or datasets.

Syntax	Arguments and performed function
`W = exp(X);`	`W` is a matrix with entries equal to the exponentials of the entries of `X`. The function works element-wise. • `X` must be a matrix or dataset
`W = log(X);`	`W` is a matrix with entries equal to the natural logarithms of the entries of `X`. The function works element-wise. • `X` must be a matrix or dataset • If `X` contains non-positive entries then the corresponding entries of `W` are set to `nan`
`W = sqrt(X);`	`W` is a matrix with entries equal to the square roots of the entries of `X`. The function works element-wise. • `X` must be a matrix or dataset • If `X` contains negative entries then the corresponding entries of `W` are set to `nan`
`W = abs(X);`	`W` is a matrix with entries equal to the absolute values of the entries of `X`. The function works element-wise. • `X` must be a matrix or dataset
`W = mod(X, Y);`	`W` is a matrix with entries equal to modula of the element-wise division `X./Y` • `X` must be a matrix or dataset • `Y` must be a matrix or dataset • The dimensions of `X` and `Y` must be equal
`W = sin(X);`	`W` is a matrix with entries equal to the sines of the entries of `X`. The function works element-wise. • `X` must be a matrix or dataset

table continues on next page

table continued from previous page

Syntax	Arguments and performed function
`W = cos(X);`	`W` is a matrix with entries equal to cosines of the entries of `X`. The function works element-wise. • `X` must be a matrix or dataset
`W = tan(X);`	`W` is a matrix with entries equal to the tangents of the entries of `X`. The function works element-wise. • `X` must be a matrix or dataset
`W = asin(X);`	`W` is a matrix with entries equal to the arcsines of the entries of `X`. The function works element-wise. • `X` must be a matrix or dataset • If `X` contains entries outside the interval $[-1, 1]$ then the corresponding entries of `W` are set to `nan`
`W = acos(X);`	`W` is a matrix with entries equal to the arccosines of the entries of `X`. The function works element-wise. • `X` must be a matrix or dataset • If `X` contains entries outside the interval $[-1, 1]$ then the corresponding entries of `W` are set to `nan`
`W = atan(X);`	`W` is a matrix with entries equal to the arctangents of the entries of `X`. The function works element-wise. • `X` must be a matrix or dataset
`W = inv(X);`	`W` is the inverse of `X`. • `X` must be a square matrix or dataset • If `X` is singular an error is produced
`W = invpd(X);`	`W` is the inverse of `X`, where `X` is symmetric and positive definite. This function works faster and is more precise than the general `inv()` function, taking advantage of the structure of `X`. • `X` must be a symmetric and positive-definite matrix or dataset • If `X` is not positive definite an error is produced
`W = det(X);`	`W` is an 1×1 matrix with value equal to the determinant of `X`. • `X` must be a square matrix
`W = trace(X);`	`W` is an 1×1 matrix with value equal to the trace of `X`. • `X` must be a square matrix
`W = diag(X);`	The function's return value depends on the size of `X`: $\rightarrow$ if `X` is an $M\times M$ matrix then `W` is an $M\times1$ vector that contains the values on the diagonal of `X` $\rightarrow$ if `X` is vector of length M then `W` is a diagonal $M\times M$ matrix that contains the entries of `X` on its diagonal. • `X` must be either a square matrix (or dataset) or a vector

B.8 Matrix decompositions & quadratures

The following statements are used to decompose matrices (or datasets) or to produce abscissae and weights for approximating integrals via Gaussian quadratures.

Syntax	Arguments and performed function
`L = chol(X);`	`L` is a lower-triangular matrix such that: `L*L' = X` • `X` must be a symmetric and positive-definite matrix or dataset • If `X` is not positive definite an error is produced
`[v, V] = eig(X);`	If `X` is an $M \times M$ symmetric matrix then `v` is an $M \times 1$ vector that contains the eigenvalues of `X` and `V` is an $M \times M$ matrix that contains the corresponding eigenvectors, such that: `V*diag(v)*inv(V)= X` • `X` must be a symmetric matrix or dataset • Only the lower triangular part of `X` is used for the decomposition. This means that if `X` is not symmetric, BayES will not produce a warning or error message
`[x, w] = quadrature(n [, s]);`	`x` is an `n`×1 vector of abscissae and `w` an `n`×1 vector of corresponding weights for a Gaussian quadrature. Depending on the value of the optional argument, the abscissae and weights could be for a Gauss-Laguerre or Gauss-Hermite quadrature. For the Gauss-Laguerre quadrature it holds: $\sum_{i=1}^{n} \mathtt{w}_i \cdot f(\mathtt{x}_i) \approx \int_0^{+\infty} e^{-x} f(x_i)\,\mathrm{d}x$ For the Gauss-Hermite quadrature it holds: $\sum_{i=1}^{n} \mathtt{w}_i \cdot f(\mathtt{x}_i) \approx \int_{-\infty}^{\infty} e^{-x^2} f(x_i)\,\mathrm{d}x$ • `i` must be an integer between 2 and 300; • `s` must be equal to either `"laguerre"` or `"hermite"`. The default value for `s` is `"laguerre"`, for which the abscissae and weights are for a Gauss-Laguerre quadrature.

B.9 Statistical functions, summing, rounding & sorting

The following statemens are used to summarize, sum, round or sort the data contained in a matrix or dataset.

Syntax	Arguments and performed function
`W = mean(X [, d]);`	`W` is a matrix with entries equal to the sample mean of the entries of `X` across dimension `d`. • `X` must be a matrix or dataset • `d` must be either 1 or 2, indicating the dimension across which the mean is computed. The default value for `d` is 1, for which the mean is calculated over rows.

table continues on next page

table continued from previous page

Syntax	Arguments and performed function
`W = var(X [, d]);`	`W` is a matrix with entries equal to the sample variance of the entries of `X` across dimension `d`. • `X` must be a matrix or dataset • `d` must be either 1 or 2, indicating the dimension across which the mean is computed. The default value for `d` is 1, for which the variance is calculated over rows.
`W = sd(X [, d]);`	`W` is a matrix with entries equal to the sample standard deviation of the entries of `X` across dimension `d`. • `X` must be a matrix or dataset • `d` must be either 1 or 2, indicating the dimension across which the standard deviation is computed. The default value for `d` is 1, for which the standard deviation is calculated over rows.
`W = min(X [, d]);`	`W` is a matrix with entries equal to the minimum of the entries of `X` across dimension `d`. • `X` must be a matrix or dataset • `d` must be either 1 or 2, indicating the dimension across which the minimum is computed. The default value for `d` is 1, for which the minimum is calculated over rows.
`W = max(X [, d] );`	`W` is a matrix with entries equal to the maximum of the entries of `X` across dimension `d`. • `X` must be a matrix or dataset • `d` must be either 1 or 2, indicating the dimension across which the maximum is computed. The default value for `d` is 1, for which the maximum is calculated over rows.
`W = median(X [ ,d]);`	`W` is a matrix with entries equal to the median of the entries of `X` across dimension `d`. • `X` must be a matrix or dataset • `d` must be either 1 or 2, indicating the dimension across which the median is computed. The default value for `d` is 1, for which the median is calculated over rows.

table continues on next page

table continued from previous page

Syntax	Arguments and performed function
`W = tabulate(v [ ,m]);`	`W` is a matrix that contains information on the distribution of the values in vector `v`. `W` has three columns: 1. the first column contains the unique values of `v`, sorted from smallest to largest 2. the second column lists the number of times each corresponding unique value in the first column appears in `v` 3. the third column lists the number of entries in `v` smaller than or equal to the corresponding value in the first column (cumulative sum of the second column) `m` is an optional argument, specifying the maximum number of unique values in `v` beyond which an error is produced. • `v` must be a vector or a dataset with a single row or column • `m` must a positive integer. The default value for `m` is 20.
`W = cov(X [, d] );`	`W` is the sample covariance matrix of the variables contained in `X`, with the variables organized across dimension `d`. • `X` must be a matrix or dataset • `d` must be either 1 or 2, indicating the dimension according to which the variables in `X` are organized. The default value for `d` is 1, in which case each column of `X` is treated as a variable.
`W = corr(X [, d] );`	`W` is the sample correlation matrix of the variables contained in `X`, with the variables organized across dimension `d`. • `X` must be a matrix or dataset • `d` must be either 1 or 2, indicating the dimension according to which the variables in `X` are organized. The default value for `d` is 1, in which case each column of `X` is treated as a variable.
`W = ceil(X);`	`W` is a matrix with dimensions equal to those of `X` and entries obtained by rounding off the entries of `X` *upwards* to the nearest integer. The function works element-wise. • `X` must be a matrix or dataset
`W = floor(X);`	`W` is a matrix with dimensions equal to those of `X` and entries obtained by rounding off the entries of `X` *downwards* to the nearest integer. The function works element-wise. • `X` must be a matrix or dataset
`W = round(X);`	`W` is a matrix with dimensions equal to those of `X` and entries obtained by rounding off the entries of `X` to the nearest integer. The function works element-wise. • `X` must be a matrix or dataset

table continues on next page

table continued from previous page

Syntax	Arguments and performed function
`W = sort(X [, d]);`	`W` is a matrix with dimensions equal to those of `X` and entries obtained by sorting, in ascending order, the values of `X` across dimension `d`. • `X` must be a matrix or dataset • `d` must be either 1 or 2, indicating the dimension across which the sorting should be done. The default value for `d` is 1, for which the entries of each column of `X` are sorted in ascending order. see also `sortrows` and `sortd`
`W = sortrows(X [, d]);`	`W` is a matrix with dimensions equal to those of `X` and entries obtained by sorting the rows of `X`, in ascending order, according to the values contained in the columns, the indices of which are provided in vector `d`. • `X` must be a matrix or dataset • `d` must be vector of integers with maximum value not greater than the number of columns of `X`. The default value for `d` is 1, for which the rows of of `X` are sorted in ascending order according to the values contained in the first column. If `d` contains more than one index, then the rows of `X` are sorted first according to the first index, and in case of duplicate values in the respective column, according to second index, and so on. see also `sort` and `sortd`
`W = sum(X [, d]);`	`W` is a matrix with entries equal to the sum of the entries of `X` across dimension `d`. • `X` must be a matrix or dataset • `d` must be either 1 or 2, indicating the dimension across which the sum is computed. The default value for `d` is 1, for which the sum is calculated over rows.
`W = logsumexp(X [, d]);`	`W` is a matrix with entries equal to the logarithm of the sum of the exponential of the entries of `X` across dimension `d`: $\mathtt{W}_j = \log \sum_i \exp\{\mathtt{X}_{ij}\}$ when `d` is one (or not provided). The function is provided to guard against overflow when calculating quantities of this form, which appear frequently in the calculation of log-marginal likelihoods. • `X` must be a matrix or dataset • `d` must be either 1 or 2, indicating the dimension across which the sum is computed. The default value for `d` is 1, for which the sum is calculated over rows.

B.10 Error function, Beta, Gamma and related mathematical functions

The following table describes statements that are used to evaluate the error function, Beta, Gamma and related functions.

Syntax	Mathematical expression	Arguments and return values
`W = erf(X);`	$\frac{2}{\sqrt{\pi}}\int_0^x e^{-t^2}\,\mathrm{d}t$	`W` is a matrix with values equal to error function evaluated at each entry of `X`. The function works element-wise. • `X` must be a matrix or dataset
`W = erfc(X);`	$\frac{2}{\sqrt{\pi}}\int_x^\infty e^{-t^2}\,\mathrm{d}t$	`W` is a matrix with values equal to complementary error function evaluated at each entry of `X`. The function works element-wise. • `X` must be a matrix or dataset
`W = betafunc(X, Y);`	$B(x) = \int_0^1 t^{x-1}(1-t)^{y-1}\,\mathrm{d}t$	`W` is a matrix with values equal to the beta function of each entry of `X` and the corresponding entry of `Y`. The function works element-wise. • `X` must be a matrix or dataset with positive entries • `Y` must be a matrix or dataset with positive entries • The dimensions of `X` must be equal to the dimensions of `Y` • If `X` or `Y` contain non-positive entries then the corresponding entries of `W` are set to `nan` or `inf`
`W = lbetafunc(X, Y);`	$\log\left[B(x,y)\right]$	`W` is a matrix with values equal to the natural logarithm of the beta function of each entry of `X` and the corresponding entry of `Y`. The function works element-wise. • `X` must be a matrix or dataset with positive entries • `Y` must be a matrix or dataset with positive entries • The dimensions of `X` must be equal to the dimensions of `Y` • If `X` or `Y` contain non-positive entries then the corresponding entries of `W` are set to `nan` or `inf`

table continues on next page

table continued from previous page

Syntax	Mathematical expression	Arguments and return values
`W = betainc(Z, X, Y);`	$I_z(x,y) = \dfrac{\int_0^z t^{x-1}(1-t)^{y-1}\,\mathrm{d}t}{B(x,y)}$	`W` is a matrix with values equal to the normalized/regularized lower incomplete beta function of each entry of `X` and the corresponding entry of `Y`. The function works element-wise. • `Z` must be a matrix or dataset with positive entries • `X` must be a matrix or dataset with non-negative entries • `X`, `Y` and `Z` must have equal dimensions • If `Z` contains negative entries or entries above one an error is produced • If `X` or `Y` contain non-positive entries then an error is produced
`W = gammafunc(X);`	$\Gamma(x) = \int_0^\infty t^{x-1}e^{-t}\,\mathrm{d}t$	`W` is a matrix with values equal to the gamma function of each entry of `X`. The function works element-wise. • `X` must be a matrix or dataset • If `X` contains non-positive integers then the corresponding entries of `W` are set to `nan` or `inf`
`W = lgammafunc(X);`	$\log[\Gamma(x)]$	`W` is a matrix with values equal to the natural logarithm of the gamma function of each entry of `X`. The function works element-wise. • `X` must be a matrix or dataset • If `X` contains non-positive entries an error is produced
`W = gammainc(Z, X);`	$\dfrac{\gamma(z,x)}{\Gamma(x)} = \dfrac{\int_0^z t^{x-1}e^{-t}\,\mathrm{d}t}{\Gamma(x)}$	`W` is a matrix with values equal to the normalized/regularized lower incomplete gamma function of each entry of `X`. The function works element-wise. • `Z` must be a matrix or dataset with positive entries • `X` must be a matrix or dataset with non-negative entries • the dimensions of `X` must be equal to the dimensions of of `Z` • If `X` or `Z` contain non-positive entries then an error is produced

table continues on next page

table continued from previous page

Syntax	Mathematical expression	Arguments and return values
`W = mvgammafunc(X, k);`	$\Gamma_k(x) = \pi^{\frac{k(k-1)}{4}} \prod_{j=1}^{k} \Gamma\left(x + \frac{1-j}{2}\right)$	`W` is a matrix with values equal to the k-dimensional gamma function of each entry of `X`. The function works element-wise. • `X` must be a matrix or dataset • `k` must be a positive integer • If `X` contains non-positive integers then the corresponding entries of `W` are set to `nan` or `inf`
`W = lmvgammafunc(X, k);`	$\log\left[\Gamma_k(x)\right]$	`W` is a matrix with values equal to the natural logarithm of the k-dimensional gamma function of each entry of `X`. The function works element-wise. • `X` must be a matrix or dataset • `k` must be a positive integer • If `X` contains non-positive integers then the corresponding entries of `W` are set to `nan` or `inf`

B.11 Probability and cumulative density functions

The following two tables describe statements that are used to evaluate probability density/mass and cumulative density functions (pdfs and cdfs) of some popular distributions.

Syntax	Mathematical expression	Arguments and return values
`W = betapdf(X, alpha, beta);` `W = betacdf(X, alpha, beta);`	$f(x) = \frac{x^{\alpha-1} \cdot (1-x)^{\beta-1}}{B(\alpha,\beta)}$ $F(x) = \frac{\int_0^x t^{\alpha-1} \cdot (1-t)^{\beta-1}\,\mathrm{d}t}{B(\alpha,\beta)}$	`W` is a matrix with dimensions equal to those of `X` and entries equal to the pdf/cdf of the beta distribution with shape parameters `alpha` and `beta`, evaluated at each entry of `X`. The function works element-wise. • `X` must be a matrix or dataset with entries between zero and one • If `X` contains negative entries or entries above one an error is produced • `alpha` must be a positive number • `beta` must be a positive number
`W = chi2pdf(X, p);` `W = chi2cdf(X, p);`	$f(x) = \frac{x^{\frac{p}{2}-1} \cdot e^{-\frac{x}{2}}}{2^{\frac{p}{2}} \cdot \Gamma\left(\frac{p}{2}\right)}$ $F(x) = \frac{\int_0^x t^{\frac{p}{2}-1} \cdot e^{-\frac{t}{2}}\,\mathrm{d}t}{2^{\frac{p}{2}} \cdot \Gamma\left(\frac{p}{2}\right)}$	`W` is a matrix with dimensions equal to those of `X` and entries equal to the pdf/cdf of the chi-squared distribution with `p` degrees of freedom, evaluated at each entry of `X`. The function works element-wise. • `X` must be a matrix or dataset with non-negative entries • If `X` contains negative entries an error is produced • `p` must be a positive number
`W = exppdf(X, lambda);` `W = expcdf(X, lambda);`	$f(x) = \lambda e^{-\lambda x}$ $F(x) = 1 - e^{-\lambda x}$	`W` is a matrix with dimensions equal to those of `X` and entries equal to the pdf/cdf of the exponential distribution with rate parameter `lambda`, evaluated at each entry of `X`. The function works element-wise. • `X` must be a matrix or dataset with non-negative entries • If `X` contains negative entries an error is produced • `lambda` must be a positive number

table continues on next page

table continued from previous page

Syntax	Mathematical expression	Arguments and return values
`W = evpdf(X, mu, sigma);` `W = evcdf(X, mu, sigma);`	$f(x) = \frac{1}{\sigma}\exp\left\{-z - e^{-z}\right\}$ $F(x) = \exp\left\{-e^{-z}\right\}$ where: $z = \frac{x-\mu}{\sigma}$	`W` is a matrix with dimensions equal to those of `X` and entries equal to the pdf/cdf of the type-I extreme-value distribution with location parameter `mu` and scale parameter `sigma`, evaluated at each entry of `X`. The function works element-wise. • `X` must be a matrix or dataset • `mu` must be a number • `sigma` must be a positive number
`W = fpdf(X, p1, p2);` `W = fcdf(X, p1, p2);`	$f(x) = \frac{\left(\frac{(p_1 x)^{p_1} p_2^{p_2}}{(p_1 x + p_2)^{p_1+p_2}}\right)^{\frac{1}{2}}}{xB\left(\frac{p_1}{2}, \frac{p_2}{2}\right)}$ $F(x) = \frac{\int_0^{\frac{p_1 x}{p_1 x + p_2}} t^{\frac{p_1}{2}-1}(1-t)^{\frac{p_2}{2}-1}\,\mathrm{d}t}{B\left(\frac{p_1}{2}, \frac{p_1}{2}\right)}$	`W` is a matrix with dimensions equal to those of `X` and entries equal to the pdf/cdf of Fisher's F distribution with numerator degrees of freedom `p1` and denominator degrees of freedom `p2`, evaluated at each entry of `X`. The function works element-wise. • `X` must be a matrix or dataset with non-negative entries • If `X` contains negative entries an error is produced • `p1` must be a positive number • `p2` must be a positive number
`W = gampdf(X, alpha, beta);` `W = gamcdf(X, alpha, beta);`	$f(x) = \frac{\beta^\alpha \cdot x^{\alpha-1} \cdot e^{-\beta x}}{\Gamma(\alpha)}$ $F(x) = \frac{\beta^\alpha \int_0^x t^{\alpha-1} \cdot e^{-\beta t}\,\mathrm{d}t}{\Gamma(\alpha)}$	`W` is a matrix with dimensions equal to those of `X` and entries equal to the pdf/cdf of the gamma distribution with shape parameter `alpha` and rate parameter `beta`, evaluated at each entry of `X`. The function works element-wise. • `X` must be a matrix or dataset with non-negative entries • If `X` contains negative entries an error is produced • `alpha` must be a positive number • `beta` must be a positive number

table continues on next page

table continued from previous page

Syntax	Mathematical expression	Arguments and return values
`W = logisticpdf(X, mu, s);` `W = logisticcdf(X, mu, s);`	$f(x) = \frac{e^{-\frac{x-\mu}{s}}}{s \cdot \left(1 + e^{-\frac{x-\mu}{s}}\right)^2}$ $F(x) = \frac{1}{1 + e^{-\frac{x-\mu}{s}}}$	`W` is a matrix with dimensions equal to those of `X` and entries equal to the pdf/cdf of the logistic distribution with mean `mu` and scale parameter `s` (variance equal to $\frac{\mathtt{s}^2\pi^2}{3}$), evaluated at each entry of `X`. The function works element-wise. • `X` must be a matrix or dataset • `mu` must be a number • `s` must be a positive number
`W = logitnpdf(X, mu, sigma);` `W = logitncdf(X, mu, sigma);`	$f(x) = \frac{\exp\left\{-\frac{\left(\log\left(\frac{x}{1-x}\right)-\mu\right)^2}{2\sigma^2}\right\}}{x(1-x)\sqrt{2\pi\sigma^2}}$ $F(x) = \int_0^x \frac{\exp\left\{-\frac{\left(\log\left(\frac{t}{1-t}\right)-\mu\right)^2}{2\sigma^2}\right\}}{t(1-t)\sqrt{2\pi\sigma^2}}\,\mathrm{d}t$	`W` is a matrix with dimensions equal to those of `X` and entries equal to the pdf/cdf of logit-Normal distribution with location parameter `mu` and scale parameter `sigma`, evaluated at each entry of `X`. The function works element-wise. • `X` must be a matrix or dataset with entries between zero and one • If `X` contains negative entries or entries greater than one an error is produced • `mu` must be a number • `sigma` must be a positive number
`W = lognpdf(X, mu, sigma);` `W = logncdf(X, mu, sigma);`	$f(x) = \frac{\exp\left\{-\frac{(\log(x)-\mu)^2}{2\sigma^2}\right\}}{x\sqrt{2\pi\sigma^2}}$ $F(x) = \int_0^x \frac{\exp\left\{-\frac{(\log(t)-\mu)^2}{2\sigma^2}\right\}}{t\sqrt{2\pi\sigma^2}}\,\mathrm{d}t$	`W` is a matrix with dimensions equal to those of `X` and entries equal to the pdf/cdf of the log-Normal distribution with location parameter `mu` and scale parameter `sigma`, evaluated at each entry of `X`. The function works element-wise. • `X` must be a matrix or dataset with non-negative entries • If `X` contains negative entries an error is produced • `mu` must be a positive number • `sigma` must be a positive number

table continues on next page

table continued from previous page

Syntax	Mathematical expression	Arguments and return values
`W = normpdf(X, mu, sigma);` `W = normcdf(X, mu, sigma);`	$f(x) = \frac{\exp\left\{-\frac{(x-\mu)^2}{\sigma^2}\right\}}{\sqrt{2\pi\sigma^2}}$ $F(x) = \int_{-\infty}^{x} \frac{\exp\left\{-\frac{(t-\mu)^2}{\sigma^2}\right\}}{\sqrt{2\pi\sigma^2}}\,\mathrm{d}t$	`W` is a matrix with dimensions equal to those of `X` and entries equal to the pdf/cdf of the normal distribution with mean parameter `mu` and standard deviation `sigma`, evaluated at each entry of `X`. The function works element-wise. • `X` must be a matrix or dataset • `mu` must be number • `sigma` must be a positive number
`W = tpdf(X, p);` `W = tcdf(X, p);`	$f(x) = \frac{\Gamma\left(\frac{p+1}{2}\right)}{\sqrt{p\pi}\Gamma\left(\frac{p}{2}\right)}\left(1+\frac{x^2}{p}\right)^{-\frac{p+1}{2}}$ $F(x) = \int_{-\infty}^{x} \frac{\Gamma\left(\frac{p+1}{2}\right)}{\sqrt{p\pi}\Gamma\left(\frac{p}{2}\right)}\left(1+\frac{t^2}{p}\right)^{-\frac{p+1}{2}}\,\mathrm{d}t$	`W` is a matrix with dimensions equal to those of `X` and entries equal to the pdf/cdf of the t distribution with `p` degrees of freedom, evaluated at each entry of `X`. The function works element-wise. • `X` must be a matrix or dataset • `p` must be a positive number
`W = truncnpdf(X, mu, sigma);` `W = truncncdf(X, mu, sigma);`	$f(x) = \frac{\frac{1}{\sigma}\phi\left(-\frac{\mu}{\sigma}\right)}{1-\Phi\left(-\frac{\mu}{\sigma}\right)}$ $F(x) = \frac{\Phi\left(\frac{x-\mu}{\sigma}\right)-\Phi\left(-\frac{\mu}{\sigma}\right)}{1-\Phi\left(-\frac{\mu}{\sigma}\right)}$ where: $\phi()$ is the standard normal pdf $\Phi()$ is the standard normal cdf	`W` is a matrix with dimensions equal to those of `X` and entries equal to the pdf/cdf of a Normal distribution with location parameter `mu` and scale parameter `sigma`, truncated from below at zero and evaluated at each entry of `X`. The function works element-wise. • `X` must be a matrix or dataset with non-negative entries • `mu` must be a number • `sigma` must be a positive number
`W = wblpdf(X, alpha, beta);` `W = wblcdf(X, alpha, beta);`	$f(x) = \frac{\alpha}{\beta}\left(\frac{x}{\beta}\right)^{\alpha-1} e^{-\left(\frac{x}{\beta}\right)^{\alpha}}$ $F(x) = 1 - e^{-\left(\frac{x}{\beta}\right)^{\alpha}}$	`W` is a matrix with dimensions equal to those of `X` and entries equal to the pdf/cdf of the Weibull distribution with shape parameter `alpha` and scale parameter `beta`, evaluated at each entry of `X`. The function works element-wise. • `X` must be a matrix or dataset with non-negative entries • If `X` contains negative entries an error is produced • `alpha` must be a positive number • `beta` must be a positive number

Syntax	Mathematical expression	Arguments and return values
`W = binompdf(X, n, p);` `W = binomcdf(X, n, p);`	$f(x) = \frac{n!}{x!\,(n-x)!}p^x\,(1-p)^{n-x}$ $F(x) = \sum_{k=0}^{x} \frac{n!}{k!\,(n-k)!}p^k\,(1-p)^{n-k}$	`W` is a matrix with dimensions equal to those of `X` and entries equal to the pdf/cdf of the binomial distribution with `n` of trials and probability of success in each trial `p`, evaluated at each entry of `X`. The function works element-wise. • `X` must be a matrix or dataset with integer entries between zero and `n` • If `X` contains negative entries or entries above `n` an error is produced • `n` must be a non-negative number • `p` must be a number between zero and one Note that the function allows for non-integer values of `n` by replacing the factorials in the expression by the Gamma function.
`W = nbinompdf(X, n, p);` `W = nbinomcdf(X, n, p);`	$f(x) = \frac{(x+n-1)!}{x!\,(n-1)!}p^x\,(1-p)^n$ $F(x) = \sum_{k=0}^{x} \frac{(x+n-1)!}{x!\,(n-1)!}p^x\,(1-p)^n$	`W` is a matrix with dimensions equal to those of `X` and entries equal to the pdf/cdf of the negative-binomial distribution with `n` failures until stopping and probability of success in each trial `p`, evaluated at each entry of `X`. The function works element-wise. • `X` must be a matrix or dataset with non-negative integer entries • If `X` contains negative entries an error is produced • `n` must be a positive number • `p` must be a number between zero and one Note that the function allows for non-integer values of `n` by replacing the factorials in the expression by the Gamma function.
`W = poissonpdf(X, lambda);` `W = poissoncdf(X, lambda);`	$f(x) = \frac{\lambda^x e^{-\lambda}}{x!}$ $F(x) = \frac{\lambda^k e^{-\lambda}}{k!}$	`W` is a matrix with dimensions equal to those of `X` and entries equal to the pdf/cdf of the Poisson distribution with rate parameter `lambda`, evaluated at each entry of `X`. The function works element-wise. • `X` must be a matrix or dataset with non-negative integer entries • If `X` contains negative entries an error is produced • `lambda` must be a positive number

B.12 Random-number generators

Random-numbers are generated using a common general seed number. The seed is set to 42 every time BayES starts and is advanced appropriately every time a random-number generator is called. The seed can be set at any point in a script file using the statement:

```
seed(<positive integer>);
```

Note that when a model function is called, another seed is used, which is internal to the particular model and does not advance the general seed. This allows reproducibility of results irrespective of how many random numbers, if any, have been generated in statements before the call to the model function. The model's internal seed is also set to 42 and this value can be altered by passing an optional argument of the form:

```
... , "seed"=<positive integer>, ...
```

to the model function.

B.12.1 Univariate distributions

The generic syntax for calling a random-number generator for a univariate distribution is:

```
W = distributionName( <parameters> [, M, N]);
```

where `distributionName` is the name of a distribution for which BayES provides random numbers (see table below), and <parameters> is a list of the distribution's parameters, as described in section B.11. These parameters must be separated by commas if more than one parameter needs to be supplied. The size of the return matrix, `W`, depends on the size of the parameters passed as arguments to the function and on whether the optional arguments, `M` and `N`, are supplied:

- if the distribution's parameters are scalars and no optional arguments are supplied then `W` is an 1×1 matrix which contains a random draw from the respective distribution.
- if the distribution's parameters are scalars then `M` and `N` can be used to request multiple random numbers. `W` in this case will be an $M \times N$ matrix which contains random draws from the distribution. If `M` is supplied then `N` must be supplied as well.
- if the distribution's parameters are matrices and their dimensions match (if more than one parameter needs to be supplied) then `W` will have the same size as the parameters and it will contain random draws from the distribution, generated by combining parameters in an element-wise fashion. Since the dimensions of `W` are determined by the dimensions of the distribution's parameters, `M` and `N` must not be supplied.
- for distributions which require specifying more than one parameter, if one of the parameters supplied is a scalar and another a matrix, then the scalar parameter is expanded to match the dimensions of the matrix parameter. In this case `W` has the same size as the matrix parameter. Again, `M` and `N` must not be supplied because the dimensions of `W` are determined by the dimensions of the distribution's parameters.

The following two tables document the random-number generating functions for continuous and discrete random variables, respectively.

Syntax	Arguments and performed function
`W = betarnd(alpha, beta [, M, N ]);`	`W` is a matrix of random numbers from a beta distribution with shape parameters `alpha` and `beta`. • `alpha` must be a matrix with positive entries • `beta` must be a matrix with positive entries • *M* must be a positive integer • *N* must be a positive integer see also `betapdf`, `betacdf`
`W = chi2rnd(p [, M, N ]);`	`W` is a matrix of random numbers from a chi-squared distribution with `p` degrees of freedom. • `p` must be a matrix with positive entries • *M* must be a positive integer • *N* must be a positive integer see also `chi2pdf`, `chi2cdf`
`W = exprnd(lambda [, M, N ]);`	`W` is a matrix of random numbers from an exponential distribution with rate parameter `lambda`. • `lambda` must be a matrix with positive entries • *M* must be a positive integer • *N* must be a positive integer see also `exppdf`, `expcdf`
`W = evrnd(mu, sigma [, M, N ]);`	`W` is a matrix of random numbers from a type-I extreme-value distribution with location parameter `mu` and scale parameter `sigma` • `mu` must be a matrix • `sigma` must be a matrix with positive entries • *M* must be a positive integer • *N* must be a positive integer see also `evpdf`, `evcdf`
`W = frnd(p1, p2 [, M, N ]);`	`W` is a matrix of random numbers from Fisher's F distribution with numerator degrees of freedom `p1` and denominator degrees of freedom `p2`. • `p1` must be a matrix with positive entries • `p2` must be a matrix with positive entries • *M* must be a positive integer • *N* must be a positive integer see also `fpdf`, `fcdf`
`W = gamrnd(alpha, beta [, M, N ]);`	`W` is a matrix of random numbers from a gamma distribution with shape parameter `alpha` and rate parameter `beta`. • `alpha` must be a matrix with positive entries • `beta` must be a matrix with positive entries • *M* must be a positive integer • *N* must be a positive integer see also `gampdf`, `gamcdf`
`W = logisticrnd(mu, s [, M, N ]);`	`W` is a matrix of random numbers from a logistic distribution with mean `mu` and scale parameter `s` (variance equal to $\frac{s^2\pi^2}{3}$). • `mu` must be a matrix • `s` must be a matrix with positive entries • *M* must be a positive integer • *N* must be a positive integer see also `logisticpdf`, `logisticcdf`

table continues on next page

table continued from previous page

Syntax	Arguments and performed function
`W = logitnrnd(mu, sigma [, M, N ]);`	`W` is a matrix of random numbers from a logit-normal distribution with location parameter `mu` and scale parameter `sigma`. • `mu` must be a matrix • `sigma` must be a matrix with positive entries • *M* must be a positive integer • *N* must be a positive integer see also `logitnpdf`, `logitncdf`
`W = lognrnd(mu, sigma [, M, N ]);`	`W` is a matrix of random numbers from a log-normal distribution with location parameter `mu` and scale parameter `sigma`. • `mu` must be a matrix • `sigma` must be a matrix with positive entries • *M* must be a positive integer • *N* must be a positive integer see also `lognpdf`, `logncdf`
`W = normrnd(mu, sigma [, M, N ]);`	`W` is a matrix of random numbers from a normal distribution with mean `mu` and standard deviation `sigma`. • `mu` must be a matrix • `sigma` must be a matrix with positive entries • *M* must be a positive integer • *N* must be a positive integer see also `normpdf`, `normcdf`
`W = trnd(p [, M, N ]);`	`W` is a matrix of random numbers from a t distribution with `p` degrees of freedom. • `p` must be a matrix • *M* must be a positive integer • *N* must be a positive integer see also `tpdf`, `tcdf`
`W = truncnrnd(mu, sigma [, M, N ]);`	`W` is a matrix of random numbers from a normal distribution with location parameter `mu` and scale parameter `sigma`, truncated from below at zero. • `mu` must be a matrix • `sigma` must be a matrix with positive entries • *M* must be a positive integer • *N* must be a positive integer see also `truncnpdf`, `truncncdf`
`W = unifrnd([ M, N ]);`	`W` is a matrix of random numbers from a uniform distribution on the interval $[0, 1]$. • *M* must be a positive integer • *N* must be a positive integer
`W = wblrnd(alpha, beta [, M, N ]);`	`W` is a matrix of random numbers from a Weibull distribution with shape parameter `alpha` and scale parameter `beta`. • `alpha` must be a matrix with positive entries • `beta` must be a matrix with positive entries • *M* must be a positive integer • *N* must be a positive integer see also `wblpdf`, `wblcdf`

Syntax	Arguments and performed function
`W = catrnd(p [, M, N ]);`	`W` is a matrix of random numbers from a categorical distribution with probability vector specified in each row of `p`. The number of outcomes is deduced from the number of columns of `p` and numbering starts at 0. That is, if `p` is a $1\times k$ vector, the possible values of the random variable are $\{0, 1, \ldots, k-1\}$ and the probability of obtaining outcome j is equal to the $(j+1)^{\text{th}}$ element of `p`. • `p` must be a matrix with non-negative entries and values such that each row sums to unity • *M* must be a positive integer • *N* must be a positive integer
`W = binomrnd(n, p [, M, N ]);`	`W` is a matrix of random numbers from a binomial distribution with `n` trials and probability of success in each trial `p`. • `n` must be a matrix with non-negative entries • `p` must be a matrix with entries between zero and one • *M* must be a positive integer • *N* must be a positive integer Note that the function allows for non-integer values of `n` by replacing the factorials in the expressions for the probability mass and cumulative density functions by the Gamma function. see also `binompdf`, `binomcdf`
`W = nbinomrnd(n, p [, M, N ]);`	`W` is a matrix of random numbers from a negative binomial distribution with `n` failures before stopping and probability of success in each trial `p`. • `n` must be a matrix with positive entries • `p` must be a matrix with entries between zero and one • *M* must be a positive integer • *N* must be a positive integer Note that the function allows for non-integer values of `n` by replacing the factorials in the expressions for the probability mass and cumulative density functions by the Gamma function. see also `nbinompdf`, `nbinomcdf`
`W = poissonrnd(lambda [, M, N ]);`	`W` is a matrix of random numbers from a Poisson distribution with rate parameter `lambda`. • `lambda` must be a matrix with positive entries • *M* must be a positive integer • *N* must be a positive integer see also `poissonpdf`, `poissoncdf`

B.12.2 Multivariate distributions

The following table describes statements that are used to generate random numbers from multivariate distributions. The general format of these statements is similar to random-number generators for univariate distributions, but, because multivariate random-number generators return vectors or matrices, the optional dimension arguments do not have the usual meaning: these generators can generate M or one random draw per call.

Syntax	Mathematical expression	Arguments and return values
`W = mvnrnd(mu, V [, M ]);`	$f(\mathbf{x}) = \frac{\lvert\mathbf{V}\rvert^{-\frac{1}{2}} e^{(\mathbf{x}-\boldsymbol{\mu})'\mathbf{V}^{-1}(\mathbf{x}-\boldsymbol{\mu})}}{(2\pi)^{\frac{k}{2}}}$	`W` is an $M \times k$ matrix of random numbers from a k-variate normal distribution with mean `mu` (either $k\times1$ or $1\times k$) and variance matrix `V` ($k\times k$). • `mu` must be a matrix with at least one dimension equal to 1 • `V` must be a square and positive-definite matrix • the dimensions of `mu` and `V` must match: if `V` is $k\times k$ then `mu` must be either $k\times1$ or $1\times k$ • `M` must be a positive integer
`W = wishrnd(V, p);`	$f(\mathbf{X}) = \frac{\lvert\mathbf{X}\rvert^{\frac{p-k-1}{2}} e^{-\frac{1}{2}\mathrm{tr}(\mathbf{V}^{-1}\mathbf{X})}}{2^{\frac{pk}{2}}\lvert\mathbf{V}\rvert^{\frac{p}{2}}\Gamma_k\left(\frac{p}{2}\right)}$	`W` is a $k\times k$ matrix of random numbers from a k-dimensional Wishart distribution with scale matrix `V` ($k\times k$) and `p` degrees of freedom. • `V` must be a square and positive-definite matrix • `p` must be a positive number
`W = iwishrnd(V, p);`	$f(\mathbf{X}) = \frac{\lvert\mathbf{X}\rvert^{-\frac{p+k+1}{2}} e^{-\frac{1}{2}\mathrm{tr}(\mathbf{V}\mathbf{X}^{-1})}}{2^{\frac{pk}{2}}\lvert\mathbf{V}\rvert^{-\frac{p}{2}}\Gamma_k\left(\frac{p}{2}\right)}$	`W` is a $k\times k$ matrix of random numbers from a k-dimensional inverse-Wishart distribution with scale matrix `V` ($k\times k$) and `p` degrees of freedom. • `V` must be a square and positive-definite matrix • `p` must be a positive number
`W = drchrnd(alpha [, M ]);`	$f(\mathbf{x}) = \frac{1}{B(\boldsymbol{\alpha})}\prod_{j=1}^{k} x_j^{\alpha_j-1}$	`W` is an $M\times k$ matrix of random numbers from a k-dimensional Dirichlet distribution with concentration parameters specified in each row of `alpha`. The dimension of the distribution, $k \geq 2$, is inferred by the number of columns of `alpha`. • `alpha` must be a matrix of positive numbers with at least 2 columns • `M` must be a positive integer

B.13 Statements for working with datasets

For all practical purposes datasets in BayES are matrices with additional structure. This means that if, for example, `D` is a dataset, then indexing operations and functions operating on matrices work on `D` in the same way as if `D` were a matrix. There are, however, some additional functions and statements that work on datasets, but not on matrices. These are documented in the following table.

Syntax	Arguments and performed function
`X = D.varname;`	`X` is a column vector with entries equal to the values of variable `varname` in dataset `D`. • `D` must be a dataset • `varname` must be the name of a variable contained in `D`
`D.varname = <math expression>;`	Creates a new variable called `varname` (defined by `<math expression>`) and adds it to dataset `D`. If `D` already has a variable called `varname` then its values are replaced. • `D` must be a dataset • `varname` must be a valid id value • `<math expression>` could by any mathematical expression that returns a scalar or a column vector with number of rows equal to the number of observations in `D` • when `<math expression>` returns a scalar, then its value is expanded to match the number of rows of `D` (note that this is the only instance where BayES will expand a scalar in the right-hand side of an assignment statement to match the dimensions of the left-hand side item)
clear`(D.varname);`	Deletes the variable called `varname` from dataset `D`. • `D` must be a dataset • `varname` must be the name of a variable contained in `D`
`D = dataset(A [, {`*`ID1, ID2, ...`*`}]);`	`D` is a dataset constructed by the data contained in matrix `A`. `ID1, ID2, ...` is a list of id values (id values inside curly brackets) to be used as the variable names. If variable names are not provided then the variables are named `_V1, _V2`, etc. • `A` must be a matrix • `ID1, ID2, ...` must be distinct id values • the number of id values provided must be equal to the number of columns in `A` see also **import**
`rename(D.oldname, newname);`	Renames variable `oldname` in dataset `D` to `newname`. • `D` must be a dataset • `oldname` must be a variable within `D` • `newname` must be an id value

table continues on next page

table continued from previous page

Syntax	Arguments and performed function
keepif(<condition> [,"dataset"=D]);	Keeps the observations in dataset D that satisfy the logical <condition>. The remaining observations (those that do not satisfy <condition>) are permanently deleted from D. If D is not provided then the function operates on the first dataset available in the current workspace. The statement has no return value and the data in D are overwritten. • D must be a dataset • <condition> could be a simple or compound logical condition, for example: – D.var1 >= D.var2 – D.var1 >= D.var2 | D.var3 == 0 see also dropif and dropmissing
dropif(<condition> [,"dataset"=D]);	Drops (permanently deletes) the observations in dataset D that satisfy the logical <condition>. The remaining observations (those that do not satisfy <condition>) are retained. If D is not provided then the function operates on the first dataset available in the current workspace. The statement has no return value and the data in D are overwritten. • D must be a dataset • <condition> could be a simple or compound logical condition, for example: – D.var1 >= D.var2 – D.var1 >= D.var2 | D.var3 == 0 see also keepif and dropmissing
W = dropmissing(X);	W is a dataset constructed by reading the entries of X, row by row, but skipping any rows in X that contain at least one missing value. An error is produced if an empty dataset results from dropping the rows of X with missing values. • X must be a matrix or dataset When the argument provided to dropmissing() is a matrix then the function returns a matrix. Therefore, this function is also documented in Section B.5. see also dropif and keepif

table continues on next page

table continued from previous page

Syntax	Arguments and performed function
`sortd(<variables> [,"dataset"=D]);`	Sorts the data in dataset `D` in ascending order according to the values of the variables, the names of which are provided in the `<variables>` list. If `D` is not provided then the function operates on the first dataset available in the current workspace. The statement has no return value and the data in `D` are overwritten. • `D` must be a dataset • `<variables>` could be one of the following: – a single variable name (id value) ex: `myVariable` – a list of variable names (id values inside curly brackets and separated by commas) ex: `{variable1, variable2}` In the latter case the data are sorted first according to `variable1`, and in case of duplicate values in `variable1`, according to `variable2`, within each group of duplicate values of `variable1` see also `sort` and `sortrows`
`summary(<variables>` `[,"dataset"=D]);`	Calculates and prints summary statistics of the variables in dataset `D`. If `D` is not provided then the function operates on the first dataset available in the current workspace. • `D` must be a dataset • `<variables>` defnines the variables for which summary statistics are calculated and could be one of the following: – a single variable name ex: `myVariable` – a list of variable names (id values inside curly brackets and separated by commas) ex: `{variable1, variable2}` – the keyword `all` which requests calculation of summary statistics for all variables in `D`
`set_ts(tid` `[,"format"=s,"dataset"=D]);`	Declares the dataset as time series, with `tid` being the variable that identifies time periods. If `D` is not provided then the function operates on the first dataset available in the current workspace. • `D` must be a dataset • `tid` must be the name of a variable contained in `D`; this variable can have either numeric or string values • `s` must be one of the following strings: – `"index"`: integer values – `"yyyyqx"`: quarterly data – `"yyyymxx"`: monthly data – `"yyyy/mm/dd"`: daily data The default value for `s` is `"index"`, in which the magnitude of the integer values indicates the time ordering and spacing of the observations. • each observation in `D` must have a unique value for the `tid` variable

table continues on next page

table continued from previous page

Syntax	Arguments and performed function
`set_pd(tid, pid [,"format"=s,"dataset"=D]);`	Declares the dataset as a panel, with `tid` being the variable that identifies time periods and `pid` the variable that identifies groups. If `D` is not provided then the function operates on the first dataset available in the current workspace. • `D` must be a dataset • `tid` must be the name of a variable contained in `D`; this variable can have either numeric or string values • `pid` must be the name of a variable contained in `D`. Its values must be integers, with equal magnitude for observations which belong to the same group • `s` must be one of the following strings: – `"index"`: integer values – `"yyyyqx"`: quarterly data – `"yyyymxx"`: monthly data – `"yyyy/mm/dd"`: daily data The default value for `s` is `"index"`, in which the magnitude of the integer values indicates the time ordering and spacing of the observations. • each observation in `D` must have a unique value for the `tid-pid` pair of variables
`set_cs(["dataset"=D]);`	Clears any time structure from dataset `D`, that was set by a call to either the `set_ts` or `set_pd` functions. If `D` is not provided then the function operates on the first dataset available in the current workspace. • `D` must be a dataset
`X = lag(varname [,l,"dataset"=D]);`	`X` is a column vector obtained by taking lags of length `l` on variable with name `varname` from dataset `D`. If `D` is not provided then the function operates on the first dataset available in the current workspace. • `D` must be a dataset, previously declared either as a time-series or panel dataset • `varname` must be the name of a variable contained in `D` • `l` must be an integer. The default value for `l` is 1
`X = diff(varname [,o,l,"dataset"=D]);`	`X` is a column vector obtained by taking seasonal differences of order `o` and seasonal length `l` on variable with name `varname` from dataset `D`. If `D` is not provided then the function operates on the first dataset available in the current workspace. • `D` must be a dataset, previously declared either as a time-series or panel dataset • `varname` must be the name of a variable contained in `D` • `o` must be a positive integer. The default value for `o` is 1 • `l` must be an integer. The default value for `l` is 1

table continues on next page

table continued from previous page

Syntax	Arguments and performed function
`X = groupmeans(varname [,"dataset"=D]);`	`X` is a column vector obtained by calculating the arithmetic mean per group of the variable with name `varname` from dataset `D`. `X` has the same length as the number of observations in `D` and missing values are generated if `varname` has only missing values for a particular group. If `D` is not provided then the function operates on the first dataset available in the current workspace. • `D` must be a dataset, previously declared as a panel dataset • `varname` must be the name of a variable contained in `D`
`X = groupvars(varname [,"dataset"=D]);`	`X` is a column vector obtained by calculating the variance per group of the variable with name `varname` from dataset `D`. `X` has the same length as the number of observations in `D` and missing values are generated if `varname` has only missing values for a particular group. If `D` is not provided then the function operates on the first dataset available in the current workspace. • `D` must be a dataset, previously declared as a panel dataset • `varname` must be the name of a variable contained in `D`
`X = groupsds(varname [,"dataset"=D]);`	`X` is a column vector obtained by calculating the standard deviation per group of the variable with name `varname` from dataset `D`. `X` has the same length as the number of observations in `D` and missing values are generated if `varname` has only missing values for a particular group. If `D` is not provided then the function operates on the first dataset available in the current workspace. • `D` must be a dataset, previously declared as a panel dataset • `varname` must be the name of a variable contained in `D`
`X = groupmedians(varname [,"dataset"=D]);`	`X` is a column vector obtained by calculating the median per group of the variable with name `varname` from dataset `D`. `X` has the same length as the number of observations in `D` and missing values are generated if `varname` has only missing values for a particular group. If `D` is not provided then the function operates on the first dataset available in the current workspace. • `D` must be a dataset, previously declared as a panel dataset • `varname` must be the name of a variable contained in `D`

table continues on next page

table continued from previous page

Syntax	Arguments and performed function
`X = groupsums(varname [,"dataset"=D]);`	`X` is a column vector obtained by calculating the sum per group of the variable with name `varname` from dataset `D`. `X` has the same length as the number of observations in `D` and missing values are generated if `varname` has only missing values for a particular group. If `D` is not provided then the function operates on the first dataset available in the current workspace. • `D` must be a dataset, previously declared as a panel dataset • `varname` must be the name of a variable contained in `D`
`X = groupcounts(varname [,"dataset"=D]);`	`X` is a column vector obtained by calculating the number of observations per group of the variable with name `varname` from dataset `D`. `X` has the same length as the number of observations in `D` and the number of observations per group excludes any observations with missing values on `varname`. If `D` is not provided then the function operates on the first dataset available in the current workspace. • `D` must be a dataset, previously declared as a panel dataset • `varname` must be the name of a variable contained in `D`

B.14 Statements for post-estimation analysis

If the results of an estimated model are saved in memory (by providing an id value in the left-hand side of a model-estimation statement), these results become available for various types of post-estimation analysis. These include calculating and presenting MCMC diagnostics, testing hypotheses and comparing models based on Bayes factors, calculating marginal effects, etc.

Syntax	Arguments and performed function
`[W =] diagnostics( ["model"=m] );`	Calculates and prints MCMC diagnostics for the model with id value `m`. If `m` is not provided then the function operates on the first model (in alphabetical order) in the current workspace. The reported diagnostics currently include: – the header printed on top of every model estimated (number of observations, Gibbs parameters, log-marginal likelihood, etc.) – an estimate of the Monte Carlo standard error for every model parameter – an estimate of the relative numerical efficiency and the inefficiency factor (Chib, 2001) for every model parameter If a left-hand-side id value, `W`, is provided then the contents of the diagnostics table are stored in `W`. • `m` must be a model

table continues on next page

table continued from previous page

Syntax	Arguments and performed function
[*W* =] plotdraws(p [,"*model*"=*m*]);	Creates a new figure window and plots four types of plots using the draws of paramater p from model m. These plots are: • the history of the draws per chain • the correlogram of the draws • the histogram of the draws • the kernel desnity of the draws per chain If m is not provided then the function operates on the first model (in alphabetical order) in the current workspace. If a left-hand-side id value, W, is provided then the title of the figure window on which the function is plotting is stored in W. • p must be the name of a parameter estimated by model m • m must be a model currently in memory
[*W* =] test(<condition1> [, <*condition2*>, ...]);	Calculates and prints the number of times and percentage that conditions <condition1>, <*condition2*>, ..., are satisfied, first individually and then jointly. These conditions can be expressed using general vectors. For example, <condition1> could be: v1>=v2, where v1 and v2 are vectors of equal length, say N. In this case the test function would count the number of times the n^{th} element of v1 is greater than or equal to the n^{th} element of v2, for $n = 1, 2, \ldots, N$. The test statement is most frequently used to test restrictions that involve the parameters of a model m. In this case <condition1> could be: m.x1<3, where test counts the number of times the parameter associated with variable x1 is smaller than 3 (note that scalar values are expanded to match the size of m.x1). If a left-hand-side id value, W, is provided then the contents of the printed table are stored in W. • <condition1>, <*condition2*>, ... must be logical conditions of the form: v1 <comparison operator> v2 where v1 and v2 are vectors of equal length and <comparison operator> is one of the following: – > – < – == – >= – <=

table continues on next page

table continued from previous page

Syntax	Arguments and performed function

`pmp( { m1, m2 [, m3, m4 ...] }`
`[, "priors"=p]`
`[, "logML_CJ"=true|false]`
`);`
	Calculates and prints the posterior model probabilities of models `m1`, `m2`, ..., assuming that the list of models provided is exhaustive. The posterior model probabilities are calculated using Bayes factors and the model prior probabilities provided in vector `p`. If the optional argument `"logML_CJ"` is set to **true** then Bayes factors are calculated using the Chib (1995)/Chib & Jeliazkov (2001) approximation of the marginal likelihood for the models this is available; otherwise, the Lewis & Raftery (1997) approximation is used for all models.
	• `m1, m2 [, m3, m4 ...]` must be models currently in memory
	• `p` must be a vector of non-negative values that sum to unity, the length of which is equal to the number of models to be compared. The default value for `p` is set such that every model to be compared is given equal prior probability.
	• `"logML_CJ"` must be either **true** or **false**. The default value for `"logML_CJ"` is **false**.

`[W =] mfx( ["point"=p]`
`[,"model"=m]`
`[,"type"=i]`
`[,"opt"=z]`
`);`
	Calculates and prints marginal effects of the type indicated by `i`, for model `m`, at the point indicated by `p`, while using `z` to pass any additional numerical input (model-specific). All arguments are optional and, therefore, the order in which they are provided does not matter.
	If `m` is not provided then the function operates on the first model (in alphabetical order) in the current workspace.
	If a left-hand-side id value, `W`, is provided then the contents of the marginal effects table are stored in `W`.
	• `p` must be either a vector of values indicating the point at which the marginal effects are to be evaluated or one of the following strings:
	– `"mean"` – `"median"` – `"x_i"`
	In the first case the marginal effects are evaluated at the sample mean, in the second at the sample median and in the third at each data point (before being averaged for reporting).
	The default value for `p` is `"mean"`.
	• `m` must be a model currently in memory
	• `i` must be a positive integer that controls the type of marginal effects to be calculated; the type of marginal effects available is model-specific (some models do not provide marginal effects at all) and the semantics of this argument are documented under each model
	• `z` must be a matrix and its meaning varies by model

table continues on next page

table continued from previous page

Syntax	Arguments and performed function
[*[p1, p2, ...]* =] predict([*"point"=p*] [,*"model"=m*] [,*"stats"=true|false*] [,*"type"=i*] [,*"opt"=z*] [,*"prefix"=f*]);	Generates predictions of the type indicated by `i`, for model `m`, at the point(s) indicated by `p`, while using `z` to pass any additional numerical input (model-specific). If `"stats"` is set to **`true`**, summary statistics for each prediction are also generated. All arguments are optional and, therefore, the order in which they are provided does not matter. If `m` is not provided then the function operates on the first model (in alphabetical order) in the current workspace. If left-hand-side id values, `[p1, p2, ...]`, are provided then the predictions and, possibly, their summary statistics, are stored in `p1`, `p2`, etc. • `p` must be either a matrix of values indicating the points at which the predictions are to be made or string `"x_i"`. In the latter case predictions are made at each data point in the dataset used by estimation and the results (point estimates and, possibly, summary statistics) are stored in the dataset with variable names prefixed by `f`. • `m` must be a model currently in memory • `i` must be a positive integer that controls the type of predictions to be generated; the type of predictions available is model-specific (some models do not provide a procedure for generating predictions at all) and the semantics of this argument are documented under each model • `z` must be a matrix and its meaning varies by model • `f` must be a valid id value and indicates the prefix in the variable names to be stored in the dataset if the `"point"\` option is set to `"x_i"`; the default value for `f` is `p_`

table continues on next page

table continued from previous page

Syntax	Arguments and performed function
`store( e, varname [,"model"=m] );`	Stores element `e` from model `m` to the dataset associated with `m` in variable `varname`. If a variable with the same name already exists in the dataset then its values are replaced by the values in `e`. If `m` is not provided then the function operates on the first model (in alphabetical order) in the current workspace. This function is used to store observation-specific estimates to the datasets used for estimating the parameters of a model. Due to possible missing values in the original dataset, there is no guarantee that the observation-specific estimates in `e` will have an one-to-one relationship to the data contained in the dataset. `store` is, therefore, used to associate the values in `e` with specific observations. Model-specific documentation of the `store` function is provided in the section where the model is defined. Not all models provide a `store` function. • `e` must be an element of model `m` • `varname` must be an id value • `m` must be a model currently in memory

`[f1, f2, ...] = forecast(`
`["horizon"=h]`
`[, "W"=M]`
`[, "model"=m]`
`);`
	Calculates and stores `h` period ahead forecasts for the endogenous variables in model `m`, using the values in matrix `M` for the exogenous variables (if any). This function is available only for dynamic models, such as ARIMA, VAR, etc. All three arguments are optional and, therefore, the order in which they are provided does not matter.
	`f1, f2, ...` are $h \times 4$ matrices whose columns contain the expected values and standard deviations of the forecasts per endogenous variable, as well as the upper and lower bounds of the respective 90% credible intervals. Upon completion of the `forecast` function, these matrices are also made available as elements of model `m`, with id values constructed by prepending "`f_`" to the respective endogenous variable name.
	If `m` is not provided then the function operates on the first model (in alphabetical order) in the current workspace.
	• `h` must be a positive integer. The default value for `h` is `1`.
	• `M` must be a $h \times K$ matrix, where K is the number of exogenous variables in the model. If the model contains exogenous variables but `"W"` is not provided, then the sample means of the exogenous variables are used.
	• If both `"horizon"` and `"W"` are provided then the number of rows of `M` must be equal to `h`. If only `"W"` is provided then the number of forecast periods is inferred from the rows of `M`.
	• `m` must be a model currently in memory

table continues on next page

table continued from previous page

Syntax	Arguments and performed function
`stabtest( ["model"=m] );`	Performs a stability test for model `m`. This function is available only for models that contain autoregressive terms (ARIMA and VAR, although not yet implemented for the VAR model). It works by treating the dynamic model as a difference equation and calculating the proportion of draws from the posterior for which the root of the characteristic polynomial with the maximum modulus lies within the unit circle. If `m` is not provided then the function operates on the first model (in alphabetical order) in the current workspace. • `m` must be a model currently in memory

`irf( ["horizon"=h]`
`[, "orthogonal"=true|false]`
`[, "scaled"=true|false]`
`[, "model"=m]`
`);`
	Calculates and plots `h` period impluse responses to a shock in the variables in model `m`. This function is available only for the VAR model. All four arguments are optional and, therefore, the order in which they are provided does not matter.
	If `m` is not provided then the function operates on the first model (in alphabetical order) in the current workspace.
	• `h` must be a positive integer. The default value for `h` is `10`.
	• `"orthogonal"` must be set to either **true** or **false**, indicating whether a shock on the orthogonalized or raw errors should be considered, respectively. The default value for `"orthogonal"` is **true**, in which case the shocks are orthogonalized using the Cholesky decomposition of the variance matrix, prior to calculating responses.
	• `"scaled"` must be set to either **true** or **false**, indicating whether a shock of size equal to the standard deviation of the error or to a unit of the respective variable should be considered, respectively. The default value for `"scaled"` is **true**, in which case the shock is equal to the one standard deviation of the error.
	• `m` must be a model currently in memory

B.15 Statements for working with strings

Strings in BayES are used, primarily, to print messages on the BayES console and in directory statements. To improve the speed of evaluation of mathematical statements, which are much more frequently used, string operations and functions form a distinct part of the BayES language. This means that indexing operations or functions that work on matrices do not work on strings. However, specialized functions that operate on strings are provided and these are documented in the following table.

Note that, as with the rest of the language, strings are case sensitive: `"my string"` is not the same as `"My String"`.

Syntax	Arguments and performed function
`x = strlength(s);`	`x` is a 1×1 matrix that stores the number of characters in string `s`. Note that escape characters are treated as separate entries. For example, the string `"\"my string\""` contains 13 characters, with each `"\""` occupying two places in the string. • `s` must be a string
`s = strcat(s1, s2 [, s3, s4,...]);`	`s` is a string that is constructed by concatenating the strings that are passed as arguments to `strcat()`. • `s1`, `s2`, *s3*, *s4*, ... must all be strings
`s = substr(r, l:u);`	`s` is a string that is constructed by extrancting the characters in `r` located from position `l` to `u` (margins included). Note that escape characters in `s` occupy two places in a string. • `r` must be a string • `u` and `l` must be positive integers with $\texttt{l} \leq \texttt{u}$
`strreplace(s, r, t);` `strreplace(s, r, l:u);`	The `strreplace()` function in the first form replaces all occurrences in string `s` of the characters in string `r` with the characters in string `t`. For example, the statement `strreplace(s, "\\", "/");` replaces all backslashes in `s` with forward slashes. In the second form the function replaces the characters in string `s` from position `l` to `u` (margins included) with the characters in string `r`. • `s`, `r` and `t` must all be strings • `u` and `l` must be positive integers with $\texttt{l} \leq \texttt{u}$
`x = strfind(s, r);`	`x` is a 1×1 matrix that stores the location in string `s` at which the first match occurs between the characters in `s` and `r`. For example, if `s` is `"Hello World"` and `r` is `"World"`, then `x` will be equal to seven. `strfind()` returns `0` if `s` does not contain `r`. • `s` must be a string • `r` must be a string
`x = strfindr(s, r);`	`x` is a 1×1 matrix that stores the location in string `s` at which the last match occurs between the characters in `s` and `r`. For example, if `s` is `"Hello World"` and `r` is `"World"`, then `x` will be equal to seven. `strfind()` returns `0` if `s` does not contain `r`. • `s` must be a string • `r` must be a string
`strcmp(s, r)`	The `strcmp()` function compares strings `s` and `r` and evaluates to **true** if the two strings match exactly and **false** otherwise. Note that the code provided in the left is not a complete statement. `strcmp(s, r)` would typically be the condition in an **if-else** or a **while** statement. • `s` must be a string • `r` must be a string

table continues on next page

table continued from previous page

Syntax	Arguments and performed function
`s = mat2str(A [, p]);`	The `mat2str()` function produces a string `s` from the contents of a 1×1 matrix `A`, using up to p digits. If p is not provided then `s` will contain the value of `A` as it appears on the BayES console after a **print**`(A);` statement. • `A` must be a 1×1 matrix • p must be a positive integer

B.16 Plotting

BayES can produce five types of plots:

1. histograms
2. scatter plots (y versus x)
3. correlograms (acf plots)
4. line plots (y versus x or the values of y versus their row index)
5. kernel density estimates

There are five basic plotting functions in BayES, each one of which corresponds to a plot type: `hist()`, `scatter()`, `acf()`, `plot()`, and `kden()`. These functions are documented in the following table and, in their simple use, they draw their respective plots on a new figure window that they initiate. For example, execution of a statement of the form:

```
plot( y, x );
```

will create a new figure window and plot the values contained in vector `y` versus the values contained in vector `x`. All five basic plot functions return the title of the figure window on which they are plotting. If, for example, the following statement is executed, while no figure windows are currently open:

```
figureTitle = plot( y, x );
```

apart from creating a new figure window and displaying the plot, BayES will create a new string item in the current workspace, with id value `figureTitle` and content `"Figure 1"`.

The five basic plotting functions differ in the number of numerical arguments they take, but all of them have the following optional arguments:

- `"title"`
- `"xlabel"`
- `"ylabel"`
- `"grid"`
- `"colors"`

These arguments, if provided, must be given after the numerical arguments of the plotting function, separated by commas and in any order. For example, the statement:

```
scatter(y, x, "title"="my title", "xlabel"="my x-axis label", "grid"="on");
```

produces a scatter plot of the values contained in vector `y` versus the values in vector `x` and sets the plot's title and x-axis label. Finally, it requests a grid to be drawn on the figure. The statement is equivalent to:

```
scatter(y, x, "xlabel"="my x-axis label", "grid"="on", "title"="my title");
```

`"title"`, `"xlabel"` and `"ylabel"` must be followed by the assignment operator, '`=`', and a string that specifies the corresponding option. `"grid"` must be followed by the assignment operator and either the string `"on"` or the string `"off"`.

The `"colors"` option specifies the colors to be used in the graph. Its value must be a matrix with three columns and all elements between zeros and one. Each row of this matrix specifies a color in RGB (red-green-blue) format, with the first row specifying the background color, the second the color of the axes and the text labels and the remaining rows specifying the colors used for plotting the data. For example, the statement:

```
scatter(y, x, "colors"=[0,0,0; 1,0,0; 0.5,0,0.5]);
```

plots a scatter plot of the values in `y` versus the values in `x` and sets the graph's background color to black (0, 0, 0), the color of the axes and labels to red (1, 0, 0) and the color of the markers used to represent the data to magenta (0.5, 0, 0.5).

For the remainder of this section the five optional arguments described above will be represented by *<plot options>*.

Multiple plots per figure window can be drawn by combining the five plotting functions with the `multiplot()` and `subplot()` functions. For example, the statement:

```
figureTitle = multiplot( 2, 3 );
```

will initialize and display a figure window that can plot 6 plots, across 2 rows. At this stage, this window will be empty and need to be populated by actual plots. Submitting the statement:

```
plot( subplot(figureTitle, 2, 1), y, x ) ;
```

after a call to `multiplot()` will plot the values in `y` versus the values in `x` in row 2, column 1 of the figure window whose title bar displays the same string as the value of `figureTitle`. Similar calls to other plotting functions can be used to populate the remaining spaces of the multiple-plot figure window. Notice that the `subplot()` function must be provided as the first argument of a plotting function. If the call to `plot()` in the previous statement did not include a call to `subplot()`, the plot would be created in a new figure window, even after an empty multiple-plot window has been initialized. Thus, the call to `subplot()` is optional and, for the remainder of this section, any possible complete call to this function will be denoted by *<subplot options>*. Note that subplots within graphs must have the same background color. The overall background color in graphs that contain multiple subplots is the background color specified for the subplot located at the upper left corner of the graph.

The **close()** function can be used to close a figure window programmatically. For example the statement:

```
close("Figure 1");
```

will close the figure window with "Figure 1" displayed in its title bar, while the statement:

```
close(all);
```

will close all currently open figure windows. The contents of figure windows can be exported using the **export()** function, which is documented in section B.4.

BayES limits the maximum number of figure windows that can be open at any given time. If a new figure window is requested when this maximum number has been reached then BayES produces an error. The function **maxfigures()** can be used to change the maximum number of figure windows using a statement like:

```
maxfigures( <positive integer> );
```

Finally, `plotdraws()` is a utility function that can be used to create a multiple-plot figure window and plot four types of plots of the draws from the posterior distribution of a parameter from a model in the current workspace. Because this function works on the results of models only, it is documented in section B.14.

Syntax	Arguments and performed function
[*W* =] acf([<*subplot options*>,] y [, *lags*, <*plot options*>]);	Plots a correlogram for the values in vector y. *lags* sets the maximum lags for which the correlation coefficients are calculated and plotted. If *lags* is not provided then its value is set equal to 30. If a left-hand-side id value, W, is provided then the title of the figure window on which the function is plotting is stored in W. • y must be a column vector • *lags* must be a positive integer • the use of <*plot options*> is described at the beginning of this section • the use of <*subplot options*> is described at the beginning of this section • if acf() is called without a call to subplot(), a new figure window is created; otherwise, the plot is drawn on an existing multiple-plot figure and at the location specified by <*subplot options*>
[*W* =] hist([<*subplot options*>,] y [, *bins*, <*plot options*>]);	Plots a histogram of the values in vector y. *bins* is the number of bins to be used for the histogram. If *bins* is not provided then its optimal value is calculated internally. If a left-hand-side id value, W, is provided then the title of the figure window on which the function is plotting is stored in W. • y must be a column vector • *bins* must be a positive integer • the use of <*plot options*> is described at the beginning of this section • the use of <*subplot options*> is described at the beginning of this section • if hist() is called without a call to subplot(), a new figure window is created; otherwise, the plot is drawn on an existing multiple-plot figure and at the location specified by <*subplot options*>
[*W* =] kden([<*subplot options*>,] y [, <*plot options*>]);	Plots the kernel density of the values in y. If y contains more than one column then each column is treated as a separate variable; the kernel density is estimated separately and plotted using a different line color on the same plot. If a left-hand-side id value, W, is provided then the title of the figure window on which the function is plotting is stored in W. • y must be a column vector or matrix • the use of <*plot options*> is described at the beginning of this section • the use of <*subplot options*> is described at the beginning of this section • if kden() is called without a call to subplot(), a new figure window is created; otherwise, the plot is drawn on an existing multiple-plot figure and at the location specified by <*subplot options*>

table continues on next page

table continued from previous page

Syntax	Arguments and performed function
[W =] plot([<subplot options>,] y [, x, <plot options>]);	Plots a line plot of the values in y versus either the values in x (if provided) or the row index of each value. If y contains more than one column then each column is plotted as a different variable versus x or the row index and using a different line color for each variable. If a left-hand-side id value, W, is provided then the title of the figure window on which the function is plotting is stored in W. • y must be a column vector or matrix • x must be a column vector with rows equal to the rows of y • the use of <plot options> is described at the beginning of this section • the use of <subplot options> is described at the beginning of this section • if plot() is called without a call to subplot(), a new figure window is created; otherwise, the plot is drawn on an existing multiple-plot figure and at the location specified by <subplot options>
[W =] scatter([<subplot options>,] y, x [, <plot options>]);	Plots a scatter plot of the values in y versus the values in x. If y contains more than one column then each column is plotted as a different variable versus the values in x and using a different color and mark symbol for each variable. If a left-hand-side id value, W, is provided then the title of the figure window on which the function is plotting is stored in W. • y must be a column vector or matrix • x must be a column vector with rows equal to the rows of y • the use of <plot options> is described at the beginning of this section • the use of <subplot options> is described at the beginning of this section • if scatter() is called without a call to subplot(), a new figure window is created; otherwise, the plot is drawn on an existing multiple-plot figure and at the location specified by <subplot options>
[W =] multiplot(i, j)	Initializes a figure window on which multiple plots can be drawn and sets its dimensions: i is the number of plots that can be drawn per column and j is the number of plots that can be drawn per row. That is, the window can be thought of as an area consisting of i rows and j columns, with each cell being able to accommodate a individual plot. If a left-hand-side id value, W, is provided then the title of the figure window which is initialized by the function is stored in W. • i must be a positive integer • j must be a positive integer see also subplot

table continues on next page

table continued from previous page

Syntax	Arguments and performed function
`subplot(s, i, j)`	This function does not form a complete statement on its one and it can only be used as the first argument to the five basic plotting functions: `hist()`, `scatter()`, `acf()`, `plot()`, and `kden()`. When these plotting functions contain a call to `subplot()`, instead of plotting the respective plot on a new figure window, they plot it on the window whose title bar displays the string in `s`. `i` specifies the row of the figure window on which the plot is displayed and `j` the column of this window. • `s` must be a string with contents equal to the string displayed on the title bar of an open figure window • `i` must be a positive integer, not greater than the number of rows of the figure window • `j` must be a positive integer, not greater than the number of columns of the figure window see also `multiplot`

B.17 system, run, pause and eval statements

The statements documented in this section are used to provide access to the machine's operating system's command shell, control program flow or pause execution. Although these statements have very little in common, all of them are likely to be used only by advanced users and, therefore, they are presented together in the following table.

Syntax	Arguments and performed function
system`(s);`	This function submits the contents of string `s` for execution to the machine's command line shell and waits for it to return. Any output from the system's command line shell is directed to the BayES console. Under Microsoft® Windows® the string `"cmd /Q /C "` is prepended to `s`. See section 3.9 for more details. • `s` must be a string with value equal to a complete statement in the machine's command-line language
run`(s);`	Executes the statements contained in the script file, the location and name of which are provided by string `s`. Once a **run** statement is encountered, a new workspace is created and program flow jumps to the first statement in the file. That is, statements contained in the script file do not have access to the items in the calling workspace and any items created during execution of these statements are deleted from memory once execution of the last statement in the script file completes. • `s` must be a string • `s` could be absolute (eg. `"C:/Files/MyScript.bsf"`) or relative to the current working directory (eg. `"./MyScript.bsf"`) see also **eval**

table continues on next page

table continued from previous page

Syntax	Arguments and performed function
eval(s);	Executes the statement contained in string `s`. During execution of the statement in `s` the BayES parser has access to the items in the calling workspace and any items created by executing `s` are stored in the calling workspace. `s` could contain multiple statements. • `s` must be a string • `s` must consist of complete BayES statement(s) see also **run**
pause(i);	Pauses execution of the current script for `i` milliseconds. If `i` is non-positive then this function has no effect. • `i` must be an integer

References

Albert, J. H., & Chib, S. (2001). Sequential ordinal modeling with applications to survival data. *Biometrics*, *57*(3), 829-836.

Burgette, L. F., & Nordheim, E. V. (2012). The trace restriction: An alternative identification strategy for the Bayesian multinomial probit model. *Journal of Business & Economic Statistics*, *30*(3), 404-410.

Chib, S. (1995). Marginal likelihood from the Gibbs output. *Journal of the American Statistical Association*, *90*(432), 1313-1321.

Chib, S. (2001). Markov chain Monte Carlo methods: Computation and inference. In J. J. Heckman & E. E. Leamer (Eds.), *Handbook of econometrics* (Vol. 5, p. 3569-3649). Elsevier.

Chib, S., & Jeliazkov, I. (2001). Marginal likelihood from the Metropolis-Hastings output. *Journal of the American Statistical Association*, *96*(453), 270-281.

Emvalomatis, G. (2011). Adjustment and unobserved heterogeneity in dynamic stochastic frontier models. *Journal of Productivity Analysis*, *37*(1), 7-16.

Greenberg, E. (2013). *Introduction to bayesian econometrics* (2nd ed.). New York, NY: Cambridge University Press.

Greene, W. (2004). Distinguishing between heterogeneity and inefficiency: stochastic frontier analysis of the World Health Organization's panel data on national health care systems. *Health Economics*, *13*(10), 959-980.

Koop, G. (2003). *Bayesian econometrics*. Chichester, UK: John Wiley & Sons.

Lancaster, T. (2004). *An introduction to modern bayesian econometrics*. Oxford, UK: Wiley-Blackwell.

Lewis, S. M., & Raftery, A. E. (1997). Estimating Bayes factors via posterior simulation with the Laplace-Metropolis estimator. *Journal of the American Statistical Association*, *92*(438), 648-655.

Malik, H. J., & Abraham, B. (1973). Multivariate logistic distributions. *The Annals of Statistics*, *1*(3), 588-590.

Tsionas, E. G. (2006). Inference in dynamic stochastic frontier models. *Journal of Applied Econometrics*, *21*(5), 669–676.