BayesX

Software for Bayesian Inference

Version 1.30

Reference Manual

developed at
University of Munich
Department of Statistics
Ludwigstr. 33
80539 Munich

developed by
Andreas Brezger
Thomas Kneib
Stefan Lang

with contributions by
Christiane Belitz
Eva–Maria Fronk
Andrea Hennerfeind
Manuela Hummel
Alexander Jerak
Petra Kragler
Leyre Osuna Echavarría

supported by
Ludwig Fahrmeir (mentally)
Leo Held (mentally)
German Science Foundation

Acknowledgements

The development of *BayesX* has been supported by grants from the German National Science Foundation (DFG), Sonderforschungsbereich 386.

Special thanks go to (in alphabetical order of first names):

Dieter Gollnow for computing and providing the map of Munich (a really hard job);
Leo Held for advertising the program;
Ludwig Fahrmeir for his patience with finishing the program and for carefully reading and correcting the manual;
Ngianga-Bakwin Kandala for being the first user of the program (a really hard job);
Samson Babatunde Adebayo for carefully reading and correcting the manual;
Ursula Becker for carefully reading and correcting the manual;

Licensing agreement

BayesX is available at http://www.stat.uni-muenchen.de/~lang/bayesx

Contents

Chapter 1

What is BayesX?

BayesX is a software tool for performing complex Bayesian inference. The main features of *BayesX* are:

Bayesian semiparametric regression based on MCMC simulation techniques

BayesX provides a powerful regression tool for analyzing regression and survival models with *structured additive predictor* (STAR). STAR models cover a number of well known model classes as special cases, e.g. *generalized additive models*, *generalized additive mixed models*, *geoadditive models*, *dynamic models*, *varying coefficient models*, and *geographically weighted regression*. *BayesX* is able to estimate nonlinear effects of continuous covariates, trends and flexible seasonal patterns of time scales, correlated and/or uncorrelated spatial effects (of geographical data) and unstructured i.i.d. Gaussian effects of unordered group indicators. The regression tool supports the most common distributions for the response variable. Supported distributions for univariate responses are Gaussian, binomial, Poisson, negative binomial, gamma, zero inflated Poisson and zero inflated negative binomial. For multicategorial responses, both multinomial logit or probit models for ordered categories of the responses as well as cumulative threshold models for unordered categories may be estimated. Recently complex models for continuous time survival analysis based on the Cox model have been added. At least some basic knowledge about Bayesian inference with MCMC techniques is strongly recommended if you are interested in using this tool. Details can be found in chapter 7 and the methodology manual.

Inference for STAR models based on methodology for mixed models

BayesX provides a second regression tool for estimating STAR models with comparable functionality as the first tool based on MCMC. This tool represents STAR models as *variance components mixed models*. Inference is then based on estimation procedures for mixed models, particularly *restricted maximum likelihood* (REML). From a Bayesian perspective this yields empirical Bayes or posterior mode estimates. Details can be found in chapter 8 and the methodology manual.

Model selection for Gaussian and non-Gaussian dag's

This tool estimates Gaussian and non-Gaussian directed acyclical graphs (dag) via reversible jump MCMC. Details are given in chapter 10

Handling and manipulation of data sets

BayesX provides a growing number of functions for handling and manipulating data sets, e.g. for reading ASCII data sets, creating new variables, obtaining summary statistics etc. Details are given in chapter 4.

Handling and manipulation of geographical maps

BayesX is able to manipulate and draw geographical maps. The regions of the map may be colored according to some numerical characteristics. For details compare chapter 5 and chapter 6.

Visualizing data

BayesX provides functions for drawing scatter plots and geographical maps. A number of additional options are provided to customize the graphs according to the personal needs of the user. Details can be found in chapter 6.

Recommendations for further reading

If you are interested in using *BayesX* it is not necessary to read the complete manual. Table 1.1 provides a guideline for reading this manual and other sources depending on your purpose and background. In any case, you should read section 2.1-section 2.5 of chapter 2 to make yourself familiar with *BayesX*.

Intended use and background	**Guideline**
Bayesian semiparametric regression based on MCMC simulation techniques. No experience with MCMC techniques.	Read first an introductory text about MCMC. A nice introduction is given e.g. in Green (2001). Read the methodology manual to make yourself familiar with STAR regression models. Proceed then with chapter 1 of the tutorial manual.
Bayesian semiparametric regression based on MCMC simulation techniques. At least a basic knowledge about MCMC techniques exists.	Read the methodology manual to make yourself familiar with STAR regression models. Proceed then with chapter 1 of the tutorial manual.
Semiparametric regression based on mixed model methodology.	Read the methodology manual to make yourself familiar with STAR regression models. Proceed then with chapter 2 of the tutorial manual.
Model selection for dag's. No experience with reversible jump MCMC.	Read first an introductory text about reversible jump MCMC. A nice introduction is given e.g. in Green (2001). Proceed with chapter 10.
Draw and color geographical maps	Read chapter 5 and section 6.1 of chapter 6.

Table 1.1: Recommendations for further reading

References

Green, P.J. (2001): A Primer in Markov Chain Monte Carlo. In: Barndorff-Nielsen, O.E., Cox, D.R. and Klüppelberg, C. (eds.), *Complex Stochastic Systems*. Chapman and Hall, London, 1-62.

Chapter 2

Getting started

This chapter provides information on the different versions of *BayesX*, how to install *BayesX* on your computer and explains the purpose of the different windows that appear after having started *BayesX*. A fourth section covers the general usage of the program and the features of the current version. Finally we describe three data sets, which serve as the basis for most of the illustrating examples in this manual.

2.1 Available versions of BayesX

BayesX is currently available in two versions, a *Java based version* and a *non-Java based version*. Both versions run only under the various versions of the Windows operating system (e.g. Windows 95, 98, 2000, NT, XP). The Java based version is implemented partly in Java, only the computerintensive parts of the program are implemented in C++. The non-Java based version is written completely in C++. The non-Java based version runs slightly faster and uses less disk space than the Java based version. The Java based version has additional features. In particular, functions for visualizing data and estimation results are available only in the Java based version. We recommend the installation of the Java based version. Both versions of *BayesX* can be downloaded at http://www.stat.uni-muenchen.de/~lang/bayesx/bayesx.html.

2.2 Installing BayesX

Installing *BayesX* is very easy. Suppose you have already downloaded one of the installation files, `bayesx.zip` for the non-Java based version or `bayesxjava.exe` for the Java based version. To install the non-Java based version unzip the file `bayesx.zip` using the *winzip* program and store the unzipped files in a temporary directory. Start the program `setup.exe`, for example by double clicking it in the *Windows explorer*. Now follow the instructions of the setup routine to install *BayesX*. To install the Java based version simply execute the file `installBayesX.exe` and follow the installation instructions.

After the successful installation of *BayesX* your installation directory contains five additional subdirectories, namely the directories `doc`, `examples`, `output`, `sfunctions` and `temp`. The `doc` directory contains the program documentation, that is this manual. The `examples` directory contains three data sets, `credit.raw`, `rents.raw` and `zambia.raw`. These data sets exemplify many of the statistical functions and routines described in the following chapters. A detailed description of the three data sets is given in section 2.6. The `examples` directory also contains some tutorial programs that illustrate the usage of most of the functions of *BayesX*, see the tutorials manual. The

`output` directory is the default directory for the program output. The `output` directory can be redefined by the user. The `sfunctions` directory contains some S-plus functions for visualizing estimation results obtained with *bayesreg objects* or *remlreg objects*, see section 9.2 for a detailed description of the S-plus functions. However, by using the Java based version of *BayesX* the S-plus functions are obsolete because this version has its own capabilities for visualizing data and results, see chapter 6 and chapter 9 for details. Finally in the `temp` directory some temporary files will be stored. Normally you will never use this directory.

The created directories and their contents are briefly summarized in Table 2.1.

Directory	Contents
`doc`	contains the manuals
`examples`	contains data set examples and tutorial programs
`output`	default directory for estimation output
`sfunctions`	contains some S-plus functions for visualizing output
`temp`	stores temporary files

Table 2.1: Subdirectories of the installation directory and their content

After a successful installation, *BayesX* can be started using the *Windows Start* button.

2.3 Manuals

BayesX is shipped together with three different manuals. The reference manual gives detailed information on the syntax of *BayesX* commands and the different objects used by *BayesX*. The methodology manual provides background information on the statistical methodology that is implemented in *BayesX*. In this manual you will also find more references on the methodological background. The tutorial manual can be used to make yourself familiar with the usage of *BayesX*. It contains some self-contained tutorials, describing how to perform semiparametric regression analyses using *BayesX*. In the Java based version of *BayesX* the manuals are available from the help menu. In both versions they can be found in the `doc` directory, which is a subdirectory of the installation directory.

2.4 Windows in BayesX

After starting *BayesX* you can see a main window with a menu bar and four additional windows within the main window. The four windows are the *command window*, the *output window*, the *review window* and the *object browser*. The purpose of these windows is described in the following four subsections. Below the menu bar is a second bar containing three buttons named BREAK, PAUSE and SUPPRESS OUTPUT. These buttons are described in the last subsection of this section.

2.4.1 The command window

The *command window* is used to enter and execute commands. By default, a command will be executed if you press the return key. You can change this default delimiter using the `delimiter` command, see section 3.4.

2.4.2 The output window

In the *output window* all commands entered in the *command window* or executed through a batch file (see section 3.5) are printed together with the program output.

The contents of the *output window* can be saved and processed with your favorite text editor. For saving the output, enter the *file menu* and click on *Save output* or *Save output as*. The contents of the *output window* can be saved in two different file formats. The default is the rich-text format. The second choice is to store the *output window* in plain ASCII format. The ASCII format however has the disadvantage that all text highlights (for example bold letters) will disappear in the saved file.

Through the *file menu* you can also clear the *output window* (i.e. delete the contents of the window) or open an already existing file.

Depending on the screen resolution of your computer, letters appearing in the *output window* may be very small or too large. The font size can be changed using the *preferences menu*.

2.4.3 The review window

In many cases subsequent commands change only slightly. The *review window* gives you a convenient way to bring back and edit past commands. In the *review window* the last 100 past commands entered during a session are shown. Click once (double click in the Java based version) on one of these past commands and it is automatically copied to the *command window*, where the command or a slightly modified version can be executed again.

2.4.4 The object browser

BayesX is object oriented although the concept goes not too far. The *object browser* is used to view the contents of the objects currently in memory. The window is split into two parts. The left part shows the different object types currently supported by *BayesX*. These are for the moment *dataset objects*, *bayesreg objects*, *remlreg objects*, *map objects*, *dag objects* and *graph objects*. By clicking on one of the object types the names of all objects of this type will appear in the right panel of the *object browser*. Double clicking on one of the names gives a visualization of the object. The visualization method depends on the respective object type. Double clicking on *dataset objects*, for example, will open a spreadsheet where you can inspect the variables and the observations of the data set. Clicking on *map objects* opens another window that contains a graphical representation of the map.

2.4.5 BREAK, PAUSE and SUPPRESS OUTPUT buttons

As already mentioned, below the menu bar a second panel can be found which contains three buttons, a BREAK button, a PAUSE button and the button SUPPRESS OUTPUT. The BREAK button is used to interrupt the process that is currently executed. Clicking on the PAUSE button interrupts the current process temporarily until the button is pressed again. If a process is paused, the caption PAUSE of the button is replaced by CONTINUE indicating that a second click on the button will continue the current process. Pausing a current process can be used to increase the execution speed of other programs currently running on your computer. Pressing the SUPPRESS OUTPUT button suppresses all output in the *output window*. The button caption changes to SHOW OUTPUT to indicate that an additional click on the button will cause the program to print the output again. Suppressing the output usually increases the execution speed of *BayesX* and saves memory.

2.5 General usage of BayesX

2.5.1 Creating objects

BayesX is object oriented, that is the first thing to do during a session is to create some objects. Currently there are six different object types available: *dataset objects*, *bayesreg objects*, *remlreg objects*, *map objects*, *dag objects* and *graph objects*. *Dataset objects* are used to handle and manipulate data sets, see chapter 4 for details. *Map objects* are used to handle geographical maps and are covered in more detail in chapter 5. The main purpose of *map objects* is to serve as auxiliary objects for estimating spatial covariate effects with *bayesreg objects* or *remlreg objects*. *graph objects* are used to visualize data (e.g. create scatterplots or color geographical maps according to some numerical characteristics), see chapter 6 for details. Probably the most important object types are *bayesreg objects* and *remlreg objects*. These objects are used to estimate Bayesian semiparametric regression models based on either Markov Chain Monte Carlo simulation techniques (*bayesreg objects*) or mixed model representations of the regression model (*remlreg objects*). See chapter 7 for a detailed description of *bayesreg objects* and chapter 8 for a detailed description of *remlreg objects*. Another important object type is the *dag object*. *Dag objects* are used to estimate Gaussian or non-Gaussian dags (direct acyclic graphs) using reversible jump MCMC simulation techniques. A detailed description of *dag objects* can be found in chapter 10. The object oriented concept does not go too far, that is inheritance or other concepts of object oriented programs or languages such as S-plus or C++ are not supported.

Creating a new object during a session is very easy. The syntax for creating a new object is:

> *objecttype objectname*

To create for example a *dataset object* with name `mydata`, simply type:

```
> dataset mydata
```

Note that there are restrictions to the naming of objects, that is some object names are not allowed. For example, one rule is that object names must begin with a (uppercase or lowercase) letter rather than a number; see section 4.13 for valid object names. The section is about valid variable names for data sets, but the same rules apply to object names.

2.5.2 Applying methods of previously defined objects

After the successful creation of an object you can apply methods for that particular object. For instance, *dataset objects* may be used to read in data stored in an ASCII file using method `infile`, to create new variables using method `generate`, to modify existing variables using method `replace` and so on. The syntax for applying methods of the objects is similar for all methods and independent of the particular object type. The general syntax is:

> *objectname. methodname* [*model*] [`weight` *varname*] [`if` *boolean expression*]
> [, *options*] [`using` *usingtext*]

Table 2.2 explains the syntax parts in more detail.

Note that [...] indicates that this part of the syntax is optional and may be omitted. Moreover for most methods only some (or even none) of the syntax parts above are meaningful. Note that the specification of invalid syntax parts is not allowed and will cause an error message.

We illustrate the concept with some simple methods of *dataset objects*. Suppose we have already created a *dataset object* with name `mydata` and want to create some variables for our data set. We first have to tell *BayesX* how many observations we want to create. This can be done with the `set` command, see also section 4.10. For example

Syntax part	Description
objectname	the name of the object to apply the method
methodname	the name of the method
model	a model specification (for example a regression model)
`weight` *varname*	specifies *varname* as a weight variable
`if` *boolean expression*	indicates that the method should be applied only if a certain condition holds
, *options*	define (or modify) options for the method
`using` *usingtext*	indicates that another object or file should be used to apply the particular method

Table 2.2: Syntax parts of methods for objects

```
> mydata.set obs = 1000
```

indicates that the data set `mydata` should have 1000 observations. Here, the *methodname* is `set` and the *model* is `obs = 1000`. Since no other syntax parts (for example `if` statements) are meaningful for this method, they are not allowed. For instance, specifying an additional weight variable `x` by typing

```
> mydata.set obs = 1000 weight x
```

will cause the error message:

```
ERROR: weight statement not allowed
```

In a second step we can now create a new variable `X`, say, that contains Gaussian (pseudo)random numbers with mean 2 and standard deviation 0.5:

```
> mydata.generate X = 2+0.5*normal()
```

Here, `generate` is the *methodname* and `X = 2+0.5*normal()` is the *model*. In this case the *model* consists of the specification of the new variable name, followed by the equal sign '=' and a mathematical expression for the new variable. As is the case with the `set` command other syntax parts are not meaningful and therefore not allowed. Suppose now we want to replace the negative values of `X` with the constant 0. This can be done using the `replace` command by typing:

```
> mydata.replace X = 0 if X < 0
```

Obviously, an additional `if` statement is meaningful and is therefore allowed, but not required.

2.6 Description of data set examples

This section describes the three data sets used to illustrate many of the features of *BayesX*. The three data sets are stored columnwise in plain ASCII-format. The first row of each data set contains the variable names separated by blanks. Subsequent rows contain the observations, one observation per row.

2.6.1 Rents for flats

According to the German rental law, owners of apartments or flats can base an increase in the amount that they charge for rent on 'average rents' for flats comparable in type, size, equipment, quality and location in a community. To provide information about these 'average rents', most of the larger cities publish 'rental guides', which can be based on regression analysis with rent as the dependent variable. The `rent94.raw` file stored in the `examples` directory is a subsample of data collected in 1994 for the rental guide in Munich. The variable of primary interest is the monthly

Variable	Description
R	monthly rent per square meter in German marks
F	floor space in square meters
A	year of construction
L	location of the building in subquarters

Table 2.3: Variables of the rent data set

rent per square meter in German Marks. Covariates characterizing the flat were constructed from almost 200 variables out of a questionnaire answered by tenants of flats. The present data set contains a small subset of these variables that are sufficient for demonstration purposes. Table 2.3 describes the variables of the data set. The data set will be used in the following chapters to demonstrate the usage of *BayesX*.

Additional to the data set, the `examples` directory contains the file `munich.bnd` that contains a map of Munich. This map proves to be useful for visualizing regression results for the explanatory variable location `L` in the data set. See chapter 5 for a description on how to incorporate geographical maps into *BayesX*.

References

Lang, S. and Brezger, A. (2002): Bayesian P-splines. *Journal of Computational and Graphical Statistics*, 13, 183-212.

2.6.2 Credit scoring

The aim of credit scoring is to model or predict the probability that a client with certain covariates ('risk factors') is to be considered as a potential risk, and therefore will probably not pay back his credit as agreed upon by contract. The data set consists of 1000 consumers' credits from a South German bank. The response variable is 'creditability', which is given in dichotomous form ($y = 0$ for creditworthy, $y = 1$ for not creditworthy). In addition, 20 covariates assumed to influence creditability were collected. The present data set (stored in the `examples` directory) contains a subset of these covariates that proved to be the main influential variables on the response variable, see Fahrmeir and Tutz (2001, ch. 2.1). Table 2.4 contains a description of the variables of the data set. Usually a binary logit model is applied to estimate the effect of the covariates on the probability of being not creditworthy. As in the case of the rents for flats example, this data set is used to demonstrate the usage of certain features of *BayesX* , see primarily subsection 7.6.1 for a Bayesian regression analysis of the data set.

References

Fahrmeir, L., Tutz, G. (2001): *Multivariate Statistical Modelling based on Generalized Linear Models.* New York: Springer–Verlag.

2.6.3 Childhood undernutrition in Zambia

Acute and chronic undernutrition is considered to be one of the worst health problems in developing countries. Undernutrition among children is usually determined by assessing the anthropometric status of the child relative to a reference standard. In our example undernutrition is measured through stunting or insufficient height for age, indicating chronic undernutrition. Stunting for a

Variable	Description
y	creditability, dichotomous with $y = 0$ for creditworthy, $y = 1$ for not creditworthy
account	running account, trichotomous with categories "no running account" (= 1), "good running account" (= 2), "medium running account" ("less than 200 DM") (= 3)
duration	duration of credit in months, metrical
amount	amount of credit in 1000 DM, metrical
payment	payment of previous credits, dichotomous with categories "good" (= 1), "bad" (= 2)
intuse	intended use, dichotomous with categories "private" (= 1) or "professional" (= 2)
marstat	marital status, with categories "married" (= 1) and "living alone" (= 2).

Table 2.4: Variables of the credit scoring data set

child i is determined using a Z-score which is defined as

$$Z_i = \frac{AI_i - MAI}{\sigma}$$

where AI refers to the child's anthropometric indicator (height at a certain age in our example), MAI refers to the median of the reference population and σ refers to the standard deviation of the reference population.

The data set contains the (standardized) Z-score for 4847 children together with several covariates that are supposed to have influence on undernutrition including the body mass index of the child's mother, the age of the child and the district the child lives in. Table 2.5 gives more information on the covariates in the data set.

This data set is used in chapter 1 and 2 of the tutorial manual.

Variable	Description
hazstd	standardized Z-score of stunting
bmi	body mass index of the mother
agc	age of the child
district	district where the child lives
rcw	mother's employment status with categories "working" (= 1) and "not working" (= −1)
*edu*1	mother's educational status with categories "complete primary but incomplete
*edu*2	secondary" (*edu*1 = 1), "complete secondary or higher" (*edu*2 = 1) and "no education or incomplete primary" (*edu*1 = *edu*2 = −1)
tpr	locality of the domicile with categories "urban" (= 1) and "rural" (= −1)
sex	gender of the child with categories "male" (= 1) and "female" (= −1)

Table 2.5: Variables in the undernutrition data set.

References

Kandala, N. B., Lang, S., Klasen, S. and Fahrmeir, L. (2001): Semiparametric Analysis of the Socio-Demographic and Spatial Determinants of Undernutrition in Two African Countries.*Research in Official Statistics*, 1, 81-100.

Chapter 3

Special Commands

This chapter describes some commands that are not connected with a particular object type. Among others, there are commands for exiting *BayesX*, opening and closing log files, saving program output, dropping objects etc..

3.1 Exiting BayesX

You can exit *BayesX* by simply typing either

```
> exit
```

or

```
> quit
```

in the *command window*.

3.2 Opening and closing log files

In a log file, program output and commands entered by the user, are stored in plain ASCII format. This makes it easy to further use the program output, for example results of statistical procedures, in your favorite text editor. Another important application of log files is the documentation of your work. You open a log file by typing:

`> logopen` [, *option*] `using` *filename*

This opens a log file that will be saved in *filename*. After opening a log file, all commands entered and all program output appearing on the screen will be saved in this file. If the log file specified in *filename* is already existing, new output is appended at the end of the file. To overwrite an existing log file option `replace` must be specified in addition. Note that it is not allowed to open more than one log file simultaneously.

An open log file can be closed by simply typing:

```
> logclose
```

Note that exiting *BayesX* automatically closes the currently open logfile.

3.3 Saving the contents of the output window

You can save the contents of the *output window* not only with the *file->save output* or *file->save output as* menu, but also using the `saveoutput` command. Saving the *output window* with the

`saveoutput` command is particularly useful in batch files, see section 3.5. The syntax for saving the *output window* is

`> saveoutput` [, *options*] `using` *filename*

where *filename* is the file (including path) in which the contents of the output will be saved.

Options

- `replace`
  By default, an error will be raised if one tries to store the contents of the *output window* in a file that is already existing. This preserves you from overwriting a file unintentionally. An already existing file can be overwritten by explicitly specifying the `replace` option.

- `type = rtf | txt`
  The *output window* can be saved under two different file types. By default, the contents of the window will be saved in rich-text format. The second possibility is to store the *output window* in plain ASCII–format. This can be done by specifying `type = txt`. To explicitly store the file in rich-text format `type = rtf` must be specified.
  DEFAULT: `type = rtf`

3.4 Changing the delimiter

By default, commands entered using the *command window* will be executed by pressing the return key. This can be inconvenient, in particular if your statements are long. In that case it may be more favorable to split a statement into several lines, and execute the command using a different delimiter than the return key. You can change the delimiter using the `delimiter` command. The syntax is

`> delimiter` = *newdel*

where *newdel* is the new delimiter. There are only two different delimiters allowed, namely the return key and the ';' (semicolon) key. To specify the ';' key as the delimiter, type

```
> delimiter = ;
```

and press return. To return to the return key as the delimiter, type

```
> delimiter = return;
```

Note that the above statement must end with a semicolon, since this was previously set to the current delimiter.

3.5 Using batch files

You can execute commands stored in a file just as if they were entered from the keyboard. This may be useful if you want to re-run a certain analysis more than once (possibly with some minor changes) or if you want to run time consuming statistical methods such as Bayesian regression based on MCMC simulation techniques (see chapter 7). You can run such batch files by simply typing

`> usefile` *filename*

This executes the commands stored in *filename* successively. *BayesX* will not stop the execution if an error occurs in one or more commands. Note that it is allowed to invoke another batch file within a batch file currently running.

Comments

Comments in batch files are allowed and are indicated by a `%` sign, that is every line starting with a `%` sign is ignored by the program.

Changing the delimiter

In particular in batch files, the readability of your program code may be improved if some (long) commands are split up into several lines. Normally this will cause errors, because *BayesX* interprets each line in your program as one statement. To overcome this problem one simply has to change the delimiter using the `delimiter` command, see section 3.4.

3.6 Dropping objects

You can delete objects by typing

`> drop` *objectlist*

This drops the objects specified in *objectlist*. The names of the objects in *objectlist* must be separated by blanks.

Chapter 4

dataset objects

Authors: Stefan Lang, Christiane Belitz and Manuela Hummel
email: lang@stat.uni-muenchen.de

Dataset objects are used to manage and manipulate data. A new *dataset object* is created by typing

`> dataset` *objectname*

where *objectname* is the name of the data set. After the creation of a *dataset object* you can apply the methods for manipulating and managing data sets discussed below.

Note that in the current version of *BayesX* **only numerical variables are allowed**. Hence, string valued variables, for example, are not yet supported by *BayesX*.

4.1 Method descriptive

Description

Method `descriptive` calculates and displays univariate summary statistics. The method computes the number of observations, the mean, median, standard deviation, minimum and maximum of variables.

Syntax

`>` *objectname*.`descriptive` *varlist* [`if` *expression*]

Method `descriptive` computes summary statistics for the variables in *varlist*. An optional `if` statement may be added to analyze only a part of the data.

Options

not allowed

Example

The statement

```
> d.descriptive x y
```

computes summary statistics for the variables `x` and `y`. The statement

```
> d.descriptive x y if x>0
```

restricts the analysis to observations with `x>0`.

4.2 Method drop

Description

Method `drop` deletes variables or observations from the data set.

Syntax

> *objectname*.`drop` *varlist*

> *objectname*.`drop if` *expression*

The first command may be used to eliminate the variables specified in *varlist* from the data set. The second statement may be used to eliminate certain observations. An observation will be removed from the data set if *expression* is true, i.e. the value of the expression is one.

Options

not allowed

Example

The statement

```
> credit.drop account duration
```

drops the variables `account` and `duration` from the credit scoring data set. With the statement

```
> credit.drop if  marstat = 2
```

all observations with `marstat=2`, i.e. all persons living alone, will be dropped from the credit scoring data set. The following statement

```
> credit.drop account duration if marstat = 2
```

will raise the error

```
ERROR: dropping variables and observations in one step not allowed
```

It is not allowed to drop variables and certain observations in one single command.

4.3 Functions and Expressions

The primary use of expressions is to generate new variables or change existing variables, see section 4.4 and section 4.9, respectively. Expressions may also be used in `if` statements to force *BayesX* to apply a method only to observations where the boolean expression in the `if` statement is true. The following are all examples of expressions:

```
2+2
log(amount)
1*(age <= 30)+2*(age > 30 & age <= 40)+3*(age > 40)
age=30
age+3.4*age^2+2*age^3
amount/1000
```

4.3.1 Operators

BayesX has three different types of operators: arithmetic, relational and logical. Each of the types is discussed below.

4.3.1.1 Arithmetic operators

The arithmetic operators are `+` (addition), `-` (subtraction), `*` (multiplication), `/` (division), `^` (raise to a power) and the prefix `-` (negation). Any arithmetic operation on a missing value or an impossible arithmetic operation (such as division by zero) yields a missing value.

Example

The expression

`(x+y^(3-x))/(x*y)`

denotes the formula

$$\frac{x + y^{3-x}}{x \cdot y}$$

and evaluates to missing if `x` or `y` is missing or zero.

4.3.1.2 Relational operators

The relational operators are `>` (greater than), `<` (less than), `>=` (greater than or equal), `<=` (less than or equal), `=` (equal) and `!=` (not equal). Relational expressions are either 1 (i.e. the expression is true) or 0 (i.e. the expression is false).

Example

Relational operators may be used to create indicator variables. The following statement generates a new variable `amountcat` (out of the already existing variable `amount`), whose value is 1 if `amount<=10` and 2 if `amount>10`.

```
> credit.generate amountcat = 1*(amount<=10)+2*(amount>10)
```

Another useful application of relational operators is in `if` statements. For example, changing an existing variable only when a certain condition holds can be done by the following command:

```
> credit.replace amount = NA if amount <= 0
```

This sets all observations missing where `amount<=0`.

4.3.1.3 Logical operators

The logical operators are `&` (and) and `|` (or).

Example

Suppose you want to generate a variable `amountind` whose value is 1 for married people with amount greater than 10 and 0 otherwise. This can be done by typing

```
> credit.generate amountind = 1*(marstat=1 & amount > 10)
```

4.3.1.4 Order of evaluation of the operators

The order of evaluation (from first to last) of operators is
`^`
`/`, `*`
`-`, `+`
`!=`, `>`, `<`, `<=`, `>=`, `=`
`&`, `|`.
Brackets may be used to change the order of evaluation.

4.3.2 Functions

Functions may appear in expressions. Functions are indicated by the function name, an opening and a closing parenthesis. Inside the parentheses one or more arguments may be specified. The argument(s) of a function may be any expression, including other functions. Multiple arguments are separated by commas. All functions return missing values when given missing values as arguments or when the result is undefined.

Functions reference

Table 4.1 references all mathematical functions; Table 4.2 references all statistical functions.

4.3.3 Constants

Table 4.3 lists all constants that may be used in expressions.

Example

The following statement generates a variable `obsnr` whose value is 1 for the first observation, 2 for the second and so on.

```
> credit.generate obsnr = _n
```

The command

```
> credit.generate nrobs = _N
```

Function	Description
abs(x)	absolute value
cos(x)	cosine of radians
exp(x)	exponential
floor(x)	returns the integer obtained by truncating x. Thus floor(5.2) evaluates to 5 as floor(5.8).
lag(x)	lag operator
log(x)	natural logarithm
log10(x)	log base 10 of x
sin(x)	sine of radians
sqrt(x)	square root

Table 4.1: List of mathematical functions.

generates a new variable *nrobs* whose values are equal to the total number of observations, say 1000, for all observations.

4.3.4 Explicit subscribing

Individual observations on variables can be referenced by subscribing the variables. Explicit subscripts are specified by the variable name with square brackets that contain an expression. The result of the subscript expression is truncated to an integer, and the value of the variable for the indicated observation is returned. If the value of the subscript expression is less than 1 or greater than the number of observations in the data set, a missing value is returned.

Example

Explicit subscribing combined with the constant `_n` (see Table 4.3) can be used to create lagged values on a variable. For example the lagged value of a variable `x` in a data set `data` can be created by

```
> data.generate xlag = x[_n-1]
```

Note that `xlag` can also be generated using the `lag` function

```
> data.generate xlag = lag(x)
```

Function	**Description**
bernoulli(p)	returns Bernoulli distributed random numbers with probability of success p. If p is not within the interval $[0;1]$, a missing value will be returned.
binomial(n,p)	returns $B(n;p)$ distributed random numbers. Both, the number of trials n and the probability of success p may be expressions. If $n<1$, a missing value will be returned. If n is not integer valued, the number of trials will be $[n]$. If p is not within the interval $[0;1]$, a missing value will be returned.
cumul(x)	cumulative distribution function
cumulnorm(x)	cumulative distribution function Φ of the standard normal distribution.
exponential(λ)	returns exponentially distributed random numbers with parameter λ. If $\lambda \leq 0$, a missing value will be returned.
gamma(μ,ν)	returns gamma distributed random numbers with mean μ and variance μ^2/ν. If μ and/or ν are less than zero, a missing value will be returned.
normal()	returns standard normally distributed random numbers; $N(\mu,\sigma^2)$ distributed random numbers may be generated with μ + σ*normal().
uniform()	uniform pseudo random number function; returns uniformly distributed pseudo-random numbers on the interval $(0,1)$

Table 4.2: List of statistical functions

Constant	Description
`_n`	contains the number of the current observation.
`_N`	contains the total number of observations in the data set.
`_pi`	contains the value of π.
`NA`	indicates a missing value
.	indicates a missing value

Table 4.3: List of constants

4.4 Method generate

Description

`generate` is used to create a new variable.

Syntax

> *objectname*.`generate` *newvar* = *expression*

Method `generate` creates a new variable with name *newvar*. See section 4.13 for valid variable names. The values of the new variable are specified by *expression*. The details of valid expressions are covered in section 4.3.

Options

not allowed

Example

The following command generates a new variable called `amount2` whose values are the square of amount in the credit scoring data set.

```
> credit.generate amount2 = amount^2
```

If you try to change the variable currently generated, for example by typing

```
> credit.generate amount2 = amount^0.5
```

the error message

```
ERROR: variable amount2 is already existing
```

will occur. This prevents you to change an existing variable unintentionally. An existing variable may be changed with method `replace`, see section 4.9.

If you want to generate an indicator variable `largeamount` whose value is 1 if `amount` exceeds a certain value, say 3.5, and 0 otherwise, the following will produce the desired result:

```
> credit.generate largeamount = 1*(amount>3.5)
```

4.5 Method infile

Description

Reads in data saved in an ASCII file.

Syntax

> *objectname*.`infile` [*varlist*] [, *options*] `using` *filename*

Reads in data stored in *filename*. The variables are given names specified in *varlist*. If *varlist* is empty, i.e. there is no *varlist* specified, it is assumed that the first row of the datafile contains the variable names separated by blanks or tabs. It is not required that the observations in the datafile are stored in a special format, except that successive observations should be separated by one or more blanks (or tabs). The first value read from the file will be the first observation of the first variable, the second value will be the first observation of the second variable, and so on. An error will occur if for some variables no values can be read for the last observation.

It is assumed that a period '.' or 'NA' indicates a missing value.

Note that in the current version of *BayesX* **only numerical variables are allowed**. Thus, the attempt to read in string valued variables, for example, will cause an error.

Options

- `missing =` *missingsigns*
  By default a dot '.' or 'NA' indicates a missing value. If you have a data set where missing values are indicated by different signs than '.' or 'NA', you can force *BayesX* to recognize these signs as missing values by specifying the `missing` option. For example `missing = MIS` defines MIS as an indicator for a missing value. Note that '.' and 'NA' remain valid indicators for missing values, even if the missing option is specified.

- `maxobs =` *integer*
  If you work with large data sets, you may observe the problem that reading in a data set using the `infile` command is very time consuming. The reason for this problem is that *BayesX* does not know the number of observations and thus the memory needed in advance. The effect is that new memory must be allocated whenever a certain amount of memory is used. To avoid this problem the `maxobs` option may be used, leading to a considerable reduction of computing time. This option forces *BayesX* to allocate in advance enough memory to store at least *integer* observations before new memory must be reallocated. Suppose for example that your data set consists approximately of 100,000 observations. Then specifying `maxobs = 105000` allocates enough memory to read in the data set quickly. Note that `maxobs = 105000` does not mean that your data set cannot hold more than 105,000 observations. This means only that new memory will/must be allocated when the number of observations of your data set exceeds the 105,000 observations limit.

Example

Suppose we want to read a data set stored in `c:\data\testdata.raw` containing two variables `var1` and `var2`. The first few rows of the datafile could look like this:

var1 var2
2 2.3

3 4.5
4 6
...

To read in this data set, we first have to create a new *dataset object*, say `testdata`, and then read the data using the `infile` command. The following two commands will produce the desired result.

```
> dataset testdata
> testdata.infile using c:\data\testdata.raw
```

If the first row of the data set file contains no variable names, the second command must be modified to:

```
> testdata.infile var1 var2 using c:\data\testdata.raw
```

Suppose furthermore that the data set you want to read in is a pretty large data set with 100,000 observations. In that case the `maxobs` option is very useful to reduce reading time. Typing for example

```
> testdata.infile var1 var2 , maxobs=101000 using c:\data\testdata.raw
```

will produce the desired result.

4.6 Method outfile

Description

Method `outfile` writes data to a disk file in ASCII format. The saved data can be read back using the `infile` command, see section 4.5.

Syntax

> *objectname*.`outfile` [*varlist*] [`if` *expression*] [*, options*] `using` *filename*

`outfile` writes the variables specified in *varlist* to the disk file with name *filename*. If *varlist* is omitted in the outfile statement, *all* variables in the data set are written to disk. Each row in the data file corresponds to one observation. Different variables are separated by blanks. Optionally, an `if` statement may be used to write only those observations to disk where a certain boolean expression, specified in *expression*, is true.

Options

- `header`
  Specifying the `header` option forces *BayesX* to write the variable names in the first row of the created data file.
- `replace`
  The `replace` option allows *BayesX* to overwrite an already existing data file. If `replace` is omitted in the option list and the file specified in *filename* is already existing, an error will be raised. This prevents you from overwriting an existing file unintentionally.

Example

The statement

```
> credit.outfile using c:\data\cr.dat
```

writes the complete credit scoring data set to `c:\data\cr.dat`. To generate two different ASCII data sets for married people and people living alone, you could type

```
> credit.outfile if marstat = 1 using c:\data\crmarried.dat
> credit.outfile if marstat = 2 using c:\data\cralone.dat
```

Suppose you only want to write the two variables `y` and `amount` to disk. You could type

```
> credit.outfile y amount using c:\data\cr.dat
```

This will raise the error message

```
ERROR: file c:\data\cr.dat is already existing
```

because `c:\data\cr.dat` has already been created. You can overwrite the file using the `replace` option

```
> credit.outfile y amount , replace using c:\data\cr.dat
```

4.7 Method pctile

Description

Method `pctile` computes and displays the 1%,5%,25%,50%,75%,95% and 99% percentiles of a variable.

Syntax

`>` *objectname*`.pctile` *varlist* [`if` *expression*]

Method `pctile` computes and displays the percentiles of the variables specified in *varlist*. An optional `if` statement may be added to compute the percentiles only for a part of the data.

Options

not allowed

Example

The statement

```
> d.pctile x y
```

computes percentiles for the variables `x` and `y`. The statement

```
> d.pctile x y if x>0
```

restricts the analysis to observations with x>0.

4.8 Method rename

Description

`rename` is used to change variable names.

Syntax

> *objectname*.`rename` *varname newname*

`rename` changes the name of *varname* to *newname*. *newname* must be a valid variable name, see section 4.13 on how to create valid variable names.

Options

not allowed

4.9 Method replace

Description

`replace` changes the values of an existing variable.

Syntax

> *objectname*.`replace` *varname* = *expression* [`if` *boolexp*]

`replace` changes the values of the existing variable *varname*. If *varname* is not existing, an error will be raised. The new values of the variable are specified in *expression*. Expressions are covered in section 4.3. An optional `if` statement may be used to change the values of the variable only if the boolean expression *boolexp* is true.

Options

not allowed

Example

The statement

```
> credit.replace amount = NA if amount<0
```

changes the values of the variable `amount` in the credit scoring data set to missing if `amount<0`.

4.10 Method set obs

Description

`set obs` changes the current number of observations in a data set.

Syntax

> *objectname*.`set obs` = *intvalue*

`set obs` raises the number of observations in the data set to *intvalue*, which must be greater or equal to the current number of observations. This prevents you from deleting parts of the data currently in memory. Observations may be eliminated using the `drop` statement, see section 4.2. The values of the additionally created observations will be set to the missing value.

4.11 Method sort

Description

Sorts the data set.

Syntax

> *objectname*.`sort` *varlist* [, *options*]

Sorts the data set with respect to the variables specified in *varlist*. Missing values are interpreted to be larger than any other number and are thus placed last.

Options

- `descending`
  If this option is specified, the data set will be sorted in descending order. The default is ascending order.

4.12 Method tabulate

Description

Method `tabulate` calculates and displays a frequency table for a variable.

Syntax

> *objectname*.`tabulate` *varlist* [`if` *expression*]

Method `tabulate` computes and displays frequency tables of the variables specified in *varlist*. An optional `if` statement may be added to restrict the analysis to a part of the data.

Options

not allowed

Example

The statement

```
> d.tabulate x y
```

displays frequency tables for the variables `x` and `y`. The statement

```
> d.tabulate x y if x>0
```

restricts the analysis to observations with `x>0`.

4.13 Variable names

A valid variable name is a sequence of letters (`A-Z` and `a-z`), digits (`0-9`), and underscores (`_`). The first character of a variable name must either be a letter or an underscore. *BayesX* respects upper and lower case letters, that is `myvar`, `Myvar` and `MYVAR` are three distinct variable names.

4.14 Example

This section contains two examples on how to work with *dataset objects*. The first example illustrates some of the methods described above, using one of the example data sets stored in the `examples` directory, the credit scoring data set. A description of this data set can be found in subsection 2.6.2. The second example shows how to simulate complex statistical models.

4.14.1 The credit scoring data set

In this section we illustrate how to code categorical variables according to one of the coding schemes, dummy or effect coding. This will be useful in regression models, where all categorical covariates must be coded in dummy or effect coding before they can added to the model.

We first create a *dataset object* `credit` and read in the data using the `infile` command.

```
> dataset credit
> credit.infile using c:\bayes\examples\credit.raw
```

We can now generate new variables to obtain dummy coded versions of the categorical covariates `account`, `payment`, `intuse` and `marstat`:

```
> credit.generate account1  = 1*(account=1)
> credit.generate account2  = 1*(account=2)
> credit.generate payment1 = 1*(payment=1)
> credit.generate intuse1 = 1*(intuse=1)
> credit.generate marstat1 = 1*(marstat=1)
```

The reference categories are chosen to be 3 for `account` and 2 for the other variables. Alternatively, we could code the variables according to effect coding. This is achieved with the following program code:

```
> credit.generate account_eff1  = 1*(account=1)-1*(account=3)
> credit.generate account_eff2  = 1*(account=2)-1*(account=3)
> credit.generate payment_eff1 = 1*(payment=1)-1*(payment=2)
> credit.generate intuse_eff1 = 1*(intuse=1)-1*(intuse=2)
> credit.generate marstat_eff1 = 1*(marstat=1)-1*(marstat=2)
```

4.14.2 Simulating complex statistical models

In this section we illustrate how to simulate complex regression models. Suppose first we want to simulate data according to the following Gaussian regression model:

$$
\begin{aligned}
y_i &= 2 + 0.5x_{i1} + \sin(x_{i2}) + \epsilon_i, \quad i = 1, \ldots, 1000 && (4.1)\\
x_{i1} &\sim U(-3, 3) \quad i.i.d. && (4.2)\\
x_{i2} &\sim U(-3, 3) \quad i.i.d. && (4.3)\\
\epsilon_i &\sim N(0, 0.5^2) \quad i.i.d. && (4.4)
\end{aligned}
$$

We first have to create a new data set `gsim`, say, and specify the desired number of observations:

```
> dataset gsim
> gsim.set obs = 1000
```

In a second step the covariates `x1` and `x2` have to be created. In this first example we assume that the covariates are uniformly distributed between -3 and 3. To generate them, we must type:

```
> gsim.generate x1 = -3+6*uniform()
> gsim.generate x2 = -3+6*uniform()
```

In a last step we can now create the response variable by typing

```
> gsim.generate y = 2 + 0.5*x1+sin(x2)+0.5*normal()
```

You could now (if you wish) estimate a Gaussian regression model with the generated data set using one of the regression tools of *BayesX*, see chapter 7 or chapter 8. Of course, more refined models could be simulated. We may for example drop the assumption of a constant variance of 0.5^2 in the error term. Suppose the variance is heteroscedastic and growing with order $\log(i)$ where i is the observation index. We can simulate such a heteroscedastic model by typing:

```
> gsim.replace y = 2 + 0.5*x1+sin(x2)+0.1*log(_n+1)*normal()
```

In this model the standard deviation is

$$\sigma_i = 0.1 * \log(i+1), \quad i = 1, \ldots, 1000.$$

Suppose now that we want to simulate data from a logistic regression model. In a logistic regression model it is assumed that (given the covariates) the response variable y_i, $i = 1, \ldots, n$, is binomially distributed with parameters n_i and π_i where n_i is the number of replications and π_i is the probability of success. For π_i one assumes that it is related to a linear predictor η_i via the logistic distribution function, that is

$$\pi_i = \frac{\exp(\eta_i)}{1 + \exp(\eta_i)}.$$

To simulate such a model we have to specify the linear predictor η_i and the number of replications n_i. We specify a similar linear predictor as in the example above for Gaussian response, namely

$$\eta_i = -1 + 0.5x_{i1} - \sin(x_{i2}).$$

For simplicity, we set $n_i = 1$ for the number of replications. The following commands generate a data set 'bin' according to the specified model:

```
> dataset bin
> bin.set obs = 1000
> bin.generate x1 = -3+6*uniform()
> bin.generate x2 = -3+6*uniform()
> bin.generate eta = -1+0.5*x1-sin(x2)
> bin.generate pi = exp(eta)/(1+exp(eta))
> bin.generate y = binomial(1,pi)
```

Note that the last three statements can be combined into a single command:

```
> bin.generate y = binomial(1,exp(-1+0.5*x1-sin(x2))/(1+exp(-1+0.5*x1-sin(x2))))
```

The first version however is much easier to read and should therefore be preferred.

Chapter 5

map objects

Authors: Stefan Lang and Andreas Brezger
email: lang@stat.uni-muenchen.de and andib@stat.uni-muenchen.de

Map objects are used to handle and store geographical maps. For the moment *map objects* serve more or less as auxiliary objects for *bayesreg objects* and *remlreg objects*, where the effect of spatial covariates on a dependent variable can be modelled via Markov random field priors. The main purpose of *map objects* in this context is to provide the neighborhood structure of the map and to compute weights associated with this neighborhood structure. The typical approach is as follows: A *map object* is created and the boundary information of a geographical map is read from an external file and stored in the *map object*. This can be achieved using the `infile` command, see section 5.1 below. Based on the boundary information, the *map object* automatically computes the neighborhood structure of the map and the weights associated with the neighborhood structure. Since there are several proposals in the spatial statistics literature for defining the weights, the user is given the choice between a couple of alternative weight definitions. After the correct initialization, the *map object* can be passed to the `regress` function of a *bayesreg object* in order to estimate regression models with spatial covariates, see chapter 7, in particular section 7.1, and the subsections about spatial covariates therein.

5.1 Method infile

5.1.1 Description

Method `infile` is used to read the boundary information of a geographical map stored in an external file. This file is called a *boundary file*, since it must contain the information about the boundaries of the different regions of the map. It is assumed that the boundary of each region is stored in form of a closed polygon, that is the boundary is represented by a set of connected straight lines. A detailed description of the structure of boundary files is given below.

As a second choice method `infile` allows to read so called *graph files*. In *graph files* the nodes and edges of a certain graph are stored. In addition, weights associated with the edges of the graph may be specified in the file. In terms of geographical maps the nodes of a graph correspond to the regions of a map and the edges specify the neighborhood structure of the map. The main advantage of *graph files* is that the neighborhood structure of a particular geographical map is already available. With *boundary files* the neighborhood structure must be computed first; a task which is relatively computer intensive.

Usually *boundary files* are read only once to compute the neighborhood structure of a geographical map. Having computed the neighborhood structure, the map can be stored as a *graph file* using the `outfile` command, see section 5.2. In subsequent sessions typically the *graph file* is used rather than the corresponding *boundary file* because reading *graph files* is less computer intensive.

5.1.2 Syntax

> *objectname*.`infile` [, *options*] `using` *filename*

Method `infile` reads the map information stored in the *boundary* or *graph file filename*. If option `graph` is specified, *BayesX* expects a *graph file*, otherwise a *boundary file* is expected. The structures of *boundary* and *graph files* are described below.

Structure of a boundary file

A *boundary file* provides the boundary information of a geographical map. For each region of the map the *boundary file* must contain the identifying name of the region, the polygons that form the boundary of the region, and the number of lines the polygon consists of. The first line always contains the region code surrounded by quotation marks and the number of lines the polygon of the region consists of. The code and the number of lines must be separated by a comma. The subsequent lines contain the coordinates of the straight lines that form the boundary of the region. The straight lines are represented by the coordinates of their end points. Coordinates must be separated by a comma.

To give an example we print a (small) part of the *boundary file* of Germany:

⋮

"6634",31
2319.26831,4344.48828
2375.45435,4399.50391
2390.67139,4446.32520
2470.26807,4405.35645
2576.78735,4379.60449
2607.22144,4337.46533
2627.12061,4356.19385
2662.23682,4355.02344
2691.50024,4311.71338

2726.61646,4310.54248
2716.08154,4256.69775
2710.22900,4227.43408
2680.96533,4234.45752
2583.81055,4165.39551
2568.59351,4096.33398
2520.60132,4042.48901
2535.81836,3941.82251
2490.16724,3920.75269
2451.53955,3903.19458
2437.49292,3924.26440
2369.60156,3933.62866
2359.06665,3951.18677
2285.32275,3969.91553
2258.40015,4061.21753
2197.53223,4049.51221
2162.41602,4086.96948
2204.55542,4091.65161
2192.85010,4125.59717
2284.15210,4220.41113
2339.16748,4292.98438
2319.26831,4344.48828
⋮

The map corresponding to the section of the *boundary file* above can be found in Figure 5.1. Note that the first and the last point must be identical (see the example above) to obtain a closed polygon.

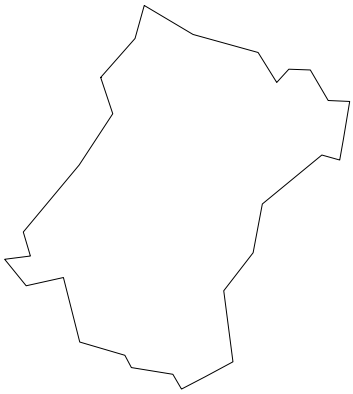


Figure 5.1: Corresponding graph of the section of the boundary file

In some cases it might happen that a region is separated into subregions that are not connected. As an illustrative example compare Figure 5.2 showing a region of Germany that is separated into 8 subregions. In this case the *boundary file* must contain the polygons of all subregions. The first row for each of these subregions must contain the region code and the number of lines the polygon of the respective subregion consists of. Note that it is not necessary that the polygons of the subregions are stored in subsequent order in the *boundary file*.

Another special case that might occur is illustrated in Figure 5.3. Here a region is completely surrounded by another region. In this case an additional line must be added to the boundary description of the *surrounded* region. The additional line must be placed after the first line and must contain the region code of the *surrounding* region. The syntax is:

`is.in`,"*region code*"

The following lines show a section of the *boundary file* of Germany, where region "9361" is totally

surrounded by region "9371":

```
        ⋮
"9361",7
is.in,"9371"
4155.84668,2409.58496
4161.69922,2449.38330
4201.49756,2461.08862
4224.90820,2478.64673
4250.66016,2418.94922
4193.30371,2387.34448
4155.84668,2409.58496
        ⋮
```

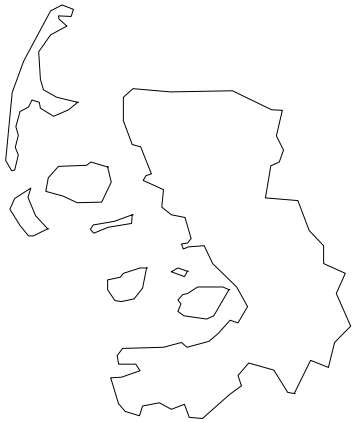


Figure 5.2: Example for a region that is divided into subregions

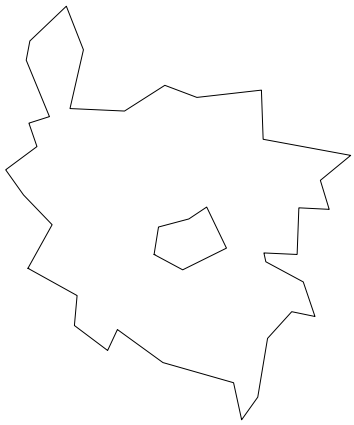


Figure 5.3: Example for a region that is totally surrounded by another region

Finally, we want to draw attention to an important limitation in the current version of *BayesX*. In most cases *map objects* serve as auxiliary objects to estimate spatial effects with *bayesreg objects* or *remlreg objects*. In this case the names of the regions of the map and the values of the spatial covariate, whose effect is estimated, must match. Since there are only numerical variables allowed in *dataset objects* (and no string valued variables), the names of the regions in the corresponding *map object* must necessarily be numbers, although there is in principle no limitation for the names of regions in *map objects*.

Structure of a graph file

A graph file stores the nodes and the edges of a graph $G = (N, E)$, see for example George and Liu (1981, Ch. 3) for a first introduction into graph theory. A graph is a convenient way of representing the neighborhood structure of a geographical map. The nodes of the graph correspond to the region codes. The neighborhood structure is represented by the edges of the graph. In some situations it may be useful to define weights associated with the edges of a graph which can be stored in the *graph file* as well.

We now describe the structure of a *graph file* as it is expected by *BayesX*. The first line of a *graph file* must contain the total number of nodes of the graph. In the remaining lines, the nodes of the graph together with their edges and associated weights are specified. One node corresponds to three consecutive lines. The first of the three lines must contain the name of the node, which may simply be the name of a geographical region. In the second line the number of edges of that particular node is given. The third line contains the corresponding edges of the node, where an edge is given by the index of a neighboring node. The index starts with zero. For example, if the fourth and the seventh node/region in the *graph file* are connected/neighbors, the edge index for the fourth node/region is 6 and for the seventh node/region 3.

We illustrate the structure of a *graph file* with an example. The following few lines are the beginning of the *graph file* corresponding to the map of (former) West Germany:

```
327
9162
3
1 2 3
9174
6
0 4 2 3 5 6
9179
6
0 1 7 3 8 6
      ⋮
```

The first line specifies the total number of nodes, in the present example 327 nodes. The subsequent three lines correspond to the node with name '9162', which is the first region in the map of West Germany. Region '9162' has 3 neighbors, namely the second, third and fourth node appearing in the graph file. Once again, note that the index starts with zero, i.e. 0 corresponds to the first node, 1 corresponds to the second node and so on. Lines 5 to 7 in the example correspond to node '9174' and its neighbors and lines 8 to 10 correspond to node '9179'.

In a *graph file* it is also possible to specify weights associated with the edges of the nodes. Since in the preceding example no weights are explicitly specified, all weights are automatically defined to be equal to one. Nonequal weights are specified in the *graph file* by simply adding them following the edges of a particular node. An example of the beginning of a *graph file* with weights is given below:

```
327
9162
3
1 2 3 0.4 1.2 0.7
9174
6
0 4 2 3 5 6 0.4 0.3 0.8 0.8 1.4 1.6
9179
6
0 1 7 3 8 6 1.2 0.8 0.2 1.8 1.7 1.3
```

⋮

Here the edges of the first node '9162' have weights 0.4, 1.2 and 0.7.

Options

- `graph`
  If `graph` is specified as an additional option, *BayesX* expects a *graph file* to be read in rather than a *boundary file*.

- `weightdef=adjacency` | `centroid` | `combnd`
  Option `weightdef` allows to specify how the weights associated with each pair of neighbors are computed. Currently there are three weight specifications available, `weightdef=adjacency`, `weightdef=centroid` and `weightdef=combnd`. If `weightdef=adjacency` is specified, for each pair of neighbors the weights are set equal to one. The so called adjacency weights are the most common ones in spatial statistics. Specifying `weightdef=centroid` results in weights proportional to the distance of the centroids of neighboring regions. More specifically, the weight w_{us} of two neighboring regions u and s is set to $w_{us} = c \cdot \exp(-d(u,s))$, where d is the Euclidian distance between the centroids of the two sites and c is a normalizing constant. The constant c is chosen in such a way that the total sum of weights is equal to the total number of neighbors, which is in analogy to adjacency weights. The third choice `weightdef=combnd` results in weights proportional to the length of the common boundary. Similarly to `weightdef=centroid`, the weights are normalized, i.e. the total sum of weights is equal to the number of neighbors.

  Note that the specification of the `weightdef` option is only meaningful if a *boundary file* is read. If a *graph file* is read instead, the option has no effect because the boundary information of regions is missing and the computation of weights is therefore impossible.

5.2 Method outfile

Description

Method `outfile` performs the reverse of the `infile` command. Using method `outfile`, the map information currently in memory is written to an external file. The map information can be written either in *boundary file* or in *graph file* format.

Syntax

> *objectname*.`outfile` [*, options*] `using` *filename*

`outfile` writes the map information to the external file specified in *filename*. The file format can be either a *boundary file* or a *graph file*. If `graph` is specified as an additional option, the file format will be a *graph file*, otherwise a *boundary file*.

Options

- `graph`
  Forces the program to store the map information in *graph file* format rather than *boundary file* format.
- `includeweights`
  Option `includeweights` is meaningful only if the storing format is a *graph file*, i.e. option `graph` is additionally specified. In that case the weights associated with the edges (neighbors) of the nodes (regions) are additionally stored.
- `replace`
  The `replace` option allows *BayesX* to overwrite an already existing file. If `replace` is omitted in the optionlist and the file specified in *filename* already exists, an error will be raised. This prevents you from overwriting an existing file unintentionally.

5.3 Method reorder

Description

Method `reorder` reorders the regions of a map in the sense that the adjacency matrix of the reordered map has the smallest envelope when compared to all other possible orderings. A new map should always be reordered before using it with *bayesreg objects* because MCMC block moves for spatial covariates will be much faster if the envelope of the posterior precision matrix is small, see section 7.1. For reordering of the regions of the map the reverse Cuthill Mc-Kee algorithm is used, see George and Liu (1981) p. 58 ff.

Syntax

> *objectname*.`reorder`

`reorder` reorders the regions of a map in order to obtain smallest envelope of the corresponding adjacency matrix.

Options

Not allowed.

Reference

George, A., Liu, J. W. (1981). *Computer Solution of Large Sparse Positive Definite Systems.* Series in computational mathematics, Prentice–Hall.

Chapter 6

graph objects

Author: Andreas Brezger
email: andib@stat.uni-muenchen.de

Graph objects are used to visualize data and estimation results obtained by other objects in *BayesX*. Currently *graph objects* can be used to draw scatterplots between variables (section 6.2 method `plot`), or to draw and color geographical maps stored in *map objects* (section 6.1 method `drawmap`). The obtained plots are either printed on the screen or stored as a postscript file for further use in other documents (e.g. LaTeX documents). A *graph object* is created by typing

```
> graph objectname
```

in the *command window*.

6.1 Method drawmap

Description

Method `drawmap` is used to draw geographical maps and color the regions according to some numerical characteristics.

Syntax

`>` *objectname*`.drawmap` [*plotvar regionvar*] [`if` *expression*] , `map`=*mapname* [*options*] `using` *dataset*

Method `drawmap` draws the map stored in the *map object mapname* and prints the graph either on the screen or stores it as a postscript file (if option `outfile` is specified). The regions with regioncode *regionvar* are colored according to the values of the variable *plotvar*. The variables *plotvar* and *regionvar* are supposed to be stored in the *dataset object dataset*. An `if` statement may be specified to use only a part of the data in *dataset*. Several options are available, e.g. for changing from grey scale to color scale or storing the map as a postscript file. See the options list below for more details.

Options

The most important option, which must always be specified, is the `map` option. Here the name of the *map object* containing the boundary information of the map to be drawn is specified. The following options are available for method `drawmap` (in alphabetical order):

- `color`

  The `color` option allows to choose between a grey scale for the colors and a colored scale. If `color` is specified a colored scale is used instead of a grey scale.

- **drawnames**

  In some situations it may be useful to print the names of the regions into the graph (although the result may be confusing in most cases). This can be done by specifying the additional option `drawnames`. By default the names of the regions are omitted in the map.

- `fontsize =` *integer*

  Specifies the font size (in pixels) for labelling the legend and writing the names of the regions (if specified). Note, that the title is scaled accordingly. The default is `fontsize=12`.

- `nolegend`

  By default a legend is drawn into the graph. By specifying the option `nolegend` the legend will be omitted.

- `lowerlimit =` *realvalue*

  Lower limit of the range to be drawn. If `lowerlimit` is omitted, the minimum numerical value in *plotvar* will be used as the lower limit instead.

- `map =` *mapname*

  mapname specifies the name of the *map object* containing the boundary information of the map to be drawn. This option must always be specified.

- `outfile =` *filename*

  If option `outfile` is specified the graph will be stored as a postscript file rather than being printed on the screen. The path and the filename must be specified in *filename*. By default, an error will be raised if the specified file is already existing or the specified folder is not existing. To overwrite an already existing file, option `replace` must be additionally specified. This prevents you from unintentionally overwriting your files.

- `replace`

  The `replace` option is only useful in combination with option `outfile`. Specifying `replace` as an additional option allows the program to overwrite an already existing file (specified in `outfile`), otherwise an error will be raised.

- `nrcolors =` *integer*

  To color the regions according to their numerical characteristics, the data are divided into a (typically large) number of ordered categories. Afterwards a color is associated with each category. The `nrcolors` option can be used to specify the number of categories (and with it the number of different colors). The maximum number of colors is 256, which is also the default value.

- `swapcolors`

  In some situations it may be favorable to swap the order of the colors, i.e. black (red) shades corresponding to large values and white (green) shades corresponding to small values. This is achieved by specifying `swapcolors`. By default, small values are colored in black shades (red shades) and large values in white shades (green shades).

- `title =` *characterstring*

  Adds a title to the graph. If the title contains more than one word, *stringvalue* must be enclosed by quotation marks (e.g. `title="my first map"`).

- `upperlimit =` *realvalue*

  Upper limit of the range to be drawn. If `upperlimit` is omitted, the maximum numerical value in *plotvar* will be used as the upper limit instead.

- `pcat`

  If you want to visualize posterior probabilities it is convenient to specify `pcat`. This forces `drawmap` to expect a column that consists only of the values -1, 0 and 1. Of course you can achieve the same result by setting `nrcolors=3`, `lowerlimit=-1` and `upperlimit=1`.

Example

This example shows how to draw the map of Munich and how to color the subquarters in Munich according to some numerical characteristics. You find the boundary file of Munich (`munich.bnd`) as well as the data set `rent94means.raw` containing the distribution of the average rents across subquarters in the subfolder `examples` of your *BayesX* installation directory. In the following we assume that *BayesX* is installed in the folder `c:\bayesx`. We first create a *dataset object* `d` and a *map object* `m` and read in the rent data set and the map of Munich:

```
> dataset d
> d.infile using c:\bayesx\examples\rent94means.raw
```

```
> map m
> m.infile using c:\bayesx\examples\munich.bnd
```

We proceed by creating a *graph object* `g` and by drawing the map of Munich:

```
> graph g
> g.drawmap , map=m
```

Figure 6.1: Map of Munich

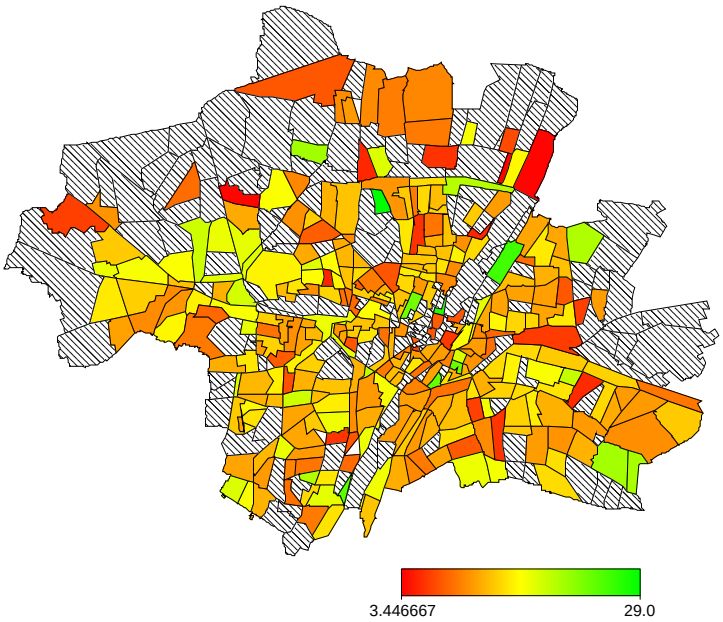


Figure 6.2: Distribution of the average rents per square meter in Munich.

The map of Munich appears on the screen in a separate window, see Figure 6.1. Before closing the window you are asked whether you want to save the map or not. If you agree the map will be stored as a postscript file in the folder you specify. Of course, the map can be directly stored in postscript format using the `outfile` option. In that case the map is not shown on the screen. Typing

```
> g.drawmap , map=m outfile= c:\temp\munich.ps
```

stores the map of Munich in the file `c:\temp\munich.ps` and the graph is not being printed on the screen.

Usually maps are drawn to visualize numerical characteristics of their regions. For instance, by typing

```
> g.drawmap R L , map=m color using d
```

the distribution of the average rents `R` across subquarters `L` are displayed, see Figure 6.2. The areas in the figure shaded with diagonal lines mark subquarters for which no data are available. The specification of the second variable `L` is required to match the names of the subquarters stored in the *map object* `m` with the data set `d`. Option `color` is specified to obtain a colored graph.

6.2 Method plot

Description

Method `plot` is used to draw scatterplots between two or more variables. Several options for labelling axes, connecting points, saving the graph etc. are available.

Syntax

> *objectname*.`plot` *xvar yvar1* [*yvar2 yvar3* ...] [`if` *expression*] [, *options*] `using` *dataset*

Method `plot` draws scatterplots of *yvar1*, *yvar2*, *yvar3* ... against *xvar* into a single graph using the data set specified in *dataset*. An `if` statement may be used to apply the method only to a part of the data. In addition, several options may be specified for labelling axes, connecting points, saving the graph in postscript format etc., see the options list below.

Options

The following options are available for method `plot` (listed in alphabetical order):

- `connect = 1|2|3|4|5`[*specifications for further variables*]

  Option `connect` specifies how points in the scatterplot are connected. There are currently 5 different specifications:

`1`	draw straight lines between the points (default)
`2`, `3`, `4`	draw dashed lines (numbers 2-4 indicate different variants)
`5`	do not connect, i.e. plot points only

  If you draw more than one scatterplot in the same graph (i.e. more than one *yvar* is specified) you can connect points for every *yvar* differently by simply specifying the corresponding number (`1,2,3,4,5`) for every *yvar*. Typing for example

  `connect=15`

  connects the points corresponding to *yvar1* and *xvar* by straight lines, but does not connect the points corresponding to *yvar2* (if specified) and *xvar*. Points corresponding to additionally specified variables *yvar*3, etc. are connected by straight lines.

  An equivalent way of specifying the different variants is available via the symbols 'l', 'd', '_', '-' and 'p', which correspond to the numbers 1-5, i.e.

  `connect=12345` is equivalent to `connect=ld_-p`

- `fontsize =` *integer*

  Specifies the font size (in pixels) for labelling axes etc. Note that the title is scaled accordingly. The default is `fontsize=12`.

- `height =` *integer*

  Specifies the height (in pixels) of the graph. The default is `height=210`.

- `linecolor = B|b|c|G|g|o|m|r|y` [*specifications for further variables*]

  Option `linecolor` specifies the color to be used for drawing lines (or points, see option `connect`) in the scatterplot. Currently the following specifications are available:

`B` black (default)
`b` blue
`c` cyan
`G` gray
`g` green
`o` orange
`m` magenta
`r` red
`y` yellow

If you draw more than one scatterplot in the same graph (i.e. more than one *yvar* is specified) you can use different colors for each *yvar* by simply specifying the corresponding symbol (`B,b,c,G,g,o,m,r,y`) for each *yvar*. Typing for example

`linecolor = Bgr`

colors the lines (points) corresponding to *yvar1* and *xvar* in black, whereas the points corresponding to *yvar2* and *yvar3* (if specified) and *xvar* are colored in green and red, respectively.

- `linewidth =` *integer*

  Specifies how thick lines should be drawn. The default is `linewidth=5`.

- `outfile =` *filename*

  If option `outfile` is specified the graph will be stored as a postscript file rather than being printed on the screen. The path and the filename must be specified in *filename*. By default, an error will be raised if the specified file is already existing or the specified folder is not existing. To overwrite an already existing file, option `replace` must be additionally specified. This prevents you from unintentionally overwriting your files.

- `pointsize =` *integer*

  Specifies the size of the points (in pixels) if drawing points rather than lines is specified. The default is `pointsize=20`.

- `replace`

  The `replace` option is useful only in combination with option `outfile`. Specifying `replace` as an additional option allows the program to overwrite an already existing file (specified in `outfile`), otherwise an error will be raised.

- `title =` *characterstring*

  Adds a title to the graph. If the title contains more than one word, *characterstring* must be enclosed by quotation marks (e.g. `title="my first title"`).

- `titlesize =` *realvalue*

  Specifies the factor by which the size of the title is scaled relative to the size of the labels of the axes (compare option `fontsize`). The default is `titlesize=1.5`.

- `width =` *integer*

  Specifies the width (in pixels) of the graph. The default is `width=356`.

- `xlab =` *characterstring*

  Labels the x-axis. If the label contains more than one word, *characterstring* must be enclosed by quotation marks (e.g. `xlab="x axis"`).

- `xlimbottom =` *realvalue*

  Specifies the minimum value at the x-axis to be drawn. The default is the minimum value in the data set. If `xlimbottom` is above the minimum value in the data set, only a part of the graph will be visible.

- `xlimtop =` *realvalue*

  Specifies the maximum value at the x-axis to be drawn. The default is the maximum value in the data set. If `xlimtop` is below the maximum value in the data set, only a part of the graph will be visible.

- `xstart =` *realvalue*

  Specifies the value where the first 'tick' on the x-axis should be drawn. The default is the minimum value on the x-axis.

- `xstep =` *realvalue*

  If `xstep` is specified, ticks are drawn at the x-axis with stepwidth *realvalue* starting at the minimum value on the x-axis (or at the value specified in option `xstart`). By default, five equally spaced ticks are drawn at the x-axis.

- `ylab =` *characterstring*

  Labels the y-axis. If the label contains more than one word, *characterstring* must be enclosed by quotation marks (e.g. `ylab="y axis"`).

- `ylimbottom =` *realvalue*

  Specifies the minimum value at the y-axis to be drawn. The default is the minimum value in the data set. If `ylimbottom` is above the minimum value in the data set, only a part of the graph will be visible.

- `ylimtop =` *realvalue*

  Specifies the maximum value at the y-axis to be drawn. The default is the maximum value in the data set. If `ylimtop` is below the maximum value in the data set, only a part of the graph will be visible.

- `ystart =` *realvalue*

  Specifies the value where the first 'tick' on the y-axis should be drawn. The default is the minimum value on the y-axis.

- `ystep =` *realvalue*

  If `ystep` is specified, ticks are drawn at the y-axis with stepwidth *realvalue* starting at the minimum value on the y-axis (or at the value specified in option `ystart`). By default, five equally spaced ticks are drawn at the y-axis.

- Further options

  In the following we describe options that may be useful if the variable at the x-axis represents dates. An example is a variable with values ranging from 1 to 19 representing the time period from January 1983 to July 1984. In this case, we naturally prefer that the x-axis is labelled in terms of dates rather than in the original coding (form 1 to 19). To achieve this we provide the options `month`, `year` and `xstep`. Options `year` and `month` are used to specify the year and the month (1 for January, 2 for February, ...) corresponding to the minimum covariate value. In the example mentioned above `year=1983` and `month=1` will produce the correct

result. In addition, option `xstep` may be specified to define the periodicity in which your data are collected. For example `xstep=12` (the default) corresponds to monthly data, while `xstep = 4`, `xstep = 2` and `xstep = 1` correspond to quarterly, half yearly and yearly data.

Example

We use the Munich rent data set `rent94.raw` to demonstrate the usage of method `plot`. You find the data set `rent94.raw` in the subfolder `examples` of your *BayesX* installation directory. In the following we assume that *BayesX* is installed in the folder `c:\bayesx`. We first read in the data by typing:

```
> dataset d
> d.infile using c:\bayesx\examples\rent94.raw
```

We now generate a *graph object* `g` and draw a scatterplot between floor space (variable `F`) and rent per square meter (variable `R`):

```
> graph g
> g.plot R F using d
```

The strange picture shown in Figure 6.3 appears on the screen. The problem is that the points are connected by straight lines and that the values of `F` are not sorted. Hence, to obtain an improved scatterplot we could either sort the data set with respect to `F` or simply avoid connecting the points. Typing

```
> d.sort F
> g.plot R F using d
```

yields the first option to improve the appearance of the scatterplot, typing

```
> g.plot R F , connect=p using d
```

yields the second option mentioned above. The corresponding graphs are shown in Figure 6.4 and Figure 6.5, respectively. To further improve the appearance of the scatterplot we add a title and label the x- and y-axes by typing

```
> g.plot R F , title="scatterplot between F and R" ylab="rent"
  xlab="floor space in square meters" connect=p using d
```

The result is shown in Figure 6.6. Finally, we add the outfile option to save the graph in postscript format:

```
> g.plot R F , title="scatterplot between F and R" ylab="rent"
  xlab="floor space in square meters" connect=p  outfile=c:\temp\plotrf.ps using d
```

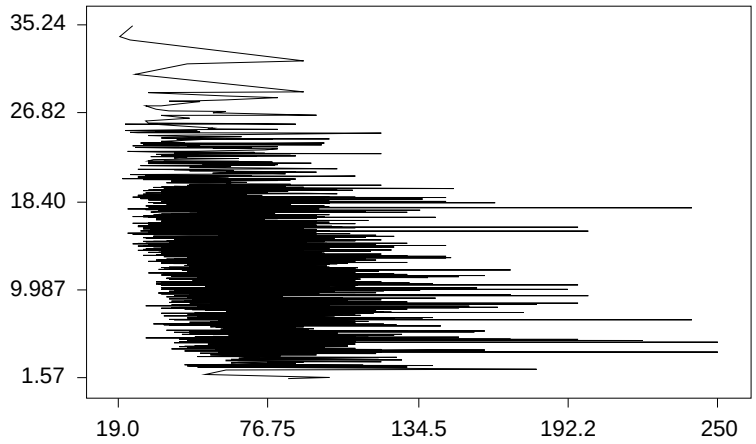


Figure 6.3: Scatterplot between floor space and rent per square meters (first try).

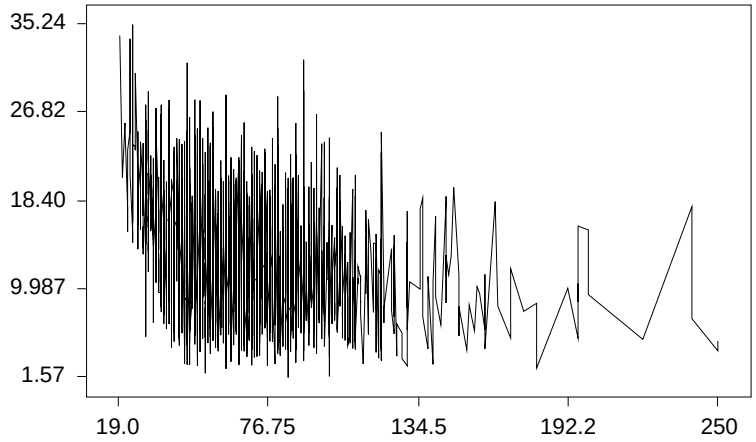


Figure 6.4: Scatterplot between floor space and rent per square meters (second try).

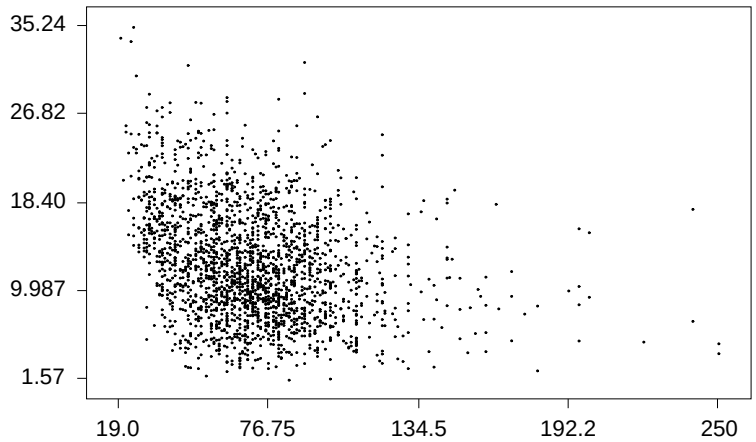


Figure 6.5: Scatterplot between floor space and rent per square meters (third try).

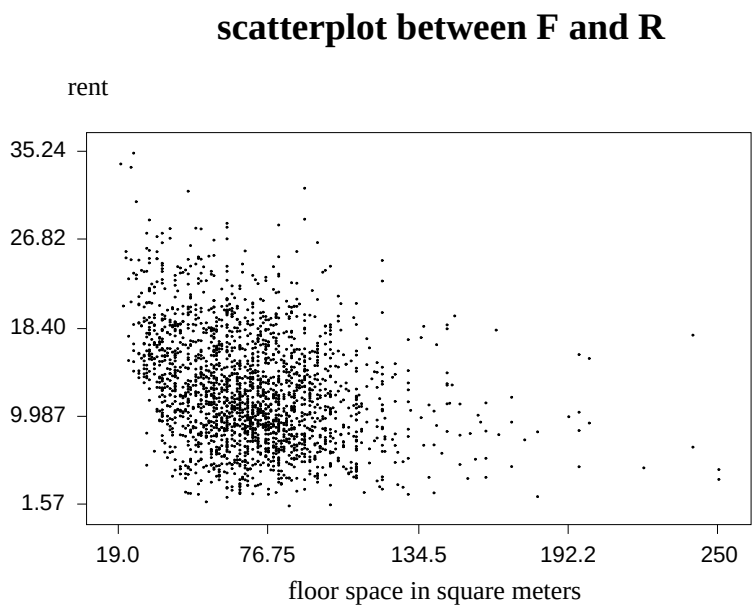


Figure 6.6: Scatterplot between floor space and rent per square meters (final try).

6.3 Method plotautocor

Description

Method `plotautocor` visualizes the autocorrelation functions obtained with method `autocor` of *bayesreg objects*, see also section 7.2.

Syntax

> *objectname*.`plotautocor` [,*options*] `using` *dataset*

Plots the autocorrelation functions stored in *dataset*. The data set must have the special structure described in section 7.2, i.e. method `plotautocor` is meaningful only if Bayesian regression models have been estimated in advance using *bayesreg objects* and autocorrelation functions of sampled parameters have been computed using method `autocor` of *bayesreg objects*.

Options

- `mean`

  If option `mean` is specified, for each lag number and model term only minimum, mean and maximum autocorrelations are plotted. This can lead to a considerable reduction in computing time and storing size.

- `outfile =` *filename*

  If option `outfile` is specified the graph will be stored as a postscript file and not printed on the screen. The path and the filename must be specified in *filename*. An error will be raised if the specified file is already existing and the `replace` option is not specified.

- `replace`

  The `replace` option is only useful in combination with option `outfile`. Specifying `replace` as an additional option allows the program to overwrite an already existing file (specified in `outfile`), otherwise an error will be raised.

6.4 Method plotsample

Description

Method `plotsample` visualizes the sampling paths of sampled parameters obtained with method `getsample` of *bayesreg objects*, see also section 7.3. The application of method `plotsample` is meaningful only if Bayesian regression models have been estimated in advance using *bayesreg objects* and sampled parameters have been computed and stored using method `getsample` of *bayesreg objects*.

Syntax

> *objectname*.`plotsample` [,*options*] `using` *dataset*

Plots sampled parameters stored in *dataset*. The data set must have the special structure described in section 7.3.

Options

- `outfile =` *filename*

  If option `outfile` is specified the graph will be stored as a postscript file and not printed on the screen. The path and the filename must be specified in *filename*. An error will be raised if the specified file is already existing and the `replace` option is not specified (see below).

- `replace`

  The `replace` option is only useful in combination with option `outfile`. Specifying `replace` as an additional option allows the program to overwrite an already existing file (specified in `outfile`), otherwise an error will be raised.

Chapter 7

bayesreg objects

Authors: Andreas Brezger, Stefan Lang and Thomas Kneib
email: andib@stat.uni-muenchen.de, lang@stat.uni-muenchen.de and kneib@stat.uni-muenchen.de

bayesreg objects are used to fit (multivariate) generalized linear models or hazard rate models with a *Structured Additive Predictor (STAR)*, see Fahrmeir, Kneib, and Lang (2003). Inference is fully Bayesian via Markov Chain Monte Carlo (MCMC) techniques. The methodological background is provided in considerable detail in the methodology manual. More details can be found in Fahrmeir and Lang (2001a), Fahrmeir and Lang (2001b), Lang and Brezger (2003), Brezger and Lang (2003), Fahrmeir and Osuna (2003), Hennerfeind, Brezger and Fahrmeir (2003), and Fahrmeir and Hennerfeind (2003). Good introductions into generalized linear models are the monographs of Fahrmeir and Tutz (2001) and Mc Cullagh and Nelder (1989). Introductions to semi- and nonparametric models are given in Green and Silverman (1994), Hastie and Tibshirani (1990), Hastie and Tibshirani (1993) and Hastie, Tibshirani and Friedman (2001). The paper of Chib and Greenberg (1995), the monograph *Markov Chain Monte Carlo in Practice* edited by Gilks, Richardson and Spiegelhalter (1996) and the article by Green (2001) give a good overview over MCMC simulation techniques.

First steps with *bayesreg objects* can be done with the tutorial like example in chapter 1 of the tutorials manual.

7.1 Method regress

7.1.1 Description

Method `regress` estimates regression or hazard rate models with Structured Additive Predictor. An introduction to the methodological background can be found in the methodology manual.

7.1.2 Syntax

>*objectname*.`regress` *model* [`weight` *weightvar*] [`if` *expression*] [, *options*] `using` *dataset*

Method `regress` estimates the regression model specified in *model* using the data specified in *dataset*. *dataset* must be the name of a *dataset object* created before. The details of correct models are covered in subsubsection 7.1.2.2. The distribution of the response variable can be either Gaussian, gamma, binomial, multinomial, Poisson, negative binomial zero inflated Poisson or zero inflated negative binomial, see also Table 7.5 for an overview about the distributions supported by *BayesX*. The response distribution is specified using option `family`, see subsubsection 7.1.2.4 below and the options list in subsection 7.1.3 for a detailed description. The default is `family=binomial` with a logit link. An `if` statement may be specified to analyze only a part of the data set, i.e. the observations where *expression* is true.

7.1.2.1 Optional weight variable

An optional weight variable *weightvar* may be specified to estimate weighted regression models. For Gaussian responses *BayesX* assumes that $y_r|\eta_r, \sigma^2 \sim N(\eta, \sigma^2/weightvar_r)$. Thus, for grouped Gaussian responses the weights must be the number of observations in the groups if the y_r's are the average of individual responses. If the y_r's are the sum of responses in every group, the weights must be the reciprocal of the number of observations in the groups. Of course, estimation of usual weighted regression models with heteroscedastic errors is also possible. In this case the weights should be proportional to the reciprocal of the heteroscedastic variances. If the response distribution is binomial, it is assumed that the values of the weight variable correspond to the number of replications and that the values of the response variable correspond to the number of successes. If `weight` is omitted, *BayesX* assumes that the number of replications is one, i.e. the values of the response must be either zero or one. For grouped Poisson data the weights must be the number of observations in a group and the y_i's are assumed to be the average of individual responses. In the case of gamma distributed responses *BayesX* assumes $y_r \sim G(\exp(\eta_r), \nu/weightvar_r)$ where $\mu_r = \exp(\eta_r)$ is the mean and $s_r = \nu/weightvar_r$ is the scale parameter.

If estimation is based on latent utility representations, the specification of weights is not allowed. Also for negative binomial, zero inflated Poisson and zero inflated negative binomial models is weighted regression not implemented.

7.1.2.2 Syntax of possible model terms

The general syntax of models is:

$depvar = term_1 + term_2 + \cdots + term_r$

depvar specifies the dependent variable in the model and $term_1$,...,$term_r$ define in which way the covariates influence the dependent variable. The different terms must be separated by '+' signs. A constant intercept is automatically included in the models and must not be specified by the user. This section reviews all possible model terms that are supported in the current version of *bayesreg objects* and provides some specific examples. Note that all described terms may be combined

in arbitrary order. An overview about the capabilities of *bayesreg objects* is given in Table 7.1. Table 7.2 shows how interactions between covariates are specified. Full details about all available options are given in subsubsection 7.1.2.3.

Throughout this section `Y` denotes the dependent variable.

Offset

Description: Adds an offset term to the predictor.

Predictor: $\eta = \cdots + offs + \cdots$

Syntax: `offs(offset)`

Example:

For example, the following model statement can be used to estimate a Poisson model with `offs` as offset term and `W1` and `W2` as fixed effects (if `family=poisson` is specified in addition):

```
Y = offs(offset) + W1 + W2
```

Fixed effects

Description: Incorporates covariate `W1` as a fixed effect into the model.

Predictor: $\eta = \cdots + \gamma_1 W1 + \cdots$

Syntax: `W1`

Example:

The following model statement causes *BayesX* to estimate a model with q fixed (linear) effects:

```
Y = W1 + W2 + ··· + Wq
```

Nonlinear effects of continuous covariates and time scales

First or second order random walk

Description: Defines a first or second order random walk prior for the effect of `X1`.

Predictor: $\eta = \cdots + f_1(X1) + \cdots$

Syntax:

`X1(rw1`[, *options*]`)`

`X1(rw2`[, *options*]`)`

Example:

Suppose we have a continuous covariate `X1`, whose effect is assumed to be nonlinear. The following model statement defines a second order random walk prior for f_1:

```
Y = X1(rw2,a=0.001,b=0.001)
```

Here, the expression `X1(rw2,a=0.001,b=0.001)` indicates, that the effect of `X1` should be incorporated nonparametrically into the model using a second order random walk prior. A first order random walk is specified in the model statement by modifying the first argument in `X1(rw2,a=0.001,b=0.001)` from `rw2` to `rw1` which yields the term

`X1(rw1,a=0.001,b=0.001)`. The second and third argument in the expression above are used to specify the hyperparameters of the inverse gamma prior for the variance. Besides the options `a` and `b` some more options are available, see subsubsection 7.1.2.3 for details.

P-spline with first or second order random walk penalty

Description: Defines a P-spline with a first or second order random walk penalty for the parameters of the spline.

Predictor: $\eta = \cdots + f_1(X1) + \cdots$

Syntax:

`X1(psplinerw1`[, *options*]`)`

`X1(psplinerw2`[, *options*]`)`

Example:

For example, a P-spline with second order random walk penalty is obtained using the following model statement:

```
Y = X1(psplinerw2)
```

By default, the degree of the spline is 3 and the number of inner knots is 20. The following model term defines a quadratic P-spline with 30 knots:

```
Y = X1(psplinerw2,degree=2,nrknots=30)
```

Full details about all possible options for P-splines are given in subsubsection 7.1.2.3.

Seasonal component for time scales

Description: Defines a seasonal effect of `time`.

Predictor: $\eta = \cdots + f_{season}(time) + \cdots$

Syntax:

`time(season`[, *options*]`)`

Example:

A seasonal component for a time scale `time` is specified for example by

```
Y = time(season,period=12).
```

Here, the second argument specifies the period of the seasonal effect. In the example above the period is 12, corresponding to monthly data.

Spatial Covariates

Markov random field

Description:

Defines a Markov random field prior for the spatial covariate `region`. *BayesX* allows an appropriate incorporation of spatial covariates with geographical information stored in the *map object* specified through the option `map`.

Predictor: $\eta = \cdots + f_{spat}(region) + \cdots$

Syntax:

`region(spatial,map=`*characterstring*`[,` *options*`])`

Example:

The specification of a Markov random field prior for spatial data has `map` as a required argument which must be the name of a *map object* (see chapter 5) that contains all necessary spatial information about the geographical map, i.e. the neighbors of each region and the weights associated with the neighbors. For example the statement

```
Y = region(spatial,map=germany)
```

defines a Markov random field prior for `region` where the geographical information is stored in the *map object* `germany`. An error will be raised if `germany` is not existing. It is advisable to reorder the regions of a map in advance to obtain a band matrix like precision matrix. This is achieved using method `reorder` of *map objects*, see section 5.3 for details.

2 dimensional P-spline with first order random walk penalty

Description:

Defines a 2 dimensional P-spline for the spatial covariate `region` based on the tensor product of 1 dimensional P-splines with a 2 dimensional first order random walk penalty for the parameters of the spline. Estimation is based on the coordinates of the centroids of the regions an observation pertains to. The centroids are computed using the geographical information stored in the *map object* specified through the option `map`.

Predictor: $\eta = \cdots + f(centroids) + \cdots$

Syntax:

`region(geospline,map=`*characterstring*`[,` *options*`])`

Example:

The specification of a 2 dimensional P-spline (*geospline*) for spatial data has `map` as a required argument which must be the name of a *map object* (see chapter 5) that contains all necessary spatial information about the geographical map, i.e. the neighbors of each region and the weights associated with the neighbors. The model term

```
Y = region(geospline,map=germany)
```

specifies a tensor product cubic P-spline with first order random walk penalty where the geographical information is stored in the *map object* `germany`.

Unordered group indicators

Unit- or cluster specific unstructured effect

Description: Defines an unstructured (uncorrelated) random effect with respect to grouping variable `grvar`.

Predictor: $\eta = \cdots + f(grvar) + \cdots$

Syntax:

`grvar(random[,` *options*`])`

Example:

BayesX also supports Gaussian i.i.d. random effects to cope with unobserved heterogeneity among units or clusters of observations. Suppose the analyzed data set contains a group indicator `grvar` that gives information about the individual or cluster a particular observation belongs to. Then an individual specific uncorrelated random effect is incorporated through the term

```
Y = grvar(random)
```

The inclusion of more than one random effects term in the model is possible allowing the estimation of multilevel models. However, we have only limited experience with multilevel models so that it is not clear how well these models can be estimated in *BayesX*.

Nonlinear baseline effect in Cox models

P-spline with second order random walk penalty

Description: Defines a P-spline with a second order random walk penalty for the parameters of the spline for the log-baseline effect $\log(\lambda_0(\texttt{time}))$.

Predictor: $\eta = \log(\lambda_0(time)) + \cdots$

Syntax:

`time(baseline`[, *options*]`)`

Example:

Suppose continuous-time survival data (`time`, `delta`) together with additional covariates (`W1`,`X1`) are given where `time` denotes the vector of observed duration times, `delta` is the vector of corresponding indicators of non-censoring, `W1` is a discrete covariate and `X1` a continuous one. The following Cox model with hazard rate λ and log-baseline effect $\log(\lambda_0(\texttt{time}))$

$$\lambda(time) = \lambda_0(time)\exp(\gamma_0 + \gamma_1 W1 + f(X1)) = \exp\left(\log(\lambda_0(time)) + \gamma_0 + \gamma_1 W1 + f(X1)\right)$$

is estimated by the model statement

```
delta = time(baseline) + W1 + X1(psplinerw2)
```

Note that a baseline term has to be specified in the model.

Varying coefficients with continuous covariates as effect modifier

First or second order random walk

Description:

Defines a varying coefficient term, where the effect of `X1` varies smoothly over the range of `X2`. Covariate `X2` is the effect modifier. The smoothness prior for f is a first or second order random walk.

Predictor: $\eta = \cdots + f(X2)X1 + \cdots$

Syntax:

`X1*X2(rw1`[, *options*]`)`

`X1*X2(rw2`[, *options*]`)`

Example:

For example, a varying coefficient term with a second order random walk smoothness prior is defined as follows:

```
Y = X1*X2(rw2)
```

P-spline with first or second order random walk penalty

Description:

Defines a varying coefficient term, where the effect of `X1` varies smoothly over the range of `X2`. Covariate `X2` is the effect modifier. The smoothness prior for f is a P-spline with first or second order random walk penalty.

Predictor: $\eta = \cdots + f(X2)X1 + \cdots$

Syntax:

`X1*X2(psplinerw1`[, *options*]`)`

`X1*X2(psplinerw2`[, *options*]`)`

Example:

For example, a varying coefficient term with a second order random walk smoothness prior is defined as follows:

```
Y = X1*X2(psplinerw2)
```

Seasonal prior

Description:

Defines a varying coefficients term where the effect of `X1` varies over the range of the effect modifier `time`. For `time` a seasonal prior is used.

Predictor: $\eta = \cdots + f_{season}(time)X1 + \cdots$

Syntax:

`X1*time(season`[, *options*]`)`

Example:

The inclusion of a varying coefficients term with a seasonal prior may be meaningful if we expect a different seasonal effect with respect to grouping variable `X1`. In this case we can include additional seasonal effects for each category of `X1` by

```
Y = X1*time(season)
```

Time-varying effects in Cox models

P-spline with second order random walk penalty

Description: Defines a varying coefficients term where the effect of `X1` varies over the range of the effect modifier `time`, i.e. variable `X1` has time-varying effect. The smoothness prior for f(`time`) is a P-spline with second order random walk penalty.

Predictor: $\eta = \log(\lambda_0(time)) + f(time)X1 \cdots$

Syntax:

`X1*time(baseline`[, *options*]`)`

Example:

Suppose continuous-time survival data (`time`, `delta`) together with an additional covariate `X1` are given, where `time` denotes the vector of observed duration times, `delta` is the vector of corresponding indicators of non-censoring. The following Cox model with hazard rate λ

$$\begin{aligned}\lambda(time) &= \lambda_0(time)\exp(\gamma_0 + f(time)X1) \\ &= \exp\left(\log(\lambda_0(time)) + \gamma_0 + f(time)X1\right)\end{aligned}$$

is estimated by the model statement

```
delta = time(baseline) + X1*time(baseline)
```

Varying coefficients with spatial covariates as effect modifiers

Markov random field

Description:

Defines a varying coefficient term where the effect of `X1` varies smoothly over the range of the spatial covariate `region`. A Markov random field is estimated for f_{spat}. The geographical information is stored in the *map object* specified through the option `map`.

Predictor: $\eta = \cdots + f_{spat}(region)X1 + \cdots$

Syntax:

`X1*region(spatial,map=`*characterstring*`[,`*options*`])`

Example:

For example the statement

```
Y = X1*region(spatial,map=germany)
```

defines a varying coefficient term with the spatial covariate `region` as the effect modifier and a Markov random field as spatial smoothness prior. Weighted Markov random fields can be estimated by including an appropriate weight definition when creating the *map object* `germany` (see section 5.1).

Varying coefficients with unordered group indicators as effect modifiers (random slopes)

Unit- or cluster specific unstructured effect

Description:

Defines a varying coefficient term where the effect of `X1` varies over the range of the group indicator `grvar`. Models of this type are usually referred to as random slope models. A Gaussian i.i.d. random effect with respect to grouping variable `grvar` is assumed for f. A main effect $\gamma X1$ is additionally estimated using a diffuse prior for γ. This means that the random slope effect $f(grvar)X1$ can be seen as the deviation from the main effect. Estimation is carried out using hierarchical centering, see Gelfand, Sahu and Carlin (1995). Note that nonsensical results are obtained if an additional fixed effect of `X1` is added in the model statement because the fixed effect is automatically estimated.

Predictor: $\eta = \cdots + \gamma X1 + f(grvar)X1 + \cdots$

Syntax:

`X1*grvar(random`[, *options*]`)`

Example:

For example, a random intercept term with incorporation of `X1` as fixed effect is specified as follows:

```
Y = X1*grvar(random)
```

If the linear effect of `X1` should be omitted, the option `nofixed` must be specified:

```
Y = X1*grvar(random,nofixed)
```

Surface estimators

2 dimensional P-spline with first order random walk penalty

Description:

Defines a 2 dimensional P-spline based on the tensor product of 1 dimensional P-splines with a 2 dimensional first order random walk penalty for the parameters of the spline.

Predictor: $\eta = \cdots + f(X1, X2) + \cdots$

Syntax:

`X1*X2(pspline2dimrw1`[, *options*]`)`

Example:

The model term

```
Y = X1*X2(pspline2dimrw1)
```

specifies a tensor product cubic P-spline with first order random walk penalty.

In many applications it is favorable to additionally incorporate the 1 dimensional main effects of `X1` and `X2` into the models. In this case the 2 dimensional surface can be seen as the deviation from the main effects. Note, that the number of inner knots has to be the same for the main effects and the interaction effect. For example, splines with 10 inner knots are estimated by

```
Y = X1(psplinerw2,nrknots=10) + X2(psplinerw2,nrknots=10)
    + X1*X2(pspline2dimrw1,nrknots=10)
```

7.1.2.3 Description of additional options for terms of bayesreg objects

All arguments described in this section are optional and may be omitted. Generally, options are specified by adding the option name to the specification of the model term type in the parentheses, separated by comma. Boolean options are specified by simply adding the option name to the options of a certain term. For example, a random intercept term with `a=b=0.001` as parameters for the inverse gamma distribution of the variance parameter, with updating according to IWLS and without incorporation of `X1` as fixed effect is specified as follows:

```
X1*grvar(random,a=0.001,b=0.001,proposal=iwls,nofixed)
```

Note that all options may be specified in arbitrary order. Table 7.3 provides explanations and the default values of all possible options. In Table 7.4 all reasonable combinations of model terms and options can be found.

Type	Syntax example	Description
offset	`offs(offset)`	Variable `offs` is an offset term.
linear effect	`W1`	Linear effect for `W1`.
first or second order random walk	`X1(rw1)` `X1(rw2)`	Nonlinear effect of `X1`.
P-spline	`X1(psplinerw1)` `X1(psplinerw2)`	Nonlinear effect of `X1`.
seasonal prior	`time(season,period=12)`	Varying seasonal effect of `time` with period 12.
Markov random field	`region(spatial,map=m)`	Spatial effect of `region` where `region` indicates the region an observation pertains to. The boundary information and the neighborhood structure is stored in the *map object* `m`.
Two dimensional P-spline	`region(geospline,map=m)`	Spatial effect of `region`. Estimates a two dimensional P-spline based on the centroids of the regions. The centroids are stored in the *map object* `m`.
random intercept	`grvar(random)`	I.i.d. (random) Gaussian effect of the group indicator `grvar`, e.g. `grvar` may be an individuum indicator when analyzing longitudinal data.
baseline in Cox models	`time(baseline)`	Nonlinear shape of the baseline effect $\lambda_0(time)$ of a Cox model. $\log(\lambda_0(time))$ is modelled by a P-spline with second order penalty.

Table 7.1: Overview over different model terms for bayesreg objects.

Type of interaction	Syntax example	Description

Varying coefficient term	`X1*X2(rw1)`
	`X1*X2(rw2)`
	`X1*X2(psplinerw1)`
	`X1*X2(psplinerw2)`
	`X1*time(season)`
		Effect of `X1` varies smoothly over the range of the continuous covariate `X2` or `time`, respectively.

random slope	`X1*grvar(random)`	The regression coefficient of `X1` varies with respect to the unit- or cluster index variable `grvar`.
Geographically weighted regression	`X1*region(spatial,map=m)`	Effect of `X1` varies geographically. Covariate `region` indicates the region an observation pertains to.
Two dimensional surface	`X1*X2(pspline2dimrw1)`	Two dimensional surface for the continuous covariates `X1` and `X2`.
Time-varying effect in Cox Models	`X1*time(baseline)`	Nonlinear, time-varying effect of `X1`.

Table 7.2: Possible interaction terms for bayesreg objects.

optionname	description	default
a,b	The options a and b specify the hyperparameters of the inverse Gamma prior for the variance τ^2.	a=0.001, b=0.001
min,max	The options min and max define the minimum and maximum block sizes between which *BayesX* randomly chooses the block size for block move updates in every iteration. If min and max are omitted the minimum and maximum block sizes are automatically determined during the burnin period such that the average acceptance rate lies between 30% and 70%. The specification of minimum and maximum block sizes is only meaningful if conditional prior proposals are applied and has no effect for Gaussian responses and (multi)categorical probit models or if proposal=iwls or proposal=iwlsmode is specified.	automatic determination
lambda	Provides a starting value for the variance parameter λ.	lambda=0.1
proposal	Specifies the type of proposal density. proposal=cp means conditional prior proposal, proposal=iwls stands for iteratively weighted least squares (IWLS) proposal and proposal=iwlsmode indicates IWLS based on posterior mode estimation.	proposal=iwls
updateW	The option updateW may be used to specify how often the IWLS weight matrix should be updated. updateW=0 means never, updateW=1 means in every iteration (which is the default), updateW=2 means in every second iteration and so on.	updateW=1
degree	Specifies the degree of the B-spline basis functions.	degree=3
nrknots	Specifies the number of inner knots for a P-spline term.	nrknots=20
gridsize	The option gridsize can be used to restrict the number of points (at the x-axis) for which estimates are computed. By default, estimates are computed at every distinct covariate value in the data set (indicated by gridsize=-1). This may be relatively time consuming in situations where the number of distinct covariate values is large. If gridsize=nrpoints is specified, estimates are computed on an equidistant grid with nrpoints knots.	gridsize=-1
derivative	The option derivative causes that first order derivatives of the estimation are computed.	-
period	The period of the seasonal effect can be specified with the option period. The default is period=12 which corresponds to monthly data.	period=12
nofixed	The option nofixed suppresses the estimation of the main effect $\gamma X1$ for random slopes.	-

Table 7.3: Optional arguments for bayesreg object terms

	rw1/rw2	season	psplinerw1/psplinerw2	spatial	random	geospline	pspline2dimrw1	baseline
`a`	realvalue	realvalue	realvalue	realvalue	realvalue	realvalue	realvalue	realvalue
`b`	realvalue	realvalue	realvalue	realvalue	realvalue	realvalue	realvalue	realvalue
`min`	∗	∗	∗	×	×	∗	∗	integer
`max`	∗	∗	∗	×	×	∗	∗	integer
`lambda`	realvalue	realvalue	realvalue	realvalue	realvalue	realvalue	realvalue	realvalue
`proposal`	•	•	•	•	○	•	•	×
`updateW`	integer	integer	integer	integer	×	integer	integer	×
`degree`	×	×	integer	×	×	integer	integer	integer
`nrknots`	×	×	integer	×	×	integer	integer	integer
`gridsize`	×	×	integer	×	×	×	integer	integer
`derivative`	×	×	△	×	×	×	×	×
`period`	×	integer	×	×	×	×	×	×
`nofixed`	×	×	×	×	△	×	×	×
`map`	×	×	×	*map object*	×	*map object*	×	×

×	not available
∗	available only if `proposal = cp`
○	admissible values are `iwls,iwlsmode`
•	admissible values are `cp,iwls,iwlsmode`
△	available as boolean option (specified without supplying a value)

Table 7.4: Terms and options for bayesreg objects

7.1.2.4 Specifying the response distribution

The current version of *BayesX* supports the most common distributions of the response. Supported univariate distributions are Gaussian, binomial (with logit or probit link), Poisson, negative binomial, gamma, zero inflated Poisson and zero inflated negative binomial. Supported multivariate models are multinomial logit or probit models for categorical responses with unordered categories, and the cumulative threshold model with probit link for categorical responses with ordered categories. Recently models for continuous time survival analysis have been added, see subsubsection 7.1.2.5. An overview over the supported models is given in Table 7.5. In *BayesX* the distribution of the response is specified by adding the additional option `family` to the options list. For instance, `family=gaussian` defines the responses to be Gaussian. However, in some cases one or more additional options associated with the specified response distribution may be specified. An example is the `reference` option for multinomial responses, which defines the reference category. In the following we give detailed instructions on how to specify the various models:

Gaussian responses

For Gaussian responses *BayesX* assumes $y_i|\eta_i, \sigma^2 \sim N(\eta_i, \sigma^2/weightvar_i)$ or equivalently in matrix notation $y|\eta, \sigma^2 \sim N(\eta, \sigma^2 C^{-1})$. Here $C = diag(weightvar_1, \ldots, weightvar_n)$ is a known weight matrix. Gaussian response is specified by adding

```
family=gaussian
```

to the options list.

An optional weight variable *weightvar* may be specified to estimate weighted regression models, see subsubsection 7.1.2.1 for details on how to specify weights. For grouped Gaussian responses the weights must be the number of observations in the groups if the y_i's are the average of individual responses. If the y_i's are the sum of responses in every group, the weights must be the reciprocal of the number of observations in the groups. Of course, estimation of usual weighted regression models if the errors are heteroscedastic is also possible. In this case the weights should be proportional to the reciprocal of the heteroscedastic variances. If a weight variable is not specified, *BayesX* assumes $weightvar_i = 1$, $i = 1, \ldots, n$.

For Gaussian responses, the additional parameter σ^2 for the overall variance of the responses must be estimated. Here, an inverse gamma prior with hyperparameters `a` and `b` is defined for σ^2. The default for the hyperparameters is `a=1` and `b=0.005`. The default values may be changed using the `aresp` and `bresp` option. For instance, by adding

```
aresp=0.01  bresp=0.01
```

to the options list, the values of `a` and `b` are both set to 0.01.

Gamma distributed responses

In the literature, the density function of the gamma distribution is parameterized in various ways. In the context of regression analysis the density is usually parameterized in terms of the mean μ and the scale parameter `s`. Then, the density of a gamma distributed random variable y is given by

$$p(y) \propto y^{s-1} \exp(-\frac{s}{\mu} y) \tag{7.1}$$

for $y > 0$. For the mean and the variance we obtain $E(y) = \mu$ and $Var(y) = \mu^2/s$. We write $y \sim G(\mu, s)$.

A second parameterization is based on hyperparameters `a` and `b` and is usally used in the context of

Bayesian hierarchical models to specify hyperpriors for variance components. The density is then given by

$$p(y) \propto y^{a-1} \exp(-by) \tag{7.2}$$

for $y > 0$. In this parameterization we obtain $E(y) = a/b$ and $Var(y) = a/b^2$ for the mean and the variance, respectively. We write $y \sim G(a, b)$

In *BayesX* a gamma distributed response is defined as in the first parameterization (7.1). For the rth observation *BayesX* assumes $y_r|\eta_r, \nu \sim G(\exp(\eta_r), \nu/weightvar_r)$ where $\mu_r = \exp(\eta_r)$ is the mean and $s = \nu/weightvar_r$ is the scale parameter. A gamma distributed response is specified by adding

```
family=gamma
```

to the options list. An optional weight variable *weightvar* may be specified to estimate weighted regression models, see subsubsection 7.1.2.1 for details on how to specify weights.

In analogy to the variance parameter in Gaussian response models, we assume a Gamma prior (second parameterization (7.2)) with hyperparameters a_ν and b_ν for the scale parameter ν, i.e. $\nu \sim Gamma(a_\nu, b_\nu)$. The default for the hyperparameters is $a_\nu = 1$ and $b_\nu = 0.005$. The default values may be changed using the `aresp` and `bresp` option. For instance, by adding

```
aresp=0.01  bresp=0.01
```

to the options list, the values of a_ν and b_ν are both set to 0.01.

Updating of the scale parameter ν is done by MH-steps based on a gamma proposal distribution with mean $E(\nu^{prop}) = \nu^c$ equal to the current state of the chain ν^c and a fixed variance $Var(\nu^{prop})$. The variance $Var(\nu^{prop})$ may be used as a tuning parameter. It is specified by using the additional option `gammavar` to the options list. For example, by adding

```
gammavar=0.0001
```

$Var(\nu^{prop}) = 0.0001$ is used in the proposal distribution. The default is `gammavar=0.001`.

It is also possible to assume a fixed nonstochastic scale parameter. The scale parameter is defined to be fixed rather than stochastic by adding

```
scalegamma = fixed
```

to the options list. The (fixed) value of the scale parameter is specified by adding:

```
scale = realvalue
```

Typing e.g.

```
scale = 1
```

defines the scale parameter $\nu = 1$.

Binomial logit and probit models

A binomial logit model is specified by adding the option

```
family=binomial
```

to the options list, and a probit model by adding

```
family=binomialprobit
```

to the list.

For logit models a weight variable may be additionally specified, see subsubsection 7.1.2.1 for details on how to specify weights. *BayesX* assumes that the weight variable corresponds to the number of replications and the response variable to the number of successes. If a weight variable is omitted, *BayesX* assumes that the number of replications is one, i.e. the values of the response must be either zero or one. For probit models the specification of a weight variable is not allowed.

Multinomial logit and probit models

A multinomial logit model is specified by adding the option

`family=multinomial`

to the options list, a multinomial probit model by adding

`family=multinomialprobit`

to the options list.

Similar to binomial logit and probit models different updating schemes are used for estimation, see the section above about binomial logit and probit models for details.

Usually a second option must be added to the options list to define the reference category. This is achieved by specifying the `reference` option. Suppose that the response variable has three categories 1,2 and 3. To define, for instance, the reference category to be 2, simply add

`reference=2`

to the options list. If this option is omitted, the *smallest* number will be used as the reference category.

Cumulative threshold models

So far, *BayesX* supports only cumulative probit models. A cumulative probit model is specified by adding

`family=cumprobit`

to the options list. The reference category will always be the largest value of the response.

An important problem with Bayesian cumulative threshold models is the mixing and convergence of MCMC samples of the threshold parameters. Usually the mixing is relatively poor implying quite large MCMC samples in order to obtain reliable estimation results. An exception are cumulative models with three categories of the response. In this case *BayesX* uses a reparameterized model for which the mixing of the threshold parameters is quite satisfactory. A description of this reparameterization can be found in Fahrmeir and Lang (2001b) or in Chen and Dey (2000). However, parameter estimates are given in the original parameterization as has been described in subsection 7.1.1 of this manual. To estimate three categorical response models without reparameterization the additional option `notransform` must be added to the options list (not recommended).

Poisson regression

A Poisson regression is specified by adding

`family=poisson`

to the options list.

A weight variable may be additionally specified, see subsubsection 7.1.2.1 for details on how to specify weights. For grouped Poisson data the weights must be the number of observations in a group and the responses are assumed to be the average of individual responses.

Negative binomial regression

A negative binomial regression is specified by adding

`family=nbinomial`

to the options list.

A weight variable can not be additionally specified.

For negative binomial responses *BayesX* assumes $y_i|\eta_i, \delta \sim NB(\eta_i, \delta)$ for a pure negative binomial formulation or equivalently $y_i|\eta_i \sim Po(\nu_i\eta_i)$ with $\nu_i|\delta \sim G(\delta, \delta)$ for a Poisson-Gamma formulation. Both alternatives are specified by setting the option `distopt=nb` (default) or `distopt=poga` respectively. The first formulation works with a negative binomial likelihood and provides estimates for the parameters in the predictor and for δ. The second formulation works with a Poisson likelihood but has an extra vector of multiplicative random effects with prior $\nu_i|\delta \sim G(\delta, \delta)$. It provides estimates for the parameters in the predictor, for δ and for the ν_i.

The prior for the scale parameter δ is $G(a, b)$ in both formulations, where `a=1` is fixed and `b` is estimated. Its prior is again a gamma distribution with fixed parameters 1 and 0.005.

Zero inflated count data models

A zero inflated regression for count data is specified by adding

```
family = zip
```

to the options list.

A weight variable can not be additionally specified.

Zero inflated count data distributions are used when the number of zero counts in the data exceeds the number of zero counts expected by the distribution. They are based on two processes. The first process is an underlying count data process, that can not be observed directly, but after a transformation through a so called selection process. This one is defined through a 0/1 variable. If we have a 0 in the selection process, the observed count will be zero, independently of the generated value by the underlying count data process. Otherwise, if we have a 1, then we will observe directly the generated value by the underlying count data process. The implemented zero inflated distributions in *BayesX* do not work with both processes directly. They are marginalized versions of a given count data distribution with respect to a 0/1 $simBern(1-\theta)$ selection process. For the count data process we have two possibilities, that can be controlled throuth the option `zipdistopt`. If we chose a Poisson distribution, we get a zero inflated Poisson distribution (`zipdistopt = zip`) denoted by $y_i|\eta_i, \theta \sim ZIP(\eta_i, \theta)$. We may combine both approaches zero inflation and overdispersion. To do this, we may chose a negative binomial distribution leading to a zero inflated negative binomial (`zipdistopt = zinb`) and denoted by $y_i|\eta_i, \delta, \theta \sim ZINB(\eta_i, \delta, \theta)$.

7.1.2.5 Continuous time survival analysis

BayesX offers two alternatives of estimating continuous time Cox models with semiparametric predictor η, which are described in subsection 5.2 of the methodology manual. The first alternative is to assume that all time-dependent values are piecewise constant, which leads to the so called *piecewise exponential model* (p.e.m.), and the second one is to estimate the log-baseline effect $\log(\lambda_0(t)) = f_0(t)$ by a P-spline with second order random walk penalty.

Piecewise exponential model (p.e.m.)

In subsection 5.2 of the methodology manual we demonstrated how continuous time survival data has to be manipulated such that a Poisson model may be used for estimation. Suppose now we have the modified data set

y	indnr	a	δ	Δ	x1	x2
0	1	0.1	1	log(0.1)	0	3
0	1	0.2	1	log(0.1)	0	3
1	1	0.3	1	log(0.05)	0	3
0	2	0.1	0	log(0.1)	1	5
0	2	0.2	0	log(0.02)	1	5
⋮	⋮	⋮	⋮	⋮	⋮	⋮

with indicator `y`, interval limit `a`, indicator of non-censoring δ and offset Δ defined as in subsection 5.2 of the methodology manual. Let `x1` be a covariate with linear effect and `x2` a continuous one with a nonlinear effect. Then the correct syntax for estimating a p.e.m. with a *bayesreg object* named `b` is e.g. as follows:

```
> b.regress y = a(rw1) + Delta(offset) + x1 + x2(psplinerw2), family=poisson ...
```

or

```
> b.regress y = a(rw2) + Delta(offset) + x1 + x2(psplinerw2), family=poisson ...
```

Note that a time-varying effect of a covariate `X` may be estimated in the p.e.m. by simply adding the term

```
X*a(rw1) or X*a(rw2)
```

to the model statement.

Specifying a P-spline prior for the log-baseline

For the estimation of a Cox model with a P-spline prior with second order random walk penalty

```
family=cox
```

has to be specified in the options list. The number of knots and degree of the P-spline prior for $f_0(t)$ may be specified in the baseline term. Note that it is compelling that there is a baseline term specified for the vector of observed duration times. The indicator of non-censoring δ_i has to be specified as the dependent variable in the model statement. Data augmentation and the specification of an offset term are not required here. To handle left truncation and time-varying covariates a variable `beginvar`, that records when the observation became at risk, may be specified by adding `begin=beginvar` to the options list.
In the example above with survival data

t	δ	x1	x2
0.25	1	0	3
0.12	0	1	5
⋮	⋮	⋮	⋮

a Cox model with a quadratic P-spline prior with 15 knots for the log-baseline would be estimated as follows:

```
> b.regress delta = t(baseline,degree=2,nrknots=15)+ x1 + x2(psplinerw2),
  family=cox
```

Note, that we assume that a *bayesreg object* `b` has been created before executing the command.

Further note that a time-varying effect of a covariate `X` may be estimated by adding the term

```
X*time(baseline)
```

to the model statement.

7.1.3 Options

Options for controlling MCMC simulations

Options for controlling MCMC simulations are listed in alphabetical order.

- `burnin =` *integer*
  Changes the number of burn-in iterations to *integer*, where *integer* must be a positive integer number or zero (i.e. no burn-in period). The number of burn-in iterations must be smaller than the number of iterations (see option `iterations`).
  DEFAULT: `burnin=2000`

- `iterations =` *integer*
  Changes the number of MCMC iterations to *integer*, where *integer* must be a positive integer number. The number of iterations must be larger than the number of burnin iterations.
  DEFAULT: `iterations=52000`

- `maxint =` *integer*
  If first or second order random walk priors are specified, in some cases the data will be slightly grouped: The range between the minimal and maximal observed covariate values will be divided into (small) intervals, and for each interval one parameter will be estimated. The grouping has almost no effect on estimation results as long as the number of intervals is large enough. With the `maxint` option the amount of grouping can be determined by the user. *integer* is the maximum number of intervals allowed. For equidistant data, `maxint = 150` for example, means that no grouping will be done as long as the number of *different* observations is equal to or below 150. For non equidistant data some grouping may be done even if the number of different observations is below 150.
  DEFAULT: `maxint=150`

- `step =` *integer*
  Defines the thinning parameter for MCMC simulation. For example, `step = 50` means, that only every 50th sampled parameter will be stored and used to compute characteristics of the posterior distribution as means, standard deviations or quantiles. The aim of thinning is to reach a considerable reduction of disk storing and autocorrelations between sampled parameters.
  DEFAULT: `step=50`

Options for specifying the response distribution

Options for specifying the response distribution are listed in alphabetical order below.

- `aresp =` *realvalue*
  Defines the value of the hyperparameter `a` for the inverse gamma prior of the overall variance parameter σ^2, if the response distribution is Gaussian. *realvalue* must be a positive real valued number.
  DEFAULT: `aresp=1`

- `bresp =` *realvalue*
  Defines the value of the hyperparameter `b` for the inverse gamma prior of the overall variance

parameter σ^2, if the response distribution is Gaussian. *realvalue* must be a positive real valued number.
DEFAULT: `bresp=0.005`

- `distopt =` *characterstring*
  Defines the implemented formulation for the negative binomial model if the response distribution is negative binomial. The two possibilities are to work with a negative binomial likelihood (`distopt=nb`) or to work with the Poisson likelihood and the multiplicative random effects (`distopt=poga`)
  DEFAULT: `distopt=nb`

- `family =` *characterstring*
  Defines the distribution of the response variable in the model. Models supported are Gaussian regression models with the identity link, binomial logit or probit models, multinomial logit or probit models for unordered categories of the response, cumulative threshold models with probit link for ordered categories of the response, and Poisson, negative binomial or their zero inflated versions with the log-link. For some distributions (e.g. multinomial) additional options may be specified to control MCMC inference. A detailed description on how to specify the distribution of the response is given in subsubsection 7.1.2.4. Table 7.5 lists all possible specifications for the distribution of the response currently supported by *BayesX*. In addition, a list of options associated with the particular response distribution is given.
  DEFAULT: `family=binomial`

- `reference =` *realvalue*
  Option `reference` is meaningful only if `family=multinomial` is specified as the response distribution. In this case `reference` defines the reference category to be chosen. Suppose, for instance, that the response is three categorical with categories 1,2, and 3. Then `reference=2` defines the value 2 to be the reference category.

- `zipdistopt =` *characterstring*
  Defines the zero inflated distribution for the regression analysis. The two possibilities are to work with a zero inflated Poisson distribution (`zipdistopt=zip`) or to work with the zero inflated negative binomial likelihood (`zipdistopt=zinb`).

value of `family`	response distribution	link	additional options
`family=gaussian`	Gaussian	identity	`aresp`, `bresp`
`family=binomialprobit` `family=binomial`	binomial binomial	probit logit	
`family=multinomialprobit` `family=multinomial`	unordered multinomial unordered multinomial	probit logit	`reference` `reference`
`family=cumprobit`	cumulative threshold	probit	
`family=poisson`	Poisson	log-link	
`family=nbinomial`	Negative Binomial	log–link	`distopt`
`family=zip`	Zero inflation	log-link	`zipdistopt`
`family=cox`	continuous-time survival data		`begin`

Table 7.5: Summary of supported response distributions.

Further options

Options are listed in alphabetical order:

- `begin =` *variablename*
  Option `begin` is meaningful only if `family=cox` is specified as the response distribution. In this case `begin` specifies the variable that records when the observation became at risk. This option can be used to handle left truncation and time-varying covariates. If `begin` is not specified, all observations are assumed to have become at risk at time 0.

- `level1 =` *integer*
  Besides the posterior means and medians, *BayesX* provides pointwise posterior credible intervals for every effect in the model. In a Bayesian approach based on MCMC simulation techniques credible intervals are estimated by computing the respective quantiles of the sampled effects. By default, *BayesX* computes (pointwise) credible intervals for nominal levels of 80% and 95 %. The option `level1` allows to redefine one of the nominal levels (95%). Adding, for instance,

  ```
  level1=99
  ```

  to the options list computes credible intervals for a nominal level of 99% rather than 95%.

- `level2 =` *integer*
  Besides the posterior means and medians, *BayesX* provides pointwise posterior credible intervals for every effect in the model. In a Bayesian approach based on MCMC simulation techniques credible intervals are estimated by computing the respective quantiles of the sampled effects. By default, *BayesX* computes (pointwise) credible intervals for nominal levels of 80% and 95 %. The option `level2` allows to redefine one of the nominal levels (80%). Adding, for instance,

  ```
  level2=70
  ```

  to the options list computes credible intervals for a nominal level of 70% rather than 80%.

- `predict`
  Option `predict` may be specified to compute samples of the deviance D, the effective number of parameters p_D and the deviance information criteria DIC of the model, see Spiegelhalter et al. (2002). The computation of these quantities is based on the unstandardized deviance which is defined as $D(\theta) = -2\log(p(y|\theta))$ where $\theta = (\mu, \sigma^2)$ for Gaussian responses, $\theta = (\mu, \delta)$ for negative binomial responses and $\theta = \mu$ for the rest of non-Gaussian responses. The effective number of parameters is defined by $p_D = \overline{D(\theta)} - D(\bar{\theta})$ where $\overline{D(\theta)}$ is the posterior mean deviance and $D(\bar{\theta})$ is the deviance of the posterior mean of θ. The deviance information criteria is defined as $DIC = \overline{D(\theta)} + p_D$. *BayesX* prints sample properties of the deviance, the effective number of parameters p_D and the DIC in the *output window* or in an open log file. The complete sample of the deviance is stored in a file with ending `deviance.raw`. The complete filename including the storage folder is given in the *output window* or the log file. The last two entries of that file contain again the effective number of parameters p_D and the DIC. Additionally, a file with ending `predictmean.raw` is created that contains for every observation the posterior mean of the predictor η_i and the expectation $E(y_i|\eta_i) = \mu_i$ as well as the saturated deviance D_i^{sat} and leverage statistics p_{D_i}. The saturated deviance is defined as $D(\mu, \sigma^2) = -2\log(p(y|\mu, \sigma^2)) + 2\log(p(y|\mu = y, \sigma^2))$. For non-Gaussian responses the variance σ^2 disappears and for negative binomial responses we have δ instead. The individual saturated deviance D_i^{sat} can be used to compute deviance residuals. The deviance residuals are given by $r_i = sign(y_i - \mu_i)\sqrt{D_i^{sat}}$. The leverage statistics p_{D_i} is defined

as the contribution of the ith observation to p_D. More details about the quantities discussed above can be found in Spiegelhalter et al. (2002). To clarify the computation of D, p_D, DIC etc. Table 7.6 provides formulas of the p.d.f. and the (unstandardized) deviance D for the different response distributions provided in *BayesX*.

distribution	**density**	**D =-2 Loglikelihood**
Gaussian	$p(y\|\mu,\sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2/c}}$ $\exp(-\frac{c}{2\sigma^2}(y-\mu)^2)$	$\log(\frac{2\pi\sigma^2}{c}) + \frac{c}{\sigma^2}(y-\mu)^2$
Binomial	$p(y\|\mu) \propto \mu^y(1-\mu)^{c-y}$	$-2y\log(\mu) - 2(c-y)\log(1-\mu)$
Poisson	$p(y\|\mu) \propto \exp((y\log(\mu)-\mu)c)$	$-2c(y\log(\mu)-\mu)$
Negative Binomial	$p(y\|\mu,\delta) \propto \frac{\Gamma(y+\delta)}{\Gamma(\delta)}\left(\frac{\delta}{\delta+\mu}\right)^\delta\left(\frac{\mu}{\delta+\mu}\right)^y$	$-2\{\log(\Gamma(y+\delta)) - \log(\Gamma(\delta)) + \delta\log(\delta)$ $+y\log(\mu) - (\delta+y)\log(\delta+\mu)\}$
Zero inflated Poisson	$p(0\|\mu,\theta) = \theta + (1-\theta)\exp(-\mu)$ $p(y\|\mu,\theta) \propto (1-\theta)\exp(-\mu)\mu^y$	$-2\log(\theta + (1-\theta)\exp(-\mu))$ $-2\log(1-\theta) - 2(y\log(\mu)-\mu)$
Zero inflated Negative Binomial	$p(0\|\mu,\delta,\theta) = \theta + (1-\theta)\left(\frac{\delta}{\delta+\mu}\right)^\delta$ $p(y\|\mu,\delta,\theta) \propto (1-\theta)\frac{\Gamma(y+\delta)}{\Gamma(\delta)}$ $\left(\frac{\delta}{\delta+\mu}\right)^\delta\left(\frac{\mu}{\delta+\mu}\right)^y$	$-2\log\left(\theta + (1-\theta)\left(\frac{\delta}{\delta+\mu}\right)^\delta\right)$ $-2\{\log(1-\theta)$ $+\log(\Gamma(y+\delta)) - \log(\Gamma(\delta)) + \delta\log(\delta)$ $+y\log(\mu) - (\delta+y)\log(\delta+\mu)\}$
Multinomial logit	$p(y\|\mu) \propto \prod \mu_j^{y_j}$	$-2(\sum y_j\log(\mu_j))$
Multinomial probit		**not available**
Cumulative probit	$p(y\|\mu) \propto \prod \mu_j^{y_j}$	$-2(\sum y_j\log(\mu_j))$

Table 7.6: Formulas of the probability densities, the unstandardized deviance and the saturated deviance for the various response distributions. The quantity c in the formulas corresponds to the weights specified in a weight statement, see weightspecification for details on how to specify weights. In the case of multinomial logit and cumulative probit models the variables y_j are indicator variables where $y_j = 1$ denotes that the j-th category of the response y has been observed.

7.1.4 Estimation output

The way the estimation output is presented depends on the estimated model. Estimation results of fixed effects are displayed in a tabular form in the *output window* and/or in a log file (if created before). Shown will be the posterior mean, the standard deviation, the 2.5% and the 97.5% quantiles. Other quantiles may be obtained by specifying the `level1` and/or `level2` option, see subsection 7.1.3 for details. Additionally a file is created where estimation results for fixed effects are replicated. The name of the file is given in the *output window* and/or in a log file. Estimation effects of nonlinear effects of continuous and spatial covariates as well as unstructured random effects are presented in a different way. Results are stored in an external ASCII-file whose contents can be read into any general purpose statistics program (e.g. STATA, S-plus) to further analyze and/or visualize the results. The structure of the files is as follows: There will be one file for every nonparametric effect in the model. The name of the files and the storing directory are displayed in the *output window* and/or a log file. The files contain ten or eleven columns depending on whether the corresponding model term is an interaction effect. The first column contains a parameter index (starting with one), the second column (and the third column if the estimated effect is a 2 dimensional P-spline) contain the values of the covariate(s) whose effect is estimated. In the following

columns the estimation results are given in form of the posterior means and the 2.5%, 10%, 50%, 90% and 97.5% quantiles. The last two columns contain posterior probabilities based on nominal levels of 95% and 80%. A value of 1 corresponds to a strictly positive 95 or 80% credible interval and a value of -1 to a strictly negative credible interval. A value of 0 indicates that the corresponding credible interval contains zero. Other quantiles may be obtained by specifying the `level1` and/or `level2` option, see subsection 7.1.3 for details. As an example compare the following few lines, that are the beginning of a file containing the results for a particular covariate, `x` say:

intnr	x	pmean	pqu2p5	pqu10	pmed	pqu90	pqu97p5	pcat95	pcat80
1	-2.778436	-0.0730973	-0.349922	-0.259827	-0.0765316	0.109233	0.211572	0	0
2	-2.723671	-0.167492	-0.39718	-0.322043	-0.168924	-0.0167056	0.075335	0	-1
3	-2.633617	-0.320366	-0.497797	-0.433861	-0.321034	-0.198619	-0.129246	-1	-1
4	-2.547761	-0.455913	-0.623495	-0.560266	-0.458746	-0.347443	-0.296006	-1	-1
5	-2.455208	-0.591498	-0.744878	-0.694039	-0.592381	-0.484629	-0.440857	-1	-1
6	-2.385378	-0.687709	-0.858153	-0.802932	-0.687944	-0.577029	-0.522391	-1	-1
7	-2.34493	-0.736406	-0.914646	-0.851548	-0.73536	-0.623369	-0.561035	-1	-1
8	-2.291905	-0.785899	-0.962212	-0.895262	-0.783646	-0.674511	-0.609532	-1	-1
9	-2.178096	-0.876173	-1.0428	-0.982029	-0.877516	-0.768126	-0.708452	-1	-1

Note that the first row always contains the names of the variables in the ten columns.

The estimated nonlinear effects can be visualized by using either the graphics capabilities of *BayesX* (Java based version only) or a couple of S-plus functions, see section 9.1 and section 9.2, respectively. Of course, any other (statistics) software package with plotting facilities may be used as well.

7.1.5 Examples

Here we give only a few examples about the usage of method `regress`. More detailed examples can be found in chapter 1 of the tutorial manual.

Suppose that we have a data set `test` with a binary response variable `y`, and covariates `x1`, `x2`, `x3` and `t`, where `t` is assumed to be a time scale measured in months. Suppose further that we have already created a *bayesreg object* `b`.

Fixed effects

We first specify a model with `y` as the response variable and fixed effects for the covariates `x1`, `x2` and `x3`. Hence the predictor is

$$\eta = \gamma_0 + \gamma_1 x1 + \gamma_2 x2 + \gamma_3 x3$$

This model is estimated by typing:

```
> b.regress y = x1 + x2 + x3, iterations=12000 burnin=2000
  family=binomial step=10 using test
```

Here, `step=10` defines the thinning parameter, i.e. only every 10th sampled parameter will be stored and used for estimation. `test` is the data set that is used for estimation. By specifying option `family=binomial`, a binomial logit model is estimated. A probit model can be estimated by specifying `family=binomialprobit`.

Additive models

Suppose now that we want to allow for possibly nonlinear effects of `x2` and `x3`. Defining cubic P-splines with second order random walk penalty as smoothness priors, we obtain

```
> b.regress y = x1 + x2(psplinerw2) + x3(psplinerw2), iterations=12000
  burnin=2000 family=binomial step=10 using test
```

which corresponds to the predictor

$$\eta = \gamma_0 + \gamma_1 x1 + f_1(x2) + f_2(x3).$$

Suppose now for a moment that the response is not binary but multicategorical with unordered categories 1, 2 and 3. In that case we can estimate either a multinomial logit or a probit model. A logit model is estimated by typing:

```
> b.regress y = x1 + x2(psplinerw2) + x3(psplinerw2), iterations=12000
  burnin=2000 family=multinomial reference=2 step=10 using test
```

That is, `family=binomial` was altered to `family=multinomial`, and the option `reference=2` was added in order to define the value 2 as the reference category. Accordingly, a multinomial probit model is estimated by typing

```
> b.regress y = x1 + x2(psplinerw2) + x3(psplinerw2), iterations=12000
  burnin=2000 family=multinomialprobit reference=2 step=10 using test
```

Time scales

In our next step we extend the model by incorporating an additional trend and a flexible seasonal component for the time scale `t`:

```
> b.regress y = x1 + x2(psplinerw2) + x3(psplinerw2) + t(psplinerw2) +
  t(season,period=12), iterations=12000 burnin=2000 family=binomial
  step=10 using test
```

Note that we passed the period of the seasonal component as a second argument.

Spatial covariates

Suppose now that we have an additional spatial covariate `region`, which indicates the geographical region an observation belongs to. To incorporate a structured spatial effect, we first have to create a *map object* and read in the boundary information of the different regions (polygons that form the regions, neighbors etc.). If you are unfamiliar with *map objects* please read chapter 5 first.

```
> map m
> m.infile using c:\maps\map.bnd
```

In a second step we reorder the regions of the map using the `reorder` command to obtain minimal bandwidths of the corresponding adjacency matrix of the map. This usually speeds up MCMC simulation for spatial effects.

```
> m.reorder
```

Since we normally need the map again in further sessions, we store the reordered map in *graph file* format, because reading *graph files* is much faster than reading *boundary files*.

```
> m.outfile , graph using c:\maps\mapgraph.gra
```

We can now extend our predictor with a spatial effect:

```
> b.regress y = x1 + x2(psplinerw2) + x3(psplinerw2) + t(psplinerw2) +
  t(season,period=12) + region(spatial,map=m), iterations=12000 burnin=2000
  family=binomial step=10 using test
```

In some situations it may be reasonable to incorporate an additional unstructured random effect into the model in order to split the total spatial effect into a structured and an unstructured

component. This is done by typing

```
> b.regress y = x1 + x2(psplinerw2) + x3(psplinerw2) + t(psplinerw2) +
  t(season,period=12) + region(spatial,map=m) + region(random), iterations=12000
  burnin=2000 family=binomial step=10 using test
```

7.2 Method autocor

Description

This method is a post estimation command, i.e. its usage is meaningful only if method `regress` has been applied before. Method `autocor` computes the autocorrelation functions of all sampled (and stored) parameters. The computed functions will be written to an external file whose name and storing path is printed in the *output window* and/or an additional log file. The computed autocorrelations can be visualized by using either method `plotautocor` of *graph objects* (Java based version only) or the S-plus function `plotautocor`.

Syntax

> *objectname*.`autocor` [, *options*]

The execution of this command computes autocorrelation functions for all sampled and stored parameters. An error will be raised if regression results are not yet available. The computed functions will be stored in an external file. The storing directory will be the current output directory of the *bayesreg object*. By default, this directory is `<INSTALLDIRECTORY>\output`, but the current output directory may be changed by redefining the global option `outfile`, see section 7.4. The filename will be the current output name extended by the ending `_autocor.raw`. By default, the output name is the name of the particular *bayesreg object*. Thus, if for example your *bayesreg object* name is `bayes`, the complete filename will be `bayes_autocor.raw`. Once again, the default output name may be changed using the global option `outfile` (section 7.4). Note that the autocorrelation file will be overwritten whenever method `autocor` is applied. As a remedy the current output directory and/or output name should be changed *before* estimating a new model, using the global option `outfile`, see section 7.4.

The structure of the file with the stored autocorrelation functions is the following: The computed functions are stored in a matrix like fashion. For every parameter the autocorrelation function will be stored columnwise, with autocorrelation for lag 1 in row 1, for lag 2 in row 2 and so on. The first column of the file contains the lag number. In addition, for each term in the estimated model, minimum, mean and maximum autocorrelations will be computed and stored. Note finally that the very first row of the file contains the column names.

The computed autocorrelations can be visualized by using either method `plotautocor` of *graph objects* (Java based version only) or the S-plus function `plotautocor`. However, since the structure of the file is very simple, the visualization of the functions can be done with every software package which has graphics capabilities.

Options

- `maxlag =` *integer*
  With the `maxlag` option, the maximum lag number for computing autocorrelations may be specified. *integer* must be a positive integer valued number.
  DEFAULT: `maxlag=250`

Examples

Suppose we have already defined a *bayesreg object* `b` and estimated a (simple) regression model with Gaussian responses using the following `regress` statement

```
> b.regress Y = X, family=gaussian using d
```

where `d` is the analyzed data set. The model contains only a fixed effect for covariate `X` and an intercept. We may now want to check the mixing of the sampled parameters (one for the overall variance parameter, one for the intercept and one for the fixed effect of `X`) by computing autocorrelation functions. The following statement computes autocorrelations up to lag 100 and stores the result in the default output directory with filename `b_autocor.raw`:

```
> b.autocor , maxlag=100
```

The default output directory is *<INSTALLDIRECTORY>*`\output`. So if, for instance, *BayesX* is installed in `c:\bayes`, the autocorrelation functions will the stored in `c:\bayes\output\b_autocor.raw`. If you wish to store the file in another directory, `c:\data` say, and under another name, for example `estimate1`, you must use the global option `outfile` before estimation (see also section 7.4). The following commands produce the desired result (program output between the different statements omitted):

```
> b.outfile = c:\data\estimate1
> b.regress Y = X, family=gaussian using d
> b.autocor , maxlag=100
```

Now the autocorrelation functions will be stored under `c:\data\estimate1_autocor.raw`.

The computed autocorrelation functions may now be visualized using the S-plus function `plotautocor`. The function plots the autocorrelation functions for all estimated parameters (in our example only three) against the lag number. If the option `mean.autocor=T` is specified, only minimum, mean and maximum autocorrelations for each term in the model are plotted against the lag number. Obviously, this is much faster than the first alternative, where the autocorrelation functions of all parameters are plotted.

The following statement in S-plus plots and stores the autocorrelations in the postscript file `c:\data\estimate1_autocor.ps`:

```
> plotautocor("c:\\data\\estimate1_autocor.raw","c:\\data\\estimate1_autocor.ps")
```

Note that double backslashes are required in S-plus to specify the directory of a file correctly. For more details about the function `plotautocor` compare subsection 9.2.5.

The autocorrelation functions can be plotted directly in *BayesX* using method `plotautocor` of *graph objects* (Java based version only). For that purpose, the computed autocorrelation functions must be first read into *BayesX* as a new data set, `a` say, and then visualized using method `plotautocor` of *graph objects*. The following commands produce the desired results:

```
> b.outfile = c:\data\estimate1
> b.regress Y = X, family=gaussian using d
> b.autocor , maxlag=100
> dataset a
> a.infile using c:\data\estimate1_autocor.raw
> graph g
> g.plotautocor using a
```

A third way for plotting autocorrelation functions is given by applying method `plotautocor` of *bayesreg objects* (Java based version only). Here autocorrelations are computed and plotted in one step.

7.3 Method getsample

Description

This method is a post estimation command, that is it is only meaningful if method `regress` has been applied before. With method `getsample` all sampled parameters will be stored in (one or more) ASCII file(s). Afterwards, sampling paths can be plotted and stored in a postscript file either by using method `plotsample` of *graph objects* (Java based version only) or by using the S-plus function `plotsample`. Of course, any other program with graphics capacities could be used as well.

Syntax

> *objectname*.`getsample`

This command stores all sampled parameters in ASCII file(s). An error will be raised, if regression results are not yet available. The storing directory will be the current output directory of the *bayesreg object*. By default, this directory is <*INSTALLDIRECTORY*>`\output`, but you can change the current output directory by redefining the global option `outfile`. The filenames will be the current output name extended by an ending depending on the type of the estimated effect. For example for fixed effects, the complete filename will be `b_FixedEffects1_sample.raw`, if `b` is the name of the *bayesreg object*. The total number of created files and their filenames are printed in the *output window* and/or an open log file. By default, the output name is the name of the *bayesreg object*. Once again, the default name may be changed using the global option `outfile`. Note that it can happen that some or all files will be overwritten, if method `getsample` is applied more than once with the same *bayesreg object*. To avoid such problems change the current output directory and/or output name *before* estimating a new model using the global option `outfile`.

The structure of the created files is as follows: The very first row contains the parameter names. In the following lines, the parameters are stored in a matrix like fashion. In the first row (to be precise the second row, since the first contains the names) the first sampled value of each parameter separated by blanks is stored. In the second row the second value is stored and so on. In the first column of each row the sampling number is printed.

Options

not allowed

Examples

Suppose we have already defined a *bayesreg object* `b` and estimated a (simple) regression model with Gaussian errors using the following `regress` statement:

```
> b.regress Y = X, family=gaussian using d
```

The model contains only a fixed effect for covariate `X` and an intercept. We may now want to check the mixing of the sampled parameters (one for the overall variance σ^2, one for the intercept and one for `X`) by storing sampled parameters in an ASCII-file and visualizing sampling paths. The statement

```
> b.getsample
```

forces *BayesX* to store the sampled parameters in files named:

b_FixedEffects1_sample.raw
b_intercept_sample.raw
b_scale_sample.raw

The storing directory is the current output directory, which is by default <INSTALLDIRECTORY\output. The current output directory can be changed using the global option outfile. For example the two commands

```
> b.outfile = c:\data\estimate1
> b.getsample
```

force the program to store the sampled parameters in the files:

c:\data\estimate1_FixedEffects1_sample.raw.
c:\data\estimate1_intercept_sample.raw
c:\data\estimate1_scale_sample.raw

The first few lines of the first file look like this:

intnr	b_1
1	-2.00499
2	-1.97745
3	-1.98498
4	-1.97108
5	-2.00004
⋮	

We can now visualize the sampling paths. If the Java based version of *BayesX* is installed we can plot sampling paths directly using method plotsample of *graph objects*. We first read in sampled parameters by typing:

```
> dataset s1
> s1.infile using c:\data\estimate1_FixedEffects1_sample.raw
```

We proceed by creating a *graph object* g and applying method plotsample:

```
> graph g
> g.plotsample using s1
```

If the Java based version is not installed we could use S-plus and the function plotsample for visualizing sampling paths. We type, for instance, in S-plus:

```
> plotsample("c:\\data\\estimate1_FixedEffects1_sample.raw")
```

7.4 Global options

The purpose of global options is to affect the global behavior of a *bayesreg object*. The main characteristic of global options is, that they are not associated with a certain method.

The syntax for specifying global options is

> *objectname.optionname* = *newvalue*

where *newvalue* is the new value of the option. The type of the value depends on the respective option.

The following global options are currently available for *bayesreg objects*:

- outfile = *filename*
  By default, the estimation output produced by the regress procedure will be written to the default output directory, which is

<INSTALLDIRECTORY>`\output`.

The default filename is composed of the name of the *bayesreg object* and the type of the file. For example, if you estimated a nonparametric effect for a covariate `X`, say, then the estimation output will be written to

<INSTALLDIRECTORY>`\output\b_nonpX.res`

where `b` is the name of the *bayesreg object*. In most cases, however, it may be necessary to save estimation results into a different directory and/or under a different filename than the default. This can be done using the `outfile` option. With the `outfile` option you have to specify the directory where the output should be stored to and in addition a base filename. The base filename should not be a complete filename. For example specifying

```
> b.outfile = c:\data\res
```

would force *BayesX* to store the estimation result for the nonparametric effect of `X` in file

`c:\data\res_nonpX.res`

- `iterationsprint =` *integer*

  By default, the current iteration number is printed in the *output window* (or in an additional log file) after every 100th iteration. This can lead to rather big and complex output files. The `iterationsprint` option allows to redefine after how many iterations the current iteration number is printed. For example `iterationsprint=1000` forces *BayesX* to print the current iterations number only after every 1000th iteration rather than after every 100th iteration.

7.5 Visualizing estimation results

Visualization of estimation results is described in chapter 9

7.6 Examples

In this Section we present a couple of complex examples about the usage of *bayesreg objects*. The first example contains a reanalysis of the 'credit scoring' data set that is described in subsection 2.6.2, which contains also a (incomplete) list of some publications where the 'credit scoring' data set has already been analyzed. The second example is a Bayesian analysis of determinants of childhood undernutrition in Zambia. The data set is described in subsection 2.6.3. This section contains also a list of publications where the data set has been analyzed. Both data sets are shipped together with *BayesX* and are stored in the directory `examples`, which is a subdirectory of the installation directory. Since the main focus here is on illustrating the usage of *bayesreg objects*, we omit any interpretation of estimated effects.

7.6.1 Binary data: credit scoring

All *BayesX* statements of this section can be found in the `examples` directory in the file `credit.prg`. In principle, the commands in `credit.prg` can be executed using the `usefile` command for running batch files, see section 3.5. Note, however, that the specified directories therein may not exist on your computer. Thus, to avoid errors, the file must be modified first to execute correctly.

7.6.1.1 Reading the data into BayesX

In order to analyze the 'credit scoring' data set, we first have to load the data set into *BayesX*. For the rest of this section we assume that *BayesX* is installed in the directory `c:\bayes`. In this case, the 'credit scoring' data set can be found in `c:\bayes\examples` under the name `credit.raw`. With the following two commands (entered in the *command window*) we first create a *dataset object* `credit` and afterwards load the data set into *BayesX* using the `infile` command:

```
> dataset credit
> credit.infile using c:\bayes\examples\credit.res
```

Since the first row of the file already contains the variable names, it is not necessary to specify variable names in the `infile` statement. If the first row of the data set does not contain the variable names, they must be additionally specified in the `infile` command, e.g. for the 'credit scoring' data set we get

```
> credit.infile y account duration amount payment intuse marstat
  using c:\bayes\examples\credit.res
```

We now compute effect coded versions of the categorical covariates `account`, `payment`, `intuse` and `marstat`:

```
> credit.generate account1  = 1*(account=1)-1*(account=3)
> credit.generate account2  = 1*(account=2)-1*(account=3)
> credit.generate payment1 = 1*(payment=1)-1*(payment=2)
> credit.generate intuse1 = 1*(intuse=1)-1*(intuse=2)
> credit.generate marstat1 = 1*(marstat=1)-1*(marstat=2)
```

The reference categories for the covariates are chosen to be 3 for `account` and 2 for the others.

7.6.1.2 Creating a bayesreg object

Before we are able to estimate Bayesian regression models, we first have to create a *bayesreg object*:

```
> bayesreg b
> b.outfile = c:\results\credit
```

The second command changes the default output directory and name (which is `c:\bayes\output\b`) to `c:\results\credit`. This means that subsequent regression output is stored in the directory `c:\results` and that all filenames start with `credit`.

7.6.1.3 Probit models

We can now start estimating models. We first describe how probit models are estimated. The estimation of probit models is slightly faster than logit models because the full conditionals of the effects are Gaussian.

We first estimate a model with fixed effects only:

```
> b.regress  y = account1 + account2 + duration + amount + payment1 + intuse1
  + marstat1, predict iterations=6000 burnin=1000 step=5 family=binomialprobit
  using credit
```

Here we specified 6000 iterations, a burnin period of 1000 iterations and a thinning parameter of 5, i.e. every 5th sampled parameter will be stored and used for estimation. The additional option `predict` is used to compute samples of the deviance, the effective number of parameters, the deviance information criteria (DIC), predicted means etc.

Executing the command yields the following output (simulation output omitted):

```
SIMULATION TERMINATED

SIMULATION RUN TIME: 13 seconds


ESTIMATION RESULTS:

  Predicted values:

  Estimated mean of predictors, expectation of response and
  individual deviances are stored in file
  c:\results\credit_predictmean.raw

  Estimation results for the Deviance:

  Unstandardized Deviance (-2*Loglikelihood(y|mu))

  Mean:             1027.05
  Std. Dev:         4.22501
  2.5% Quantile:    1021.01
  10% Quantile:     1022.48
  50% Quantile:     1026.29
  90% Quantile:     1032.73
  97.5% Quantile:   1037.28

  Saturated Deviance (-2*Loglikelihood(y|mu) + 2*Loglikelihood(y|mu=y))

  Mean:             1027.05
  Std. Dev:         4.22501
  2.5% Quantile:    1021.01
  10% Quantile:     1022.48
  50% Quantile:     1026.29
  90% Quantile:     1032.73
  97.5% Quantile:   1037.28

  Samples of the deviance are stored in file
  c:\results\credit_deviance_sample.raw

  Estimation results for the DIC:

  DIC based on the unstandardized Deviance

  Deviance(bar_mu):          1018.85
  pD:                        8.20049
  DIC:                       1035.26

  DIC based on the saturated Deviance

  Deviance(bar_mu):          1018.85
  pD:                        8.20049
  DIC:                       1035.26
```

```
FixedEffects1


Acceptance rate:    100 %


Variable  mean          Std. Dev.      2.5% quant.    median         97.5% quant.
const     -0.715437     0.117673       -0.9464        -0.710836      -0.49298
account1  -0.629553     0.0684571      -0.764526      -0.6244        -0.50343
account2  0.50699       0.0625032      0.389608       0.506698       0.63888
duration  0.0204795     0.00471734     0.0111948      0.0205776      0.0298505
amount    0.0172111     0.0189096      -0.0173778     0.0167277      0.0542619
payment1  -0.2919       0.0762402      -0.438703      -0.288309      -0.152512
intuse1   -0.137287     0.0477423      -0.228492      -0.137755      -0.041511
marstat1  -0.158195     0.0476991      -0.254198      -0.157534      -0.0663364

Results for fixed effects are also stored in file
c:\results\credit_FixedEffects1.res
```

Somewhat surprisingly, we observe that the amount of credit seems to have no ('significant') influence on the response. To check this phenomenon more carefully, we run a second estimation, now allowing for possibly nonlinear effects of the continuous covariates `amount` and `duration`. We choose cubic P-splines with second order random walk penalty as smoothness priors and modify the `regress` statement above according to the new model:

```
> b.regress  y = account1 + account2 + duration(psplinerw2) + amount(psplinerw2)
  + payment1 + intuse1 + marstat1, predict iterations=6000 burnin=1000 step=5
  family=binomialprobit using credit
```

We get the following output for the nonlinear functions (output for the rest omitted):

```
f_duration_pspline


Acceptance rate:    100 %

Results are stored in file c:\results\credit_f_duration_pspline.res

Postscript file is stored in file c:\results\credit_f_duration_pspline.ps

Results may be visualized using method 'plotnonp'
Type for example: objectname.plotnonp 1


f_duration_pspline_variance


Acceptance rate:    100 %

Estimation results for the variance component:

Mean:             0.00787338
Std. dev.:        0.0100301
2.5% Quantile:    0.00128376
10% Quantile:     0.00190016
50% Quantile:     0.00487717
```

```
90% Quantile:     0.0157856
97.5% Quantile:   0.0315107

Results for the variance component are also stored in file
c:\results\credit_f_duration_pspline_var.res


f_amount_pspline


Acceptance rate:    100 %

Results are stored in file c:\results\credit_f_amount_pspline.res

Postscript file is stored in file c:\results\credit_f_amount_pspline.ps

Results may be visualized using method 'plotnonp'
Type for example: objectname.plotnonp 3


f_amount_pspline_variance


Acceptance rate:    100 %

Estimation results for the variance component:

Mean:             0.00870842
Std. dev.:        0.0115895
2.5% Quantile:    0.0013844
10% Quantile:     0.00220351
50% Quantile:     0.00570725
90% Quantile:     0.0164858
97.5% Quantile:   0.0340326

Results for the variance component are also stored in file
c:\results\credit_f_amount_pspline_var.res
```

We visualize estimated effects for `amount` and `duration` using method `plotnonp` (as advised by the program):

```
> b.plotnonp 1 , outfile="c:\results\credit_duration.ps"
> b.plotnonp 3 , outfile="c:\results\credit_amount.ps"
```

This produces the graphs (stored in postscript files) shown in Figure 7.1.

We add a title, x-axis and y-axis labels by typing

```
> b.plotnonp 1 , outfile="c:\results\credit_duration.ps" replace
  xlab="duration" ylab="f(duration)" title="effect of duration"
> b.plotnonp 3 , outfile="c:\results\credit_amount.ps" replace
  xlab="amount" ylab="f(amount)" title="effect of amount"
```

and obtain the improved graphs shown in Figure 7.2. The option `replace` is specified to allow *BayesX* to overwrite the previously generated postscript files. If the `outfile` option is omitted, the graphs are printed on the screen rather than being stored as postscript files.

We now want to check the mixing of the generated Markov chains, although the mixing for probit models is usually excellent. For that reason we compute and plot the autocorrelation functions by

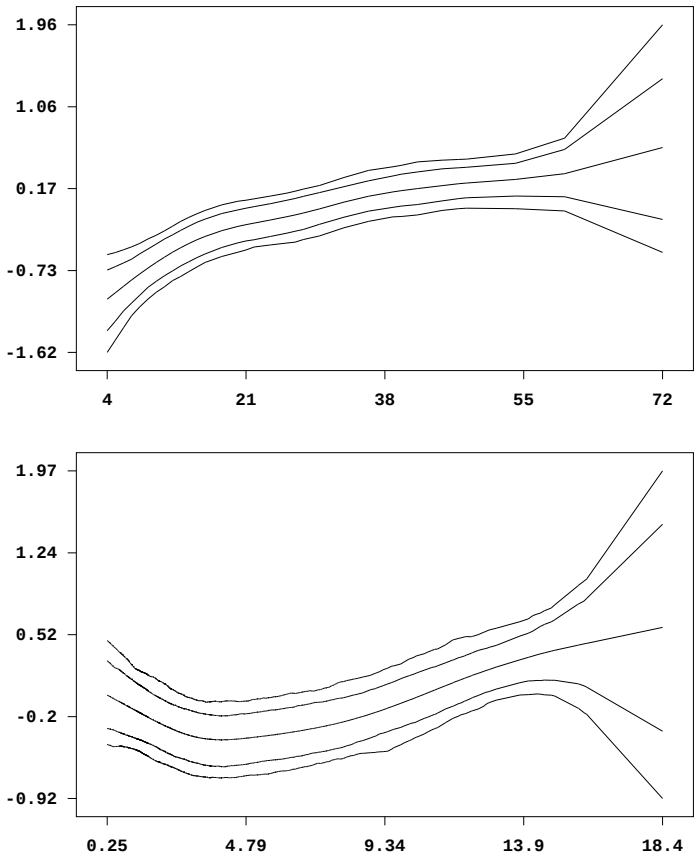


Figure 7.1: Estimated effects of duration and amount of credit. Shown is the posterior mean within 80% and 95% credible regions.

typing:

```
> b.plotautocor , outfile="c:\results\credit_autocor.ps"
```

We obtain the file `c:\results\credit_autocor.ps` containing 9 pages of autocorrelation functions for all parameters in the model. The first page of this file is shown in Figure 7.3. We see that autocorrelations die off very quickly.

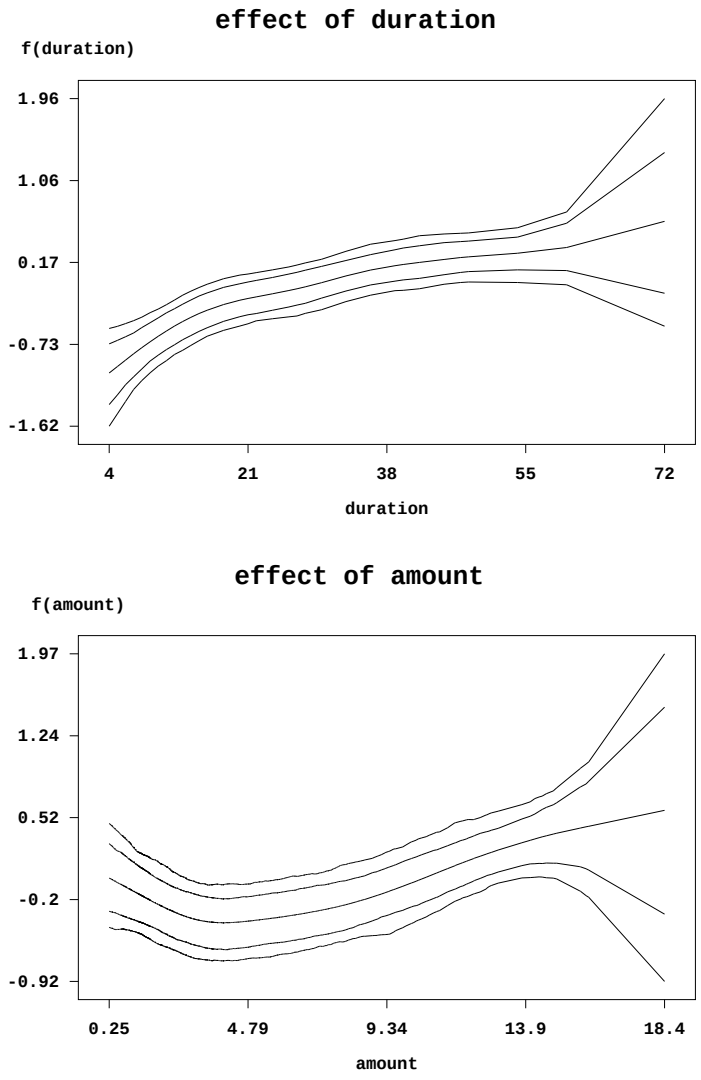


Figure 7.2: Improved plots of the effect of `duration` *and* `amount`.

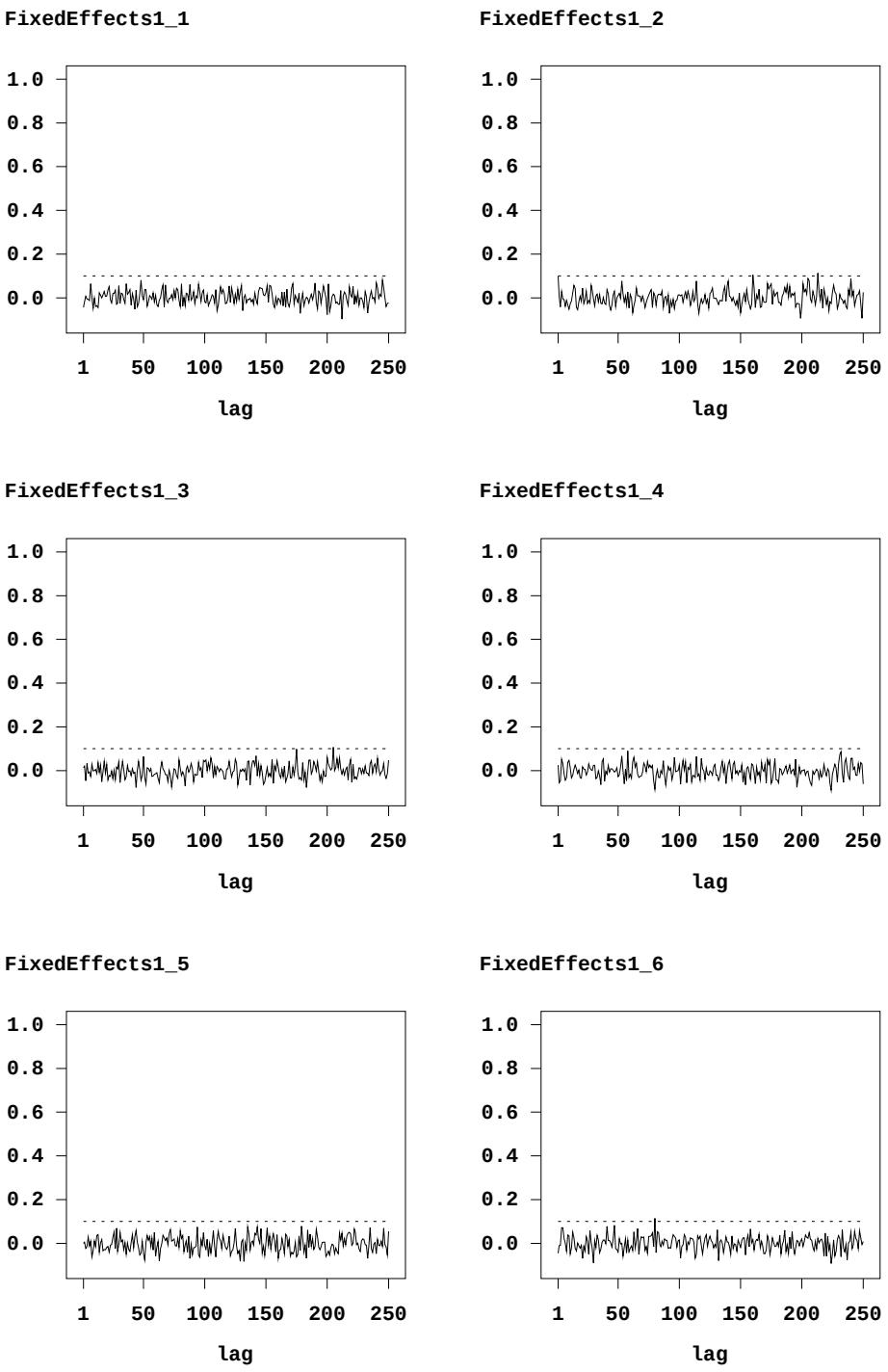


Figure 7.3: First page of the autocorrelation file.

7.6.1.4 Logit models

A logit model rather than a probit model is estimated by replacing `family=binomialprobit` with `family=binomial`:

```
> b.regress  y = account1 + account2 + duration(psplinerw2) + amount(psplinerw2)
  + payment1 + intuse1 + marstat1, predict iterations=6000 burnin=1000 step=5
  family=binomial using credit
```

In contrast to binary probit models, the full conditionals for the regression coefficients are no longer Gaussian. *BayesX* offers 3 different types of proposal densities. These are iteratively weighted least squares (IWLS) proposals based either on the current state of the parameters or on the posterior modes as described in subsubsection 4.1.3 or Brezger and Lang (2003), and conditional prior proposals as described in Fahrmeir and Lang (2001b). We recommend the usage of IWLS proposals, since no tuning is required and mixing properties are superior to those of conditional prior proposals. The default are IWLS proposals based on the current state of the parameters. The following statement causes *BayesX* to use IWLS proposals based on posterior modes, which usually yield even higher acceptance probabilities compared to ordinary IWLS proposals:

```
> b.regress  y = account1 + account2 + duration(psplinerw2,proposal=iwlsmode)
  + amount(psplinerw2,proposal=iwlsmode) + payment1 + intuse1 + marstat1,
  predict iterations=6000 burnin=1000 step=5
  family=binomial using credit
```

As for the probit model, we visualize the estimated nonlinear effects of `duration` and `amount` using method `plotnonp`:

```
> b.plotnonp 1 , outfile="c:\results\credit_logit_duration.ps" replace
  xlab="duration" ylab="f(duration)" title="effect of duration"
> b.plotnonp 3 , outfile="c:\results\credit_logit_amount.ps" replace
  xlab="amount" ylab="f(amount)" title="effect of amount"
```

The resulting graphs are shown in Figure 7.4. As could have been expected only the scale of the estimated effects differs (because of the logit link).

Once again, to check the mixing of the sampled parameters we compute and plot the autocorrelation functions using method `plotautocor`:

```
> b.plotautocor , outfile="c:\results\credit_logit_autocor.ps"
```

The first page of the resulting postscript file is shown in Figure 7.5. As can be seen, the autocorrelations for the logit model with IWLS proposals are almost as low as for the probit model.

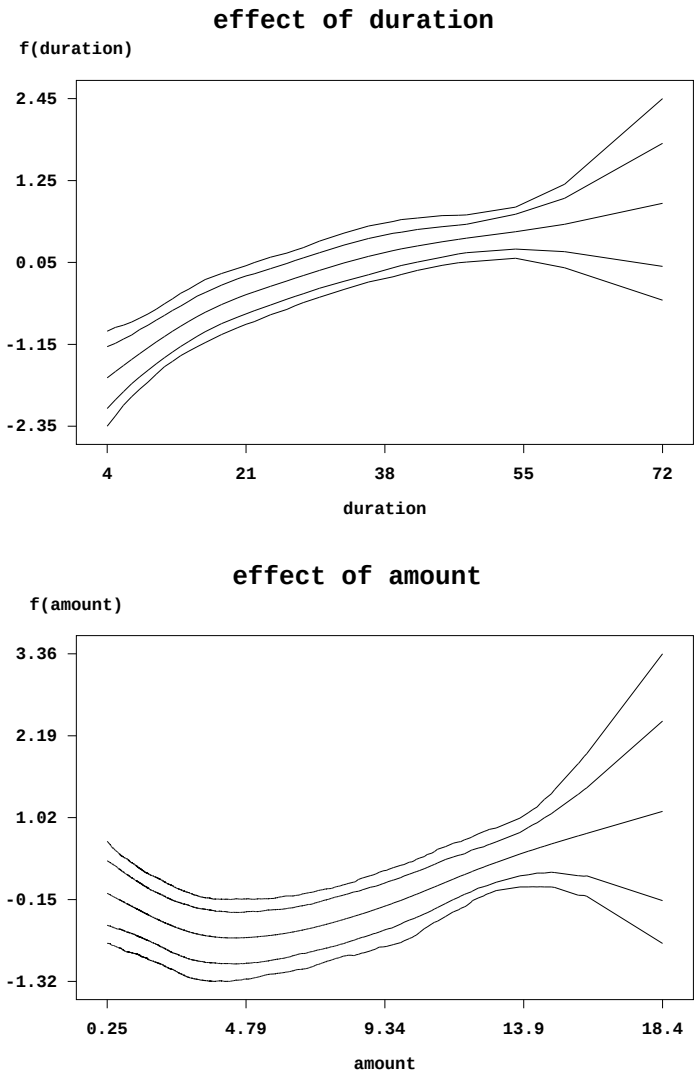


Figure 7.4: Effect of `duration` *and* `amount`*, if a logit model is estimated rather than a probit model.*

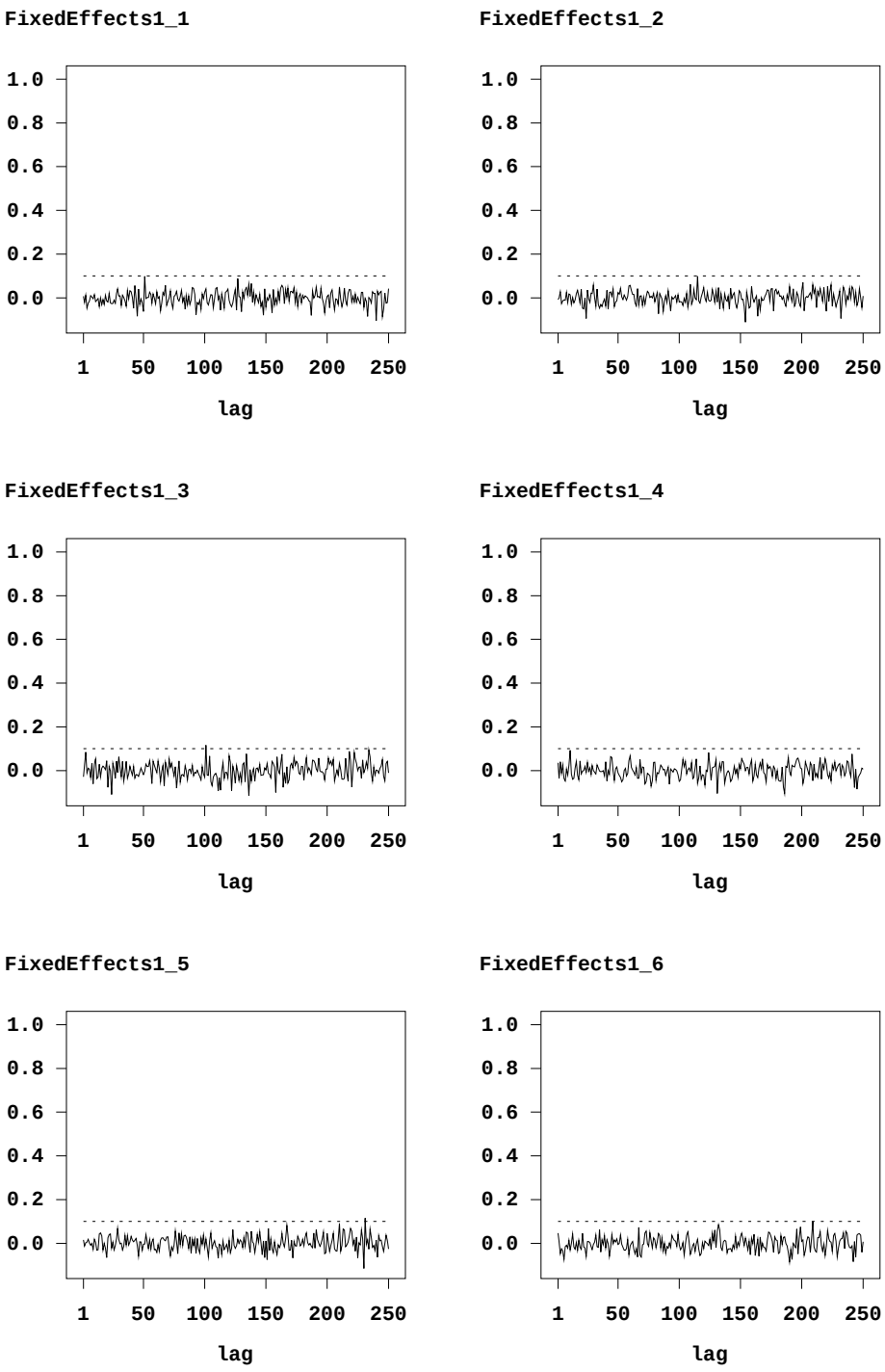


Figure 7.5: First page of the autocorrelation file, if a logit model is estimated.

7.6.1.5 Varying the hyperparameters

In the preceding examples we used the default hyperparameters `a=0.001` and `b=0.001` for the inverse gamma prior of the variances. In some situations, however, the estimated nonlinear functions may considerably depend on the particular choice of hyperparameters `a` and `b`. This may be the case for very low signal to noise ratios or/and small sample sizes. It is therefore highly recommended to estimate all models under consideration using a (small) number of *different* choices for `a` and `b` (e.g. `a=1,b=0.005`; `a=0.001,b=0.001`; `a=0.0001,b=0.0001`) to assess the dependence of results on minor changes in the model assumptions. In that sense, the variation of hyperparameters can be used as a tool for model diagnostics.

We estimate our probit model from subsubsection 7.6.1.3 again, but now with hyperparameters `a=1.0`, `b=0.005` and `a=0.0001`, `b=0.0001`, respectively.

```
> b.regress  y = account1 + account2 + duration(psplinerw2,a=1.0,b=0.005) +
  amount(psplinerw2,a=1.0,b=0.005) + payment1 + intuse1 + marstat1,
  predict iterations=6000 burnin=1000 step=5 family=binomialprobit using credit
> b.regress  y = account1 + account2 + duration(psplinerw2,a=0.0001,b=0.0001) +
  amount(psplinerw2,a=0.0001,b=0.0001) + payment1 + intuse1 + marstat1,
  predict iterations=6000 burnin=1000 step=5 family=binomialprobit using credit
```

Figure 7.6 shows the estimated nonlinear effects of variables `duration` and `amount` with the different choices for `a` and `b`. We see that in this example estimation results differ only slightly for the different choices of `a` and `b`.

7.7 References

Besag, J., Green, P., Higdon, D. and Mengersen, K. (1995): Bayesian computation and stochastic systems (with discussion). *Statistical Science*, 10, 3-66.

Besag, J. and Kooperberg, C. (1995): On conditional and intrinsic autoregressions. *Biometrika*, 82, 733-746.

Besag, J., York, J. and Mollie, A. (1991): Bayesian image restoration with two applications in spatial statistics (with discussion). *Annals of the Institute of Statistical Mathematics*, 43, 1-59.

Biller, C. (2000): *Bayesianische Ansätze zur nonparametrischen Regression.* Skaker Verlag, Aachen.

Brezger, A. (2000): *Bayesianische P-splines.* Master thesis, University of Munich.

Brezger, A. and Lang, S. (2003): Generalized additive regression based on Bayesian P-splines. *Computational Statistics and Data Analysis* (to appear).

Clayton, D. (1996): Generalized linear mixed models. In: Gilks, W., Richardson S. and Spiegelhalter D. (eds), *Markov Chain Monte Carlo in Practice*. London: Chapman and Hall, 275-301.

Chen, M.H. and Dey, D.K. (2000): Bayesian Analysis for Correlated Ordinal Data Models. *Generalized linear models: A Bayesian perspective* (ed. Dey, D.K., Ghosh, S.K. and Mallick, B.K.), 8,133-159, Marcel Dekker, New York.

Chib, S. and Greenberg, E. (1995): Understanding the Metropolis-Hastings Algorithm. *The American Statistician*, 49, 327-335.

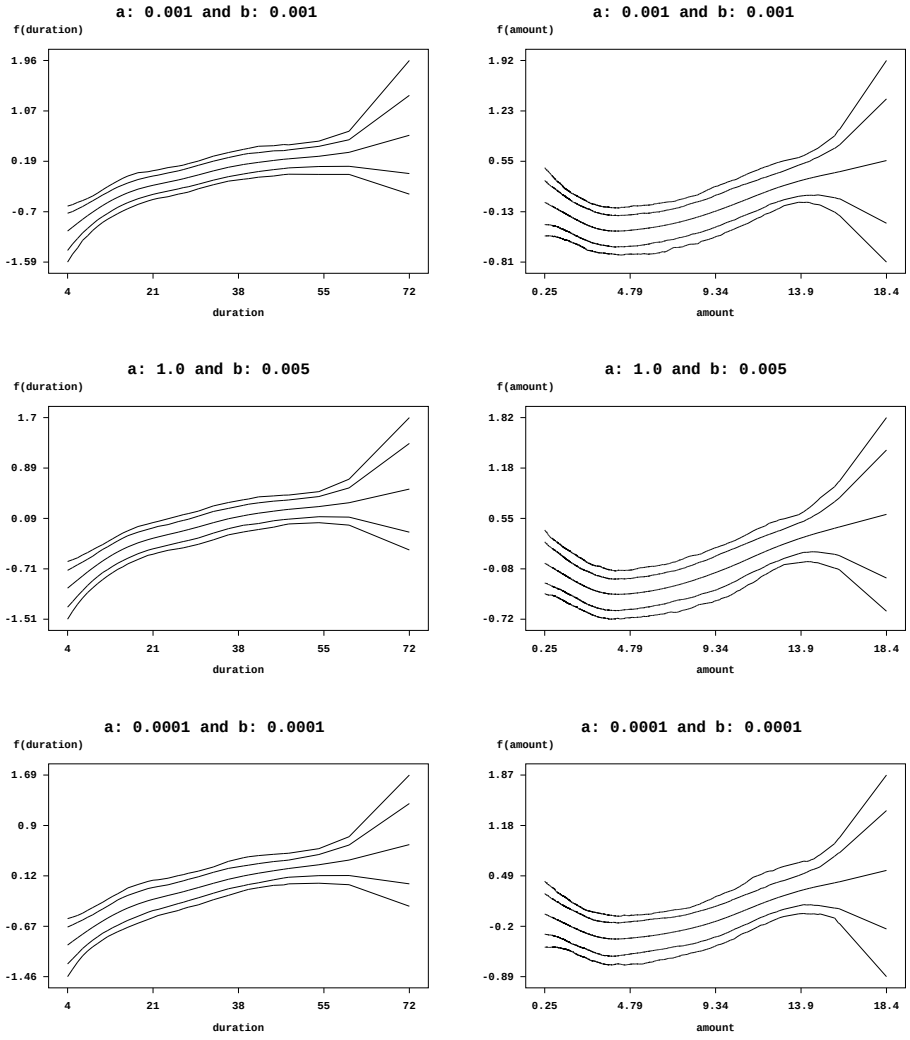


Figure 7.6: Results for the effect of `duration` *and* `amount` *for different values of the hyperparameters for the variances.*

Eilers, P.H.C. and Marx, B.D. (1996): Flexible smoothing using B-splines and penalized likelihood (with comments and rejoinder). *Statistical Science*, 11 (2), 89-121.

Fahrmeir, L. and Lang, S. (2001a): Bayesian Inference for Generalized Additive Mixed Models Based on Markov Random Field Priors. *Journal of the Royal Statistical Society C*, 50, 201-220.

Fahrmeir, L. and Lang, S. (2001b): Bayesian Semiparametric Regression Analysis of Multicategorical Time-Space Data. *Annals of the Institute of Statistical Mathematics*, 53, 10-30.

Fahrmeir, L. and Tutz, G. (2001): *Multivariate Statistical Modelling based on Generalized Linear Models.* New York: Springer-Verlag.

Gamerman, D. (1997): Efficient Sampling from the posterior distribution in generalized linear models. *Statistics and Computing*, 7, 57-68.

Gelfand, A.E., Sahu, S.K. and Carlin, B.P. (1996): Efficient Parametrizations for General-

ized Linear Mixed Models. In: Bernardo, J.M., Berger, J.O., Dawid, A.P. and Smith, A.F.M. (eds.), *Bayesian Statistics, 5.* Oxford University Press, 165-180.

George, A. and Liu, J.W. (1981). *Computer Solution of Large Sparse Positive Definite Systems.* Series in computational mathematics, Prentice-Hall.

Green, P.J. (2001): A Primer in Markov Chain Monte Carlo. In: Barndorff-Nielsen, O.E., Cox, D.R. and Klüppelberg, C. (eds.), *Complex Stochastic Systems.* Chapmann and Hall, London, 1-62.

Green, P.J. and Silverman, B. (1994): *Nonparametric Regression and Generalized Linear Models.* Chapman and Hall, London.

Hastie, T. and Tibshirani, R. (1990): *Generalized additive models.* Chapman and Hall, London.

Hastie, T. and Tibshirani, R. (1993): Varying-coefficient Models. *Journal of the Royal Statistical Society B*, 55, 757-796.

Hastie, T. and Tibshirani, R. (2000): Bayesian Backfitting. *Statistical Science*, 15, 193-223.

Hastie, T., Tisbshirani, R. and Friedman, J. (2001): *The Elements of Statistical Learning: Data Mining, Inference and Prediction.* New York: Sprigner-Verlag.

Knorr-Held, L. (1999): Conditional Prior Proposals in Dynamic Models. *Scandinavian Journal of Statistics*, 26, 129-144.

Kragler, P. (2000): Statistische Analyse von Schadensfällen privater Krankenversicherungen. Master thesis, University of Munich.

Lang, S. (1996): Bayesianische Inferenz in Modellen mit variierenden Koeffizienten. Master thesis, University of Munich.

Lang, S. and Brezger, A. (2004): Bayesian P-splines. *Journal of Computational and Graphical Statistics*, 13, 183-212.

McCullagh, P. and Nelder, J.A. (1989): *Generalized Linear Models.* Chapman and Hall, London.

Osuna, L. (2004): *Semiparametric Bayesian Count Data Models.* Dr. Hut Verlag, München.

Rue, H. (2001): Fast Sampling of Gaussian Markov Random Fields with Applications. *Journal of the Royal Statistical Society B*, 63, 325-338.

Spiegelhalter, D.J., Best, N.G., Carlin, B.P. and van der Linde, A. (2002): Bayesian measures of model complexity and fit. *Journal of the Royal Statistical Society B*, 65, 583-639.

Chapter 8

remlreg objects

Authors: Andreas Brezger, Thomas Kneib and Stefan Lang
email: andib@stat.uni-muenchen.de, kneib@stat.uni-muenchen.de, and lang@stat.uni-muenchen.de

Remlreg objects are used to fit generalized linear models with a *structured additive predictor (STAR)*, see Fahrmeir, Kneib and Lang (2004). Inference is based on mixed model representations of the regression models and may be seen as empirical Bayes / posterior mode estimation. The methodological background is provided in considerable detail in the methodology manual. More details on models for univariate responses can be found in Fahrmeir, Kneib and Lang (2004), Kneib and Fahrmeir (2004a) deals with models for multicategorical responses. A description of models for continuous time survival analysis based on the Cox model can be found in Kneib and Fahrmeir (2004b) Good introductions into generalized linear models are the monographs of Fahrmeir and Tutz (2001) and McCullagh and Nelder (1989). Introductions to semi- and nonparametric models are given in Green and Silverman (1994), Hastie and Tibshirani (1990), Hastie and Tibshirani (1993) and Hastie, Tibshirani and Friedman (2001).

First steps with *remlreg objects* can be done with the example in chapter 2 of the tutorial manual.

8.1 Method regress

8.1.1 Syntax

> *objectname*.`regress` *model* [`weight` *weightvar*] [`if` *expression*] [, *options*] `using` *dataset*

Method `regress` estimates the regression model specified in *model* using the data specified in *dataset*. *dataset* must be the name of a *dataset object* created before. The details of correct models are covered in subsubsection 8.1.1.2. The distribution of the response variable can be either Gaussian, binomial, multinomial, Poisson or gamma, see also Table 8.5 for an overview about the models supported by *remlreg objects*. The response distribution is specified using option `family`, see subsubsection 8.1.1.4 below. The default is `family=binomial` with a logit link. An `if` statement may be specified to analyze only a part of the data set, i.e. the observations where *expression* is true.

8.1.1.1 Optional weight variable

An optional weight variable *weightvar* may be specified to estimate weighted regression models. For Gaussian responses *BayesX* assumes that $y_i|\eta_i, \sigma^2 \sim N(\eta_i, \sigma^2/weightvar_i)$. Thus, for grouped

Gaussian responses the weights must be the number of observations in the groups if the y_i's are the average of individual responses. If the y_i's are the sum of responses in every group, the weights must be the reciprocal of the number of observations in the groups. Of course, estimation of usual weighted regression models if the errors are heteroscedastic is also possible. In this case the weights should be proportional to the reciprocal of the heteroscedastic variances. If the response distribution is binomial, it is assumed that the values of the weight variable correspond to the number of replications and that the values of the response variable correspond to the number of successes. If weight is omitted, *BayesX* assumes that the number of replications is one, i.e. the values of the response must be either zero or one. For grouped Poisson data the weights must be the number of observations in a group and the y_i's are assumed to be the average of individual responses. Weights are not allowed for models with multicategorical response.

8.1.1.2 Syntax of possible model terms

The general syntax of models for *remlreg objects* is:

$depvar = term_1 + term_2 + \cdots + term_r$

depvar specifies the dependent variable in the model and $term_1$,...,$term_r$ define in which way the covariates influence the dependent variable. The different terms must be separated by '+' signs. A constant intercept is automatically included in the models and must not be specified by the user. This section reviews all possible model terms that are supported in the current version of *remlreg objects* and provides some specific examples. Note that all described terms may be combined in arbitrary order. An overview about the capabilities of *remlreg objects* is given in Table 8.1. Table 8.2 shows how interactions between covariates are specified. Full details about all available options are given in subsubsection 8.1.1.3.

Throughout this section Y denotes the dependent variable.

Offset

Description: Adds an offset term to the predictor.

Predictor: $\eta = \cdots + offs + \cdots$

Syntax: `offs(offset)`

Example:

For example, the following model statement can be used to estimate a poisson model with `offs` as offset term and `W1` and `W2` as fixed effects (if `family=poisson` is specified in addition):

```
Y = offs(offset) + W1 + W2
```

Fixed effects

Description: Incorporates covariate `W1` as a fixed effect into the model.

Predictor: $\eta = \cdots + \gamma_1 W1 + \cdots$

Syntax: `W1`

Example:

The following model statement causes `regress` to estimate a model with q fixed (linear) effects:

```
Y = W1 + W2 + ··· + Wq
```

Type	Syntax example	Description
offset	`offs(offset)`	Variable `offs` is an offset term.
linear effect	`W1`	Linear effect for `W1`.
first or second order random walk	`X1(rw1)` `X1(rw2)`	Nonlinear effect of `X1`.
P-spline	`X1(psplinerw1)` `X1(psplinerw2)`	Nonlinear effect of `X1`.
seasonal prior	`time(season,period=12)`	Varying seasonal effect of `time` with period 12.
Markov random field	`region(spatial,map=m)`	Spatial effect of `region` where `region` indicates the region an observation pertains to. The boundary information and the neighborhood structure is stored in the *map object* `m`.
Two dimensional P-spline	`region(geospline,map=m)`	Spatial effect of `region`. Estimates a two dimensional P-spline based on the centroids of the regions. The centroids are stored in the *map object* `m`.
Stationary Gaussian random field	`region(geokriging)`	Spatial effect of `region`. Estimates a stationary Gaussian random field based on the centroids of the regions. The centroids are stored in the *map object* `m`.
random intercept	`grvar(random)`	I.i.d. (random) Gaussian effect of the group indicator `grvar`, e.g. `grvar` may be an individuum indicator when analyzing longitudinal data.
baseline in Cox models	`time(baseline)`	Nonlinear shape of the baseline effect $\lambda_0(time)$ of a Cox model. $\log(\lambda_0(time))$ is modelled by a P-spline with second order penalty.

Table 8.1: Overview over different model terms for remlreg objects.

Nonlinear effects of metrical covariates and time scales

First or second order random walk

Description: Defines a first or second order random walk prior for the effect of `X1`.

Predictor: $\eta = \cdots + f_1(X1) + \cdots$

Syntax:

`X1(rw1`[, *options*]`)`

`X1(rw2`[, *options*]`)`

Example:

Suppose we have a continuous covariate `X1`, whose effect is assumed to be nonlinear. The following model statement defines a second order random walk prior for f_1:

```
Y = X1(rw2)
```

Here, the expression `X1(rw2)` indicates, that the effect of `X1` should be incorporated nonparametrically into the model using a second order random walk prior. A first order random walk is specified in the model statement by modifying `rw2` to `rw1` which yields the term `X1(rw1)`.

P-spline with first or second order random walk penalty

Description: Defines a P-spline with a first or second order random walk penalty for the parameters of the spline.

Predictor: $\eta = \cdots + f_1(X1) + \cdots$

Type of interaction	Syntax example	Description

Varying coefficient term	`X1*X2(rw1)`
	`X1*X2(rw2)`
	`X1*X2(psplinerw1)`
	`X1*X2(psplinerw2)`
	`X1*time(season)`
		Effect of `X1` varies smoothly over the range of the continuous covariate `X2` or `time`, respectively.

random slope	`X1*grvar(random)`	The regression coefficient of `X1` varies with respect to the unit- or cluster index variable `grvar`.
Geographically weighted regression	`X1*region(spatial,map=m)`	Effect of `X1` varies geographically. Covariate `region` indicates the region an observation pertains to.
Two dimensional surface	`X1*X2(pspline2dimrw1)`	Two dimensional surface for the continuous covariates `X1` and `X2`.
Stationary Gaussian random field	`X1*X2(kriging)`	Stationary Gaussian random field for coordinates `X1` and `X2`.
Time-varying effect in Cox Models	`X1*time(baseline)`	Nonlinear, time-varying effect of `X1`.

Table 8.2: *Possible interaction terms for remlreg objects.*

Syntax:

`X1(psplinerw1`[, *options*]`)`

`X1(psplinerw2`[, *options*]`)`

Example:

For example, a P-spline with second order random walk penalty is obtained using the following model statement:

```
Y = X1(psplinerw2)
```

By default, the degree of the spline is 3 and the number of inner knots is 20. The following model term defines a quadratic P-spline with 30 knots:

```
Y = X1(psplinerw2,degree=2,nrknots=30)
```

Seasonal component for time scales

Description: Defines a seasonal effect of `time`.

Predictor: $\eta = \cdots + f_{season}(time) + \cdots$

Syntax:

`time(season`[, *options*]`)`

Example:

A seasonal component for a time scale `time` is specified for example by

`Y = time(season,period=12)`.

Here, the second argument specifies the period of the seasonal effect. In the example above the period is 12, corresponding to monthly data.

Nonlinear baseline effect in Cox models

P-spline with second order random walk penalty

Description: Defines a P-spline with second order random walk penalty for the parameters of the spline for the log-baseline effect $\log(\lambda_0(\texttt{time}))$.

Predictor: $\eta = \log(\lambda_0(time)) + \cdots$

Syntax:

`time(baseline`[, *options*]`)`

Example:

Suppose continuous-time survival data (`time`, `delta`) together with additional covariates (`W1`,`X1`) are given, where `time` denotes the vector of observed duration times, `delta` is the vector of corresponding indicators of non-censoring, `W1` is a discrete covariate and `X1` a continuous one. The following Cox model with hazard rate λ and log-baseline effect $\log(\lambda_0(\texttt{time}))$

$$\begin{aligned} \lambda(time) &= \lambda_0(time)\exp(\gamma_0 + \gamma_1 W1 + f(X1)) \\ &= \exp\left(\log(\lambda_0(time)) + \gamma_0 + \gamma_1 W1 + f(X1)\right) \end{aligned}$$

is estimated by the model statement

```
delta = time(baseline) + W1 + X1(psplinerw2)
```

Spatial Covariates

Markov random field

Description:

Defines a Markov random field prior for the spatial covariate `region`. *Remlreg objects* allow an appropriate incorporation of spatial covariates with geographical information stored in the *map object* specified through the option `map`.

Predictor: $\eta = \cdots + f_{spat}(region) + \cdots$

Syntax:

`region(spatial,map=`*characterstring*[, *options*]`)`

Example:

The specification of a Markov random field prior for spatial data has `map` as a required argument which must be the name of a *map object* (see chapter 5) that contains all necessary spatial information about the geographical map, i.e. the neighbors of each region and the weights associated with the neighbors. For example the statement

```
Y = region(spatial,map=germany)
```

defines a Markov random field prior for `region` where the geographical information is stored in the *map object* `germany`. An error will be raised if `germany` is not existing.

2 dimensional P-spline with first order random walk penalty

Description:

Defines a 2 dimensional P-spline for the spatial covariate `region` based on the tensor product of 1 dimensional P-splines with a 2 dimensional first order random walk penalty for the parameters of the spline. Estimation is based on the coordinates of the centroids of the regions an observation pertains to. The centroids are computed using the geographical information stored in the *map object* specified through the option `map`.

Predictor: $\eta = \cdots + f(centroids) + \cdots$

Syntax:

`region(geospline,map=`*characterstring*[, *options*]`)`

Example:

The specification of a 2 dimensional P-spline (`geospline`) for spatial data has `map` as a required argument which must be the name of a *map object* (see chapter 5) that contains all necessary spatial information about the geographical map, i.e. the neighbors of each region and the weights associated with the neighbors. The model term

```
Y = region(geospline,map=germany)
```

specifies a tensor product cubic P-spline with first order random walk penalty where the geographical information is stored in the *map object* `germany`.

Unordered group indicators

Unit- or cluster specific unstructured effect

Description: Defines an unstructured (uncorrelated) random effect with respect to grouping variable `grvar`.

Predictor: $\eta = \cdots + f(grvar) + \cdots$

Syntax:

`grvar(random`[, *options*]`)`

Example:

Method regress supports Gaussian i.i.d. random effects to cope with unobserved heterogeneity among units or clusters of observations. Suppose the analyzed data set contains a group indicator `grvar` that gives information about the individual or cluster a particular observation belongs to. Then an individual specific uncorrelated random effect is incorporated through the term

```
Y = grvar(random)
```

The inclusion of more than one random effect term in the model is possible, allowing the estimation of multilevel models. However, we have only limited experience with multilevel models so that it is not clear how well these models can be estimated using *remlreg objects*.

Varying coefficients with metrical covariates as effect modifier

First or second order random walk

Description:

Defines a varying coefficient term, where the effect of `X1` varies smoothly over the range of `X2`. Covariate `X2` is the effect modifier. The smoothness prior for f is a first or second order random walk.

Predictor: $\eta = \cdots + f(X2)X1 + \cdots$

Syntax:

`X1*X2(rw1`[, *options*]`)`

`X1*X2(rw2`[, *options*]`)`

Example:

For example, a varying coefficient term with a second order random walk smoothness prior is defined as follows:

```
Y = X1*X2(rw2)
```

P-spline with first or second order random walk penalty

Description:

Defines a varying coefficient term, where the effect of `X1` varies smoothly over the range of `X2`. Covariate `X2` is the effect modifier. The smoothness prior for f is a P-spline with first or second order random walk penalty.

Predictor: $\eta = \cdots + f(X2)X1 + \cdots$

Syntax:

`X1*X2(psplinerw1`[, *options*]`)`

`X1*X2(psplinerw2`[, *options*]`)`

Example:

For example, a varying coefficient term with a second order random walk smoothness prior is defined as follows:

```
Y = X1*X2(psplinerw2)
```

Seasonal prior

Description:

Defines a varying coefficients term where the effect of `X1` varies over the range of the effect modifier `time`. For `time` a seasonal prior is used.

Predictor: $\eta = \cdots + f_{season}(time)X1 + \cdots$

Syntax:

`X1*time(season`[, *options*]`)`

Example:

The inclusion of a varying coefficients term with a seasonal prior may be meaningful if we expect a different seasonal effect with respect to grouping variable `X1`. In this case we can include additional seasonal effects for each category of `X1` by

```
Y = X1*time(season)
```

Time-varying effects in Cox models

P-spline with second order random walk penalty

Description: Defines a varying coefficients term where the effect of `X1` varies over the range of the effect modifier `time`, i.e. variable `X1` has time-varying effect. The smoothness prior for $f($`time`$)$ is a P-spline with second order random walk penalty.

Predictor: $\eta = \log(\lambda_0(time)) + f(time)X1 \cdots$

Syntax:

`X1*time(baseline`[, *options*]`)`

Example:

Suppose continuous-time survival data (`time`, `delta`) together with an additional covariate `X1` are given, where `time` denotes the vector of observed duration times, `delta` is the vector of corresponding indicators of non-censoring. The following Cox model with hazard rate λ

$$\begin{aligned} \lambda(time) &= \lambda_0(time)\exp(\gamma_0 + f(time)X1) \\ &= \exp\left(\log(\lambda_0(time)) + \gamma_0 + f(time)X1\right) \end{aligned}$$

is estimated by the model statement

```
delta = time(baseline) + X1*time(baseline)
```

Varying coefficients with spatial covariates as effect modifiers

Markov random field

Description:

Defines a varying coefficient term where the effect of `X1` varies smoothly over the range of the spatial covariate `region`. A Markov random field is estimated for f_{spat}. The geographical information is stored in the *map object* specified through the option `map`.

Predictor: $\eta = \cdots + f_{spat}(region)X1 + \cdots$

Syntax:

`X1*region(spatial,map=`*characterstring* `[,` *options*`])`

Example:

For example the statement

```
Y = X1*region(spatial,map=germany)
```

defines a varying coefficient term with the spatial covariate `region` as the effect modifier and a Markov random field as spatial smoothness prior. Weighted Markov random fields can be estimated by including an appropriate weight definition when creating the *map object* `germany` (see section 5.1).

Varying coefficients with unordered group indicators as effect modifiers (random slopes)

Unit- or cluster specific unstructured effect

Description:

Defines a varying coefficient term where the effect of `X1` varies over the range of the group indicator `grvar`. Models of this type are usually referred to as models with random slopes. A Gaussian i.i.d. random effect with respect to grouping variable `grvar` is assumed for f.

Predictor: $\eta = \cdots + f(grvar)X1 + \cdots$

Syntax:

`X1*grvar(random`[, *options*]`)`

Example:

For example, a random slope is specified as follows:

```
Y = X1*grvar(random)
```

Note, that in contrast to *bayesreg objects* no main effects are included automatically. If main effects should be included in the model, they have to be specified as additional fixed effects. The syntax for obtaining the predictor

Predictor: $\eta = \cdots + \gamma X1 + f(grvar)X1 + \cdots$

would be

`X1 + X1*grvar(random`[, *options*]`)`

Surface estimators

2 dimensional P-spline with first order random walk penalty

Description:

Defines a 2 dimensional P-spline based on the tensor product of 1 dimensional P-splines with a 2 dimensional first order random walk penalty for the parameters of the spline.

Predictor: $\eta = \cdots + f(X1, X2) + \cdots$

Syntax:

`X1*X2(pspline2dimrw1`[, *options*]`)`

Example:

The model term

```
Y = X1*X2(pspline2dimrw1)
```

specifies a tensor product cubic P-spline with first order random walk penalty.

In many applications it is favorable to additionally incorporate the 1 dimensional main effects of `X1` and `X2` into the models. In this case the 2 dimensional surface can be seen as the deviation from the main effects. Note, that in contrast to *bayesreg objects* the number of inner knots and the degree of the spline may be different for the main effects and for the interaction. For example, a model with 20 inner knots for the main effects and 10 inner knots for the 2 dimensional P-Spline is estimated by

```
Y = X1(psplinerw2,nrknots=20) + X2(psplinerw2,nrknots=20)
    + X1*X2(pspline2dimrw1,nrknots=10)
```

Stationary Gaussian random field

Description:

Defines that the parameters of the locations follow a stationary Gaussian random field. Depending on the chosen options, locations are given either by the distinct pairs of `X1` and `X2` or by a subset of these pairs, which we will also refer to as knots. Note that, although stationary Gaussian random fields can be used to estimate surfaces depending on arbitrary variables `X1` and `X2`, they are defined based on *isotropic* correlation functions. This means that correlations between sites that have the same distance also have the same correlation, regardless of direction and the sites location. Therefore, if Gaussian random fields shall be used to estimate interactions between variables that do not represent longitude and latitude, these variables have to be standardized appropriately.

Predictor: $\eta = \cdots + f(X1, X2) + \cdots$

Syntax:

`X1*X2(kriging`[, *options*]`)`

Example:

The model term

```
Y = X1*X2(kriging,nrknots=100)
```

specifies a stationary Gaussian random field for the effect of `X1` and `X2` with 100 knots, which are computed based on the space filling algorithm described in section 3.2 of the methodology manual. If all distinct pairs of `X1` and `X2` shall be used as knots, we have to specify

```
Y = X1*X2(kriging,full)
```

Note, that the knots computed by the space filling algorithm are stored in the outfile directory of the *remlreg object*. These knots can be read into a *dataset object* which may be passed in the call of method `regress` if we want to use the same knots as in previous calls:

```
dataset kn
kn.infile using knotfile
Y = X1*X2(kriging,knotdata=kn)
```

To determine the actual number of knots, the options are interpreted in a specific sequence. If option `full` is specified, both `nrknots` and `knotdata` are ignored. Similarly, `nrknots` is ignored if `knotdata` is specified.

8.1.1.3 Description of additional options for terms of remlreg objects

All arguments described in this section are optional and may be omitted. Generally, options are specified by adding the option name to the specification of the model term type in the parentheses, separated by comma. Note that all options may be specified in arbitrary order. Table 8.3 provides explanations and the default values of all possible options. In Table 8.4 all reasonable combinations of model terms and options can be found.

8.1.1.4 Specifying the response distribution

The current version of *BayesX* supports the most common univariate distributions of the response for the use with *remlreg objects*. These are Gaussian, binomial (with logit or probit link), Poisson and gamma. An overview over the supported models is given in Table 8.5. In *BayesX* the distribution of the response is specified by adding the additional option `family` to the options list of the

optionname	description	default
`lambdastart`	Provides a starting value for the smoothing parameter λ.	`lambdastart=10`
`degree`	Specifies the degree of the B-spline basis functions.	`degree=3`
`nrknots`	Specifies the number of inner knots for a P-spline term or the number of knots for a kriging term.	`nrknots=20` (P-splines) `nrknots=100` (kriging)
`knotdata`	*Dataset object* containing the knots to be used with the kriging term	no default.
`full`	Specifies that all distinct locations should be used as knots with the kriging term.	-
`nu`	The smoothness parameter ν of the Matèrn correlation function.	`nu=1.5`
`maxdist`	Specifies the value c that is used to determine the scale parameter ρ of the Matèrn correlation function. Compare section 3.2 of the methodology manual.	default depends on `nu`
`p`	Defines the parameter p used in the coverage criterion of the space filling algorithm.	`p=-20`
`q`	Defines the parameter q used in the coverage criterion of the space filling algorithm.	`q=20`
`maxsteps`	Specifies the maximum number of steps to be performed by the space filling algorithm.	`maxsteps=300`
`gridchoice`	How to choose grid points for numerical integration in Cox models. May be either '`quantiles`' or '`equidistant`'.	`gridchoice=quantiles`
`tgrid`	Number of equidistant time points to be used for numerical integration in Cox models. Only meaningful if `gridchoice=equidistant`.	`tgrid=100`
`nrquantiles`	Number of quantiles that are used to define the grid points for numerical integration in Cox models. First a grid of `nrquantiles` quantiles is computed, then the grid for integration is defined by `nrbetween` equidistant points between each quantile. Only meaningful if `gridchoice=quantiles`.	`nrquantiles=50`
`nrbetween`	Number of points between quantiles that are used to define the grid points for numerical integration in Cox models. First a grid of `nrquantiles` quantiles is computed, then the grid for integration is defined by `nrbetween` equidistant points between each quantile. Only meaningful if `gridchoice=quantiles`.	`nrbetween=5`
`map`	*Map object* for spatial effects.	no default
`period`	The period of the seasonal effect can be specified with the option `period`. The default is `period=12` which corresponds to monthly data.	`period=12`

Table 8.3: Optional arguments for remlreg object *terms*

regress command. For instance, `family=gaussian` defines the responses to be Gaussian. In the following we give detailed instructions on how to specify the different models:

Gaussian responses

For Gaussian responses *BayesX* assumes $y_i|\eta_i, \sigma^2 \sim N(\eta_i, \sigma^2/weightvar_i)$ or equivalently in matrix notation $y|\eta, \sigma^2 \sim N(\eta, \sigma^2 C^{-1})$. Here $C^{-1} = diag(weightvar_1, \ldots, weightvar_n)$ is a known covariance matrix. Gaussian response is specified by adding

```
family=gaussian
```

to the options list.

An optional weight variable *weightvar* may be specified to estimate weighted regression models, see subsection 8.1.1 for the syntax. For grouped Gaussian responses the weights must be the number of observations in the groups if the y_i's are the average of individual responses. If the y_i's are the sum of responses in every group, the weights must be the reciprocal of the number of observations in the groups. Of course, estimation of usual weighted regression models if the errors are heteroscedastic is also possible. In this case the weights should be proportional to the reciprocal of the heteroscedastic variances. If a weight variable is not specified, *BayesX* assumes $weightvar_i = 1$, $i = 1, \ldots, n$.

	rw1/rw2	season	psplinerw1/psplinerw2	spatial	random	geospline	pspline2dimrw1	kriging	geokriging	baseline
`lambdastart`*	realvalue	realvalue	realvalue	realvalue	realvalue	realvalue	realvalue	realvalue	realvalue	realvalue
`degree`	×	×	integer	×	×	integer	integer	×	×	integer
`nrknots`	×	×	integer	×	×	integer	integer	integer	×	integer
`knotdata`	×	×	×	×	×	×	×	*dataset object*	*dataset object*	×
`full`	×	×	×	×	×	×	×	△	△	×
`nu`	×	×	×	×	×	×	×	•	•	×
`maxdist`*	×	×	×	×	×	×	×	realvalue	realvalue	×
`p`**	×	×	×	×	×	×	×	realvalue	realvalue	×
`q`*	×	×	×	×	×	×	×	realvalue	realvalue	×
`maxsteps`	×	×	×	×	×	×	×	integer	integer	×
`gridchoice`	×	×	×	×	×	×	×	×	×	○
`tgrid`	×	×	×	×	×	×	×	×	×	integer
`nrquantiles`	×	×	×	×	×	×	×	×	×	integer
`nrbetween`	×	×	×	×	×	×	×	×	×	integer
`period`	×	integer	×	×	×	×	×	×	×	×
`map`	×	×	×	*map object*	×	*map object*	×	×	*map object*	×

*	positive values only
**	negative values only
×	not available
•	admissible values are `0.5,1.5,2.5,3.5`
△	available as boolean option (specified without supplying a value)
○	admissible values are '`quantiles`' and '`equidistant`'

Table 8.4: Terms and options for remlreg objects

value of `family`	response distribution	link	options
`family=gaussian`	Gaussian	identity	

`family=binomial`
`family=binomialprobit`
`family=binomialcomploglog`
	binomial
	binomial
	binomial
		logit
		probit
		complementary log-log	

`family=multinomial`	unordered multinomial	logit	`reference`
`family=cumprobit` `family=cumlogit`	cumulative multinomial cumulative multinomial	probit logit	
`family=poisson`	Poisson	log-link	
`family=gamma`	gamma	log-link	
`family=cox`	continuous-time survival data		

Table 8.5: Summary of supported response distributions.

Binomial logit, probit and complementary log-log models

A binomial logit model is specified by adding the option

`family=binomial`

a probit model by adding

`family=binomialprobit`

and a complementary log-log model by adding

`family=binomialcomploglog`

to the option list.

A weight variable may be additionally specified, see subsection 8.1.1 for the syntax. *BayesX* assumes that the weight variable corresponds to the number of replications and the response variable to the number of successes. If the weight variable is omitted, *BayesX* assumes that the number of replications is one, i.e. the values of the response must be either zero or one.

Multinomial logit models

So far *remlreg objects* support only multinomial logit models. A multinomial logit model is specified by adding the option

`family=multinomial`

to the options list.

Usually a second option must be added to the options list to define the reference category. This is achieved by specifying the `reference` option. Suppose that the response variable has three categories 1,2 and 3. To define, for instance, the reference category to be 2, simply add

`reference=2`

to the options list. If this option is omitted, the *smallest* number will be used as the reference category.

Cumulative logit and probit models

A cumulative logit model is specified by adding

`family=cumlogit`

to the options list, a cumulative probit model by adding

```
family=cumprobit
```

to the options list. The reference category will always be the largest value of the response.

Note, that in contrast to *bayesreg objects remlreg objects* can deal with an arbitrary number of ordered categories. However, for more than 5 categories estimation will become rather computer intensive and time demanding.

Poisson regression

A Poisson regression is specified by adding

```
family=poisson
```

to the options list.

A weight variable may be additionally specified, see subsection 8.1.1 for the syntax. For grouped Poisson data the weights must be the number of observations in a group and the responses are assumed to be the average of individual responses.

Gamma distributed responses

In the literature, the density function of the gamma distribution is parameterized in various ways. In the context of regression analysis the density is usually parameterized in terms of the mean μ and the scale parameter s. Then, the density of a gamma distributed random number y is given by

$$p(y) \propto y^{s-1} \exp(-\frac{s}{\mu} y) \tag{8.1}$$

for $y > 0$. For the mean and the variance we obtain $E(y) = \mu$ and $Var(y) = \mu^2/s$. We write $y \sim G(\mu, s)$

A second parameterization is based on hyperparameters a and b and is usually used in the context of Bayesian hierarchical models to specify hyperpriors for variance components. The density is then given by

$$p(y) \propto y^{a-1} \exp(-by) \tag{8.2}$$

for $y > 0$. In this parameterization we obtain $E(y) = a/b$ and $Var(y) = a/b^2$ for the mean and the variance, respectively. We write $y \sim G(a, b)$

In *BayesX* gamma distributed response is defined in the first parameterization (8.1). For the rth observation *BayesX* assumes $y_r|\eta_r, \nu \sim G(\exp(\eta_r), \nu/weightvar_r)$ where $\mu_r = \exp(\eta_r)$ is the mean and $s = \nu/weightvar_r$ is the scale parameter. Gamma distributed response is specified by adding

```
family=gamma
```

to the options list. An optional weight variable *weightvar* may be specified to estimate weighted regression models. In this case the weights should be proportional to the reciprocal of the heteroscedastic variances, see subsection 8.1.1 for the syntax.

Continuous time survival analysis

BayesX offers two alternatives of estimating continuous time Cox models with semiparametric predictor η, which are described in section 5.2 of the methodology manual. The first alternative is to assume that all time-dependent values are piecewise constant, which leads to the so called *piecewise exponential model* (p.e.m.), and the second one is to estimate the log-baseline effect $\log(\lambda_0(t)) = f_0(t)$ by a P-spline with second order random walk penalty.

Piecewise exponential model (p.e.m.)

In section 5.2 of the methodology manual we demonstrated how continuous time survival data has to be manipulated such that a Poisson model may be used for estimation. Suppose now we have the modified data set

y	indnr	a	δ	Δ	x1	x2
0	1	0.1	1	log(0.1)	0	3
0	1	0.2	1	log(0.1)	0	3
1	1	0.3	1	log(0.05)	0	3
0	2	0.1	0	log(0.1)	1	5
0	2	0.2	0	log(0.02)	1	5
⋮	⋮	⋮	⋮	⋮	⋮	⋮

with indicator `y`, interval limit `a`, indicator of non-censoring δ and offset Δ defined as in section 5.2 of the methodology manual. Let `x1` be a covariate with linear effect and `x2` a continuous one with a nonlinear effect. Then the correct syntax for estimating a p.e.m. with a *remlreg object* named `r` is e.g. as follows:

```
> r.regress y = a(rw1) + Delta(offset) + x1 + x2(psplinerw2), family=poisson ...
```

or

```
> r.regress y = a(rw2) + Delta(offset) + x1 + x2(psplinerw2), family=poisson ...
```

Note that a time-varying effect of a covariate `X` may be estimated in the p.e.m. by adding the term

`X*a(rw1) or X*a(rw2)`

to the model statement.

Specifying a P-spline prior for the log-baseline

For the estimation of a Cox model with a P-spline prior with second order random walk penalty

`family=cox`

has to be specified in the options list. The number of knots and degree of the P-spline prior for $f_0(t)$ may be specified in the baseline term. The indicator of non-censoring δ_i has to be specified as the dependent variable in the model statement. Data augmentation and the specification of an offset term are not required here.

In the example above with survival data

t	δ	x1	x2
0.25	1	0	3
0.12	0	1	5
⋮	⋮	⋮	⋮

a Cox model with a quadratic P-spline prior with 15 knots for the log-baseline would be estimated as follows:

```
> r.regress delta = t(baseline,degree=2,nrknots=15)+ x1 + x2(psplinerw2),
  family=cox ...
```

Note, that we assume that a *remlreg object* `r` has been created before executing the command.

8.1.2 Options

Options for controlling the estimation process

Options for controlling estimation process are listed in alphabetical order.

- `eps` = *realvalue*

  Defines the termination criterion of the estimation process. If both the relative changes in the regression coefficients and the variance parameters are less than `eps`, the estimation process is assumed to have converged.
  DEFAULT: `eps = 0.00001`

- `lowerlim` = *realvalue*
  Since small variances are near to the boundary of their parameter space, the usual Fisher-scoring algorithm for their determination has to be modified. If the fraction of the penalized part of an effect relative to the total effect is less than `lowerlim`, the estimation of the corresponding variance is stopped and the estimator is defined to be the current value of the variance (see section 4.2 of the methodology manual for details).
  DEFAULT: `lowerlim = 0.001`

- `maxit` = *integer*
  Defines the maximum number of iterations to be used in estimation. Since the estimation process will not necessarily converge, it may be useful to define an upper bound for the number of iterations. Note, that *BayesX* produces results based on the current values of all parameters even if no convergence could be achieved within `maxit` iterations, but a warning message will be printed in the *output window*.
  DEFAULT: `maxit=400`

- `maxchange` = *realvalue*
  Defines the maximum value that is allowed for relative changes in parameters in one iteration to prevent the program from crashing because of numerical problems. Note, that *BayesX* produces results based on the current values of all parameters even if the estimation procedure is stopped due to numerical problems, but an error message will be printed in the *output window*.
  DEFAULT: `maxchange=1000000`

Further options

- `level1` = *integer*
  Besides the posterior mode, `regress` provides (approximate) pointwise posterior credible intervals for every effect in the model. By default, *BayesX* computes credible intervals for nominal levels of 80% and 95%. The option `level1` allows to redefine one of the nominal levels (95%). Adding, for instance,

  ```
  level1=99
  ```

  to the option list leads to the computation of credible intervals for a nominal level of 99% rather than 95%.

- `level2` = *integer*
  Besides the posterior mode, `regress` provides (approximate) pointwise posterior credible intervals for every effect in the model. By default, *BayesX* computes credible intervals for

nominal levels of 80% and 95%. The option `level2` allows to redefine one of the nominal levels (95%). Adding, for instance,

```
level2=70
```

to the option list leads to the computation of credible intervals for a nominal level of 70% rather than 80%.

8.1.3 Estimation output

The way the estimation output is presented depends on the estimated model. Estimation results for fixed effects are displayed in a tabular form in the *output window* and/or in a log file (if created before). Shown will be the posterior mode, the standard deviation, p-values and an approximate 95% credible interval. Other credible intervals may be obtained by specifying the `level1` option, see subsection 8.1.2 for details. Additionally a file is created where estimation results for fixed effects are replicated. The name of the file is given in the *output window* and/or in a log file.

Estimation effects for nonparametric effects are presented in a different way. Here, results are stored in external ASCII-files whose contents can be read into any general purpose statistics program (e.g. STATA, S-plus) to further analyze and/or visualize the results. The structure of these files is as follows: There will be one file for every nonparametric effect in the model. The name of the files and the storing directory are displayed in the *output window* and/or a log file. The files contain ten or eleven columns, depending on whether the corresponding model term is an interaction effect. The first column contains a parameter index (starting with one), the second column (and the third column if the estimated effect is a 2 dimensional P-spline) contain the values of the covariate(s) whose effect is estimated. In the following columns the estimation results are given in form of the posterior mode, the lower boundaries of the (approximate) 95% and 80% credible intervals, the standard deviation and the upper boundaries of the 80% and 95% credible intervals. The last two columns contain approximations to the posterior probabilities based on nominal levels of 95% and 80%. A value of 1 corresponds to a strictly positive 95% or 80% credible interval and a value of -1 to a strictly negative credible interval. A value of 0 indicates that the corresponding credible interval contains zero. Other credible intervals and posterior probabilities may be obtained by specifying the `level1` and/or `level2` option, see subsection 8.1.2 for details. As an example compare the following lines, which are the beginning of a file containing the results for a nonparametric effect of a particular covariate, x say:

intnr	x	pmode	ci95lower	ci80lower	std	ci80upper	ci95upper	pcat95	pcat80
1	-2.87694	-0.307921	-0.886815	-0.686408	0.295295	0.070567	0.270973	0	0
2	-2.86203	-0.320479	-0.885375	-0.689815	0.288154	0.0488558	0.244416	0	0
3	-2.8515	-0.329367	-0.88473	-0.69247	0.283292	0.0337362	0.225997	0	0
4	-2.85066	-0.330072	-0.884692	-0.692689	0.282913	0.0325457	0.224549	0	0
5	-2.82295	-0.3535	-0.884544	-0.700703	0.270887	-0.00629671	0.177545	0	-1
6	-2.79856	-0.37418	-0.886192	-0.708939	0.261178	-0.0394208	0.137832	0	-1
7	-2.79492	-0.377272	-0.886579	-0.710263	0.259798	-0.0442813	0.132035	0	-1
8	-2.79195	-0.379788	-0.886921	-0.711358	0.258689	-0.0482183	0.127345	0	-1
9	-2.78837	-0.382834	-0.887367	-0.712704	0.257363	-0.0529641	0.1217	0	-1

Note that the first row of the files always contains the names of the columns.

The estimated nonlinear effects can be visualized using either the graphics capabilities of *BayesX* (Java based version only) or a couple of S-plus functions, see section 9.1 and section 9.2, respectively. Of course, any other (statistics) software package with plotting facilities may be used as well.

Estimation results for the variances and the smoothing parameters of nonparametric effects are

printed in the *output window* and/or a log file. Additionally, a file is created containing the same information. For example, the file corresponding to the nonparametric effect presented above contains:

```
variance   smoothpar   stopped
0.0492324   20.3118   0
```

The value in the last row indicates whether the estimation of the variance has been stopped before convergence. A value of 1 corresponds to a 'stopped' variance.

8.1.4 Examples

Here we give only a few examples about the usage of method `regress`. A more detailed, tutorial like example can be found in chapter 2 of the tutorial manual.

Suppose that we have a data set `test` with a binary response variable `y`, and covariates `x1`, `x2`, `x3`, `t` and `region`, where `t` is assumed to be a time scale measured in months and `region` indicates the geographical region an observation belongs to. Suppose further that we have already created a *remlreg object* `r`.

Fixed effects

We first specify a model with `y` as the response variable and fixed effects for the covariates `x1`, `x2` and `x3`. Hence the predictor is

$$\eta = \gamma_0 + \gamma_1 x1 + \gamma_2 x2 + \gamma_3 x3$$

This model is estimated by typing:

```
> r.regress y = x1 + x2 + x3, family=binomial using test
```

By specifying option `family=binomial`, a binomial logit model is estimated. A probit model can be estimated by specifying `family=binomialprobit`.

Additive models

Suppose now that we want to allow for possibly nonlinear effects of `x2` and `x3`. Defining cubic P-splines with second order random walk penalty as smoothness priors, we obtain

```
> r.regress y = x1 + x2(psplinerw2) + x3(psplinerw2), family=binomial using test
```

which corresponds to the predictor

$$\eta = \gamma_0 + \gamma_1 x1 + f_1(x2) + f_2(x3).$$

Suppose now for a moment that the response is not binary but multicategorical with unordered categories 1, 2 and 3. In that case we can estimate a multinomial logit model. Such a model is estimated by typing:

```
> r.regress y = x1 + x2(psplinerw2) + x3(psplinerw2), family=multinomial
  reference=2 using test
```

That is, `family=binomial` was altered to `family=multinomial`, and the option `reference=2` was added in order to define the value 2 as the reference category.

Time scales

In our next step we extend the model by incorporating an additional trend and a flexible seasonal component for the time scale `t`:

```
> r.regress y = x1 + x2(psplinerw2) + x3(psplinerw2) +
  t(psplinerw2) + t(season,period=12), family=binomial using test
```

Note that we passed the period of the seasonal component as a second argument.

Spatial covariates

To incorporate a structured spatial effect, we first have to create a *map object* and read in the boundary information of the different regions (polygons that form the regions, neighbors etc.). If you are unfamiliar with *map objects* please read chapter 5 first.

```
> map m
> m.infile using c:\maps\map.bnd
```

Since we usually need the map again in further sessions, we store it in *graph file* format, because reading *graph files* is much faster than reading *boundary files*.

```
> m.outfile , graph using c:\maps\mapgraph.gra
```

We can now extend our predictor with a spatial effect:

```
> r.regress y = x1 + x2(psplinerw2) + x3(psplinerw2) + t(psplinerw2)
  + t(season,period=12) + region(spatial,map=m), family=binomial using test
```

In some situations it may be reasonable to incorporate an additional unstructured random effect into the model in order to split the total spatial effect into a structured and an unstructured component. This is done by typing

```
> r.regress y = x1 + x2(psplinerw2) + x3(psplinerw2) + t(psplinerw2)
  + t(season,period=12) + region(spatial,map=m) +region(random),
  family=binomial using test
```

8.2 Global options

The purpose of global options is to affect the global behavior of a *remlreg object*. The main characteristic of global options is, that they are not associated with a certain method.

The syntax for specifying global options is

objectname.optionname = *newvalue*

where *newvalue* is the new value of the option. The type of the value depends on the respective option.

Currently only one global option is available for *remlreg objects*:

- `outfile =` *filename*
  By default, the estimation output produced by the `regress` procedure will be written to the default output directory, which is

  <INSTALLDIRECTORY>`\output`.

  The default file name is composed of the name of the *remlreg object* and the type of the file. For example, if you estimated a nonparametric effect for a covariate `X`, say, using P-spline then the estimation output will be written to

<INSTALLDIRECTORY>`\output\r_f_X_pspline.res`

where `r` is the name of the *remlreg object*. In most cases, however, it may be necessary to save estimation results into a different directory and/or under a different file name than the default. This can be done using the `outfile` option. With the outfile option you have to specify the directory where the output should be stored to and in addition a base file name. The base file name should not be a complete file name. For example specifying

```
outfile = c:\data\res1
```

would force *BayesX* to store the estimation result for the nonparametric effect of `X` in file

`c:\data\res1_f_X_pspline.res`

8.3 Visualizing estimation results

Visualization of estimation results is described in chapter 9

8.4 References

Fahrmeir, L., Kneib, T. and Lang, S. (2004): Penalized structured additive regression for space-time data: A Bayesian perspective. *Statistica Sinica*, 14, 715-745.

Fahrmeir, L. and Tutz, G. (2001): *Multivariate Statistical Modelling based on Generalized Linear Models.* New York: Springer–Verlag.

Green, P.J. and Silverman, B. (1994): *Nonparametric Regression and Generalized Linear Models.* Chapman and Hall, London.

Hastie, T. and Tibshirani, R. (1990): *Generalized additive models.* Chapman and Hall, London.

Hastie, T. and Tibshirani, R. (1993): Varying-coefficient Models. *Journal of the Royal Statistical Society B* , 55, 757-796.

Hastie, T., Tisbshirani, R. and Friedman, J. (2001): *The Elements of Statistical Learning: Data Mining, Inference and Prediction.* New York: Sprigner–Verlag.

Kneib, T. and Fahrmeir, L. (2004a): Structured additive regression for multicategorical space-time data: A mixed model approach. SFB 386 discussion Paper 377, University of Munich. Available from www.stat.uni-muenchen.de/∼kneib/papers.html

Kneib, T. and Fahrmeir, L. (2004b): A mixed model approach for structured hazard regression. SFB 386 discussion Paper 400, University of Munich. Available from www.stat.uni-muenchen.de/∼kneib/papers.html

Brezger, A. (2000): *Bayesianische P-splines.* Master thesis, University of Munich.

Lin, X. and Zhang, D. (1999): Inference in generalized additive mixed models by using smoothing splines. *Journal of the Royal Statistical Society B*, 61, 381–400.

McCullagh, P. and Nelder, J.A. (1989): *Generalized Linear Models.* Chapman and Hall, London.

Chapter 9

Visualizing estimation results

In this chapter we show, how estimation results produced with one of the regression tools described in the two previous sections can be visualized. In general, there are two possibilities to visualize results: Within the Java based version, special functions can be applied to regression objects. Since both regression tools provide almost the same possibilities to visualize results, we describe them simultaneously in the next section. Tools for the visualization of autocorrelations for MCMC samples are described in subsection 9.1.3. An alternative way to visualize results is to use the S-Plus functions shipped together with BayesX. These functions are described in section 9.2.

9.1 BayesX functions

The Java based version allows to visualize estimation results directly in *BayesX* and immediately after estimation. The *output window* and/or the log file describes how to visualize the estimated effects for a particular model term. Nonlinear effects of continuous covariates and time scales are plotted with method `plotnonp`. Spatial effects are visualized with method `drawmap`. When using *bayesreg objects*, autocorrelation functions of sampled parameters can be visualized with method `plotautocor`.

9.1.1 Method plotnonp

Description

Method `plotnonp` is a post estimation command, i.e. it is meaningful only if method `regress` has been applied before. The method allows to plot estimated effects of nonlinear covariate effects immediately after estimation. This command is available only in the Java based version.

Syntax

> *objectname*.`plotnonp` *termnumber* [, *options*]

Plots the estimated effect with term number *termnumber*. The term number will be printed in the *output window* and/or an open log file. Several options are available for labelling axis, adding a title, etc., see the options list below. Note that method `plotnonp` can be applied only if random walks, P-splines or seasonal components are used as priors.

Options

The following options are available for method `plotnonp` (listed in alphabetical order):

- `connect=1|2|3|4|5`[*specifications for further variables*]

  Option `connect` specifies how points in the scatterplot are connected. There are currently 5 different specifications:

`1`	draw straight lines between the points (default)
`2, 3, 4`	draw dashed lines (numbers 2-4 indicate different variants)
`5`	do not connect, i.e. plot points only

  If you draw more than one scatterplot in the same graph (i.e. more than one *yvar* is specified) you can connect points for every *yvar* differently by simply specifying the corresponding number (`1,2,3,4,5`) for every *yvar*. Typing for example

  `connect=15`

  connects the points corresponding to *yvar1* and *xvar* by straight lines, but does not connect the points corresponding to *yvar2* (if specified) and *xvar*. Points corresponding to additionally specified variables *yvar*3, etc. are connected by straight lines.

  An equivalent way of specifying the different variants is available via the symbols 'l', 'd', '_', '-' and 'p', which correspond to the numbers 1-5, i.e.

  `connect=12345` is equivalent to `connect=ld_-p`

- `fontsize =` *integer*

  Specifies the font size (in pixels) for labelling axes etc. Note that the title is scaled accordingly. The default is `fontsize=12`.

- `height =` *integer*

  Specifies the height (in pixels) of the graph. The default is `height=210`.

- `levels = all |1|2|none`

  By default, `plotnonp` plots the estimated nonlinear covariate effect together with the pointwise credible intervals based on nominal levels of 80% and 95% (the nominal levels may be changed using the options level1 and/or level2). Option `levels` allows to omit completely pointwise credible intervals in the graphs (`levels=none`), print only the 95% credible intervals (`levels=1`) or to print only the 80% credible intervals (`levels=2`).

- `linecolor = B|b|c|G|g|o|m|r|y` [*specifications for further variables*]

  Option `linecolor` specifies the color to be used for drawing lines (or points, see option `connect`) in the scatterplot. Currently the following specifications are available:

  `B` black (default)
  `b` blue
  `c` cyan
  `G` gray
  `g` green
  `o` orange
  `m` magenta
  `r` red
  `y` yellow

  If you draw more than one scatterplot in the same graph (i.e. more than one *yvar* is specified) you can use different colors for each *yvar* by simply specifying the corresponding symbol (`B,b,c,G,g,o,m,r,y`) for each *yvar*. Typing for example

  ```
  linecolor = Bgr
  ```

  colors the lines (points) corresponding to *yvar1* and *xvar* in black, whereas the points corresponding to *yvar2* and *yvar3* (if specified) and *xvar* are colored in green and red, respectively.

- `linewidth =` *integer*

  Specifies how thick lines should be drawn. The default is `linewidth=5`.

- `outfile =` *characterstring*

  If option `outfile` is specified the graph will be stored as a postscript file rather than being printed on the screen. The path and the filename must be specified in *characterstring*. By default, an error will be raised if the specified file is already existing or the specified folder is not existing. To overwrite an already existing file, option `replace` must be additionally specified. This prevents you from unintentionally overwriting your files.

- `pointsize =` *integer*

  Specifies the size of the points (in pixels) if drawing points rather than lines is specified. The default is `pointsize=20`.

- `replace`

  The `replace` option is useful only in combination with option `outfile`. Specifying `replace` as an additional option allows the program to overwrite an already existing file (specified in `outfile`), otherwise an error will be raised.

- `title =` *characterstring*

  Adds a title to the graph. If the title contains more than one word, *characterstring* must be enclosed by quotation marks (e.g. `title="my first title"`).

- `titlesize =` *realvalue*

  Specifies the factor by which the size of the title is scaled relative to the size of the labels of the axes (compare option `fontsize`). The default is `titlesize=1.5`.

- `width =` *integer*

  Specifies the width (in pixels) of the graph. The default is `width=356`.

- `xlab =` *characterstring*

  Labels the x-axis. If the label contains more than one word, *characterstring* must be enclosed by quotation marks (e.g. `xlab="x axis"`).

- `xlimbottom =` *realvalue*

  Specifies the minimum value at the x-axis to be plotted. The default is the minimum value in the data set. If `xlimbottom` is above the minimum value in the data set, only a part of the graph will be visible.

- `xlimtop =` *realvalue*

  Specifies the maximum value at the x-axis to be plotted. The default is the maximum value in the data set. If `xlimtop` is below the maximum value in the data set, only a part of the graph will be visible.

- `xstart =` *realvalue*

  Specifies the value where the first 'tick' on the x-axis should be drawn. The default is the minimum value on the x-axis.

- `xstep =` *realvalue*

  If `xstep` is specified, ticks are drawn at the x-axis with stepwidth *realvalue* starting at the minimum value on the x-axis (or at the value specified in option `xstart`). By default, five equally spaced ticks are drawn at the x-axis.

- `ylab =` *characterstring*

  Labels the y-axis. If the label contains more than one word, *characterstring* must be enclosed by quotation marks (e.g. `ylab="y axis"`).

- `ylimbottom =` *realvalue*

  Specifies the minimum value at the y-axis to be plotted. The default is the minimum value in the data set. If `ylimbottom` is above the minimum value in the data set, only a part of the graph will be visible.

- `ylimtop =` *realvalue*

  Specifies the maximum value at the y-axis to be plotted. The default is the maximum value in the data set. If `ylimtop` is below the maximum value in the data set, only a part of the graph will be visible.

- `ystart =` *realvalue*

  Specifies the value where the first 'tick' on the y-axis should be drawn. The default is the minimum value on the y-axis.

- `ystep =` *realvalue*

  If `ystep` is specified, ticks are drawn at the y-axis with stepwidth *realvalue* starting at the minimum value on the y-axis (or at the value specified in option `ystart`). By default, five equally spaced ticks are drawn at the y-axis.

Examples

Suppose we have already created a regression object `reg` and have estimated a regression model with Gaussian errors using a command like

```
> reg.regress Y = X(psplinerw2), family=gaussian using d
```

where `Y` is the response variable and `X` the only explanatory variable. The effect of `X` is modelled nonparametrically using Bayesian P-splines. In the *output window* we obtain the following estimation output for the effect of `X`:

```
f_x

Results are stored in file
c:\results\reg_f_x_pspline.res
Results may be visualized using method plotnonp
Type for example: objectname.plotnonp 0
```

The term number of the effect of X is 0, i.e. by typing

```
> reg.plotnonp 0
```

we obtain the plot shown in Figure 9.1.

Of course, a title, axis labels etc. can be added. For example by typing

```
> reg.plotnonp 0 , title="my title" xlab="x axis"
```

we obtain the plot shown in Figure 9.2.

By default, the plots appear in an additional window on the screen. They can be directly stored in postscript format by adding option `outfile`. For example by typing

```
> reg.plotnonp 0 , title="my title" xlab="x axis" outfile="c:\results\result1.ps"
```

the graph is stored in postscript format in the file `c:\results\result1.ps`.

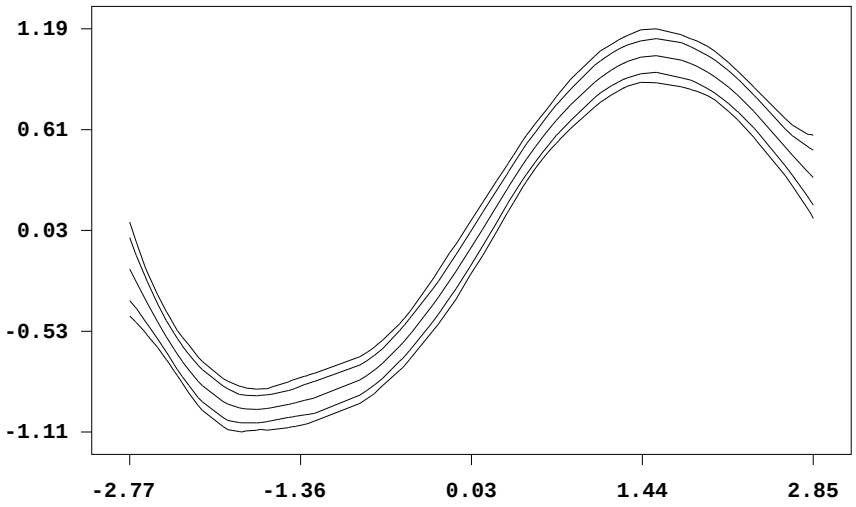


Figure 9.1: Illustration for the usage of method `plotnonp`

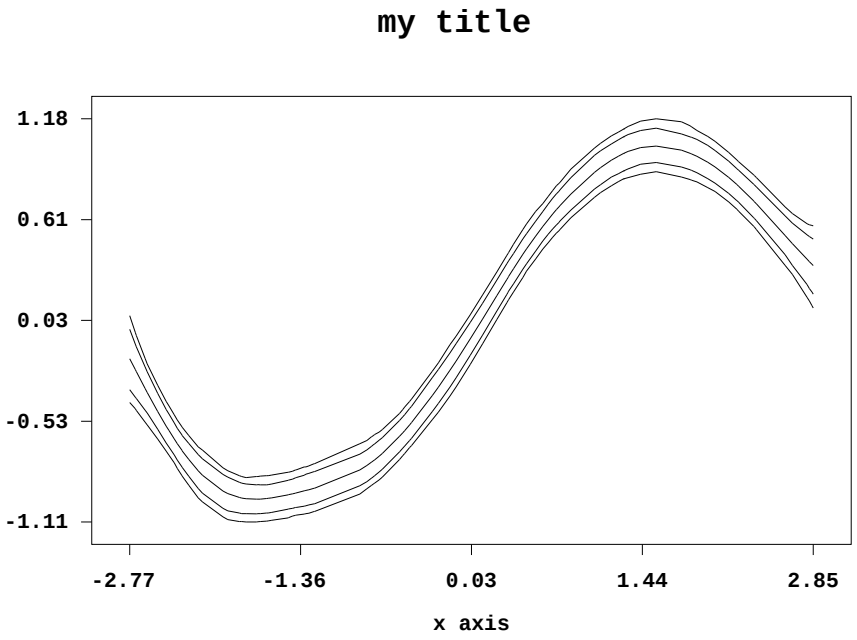


Figure 9.2: Second illustration for the usage of method `plotnonp`

9.1.2 Method drawmap

Description

Method `drawmap` is a post estimation command, i.e. it is meaningful only if method `regress` has been applied before. The method allows to visualize estimated effects of spatial covariates immediately after estimation. This command is available only in the Java based version.

Syntax

> *objectname*.`drawmap` *termnumber* [, *options*]

Visualizes the effect of a spatial covariate by coloring the regions of the corresponding geographical map according to the estimated effect (or other characteristics of the posterior). The term number *termnumber* identifies the model term and can be found in the *output window* and/or an open log file. Several options are available for adding a title or changing the color scale etc., see the options list below. Note that method `drawmap` can be applied only if Markov random fields, geosplines or geokriging are used as priors.

Options

The following options are available for method `drawmap` (in alphabetical order):

- `color`

  The `color` option allows to choose between a grey scale for the colors and a colored scale. If `color` is specified a colored scale is used instead of a grey scale.

- `drawnames`

  In some situations it may be useful to print the names of the regions into the graph (although the result may be confusing in most cases). This can be done by specifying the additional option `drawnames`. By default the names of the regions are omitted in the map.

- `nolegend`

  By default a legend is drawn into the graph. By specifying the option `nolegend` the legend will be omitted.

- `lowerlimit =` *realvalue*

  Lower limit of the range to be drawn. If `lowerlimit` is omitted, the minimum numerical value in `plotvar` will be used instead as the lower limit.

- `outfile =` *characterstring*

  If option `outfile` is specified the graph will be stored as a postscript file rather than being printed on the screen. The path and the filename must be specified in *characterstring*. By default, an error will be raised if the specified file is already existing or the specified folder is not existing. To overwrite an already existing file, option `replace` must be additionally specified. This prevents you from unintentionally overwriting your files.

- `plotvar =` *variablename*

  By default, the regions of the map are colored according to the estimated spatial effect. Option `plotvar` allows to color the map according to other characteristics of the posterior by explicitly specifying the name of the variable to be plotted. Compare the header of the file containing the estimation results to see all variables available for plotting.

- `replace`

  The `replace` option is only useful in combination with option `outfile`. Specifying `replace` as an additional option allows the program to overwrite an already existing file (specified in `outfile`), otherwise an error will be raised.

- `nrcolors =` *integer*

  To color the regions according to their numerical characteristics, the data are divided into a (typically large) number of ordered categories. Afterwards a color is associated with each category. The `nrcolors` option can be used to specify the number of categories (and with it the number of different colors). The maximum number of colors is 256, which is also the default value.

- `swapcolors`

  In some situations it may be favorable to swap the order of the colors, i.e. black (red) shades corresponding to large values and white (green) shades corresponding to small values. This is achieved by specifying `swapcolors`. By default, small values are colored in black shades (red shades) and large values in white shades (green shades).

- `title =` *characterstring*

  Adds a title to the graph. If the title contains more than one word, *characterstring* must be enclosed by quotation marks (e.g. `title="my first map"`).

- `upperlimit =` *realvalue*

  Upper limit of the range to be plotted. If `upperlimit` is omitted, the maximum numerical value in `plotvar` will be used instead as the upper limit.

- `pcat`

  If you want to visualize the values of the columns `pcat80` or `pcat95` it is convenient to specify `pcat`. This forces `drawmap` to expect a column that consists only of the values -1, 0 and 1. Of course you can achieve the same result by setting `nrcolors=3`, `lowerlimit=-1` and `upperlimit=1`.

Examples

Suppose we have already created a regression object `reg` and have estimated a regression model with Gaussian errors using something like

```
> map m
> m.infile using c:\maps\map1.bnd
> reg.regress Y = region(spatial,map=m), family=gaussian using d
```

where `Y` is the response variable and `region` the only explanatory variable. The effect of the spatial covariate `region` is modelled nonparametrically using a Markov random field. In the *output window* we obtain the following estimation output for the effect of `region`:

```
f_spat_region

Results are stored in file
c:\results\reg_f_region_spatial.res
Results may be visualized using method 'drawmap'
Type for example: objectname.drawmap 0
```

The term number of the effect of `region` is 0, i.e. by typing

```
> reg.drawmap 0
```

we obtain the map shown in Figure 9.3 where the regions are colored according to the estimated spatial effect.

By default the regions are colored in grey scale. A color scale is obtained by adding option `color`. A title can be added as well. For example by typing

```
> reg.drawmap 0 , color title="my title"
```

we obtain the map shown in Figure 9.4.

By default, the maps appear in an additional window on the screen. They can be directly stored in postscript format by adding option `outfile`. For example by typing

```
> reg.drawmap 0 , color title="my title" outfile="c:\results\result1.ps"
```

the colored map is stored in postscript format in the file `c:\results\result1.ps`.

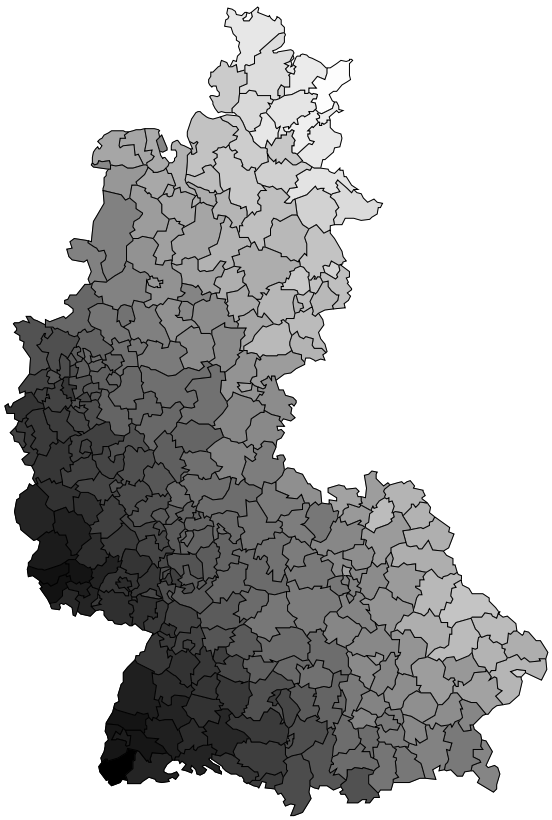


Figure 9.3: Illustration for the usage of method `drawmap`

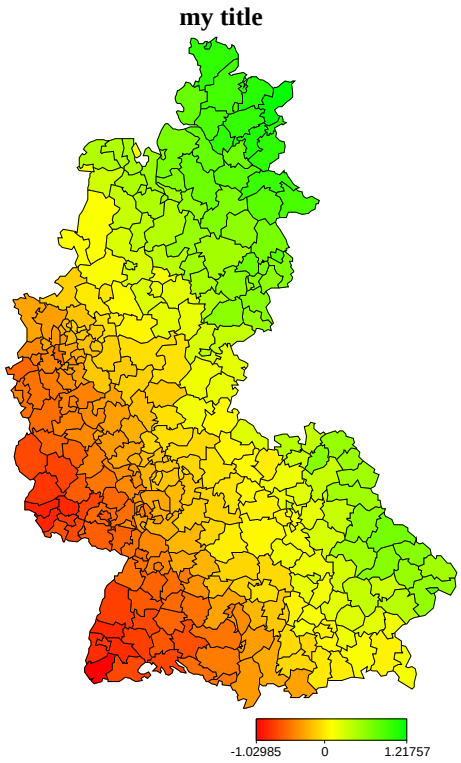


Figure 9.4: Second illustration for the usage of method `drawmap`

9.1.3 Method plotautocor

Description

Method `plotautocor` is a post estimation command, i.e. it is meaningful only if method `regress` has been applied before. Method `plotautocor` computes and visualizes the autocorrelation functions of the parameters in the model. This method is only applicable to *bayesreg objects*.

Syntax

> *objectname*.`plotautocor` [, *options*]

Computes and visualizes the autocorrelation functions in the model. Several options are available for specifying the maximum lag for autocorrelations, storing the graphs in postscript format etc., see the options list below.

Options

The following options are available for method `plotautocor` (in alphabetical order):

- `maxlag =` *integer*

  Option `maxlag` may be used to specify the maximum lag for autocorrelations. The default is `maxlag=250`.

- `mean`

  If option `mean` is specified, for each lag number and model term only minimum, mean and maximum autocorrelations are plotted. This can lead to a considerable reduction in computing time and storing size.

- `outfile =` *characterstring*

  If option `outfile` is specified the graph will be stored as a postscript file and not printed on the screen. The path and the filename must be specified in *characterstring*. An error will be raised if the specified file is already existing and the `replace` option is not specified.

- `replace`

  The `replace` option is only useful in combination with option `outfile`. Specifying `replace` as an additional option allows the program to overwrite an already existing file (specified in `outfile`), otherwise an error will be raised.

Examples

Suppose we have already created a *bayesreg object* `reg` and have estimated a regression model with Gaussian errors using

```
> reg.regress Y = X(psplinerw2), family=gaussian using d
```

where `Y` is the response variable and `X` the only explanatory variable. The effect of `X` is modelled nonparametrically using Bayesian P-splines. We can now check the mixing of sampled parameters by computing and drawing the autocorrelation functions up to a maximum lag of 150:

```
> reg.plotautocor , maxlag=150 outfile="c:\results\autocor.ps"
```

In this example the autocorrelation functions are not shown on the screen but stored in postscript format in the file `c:\results\autocor.ps`. If option `outfile` is omitted, the functions are plotted on the screen. The resulting file contains 5 pages. As an example, the first page of the file is shown in Figure 9.5.

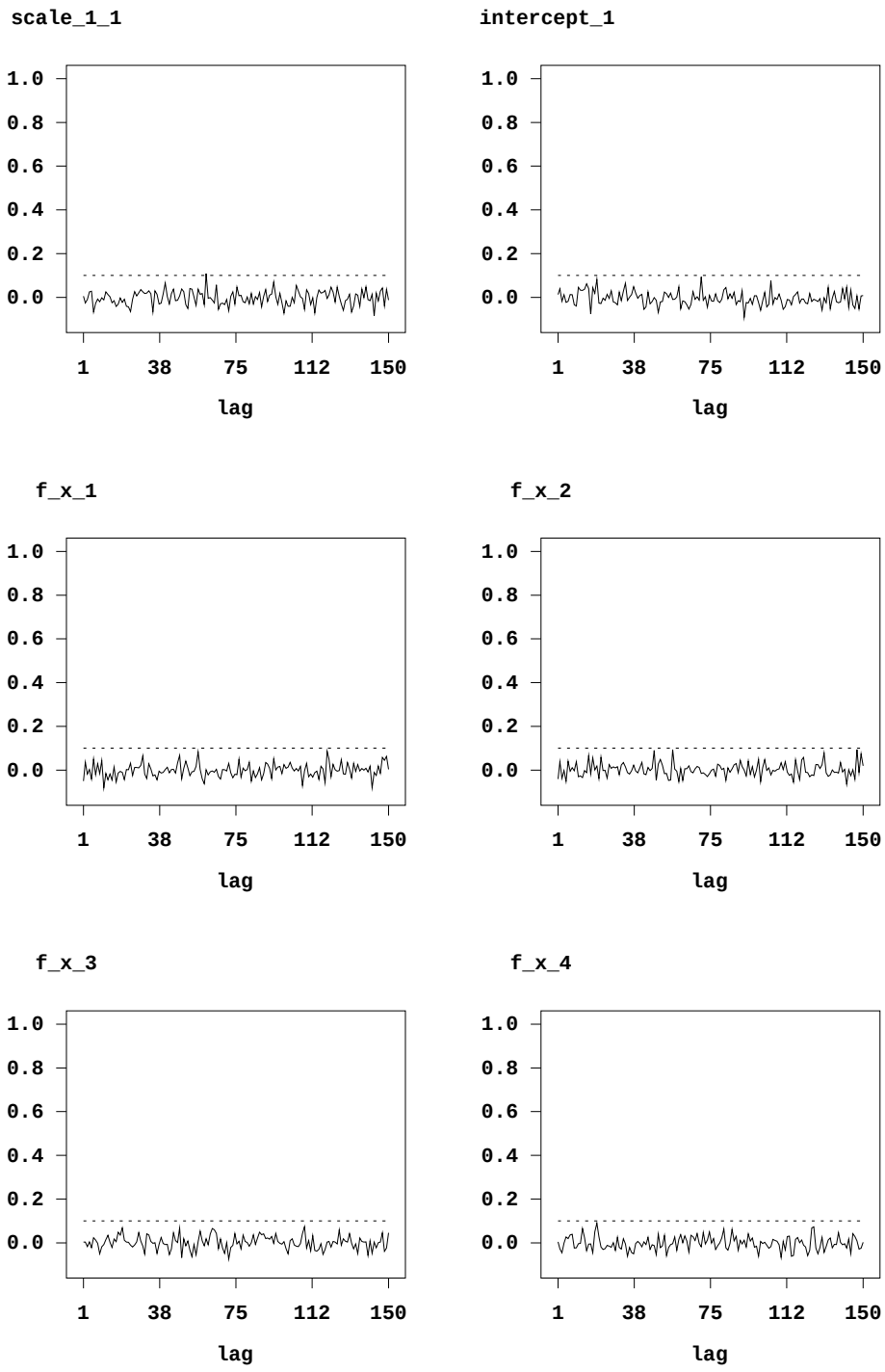


Figure 9.5: Illustration for the usage of method `plotautocor`

9.2 S-plus functions

Since only the Java based version of *BayesX* provides capabilities for visualizing estimation results, some S-plus functions for plotting estimated functions are shipped together with *BayesX*. These functions can be found in the subdirectory `sfunctions` of the installation directory. Table 9.1 gives a first overview over the different functions and their abilities. The usage of the functions is very simple so that also users not familiar with the S-plus environment should be able to apply the functions without any difficulties. The following subsections describe how to install the functions in S-plus and give a detailed description of the usage of the respective functions.

Functionname	**Description**
`plotnonp`	visualizes estimated nonparametric functions
`plotautocor`	visualizes autocorrelation functions
`plotsample`	visualizes sampling paths of sampled parameters
`readbndfile`	reads in boundaries of geographical maps
`drawmap`	visualizes estimation results for spatial covariates
`plotsurf`	visualizes estimated 2 dimensional surfaces

Table 9.1: Overview over S-plus functions

9.2.1 Installation of the functions

Installation of the different functions is very easy. The S-plus code for the functions is stored in the directory <*INSTALLDIRECTORY*>`\sfunctions` in the ASCII text file `plot.s`. To install the functions you first have to start S-plus. Afterwards the functions will be installed by entering

`> source("`<*INSTALLDIRECTORY*>`\\sfunctions\\plot.s")`

in the *Commands Window* of S-plus. Note that a double backslash is required in S-plus to specify a directory correctly. For use with the R package the file `plot.r` is supplied, which contains slightly modified versions of the S-plus functions. Note that the `plotsurf` function is not available for R.

9.2.2 Plotting nonparametric functions

This subsection describes the usage of the function `plotnonp` for visualizing nonparametric function estimates.

Suppose that a Bayesian regression model has already been estimated with predictor

$$\eta = \ldots + f(X) + \ldots,$$

where the effect of `X` is modelled nonparametrically using for example a first or second order random walk prior. Unless the directory for estimation output has been changed using the global option `outfile` (see section 7.4 and section 8.2), estimation results for the nonparametric effect of `X` are stored in the directory

<*INSTALLDIRECTORY*>`\output`

that is in the subdirectory `output` of the installation directory. The filename is

objectname`_f_X_rw.res`

that is it is composed of the name of the regression object and the covariate name. For the following we assume that `c:\bayes` is the installation directory and `reg` is the name of the regression object. In this case results for the effect of `X` are stored in:

`c:\bayes\output\reg_f_X_rw.res`

The structure of such a file has already been described in section 7.1 (*bayesreg objects*) and section 8.1 (*remlreg objects*). Although it is possible (and very easy) to visualize the estimated nonparametric function with any software package that has plotting capabilities, a fast and easy way of plotting estimation results without knowing the particular structure of the results-file is desirable. This is the task of the S-plus function `plotnonp`.

The function has only one required and many optional arguments. The required argument is the directory and the filename where nonparametric estimation results are stored. For example by entering the command

```
> plotnonp("c:\\bayes\\output\\reg_f_X_rw.res")
```

a S-plus *graphic window* will be opened with the plotted function estimate. The function plots the estimated effect together with the posterior 2.5%, 10%, 90% and 97.5% quantiles. One advantage of the function is that after its application no permanent objects will remain in the S-plus environment.

Besides the required argument a lot of optional arguments may be passed to the function. Among others there are options for plotting the graphs in a postscript file rather than the screen, labelling the axes, specifying the minimum/maximum value on the x/y axes and so on. The following optional arguments can be passed to `plotnonp`:

- `psname = "`*filename (including path)*`"`
  Name of the postscript output file. If `psname` is specified the graph will be stored in a postscript file and will not appear on the screen.
- `level = 0/1/2`
  Specifies whether to plot only the 95% credible intervals (`level=1`) or only the 80% credible intervals (`level=2`). Default value is `level=0`, i.e. both.
- `ylimtop =` *realvalue*
  Specifies the maximum value on the y-axis (vertical axis).
- `ylimbottom =` *realvalue*
  Specifies the minimum value on the y-axis.
- `xlab = "`*characterstring*`"`
  `xlab` is used to label the x-axis (horizontal axis).
- `ylab = "`*characterstring*`"`
  `ylab` is used to label the y-axis.
- `maintitle = "`*characterstring*`"`
  Adds a title to the graph.
- `subtitle = "`*characterstring*`"`
  Adds a subtitle to the graph.
- `linecol =` *integer*
  Specifies the color of the credible intervals. Default value is `linecol=3`.
- `linetype =` *integer*
  Specifies the line type for the credible intervals. Default value is `linetype=1` (solid).

As an illustration compare the following S-plus statement:

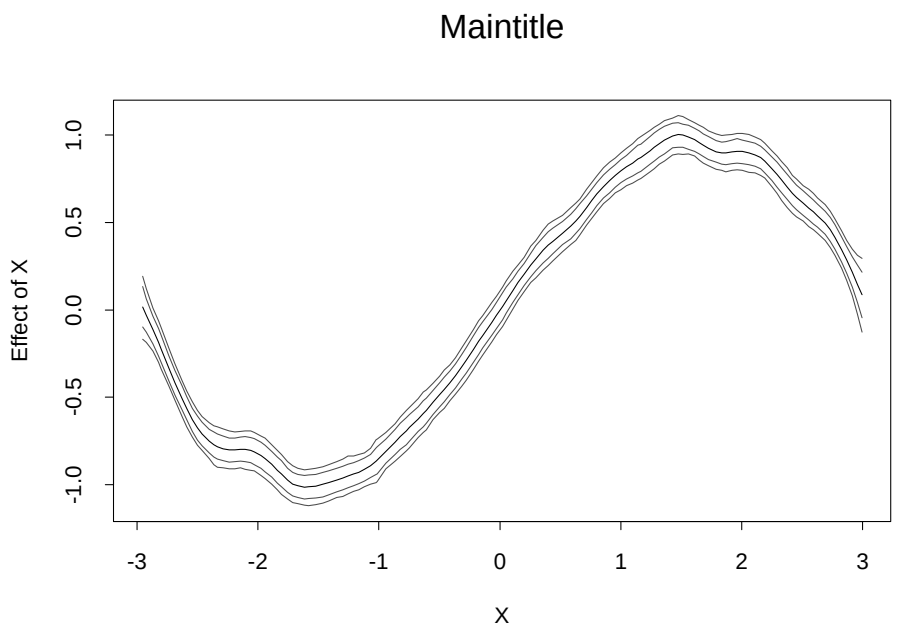


Figure 9.6: Illustration for the usage of `plotnonp`

```
> plotnonp("c:\\bayes\\reg_f_X_rw.res", psname="c:\\bayes\\reg_f_X_rw.ps",
  maintitle="Maintitle",ylab="effect of X",xlab="X")
```

This statement draws the estimated effect of `X` and stores the graph in the postscript file `"c:\\bayes\\reg_f_X_rw.ps"`. A title, a x-axis and y-axis label are added to the graph. For illustration purposes, the resulting graph is shown in Figure 9.6.

In some situations the effect of a covariate representing dates must be plotted. Suppose for example that a covariate has values ranging from 1 to 19 representing the time period from January 1983 to July 1984. In this case, we naturally prefer that the x-axis is labelled in terms of dates rather than in the original coding (from 1 to 19). To achieve this, function `plotnonp` provides the three additional options `year`, `month` and `step`. Options `year` and `month` are used to specify the year and the month (1 for January, 2 for February, ...) corresponding to the minimum covariate value. In the example mentioned above `year=1983` and `month=1` will produce the correct result. In addition, option `step` may be specified to define the periodicity in which your data are collected. For example `step=12` (the default) corresponds to monthly data, while `step=4`, `step=2` and `step=1` correspond to quarterly, half yearly and yearly data. We illustrate the usage of `year`, `month` and `step` with our example. Suppose we estimated the effect of calendar time `D`, say, on a certain dependent variable, where the range of the data is as described above. Then the following S-plus function call will produce the postscript file shown in Figure 9.7:

```
> plotnonp("c:\\bayes\\reg_f_D_pspline.res", psname="c:\\bayes\\reg_f_D_pspline.ps",
  year=1983,month=1,step=12,xlab="date", ylab= " ")
```

Note, that `ylab=" "` forces S-plus to omit the y axis label. If `ylab` (as well as `xlab`) is omitted, default labels will be given to the two axis.

Finally, we note that all options that can be passed to the `plot` function of S-plus may also be passed to function `plotnonp`. Thus, function `plotnonp` is more or less a specialized version of the S-plus `plot` function.

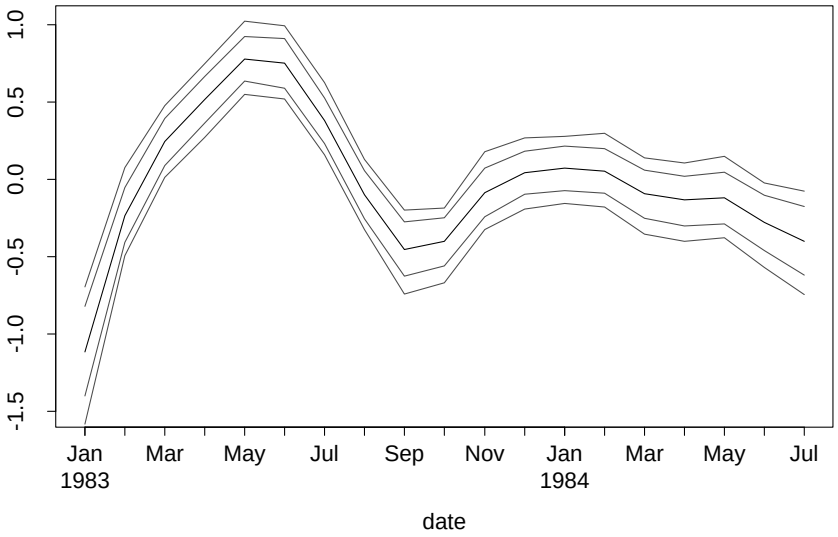


Figure 9.7: Illustration for the usage of `plotnonp`

Figure 9.8: Map of Munich

9.2.3 Drawing geographical maps

This subsection describes how to visualize estimation results of spatial covariates, where the observations represent the location or site in connected geographical regions. A typical example for a spatial covariate is given in the 'rents for flats' example, see subsection 2.6.1, where the covariate `L` indicates the location (in subquarters) of the flat in Munich. Figure 9.8 shows a map of Munich separated into subquarters.

Typically, the effect of such a spatial covariate is incorporated into a regression model via an unstructured or structured random effect. In the latter case a spatial smoothness prior for the spatial covariate is specified that penalizes too abrupt changes of the estimated effect in neighboring sites. In some situations the incorporation of both, an unstructured and a structured effect, may also be appropriate. Details on how to incorporate spatial covariates into a semiparametric regression model are given in section 7.1 (*bayesreg objects*) and section 8.1 (*remlreg objects*). For the rest of this section we assume that an effect of a spatial covariate has already been estimated and that we now want to visualize the estimation results. This can be easily done with the two S-

plus functions `readbndfile` and `drawmap`. Function `readbndfile` is used to read the boundary information of a map that is stored in a so called boundary file and to store this information as a permanent S-plus *map object*. The boundary file contains mainly the polygons which form the different geographical regions of the map. The required structure of such a file is described below. After the successful reading of the boundary information of a map, the second function `drawmap` may be used to draw and print the map either on the screen or into a postscript file. There are several possible ways to draw the map. In the simplest case the map can be drawn without any estimation effects, i.e. only the boundaries of the different regions or sites are drawn, see Figure 9.8 for an example. In practice, however, one usually wants to color the regions of the map according to some numerical characteristics. As an example compare Figure 9.9 in which the subquarters of Munich are colored according to the frequency of flats in the 'rents for flats' data set located in the respective subquarter. Subquarters colored in red contain less flats compared to subquarters colored in green. In striped areas no observations are available.

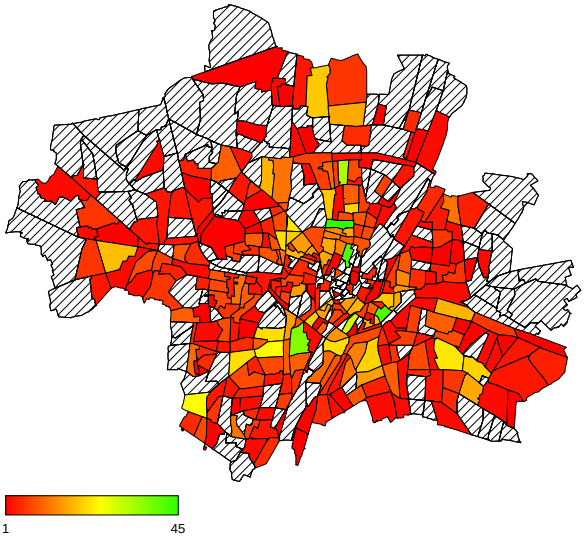


Figure 9.9: relative frequencies of observed flats in the 'rents for flats' data set

In the following we give a detailed description of the usage of the functions `readbndfile` and `drawmap`.

Function readbndfile

Function `readbndfile` is used to read in boundary information stored in a boundary file into S-plus. The function has two required arguments. The first argument is the filename of the boundary file to read in. The second argument specifies the name of the *map object* in S-plus (recall that the map information is stored as a permanent S-plus object). To give an example, suppose that *BayesX* is installed in the directory `c:\bayes` and that we want to read in the map of Munich. In this case the boundary file of the map is stored in the subdirectory `examples` of the installation directory, that is in `c:\bayes\examples`. The name of the boundary file is simply `munich.bnd`. The following function call reads in the boundary information of Munich and stores the map permanently in S-plus:

```
> readbndfile("c:\\bayes\\examples\\munich.bnd","munich")
```

Once again, note that double backslashes are required in S-plus to specify a directory. The second argument in the statement above is "munich", i.e. the name of the map object is simply `munich`.

To refer to the map of Munich in subsequent statements and function calls, the quotation marks must be omitted.

Function drawmap

Function `drawmap` is used to draw geographical maps and color the regions according to some numerical characteristics. There is only one required argument that must be passed to `drawmap`, that is the name of the map to be drawn. Provided that the map has already been read into S-plus (via function `readbndfile`), the following statement draws the map of Munich in a S-plus graphic-window on the screen:

```
> drawmap(map=munich)
```

Storing the map in a postscript file rather than drawing it on the screen can be achieved by specifying the name of the postscript file using the `outfile` option. For example the command

```
> drawmap(map=munich,outfile="c:\\bayes\\munich.ps")
```

produces a postscript file named `munich.ps` with the map of Munich.

However, in most cases one not only wants to draw the boundaries of a geographical map, but also color the regions according to some numerical characteristics. Suppose for example that we have already estimated a location specific effect on the monthly rents in the 'rents for flats' data set. Suppose further that the estimated effects are stored in `c:\bayes\output\reg_f_L_spatial.res`. The structure of the file is described in detail in section 7.1 (*bayesreg objects*) and section 8.1 (*remlreg objects*).

Suppose now that we want to visualize estimation results for the spatial covariate `L` by coloring the subquarters of Munich according to the posterior mean estimated using a *bayesreg object*. Compared to the S-plus statement above, (at least) three more arguments must be passed to function `drawmap`; the argument `dfile` that specifies the filename of estimated results, the argument `plotvar` that specifies the variable to be plotted and the argument `regionvar` that specifies which column of the file, containing estimation results, stores the region names. The following statement produces the desired result:

```
> drawmap(map=munich,outfile="c:\\bayes\\munich.ps",
  dfile="c:\\bayes\\output\\reg_f_L_spatial.res", plotvar="pmean",regionvar="L")
```

Note that the right hand side of options `plotvar` and `regionvar` must be enclosed by quotation marks. If we want to color the subquarters of Munich according to the posterior mode estimated using a *remlreg object* we have to change the statement to

```
> drawmap(map=munich,outfile="c:\\bayes\\munich.ps",
  dfile="c:\\bayes\\output\\reg_f_L_spatial.res", plotvar="pmode",regionvar="L")
```

Optional arguments of function drawmap

Besides the arguments discussed so far there are some more optional arguments that can be passed to `drawmap`. They are listed and described below together with a summary of the arguments already described:

- `map = "`*characterstring*`"`

  Name of the S-plus *map object*. Use function `readbndfile` to read in geographical maps into S-plus.

- `dfile = "`*filename (including path)*`"`

Filename (including path) of the file containing numerical characteristics of the regions of the map. The file must contain at least two columns, one column that lists the names of the regions and one column containing the numerical characteristics of the respective regions. It is important that the names of the regions listed match with the region names stored in the S-plus *map object*. The first row of the file must contain the names of the columns.

- `outfile = "`*filename (including path)*`"`

  Filename (including path) of the postscript file where the map should be stored.

- `regionvar = "`*characterstring*`"`

  Name of the column in the data file containing the region names (see also argument `dfile`). Note that the right hand side must be enclosed by quotation marks.

- `plotvar = "`*characterstring*`"`

  Name of the column in the data file containing the numerical characteristics of the regions (see also argument `dfile`). Note that the right hand side must be enclosed by quotation marks.

- `lowerlimit =` *realvalue*

  Lower limit of the range to be drawn. If `lowerlimit` is omitted, the minimum numerical value in the `plotvar` column will be used instead as the lower limit.

- `upperlimit =` *realvalue*

  Upper limit of the range to be drawn. If `upperlimit` is omitted, the maximum numerical value in the `plotvar` column will be used instead as the upper limit.

- `nrcolors =` *integer*

  To color the regions according to their numerical characteristics, the data are divided into a (typically large) number of ordered categories. Afterwards a color is associated with each category. The `nrcolors` option can be used to specify the number of categories (and with it the number of different colors). Default value is 100.

- `pstitle = "`*characterstring*`"`

  Adds a title to the graph. Note that the right hand side must be enclosed by quotation marks.

- `color = T/F`

  The `color` option allows to choose between a grey scale for the colors and a colored scale. The default is `color=F`, which means a grey scale.

- `legend = T/F`

  By default a legend is drawn into the graph. To omit the legend in the graph, `legend=F` must be passed as an additional argument.

- `drawnames = T/F`

  In some situations it may be favorable to print the names of the regions into the graph (although the result may be confusing in most cases). This can be done by specifying the additional option `drawnames=T`. By default the names of the regions are omitted in the graph.

- `swapcolors = T/F`

  In some situations it may be favorable to swap the order of the colors, i.e. red shades corresponding to large values and green shades corresponding to small values. This is achieved by specifying `swapcolors=T`. By default small values are colored in red shades and large values in green shades.

- `pcat = T/F`

  If you want to visualize the values of the columns `pcat80` or `pcat95` it is convenient to specify `pcat=T`. This forces `drawmap` to expect a column that consists only of the values -1, 0 and 1. Of course you can achieve the same result by setting `nrcolors=3`, `lowerlimit=-1` and `upperlimit=1`. The default is `pcat=F`.

9.2.4 Plotting 2 dimensional surfaces

This subsection describes the usage of the function `plotsurf` for visualizing 2 dimensional surfaces. The function `plotsurf` merely invokes different S-plus functions for visualizing 2 dimensional data. Thus, users familiar with S-plus may prefer to use this functions directly to gain more flexibility. Note that this function is only available for S-plus and not for R.

Suppose that a Bayesian regression model has already been estimated with predictor

$$\eta = \ldots + f(X1, X2) + \ldots,$$

where the interaction effect of `X1` and `X2` is modelled nonparametrically using 2 dimensional P-splines and that the estimation results are stored in file:

`c:\bayes\output\reg_f_X1_X2_pspline.res`

The S-plus function `plotsurf` requires at least one argument, which is the name (including path) of the file containing the estimation results. For example the command

```
> plotsurf("c:\\bayes\\output\\reg_f_X1_X2_pspline.res")
```

plots the estimated surface against `X1` and `X2` on the screen. There are several additional options that can be passed, for example for changing the plot type or storing the graph as a postscript file rather than displaying it on the screen. The following list describes all possible arguments that may be passed to the function `plotsurf`:

- `data = "`*filename (including path)*`"`

  Name (including path) of the file containing the estimation results. The file must contain at least 3 columns, one for the x-axis, one for the y-axis and one for the z-axis. The file must contain a header.

- `outfile = "`*filename (including path)*`"`

  Name (including path) of the postscript file where the graph should be stored. This option is only meaningful for `mode=2` and `mode=3`.

- `cols =` *3 column vector*

  This option is only meaningful, if the argument `data` is specified. In this case `cols` gives the columns of the object or data file passed to the argument `data`, that should be used as values for the x, y and z axis. The default is `cols=c(2,3,4)` which corresponds to plotting the estimated surface against `X1` and `X2`.

- `mode = 1/2/3/4/5`

  This option specifies the plot type. Currently there are 5 different types available. The default is `mode=1`, which corresponds to what is called a 'Surface Plot' in S-plus.

9.2.5 Plotting autocorrelation functions

This section describes how to visualize autocorrelation functions of sampled parameters using the S-plus function `plotautocor`.

To compute autocorrelation functions, the post-estimation command `autocor` must be applied, see section 7.2 for details. For the rest of this section we assume that autocorrelations are already computed and stored in file:

`c:\bayes\output\reg_autocor.raw`

The minimum number of arguments required for the function is one, namely the file where the computed autocorrelation functions are stored. In this case a S-plus *graphic window* will be opened and the autocorrelation functions are plotted on the screen. To store autocorrelations in a postscript file, an output filename must be specified as a second argument. Thus, the S-plus command

```
> plotautocor("c:\\bayes\\output\\reg_autocor.raw")
```

prints autocorrelations on the screen, while the statement

```
> plotautocor("c:\\bayes\\output\\reg_autocor.raw","c:\\bayes\\output\\reg_autocor.ps")
```

forces S-plus to store the autocorrelation graphs in the postscript file `c:\bayes\output\reg_autocor.ps`.

In particular for regression models with a large number of parameters the execution of function `plotautocor` can be very time consuming. Moreover, the size of the resulting postscript file can be very large. To avoid such problems `plotautocor` provides the additional argument `mean.autocor`. If `mean.autocor=T` is specified, for each lag number and model term only minimum, mean and maximum autocorrelations are plotted, leading in most cases to a considerable reduction in computing time and storing size.

9.2.6 Plotting sampled parameters

This section describes how to plot sampled parameters using the S-plus function `plotsample`. Before applying function `plotsample`, sampled parameters must be stored in ASCII-format using the post-estimation command `getsample`. See section 7.3 for details, but note that sampled parameters will be stored in several different files, typically one file for each term in the model.

Suppose now that we want to visualize sampling paths for the parameters of the nonlinear effect of a covariate X. Assume further that sampled parameters are stored in the ASCII file

`c:\bayes\output\reg_X_sample.raw`.

As most other functions, `plotsample` provides two possibilities of drawing sampled parameters. The first possibility is to print the graphs on the screen, and the second is to store them into a postscript file. To print the sampling paths on the screen, only the filename (including path) of the ASCII file where sampled parameters are stored must be passed to the function. For the example mentioned above the corresponding command is:

```
> plotsample("c:\\bayes\\output\\reg_X_sample.raw")
```

If sampling paths should be drawn into a postscript file rather than on the screen, the filename of the resulting postscript file must be specified as a second argument. Thus, for our example we get:

```
> plotsample("c:\\bayes\\output\\reg_X_sample.raw","c:\\bayes\\output\\reg_X_sample.ps")
```

In addition, all options that are available for the S-plus function `plot` may be passed to function `plotsample`, see the S-plus documentation for details.

Chapter 10

DAG Objects

Author: Eva–Maria Fronk
email: fronk@stat.uni-muenchen.de

Dag objects are needed to estimate dag models using reversible jump MCMC. The considered variables may be Gaussian or binary, even the mixed case of a conditional Gaussian distribution is possible. A general introduction into graphical models can be found in Lauritzen (1996). For a description of the more particular Gaussian dags see for instance Geiger and Heckerman (1994). We refer to Brooks (1998) or Gilks (1996) for an introduction into MCMC simulation techniques. For the more general reversible jump MCMC have a look at Green (1995); for reversible jump MCMC in context of graphical models at Giudici and Green (1999). The following explanations to the statistical background of the program can be found in more detail in Fronk and Giudici (2000).

10.1 Method estimate

10.1.1 Description

The method `estimate` estimates the dependency structure of the given variable which is represented by a dag. Furthermore, the parameters of this model are estimated. This is done within a Bayesian framework; we assume prior distributions for the unknown parameters and use MCMC techniques for estimation. In the following we first focus on the Gaussian case and describe the statistical model which is assumed for the variables. Some factorizations which result from the properties of dags are also given. To represent the dags we rely on the concept of adjacency matrices which is briefly explained and necessary to understand the output. We finally give some brief information about the used algorithm without going into details. Finally, we address the situation of binary and mixed (i.e. continuous and binary) variables, too, which is reduced to the Gaussian case by introducing latent variables.

Model Assumptions

A Gaussian dag d can be represented as a regression model for each variable X_i, $i = 0, \ldots, p-1$, given the parents of X_i, denoted by $\boldsymbol{X}_{pa(i)}$,

$$X_i \mid \boldsymbol{x}_{pa(i)}, \boldsymbol{\beta}_{i|pa(i)}, \sigma^2_{i|pa(i)}, d \sim \mathrm{N}(\beta_{i0} + \sum_{x_l \in pa(x_i)} \beta_{il} x_l, \sigma^2_{i|pa(i)}).$$

The joint distribution of all variables $\mathbf{X} = (X_0, \ldots, X_{p-1})'$ is then given by

$$p(\boldsymbol{x} \mid \boldsymbol{\beta}, \boldsymbol{\sigma}^2) = \prod_{i=0}^{p-1} p(x_i \mid \boldsymbol{x}_{pa(i)}, \boldsymbol{\beta}_{i|pa(i)}, \sigma^2_{i|pa(i)}),$$

where $\boldsymbol{\beta}_{i|pa(i)}$ is the $|pa(i)| + 1$-dimensional vector of the intercept β_{i0} and the $|pa(i)|$ regression coefficients of X_i. Furthermore, $\sigma^2_{i|pa(i)}$ is the partial variance of X_i given its parents $\boldsymbol{x}_{pa(i)}$. Let $\boldsymbol{\beta} = (\boldsymbol{\beta}'_{0|pa(1)}, \ldots, \boldsymbol{\beta}'_{p-1|pa(p)})'$ denote the vector of the $\boldsymbol{\beta}_{i|pa(i)}$'s and accordingly $\boldsymbol{\sigma}^2 = (\sigma^2_{0|pa(1)}, \ldots, \sigma^2_{p-1|pa(p)})'$ the vector of the conditional variances $\sigma^2_{i|pa(i)}$.

The vector $\boldsymbol{\beta}_{i|pa(i)}$ is assumed to be normally distributed with mean $\boldsymbol{b}_{i|pa(i)}$ and covariance matrix $\frac{1}{\alpha_i}\sigma^2_{i|pa(i)}\boldsymbol{I}$, where α_i is a known scaling factor. For the sake of simplicity, we shall assume $\alpha_i = \alpha$. Formally:

$$\boldsymbol{\beta}_{i|pa(i)} \mid \sigma^2_{i|pa(i)}, d \sim \mathrm{N}_{|pa(i)|+1}\left(\boldsymbol{b}_{i|pa(i)},\, \frac{1}{\alpha}\sigma^2_{i|pa(i)}\boldsymbol{I}\right).$$

This implies that the coefficients of a regression model are assumed to be mutually independent. For the partial variance $\sigma^2_{i|pa(i)}$ we use an inverse gamma prior with parameters $\delta_{i|pa(i)}$ and $\lambda_{i|pa(i)}$:

$$\sigma^2_{i|pa(i)} \mid d \sim \mathrm{IG}\left(\delta_{i|pa(i)},\, \lambda_{i|pa(i)}\right).$$

Finally, by supposing that there exist D possible dags, which, in the absence of subject-matter information, have all the same probability, we get a discrete uniform distribution for d: $p(d) = 1/D$. Taking advantage of the well-known factorization property of the joint distribution

$$p(\boldsymbol{x} \mid \boldsymbol{\beta}, \boldsymbol{\sigma}^2, d) \;=\; \prod_{i=0}^{p-1} p(x_i \mid \boldsymbol{x}_{pa(i)}, \boldsymbol{\beta}_{i|pa(i)}, \sigma^2_{i|pa(i)})$$

and the "global parameter independences"

$$\begin{aligned} p(\boldsymbol{\beta} \mid \boldsymbol{\sigma}^2, d) &= \prod_{i=0}^{p-1} p(\boldsymbol{\beta}_{i|pa(i)} \mid \boldsymbol{\sigma}^2_{i|pa(i)}), \\ \text{and} \qquad p(\boldsymbol{\sigma}^2 \mid d) &= \prod_{i=0}^{p-1} p(\sigma^2_{i|pa(i)}) \end{aligned}$$

(for a detailed description see Geiger and Heckerman, 1999), we get the joint distribution:

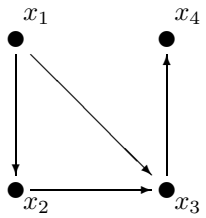
$$\begin{aligned} & p(\boldsymbol{x}, \boldsymbol{\beta}, \boldsymbol{\sigma}^2, d) \\ &= p(\boldsymbol{x} \mid \boldsymbol{\beta}, \boldsymbol{\sigma}^2, d)\, p(\boldsymbol{\beta} \mid \boldsymbol{\sigma}^2, d)\, p(\boldsymbol{\sigma}^2 \mid d)\, p(d) \\ &= \prod_{i=0}^{p-1} p(\boldsymbol{x}_i \mid \boldsymbol{x}_{pa(i)}, \boldsymbol{\beta}_{i|pa(i)}, \sigma^2_{i|pa(i)}) \prod_{i=0}^{p-1} p(\boldsymbol{\beta}_{i|pa(i)} \mid \sigma^2_{i|pa(i)}) \\ &\quad \prod_{i=0}^{p-1} p(\sigma^2_{i|pa(i)})\, p(d) \end{aligned}$$

Representation of DAGs

To represent dags we rely on the concept of adjacency matrices. For a given graph $\mathcal{G} = (V, E)$ with $|V| = p$, the adjacency matrix of $\mathcal{G}$ is defined as the $(p \times p)$-matrix A, $[A]_{ij} = a_{ij}$, with

$$a_{ij} = \begin{cases} 1, & \text{if } (v_i, v_j) \in E \\ 0, & \text{if } (v_i, v_j) \notin E. \end{cases}$$

In general, all three types of graphs (undirected, directed and chain graphs) can be uniquely represented by the corresponding adjacency matrix. Note that regarding dags, as we do, the parents of the vertex i are indicated by the i-th column, while its children are given in the i-th row. We use the representation via adjacency matrices also to check the acyclicity of the graph. For an illustration of this concept consider the graph in Figure 10.1.



$$A = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

Figure 10.1: A directed acyclic graph containing and the corresponding adjacency matrix A.

Reversible Jump Algorithm for Continuous Variables

We are not only interested in estimating the parameters for a given dag d but also want to learn about the structure of d itself. So we need to construct a Markov chain which has $\pi(d, \boldsymbol{\mu}, \boldsymbol{\beta}, \boldsymbol{\sigma}^2 \mid \boldsymbol{x})$ as its invariant distribution. Changing the dag like adding or deleting a directed edge implies also a changing in the dimension of the parameter space. To deal with this situation we use a reversible jump algorithm. Reversible jump MCMC was proposed and described by Green (1995); it can be regarded as a generalization of the usual MCMC and allows to sample simultaneously from parameter spaces of different dimensions.

Our algorithm can be briefly summarized by the following moves, which produce a Markov chain in the state space that is made up by the vector of unknowns $(d, \boldsymbol{\beta}, \boldsymbol{\sigma}^2)$:

1. Updating the dag d by adding, switching or deleting a directed edge, remaining always in the class of directed acyclic graphs. When adding or deleting an edge this move involves a change in dimensionality of the parameter space.

2. Update $\boldsymbol{\beta}_{i|pa(i)}$, $i = 0, \ldots, p-1$.

3. Update $\sigma^2_{i|pa(i)}$, $i = 0, \ldots, p-1$.

For a detailed explanation of the different steps in the continuous case, see Fronk and Giudici (2000). For a simplification of the oftentimes crucial switch step, see Fronk (2002) and also the explanations of the option *switch* in 10.1.3.

Reversible Jump Algorithm for Binary Variables

Now we consider the situation of p binary variables of which the joint distribution is assumed to be multinomial. The influence to a variable X_i from its known parents $\boldsymbol{x}_{pa(i)}$ shall be given by a probit model, i.e.

$$p_i \;=\; E(X_i \mid \boldsymbol{x}_{pa(i)}) \;=\; \Phi(\boldsymbol{x}'_{pa(i)}\boldsymbol{\beta}_{i|pa(i)}, \sigma^2_{i|pa(i)}), \tag{10.1}$$

where $i = 1, \ldots, p$ and $\Phi(\mu, \sigma^2)$ denotes the cdf of the normal distribution. This binary situation is reduced to the continuous one by sampling a latent variable, a so-called utility, Z_i for each binary variable X_i. The general idea is found in Albert and Chib (1993). Here, we first focus on the situation that we are only interested in the main effects. I.e. we do not take any interactions into account although they now of course can occur as we do not longer consider the Gaussian case. The algorithm, which does not account for interactions, can be briefly summarized as:

1. For X_i, $i = 0, \ldots, p-1$, draw Z_i from its full conditional $\mathrm{N}(\boldsymbol{x}'_{pa(i)}\boldsymbol{\beta}_{i|pa(i)}, 1)$, which is truncated at the left by 0 if $x_i = 1$ and at the right if $x_i = 0$.
2. Add, delete, or switch a directed edge like in the Gaussian case, but take the utility Z_i instead of X_i as response in ith regression model; the covariables $\boldsymbol{x}_{pa(i)}$ of the ith model remain unchanged.
3. Update $\boldsymbol{\beta}_{i|pa(i)}$, $i = 0, \ldots, p-1$.
4. Update $\sigma^2_{i|pa(i)}$, $i = 0, \ldots, p-1$.

To be able to take interactions into account, inside the algorithm interactions are treated as own variables. Due to the enormous complexity we restrict ourselves to two way interactions which seem sufficient for most situations in practice. For details, see Fronk (2002).

Reversible Jump Algorithm for Mixed Case

For the mixed case, we assume the considered continuous and binary variables to follow a conditional Gaussian (CG) distribution. For a general introduction we refer to Lauritzen (1996). The univariate conditioned distribution of $f(x_i \mid \boldsymbol{x}_{pa(i)})$ are then CG regressions (see Lauritzen and Wermuth, 1989) and can be represented by a normal regression resp. a probit model with mixed covariables.

1. For all variables X_i, $i = 0, \ldots, p-1$,

   If X_i is discrete,

   For all observations X_{ki}, $k = 1, \ldots, n$,
   draw utility Z_{ki} from full conditional $Z_{ki} \mid x_{ki}, \boldsymbol{x}_{kpa(i)}\boldsymbol{\beta}_{i|pa(i)}$

2. Update d, i.e. cancel, add, or switch the directed edge $X_j \to X_i$; thereby distinguish

   - Response X_i is continuous:
     (a) Take the algorithm for the Gaussian case, where now the covariables $pa(X_i)$ can be continuous or binary
   - Response X_i is discrete
     (a) Replace binary response X_i by continuous utility Z_i
     (b) Consider the new or vanishing interactions among the parents of X_i and possibly X_j, that can be pairwise discrete or mixed

(c) Carry out birth, death or switch step

3. Update $\boldsymbol{\beta}_{i|pa(i)}$, $i = 0, \ldots, p-1$.

4. Update $\sigma^2_{i|pa(i)}$, $i = 0, \ldots, p-1$.

For detailed explanations, see again Fronk (2002).

Remark about Markov-equivalence

Our algorithm does not take care about the so-called Markov equivalence, which describes the fact that different dags can represent the same statistical model. Equivalent dags can be summarized to equivalent classes which again can be represented by one single graph, the essential graph. Of course model selection could be done in a more effective way if only the space of those essential graphs would be considered. This will be a task of our research in future. For more details concerning Markov-equivalence we refer to papers of Andersson et al. (1997a, b) and Chickering (1995).

10.1.2 Syntax

The creation of objects has been described in general in subsection 2.5.1; in the context of dag models the corresponding dag object is created by:

`dag` *objectname*,

To perform a model selection as described above call:

objectname.`estimate` *variables* [`if` *expression*], [*options*] `using` *dataset*

Then the method `estimate` estimates the dag for the variables given in *variables* which have to be defined in *dataset*. The parameters of the via the dag defined regression models are also estimated. An if-statement may be specified to analyze only a part of the data set, i.e. only those observations where *expression* is true. There are several facultative *options* concerning the (start) parameters of the algorithm or the kind of output at the end. They are listed in the next paragraphs.

10.1.3 Options

Options for controlling MCMC simulations

The following options correspond to those given on page 75 and are therefore only briefly explained.

- `burnin =` b
  Changes the number of burnin iterations from 2000 to b; it is a positive integer number with $0 < b < 500001$.
  DEFAULT: `burnin = 2000`

- `iterations =` i
  Changes the number of MCMC iterations from 52000 to i; it is a positive integer number with $0 < i < 10000000$.
  DEFAULT: `iterations = 52000`

- `step = s`
  Changes the thinning parameter of MCMC iterations from 50 to s; it is a positive integer number with $0 < s < 1000$.
  DEFAULT: `step = 50`

Options for initial values of algorithm

- **Changing hyperparameters of partial variances**
  As already mentioned we assume $\sigma^2_{i|pa(i)} \sim IG(\delta_{i|pa(i)}, \lambda_{i|pa(i)})$ for $i = 0, \ldots, p-1$. By the following two commands the values of the two hyperparameters $\delta_{i|pa(i)}$ and $\lambda_{i|pa(i)}$ can be freely chosen. If this is not done the default values correspond to a non-informative gamma distribution.

  - `delta = c`
    Specifies the first parameter of the inverse gamma distribution of the partial variances, $\delta_{i|pa(i)}$, is set equal to d. Otherwise it is equal to 1. The value c has to be of type realvalue with $0 < c < 20$.
    DEFAULT: `delta = 1`
  - `lambda = l`
    Specifies the second parameter of the inverse gamma distribution of the partial variances, $\lambda_{i|pa(i)}$, is set equal to l. Otherwise it is equal to 0.005. The value d has to be of type realvalue with $0 < d < 20$.
    DEFAULT: `lambda = 0.005`

- **Choosing special graph to start from**
  Usually the algorithm starts from the independent model, that means from a dag without any edges. This can be changed by the command

  `type = 0/1/2/3/4`
  DEFAULT: `type = 0`

  where the different values have the following meanings:

  - `type=0`
    Algorithm starts from an **independent** model with no edges.
  - `type=1`
    Algorithm starts from a **complete** model where all edges are directed from "lower" variables to "higher" ones, i.e. $x_i \to x_j$, $\forall i < j$.
  - `type=2`
    Algorithm starts from a **complete** model where all edges are directed from "higher" variables to "lower" ones, i.e. $x_j \to x_i$, $\forall i < j$.
  - `type=3`
    Algorithm starts from a model where there is an edge from each variable to the next "higher" one, like a **chain**, i.e. $x_i \to x_j$, $\forall i = j + 1$.
  - `type=4`
    Algorithm starts from a model where there is an edge from each variable to the next "lower" one, like a **chain**, i.e. $x_j \to x_i$, $\forall i = j + 1$.

Options concerning the way of model selection

- **Kind of switch step**
  There are three ways how the switch step can be carried out in the rj-algorithm. The first one is similar to the performance of a birth or death step (i.e. adding or deleting an edge): A proposal is made and then accepted by its corresponding acceptance ratio. As it may be very complicate to calculate a good proposal, a simplification can be achieved by the consideration if the switch step leads to an equivalent dag model. If this holds true, the given and the proposed dag should be statistically indistinguishable. The proposed dag can therefore be accepted with probability 0.5. The kind of switch step can be chosen by the command

  `switch = normal/equi/mix`,
  DEFAULT: `switch = normal`

  which differ in the following way:

  - `switch=normal`
    The switch step is carried out by proposing the new dag and accepting it with the corresponding acceptance probability. The transformation into equivalent model may occur very seldom and, consequently, the acceptance ratio very low.
  - `switch=equi`
    The switch step is only allowed if it results into an equivalent model. In this case, it is performed with a probability of 0.5. Transformations into a non-equivalent model can only occur by a birth or death step.
  - `switch=mix`
    This command causes a mixture of both procedures described above: If the proposed switch step leads to an equivalent model it is accepted with probability 0.5. If it results into a non-equivalent model a proposal is made and accepted by the corresponding acceptance ratio.

- **Kind of distribution family / interactions**
  There are three different types of data sets as they can consists of continuous, binary, or mixed variables which results in the assumption of a Gaussian, a multinomial, or a conditional Gaussian distribution. Dependent on the kind of data set the rj-algorithm for the model selection changes as described above.This can be indicated by the optional command

  `family = continuous/discrete/mixed`.

  In the case that the model selection for a binary data set shall be carried out accounting for interactions the command

  `family = discrete_ia`

  is needed instead of `family = discrete`. In this case, a special option concerning the output is given by the command `detail_ia` which is explained below.

- **Restriction to the search space**
  It is possible to restrict the search space, i.e. to state an (missing) edge as fix or determinate the orientation of an edge. This is done by writing the restrictions into a file *restrict* which is then read by the command

`fix_file =` *path_of_restrict*

The restriction is then given by a $p \times p$ matrix that lies under the path *path_of_restrict*. The matrix is allowed to have three possible entries, namely 0,1, and 2 which have the following meaning: An entry of 2 corresponds to no restriction of the corresponding edge, it may occur or not. An entry of 1 indicates that this edge has to exist in each graph of the Markov chain, whereas 0 denotes that the corresponding edge must not occur.

Options concerning the output

- **Estimated regression coefficients**
  As already mentioned, the parameters of each regression model are estimated in every iteration. Because of the fact that the qualitative structure of the dag is usually of greater interest than the quantitative estimations of the regression coefficients, in the standard output these estimated parameters are omitted. Nevertheless

  `print_dags`

  gives the mean, the 10%, 50% and the 90% quantile of every parameter of all regression models. As the model space for dags is very huge we abandon the possibility to store the estimated values for each dag. To perform the necessary calculation *BayesX* creates a temporary file under the device *c:\\...*. For this purpose, its important to ensure that a device with this name exists. Otherwise the user has to provide an alternative path for the storage file by the command

  `store_file =` *alternative_path*

- **Estimated coefficients of interactions**

- **Criteria for the listed models**
  As model selection for dags is performed in an extremely huge search space one might not want to get a list of all models which have been visited during MCMC estimation regardless of the relative frequency of their appearance. The option

  `print_models = all/prob/limit/normal`,
  DEFAULT: `print_models = normal`

  allows to focus on special criterions for the models printed in the output.

  - `print_models = all`
    All models which have been visited by the Markov chain are printed.
  - `print_models = prob`
    The most frequent models of the chain are printed except for those which are the less frequent ones and have altogether a probability of alpha=0.05. The value of alpha can be changed as it is explained a few lines below.
  - `print_models = limit`
    Here, the first 10 models with the highest frequencies are printed. The number of listed models can be made different from 10 as it is explained a few lines below.
  - `print_models = normal`
    The option `normal` is a mixture of `limit` and `prob`, as it chooses the one which produces less models. The default parameters are again alpha=0.05 and number=10.

Number of different dags visited by the algorithm: 16

********* DIFFERENT MODELS sorted by frequencies *********
******************** all models *********************

010	*000*	*000*	*1*	*3894*	*0.3890*
000	*100*	*000*	*1*	*3814*	*0.3810*
000	*100*	*100*	*2*	*806*	*0.0806*

⋮

Figure 10.2: Example for model listing in the output.

The default setting of `print_models` is `print_models=normal`.

- `number =` n
  Changes the number of printed models in the option `print_models = limit` to n. The variable number has to be of type realvalue with $0 \leq n \leq 10000$.
  DEFAULT = 10

- `alpha =` a
  Sets alpha in the option `print_models = prob` equal to a. That means when using `print_models = prob` the most frequent models which unify 1-a of the posterior probability are printed. The variable alpha has to be of type intvalue with $a \in [0, 1]$.
  DEFAULT = 0.05

- `printit =` p
  Prints every p-th iteration in the output window instead of every 100-th. The printing of the iterations can be suppressed by setting p higher than the number of iterations. The variable printit has to be of type intvalue with $0 < p < 10000001$.
  DEFAULT = 100

10.1.4 Estimation Output

The output can be written to a file by opening a logfile before using the estimation command. It has to be closed afterwards. The use of logfiles is described in detail in section 3.2. The output itself is structured as follows:

- **Listing of different dags:**
  The different models are listed by their adjacency matrices. In order to save space, the different rows are printed in one line with a blank indicating the beginning of a new one. The number of edges is given as well as the absolute and relative frequency of the model. For example the first line of the exemplifying output in Figure 10.2 gives the information that the most frequent model is represented by the adjacency matrix $A = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$ and contains 1 edge. It occurred in the thinned out Markov chain for 3894 times which corresponds to a relative frequency of about 0.389. Notice, that different dags may represent the same statistical model as they may be Markov-equivalent.

- **Listing of essential graphs**
  Additional to the dags, the essential graphs are printed, too. I.e. those dags, which are

Number of different equivalent classes visited by the algorithm: 6

******* DIFFERENT EQUIVALENCE CLASSES sorted by frequencies ********
************************** all models ******************************

Skeleton: 010 000 000
No immoralities.
Number of edges: 1 Abs.freq.: 7708 Rel.freq.: 0.771

Skeleton: 011 000 000
Immoralities: (0;1,2)
Number of edges: 2 Abs.freq.: 806 Rel.freq.: 0.0806

Skeleton: 011 000 000
No immoralities.
Number of edges: 2 Abs.freq.: 523 Rel.freq.: 0.0523

⋮

Figure 10.3: Example for listing of equivalence classes in the output.

equivalent to each other, are summarized and represented by their essential graph. The representation of the essential graph, which can contain undirected as well as directed edges, is as follows. First the underlying graph, the skeleton, is given by the adjacency matrix as described above. But now, the entries indicate always an undirected edge. (E.g. the undirected graph a–b of the two variables a and b is given by 01 00.) Then the immoralities of the essential graph are listed. Remember that within an essential graph an oriented edge can only occur as a part of an immorality $b \to a \leftarrow c$ which is here represented by the triple (a;b,c). The example output of Figure Figure 10.3 shows that the first two dag models of Figure 10.2 which are equivalent have been summarized to their representing essential graph X_0–X_1 X_2. The next most frequent statistical model is represented by the essential graph $X_0 \to X_2 \leftarrow X_1$ which is given by our representation as the skeleton matrix 011 000 000 and the immorality (0;1,2).

- **Averaged adjacency matrix:**
  The (i, j)-th element of the averaged adjacency matrix gives the estimated posterior probability of the presence of the edge $i \to j$ in the true dag.

- **Mean of skeletons:**
  The skeleton of a dag is defined as the same graph without regarding the directions of the edges. Equivalent dags have at least the same skeleton. So it may be helpful to have also a look at the averaged matrix of the skeletons, which is of course symmetric.

- **Correlation:**
  The marginal and the partial correlation matrices of the regarded data set is given, too.

- **Ratios:**
  We give some short information about the acceptance ratios for the birth-, death- and switch-

steps which denotes the cases where an edge is added, dropped or switched. The first two cases imply a change in dimension and are therefore sampled by a reversible jump step.

- **Estimated parameters:**
  If the option `print_dags` is used, the estimated regression coefficients $\beta_{i|-i}$, $i = 0, \ldots, p-1$, are listed at the end. The notation $-i$ denotes all variables except for i. Besides the mean of the sampled Markov chain for each parameter there is also the 10%, the 50% and the 90% quantile given. As in equivalent models the direction of edges and, thus, also the regression models vary, in most cases the estimated regression coefficients do not give a deeper insight into the model and have to be interpreted in a very careful way.

10.1.5 References

Andersson, S. A., Madigan, D., and Perlman, M. D. (1997a). A Characterization of Markov equivalence Classes for Acyclic Digraphs. *The Annals of Statistics*, **25**, 505–541.

Andersson, S. A., Madigan, D., and Perlman, M. D. (1997b). On the Markov equivalence of Chain Graphs, Undirected Graphs, and Acyclic Digraphs. *Scandinavian Journal of Statistics*, **24**, 81–102.

Chickering, D. M. (1995). A Transformational Characterization of Equivalent Bayesian Network Structures. In P. Besnard and S. Hanks (Eds.), *Uncertainty in Artificial Intelligence, Proceedings of the Eleventh Conference*, San Francisco: Morgan Kaufmann, pp. 87 – 98

Brooks, S. P. (1998). Markov Chain Monte Carlo Method and its Application. *The Statistician*, **47**, 69–100.

Fronk, E.–M. and Giudici, P. (2000). Markov Chain Monte Carlo model selection for DAG Models. *Discussion Paper*, Universität München.

Geiger, D. and Heckerman, D. (1994). Learning Gaussian Networks. *Proceedings of the Tenth Conference on Uncertainty in Artificial Intelligence.* Morgan Kaufmann, 235–243.

Geiger, D. and Heckerman, D. (1999). Parameter priors for directed acyclic graphical models and the characterisation of several probability distributions. Submitted for publication.

Gilks, W. R., Richardson, S., and Spiegelhalter, D. J. (1996). *Markov Chain Monte Carlo in Practice.*,Chapman and Hall, London.

Giudici, P. and Green, P. J. (1999). Decomposable Graphical Gaussian Model Determination. *Biometrika*, **86**, 785–801.

Green, P. J. (1995). Reversible Jump Markov Chain Monte Carlo Computation and Bayesian Model Determination. *Biometrika*, **82**, 711–32.

Lauritzen, S. L. (1996). *Graphical Models*, Clarendon Press, Oxford.

Index