BayesX

Software for Bayesian Inference

Version 1.30

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;

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 chapter 8.

- **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 7 and chapter 9.

- **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 2.4 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 chapter 7 to make yourself familiar with STAR regression models. Proceed then with the tutorial like subsection 8.6.2 of chapter 8.
Bayesian semiparametric regression based on MCMC simulation techniques. At least a basic knowledge about MCMC techniques exists.	Read chapter 7 to make yourself familiar with STAR regression models. Proceed then with the tutorial like subsection 8.6.2 of chapter 8.
Semiparametric regression based on mixed model methodology.	Read chapter 7 to make yourself familiar with STAR regression models. Proceed then with the tutorial like subsection 9.4.1 of chapter 9.
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 2.5. The `examples` directory also contains some tutorial programs that illustrate the usage of most of the functions of *BayesX*, see section 8.6 and section 9.4 . 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 subsection 8.4.2 and subsection 9.2.2 for detailed descriptions 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 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 program description
`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 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.

2.3.1 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.

2.3.2 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*.

2.3.3 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.

2.3.4 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.

2.3.5 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.

2.4 General usage of BayesX

2.4.1 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 8 for a detailed description of *bayesreg objects* and chapter 9 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.

2.4.2 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.

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

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

```
> 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.

2.5 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.

2.5.1 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 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*.

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

References

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

2.5.2 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 8.6.1 for a Bayesian regression analysis of the data set.

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

References

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

2.5.3 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 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 the tutorial like examples in subsection 8.6.2 and subsection 9.4.1.

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 8). 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

Examples

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

Examples

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).

Examples

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.

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.

4.3.3 Constants

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

Examples

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
```

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

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

4.3.4 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.

Examples

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)
```

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

Examples

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.

Examples

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.

Examples

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

Examples

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

Examples

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 Examples

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 2.5.2. 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 8 or chapter 9. 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 8, in particular section 8.1, 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.

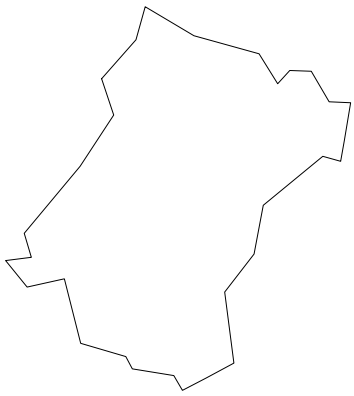


Figure 5.1: Corresponding graph of the section of the boundary file

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
⋮

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
    ⋮
```

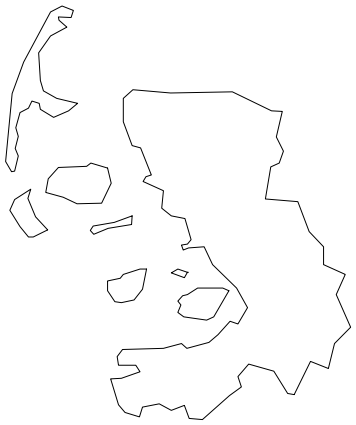


Figure 5.2: Example for a region that is divided into subregions

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

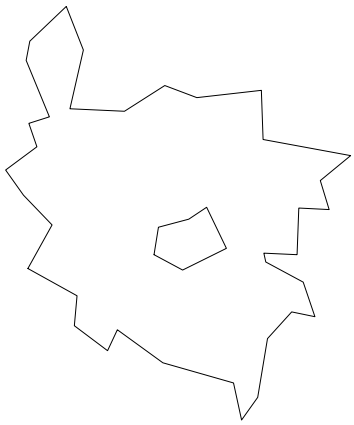


Figure 5.3: Example for a region that is totally surrounded by another region

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 8.1. 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

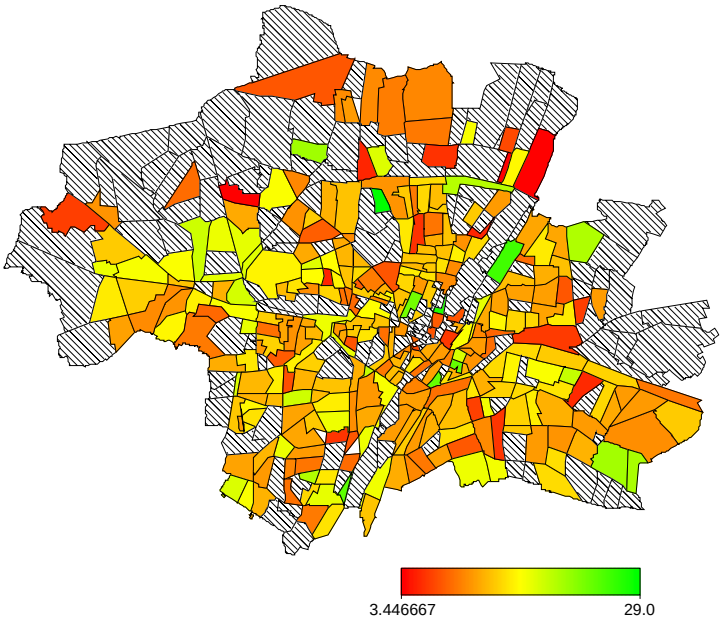


Figure 6.2: Distribution of the average rents per square meter in Munich.

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=p|l[specifications for further variables]**

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

  `p` do not connect, i.e. plot points only
  `l` (letter l) draw straight lines between the points (default)

  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 symbol (`p,l`) for every *yvar*. Typing for example

  ```
  connect = lp
  ```

  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.

- **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"`).

- **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.

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
```

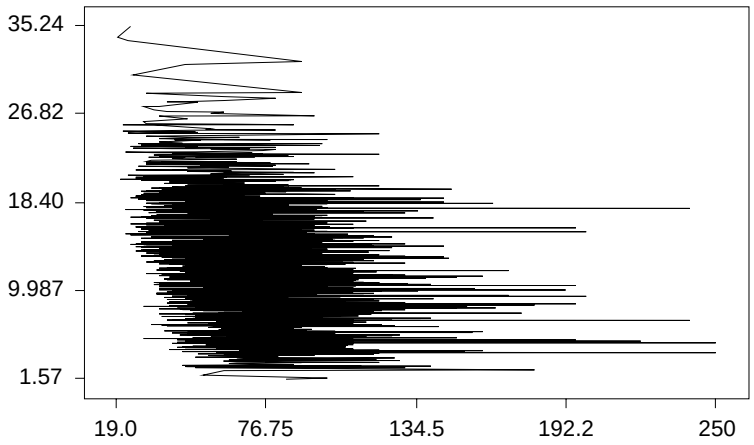


Figure 6.3: Scatterplot between floor space and rent per square meters (first try).

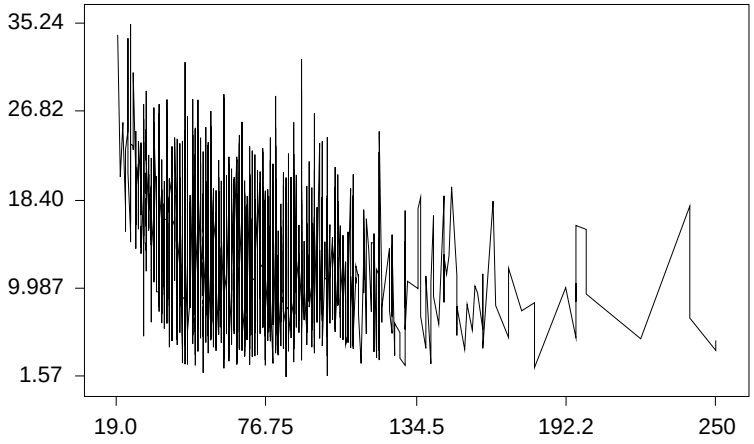


Figure 6.4: Scatterplot between floor space and rent per square meters (second try).

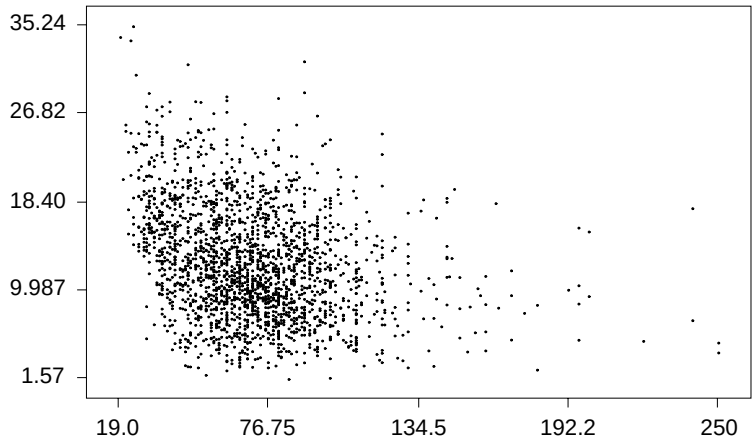


Figure 6.5: Scatterplot between floor space and rent per square meters (third try).

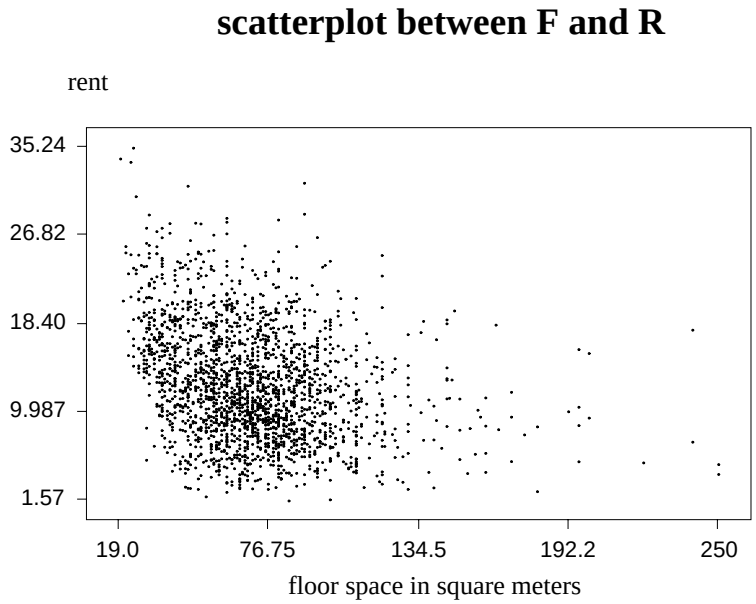


Figure 6.6: Scatterplot between floor space and rent per square meters (final try).

6.3 Method plotautocor

Description

Method `plotautocor` visualizes the autocorrelation functions obtained with method `autocor` of *bayesreg objects*, see also section 8.2.

Syntax

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

Plots the autocorrelation functions stored in *dataset*. The data set must have the special structure described in section 8.2, 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 8.3. 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 8.3.

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

Bayesian structured additive regression

In this chapter we provide the methodological background for the two regression tools presented in chapter 8 and chapter 9. The first regression tool (*bayesreg objects*) relies on Markov chain Monte Carlo simulation techniques. The second regression tool (*remlreg objects*) is based on mixed model representations of regression models and inference is based on restricted maximum likelihood (REML). Both regression tools are able to estimate complex semiparametric regression and survival models with *structured additive predictor*, STAR, Fahrmeir, Kneib and Lang, 2004). STAR models cover a number of well known model classes as special cases, including *generalized additive models* (Hastie and Tibshirani, 1990), *generalized additive mixed models* (Lin and Zhang, 1999), *geoadditive models* (Kammann and Wand, 2003), *varying coefficient models* (Hastie and Tibshirani, 1993), and *geographically weighted regression* (Fotheringham, Brunsdon, and Charlton, 2002).

7.1 Observation model

Bayesian generalized linear models (e.g. Fahrmeir and Tutz, 2001) assume that, given covariates u and unknown parameters γ, the distribution of the response variable y belongs to an exponential family, i.e.

$$p(y \mid u) = \exp\left(\frac{y\theta - b(\theta)}{\phi}\right) c(y, \phi) \tag{7.1}$$

where $b(\cdot)$, $c(\cdot)$, θ and ϕ determine the respective distributions. A list of the most common distributions and their specific parameters can be found e.g. in Fahrmeir and Tutz (2001), page 21. The mean $\mu = E(y|u, \gamma)$ is linked to a linear predictor η by

$$\mu = h(\eta) \qquad \eta = u'\gamma. \tag{7.2}$$

Here h is a known response function, and γ are unknown regression parameters.

In most practical regression situations, however, we are facing at least one of the following problems:

- For the *continuous covariates* in the data set, the assumption of a strictly linear effect on the predictor may be not appropriate.
- Observations may be *spatially correlated*.
- Observations may be *temporally correlated*.
- Heterogeneity among individuals or units may be not sufficiently described by covariates. Hence, unobserved *unit or cluster specific heterogeneity* must be considered appropriately.

To overcome the difficulties, we replace the strictly linear predictor in (7.2) by a structured additive predictor

$$\eta_r = f_1(x_{r1}) + \ldots + f_p(x_{rp}) + u_r'\gamma, \tag{7.3}$$

where r is a generic observation index, the x_j denote generic covariates of different type and dimension, and f_j are (not necessarily smooth) functions of the covariates. The functions f_j comprise usual nonlinear effects of continuous covariates, time trends and seasonal effects, two dimensional surfaces, varying coefficient models, i.i.d. random intercepts and slopes and spatially correlated random effects. In order to demonstrate the generality of the approach we point out some special cases of (7.3) well known from the literature:

- *Generalized additive model (GAM) for cross-sectional data*
  A GAM is obtained if the x_j, $j = 1, \ldots, p$, are univariate and continuous and f_j are smooth functions. In *BayesX* the functions f_j are modelled either by random walk priors or P-splines, see subsection 7.2.1.

- *Generalized additive mixed model (GAMM) for longitudinal data*
  Consider longitudinal data for individuals $i = 1, \ldots, n$, observed at time points $t \in \{t_1, t_2, \ldots\}$. For notational simplicity we assume the same time points for every individual, but generalizations to individual specific time points are obvious. A GAMM extends a GAM by introducing individual specific random effects, i.e.

  $$\eta_{it} = f_1(x_{it1}) + \ldots + f_k(x_{itk}) + b_{1i}w_{it1} + \cdots + b_{qi}w_{itq} + u_{it}'\gamma$$

  where $\eta_{it}, x_{it1}, \ldots, x_{itk}, w_{it1}, \ldots, w_{itq}, u_{it}$ are predictor and covariate values for individual i at time t and $b_i = (b_{1i}, \ldots, b_{qi})$ is a vector of q i.i.d. random intercepts (if $w_{itj} = 1$) or random slopes. The random effects components are modelled by i.i.d. Gaussian priors, see subsection 7.2.3. GAMM's can be subsumed into (7.3) by defining $r = (i, t)$, $x_{rj} = x_{itj}$, $j = 1, \ldots, k$, $x_{r,k+h} = w_{ith}$, and $f_{k+h}(x_{r,k+h}) = b_{hi}w_{ith}$, $h = 1, \ldots, q$. Similarly, GAMM's for cluster data can be written in the general form (7.3).

- *Geoadditive Models*
  In many situations additional geographic information for the observations in the data set is available. As an example compare one of our demonstrating examples in subsection 8.6.2 and subsection 9.4.1 on the determinants of childhood undernutrition in Zambia. Here, the district where the mother of a child lives is given and may be used as an indicator for regional differences in the health status of children. A reasonable predictor for such data is given by

  $$\eta_r = f_1(x_{r1}) + \ldots + f_k(x_{rk}) + f_{spat}(s_r) + u_r'\gamma \tag{7.4}$$

  where f_{spat} is an additional spatially correlated effect of the location s_r an observation pertains to. Models with a predictor that contains a spatial effect are also called geoadditive models, see Kammann and Wand (2003). In *BayesX*, the spatial effect may be modelled by Markov random fields (Besag, York and Mollie (2003)) or two dimensional P-splines (Lang and Brezger, 2003), compare subsection 7.2.2.

- *Varying coefficient model (VCM) - Geographically weighted regression*
  A VCM as proposed by Hastie and Tibshirani (1993) is defined by

  $$\eta_r = g_1(w_{r1})z_{r1} + \cdots + g_p(w_{rp})z_{rp},$$

  where the effect modifiers w_{rj} are continuous covariates or time scales and the interacting variables z_{rj} are either continuous or categorical. A VCM can be cast into (7.3) with $x_{rj} =$

(w_{rj}, z_{rj}) and by defining the special function $f_j(x_{rj}) = f_j(w_{rj}, z_{rj}) = g_j(w_{rj})z_{rj}$. Note that in *BayesX* the effect modifiers are not necessarily restricted to be continuous variables as in Hastie and Tibshirani (1993). E.g. the geographical location may be used as effect modifier as well. VCM's with spatially varying regression coefficients are well known in the geography literature as *geographically weighted regression*, see e.g. Fotheringham, Brunsdon, and Charlton (2002).

- *ANOVA type interaction model*
  Suppose w_r and z_r are two continuous covariates. Then, the effect of w_r and z_r may be modelled by a predictor of the form

$$\eta_r = f_1(w_r) + f_2(z_r) + f_{1|2}(w_r, z_r) + \ldots,$$

  see e.g. Chen (1993). The functions f_1 and f_2 are the main effects of the two covariates and $f_{1|2}$ is a two dimensional interaction surface which can be modelled e.g. by two dimensional P-splines, see subsection 7.2.4. The interaction can be cast into the form (7.3) by defining $x_{r1} = w_r$, $x_{r2} = z_r$ and $x_{r3} = (w_r, z_r)$.

At first sight it may look strange to use one general notation for nonlinear functions of continuous covariates, i.i.d. random intercepts and slopes, and spatially correlated effects as in (7.3). However, the unified treatment of the different components in the model has several advantages:

- Since we adopt a Bayesian perspective it is generally not necessary to distinguish between fixed and random effects because in a Bayesian approach all unknown parameters are assumed to be random.
- As we will see below in section 7.2 the priors for smooth functions, two dimensional surfaces, i.i.d., serially and spatially correlated effects can be cast into a general form.
- The general form of the priors also allows rather general and unified estimation procedures, see section 7.3. As a side effect the implementation and description of these procedures is considerably facilitated.

7.2 Prior assumptions

For Bayesian inference, the unknown functions $f_1, \ldots, f_p$ in (7.3), more exactly corresponding vectors of function evaluations, and the fixed effects parameters γ are considered as random variables and must be supplemented by appropriate prior assumptions.

In the absence of any prior knowledge diffuse priors are the appropriate choice for fixed effects parameters, i.e.

$$\gamma_j \propto const$$

Another common choice not yet supported by *BayesX* are informative multivariate Gaussian priors with mean μ_0 and covariance matrix Σ_0.

Priors for the unknown functions $f_1, \ldots, f_p$ depend on the *type of the covariates* and on *prior beliefs about the smoothness of* f_j. In the following we will always be able to express the vector of function evaluations $f_j = (f_j(x_{1j}), \ldots, f_j(x_{nj}))'$ of an unknown function f_j as the matrix product of a design matrix X_j and a vector of unknown parameters β_j, i.e.

$$f_j = X_j\beta_j. \tag{7.5}$$

Then, we obtain the predictor (7.3) in matrix notation as

$$\eta = X_1\beta_1 + \cdots + X_p\beta_p + U\gamma, \tag{7.6}$$

where U corresponds to the usual design matrix for fixed effects.

A prior for a function f_j is now defined by specifying a suitable design matrix X_j and a prior distribution for the vector β_j of unknown parameters. The general form of the prior for β_j is given by

$$p(\beta_j|\tau_j^2) \propto \frac{1}{(\tau_j^2)^{rank(K_j)/2}} \exp\left(-\frac{1}{2\tau_j^2}\beta_j' K_j \beta_j\right), \tag{7.7}$$

where K_j is a *penalty matrix* that shrinks parameters towards zero or penalizes too abrupt jumps between neighboring parameters. In most cases K_j will be rank deficient and therefore the prior for β_j is partially improper.

The variance parameter τ_j^2 is equivalent to the inverse smoothing parameter in a frequentist approach and controls the trade off between flexibility and smoothness.

In the following we will describe specific priors for different types of covariates and functions f_j.

7.2.1 Priors for continuous covariates and time scales

Several alternatives have been proposed for specifying smoothness priors for continuous covariates or time scales. These are *random walk priors* or more generally *autoregressive priors* (see Fahrmeir and Lang, 2001a, and Fahrmeir and Lang, 2001b), *Bayesian P-splines* (Lang and Brezger, 2003) and *Bayesian smoothing splines* (Hastie and Tibshirani, 2000). *BayesX* supports random walk priors and P-splines.

7.2.1.1 Random walks

Suppose first that x_j is a time scale or continuous covariate with equally spaced ordered observations

$$x_j^{(1)} < x_j^{(2)} < \cdots < x_j^{(M_j)}.$$

Here $M_j \leq n$ denotes the number of *different* observed values for x_j in the data set. A common approach in dynamic or state space models is to estimate one parameter β_{jm} for each distinct $x_j^{(m)}$, i.e. $f_j(x_j^{(m)}) = \beta_{jm}$, and penalize too abrupt jumps between successive parameters using random walk priors. Most commonly used are first or second order random walk models

$$\beta_{jm} = \beta_{j,m-1} + u_{jm} \quad \text{or} \quad \beta_{jm} = 2\beta_{j,m-1} - \beta_{j,m-2} + u_{jm} \tag{7.8}$$

with Gaussian errors $u_{jm} \sim N(0, \tau_j^2)$ and diffuse priors $\beta_{j1} \propto$ const, or β_{j1} and $\beta_{j2} \propto$ const, for initial values, respectively. Both specifications act as smoothness priors penalizing too rough functions f_j. A first order random walk penalizes too abrupt jumps $\beta_{jm} - \beta_{j,m-1}$ between successive states and a second order random walk penalizes deviations from the linear trend $2\beta_{j,m-1} - \beta_{j,m-2}$. The joint distribution of the regression parameters β_j is easily computed as a product of conditional densities defined by (7.8) and can be brought into the general form (7.7). The penalty matrix is of the form $K_j = D'D$ where D is a first or second order difference matrix. For example, for a random walk of first order the penalty matrix is given by:

$$K_j = \begin{pmatrix} 1 & -1 & & & \\ -1 & 2 & -1 & & \\ & \ddots & \ddots & \ddots & \\ & & -1 & 2 & -1 \\ & & & -1 & 1 \end{pmatrix}.$$

The design matrix X_j is a simple 0/1 matrix where the number of columns is equal to the number of parameters, respectively the number of distinct covariate values. If for observation r the value of x_j is l, then the element in the r-th row and l-th column of X_j is one and zero otherwise.

In the case of non-equally spaced observations slight modifications of the priors defined in (7.8) are necessary, see Fahrmeir and Lang (2001a) for details.

If x_j is a time scale we may introduce an additional seasonal effect of x_j. A common smoothness prior for a seasonal component $f_j(x_j^{(m)}) = \beta_{jm}$ is given by

$$\beta_{jm} = -\beta_{j,m-1} - \cdots - \beta_{j,m-per-1} + u_{jm} \tag{7.9}$$

where $u_{jm} \sim N(0, \tau_j^2)$ and per is the period, for instance $per = 12$ for monthly data. Compared to a dummy variable approach this specification has the advantage that it allows for a time varying rather than a time constant seasonal effect.

7.2.1.2 P-splines

A closely related approach for continuous covariates is based on P-splines introduced by Eilers and Marx (1996). The approach assumes that an unknown smooth function f_j of a covariate x_j can be approximated by a polynomial spline of degree l defined on a set of equally spaced knots $x_j^{min} = \zeta_0 < \zeta_1 < \ldots < \zeta_{d-1} < \zeta_d = x_j^{max}$ within the domain of x_j. Such a spline can be written in terms of a linear combination of $M_j = d + l$ B-spline basis functions B_m, i.e.

$$f_j(x_j) = \sum_{m=1}^{M_j} \beta_{jm} B_m(x_j).$$

Here $\beta_j = (\beta_{j1}, \ldots, \beta_{jM_j})'$ corresponds to the vector of unknown regression coefficients. The $n \times M_j$ design matrix X_j consists of the basis functions evaluated at the observations x_{rj}, i.e. $X_j(r, m) = B_m(x_{rj})$. The crucial point is the choice of the number of knots. For a small number of knots, the resulting spline may be not flexible enough to capture the variability of the data. For a large number of knots, estimated curves tend to overfit the data and, as a result, too rough functions are obtained. As a remedy Eilers and Marx (1996) suggest a moderately large number of equally spaced knots (usually between 20 and 40) to ensure enough flexibility, and to define a roughness penalty based on first or second order differences of adjacent B-Spline coefficients to guarantee sufficient smoothness of the fitted curves. This leads to penalized likelihood estimation with penalty terms

$$P(\lambda_j) = \lambda_j \sum_{m=k+1}^{M_j} (\Delta^k \beta_{jm})^2, \quad k = 1, 2 \tag{7.10}$$

where λ_j is the smoothing parameter. First order differences penalize abrupt jumps $\beta_{jm} - \beta_{j,m-1}$ between successive parameters and second order differences penalize deviations from the linear trend $2\beta_{j,m-1} - \beta_{j,m-2}$. In a Bayesian approach we use the stochastic analogue of difference penalties, i.e. first or second order random walks, as a prior for the regression coefficients. Note that simple first or second order random walks can be regarded as P-splines of degree $l = 0$ and are therefore a special case. More details about Bayesian P-splines can be found in Lang and Brezger (2003) and Brezger and Lang (2003).

7.2.2 Priors for spatial effects

Suppose that the index $s \in \{1, \ldots, S\}$ represents the location or site in connected geographical regions. For simplicity we assume that the regions are labelled consecutively. A common way to

introduce a spatially correlated effect is to assume that neighboring sites are more alike than two arbitrary sites. Thus for a valid prior definition a set of neighbors for each site s must be defined. For geographical data one usually assumes that two sites s and s' are neighbors if they share a common boundary.

The simplest (but most often used) spatial smoothness prior for the function evaluations $f_j(s) = \beta_{js}$ is

$$\beta_{js}|\beta_{js'},\ s \neq s', \tau_j^2 \sim N\left(\frac{1}{N_s}\sum_{s'\in\partial_s}\beta_{js'}, \frac{\tau_j^2}{N_s}\right), \tag{7.11}$$

where N_s is the number of adjacent sites and $s' \in \partial_s$ denotes that site s' is a neighbor of site s. Thus the (conditional) mean of β_{js} is an unweighted average of function evaluations of neighboring sites. The prior is a direct generalization of a first order random walk to two dimensions and is called a Markov random field (MRF).

A more general prior including (7.11) as a special case is given by

$$\beta_{js}|\beta_{js'}\ s \neq s', \tau_j^2 \sim N\left(\sum_{s'\in\partial_s}\frac{w_{ss'}}{w_{s+}}\beta_{js'}, \frac{\tau_j^2}{w_{s+}}\right), \tag{7.12}$$

where $w_{ss'}$ are known weights and $+$ denotes summation over the missing subscript. Such a prior is called a Gaussian intrinsic autoregression, see Besag, York and Mollie (1991) and Besag and Kooperberg (1995). Other weights than $w_{ss'} = 1$ as in (7.11) are based on the common boundary length of neighboring sites, or on the distance of the centroids of two sites. All these spatial priors are supported by *BayesX*, see page 36 for more details.

The $n \times S$ design matrix X is a 0/1 incidence matrix. Its value in the i-th row and the s-th column is 1 if the i-th observation is located in site or region s, and zero otherwise. The $S \times S$ penalty matrix K has the form of an adjacency matrix.

If exact locations $s = (s_x, s_y)$ are available, we can use two-dimensional surface estimators to model spatial effects. One option are two-dimensional P-splines, see subsection 7.2.1. Another option are Gaussian random field (GRF) priors, originating from geostatistics. They can be seen as two-dimensional surface smoothers based on special basis functions, e.g. radial basis functions, and have been used by Kammann and Wand (2003) to model the spatial component in Gaussian regression models. The spatial component $f_j(s) = \beta_s$ is then assumed to follow a zero mean stationary Gaussian random field $\{\beta_s : s \in \mathbb{R}^2\}$ with variance τ_j^2 and isotropic correlation function $cov(\beta_s, \beta_{s+h}) = C(||h||)$. This means that correlations between sites that are $||h||$ units apart are the same, regardless of direction and the sites location. For a finite array $s \in \{1, \ldots, S\}$ of sites as in image analysis or in our application to forest health data, the prior for $\beta_j = (\beta_1, \ldots, \beta_s)'$ is of the general form (7.7) with $K = C^{-1}$ and

$$C(i,j) = C(||s_i - s_j||), 1 \leq i, j \leq n.$$

The design matrix X is again a 0/1 incidence matrix.

Several proposals for the choice of the correlation function $C(r)$ have been made. In the kriging literature, the Matérn family $C(r; \rho, \nu)$ is highly recommended. For prechosen values $\nu = m + 1/2$, $m = 0, 1, 2, \ldots$ of the smoothness parameter ν simple correlation functions $C(r; \rho)$ are obtained, e.g.

$$C(r; \rho) = \exp(-|r|)(1 + |r|)$$

with $\nu = 1.5$. The parameter ρ controls how fast correlations die out with increasing $r = ||h||$. It can be determined in a preprocessing step or may be estimated with variance components by

restricted maximum likelihood. A simple rule to choose ρ is

$$\hat{\rho} = \max_{i,j} ||s_i - s_j||/c$$

ensuring scale invariance. The constant $c > 0$ is chosen in such a way, that $C(c)$ is small, e.g. 0.001. Therefore the different values of $||s_i - s_j||/\hat{\rho}$ are spread out over the r-axis of the correlation function. This choice of ρ has proved to work well in our experience.

Although we described them separately, approaches for exact locations can also be used in the case of connected geographical regions, e.g. based on the centroids of the regions. Conversely, we can also apply MRFs to exact locations if neighborhoods are defined based on a distance measure. Furthermore GRFs may be approximated by MRFs, see Rue and Tjelmeland (2002). In general, it is not clear which of the different approaches leads to the "best" fits. For data observed on a discrete lattice MRFs seem to be most appropriate. If the exact locations are available, surface estimators may be more natural, particularly because predictions for unobserved locations are available. However, in some situations surface estimators lead to an improved fit compared to MRF's even for discrete lattices and vice versa. A general approach that can handle both situations is given by Müller et al. (1997).

From a computational point of view MRF's and P-splines are preferable to GRF's because their posterior precision matrices are band matrices or can be transformed into a band matrix like structure. The special structure of the matrices considerably speeds up computations, at least for inference based on MCMC techniques. For inference based on mixed models, the main difference between GRFs and MRFs, considering their numerical properties, is the dimension of the penalty matrix. For MRFs the dimension of K equals the number of different regions S and is therefore independent from the sample size. On the other side, for GRFs, the dimension of K is given by the number of distinct locations, which is usually close to the sample size. So the number of regression coefficients used to describe a MRF is usually much smaller than for a GRF and therefore the estimation of GRFs is computationally much more expensive. To overcome this difficulty Kammann and Wand (2003) propose low-rank kriging to approximate stationary Gaussian random fields. Note first, that we can define GRFs equivalently based on a design matrix X with entries $X(i,j) = C(||s_i - s_j||)$ and penalty matrix $K = C$. To reduce the dimensionality of the estimation problem we define a subset of knots $\mathcal{D} = \{\kappa_1, \ldots, \kappa_M\}$ of the set of distinct locations $\mathcal{C}$. These knots can be chosen to be "representative" for the set of distinct locations $\mathcal{C}$ based on a space filling algorithm. Therefore consider the distance measure

$$d(s, \mathcal{D}) = \left(\sum_{\kappa \in \mathcal{D}} ||s - \kappa||^p \right)^{\frac{1}{p}},$$

with $p < 0$, between any location $s \in \mathcal{D}$ and a possible set of knots $\mathcal{C}$. Obviously this distance measure is zero for all knots. Using a simple swapping algorithm to minimize the overall coverage criterion

$$\left(\sum_{s \in \mathcal{C}} d(s, \mathcal{D})^q \right)^{\frac{1}{q}}$$

with $q > 0$ (compare Johnson et al. (1990) and Nychka and Saltzman (1998) for details) yields an optimal set of knots $\mathcal{D}$. Based on these knots we define the approximation $f_j = X\beta_j$ with the $n \times M$ design matrix $X(i,j) = C(||s_i - \kappa_j||)$, penalty matrix $K = C$ and $C(i,j) = C(||\kappa_i - \kappa_j||)$. The number of knots M allows us to control the trade-off between the accuracy of the approximation (M close to the sample size) and the numerical simplification (M small).

7.2.3 Unordered group indicators and unstructured spatial effects

In many situations we observe the problem of heterogeneity among clusters of observations caused by unobserved covariates. Neglecting unobserved heterogeneity may lead to considerably biased estimates for the remaining effects. Suppose $c \in \{1, \ldots, C\}$ is a cluster variable indicating the cluster a particular observation belongs to. A common approach to overcome the difficulties of unobserved heterogeneity is to introduce additional Gaussian i.i.d. effects $f_j(c) = \beta_{jc}$ with

$$\beta_{jc} \sim N(0, \tau_j^2), \quad c = 1, \ldots, C. \tag{7.13}$$

The design matrix X_j is again a $n \times C$ 0/1 incidence matrix and the penalty matrix is the identity matrix, i.e. $K_j = I$. From a classical perspective, (7.13) defines i.i.d. *random effects*. However, from a Bayesian point of view all unknown parameters are assumed to be random and hence the notation "random effects" in this context is misleading. We think of (7.13) more as an approach for modelling an unsmooth function.

Prior (7.13) may also be used for a more sophisticated modelling of spatial effects. In some situation it may be useful to split up a spatial effect f_{spat} into a spatially correlated (structured) part f_{str} and a spatially uncorrelated (unstructured) part f_{unstr}, i.e.

$$f_{spat} = f_{str} + f_{unstr}.$$

A rationale is that a spatial effect is usually a surrogate of many unobserved influential factors, some of them may obey a strong spatial structure and others may be present only locally. By estimating a structured and an unstructured component we aim at distinguishing between the two kinds of influential factors, see also Besag, York and Mollie (1991). For the smooth spatial part we assume Markov random field priors or two dimensional surface smoothers as described in the next section. For the uncorrelated part we may assume prior (7.13).

7.2.4 Modelling interactions

The models considered so far are not appropriate for modelling interactions between covariates. A common approach is based on varying coefficient models introduced by Hastie and Tibshirani (1993) in the context of smoothing splines. Here, the effect of covariate z_{rj}, $j = 1, \ldots, p$ is assumed to vary smoothly over the range of a second covariate w_{rj}, i.e.

$$f_j(w_{rj}, z_{rj}) = g_j(w_{rj}) z_{rj}. \tag{7.14}$$

In most cases the interacting covariate z_{rj} is categorical whereas the effect modifier may be either metrical, spatial or an unordered group indicator. For the nonlinear function g_j we may assume the priors already defined in subsection 7.2.1 for continuous effect modifiers, subsection 7.2.2 for spatial effect modifiers and subsection 7.2.3 for unordered group indicators as effect modifiers. In Hastie and Tibshirani (1993) only continuous effect modifiers have been considered. Models with spatial effect modifiers are used in Fahrmeir, Lang, Wolff and Bender (2003) and Gamerman, Moreira and Rue (2003). From a classical point of view, models with unordered group indicators as effect modifiers are called models with *random slopes*.

In matrix notation we obtain for the vector of function evaluations

$$f_j = diag(z_{1j}, \ldots, z_{nj}) X_j^* \beta_j$$

where X_j^* is the design matrix corresponding to the prior for g_j. Hence the overall design matrix is given by $X_j = diag(z_{1j}, \ldots, z_{nj}) X_j^*$.

Suppose now that both interacting covariates are continuous. In this case, a flexible approach for modelling interactions can be based on (nonparametric) two dimensional surface fitting. In *BayesX* surface fitting is based on two dimensional P-splines described in more detail in Lang and Brezger (2003) and Brezger and Lang (2003). The assumption is that the unknown surface $f_j(w_{rj}, z_{rj})$ can be approximated by the tensor product of two one dimensional B-splines, i.e.

$$f_j(w_{rj}, z_{rj}) = \sum_{m_1=1}^{M_j} \sum_{m_2=1}^{M_j} \beta_{j,m_1m_2} B_{j,m_1}(w_{rj}) B_{j,m_2}(z_{rj}).$$

Similar to one-dimensional P-splines, the $n \times M_j^2$ design matrix X_j is composed of products of basis functions. Priors for $\beta_j = (\beta_{j,11}, \ldots, \beta_{j,M_jM_j})'$ are now based on spatial smoothness priors common in spatial statistics, e.g. two dimensional first order random walks. The most commonly used prior specification based on the four nearest neighbors can be defined by

$$\beta_{j,m_1m_2}|\cdot \sim N\left(\frac{1}{4}(\beta_{j,m_1-1,m_2} + \beta_{j,m_1+1,m_2} + \beta_{j,m_1,m_2-1} + \beta_{j,m_1,m_2+1}), \frac{\tau_j^2}{4}\right) \tag{7.15}$$

for $m_1, m_2 = 2, \ldots, M_j$ and appropriate changes for corners and edges. The prior can be easily brought into the general form (7.7).

7.2.5 Mixed Model representation

You may skip this section if you are not interested in using the regression tool based on mixed model methodology (*remlreg objects* in chapter 9).

In this section, we show how STAR models can be represented as generalized linear mixed models (GLMM) after appropriate reparametrization, see also Lin and Zhang (1999) and Green (1987) in the context of smoothing splines. In fact, model (7.2) with the structured additive predictor (7.6) can always be expressed as a GLMM. This provides the key for simultaneous estimation of the functions f_j, $j = 1, \ldots, p$ and the variance parameters τ_j^2 in the empirical Bayes approach described in subsection 7.3.2 and used for estimation by *remlreg objects*. To rewrite the model as a GLMM, the general model formulation is useful again. We proceed as follows:

The vectors of regression coefficients β_j, $j = 1, \ldots, p$, are decomposed into an *unpenalized* and a *penalized part*. Suppose that the j-th coefficient vector has dimension $M_j \times 1$ and the corresponding penalty matrix K_j has rank rk_j. Then we define the decomposition

$$\beta_j = X_j^{unp}\beta_j^{unp} + X_j^{pen}\beta_j^{pen}, \tag{7.16}$$

where the columns of the $M_j \times (M_j - rk_j)$ matrix X_j^{unp} contain a basis of the nullspace of K_j. The $M_j \times rk_j$ matrix X_j^{pen} is given by $X_j^{pen} = L_j(L_j'L_j)^{-1}$ where the $M_j \times rk_j$ matrix L_j is determined by the decomposition of the penalty matrix K_j into $K_j = L_jL_j'$. A requirement for the decomposition is that $L_j'X_j^{unp} = 0$ and $X_j^{unp}L_j' = 0$ hold. Hence the parameter vector β_j^{unp} represents the part of β_j which is not penalized by K_j whereas the vector β_j^{pen} represents the deviations of the parameters β_j from the nullspace of K_j.

In general, the decomposition $K_j = L_jL_j'$ of K_j can be obtained from the spectral decomposition $K_j = \Gamma_j\Omega_j\Gamma_j'$. The $(rk_j \times rk_j)$ diagonal matrix Ω_j contains the positive eigenvalues ω_{jm}, $m = 1, \ldots, rk_j$, of K_j in descending order, i.e. $\Omega_j = diag(\omega_{j1}, \ldots, \omega_{j,rk_j})$. Γ_j is a $(M_j \times rk_j)$ orthogonal matrix of the corresponding eigenvectors. From the spectral decomposition we can choose $L_j = \Gamma_j\Omega_j^{\frac{1}{2}}$. In some cases a more favorable decomposition can be found. For instance, for P-splines defined in subsection 7.2.1 a more favorable choice for L_j is given by $L_j = D'$ where D is the first or

second order difference matrix. Of course, for (the "random effects") prior (7.13) of subsection 7.2.3 a decomposition of $K_j = I$ is not necessary. Also, the unpenalized part vanishes completely.

The matrix X_j^{unp} is the identity vector **1** for P-splines with first order random walk penalty and Markov random fields. For P-splines with second order random walk penalty X_j^{unp} is a two column matrix whose first column is again the identity vector and the second column is composed of the (equidistant) knots of the spline.

From the decomposition (7.16) we get

$$\frac{1}{\tau_j^2}\beta_j' K_j \beta_j = \frac{1}{\tau_j^2}(\beta_j^{pen})' \beta_j^{pen}.$$

From the general prior (7.7) for β_j it follows that

$$p(\beta_{jm}^{unp}) \propto const, \qquad m = 1, \ldots, M_j - rk_j$$

and

$$\beta_j^{pen} \sim N(0, \tau_j^2 I). \tag{7.17}$$

Finally, by defining the matrices $\tilde{U}_j = X_j X_j^{unp}$ and $\tilde{X}_j = X_j X_j^{pen}$, we can rewrite the predictor (7.6) as

$$\eta = \sum_{j=1}^{p} X_j\beta_j + U\gamma = \sum_{j=1}^{p}(\tilde{U}_j\beta_j^{unp} + \tilde{X}_j\beta_j^{pen}) + U\gamma = \tilde{U}\beta^{unp} + \tilde{X}\beta^{pen}.$$

The design matrix $\tilde{X}$ and the vector β^{pen} are composed of the matrices $\tilde{X}_j$ and the vectors β_j^{pen}, respectively. More specifically, we obtain $\tilde{X} = (\tilde{X}_1\ \tilde{X}_2\ \cdots\ \tilde{X}_p)$ and the stacked vector $\beta^{pen} = ((\beta_1^{pen})', \ldots, (\beta_p^{pen})')'$. Similarly the matrix $\tilde{U}$ and the vector β^{unp} are given by $\tilde{U} = (\tilde{U}_1\ \tilde{U}_2\ \cdots\ \tilde{U}_p\, U)$ and $\beta^{unp} = ((\beta_1^{unp})', \ldots, (\beta_p^{unp})', \gamma')'$.

Finally, we obtain a GLMM with fixed effects β^{unp} and random effects $\beta^{pen} \sim N(0, \Lambda)$ where $\Lambda = diag(\tau_1^2, \ldots, \tau_1^2, \ldots, \tau_p^2, \ldots, \tau_p^2)$. Hence, we can utilize GLMM methodology for simultaneous estimation of smooth functions and the variance parameters τ_j^2, see the next section.

The mixed model representation also enables us to examine the identification problem inherent to nonparametric regression from a different angle. Except for i.i.d. Gaussian effects (7.13), the design matrices $\tilde{U}_j$ for the unpenalized parts contain the identity vector. Provided that there is at least one nonlinear effect and that γ contains an intercept, the matrix $\tilde{U}$ has not full column rank. Hence, all identity vectors in $\tilde{U}$ except for the intercept must be deleted to guarantee identifiability.

7.3 Inference

BayesX provides two alternative approaches for Bayesian inference. *bayesreg objects* (chapter 8) estimate STAR models using MCMC simulation techniques described in subsection 7.3.1. *remlreg objects* (chapter 9) use mixed model representations of STAR models for empirical Bayesian inference, see subsection 7.3.2.

7.3.1 Full Bayesian inference based on MCMC techniques

This section may be skipped if you are not interested in using the regression tool for full Bayesian inference based on MCMC techniques (*bayesreg objects* in chapter 8).

For full Bayesian inference, the unknown variance parameters τ_j^2 are also considered as random and estimated simultaneously with the unknown regression parameters β_j. Therefore, hyperpriors

are assigned to the variances τ_j^2 in a further stage of the hierarchy by highly dispersed (but proper) inverse Gamma priors $p(\tau_j^2) \sim IG(a_j, b_j)$. The probability density is given by

$$\tau_j^2 \propto (\tau_j^2)^{-a_j-1} \exp\left(-\frac{b_j}{\tau_j^2}\right).$$

The prior for τ_j^2 must not be diffuse in order to obtain a proper posterior for β_j. A common choice for the hyperparameters are small values for a_j and b_j, e.g. $a_j = b_j = 0.001$ which is also the default in *BayesX*.

In some situations, the estimated nonlinear functions f_j may considerably depend on the particular choice of hyperparameters a_j and b_j. 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_j and b_j 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.

Bayesian inference is based on the posterior of the model which is given by

$$p(\beta_1, \ldots, \beta_p, \tau_1^2, \ldots, \tau_p^2, \gamma|y) \quad \propto \quad L(y, \beta_1, \ldots, \beta_p, \gamma) \prod_{j=1}^{p} \left(p(\beta_j|\tau_j^2)p(\tau_j^2)\right) \tag{7.18}$$

where $L(\cdot)$ denotes the likelihood which is the product of individual likelihood contributions.

In many practical situations (as is the case here) the posterior distribution is numerically intractable. A common technique to overcome these problems are Markov Chain Monte Carlo (MCMC) simulation methods that have become very popular recently. MCMC methods allow the drawing of random numbers from the numerically intractable posterior distribution and in this way the estimation of characteristics of the posterior like means, standard deviations or quantiles via their empirical analogues. The main idea is very simple. Instead of drawing directly from the posterior (which is impossible in most cases anyway) a Markov chain is created, whose iterations of the transition kernel converge to the posterior. In this way a sample of dependent random numbers of the posterior is obtained. As a rule, the first part of the sample is discarded to take into account the time the algorithm needs for convergence to the posterior. This part is known as burn-in period. In *BayesX* the user has some control over the MCMC simulations to fit a certain model by specifying certain options. Among others, there are options for specifying the number of burn-in iterations and the total number of iterations, see chapter 8 for more details.

BayesX provides a number of different sampling schemes, which depend mainly on the distribution of the response. The first sampling scheme is for Gaussian responses. The second sampling scheme is particularly useful for (multi)categorical responses and uses the sampling scheme for Gaussian responses as a building block. The third sampling scheme is based on IWLS proposals and is used for general responses from an exponential family. A last sampling scheme is based on conditional prior proposals. We start with the sampling scheme for Gaussian responses.

7.3.1.1 Gaussian responses

Suppose first that the distribution of the response variable is Gaussian, i.e. $y_i|\eta_i, \sigma^2 \sim N(\eta_i, \sigma^2/c_i)$, $i = 1, \ldots, n$ or $y|\eta, \sigma^2 \sim N(\eta, \sigma^2 C^{-1})$ where $C = diag(c_1, \ldots, c_n)$ is a known weight matrix. In this case full conditionals for fixed effects as well as nonlinear functions f_j are multivariate Gaussian. Thus a Gibbs sampler can be used where posterior samples are drawn directly from the multivariate Gaussian distributions. The full conditional $\gamma|\cdot$ for fixed effects with diffuse priors is Gaussian with mean

$$E(\gamma|\cdot) = (U'CU)^{-1}U'C(y - \tilde{\eta}) \tag{7.19}$$

and covariance matrix

$$Cov(\gamma|\cdot) = \sigma^2(U'CU)^{-1} \tag{7.20}$$

where U is the design matrix of fixed effects and $\tilde{\eta}$ is the part of the additive predictor associated with the other effects in the model (for example nonparametric terms). Similarly, the full conditional for the regression coefficients β_j of a function f_j is Gaussian with mean

$$m_j = E(\beta_j|\cdot) = \left(\frac{1}{\sigma^2}X_j'CX_j + \frac{1}{\tau_j^2}K_j\right)^{-1}\frac{1}{\sigma^2}X_j'C(y-\tilde{\eta}) \tag{7.21}$$

and covariance matrix

$$Cov(\beta_j|\cdot) = P_j^{-1} = \left(\frac{1}{\sigma^2}X_j'CX_j + \frac{1}{\tau_j^2}K_j\right)^{-1}. \tag{7.22}$$

Although the full conditional is Gaussian, drawing random samples in an efficient way is not trivial, since linear equation systems with a high dimensional precision matrix P_j must be solved in every iteration of the MCMC scheme. Following Rue (2001), drawing random numbers from $p(\beta_j|\cdot)$ is as follows: We first compute the Cholesky decomposition $P_j = LL'$. We proceed by solving $L'\beta_j = z$, where z is a vector of independent standard Gaussians. It follows that $\beta_j \sim N(0, P_j^{-1})$. We then compute the mean m_j by solving $P_j m_j = \frac{1}{\sigma^2}X_j'C(y-\tilde{\eta})$. This is achieved by first solving $L\nu = \frac{1}{\sigma^2}X_j'C(y-\tilde{\eta})$ by forward substitution followed by backward substitution $L'm_j = \nu$. Finally, adding m_j to the previously simulated β_j yields $\beta_j \sim N(m_j, P_j^{-1})$.

In all cases, the posterior precision matrices P_j can be brought into a band matrix like structure with bandsize depending on the prior. If f_j corresponds to a spatially correlated effect, the posterior precision matrix is usually a sparse matrix but not a band matrix. In this case the regions of a geographical map must be *reordered*, using the *reverse Cuthill-McKee algorithm*, to obtain a band matrix like precision matrix. Random samples from the full conditional can now be drawn in a very efficient way using Cholesky decompositions for band matrices or band matrix like matrices. In our implementation we use the *envelope method* for band matrix like matrices as described in George and Liu (1981).

The full conditionals for the variance parameters τ_j^2, $j = 1, \ldots, p$, and σ^2 are all inverse Gamma distributions with parameters

$$a_j' = a_j + \frac{rank(K_j)}{2} \quad \text{and} \quad b_j' = b_j + \frac{1}{2}\beta_j'K_j\beta_j \tag{7.23}$$

for τ_j^2. For σ^2 we obtain

$$a_\sigma' = a_\sigma + \frac{n}{2} \quad \text{and} \quad b_\sigma' = b_\sigma + \frac{1}{2}\epsilon'\epsilon \tag{7.24}$$

where ϵ is the usual vector of residuals.

We note that in *BayesX* the response variable is standardized prior to estimation in order to avoid numerical problems with too large or too small values of the response. All results are, however, retransformed into the original scale.

We finally summarize the sampling scheme for Gaussian responses:

Sampling scheme 1:

1. *Initialization:*
   Compute the posterior mode for $\beta_1, \ldots, \beta_p$ and γ given fixed (usually small) smoothing parameters $\lambda_j = \sigma^2/\tau_j^2$, by default *BayesX* uses $\lambda_j = 0.1$. This value may be changed by the user. The mode is computed via backfitting. Use the posterior mode estimates as the current state β_j^c, $(\tau_j^2)^c$, γ^c of the chain.

2. *Update regression parameters* γ
   Update regression parameters γ by drawing from the Gaussian full conditional with mean and covariance in (7.19) and (7.20).

3. *Update regression parameters* β_j
   Update β_j for $j = 1, \ldots, p$ by drawing from the Gaussian full conditionals with mean and covariance matrix given in (7.21) and (7.22).

4. *Update variance parameters* τ_j^2 *and* σ^2
   Update variance parameters by drawing from inverse gamma full conditionals with parameters given in (7.23) and (7.24).

7.3.1.2 (Multi)categorical Response

For most models with categorical responses efficient sampling schemes based on latent utility representations can be developed. The seminal paper by Albert and Chib (1993) develops algorithms for probit models with ordered categorical responses. The case of probit models with unordered multicategorical responses is dealt with e.g. in Fahrmeir and Lang (2001b). Recently, another important data augmentation approach for binary logit models has been presented by Holmes and Knorr-Held (2003). The adaption of these sampling schemes to STAR models used in *BayesX* is more or less straightforward. We briefly illustrate the concept for binary data, i.e. y_i takes only the values 0 or 1. We first assume a probit model. Conditional on the covariates and the parameters, y_i follows a Bernoulli distribution $y_i \sim B(1, \mu_i)$ with conditional mean $\mu_i = \Phi(\eta_i)$ where Φ is the cumulative distribution function of a standard normal distribution. Introducing latent variables

$$L_i = \eta_i + \epsilon_i, \tag{7.25}$$

with $\epsilon_i \sim N(0, 1)$, we define $y_i = 1$ if $L_i > 0$ and $y_i = 0$ if $L_i < 0$. It is easy to show that this corresponds to a binary probit model for the y_i's. The posterior of the model augmented by the latent variables depends now on the additional parameters L_i. Thus, an additional sampling step for updating the L_i's is required. Fortunately, sampling the L_i's is relatively easy and fast because the full conditionals are truncated normal distributions. More specifically, $L_i|\cdot \sim N(\eta_i, 1)$ truncated at the left by 0 if $y_i = 1$ and truncated at the right if $y_i = 0$. The advantage of defining a probit model through the latent variables L_i is that the full conditionals for the regression parameters β_j (and γ) are Gaussian with precision matrix and mean given by

$$P_j = X_j' X_j + \frac{1}{\tau_j^2} K_j, \quad m_j = P_j^{-1} X_j' (L - \tilde{\eta}). \tag{7.26}$$

Hence, the efficient and fast sampling schemes for Gaussian responses can be used with slight modifications. Updating of β_j and γ can be done exactly as described in sampling scheme 1 using the current values L^c of the latent utilities as (pseudo) responses and setting $\sigma^2 = 1$, $C = I$.

For binary logit models, the sampling schemes become slightly more complicated. A logit model can be expressed in terms of latent utilities by assuming $\epsilon_i \sim N(0, \lambda_i)$ in (7.25) with $\lambda_i = 4\psi_i^2$, where ψ_i follows a Kolmogorov-Smirnov distribution (Devroye, 1986). Hence, ϵ_i is a scale mixture of normal form with a marginal logistic distribution (Andrews and Mallows, 1974). The full conditionals for the $L_i's$ are still truncated normals with $L_i|\cdot \sim N(\eta_i, \lambda_i)$ but additional drawings from the conditional distributions of λ_i are necessary. Drawing random numbers from the λ_i's is quite complicated, see Holmes and Knorr-Held (2003) for details.

Similar updating schemes may be developed for multinomial probit models with unordered categories and cumulative threshold models for ordered categories of the response, see Fahrmeir and

Lang (2001b) for details. *BayesX* supports both models. The cumulative threshold model is, however, restricted to three response categories. For multinomial logit models updating schemes based on latent utilities are not available.

7.3.1.3 General uni- or multivariate response from an exponential family

Let us now turn our attention to general responses from an exponential family (7.1). In this case the full conditionals are no longer Gaussian, so that more refined algorithms are needed.

BayesX supports several updating schemes based on *IWLS (iteratively weighted least squares) proposals* as proposed by Gamerman (1997) in the context of generalized linear mixed models. As an alternative *conditional prior proposals* as proposed by Knorr-Held (1999) for estimating dynamic models may be used.

The basic idea behind IWLS proposals is to combine Fisher scoring or IWLS (e.g. Fahrmeir and Tutz, 2001) for estimating regression parameters in generalized linear models, and the Metropolis-Hastings algorithm. More precisely, the goal is to approximate the full conditionals of regression parameters β_j and γ by a Gaussian distribution, obtained by accomplishing *one* Fisher scoring step in every iteration of the sampler. Suppose we want to update the regression coefficients β_j of the function f_j with current state β_j^c of the chain. Then, according to IWLS, a new value β_j^p is proposed by drawing a random number from the multivariate Gaussian proposal distribution $q(\beta_j^c, \beta_j^p)$ with precision matrix and mean

$$P_j = X_j' W(\beta_j^c) X_j + \frac{1}{\tau_j^2} K_j, \quad m_j = P_j^{-1} X_j' W(\beta_j^c)(\tilde{y}(\beta_j^c) - \tilde{\eta}). \tag{7.27}$$

Here, $W(\beta_j^c) = diag(w_1, \ldots, w_n)$ is the usual weight matrix for IWLS with weights $w_i^{-1}(\beta_j^c) = b"(\theta_i)\{g'(\mu_i)\}^2$ obtained from the current state β_j^c. The vector $\tilde{\eta}$ is the part of the predictor associated with all remaining effects in the model. The working observations $\tilde{y}_i$ are defined as

$$\tilde{y}_i(\beta_j^c) = \eta_i + (y_i - \mu_i) g'(\mu_i).$$

The sampling scheme may be summarized as follows:

Sampling scheme 2 (IWLS-proposals):

1. *Initialization*
   Compute the posterior mode for $\beta_1, \ldots, \beta_p$ and γ given fixed smoothing parameters $\lambda_j = 1/\tau_j^2$. By default, *BayesX* uses $\lambda_j = 0.1$ but the value may be changed by the user. The mode is computed via backfitting within Fisher scoring. Use the posterior mode estimates as the current state β_j^c, $(\tau_j^2)^c$, γ^c of the chain.

2. *Update* γ
   Draw a proposed new value γ^p from the Gaussian proposal density $q(\gamma^c, \gamma^p)$ with mean

   $$m_\gamma = (U'W(\gamma^c)U)^{-1} U'W(\gamma^c)(y - \tilde{\eta})$$

   and precision matrix

   $$P_\gamma = U'W(\gamma^c)U.$$

   Accept γ^p as the new state of the chain γ^c with acceptance probability

   $$\alpha = \frac{L(y, \ldots, \gamma^p)}{L(y, \ldots, \gamma^c)} \frac{q(\gamma^p, \gamma^c)}{q(\gamma^c, \gamma^p)},$$

   otherwise keep γ^c as the current state.

3. *Update* β_j
   Draw for $j = 1, \ldots, p$ a proposed new value β_j^p from the Gaussian proposal density $q(\gamma^c, \gamma^p)$ with mean and precision matrix given in (7.27). Accept β_j as the new state of the chain β_j^c with probability

$$\alpha = \frac{L(y, \ldots, \beta_j^p, (\tau_j^2)^c, \ldots, \gamma^c)}{L(y, \ldots, \beta_j^c, (\tau_j^2)^c, \ldots, \gamma^c)} \frac{p(\beta_j^p \mid (\tau_j^2)^c)}{p(\beta_j^c \mid (\tau_j^2)^c)} \frac{q(\beta_j^p, \beta_j^c)}{q(\beta_j^c, \beta_j^p)},$$

   otherwise keep β_j^c as the current state.

4. *Update* τ_j^2
   Update variance parameters by drawing from inverse gamma full conditionals with parameters given in (7.23).

A slightly different updating scheme computes the mean and the precision matrix of the proposal distribution based on the current posterior mode m_j^c (from the last iteration) rather than the current β_j^c, i.e. (7.27) is replaced by

$$P_j = X_j' W(m_j^c) X_j + \frac{1}{\tau_j^2} K_j, \quad m_j = P_j^{-1} X_j' W(m_j^c)(\tilde{y}(\beta_j^c) - \tilde{\eta}). \tag{7.28}$$

The difference of using m_j^c rather than β_j^c is that the proposal is *independent* of the current state of the chain, i.e. $q(\beta_j^c, \beta_j^p) = q(\beta_j^p)$. Hence, it is not required to recompute P_j and m_j when computing the proposal density $q(\beta_j^p, \beta_j^c)$.

Usually acceptance rates are significantly higher compared to sampling scheme 2. This is particularly useful for updating spatial effects based on Markov random fields because in many cases sampling scheme 2 yields quite low acceptance rates. We summarize the sampling scheme as follows:

Sampling scheme 3 (IWLS-proposals based on current mode):

1. *Initialization*
   Compute the posterior mode for $\beta_1, \ldots, \beta_p$ and γ given fixed smoothing parameters $\lambda_j = 1/\tau_j^2$. By default, *BayesX* uses $\lambda_j = 0.1$ but the value may be changed by the user. The mode is computed via backfitting within Fisher scoring. Use the posterior mode estimates as the current state β_j^c, $(\tau_j^2)^c$, γ^c of the chain. Define m_j^c and m_γ^c as the current mode.

2. *Update* γ
   Draw a proposed new value γ^p from the Gaussian proposal density $q(\gamma^c, \gamma^p)$ with mean

$$m_\gamma = (U' W(m_\gamma^c) U)^{-1} U' W(m_\gamma^c)(y - \tilde{\eta})$$

   and precision matrix

$$P_\gamma = U' W(m_\gamma^c) U.$$

   Accept γ^p as the new state of the chain γ^c with acceptance probability

$$\alpha = \frac{L(y, \ldots, \gamma^p)}{L(y, \ldots, \gamma^c)} \frac{q(\gamma^p, \gamma^c)}{q(\gamma^c, \gamma^p)},$$

   otherwise keep γ^c as the current state.

3. *Update* β_j
   Draw for $j = 1, \ldots, p$ a proposed new value β_j^p from the Gaussian proposal density $q(\beta_j^c, \beta_j^p)$

with mean and precision matrix given in (7.28). Accept β_j^p as the new state of the chain β_j^c with probability

$$\alpha = \frac{L(y,\ldots,\beta_j^p,(\tau_j^2)^c,\ldots,\gamma^c)}{L(y,\ldots,\beta_j^c,(\tau_j^2)^c,\ldots,\gamma^c)} \frac{p(\beta_j^p \mid (\tau_j^2)^c)}{p(\beta_j^c \mid (\tau_j^2)^c)} \frac{q(\beta_j^p,\beta_j^c)}{q(\beta_j^c,\beta_j^p)},$$

otherwise keep β_j^c as the current state.

4. *Update* τ_j^2
   Update variance parameters by drawing from inverse gamma full conditionals with parameters given in (7.23).

7.3.2 Empirical Bayesian inference based on GLMM methodology

This section may be skipped if you are not interested in using the regression tool based on mixed model methodology (*remlreg objects* in chapter 9).

For empirical Bayes inference variances τ_j^2 are considered as constants. In terms of the GLMM representation outlined in subsection 7.2.5 the posterior is given by

$$p(\beta^{unp},\beta^{pen}|y) \quad \propto \quad L(y,\beta^{unp},\beta^{pen}) \prod_{j=1}^{p} \left(p(\beta_j^{pen}|\tau_j^2) \right) \tag{7.29}$$

where $p(\beta_j^{pen}|\tau_j^2)$ is defined in (7.17).

Based on the GLMM representation, regression and variance parameters can be estimated using iteratively weighted least squares (IWLS) and (approximate) restricted maximum likelihood (REML) developed for GLMM's. Estimation is carried out iteratively in two steps. The two steps for updating estimates are:

1. Obtain updated estimates $\hat{\beta}^{unp}$ and $\hat{\beta}^{pen}$ given the current variance parameters as the solutions of the linear equation system

$$\begin{pmatrix} \tilde{U}'W\tilde{U} & \tilde{U}'W\tilde{X} \\ \tilde{X}'W\tilde{U} & \tilde{X}'W\tilde{X} + \tilde{\Lambda}^{-1} \end{pmatrix} \begin{pmatrix} \beta^{unp} \\ \beta^{pen} \end{pmatrix} = \begin{pmatrix} \tilde{U}'W\tilde{y} \\ \tilde{X}'W\tilde{y} \end{pmatrix}. \tag{7.30}$$

   The $(n \times 1)$ vector $\tilde{y}$ and the $n \times n$ diagonal matrix $W = diag(w_1,\ldots,w_n)$ are the usual working observations and weights in generalized linear models, see Fahrmeir and Tutz (2001), Chapter 2.2.1.

2. Updated estimates for the variance parameters $\hat{\tau}_j^2$ are obtained by maximizing the (approximate) restricted log likelihood

$$\begin{aligned} l^*(\tau_1^2,\ldots,\tau_p^2) \quad = \quad & -\tfrac{1}{2}\log(|\Sigma|) - \tfrac{1}{2}\log(|\tilde{U}\Sigma^{-1}\tilde{U}|) \\ & -\tfrac{1}{2}(\tilde{y}-\tilde{U}\hat{\beta}^{unp})'\Sigma^{-1}(\tilde{y}-\tilde{U}\hat{\beta}^{unp}) \end{aligned} \tag{7.31}$$

   with respect to the variance parameters $\tau_1^2,\ldots,\tau_p^2$. Here, $\Sigma = W^{-1} + \tilde{X}\Lambda\tilde{X}'$ is an approximation to the marginal covariance matrix of $\tilde{y}|\beta^{pen}$.

The two estimation steps are iterated until convergence. We maximize (7.31) through a computationally efficient alternative to the usual Fisher scoring iterations as described e.g. in Harville (1977), see Fahrmeir, Kneib and Lang (2003) for details.

Note, that convergence problems may occur, if one of the parameters τ_j^2 is small. In this case the maximum of the restricted likelihood may be on the boundary of the parameter space so that Fisher scoring is no appropriate way to find the REML-estimates $\hat{\tau}^2$. Therefore, the estimation of small variances τ_j^2 is stopped in the current implementation, if the criterion

$$c(\tau_j^2) = \frac{||\tilde{X}_j \hat{\beta}_j^{pen}||}{||\hat{\eta}||} \tag{7.32}$$

is smaller than the user-specified value `lowerlim` (see subsection 9.1.2). This usually corresponds to small values of the variances τ_j^2 but defines "small" in a data driven way.

7.4 Survival analysis and competing risks models

Discrete time duration and competing risks models can be estimated by expressing such models as categorical regression models. This is sketched in subsection 7.4.1 while subsection 7.4.2 deals with continuous time survival analysis. *BayesX* offers two possibilities of estimating continuous time hazard rate models, the piecewise exponential model and nonparametric as well as spatial extensions of the well known Cox model.

7.4.1 Discrete time duration data

In applications, duration data are often measured on a discrete time scale or are grouped in intervals. In this section we show how data of this kind can be written as categorical regression models. Estimation is then based on methodology for categorical regression models as described in the previous sections. We start by assuming that there is only one type of failure event.

Let the duration time scale be divided into intervals $[a_0, a_1), [a_1, a_2), \ldots, [a_{q-1}, a_q), [a_q, a_\infty)$. Usually $a_0 = 0$ is assumed and a_q denotes the final follow up. Identifying the discrete time index t with interval $[a_{t-1}, a_t)$, duration time T is considered as discrete, where $T = t \in \{1, \ldots, q+1\}$ denotes end of duration within the interval $t = [a_{t-1}, a_t)$. In addition to duration T, a sequence of possibly time-varying covariate vectors u_t is observed. Let $u_t^* = (u_1, \ldots, u_t)$ denote the history of covariates up to interval t. Then the discrete hazard function is given by

$$\lambda(t; u_t^*) = \text{pr}(T = t \mid T \geq t, u_t^*), \quad t = 1, \ldots, q,$$

that is the conditional probability for the end of duration in interval t, given that the interval is reached and the history of the covariates. Hazard functions are usually specified by binary response models. Common choices are binary logit, probit or grouped Cox models. So far *BayesX* supports only logit and probit models.

For a sample of individuals i, $i = 1, \ldots, n$, let T_i denote duration times and C_i right censoring times. Duration data are usually given by $(t_i, \delta_i, u_{it_i}^*)$, $i = 1, \ldots, n$, where $t_i = \min(T_i, C_i)$ is the observed discrete duration time, $\delta_i = 1$ if $T_i < C_i$, $\delta_i = 0$ else is the censoring indicator, and $u_{it_i}^* = (u_{it}, t = 1, \ldots, t_i)$ is the observed covariate sequence. We assume that censoring is noninformative and occurs at the end of the interval, so that the risk set R_t includes all individuals who are censored in interval t.

We define binary event indicators y_{it}, $i \in R_t$, $t = 1, \ldots, t_i$, by

$$y_{it} = \begin{cases} 1 & \text{if } t = t_i \text{ and } \delta_i = 1 \\ 0 & \text{else.} \end{cases}$$

Then the duration process of individual i can be considered as a sequence of binary decisions between remaining in the transient state $y_{it} = 0$ or leaving for the absorbing state $y_{it} = 1$, i.e.

end of duration at t. For $i \in R_t$, the hazard function for individual i can be modelled by binary response models

$$\text{pr}(y_{it} = 1 \mid u_{it}^*) = h(\eta_{it}), \tag{7.33}$$

with appropriate predictor η_{it} and response function $h : \mathbf{R} \to (0, 1)$. *BayesX* supports the response functions $h(\cdot) = \exp(\cdot)/\{1 + \exp(\cdot)\}$ for the logit model and $h(\cdot) = \Phi(\cdot)$ for the probit model. Traditionally a linear predictor is assumed, i.e.

$$\eta_{it} = \gamma_0(t) + u_{it}'\gamma. \tag{7.34}$$

The sequence $\gamma_0(t)$, $t = 1, \ldots, q$, of parameters represents the baseline effect. In *BayesX* we may assume a structured additive predictor

$$\eta_{it} = f_0(t) + f_1(x_{it1}) + \ldots + f_p(x_{itp}) + u_{it}'\gamma. \tag{7.35}$$

Again, the x_j denote generic covariates of different types and dimension, and f_j are (not necessarily smooth) functions of the covariates. The baseline effect $f_0(t)$ may be modelled by random walk priors or P-splines.

We demonstrate with an example how discrete time survival data must be manipulated such that binary logit or probit models may be used for estimation. Suppose the first few observations of a data set are given as follows:

t	δ	x1	x2
4	0	0	2
3	1	1	0
⋮	⋮	⋮	⋮

The first individual is censored ($\delta = 0$) and the observed duration time is 4. The second individual is uncensored with duration time 3. Now we augment the data set as follows:

y	indnr	t	δ	x1	x2
0	1	1	0	0	2
0	1	2	0	0	2
0	1	3	0	0	2
0	1	4	0	0	2
0	2	1	1	1	0
0	2	2	1	1	0
1	2	3	1	1	0
⋮	⋮	⋮	⋮	⋮	⋮

For the first individual we have now 4 observations (because the observed duration time is 4). The event indicator y is always 0 because of the censoring. For the second individual we obtain 3 observations and the event indicator jumps at time t=3 from 0 to 1. Now we can estimate a logit or probit model with y as the response and covariates t, x1, x2.

So far we have only considered situations with one type of failure. Suppose now that we may distinguish several types of failure. E.g. in a study on unemployment durations Fahrmeir and Lang (2001b) distinguished between full- and part time jobs that end unemployment duration. Models of this kind are often referred to as competing risks models.

Let $R \in \{1, \ldots, m\}$ denote distinct events of failure. Then the cause specific discrete hazard function resulting from cause or risk r is given by

$$\lambda_r(t|u_t, x_t) = P(T = t, R = r|T \geq t, u_t, x_t).$$

Modelling $\lambda_r(t|u_t, x_t)$ may be based on multicategorial regression models. E.g. assuming a multinomial logit model yields

$$\lambda_r(t|u_t) = \frac{\exp(\eta_r)}{1 + \sum_{s=1}^{m} \exp(\eta_s)}$$

with structured additive predictors

$$\eta_r = f_{0r}(t) + f_{1r}(x_{t1}) + \ldots + f_{pr}(x_{tp}) + u_t'\gamma_r. \tag{7.36}$$

An alternative is the multinomial probit model.

The following example demonstrates how data with several types of failure must be manipulated such that multicategorical regression models may be used for estimation. Suppose we have data with 2 terminating events R=1 and R=2. The first few observations of a data set are given as follows:

t	δ	R	x1	x2
4	0	1	0	2
3	1	2	1	0
5	1	1	0	3
⋮	⋮	⋮	⋮	⋮

The first individual is censored ($\delta = 0$) and the observed duration time is 4. The second individual is uncensored with duration time 3 and terminating event R=2. The third individual is uncensored with duration time 5 and terminating event R=1. We augment the data set as follows:

y	indnr	t	δ	x1	x2
0	1	1	0	0	2
0	1	2	0	0	2
0	1	3	0	0	2
0	1	4	0	0	2
0	2	1	1	1	0
0	2	2	1	1	0
2	2	3	1	1	0
0	3	1	1	0	3
0	3	2	1	0	3
0	3	3	1	0	3
0	3	4	1	0	3
1	3	5	1	0	3
⋮	⋮	⋮	⋮	⋮	⋮

For the first individual we create 4 observations (because the observed duration time is 4). The event indicator y is always 0 because of the censoring. For the second individual we obtain 3 observations and the event indicator jumps at time t=3 from 0 to 2. For the third individual the event indicator jumps at time 5 from 0 to 1. Now we can estimate a multinomial logit or probit model with y as the response, reference category 0, and covariates t, x1, x2.

7.4.2 Continuous time survival analysis

In applications, where duration time t is measured on a continuous time scale, grouping the data for a discrete time analysis is possible, but causes a loss of information. In this section we will shortly introduce the continuous time Cox model and describe the two alternatives *BayesX* offers for the estimation of such models. 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.). Data augmentation is needed here, but estimation is then based on methodology for Poisson regression, and the inclusion of time-varying effects does not imply any difficulties. The second alternative is to estimate the log-baseline effect by a P-spline of arbitrary degree. This approach is less restrictive and does not demand data augmentation.

Let $u_t^* = \{u_s, 0 \leq s \leq t\}$ denote the history of possibly time-varying covariates up to time t. Then the continuous hazard function is given by

$$\lambda(t; u_t^*) = \lim_{\Delta t \downarrow 0} \frac{P(t \leq T < t + \Delta t | T \geq t, u_t^*)}{\Delta t}$$

that is the conditional instantaneous rate of end of duration at time t, given that time t is reached and the history of the covariates. In the Cox model the individual hazard rate is modelled by

$$\lambda_i(t) = \lambda_0(t) \cdot \exp(\eta_{it}) = \exp(f_0(t) + \eta_{it}) \tag{7.37}$$

where $\lambda_0(t)$ is the baseline hazard, ($f_0(t) = \log(\lambda_0(t))$ is the log-baseline hazard) and η_{it} is an appropriate predictor. Traditionally the predictor is linear and the baseline hazard is unspecified. In *BayesX* however, a structured additive predictor may be assumed and the baseline effect is estimated nonparametrically either by a piecewise constant function (i.e. assuming a p.e.m.) or by a P-spline.

Piecewise exponential model (p.e.m.)

The basic idea of the p.e.m. is to assume that all values that depend on time t are piecewise constant on a grid

$$(0, a_1], (a_1, a_2], \ldots, (a_{s-1}, a_s], \ldots, (a_{t-1}, a_t], (a_t, \infty),$$

where a_t is the largest of all observed duration times $t_i, i = 1, \ldots, n$. This grid may be equidistant or, for example, according to quantiles. The assumption of a p.e.m. is a quite convenient one as estimation can be based on methodology for Poisson regression models. For this purpose the data set has to be modified as described below.

Let δ_i be an indicator of non-censoring (i.e. $\delta_i = 1$ if observation i is uncensored, 0 else) and $\gamma_{0s}, s = 1, 2, \ldots$ the piecewise constant log-baseline effect. We define an indicator variable y_{is} as well as an offset Δ_{is} as follows:

$$y_{is} = \begin{cases} 1 & t_i \in (a_{s-1}, a_s],\ \delta_i = 1 \\ 0 & \text{else.} \end{cases}$$

$$\Delta'_{is} = \begin{cases} a_s - a_{s-1}, & a_s < t_i \\ t_i - a_{s-1}, & a_{s-1} < t_i \leq a_s \\ 0, & a_{s-1} \geq t_i \end{cases}$$

$$\Delta_{is} = \log \Delta'_{is} \quad (\Delta_{is} = -\infty \text{ if } \Delta'_{is} = 0).$$

The likelihood contribution of observation i in the interval $(a_{s-1}, a_s]$ is

$$L_{is} = \exp\left(y_{is}(\gamma_{0s} + \eta_{is}) - \exp(\Delta_{is} + \gamma_{0s} + \eta_{is})\right).$$

As this likelihood is proportional to a Poisson likelihood with offset, estimation can be executed using Poisson regression with response variable y, (log-)offset Δ and a as a continuous covariate. Due to the assumption of a piecewise constant hazard rate the estimated log-baseline is a step function on the defined grid. To get a "smooth step function" a random walk prior is specified for the parameters γ_{0s}.
In practice this means that the data set has to be modified in such a way that for every individual i there is an observation row for each interval $(a_{s-1}, a_s]$ beginning with the first one up to the interval in that duration time t_i ends. Instead of the indicator of non-censoring δ_i the modified data set contains the indicator y_{is}, instead of duration time t_i the variable a_s as well as the offset Δ_{is} (covariates are duplicated). To give a short example, if we have an equidistant grid with length 0.1 the observations

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

have to be modified to

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
⋮	⋮	⋮	⋮	⋮	⋮	⋮

We could now estimate a Poisson model with offset Δ, y as the response, a as a covariate with random walk prior and x1 and x2 as covariates with appropriate priors.

Specifying a P-spline prior for the log-baseline

The p.e.m. can be seen as a model where the log-baseline $f_0(t)$ in (7.37) is modelled by a P-spline (see (7.2.1)) of degree 0, which is quite convenient as it simplifies the calculation of the likelihood, but may be too restrictive since the baseline effect is estimated by a step-function that may not be appropriate for continuous data. A more general way of estimating the nonlinear shape of the baseline effect is to assume a P-spline prior of arbitrary degree instead. Unlike the p.e.m. such a model can not be estimated within the context of GAMs, but specific methods for extended Cox models are also implemented in *BayesX*. The individual likelihood for continuous survival data is given by

$$L_i \quad = \quad \lambda_i(t_i)^{\delta_i} \cdot \exp\left(-\int_0^{t_i} \lambda_i(u)du\right).$$

Inserting (7.37) we get

$$L_i \quad = \quad \exp(f_0(t_i) + \eta_{it_i})^{\delta_i} \cdot \exp\left(-\int_0^{t_i} \exp(f_0(u) + \eta_{iu})du\right).$$

If the degree of the P-spline prior for $f_0(t)$ is greater than one the integral can not be calculated analytically anymore. For linear P-splines the integral can still be solved but the computational effort is quite high. Therefore *BayesX* makes use of the trapezoidal rule for a numerical approximation.

7.5 References

Albert, J. and Chib, S., (1993): Bayesian analysis of binary and polychotomous response data. *Journal of the American Statistical Association*, 88, 669-679.

Andrews, D.F. and Mallows, C.L. (1974): Scale mixtures of normal distributions. *Journal of the Royal Statistical Society B*, 36, 99-102.

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

Devroye, L. (1986): *Non-Uniform Random Variate Generation.* Springer-Verlag, New York.

Fahrmeir, L. and Hennerfeind, A., (2003): Nonparametric Bayesian hazard rate models based on penalized splines. SFB 386 Discussion paper 361, University of Munich.

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 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 Osuna, L. (2003): Structured count data regression. SFB 386 Discussion paper 334, University of Munich.

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

Fotheringham, A.S., Brunsdon, C., and Charlton, M.E., 2002: *Geographically Weighted Regression: The Analysis of Spatially Varying Relationships.* Wiley, Chichester.

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 Generalized 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. (1987): Penalized likelihood for general semiparametric regression models. *International Statistical Review*, 55, 245–259.

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.

Harville, D. A. (1977): Maximum Likelihood approaches to variance component estimation and to related problems. *Journal of the American Statistical Association*, 72, 320–338.

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: Springer–Verlag.

Hennerfeind, A., Brezger, A., Fahrmeir, L. (2003): Geoadditive Survival models. SFB 386 Discussion paper 333, University of Munich.

Johnson, M.E., Moore, L.M. and Ylvisaker, D., (1990): Minimax and maximin designs. *Journal of Statistical Planning and Inference* , 26, 131–148.

Kammann, E. E. and Wand, M. P., (2003): Geoadditive Models. *Journal of the Royal Statistical Society C*, 52, 1-18.

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.

Kneib, T. and Fahrmeir, L., (2004b): A mixed model approach to structured hazard regression. SFB 386 Discussion paper 400, University of Munich.

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. and Brezger, A. (2003): Bayesian P-splines. *Journal of Computational and Graphical Statistics*, 13, 183-212.

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.

Nychka, D. and Saltzman, N., (1998): *Design of Air-Quality Monitoring Networks*, Lecture Notes in Statistics, 132, 51–76.

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.

Rue, H. and Tjelmeland, H., (2002): Fitting Gaussian Markov Random Fields to Gaussian Fields. *Scandinavian Journal of Statistics*, 29, 31-49.

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

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 chapter 7. 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 examples in section 8.6.

8.1 Method regress

8.1.1 Description

Method `regress` estimates regression or hazard rate models with Structured Additive Predictor. An introduction to the methodological background can be found in chapter 7.

8.1.2 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 8.1.2.2. 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 8.5 for an overview about the distributions supported by *BayesX*. The response distribution is specified using option `family`, see subsubsection 8.1.2.4 below and the options list in subsection 8.1.3 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.

8.1.2.1 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.

8.1.2.2 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 8.1. Table 8.2 shows how interactions between covariates are specified. Full details about all available options are given in subsubsection 8.1.2.3.

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 8.1.2.3 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 8.1.2.3.

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 using one of the Markov random field priors (7.11) or (7.12) 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` the seasonal prior (7.9) 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 the spatial smoothness prior (7.11), or the more general prior (7.12) depending on the weight definition in the *map object* `germany`.

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)
```

8.1.2.3 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 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.

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 8.1: 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 8.2: 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 8.3: 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`	$*$	$*$	$*$	$\times$	$\times$	$*$	$*$	integer
`max`	$*$	$*$	$*$	$\times$	$\times$	$*$	$*$	integer
`lambda`	realvalue	realvalue	realvalue	realvalue	realvalue	realvalue	realvalue	realvalue
`proposal`	$\bullet$	$\bullet$	$\bullet$	$\bullet$	$\circ$	$\bullet$	$\bullet$	$\times$
`updateW`	integer	integer	integer	integer	$\times$	integer	integer	$\times$
`degree`	$\times$	$\times$	integer	$\times$	$\times$	integer	integer	integer
`nrknots`	$\times$	$\times$	integer	$\times$	$\times$	integer	integer	integer
`gridsize`	$\times$	$\times$	integer	$\times$	$\times$	integer	integer	integer
`derivative`	$\times$	$\times$	$\triangle$	$\times$	$\times$	$\times$	$\times$	$\times$
`period`	$\times$	integer	$\times$	$\times$	$\times$	$\times$	$\times$	$\times$
`nofixed`	$\times$	$\times$	$\times$	$\times$	$\triangle$	$\times$	$\times$	$\times$
`map`	$\times$	$\times$	$\times$	*map object*	$\times$	*map object*	$\times$	$\times$

$\times$	not available
$*$	available only if `proposal = cp`
$\circ$	admissible values are `iwls,iwlsmode`
$\bullet$	admissible values are `cp,iwls,iwlsmode`
$\triangle$	available as boolean option (specified without supplying a value)

Table 8.4: Terms and options for bayesreg objects

8.1.2.4 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 8.1.2.5. 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. 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 8.1.2.1 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

$$p(y) \propto y^{s-1} \exp(-\frac{s}{\mu} y) \tag{8.1}$$

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 usally used in the context of Bayesian hierarchical models to specify hyperpriors for variance components. The density is then given by

$$p(y) \propto y^{a-1} \exp(-by) \tag{8.2}$$

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* a gamma distributed response is defined as 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. 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 8.1.2.1 for details on how to specify weights.

In analogy to the variance parameter in Gaussian response models, we assume a Gamma prior (second parameterization (8.2)) 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 8.1.2.1 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 8.1.1 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 8.1.2.1 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 $y_i|\eta_i, \delta, \theta \sim ZINB(\eta_i, \delta, \theta)$.

8.1.2.5 Continuous time survival analysis

BayesX offers two alternatives of estimating continuous time Cox models with semiparametric predictor η, which are described in (subsection 7.4.2). 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 7.4.2) 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 7.4.2). 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.

8.1.3 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 8.1.2.4. Table 8.5 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`).

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

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 8.5: Summary of supported response distributions.

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 8.6 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 8.6: 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.

8.1.4 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 8.1.3 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 8.1.3 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 subsection 8.4.1 and subsection 8.4.2, respectively. Of course, any other (statistics) software package with plotting facilities may be used as well.

8.1.5 Examples

Here we give only a few examples about the usage of method `regress`. More detailed examples can be found in section 8.6.

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
```

8.2 Method autocor

8.2.1 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`.

8.2.2 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 8.5. 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 8.5). 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 8.5.

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.

8.2.3 Option

- **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`

8.2.4 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 8.5). 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 subsubsection 8.4.2.4.

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.

8.3 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")
```

8.4 Visualizing estimation results

8.4.1 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. Autocorrelation functions of sampled parameters can be visualized with method plotautocor.

8.4.1.1 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 or P-splines are used as priors.

Options

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

- **connect=p|l[specifications for further variables]**

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

p	do not connect, i.e. plot points only
l	(letter l) draw straight lines between the points (default)

  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 symbol (p,l) for every *yvar*. Typing for example

  ```
  connect = lp
  ```

  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.

- **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 posterior mean of a 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"`).

- **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 *bayesreg object* `b` and have estimated a regression model with Gaussian errors using

```
> b.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

Acceptance rate:    100 %

Results are stored in file
c:\results\b_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

```
> b.plotnonp 0
```

we obtain the plot shown in Figure 8.1.

Of course, a title, axis labels etc. can be added. For example by typing

```
> b.plotnonp 0 , title="my title" xlab="x axis"
```

we obtain the plot shown in Figure 8.2.

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

```
> b.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`.

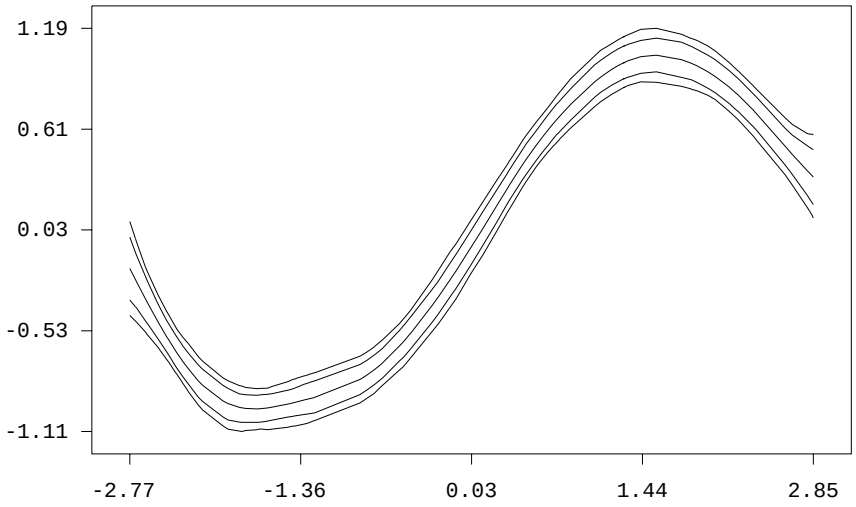


Figure 8.1: Illustration for the usage of method `plotnonp`

8.4.1.2 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*]

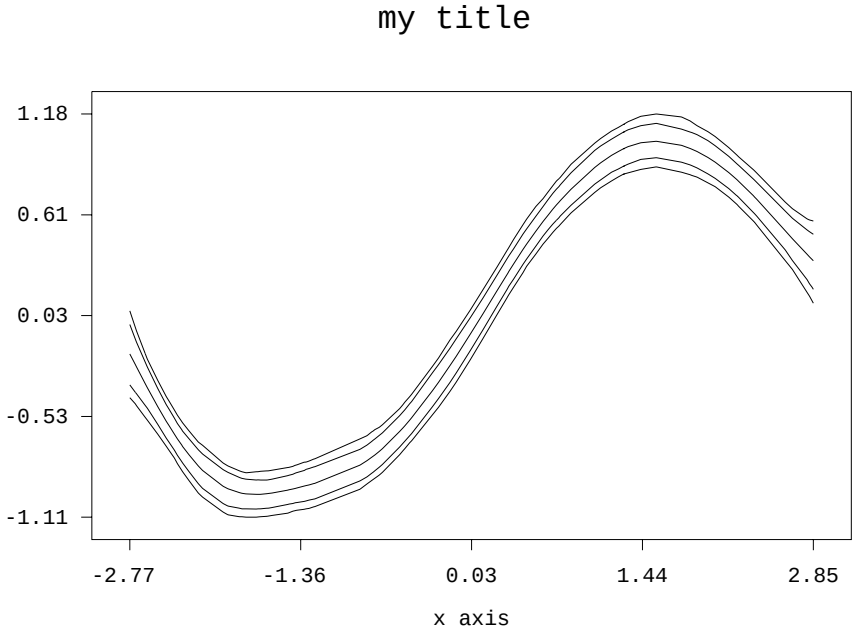


Figure 8.2: Second illustration for the usage of method `plotnonp`

Visualizes the effect of a spatial covariate by coloring the regions of the corresponding geographical map according to the estimated posterior means (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 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 posterior means. 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 *bayesreg object* `b` and have estimated a regression model with Gaussian errors using

```
> map m
> m.infile using c:\maps\map1.bnd
> b.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

  Acceptance rate:    100 %

  Results are stored in file
  c:\results\b_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

```
> b.drawmap 0
```

we obtain the map shown in Figure 8.3 where the regions are colored according to the estimated posterior mean.

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

```
> b.drawmap 0 , color title="my title"
```

we obtain the map shown in Figure 8.4.

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

```
> b.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`.

8.4.1.3 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.

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):

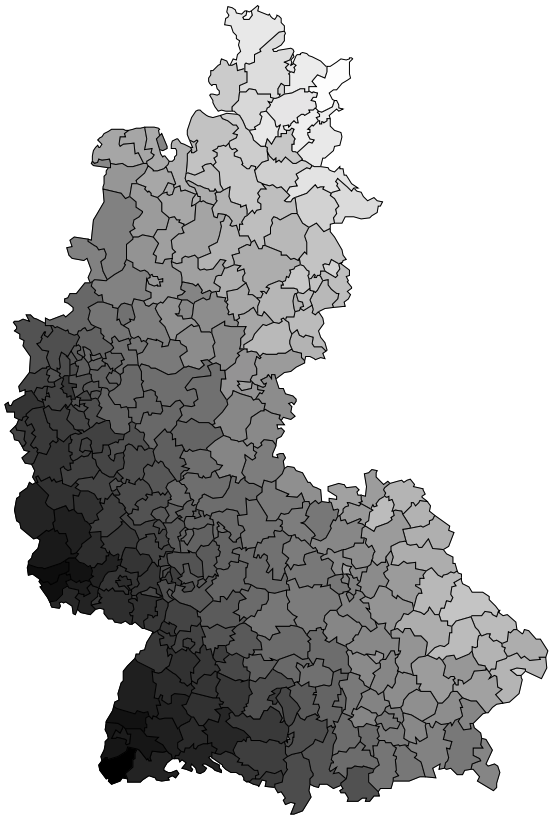


Figure 8.3: Illustration for the usage of method `drawmap`

- **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.

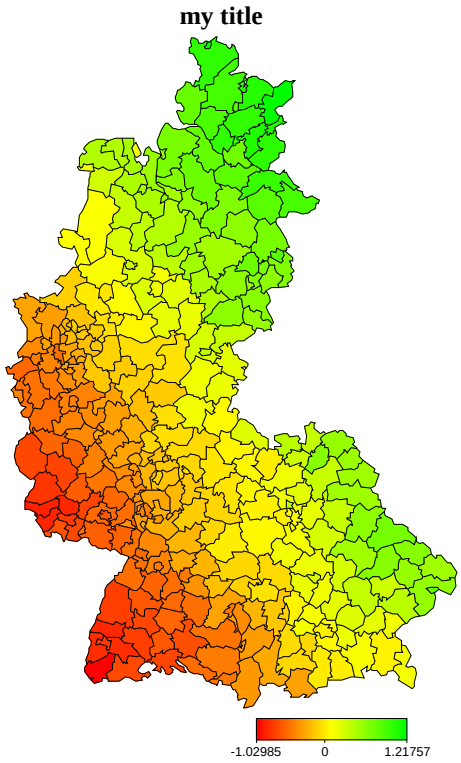


Figure 8.4: Second illustration for the usage of method `drawmap`

- **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* `b` and have estimated a regression model with Gaussian errors using

```
> b.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:

```
> b.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 8.5.

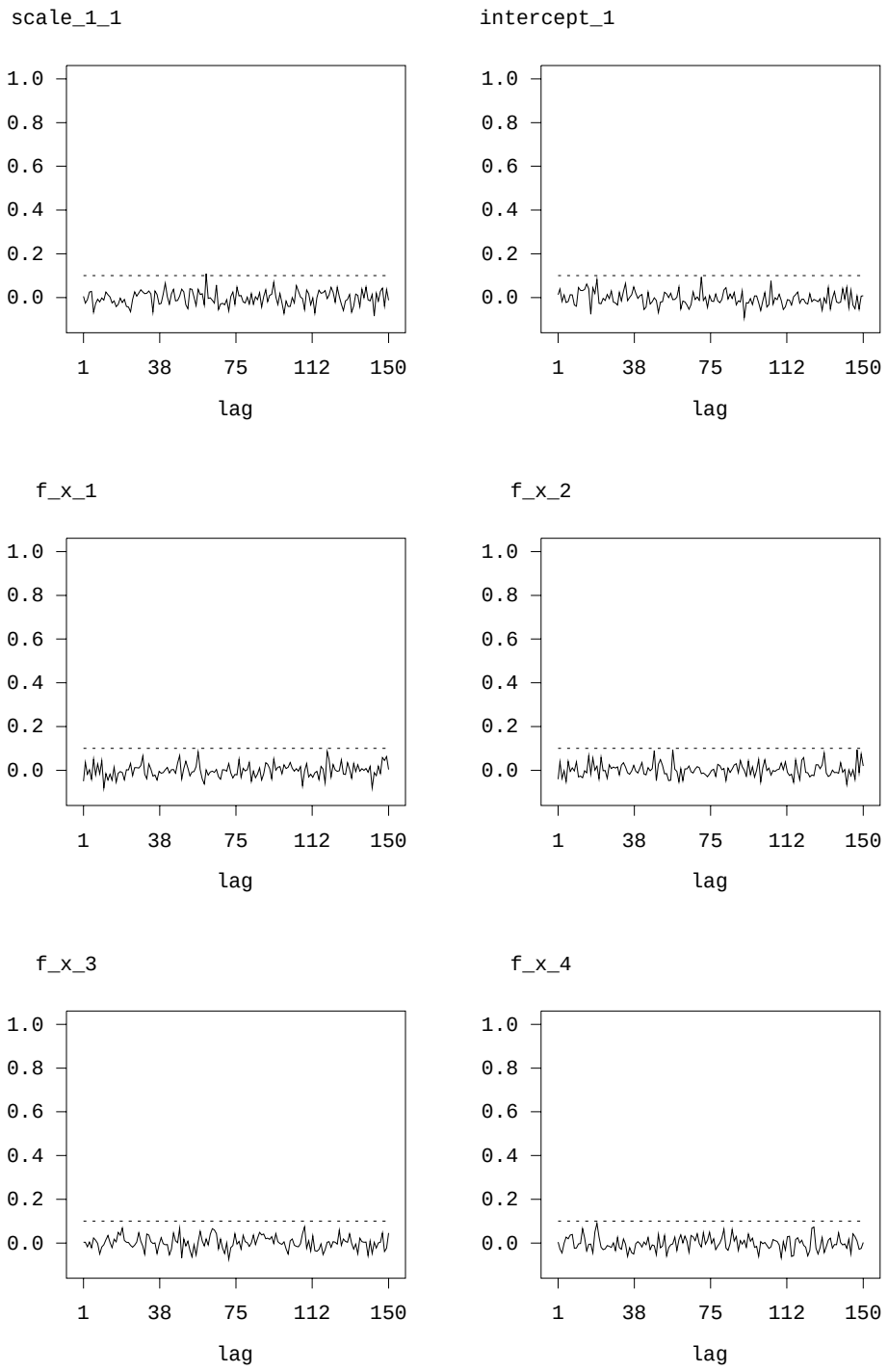


Figure 8.5: Illustration for the usage of method `plotautocor`

8.4.2 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 8.7 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 8.7: Overview over S-plus functions

8.4.2.1 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.

8.4.2.2 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 8.5), 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 *bayesreg object* and the covariate name. For the following we assume that `c:\bayes` is the installation directory and `b` is the name of the *bayesreg object*. In this case results for the effect of `X` are stored in:

`c:\bayes\output\b_f_X_rw.res`

The structure of the file has already been described in section 8.1. 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\\b_f_X_rw.res")
```

a S-plus *graphic window* will be opened with the plotted function estimate. The function plots the posterior mean 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:

```
> plotnonp("c:\\bayes\\b_f_X_rw.res", psname="c:\\bayes\\b_f_X_rw.ps",
  maintitle="Maintitle",ylab="effect of X",xlab="X")
```

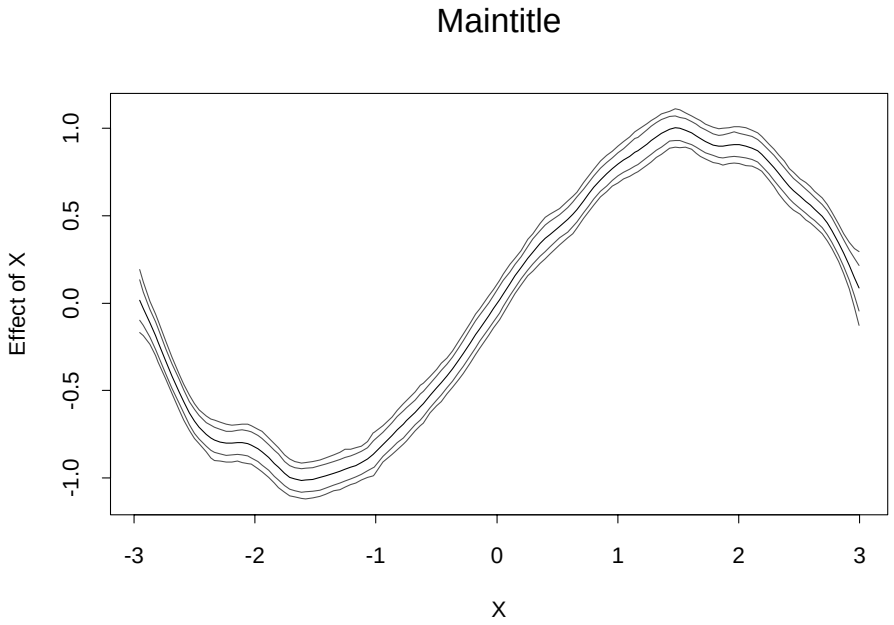


Figure 8.6: Illustration for the usage of `plotnonp`

This statement draws the estimated effect of `X` and stores the graph in the postscript file `"c:\\bayes\\b_f_X_rw.ps"`. A title, a x-axis and y-axis label is added to the graph. For illustration purposes, the resulting graph is shown in Figure 8.6.

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 8.7:

```
> plotnonp("c:\\bayes\\b_f_D_pspline.res", psname="c:\\bayes\\b_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.

8.4.2.3 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

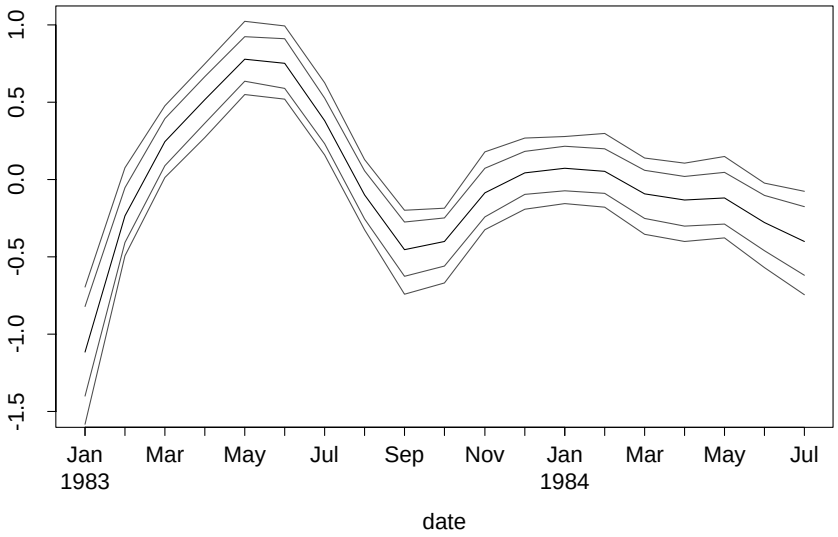


Figure 8.7: Illustration for the usage of `plotnonp`

Figure 8.8: Map of Munich

spatial covariate is given in the 'rents for flats' example, see subsection 2.5.1, where the covariate `L` indicates the location (in subquarters) of the flat in Munich. Figure 8.8 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 8.1. 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 8.8 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 8.9 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.

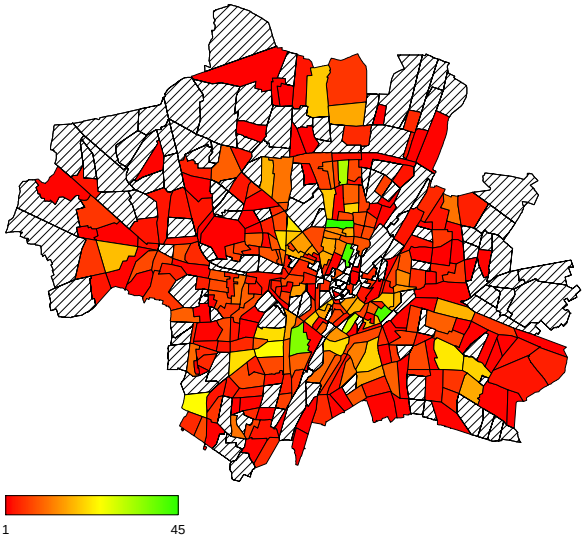


Figure 8.9: relative frequencies of observed flats in the 'rents for flats' data set

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\b_f_L_spatial.res`. The structure of the file is described in detail in section 8.1. It contains 10 columns, one column with a row counter, one column containing the values of the covariate (here the location) and four columns containing estimated effects for the values of the covariate. These are the posterior mean and the posterior 2.5%, 10%, 50%, 90% and 97.5% quantiles. The last two columns contain posterior probabilities based on nominal levels of 95% and 80%. The number 1 corresponds to a strictly positive credible interval, the number -1 corresponds to a strictly negative credible interval. A value of 0 indicates that the corresponding credible interval contains zero. The first row contains the column names. For the covariate `L` (location) for example, the first row of the file is given by:

intnr L pmean pqu2p5 pqu10 pmed pqu90 pqu97p5 pcat95 pcat80

Suppose now that we want to visualize estimation results for the spatial covariate `L` by coloring the subquarters of Munich according to the estimated posterior mean. 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\\b_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.

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`.

8.4.2.4 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 8.2 for details. For the rest of this section we assume that autocorrelations are already computed and stored in file:

`c:\bayes\output\b_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\\b_autocor.raw")
```

prints autocorrelations on the screen, while the statement

```
> plotautocor("c:\\bayes\\output\\b_autocor.raw","c:\\bayes\\output\\b_autocor.ps")
```

forces S-plus to store the autocorrelation graphs in the postscript file `c:\bayes\output\b_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.

8.4.2.5 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 8.3 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\b_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\\b_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\\b_X_sample.raw","c:\\bayes\\output\\b_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.

8.4.2.6 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.

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\b_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\\b_f_X1_X2_pspline.res")
```

prints the posterior means 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 posterior means 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.

8.5 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.

8.6 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 2.5.2, 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 2.5.3. 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.

8.6.1 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.

8.6.1.1 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.

8.6.1.2 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`.

8.6.1.3 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 8.10.

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 8.11. 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

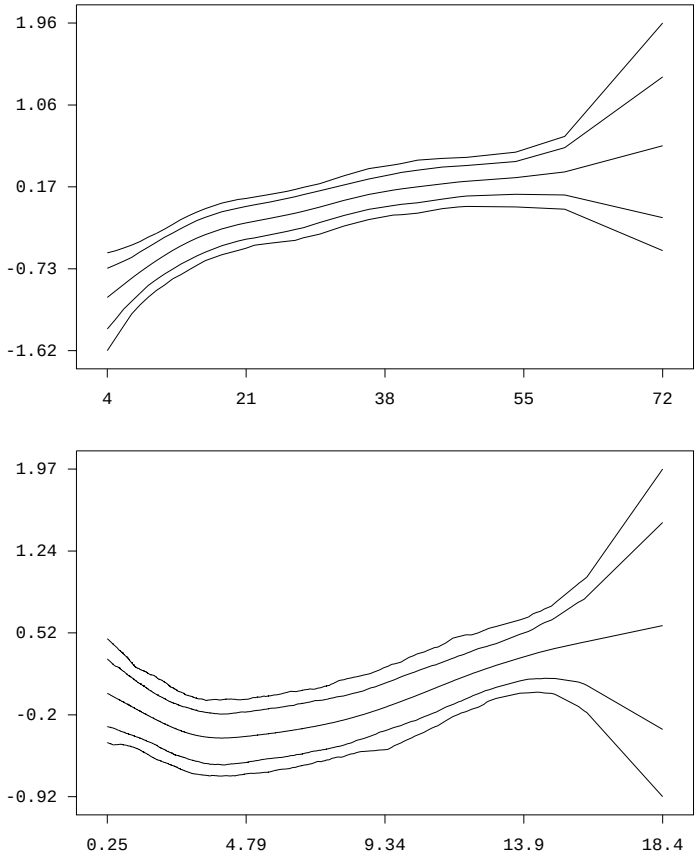


Figure 8.10: Estimated effects of duration and amount of credit. Shown is the posterior mean within 80% and 95% credible regions.

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 8.12. We see that autocorrelations die off very quickly.

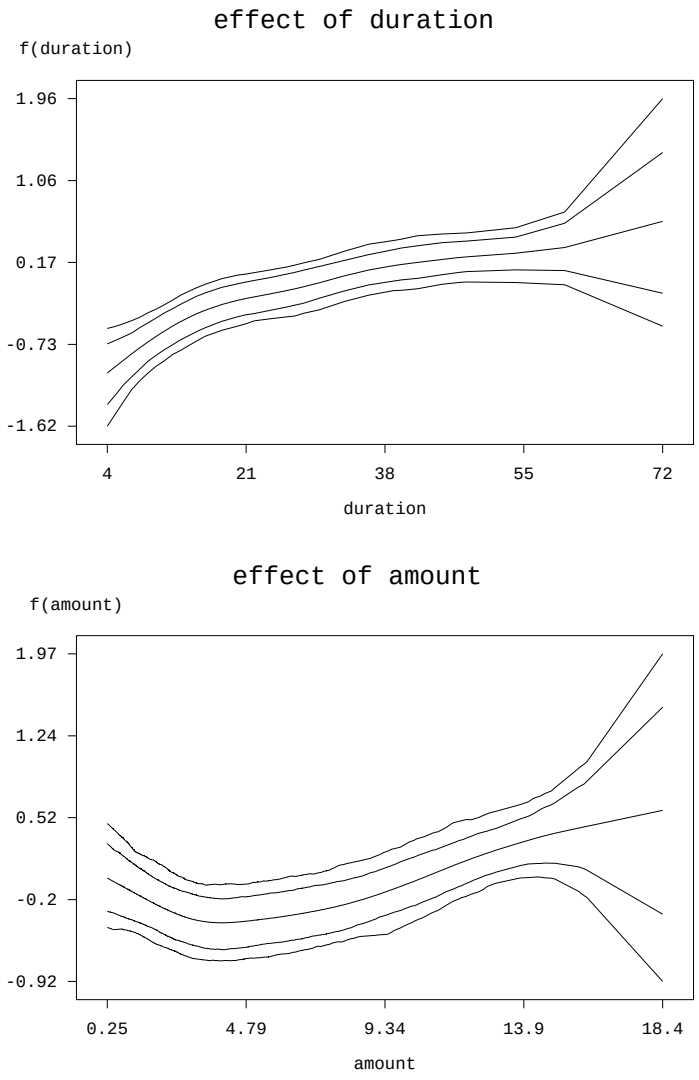


Figure 8.11: Improved plots of the effect of `duration` *and* `amount`.

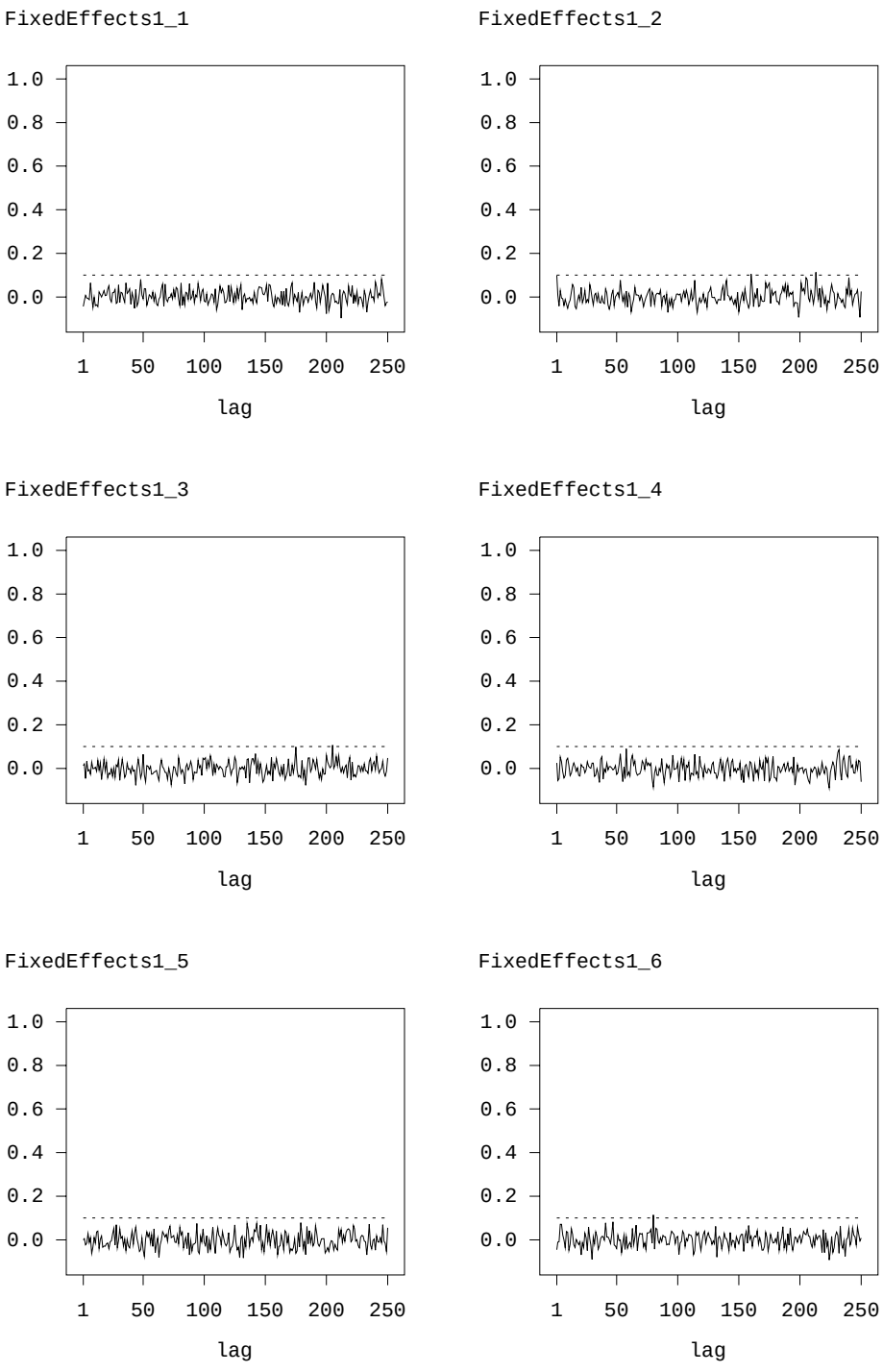


Figure 8.12: First page of the autocorrelation file.

8.6.1.4 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 7.3.1.3 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 8.13. 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 8.14. As can be seen, the autocorrelations for the logit model with IWLS proposals are almost as low as for the probit model.

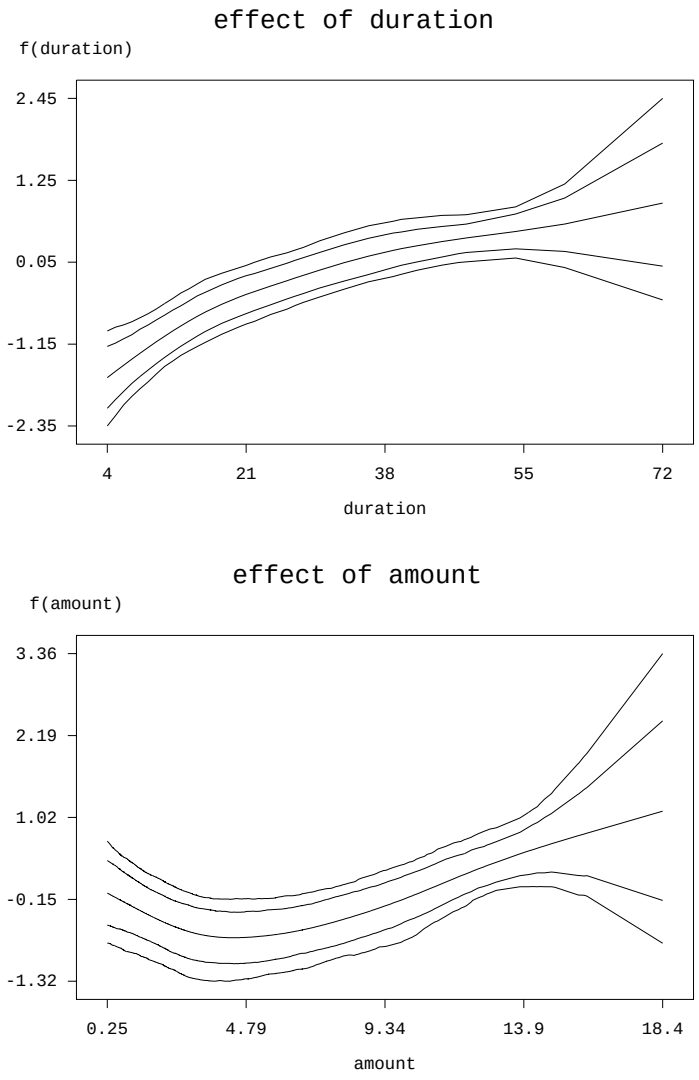


Figure 8.13: Effect of `duration` *and* `amount`*, if a logit model is estimated rather than a probit model.*

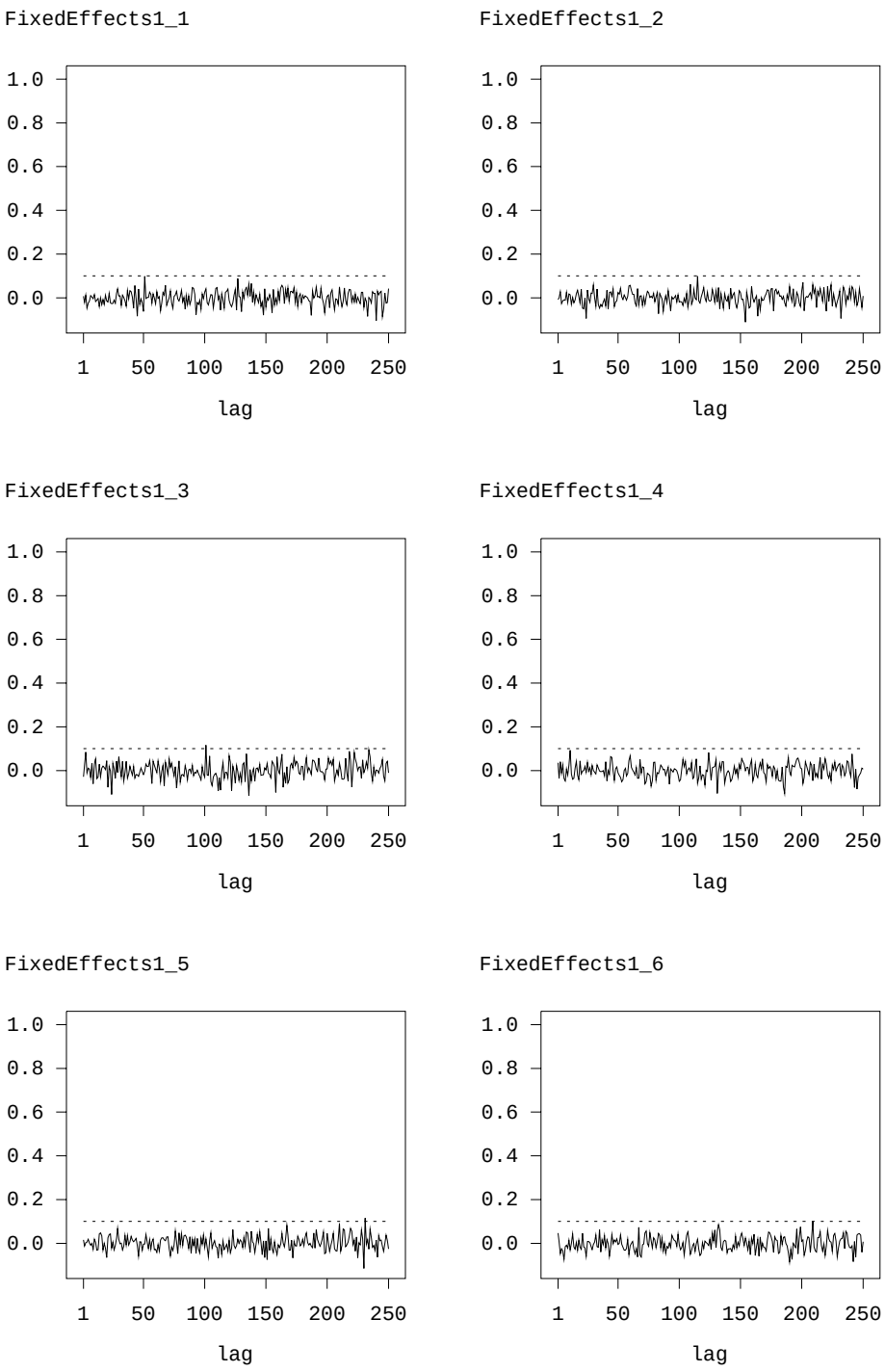


Figure 8.14: First page of the autocorrelation file, if a logit model is estimated.

8.6.1.5 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 8.6.1.3 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 8.15 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`.

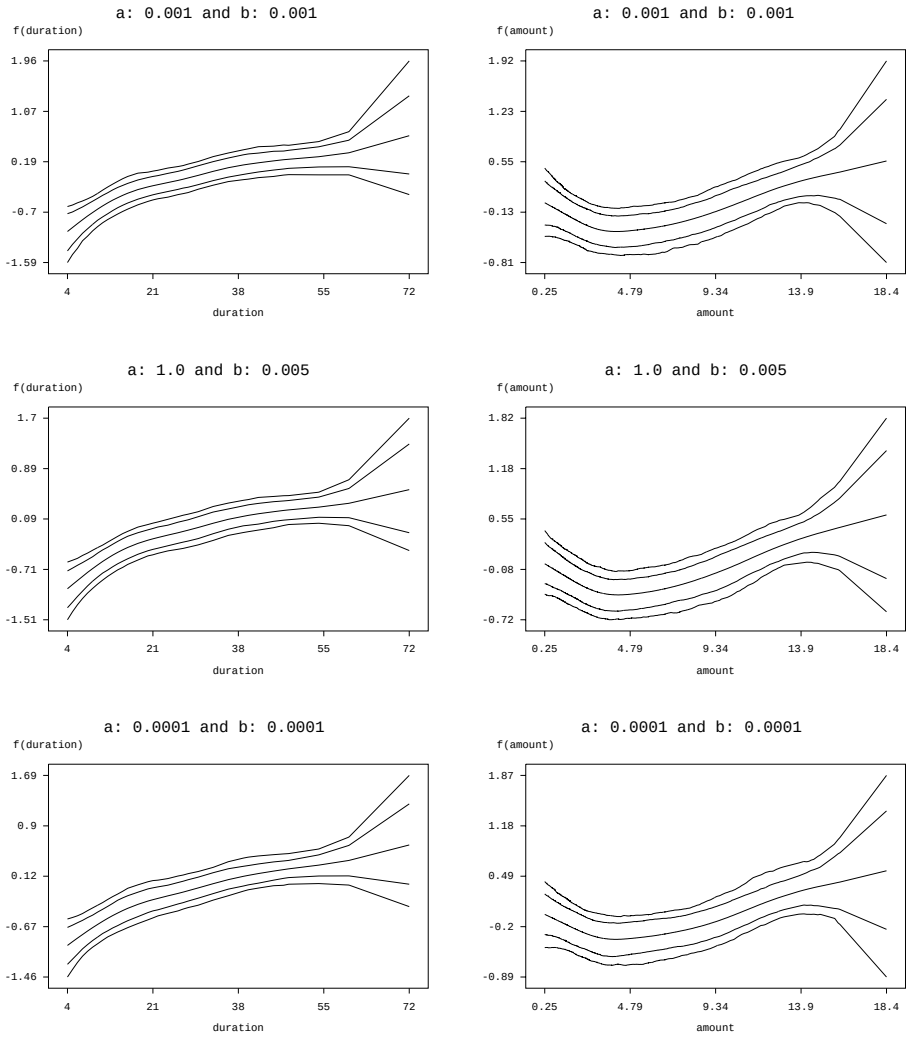


Figure 8.15: Results for the effect of `duration` *and* `amount` *for different values of the hyperparameters for the variances.*

8.6.2 Determinants of childhood undernutrition in Zambia

In this subsection we present a tutorial like example for the usage of *bayesreg objects*. We use data on undernutrition of children in Zambia, compare subsection 2.5.3 for a description of the data set. The files containing the data and the map can be found in the subdirectory `examples` of the installation directory together with the batch file `mcmctutorial.prg` containing all commands used in the following subsections.

The rest of the example is separated in seven parts dealing with the different steps of estimating a regression model. In subsubsection 8.6.2.1 we create a *dataset object* to incorporate, handle and manipulate the data. We will also give a brief description of some methods that may be applied to *dataset objects*. Since we want to estimate a spatial effect of the district in which a child lives, we need the boundaries of the districts to compute the neighborhood information of the map of Zambia. This information will be stored in a *map object*. subsubsection 8.6.2.2 describes how to create and handle these objects. Estimation of the regression model is carried out in subsubsection 8.6.2.3 using a *bayesreg object*. The next two sections describe how to visualize the estimation results and how to customize the obtained graphics. subsubsection 8.6.2.6 describes post estimation commands which can be used to investigate the sampling paths and the autocorrelation functions of the estimated parameters. In a last section we perform a sensitivity analysis to assess the impact of hyperparameter choices on our estimation results.

Please note, that all paths within the following subsections must be changed according to the storage location of the corresponding files on your hard disk.

8.6.2.1 Reading data set information

In a first step we read the available data set information into *BayesX*. Therefore we create a *dataset object* named `d`:

```
> dataset d
```

We store the data in `d` using the method `infile`:

```
> d.infile, maxobs=5000 using c:\data\zambia.raw
```

Note, that we assume the data to be provided in the external file `c:\data\zambia.raw`. The first few lines of this file look like this:

```
hazstd bmi agc district rcw edu1 edu2 tpr sex
0.0791769   21.83   4   81   -1   1   0   1   -1
-0.2541965   21.83   26   81   -1   1   0   1   -1
-0.1599823   20.43   56   81   1   -1   -1   1   1
0.1733911   22.27   6   81   -1   0   1   1   1
```

In our example the file contains the variable names in the first line. Therefore it is not necessary to specify them in the `infile` command. If the file contained only the data without variable names, we would have to supply them after the keyword `infile`:

```
> d.infile hazstd bmi agc district rcw edu1 edu2 tpr sex, maxobs=5000
  using c:\data\zambia.raw
```

Option `maxobs` can be used to speed up the execution time of the `infile` command. If `maxobs` is specified, *BayesX* allocates enough memory to store all the data while the total amount of required memory is unknown in advance if `maxobs` remains unspecified. For larger data sets this may cause *BayesX* to start reading the data set information several times because the currently allocated memory is exceeded. However, this is only meaningful for larger data sets with more than 10000 observations and could therefore be omitted in our example.

A second option that may be added to the `infile` command is the `missing` option to indicate missing values. Specifying for example `missing=M` defines the letter 'M' as an indicator for a missing value. The default for missing values are a period '.' and 'NA' (which remain valid indicators for missing values even if an additional indicator is defined by the `missing` option).

After having read in the data set information we can inspect the data visually. Executing the command

```
> d.describe
```

opens an *object-viewer window* containing the data in form of a spreadsheet. This can also be achieved by double-clicking on the *dataset object* in the *object browser*.

Further methods allow to examine the variables in the *dataset object*. For a categorial variable, e.g. `sex`, the `tabulate` command may be used to produce a frequency table:

```
> d.tabulate sex
```

resulting in

```
Variable: sex

          Value        Obs          Freq           Cum
             -1       2451        0.5057        0.5057
              1       2396        0.4943             1
```

being printed in the *output window*. For continuous variables the `descriptive` command prints several characteristics of the variable in the *output window*. E.g., executing

```
> d.descriptive bmi
```

leads to

```
   Variable     Obs        Mean      Median         Std         Min         Max
        bmi    4847   21.944349        21.4   3.2879659        12.8       39.29
```

8.6.2.2 Map objects

In the following we want to estimate a spatially correlated effect of the district in which a child lives. Therefore we need the boundaries of the districts in Zambia to compute the neighborhood information of the map of Zambia. We therefore create a *map object*

```
> map m
```

and read in the boundaries using the `infile` command of *map objects*:

```
> m.infile using c:\data\zambia.bnd
```

Having read in the boundary information, *BayesX* automatically computes the neighborhood matrix of the map.

The file following the keyword `using` is assumed to contain the boundaries in form of closed polygons. Compare chapter 5 for a description of the expected file format.

Map objects may be visualized using method `describe`:

```
> m.describe
```

resulting in the graph shown in Figure 8.16. Additionally, `describe` prints further information about the *map object* in the *output window* including the name of the object, the number of regions, the minimum and maximum number of neighbors and the bandwidth of the corresponding adjacency or neighborhood matrix:

```
MAP m
Number of regions: 54
Minimum number of neighbors: 1
Maximum number of neighbors: 9
Bandsize of corresponding adjacency matrix: 24
```

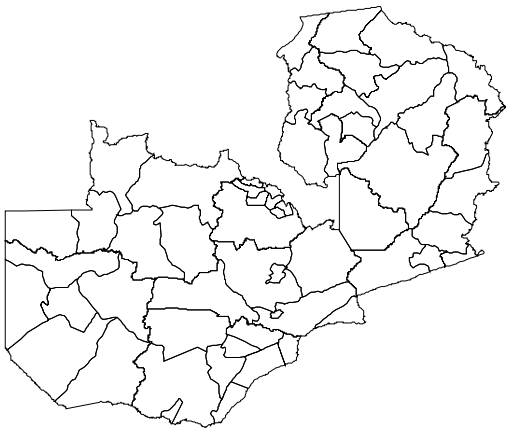


Figure 8.16: The districts within Zambia.

The numerical complexity associated with the estimation of structured spatial effects using MCMC techniques depends essentially on the structure of the neighborhood matrix. Often the geographical information stored in a boundary file does not represent the 'ideal' ordering (as regards to the estimation problem) of the districts or regions. Therefore it may be useful to reorder the map using method `reorder`:

```
> m.reorder
```

Usually reordering results in a smaller bandwidth although the bandwidth is not the criterion that is minimized by `reorder`. Instead the *envelope* of the neighborhood matrix is minimized (compare George and Liu 1981).

In order to avoid reordering the *map object* every time you start *BayesX* it is useful to store the reordered version in a separate file. This can be achieved using the `outfile` command of *map objects*:

```
> m.outfile, replace using c:\data\zambiasort.bnd
```

The reordered map is now stored in the given file. Note, that specifying the option `replace` allows *BayesX* to overwrite an existing file with the same name. Without this option an error message would be raised if the given file is already existing.

Reading the boundary information from an external file and computing the neighborhood matrix may be a computationally intensive task if the map contains a large number of regions or if the polygons are given in great detail. To avoid doing these computation in every *BayesX* session, we store the neighborhood information in a so-called *graph file* using method `outfile` together with the `graph` option (compare chapter 5 for a description of *graph files*):

```
> m.outfile, replace graph using c:\data\zambiasort.gra
```

To see how storing maps in *graph files* affects the computation time of the `infile` command, we create a second *map object* and read in the information from the graph file. Again, we have to specify the keyword `graph`:

```
> map m1
```

```
> m1.infile, graph using c:\data\zambiasort.gra
```

As you should have noticed, reading geographical information from a *graph file* is usually much faster than reading from a *boundary file*. However, using *graph files* also has a drawback. Since they do no longer contain the full information on the polygons forming the map, we can not visualize a *map object* created from a *graph file*. Trying to do so

```
> m1.describe
```

raises an error message. This implies, that visualizing estimation results of spatial effects can only be based on *map objects* created from *boundary files*, although estimation can be carried out using *graph files*. Since we will work with the *map object* `m` in the following, we delete `m1`:

```
> drop m1
```

8.6.2.3 Bayesian semiparametric regression

To estimate a regression model using MCMC techniques we first create a *bayesreg object*:

```
> bayesreg b
```

By default estimation results are written to the subdirectory `output` of the installation directory. In this case the default filenames are composed of the name of the *bayesreg object* and the type of the specific file. Usually it is more convenient to store the results in a user-specified directory. To define this directory we use the `outfile` command of *bayesreg objects*:

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

Note, that `outfile` does not only specify a directory but also a base filename (the character `b` in our example). Therefore executing the command above leads to storage of the results in the directory `c:\data` and all filenames start with the character `b`. Of course the base filename may be different from the name of the *bayesreg object*.

In addition to parameter estimates *BayesX* also gives acceptance rates for the different effects and some further information on the estimation process. In contrast to parameter estimates this information is not stored automatically but is printed in the *output window*. Therefore it is useful to store the contents of the *output window*. This can be achieved automatically by opening a log file using the `logopen` command

```
> logopen, replace using c:\data\logmcmc.txt
```

After opening a log file, every information written to the *output window* is also stored in the log file. Option `replace` allows *BayesX* to overwrite an existing file with the same name as the specified log file. Without `replace` results are appended to an existing file.

The model presented in Kandala et al. (2001) is given by the following semiparametric predictor:

$$\eta = \gamma_0 + \gamma_1 rcw + \gamma_2 edu1 + \gamma_3 edu2 + \gamma_4 tpr + \gamma_5 sex + f_1(bmi) + f_2(agc) + f^{str}(district) + f^{unstr}(district)$$

The two continuous covariates `bmi` and `agc` are assumed to have a possibly nonlinear effect on the Z-score and are therefore modelled nonparametrically (as P-splines with second order random walk prior in our example). The spatial effect of the district is split up into a spatially correlated part $f^{str}(district)$ and an uncorrelated part $f^{unstr}(district)$, see Fahrmeir and Lang (2001b) for a motivation. The correlated part is modelled by a Markov random field prior, where the neighborhood matrix and possible weights associated with the neighbors are obtained from the *map object* `m`. The uncorrelated part is modelled by an i.i.d. Gaussian effect.

To estimate the model we use method `regress` of *bayesreg objects*:

```
> b.regress hazstd = rcw + edu1 + edu2 + tpr + sex + bmi(psplinerw2)
  + agc(psplinerw2) + district(spatial,map=m) + district(random),
  family=gaussian iterations=12000 burnin=2000 step=10 predict using d
```

Options `iterations`, `burnin` and `step` define properties of the MCMC-algorithm. The total number of MCMC iterations is given by `iterations` while the number of burn in iterations is given by `burnin`. Therefore we obtain a sample of 10000 random numbers with the above specifications. Since, in general, these random numbers are correlated, we do not use all of them but thin out the Markov chain by the thinning parameter `step`. Specifying `step=10` as above forces *BayesX* to store only every 10th sampled parameter which leads to a random sample of length 1000 for every parameter in our example.

Note, that the choice of `iterations` also affects computation time. On a 2.4 GHz PC estimation of our model took about 1 minute and 5 seconds, which is rather fast in regard of the complexity of the model.

If option `predict` is specified, samples of the deviance, the effective number of parameters p_D, and the deviance information criteria DIC of the model are computed, see Spiegelhalter et al. (2002). In addition, estimates for the linear predictor and the expectation of every observation are obtained.

In the following we reproduce the content of the *output window* to make the user familiar with the estimation results produced by *BayesX*:

```
ESTIMATION RESULTS:

  Predicted values:

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

  Estimation results for the deviance:

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

  Mean:             12688.959
  Std. Dev:         12.615837
  2.5% Quantile:    12663.847
  10% Quantile:     12673.03
  50% Quantile:     12688.804
  90% Quantile:     12705.921
  97.5% Quantile:   12714.078

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

  Mean:             4848.1335
  Std. Dev:         98.563486
  2.5% Quantile:    4657.7394
  10% Quantile:     4719.1869
  50% Quantile:     4847.534
  90% Quantile:     4971.7679
  97.5% Quantile:   5059.5874

  Samples of the deviance are stored in file
  c:\data\b_deviance_sample.raw

  Estimation results for the DIC:

  DIC based on the unstandardized deviance

  Deviance(bar_mu):          12639.654
  pD:                        49.305405
  DIC:                       12738.265
```

```
DIC based on the saturated deviance

Deviance(bar_mu):           4797.8139
pD:                         50.31962
DIC:                        4898.4532

Estimation results for the scale parameter:

Acceptance rate:   100 %

Mean:             0.802517
Std. dev.:        0.0164098
2.5% Quantile:    0.768981
10% Quantile:     0.782025
50% Quantile:     0.802168
90% Quantile:     0.824066
97.5% Quantile:   0.83595


FixedEffects1

Acceptance rate:    100 %

Variable  mean           Std. Dev.      2.5% quant.    median         97.5% quant.
const     0.102975       0.0493194      0.00460694     0.102048       0.201918
rcw       0.00782474     0.0129786      -0.0177587     0.0079339      0.0325389
edu1      -0.0612525     0.0268997      -0.11368       -0.0622293     -0.00870588
edu2      0.234627       0.0468064      0.146532       0.23578        0.322222
tpr       0.0891162      0.0218746      0.0476786      0.0893937      0.133562
sex       -0.058801      0.0130027      -0.083714      -0.0593365     -0.031744

Results for fixed effects are also stored in file
c:\data\b_FixedEffects1.res


f_bmi_pspline

Acceptance rate:    100 %

Results are stored in file
c:\data\b_f_bmi_pspline.res

Postscript file is stored in file
c:\data\b_f_bmi_pspline.ps

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


f_bmi_pspline_variance

Acceptance rate:    100 %

Estimation results for the variance component:

Mean:             0.00192786
Std. dev.:        0.00268103
2.5% Quantile:    0.000281651
10% Quantile:     0.000452872
50% Quantile:     0.00119819
90% Quantile:     0.00380296
97.5% Quantile:   0.00806144
```

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


f_agc_pspline

Acceptance rate:    100 %

Results are stored in file
c:\data\b_f_agc_pspline.res

Postscript file is stored in file
c:\data\b_f_agc_pspline.ps

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


f_agc_pspline_variance

Acceptance rate:    100 %

Estimation results for the variance component:

Mean:             0.00600587
Std. dev.:        0.00993897
2.5% Quantile:    0.00119369
10% Quantile:     0.00169024
50% Quantile:     0.00397818
90% Quantile:     0.0107538
97.5% Quantile:   0.0227737

Results for the variance component are also stored in file
c:\data\b_f_agc_pspline_var.res


f_district_spatial

Acceptance rate:    100 %

Results are stored in file
c:\data\b_f_district_spatial.res

Postscript file is stored in file
c:\data\b_f_district_spatial.ps

Results may be visualized in BayesX using method 'drawmap'
Type for example: objectname.drawmap 5


f_district_spatial_variance

Acceptance rate:    100 %

Estimation results for the variance component:

Mean:             0.0359038
Std. dev.:        0.0176849
2.5% Quantile:    0.0117425
10% Quantile:     0.0168868
50% Quantile:     0.0321435
```

```
90% Quantile:     0.0593765
97.5% Quantile:   0.0807406

Results for the variance component are also stored in file
c:\data\b_f_district_spatial_var.res


f_district_random

Acceptance rate:    100 %

Results for random effects are stored in file
c:\data\b_f_district_random.res

Results for the sum of the structured and unstructured
spatial effects are stored in file
c:\data\b_district_spatialtotal.res


f_district_random_variance

Acceptance rate:    100 %

Estimation results for the variance component:

Mean:             0.0077143
Std. dev.:        0.00580379
2.5% Quantile:    0.000703806
10% Quantile:     0.00152536
50% Quantile:     0.00648848
90% Quantile:     0.0153428
97.5% Quantile:   0.0215434

Results for the variance component are also stored in file
c:\data\b_f_district_random_var.res

Files of model summary:

------------------------------------------------------------------------

Batch file for visualizing effects of nonlinear functions is stored in file
c:\data\b_graphics.prg

NOTE: 'input filename' must be substituted by the filename of the boundary-file

------------------------------------------------------------------------

Batch file for visualizing effects of nonlinear functions
in S-Plus is stored in file
c:\data\b_splus.txt

NOTE: 'input filename' must be substituted by the filename of the boundary-file

------------------------------------------------------------------------

Latex file of model summaries is stored in file
c:\data\b_model_summary.tex

------------------------------------------------------------------------
```

In addition to the information being printed to the *output window* results for each effect are written

to external ASCII files. The names of these files are given in the *output window*, compare the previous pages. The files contain the posterior mean and median, the posterior 2.5%, 10%, 90% and 97.5% quantiles, and the corresponding 95% and 80% posterior probabilities of the estimated effects. For example, the beginning of the file `c:\data\b_f_bmi_pspline.res` for the effect of `bmi` looks like this:

intnr	bmi	pmean	pqu2p5	pqu10	pmed	pqu90	pqu97p5	pcat95	pcat80
1	12.8	-0.284065	-0.660801	-0.51678	-0.283909	-0.0585753	0.085998	0	-1
2	13.15	-0.276772	-0.609989	-0.483848	-0.275156	-0.070517	0.0572406	0	-1
3	14.01	-0.258674	-0.515628	-0.416837	-0.257793	-0.10009	-0.00289024	-1	-1

The posterior quantiles and posterior probabilities may be changed by the user using the options `level1` and `level2`. For example specifying `level1=99` and `level2=70` in the options list of the `regress` command leads to the computation of 0.5%, 15%, 85% and 99.5% quantiles of the posterior. The defaults are `level1=95` and `level2=80`.

Some nonparametric effects are visualized by *BayesX* automatically and the resulting graphs are stored in ps format. E.g. the effect of `bmi` is visualized in the file `c:\data\b_f_bmi_pspline.ps` (compare the results on the previous pages for the other filenames). In addition to the postscript files a file containing the commands to reproduce the graphics is stored in the output directory. In our example the name of the file is `c:\data\b_graphics.prg`. The advantage is that additional options may be added by the user to customize the graphs (compare the following two subsubsections).

Moreover a file with ending `.tex` is created in the outfile directory. This file contains a summary of the estimation results and may be compiled using LaTeX.

Having finished the estimation we may close the log file by typing

```
> logclose
```

Note, that the log file is closed automatically when you exit *BayesX*.

8.6.2.4 Visualizing estimation results

BayesX provides three possibilities to visualize estimation results:

- As mentioned in the previous subsubsection, certain results are automatically visualized by *BayesX* and stored in postscript files.
- Post estimation commands of *bayesreg objects* allow to visualize results after having executed a `regress` command.
- *Graph objects* may be used to produce graphics using the ASCII files containing the estimation results. In principle *graph objects* allow the visualization of any content of a *dataset object*. *Graph files* are also used in the batch file containing the commands to reproduce the automatically generated graphics.

In this subsubsection we describe the general usage of the post estimation commands as well as the commands for the usage with *graph objects* to enable the user to reproduce the automatically generated plots directly in *BayesX*. subsubsection 8.6.2.5 describes how to customize plots.

Post estimation commands After having estimated a regression model plots for nonparametric effects of continuous covariates can be produced using the post estimation command `plotnonp`:

```
> b.plotnonp 1
```

and

```
> b.plotnonp 3
```

produce the graphs shown in Figure 8.17 in an *object-viewer window*. The numbers following the `plotnonp` command depend on the order in which the model terms have been specified. The numbers are supplied in the *output window* after estimation, compare the results in the previous subsubsection.

By default the plots contain the posterior mean and pointwise credible intervals according to the levels specified in the `regress` command. So by default the plot includes pointwise 80% and 95% credible intervals.

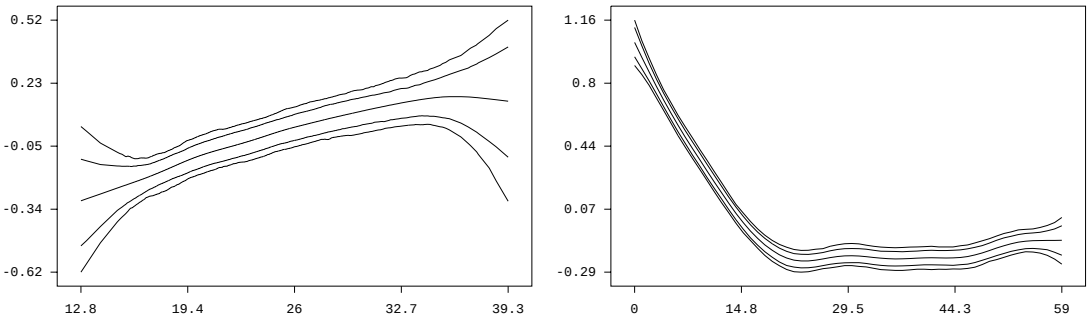


Figure 8.17: Effect of the body mass index of the child's mother and of the age of the child together with pointwise 80% and 95% credible intervals.

A plot may be stored in ps format using the `outfile` option. Executing

```
> b.plotnonp 1, replace outfile = c:\data\f_bmi.ps
```

stores the plot for the estimated effect of `bmi` in the file `c:\data\f_bmi.ps`. Again, specifying `replace` allows *BayesX* to overwrite an existing file. Note, that *BayesX* does not display the graph on the screen if the option `outfile` is specified.

Estimation results for spatial effects are best visualized by drawing the respective map and coloring the regions of the map according to some characteristic of the posterior, e.g. the posterior mean. For the structured spatial effect this can be achieved using the post estimation command `drawmap`

```
> b.drawmap 5
```

which results in the graph shown in Figure 8.18.

Graph Objects The commands presented in the previous paragraph work only after having estimated a regression model in the current *BayesX* session but it may also be useful to visualize results of former analyses. This can be achieved using *graph objects*. Note again, that *graph files* are also used in the batch file containing the commands to reproduce the automatically generated graphics. Therefore the purpose of this paragraph is also to enable the user to understand the content of this batch file.

First we read the estimation results into a *dataset object*. For example the estimation results for the effect of `bmi` can be read into *BayesX* by executing the commands

```
> dataset res
> res.infile using c:\data\b_f_bmi_pspline.res
```

Now the estimation results (or any content of a *dataset object*) may be visualized using a *graph object* which we create by typing

```
> graph g
```

The results stored in the *dataset object* `res` are now visualized using the `plot` command of *graph*

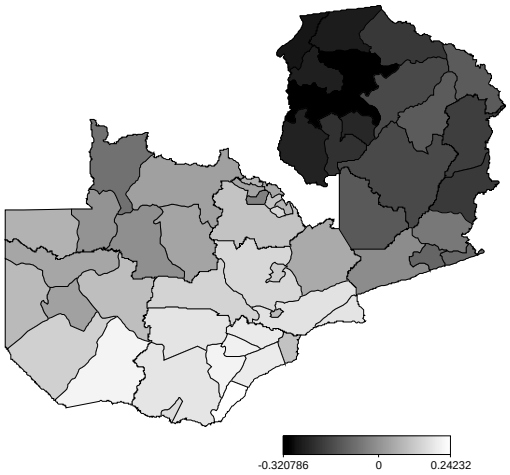


Figure 8.18: Posterior mean of the structured spatial effect.

objects. Executing

```
> g.plot bmi pmean pqu2p5 pqu10 pqu90 pqu97p5 using res
```

reproduces the graph in Figure 8.17.

Similar as for `plotnonp`, the direct usage of the `drawmap` command is only possible after executing a `regress` command. However, using *graph objects* again allows us to visualize results that have been stored in a file.

First we read the information contained in this file into a *dataset object*. For example the following command

```
> res.infile using c:\data\b_f_district_spatial.res
```

stores the estimation results for the structured spatial effect in the *dataset object* `res`. Now we can visualize the posterior mean using method `drawmap` of *graph objects* leading again to the graph shown in Figure 8.18:

```
> g.drawmap pmean district, map=m using res
```

Since – in contrast to a *bayesreg object* – no *map object* is associated with a *graph object* we have to specify the map that we want to use explicitly in the options list.

Using *graph objects* also allows us to plot other characteristics of the posterior than the posterior mean. For instance the posterior 95% probabilities may be visualized by

```
> g.drawmap pcat95 district, map=m using res
```

The result is shown in Figure 8.19.

A further advantage of *graph objects* is, that they allow to visualize the estimation results for the uncorrelated spatial effects. Since these are modelled as unstructured random effects, *BayesX* is unable to recognize them as spatial effects. However, proceeding as follows gives us the possibility to plot the unstructured spatial effect shown in Figure 8.20:

```
> res.infile using c:\data\b_f_district_random.res
> g.drawmap pmean district, map=m using res
```

8.6.2.5 Customizing graphics

This subsubsection describes how to customize graphics created in *BayesX*. All options are described for the usage with the post estimation commands but may be used with graph files as well. So the

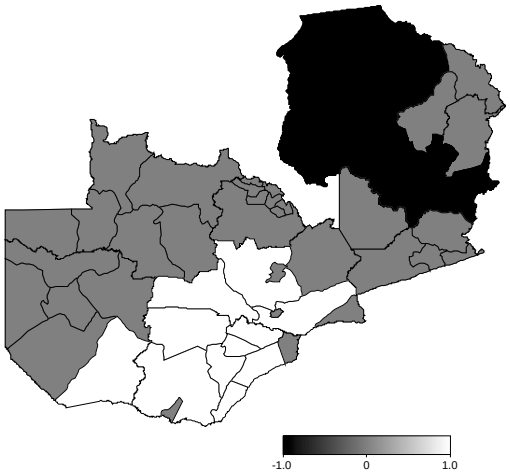


Figure 8.19: Posterior 95% probability of the structured spatial effect.

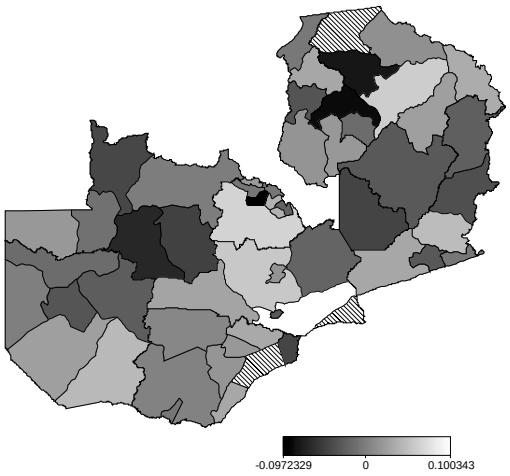


Figure 8.20: Posterior mean of the unstructured spatial effect.

options presented in this subsubsection also enable the user to modify the batch file containing the commands to reproduce the automatically generated graphics.

For the presentation of nonparametric effects it may be desirable to include only one of the credible intervals into the plot. This is achieved by specifying the `levels` option. Possible values of this option are `1` and `2`, corresponding to the levels specified in the `regress` command (compare subsubsection 8.6.2.3). If the default values of `level1` and `level2` have been used, specifying `levels=2` in the `plotnonp` command causes *BayesX* to plot the 80% credible interval only (Figure 8.21):

```
> b.plotnonp 1, levels=2
```

It may be useful to add some more information to the graphs of nonparametric effects to distinguish more obviously between different covariates. Ways to do so are the specification of a title or the specification of axis labels. Both possibilities are supported by *BayesX* as demonstrated in the following examples (compare Figure 8.22 for the resulting plots):

```
> b.plotnonp 1, title="Mother body mass index"
> b.plotnonp 1, xlab="bmi" ylab="f_bmi" title="Mother body mass index"
```

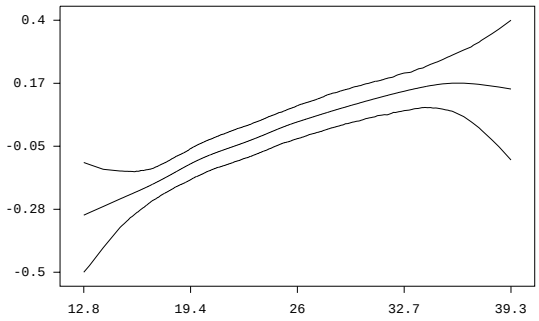


Figure 8.21: Effect of the body mass index of the child's mother with pointwise 80% credible interval only.

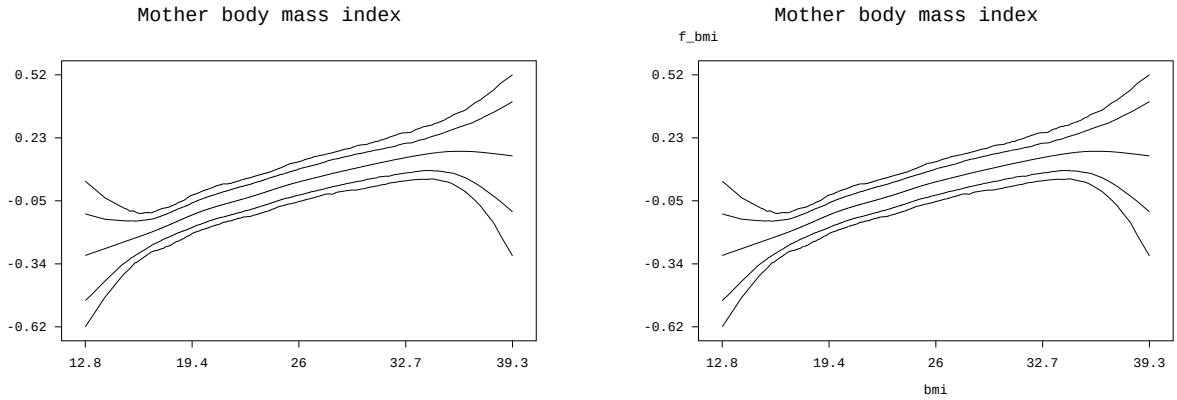


Figure 8.22: Specification of title and axis labels.

By default *BayesX* displays x- and y-axis with five equidistant ticks according to the range of the data that is to be visualized. These defaults may be overwritten using the options `xlimbottom`, `xlimtop` and `xstep` for the x-axis and `ylimbottom`, `ylimtop` and `ystep` for the y-axis, respectively. The usage of these options is more or less self-explanatory and is demonstrated in the following commands which lead to the graph shown in Figure 8.23.

```
> r.plotnonp 1, xlab="bmi" ylab="f_bmi" title="Mother body mass index"
  ylimbottom=-0.8 ylimtop=0.6 ystep=0.2 xlimbottom=12 xlimtop=40
```

Figure 8.23 also includes a graph for the effect of the age of the child that is customized in the same way as for the effect of `bmi`.

```
> r.plotnonp 3, xlab="age" ylab="f_age" title="Age of the child in months"
  ylimbottom=-0.3  ystep=0.3 xlimbottom=0 xlimtop=60 xstep=10
```

Now we turn to the options for method `drawmap`. By default `drawmap` uses grey scales to represent different values of the posterior mean. Using the option `color` forces *BayesX* to use different colors instead. Here the default would be to represent higher values through green colors and smaller values through red colors. Specifying `swapcolors` switches this definition. Therefore the following command

```
> b.drawmap 5, color swapcolors
```

leads to the graph shown in Figure 8.24 with higher values being represented through red colors and smaller values through green colors.

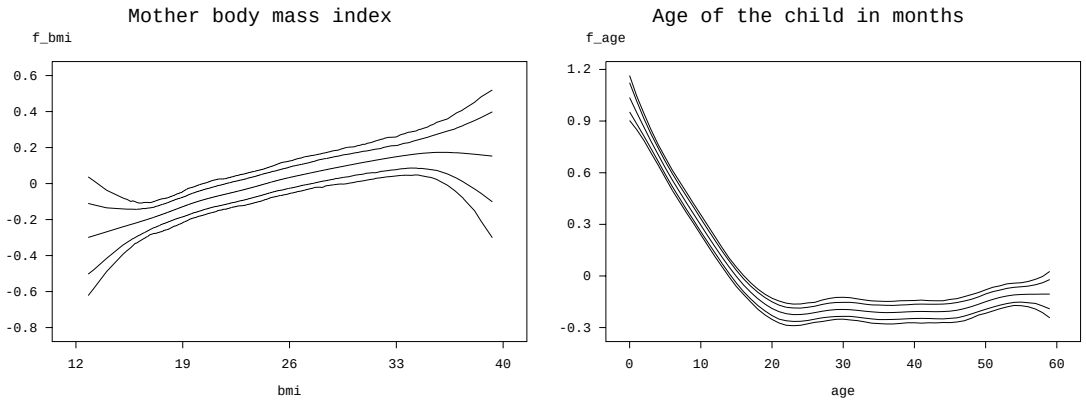


Figure 8.23: Re-defining x- and y-axis.

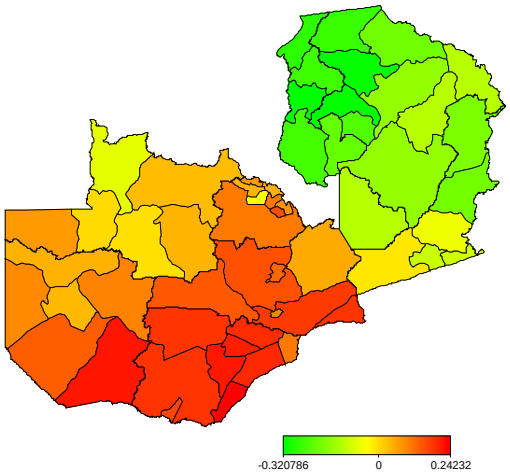


Figure 8.24: Posterior mean of the structured spatial effect in color.

Similar options as for the visualization of nonparametric effects exist for method `drawmap`. For example, a title may be included by specifying the option `title`

```
> b.drawmap 5, color swapcolors title="Structured spatial effect"
```

or the range of values to be displayed may be defined using the options `lowerlimit` and `upperlimit`:

```
> b.drawmap 5, color swapcolors title="Structured spatial effect" lowerlimit=-0.3
  upperlimit=0.3
```

The graph produced by the second command is shown in Figure 8.25.

8.6.2.6 Autocorrelation functions and sampling paths

Bayesreg objects provide some post estimation commands to get sampled parameters or to plot autocorrelation functions of sampled parameters. For example

```
> b.plotautocor, maxlag=250
```

computes and displays the autocorrelation functions for all estimated parameters with `maxlag` specifying the maximum lag number (Figure 8.26 shows a small part of the resulting graph).

If the number of parameters is large this may be computationally expensive, so *BayesX* pro-

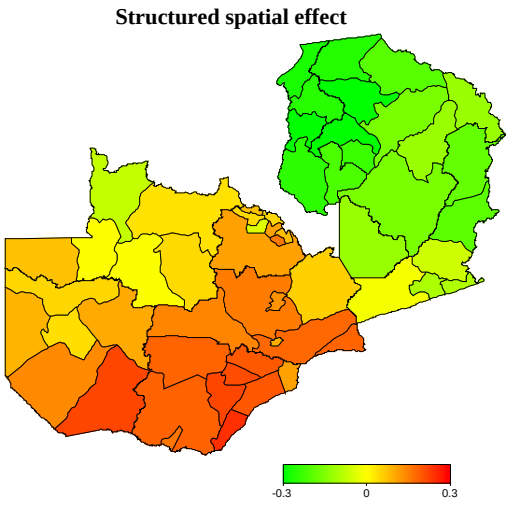


Figure 8.25: Specifying a title and the range of the plot for spatial effects.

vides a second possibility to compute autocorrelation functions. Adding the option `mean` to the `plotautocor` command as in

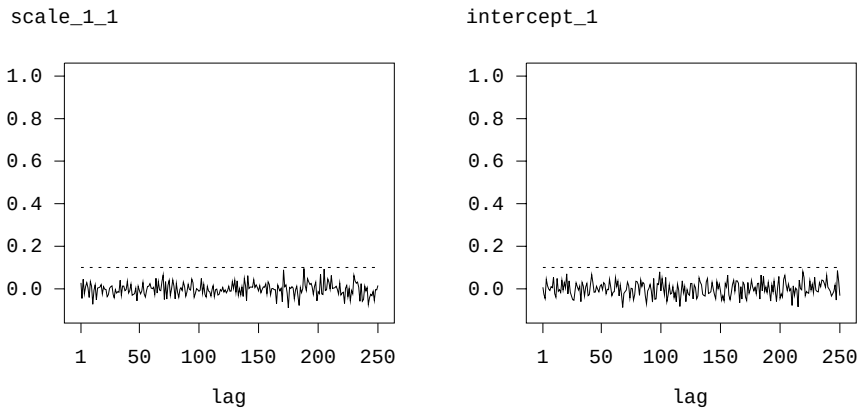


Figure 8.26: Autocorrelation function for the scale parameter and the intercept.

```
> b.plotautocor, mean
```

leads to the computation of only the minimum, mean and maximum autocorrelation functions. The result for the scale parameter is shown in Figure 8.27.

Note, that executing the `plotautocor` command also stores the computed autocorrelation functions in a file named `autocor.raw` in the output directory of the *bayesreg object*.

To save memory, the sampling paths of the estimated parameters are only stored temporarily by default and will be destroyed, when the corresponding *bayesreg object* is deleted. If we want to store the sampling paths permanently, we have to execute the `getsample` command

```
> b.getsample
```

which stores the sampled parameters in ASCII files in the output directory. To avoid too large files, the samples are typically partitioned into several files. Executing the `getsample` command also produces postscript files of the sampling paths in the output directory (compare Figure 8.28 for the content of one of these files).

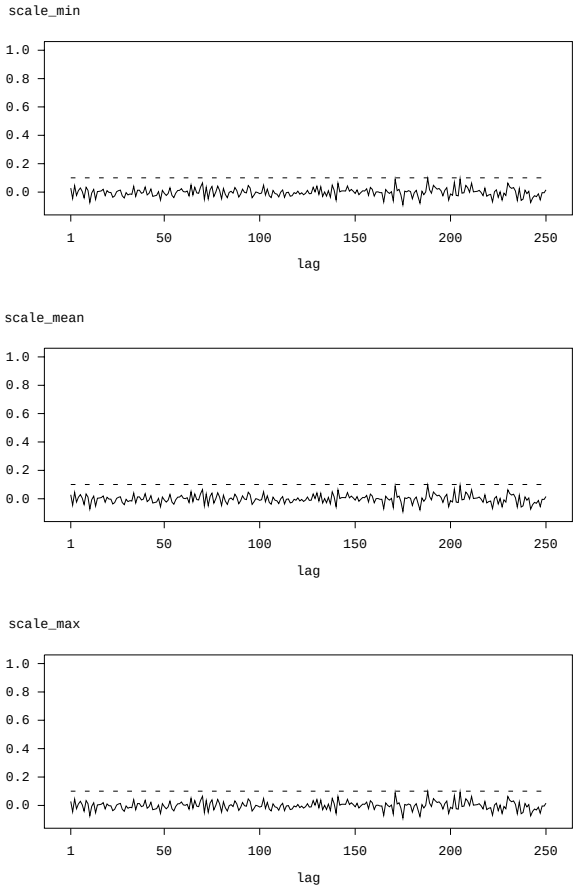


Figure 8.27: Minimum, mean and maximum autocorrelation function for the scale parameter.

8.6.2.7 Sensitivity analysis

In some situations the estimation results of a full Bayesian semiparametric regression model depend on the choice of hyperparameters, e.g. the parameters `a` and `b` defining the inverse gamma prior of the variances of nonparametric and spatial effects. It is often recommended to check how sensitive the results are with respect to changes in the hyperparameters. In the following we will re-estimate the model from subsubsection 8.6.2.3 with different choices for the hyperparameters `a` and `b` for each effect in the model. The standard choices for `a` and `b` are `a=b=0.001`. As a first trial we choose a smaller value for `a` and `b`:

```
> b.regress hazstd = rcw + edu1 + edu2 + tpr + sex
  + bmi(psplinerw2,a=0.00001,b=0.00001) + agc(psplinerw2,a=0.00001,b=0.00001)
  + district(spatial,map=m,a=0.00001,b=0.00001)
  + district(random,a=0.00001,b=0.00001), family=gaussian iterations=12000
  burnin=2000 step=10 predict using d
```

Figure 8.29 shows the results for the nonparametric effects with this choice of hyperparameters. Obviously, the estimated functions are somewhat smoother but they do not differ that much from the estimates with the standard choices.

Now we try two further choices for the hyperparameters, with both `a=1` and `b` small. We estimate models with `b=0.005` and `b=0.00005`:

```
> b.regress hazstd = rcw + edu1 + edu2 + tpr + sex + bmi(psplinerw2,a=1,b=0.005)
```

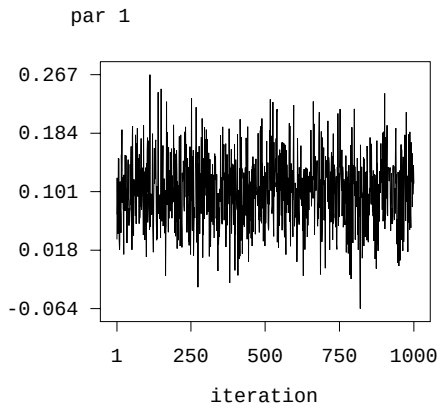


Figure 8.28: Sampling path of the intercept.

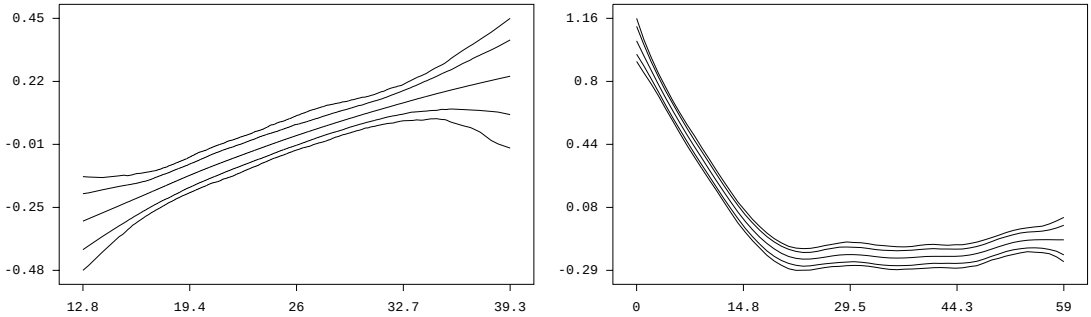


Figure 8.29: Results for the nonparametric effects with hyperparameters `a=b=0.00001` *for nonparametric and spatial effects.*

```
  + agc(psplinerw2,a=1,b=0.005) + district(spatial,map=m,a=1,b=0.005)
  + district(random,a=1,b=0.005), family=gaussian iterations=12000 burnin=2000
  step=10 predict using d
> b.regress hazstd = rcw + edu1 + edu2 + tpr + sex + bmi(psplinerw2,a=1,b=0.00005)
  + agc(psplinerw2,a=1,b=0.00005) + district(spatial,map=m,a=1,b=0.00005)
  + district(random,a=1,b=0.00005), family=gaussian iterations=12000 burnin=2000
  step=10 predict using d
```

Figure 8.30 and Figure 8.31 contain the results for the nonparametric effects for the two choices of hyperparameters.

8.7 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.

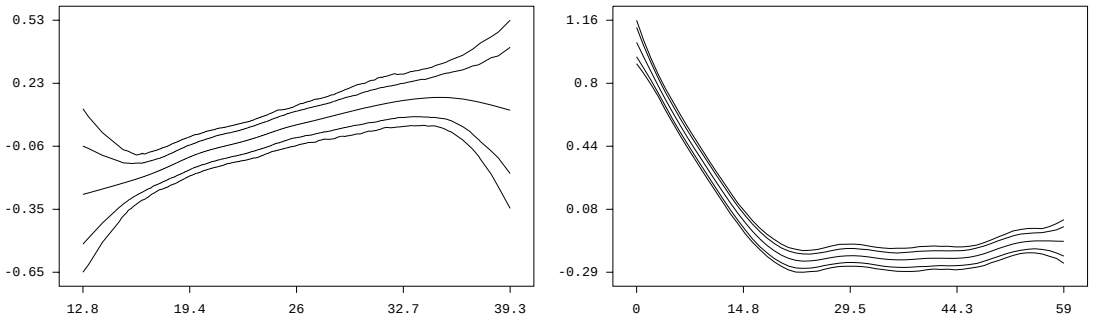


Figure 8.30: Results for the nonparametric effects with hyper parameters `a=1` *and* `b=0.005` *for nonparametric and spatial effects.*

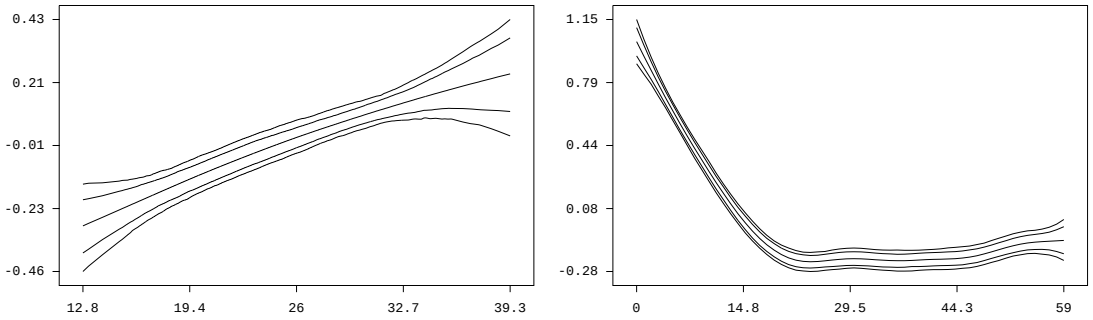


Figure 8.31: Results for the nonparametric effects with hyper parameters `a=1` *and* `b=0.00005` *for nonparametric and spatial effects.*

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.

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 Generalized 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 9

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 chapter 7. 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 tutorial like example in section 9.4.

9.1 Method regress

9.1.1 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 9.1.1.2. The distribution of the response variable can be either Gaussian, binomial, multinomial, Poisson or gamma, see also Table 9.5 for an overview about the models supported by *remlreg objects*. The response distribution is specified using option `family`, see subsubsection 9.1.1.4 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.

9.1.1.1 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.

9.1.1.2 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 9.1. Table 9.2 shows how interactions between covariates are specified. Full details about all available options are given in subsubsection 9.1.1.3.

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 9.1: 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.

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 9.2: *Possible interaction terms for remlreg objects.*

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)
```

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 using one of the Markov random field priors (7.11) or (7.12) 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` the seasonal prior (7.9) 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 the spatial smoothness prior (7.11), or the more general prior (7.12) depending on the weight definition in the *map object* `germany`.

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 subsection 7.2.4. 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.

9.1.1.3 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 9.3 provides explanations and the default values of all possible options. In Table 9.4 all reasonable combinations of model terms and options can be found.

9.1.1.4 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

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 subsection 7.2.4.	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 9.3: Optional arguments for remlreg object *terms*

and gamma. An overview over the supported models is given in Table 9.5. In *BayesX* the distribution of the response is specified by adding the additional option `family` to the options list of the 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 9.1.1 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 9.4: 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 9.5: 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 9.1.1 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 9.1.1 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

$$p(y) \propto y^{s-1} \exp(-\frac{s}{\mu} y) \tag{9.1}$$

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

$$p(y) \propto y^{a-1} \exp(-by) \tag{9.2}$$

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 (9.1). 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 9.1.1 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 subsection 7.4.2. 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 7.4.2 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 7.4.2. Let `x1` be a covariate with linear effect and `x2` a continuous one with a non-linear 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.

9.1.2 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 Equation 7.32 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`

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%.

9.1.3 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 9.1.2 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 9.1.2 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 subsection 9.2.1 and subsection 9.2.2, 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.

9.1.4 Examples

Here we give only a few examples about the usage of method `regress`. A more detailed, tutorial like example can be found in section 9.4.

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
```

9.2 Visualizing estimation results

9.2.1 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 describe how to visualize the estimated effects for a particular model term. Nonlinear effects of metrical covariates and time scales are plotted with method `plotnonp`. Spatial effects are visualized with method `drawmap`.

9.2.1.1 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 in the Java-based version only.

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 and depends on the sequence of the terms in the model. 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 or P-splines are used as priors.

Options

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

- **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 posterior mode of a 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`).

- **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 file name 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 specified additionally. This prevents you from unintentionally overwriting your files.

- **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"`).

- **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.

- **xstep = realvalue**

  If `xstep` is specified, ticks are drawn at the x-axis with stepwidth *realvalue* starting at the minimum x value (or at the value specified in option `xlimbottom`). 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.

- **ystep = realvalue**

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

Examples

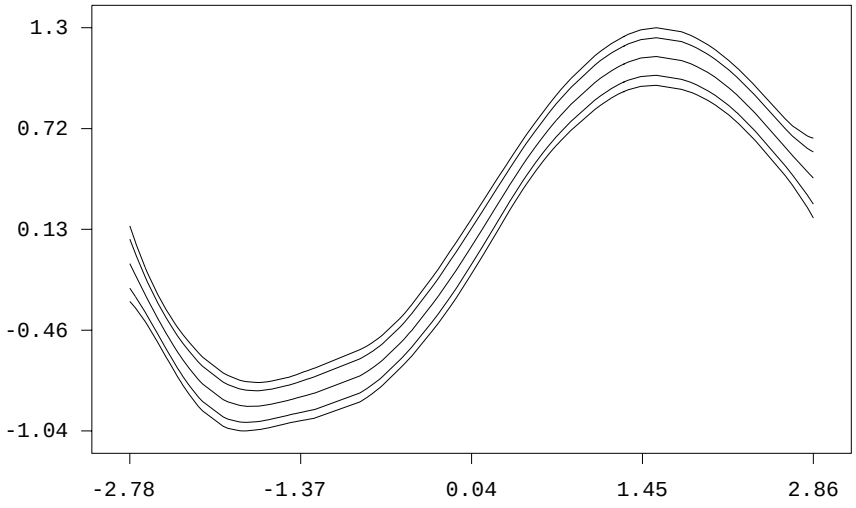


Figure 9.1: Illustration for the usage of method `plotnonp`

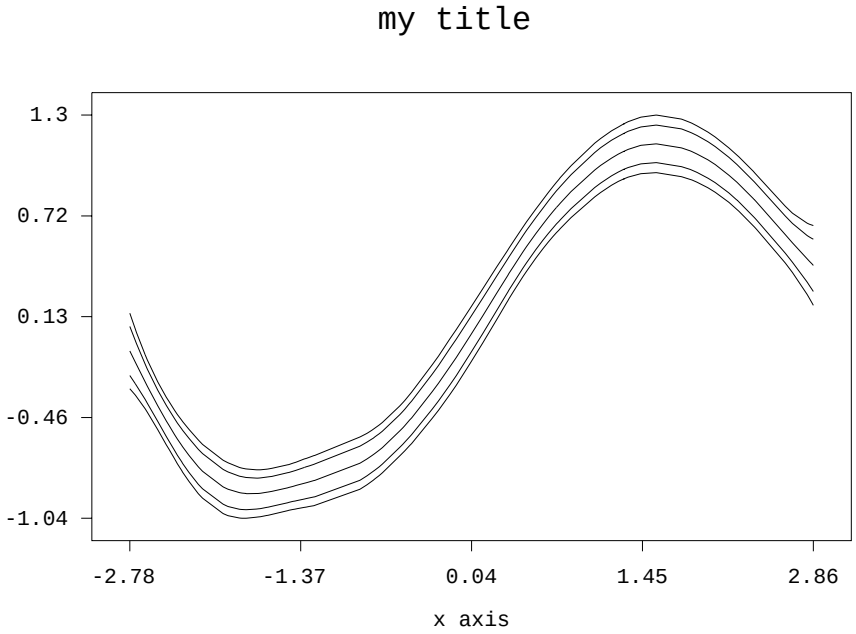


Figure 9.2: Second illustration for the usage of method `plotnonp`

Suppose we have already created a *remlreg object* `r` and have estimated a regression model with Gaussian errors using

```
> r.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_pspline

Estimated variance: 0.0258577
Estimated smoothing parameter: 3.22475

Variance and smoothing parameter are stored in file
c:\bayes\output\r_f_x_pspline_var.res

Results are stored in file
c:\bayes\output\r_f_x_pspline.res

Postscript file is stored in file
c:\bayes\output\r_f_x_pspline.ps

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

The term number of the effect of `X` is 1, i.e. by typing

```
> r.plotnonp 1
```

we obtain the plot shown in Figure 9.1.

Of course, a title, axis labels etc. can be added. For example by typing

```
> r.plotnonp 1, 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

```
> r.plotnonp 1 , 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`.

9.2.1.2 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 in the Java-based version only.

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 posterior mode (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 a *map object* is associated

with the effect that is to be visualized. In the current version this is true for Markov random fields and geosplines (compare subsubsection 9.1.1.2).

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 graph.

- **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.

- **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 file name 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 posterior mode. 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 drawn. If `upperlimit` is omitted, the maximum numerical value in `plotvar` will be used as the upper limit.

- **pcat**

  If you want to visualize the 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`.

Examples

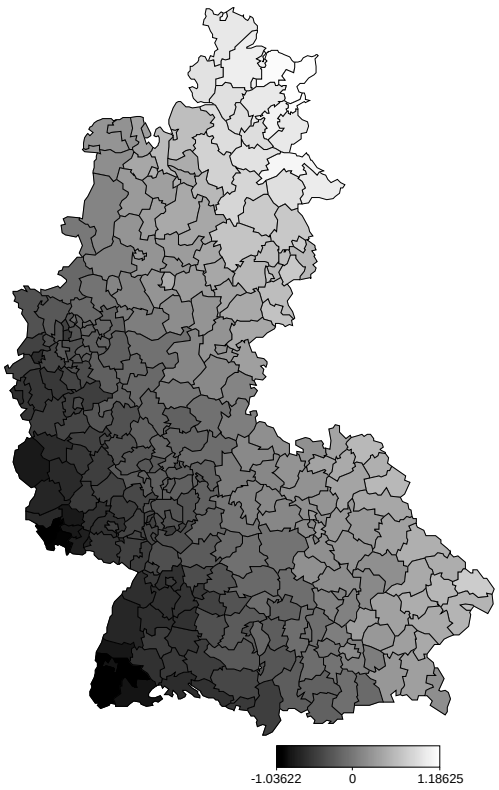


Figure 9.3: Illustration for the usage of method `drawmap`

Suppose we have already created a *remlreg object* `r` and have estimated a regression model with Gaussian errors using

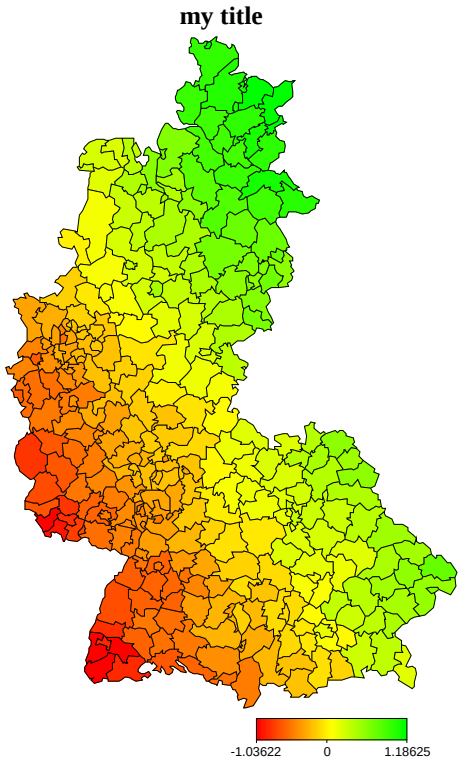


Figure 9.4: Second illustration for the usage of method `drawmap`

```
> map m
> m.infile using c:\maps\map1.bnd
> r.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_region_spatial

  Estimated variance: 0.0808068
  Estimated smoothing parameter: 1.0688

  Variance and smoothing parameter are stored in file
  c:\bayes\output\r_f_region_spatial_var.res

  Results are stored in file
  c:\bayes\output\r_f_region_spatial.res

  Postscript file is stored in file
  c:\bayes\output\r_f_region_spatial.ps

  Results may be visualized in BayesX using method 'drawmap'
  Type for example: objectname.drawmap 1
```

The term number of the effect of `region` is 1, i.e. by typing

```
> r.drawmap 1
```

we obtain the map shown in Figure 9.3 where the regions are colored according to the estimated posterior mode.

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

```
> r.drawmap 1, 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

```
> r.drawmap 1 , color title="my title" outfile="c:\results\result1.ps"
```

the colored map is stored in postscript format in the file `c:\results\result1.ps`.

9.2.2 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 9.6 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 (for *bayesreg objects*)
`plotsample`	visualizes sampling paths of sampled parameters (for *bayesreg objects*)
`readbndfile`	reads in boundaries of geographical maps
`drawmap`	visualizes estimation results for spatial covariates
`plotsurf`	visualizes estimated 2 dimensional surfaces

Table 9.6: Overview over S-plus functions

9.2.2.1 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.

9.2.2.2 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 9.3), 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 file name is

objectname`_f_X_rw.res`

that is it is composed of the name of the *remlreg object* and the covariate name. For the following we assume that `c:\bayes` is the installation directory and `r` is the name of the *remlreg object*. In this case results for the effect of `X` are stored in:

```
c:\bayes\output\r_f_X_rw.res
```

The structure of the file has already been described in section 9.1. 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 file name where nonparametric estimation results are stored. For example by entering the command

```
> plotnonp("c:\\bayes\\output\\r_f_X_rw.res")
```

an S-plus graphic-window will be opened with the plotted function estimate. The function plots the posterior mode together with 80% and 95% credible intervals. 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 on 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:

```
> plotnonp("c:\\bayes\\r_f_X_rw.res", psname="c:\\bayes\\r_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\\r_f_X_rw.ps`. A title and labels for x-axis and y-axis are added to the graph. For illustration purposes, the resulting graph is shown in Figure 9.5.

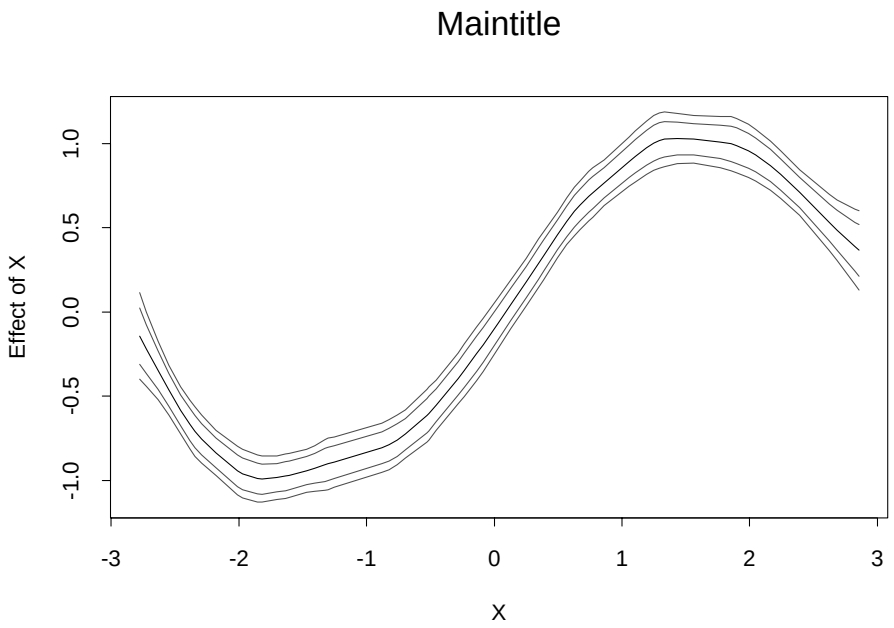


Figure 9.5: Illustration for the usage of `plotnonp`

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 9.6:

```
> plotnonp("c:\\bayes\\r_f_D_pspline.res", psname="c:\\bayes\\r_f_D_pspline.ps",
  year=1983,month=1,step=12,xlab="date", ylab=" ")
```

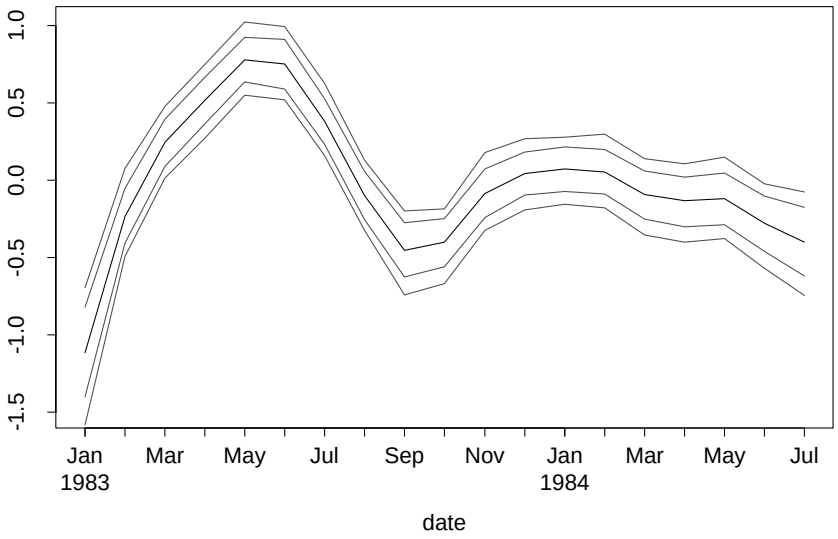


Figure 9.6: Illustration for the usage of `plotnonp`

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.

9.2.2.3 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 2.5.1, where the covariate `L` indicates the location (in subquarters) of the flat in Munich. Figure 9.7 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 9.1. For the rest of this section we assume that an effect of a spatial covariate has already been estimated and that we want to visualize the estimation results. This can easily be 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 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 in chapter 5. 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

Figure 9.7: Map of Munich

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 9.7 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 9.8 in which the subquarters of Munich are colored according to the frequency of flats in the rent 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.

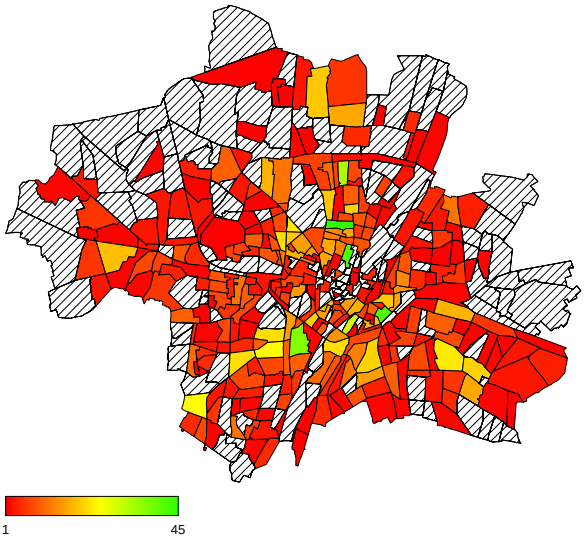


Figure 9.8: relative frequencies of observed flats in the 'rents for flats' data set

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 file name 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 `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. The only required argument that must be passed to `drawmap` 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 does not only want to draw the boundaries of a geographical map, but also to 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\r_f_L_spatial.res`. The structure of the file is described in detail in section 9.1.

Suppose now that we want to visualize estimation results for the spatial covariate location by coloring the subquarters of Munich according to the estimated posterior mode. Compared to the S-plus statement above, (at least) three more arguments must be passed to function `drawmap`; the argument `dfile` that specifies the file name of estimated results, the argument `plotvar` that specifies the variable to be plotted and the argument `regionvar` that specifies which column of the file stores the region names. The following statement produces the desired result:

```
> drawmap(map=munich,outfile="c:\\bayes\\munich.ps", plotvar="pmode",regionvar="L",
  dfile="c:\\bayes\\output\\b_f_L_spatial.res")
```

Note that the right hand side of options `plotvar` and `regionvar` must be enclosed by quotation marks.

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 that have already been mentioned:

- **map = mapname**

  Name of the S-plus *map object*. Use function `readbndfile` to read in geographical maps into S-plus.

- **dfile = "filename (including path)"**

Name (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 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)"**

  Name (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 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 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). The default value is `nrcolors=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`.

9.2.2.4 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.

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\r_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\\b_f_X1_X2_pspline.res")
```

prints the posterior mode 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 posterior mode against `X1` and `X2`.

- **mode = 1/2/3/4/5**

  This option specifies the plot type. Currently 5 different types are available. The default is `mode=1`, which corresponds to what is called a 'Surface Plot' in S-plus.

9.3 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`

9.4 Examples

9.4.1 Determinants of childhood undernutrition in Zambia

To make the user familiar with the usage of *remlreg objects* we present a tutorial like example in this subsection. We use data on undernutrition of children in Zambia, compare subsection 2.5.3 for a description of the data set. The files containing the data and the map can be found in the subdirectory `examples` of the installation directory together with the batch file `remltutorial.prg` containing all commands used in the following subsections.

The rest of the example is organized as follows: In subsubsection 9.4.1.1 we create a *dataset object* to incorporate, handle and manipulate the data. We will also give a brief description of some methods that may be applied to *dataset objects*. Since we want to estimate a spatial effect of the district in which a child lives, we need the boundaries of the districts to compute the neighborhood information of the map of Zambia. This information will be stored in a *map object*. subsubsection 9.4.1.2

describes how to create and handle these objects. Estimation of the regression model is carried out in subsubsection 9.4.1.3 using a *remlreg object*. The last two subsubsections describe how to visualize the estimation results and how to customize the obtained graphics.

Please note, that all paths within the following subsections must be changed according to the storage location of the corresponding files on your hard disk.

9.4.1.1 Reading data set information

In a first step we read the available data set information into *BayesX*. Therefore we create a *dataset object* named `d`:

```
> dataset d
```

We store the data in `d` using the method `infile`:

```
> d.infile, maxobs=5000 using c:\data\zambia.raw
```

Note, that we assume the data to be provided in the external file `c:\data\zambia.raw`. The first few lines of this file look like this:

```
hazstd bmi agc district rcw edu1 edu2 tpr sex
0.0791769   21.83   4   81   -1   1   0   1   -1
-0.2541965   21.83   26   81   -1   1   0   1   -1
-0.1599823   20.43   56   81   1   -1   -1   1   1
0.1733911   22.27   6   81   -1   0   1   1   1
```

In our example the file contains the variable names in the first line. Therefore it is not necessary to specify them in the `infile` command. If the file contained only the data without variable names, we would have to supply them after the keyword `infile`:

```
> d.infile hazstd bmi agc district rcw edu1 edu2 tpr sex, maxobs=5000
  using c:\data\zambia.raw
```

Option `maxobs` can be used to speed up the execution time of the `infile` command. If `maxobs` is specified, *BayesX* allocates enough memory to store all the data while the total amount of required memory is unknown in advance if `maxobs` remains unspecified. For larger data sets this may cause *BayesX* to start reading the data set information several times because the currently allocated memory is exceeded. However, this is only meaningful for larger data sets with more than 10,000 observations and could therefore be omitted in our example.

A second option that may added to the `infile` command is the `missing` option to indicate missing values. Specifying for example `missing = M` defines the letter 'M' as an indicator for a missing value. The default for missing values are a period '.' and 'NA' (which remain valid indicators for missing values even if an additional indicator is defined by the `missing` option).

After having read in the data set information we can inspect the data visually. Executing the command

```
> d.describe
```

opens an *Object-Viewer* window containing the data in form of a spreadsheet. This can also be achieved by double-clicking on the *dataset object* in the *object browser*.

Further methods allow to examine the variables in the *dataset object*. For a categorial variable, e.g. `sex`, the `tabulate` command may be used to produce a frequency table:

```
> d.tabulate sex
```

resulting in

```
Variable: sex
```

Value	Obs	Freq	Cum
-1	2451	0.5057	0.5057
1	2396	0.4943	1

being printed in the *output window*. For continuous variables the `descriptive` command prints several characteristics of the variable in the *output window*. E.g., executing

```
> d.descriptive bmi
```

leads to

Variable	Obs	Mean	Median	Std	Min	Max
bmi	4847	21.944349	21.4	3.2879659	12.8	39.29

9.4.1.2 Map objects

In the following we want to estimate a spatially correlated effect of the district in which a child lives. Therefore we need the boundaries of the districts in Zambia to compute the neighborhood information of the map of Zambia. We therefore create a *map object*

```
> map m
```

and read in the boundaries using the `infile` command of *map objects*:

```
> m.infile using c:\data\zambia.bnd
```

Having read in the boundary information, *BayesX* automatically computes the neighborhood matrix of the map.

The file following the keyword `using` is assumed to contain the boundaries in form of closed polygons. Compare chapter 5 for a description of the expected file format.

Map objects may be visualized using method `describe`:

```
> m.describe
```

resulting in the graph shown in Figure 9.9. Additionally, `describe` prints further information about the *map object* in the *output window* including the name of the object, the number of regions, the minimum and maximum number of neighbors and the bandwidth of the corresponding adjacency or neighborhood matrix:

```
MAP m
Number of regions: 54
Minimum number of neighbors: 1
Maximum number of neighbors: 9
Bandsize of corresponding adjacency matrix: 24
```

Reading the boundary information from an external file and computing the neighborhood matrix may be a computationally intensive task if the map contains a huge number of regions or if the polygons are given in great detail. To avoid doing these computations every time *BayesX* is restarted, we may store the computed neighborhood information in a so-called *graph file* to make it directly available (compare chapter 5 for the structure of a graph file). To do this, we use the method `outfile` together with the `graph` option as in the following example:

```
> m.outfile, replace graph using c:\data\zambia.gra
```

Note, that specifying the option `replace` allows *BayesX* to overwrite an existing file with the same name. Without this option an error message would be raised if the given file is already existing.

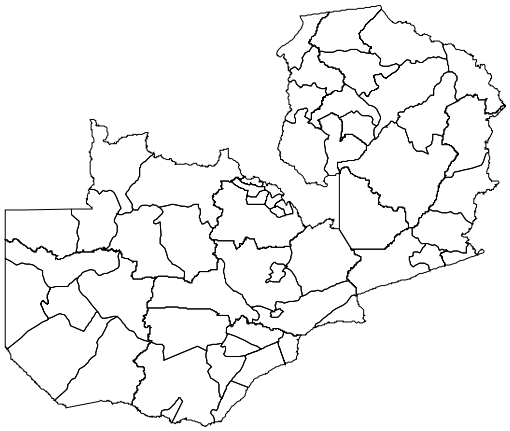


Figure 9.9: The districts within Zambia.

Leaving out the keyword `graph` in the above command leads to storage of the map in boundary format.

To see how storing maps in *graph files* affects the computation time of the `infile` command, we create a second *map object* and read in the information from the graph file. Again, we have to specify the keyword `graph`:

```
> map m1
> m1.infile, graph using c:\data\zambia.gra
```

As you should have noticed, reading geographical information from a *graph file* is usually much faster than reading from a *boundary file*. However, using *graph files* also has a drawback. Since they do no longer contain the full information on the polygons forming the map, we can not visualize a *map object* created from a *graph file*. Trying to do so

```
> m1.describe
```

raises an error message. This implies, that visualizing estimation results of spatial effects can only be based on *map objects* created from *boundary files*, although estimation can be carried out using *graph files*. Since we will work with the *map object* `m` in the following, we delete `m1`:

```
> drop m1
```

9.4.1.3 Bayesian semiparametric regression

To estimate a regression model based on mixed model techniques, we first create a *remlreg object*:

```
> remlreg r
```

By default estimation results are written to the subdirectory `output` of the installation directory. In this case the default filenames are composed of the name of the *remlreg object* and the type of the specific file. Usually it is more convenient to store the results in a user-specified directory. To define this directory we use the `outfile` command of *remlreg objects*:

```
> r.outfile = c:\data\r
```

Note, that `outfile` does not only specify a directory but also a base file name (the character 'r' in our example). Therefore executing the command above leads to storage of the results in the directory `c:\data` and all filenames start with the character 'r'. Of course the base file name may be different from the name of the *remlreg object*.

In addition to parameter estimates *BayesX* also produces some further information on the estimation process. In contrast to parameter estimates this information is not stored automatically but is printed in the *output window*. Therefore it is useful to store the contents of the *output window*. This can be achieved automatically by opening a *log file* using the `logopen` command

```
> logopen, replace using c:\data\logreml.txt
```

After opening a *log file*, every information written to the *output window* is also stored in the log file. Option `replace` allows *BayesX* to overwrite an existing file with the same name as the specified *log file*. Without `replace` results are appended to an existing file.

The model presented in Kandala et al. (2001) is given by the following semiparametric predictor:

$$\eta = \gamma_0 + \gamma_1 rcw + \gamma_2 edu1 + \gamma_3 edu2 + \gamma_4 tpr + \gamma_5 sex + f_1(bmi) + f_2(agc) + f^{str}(district) + f^{unstr}(district)$$

The two continuous covariates `bmi` and `agc` are assumed to have a possibly nonlinear effect on the Z-score and are therefore modelled nonparametrically (as P-splines with second order random walk prior in our example). The spatial effect of the district is split up into a spatially correlated part $f^{str}(district)$ and an uncorrelated part $f^{unstr}(district)$, see Fahrmeir and Lang (2001b) for a motivation. The correlated part is modelled by a Markov random field prior, where the neighborhood matrix and possible weights associated with the neighbors are obtained from the *map object* `m`. The uncorrelated part is modelled by an i.i.d. Gaussian effect.

To estimate the model we use method `regress` of *remlreg objects*:

```
> r.regress hazstd = rcw + edu1 + edu2 + tpr + sex + bmi(psplinerw2)
  + agc(psplinerw2) + district(spatial,map=m) + district(random),
  family=gaussian lowerlim=0.01 eps=0.0005 using d
```

Options `lowerlim` and `eps` control the estimation process. 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 chapter 7 in the manual for details). The option `eps` defines the termination criterion for the estimation process. The default values are `lowerlim=0.001` and `eps=0.00001`. However, since our analysis is only for explanatory purpose we chose somewhat weaker conditions resulting in a faster 'convergence' of the algorithm. On a 2.4 GHz PC estimation of our model took about 2 minutes and 30 seconds with the above specifications of `lowerlim` and `eps`.

A further option of method `regress` is `maxit`, defining the maximum number of iterations that should be performed in the estimation. Note, that *BayesX* produces results based on the current values of all parameters even if no convergence could be achieved within `maxit` iterations. In this case however a warning message will be printed in the *output window*.

In the following we reproduce the content of the *output window* to make the user familiar with the estimation results produced by *BayesX*:

```
ESTIMATION RESULTS:

  Estimated scale parameter: 0.802145

  Scale parameter is also stored in file
  c:\data\r_scale.res


  f_bmi_pspline
```

```
Estimated variance:  1.14816e-05
Inverse variance:    87095.9
Smoothing parameter: 69863.5
(Smoothing parameter = scale / variance)
NOTE: Estimation of the variance was stopped after iteration 6
      because the corresponding penalized part was small relative to the linear predictor.

Variance and smoothing parameter are stored in file
c:\data\r_f_bmi_pspline_var.res

Results are stored in file
c:\data\r_f_bmi_pspline.res

Postscript file is stored in file
c:\data\r_f_bmi_pspline.ps

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


f_agc_pspline

Estimated variance:  0.00322146
Inverse variance:    310.418
Smoothing parameter: 249
(Smoothing parameter = scale / variance)

Variance and smoothing parameter are stored in file
c:\data\r_f_agc_pspline_var.res

Results are stored in file
c:\data\r_f_agc_pspline.res

Postscript file is stored in file
c:\data\r_f_agc_pspline.ps

Results may be visualized using method 'plotnonp'
Type for example: objectname.plotnonp 2


f_district_spatial

Estimated variance:  0.0294012
Inverse variance:    34.0123
Smoothing parameter: 27.2828
(Smoothing parameter = scale / variance)

Variance and smoothing parameter are stored in file
c:\data\r_f_district_spatial_var.res

Results are stored in file
c:\data\r_f_district_spatial.res

Postscript file is stored in file
c:\data\r_f_district_spatial.ps

Results may be visualized in BayesX using method 'drawmap'
Type for example: objectname.drawmap 3
```

```
f_district_random

Estimated variance:  0.00806668
Inverse variance:    123.967
Smoothing parameter: 99.4393
(Smoothing parameter = scale / variance)

Variance and smoothing parameter are stored in file
c:\data\r_f_district_random_var.res

Results for random effects are stored in file
c:\data\r_f_district_random.res


FixedEffects

Variable  Post. Mode     Std. Dev.      p-value        95% Confidence Interval
const     0.0610357      0.0341574      0.0367456      -0.00592622    0.127998
rcw       0.00767158     0.0136564      0.286931       -0.0191003     0.0344434
edu1      -0.0605105     0.0261369      0.0103181      -0.111749      -0.00927192
edu2      0.234917       0.0459925      4.41249e-06    0.144754       0.325081
tpr       0.0904093      0.0218891      6.17648e-05    0.047498       0.133321
sex       -0.0585716     0.0129304      2.04243e-05    -0.0839203     -0.0332229

Results for fixed effects are also stored in file
c:\data\r_FixedEffects.res


Additive predictor and expectations

Additive predictor and expectation for each observation are stored in file
c:\data\r_predict.raw

Files of model summary:

------------------------------------------------------------------------------

Batch file for visualizing effects of nonlinear functions is stored in file
c:\data\r_graphics.prg

NOTE: 'input filename' must be substituted by the filename of the boundary-file

------------------------------------------------------------------------------

Batch file for visualizing effects of nonlinear functions
in S-Plus is stored in file
c:\data\r_splus.txt

NOTE: 'input filename' must be substituted by the filename of the boundary-file

------------------------------------------------------------------------------

Latex file of model summaries is stored in file
c:\data\r_model_summary.tex

------------------------------------------------------------------------------
```

In addition to the information being printed to the *output window* results for each effect are written to external ASCII files. The names of these files are given in the *output window*, compare the previous pages. For the variance parameters the files contain the variance as well as the corresponding smoothing parameter. For the different terms of the model the files contain the posterior mode,

the 80% and 95% credible interval, the standard deviations and the corresponding 95% and 80% posterior probabilities of the estimated effects. Note, that credible intervals and posterior probabilities are based on a normal approximation of the posterior. For example, the beginning of the file `c:\data\r_f_bmi_pspline.res` for the effect of `bmi` looks like this:

intnr	bmi	pmode	ci95lower	ci80lower	std	ci80upper	ci95upper	pcat95	pcat80
1	12.8	-0.22305	-0.304661	-0.276409	0.0416301	-0.169692	-0.141439	-1	-1
2	13.15	-0.215246	-0.292828	-0.26597	0.0395749	-0.164522	-0.137663	-1	-1
3	14.01	-0.19607	-0.264173	-0.240597	0.0347394	-0.151544	-0.127968	-1	-1

The credible intervals and posterior probabilities that are computed for every effect may be changed by the user using the options `level1` and `level2`. For example specifying `level1=99` and `level2=70` in the option list of the `regress` command leads to the computation of 70% and 99% credible intervals and posterior probabilities. The defaults are `level1=95` and `level2=80`.

Some nonparametric effects are visualized by *BayesX* automatically and the resulting graphs are stored in postscript format. E.g. the effect of `bmi` is visualized in the file `c:\data\b_f_bmi_pspline.ps` (compare the results on the previous pages for the other filenames). A batch file to reproduce the plots is stored in the output directory. In our example the name of the file is `c:\data\r_graphics.prg`. The advantage is that additional options may be added by the user to customize the graphs (compare the following two subsubsections).

Moreover a file with ending `.tex` is created in the outfile directory. This file contains a summary of the estimation results and may be compiled using LaTeX.

Having finished the estimation we may close the *log file* by typing

```
> logclose
```

Note, that the *log file* is closed automatically when you exit *BayesX*.

9.4.1.4 Visualizing estimation results

BayesX provides three possibilities to visualize estimation results:

- As mentioned in the previous subsubsection, certain results are automatically visualized by *BayesX* and stored in postscript files.
- Post estimation commands of *bayesreg objects* allow to visualize results after having executed a `regress` command.
- *Graph objects* may be used to produce graphics using the ASCII files containing the estimation results. In principle *graph objects* allow the visualization of any content of a *dataset object*. *Graph files* are also used in the batch file containing the commands to reproduce the automatically generated graphics.

In this subsubsection we describe the general usage of the post estimation commands as well as the commands for the usage with *graph objects* to enable the user to reproduce the automatically generated plots directly in *BayesX*. subsubsection 9.4.1.5 describes how to customize plots.

Post estimation commands After having estimated a regression model plots for nonparametric effects of metrical covariates can be produced using the post estimation command `plotnonp`:

```
> r.plotnonp 1
```

and

```
> r.plotnonp 2
```

produce the graphs shown in Figure 9.10 in an *object-viewer window*. The numbers following the `plotnonp` command depend on the order in which the model terms have been specified. The numbers are supplied in the *output window* after estimation, compare the results in the previous subsubsection.

By default the plots contain the posterior mode and pointwise credible intervals according to the levels specified in the `regress` command. So by default the plots include pointwise 80% and 95% credible intervals.

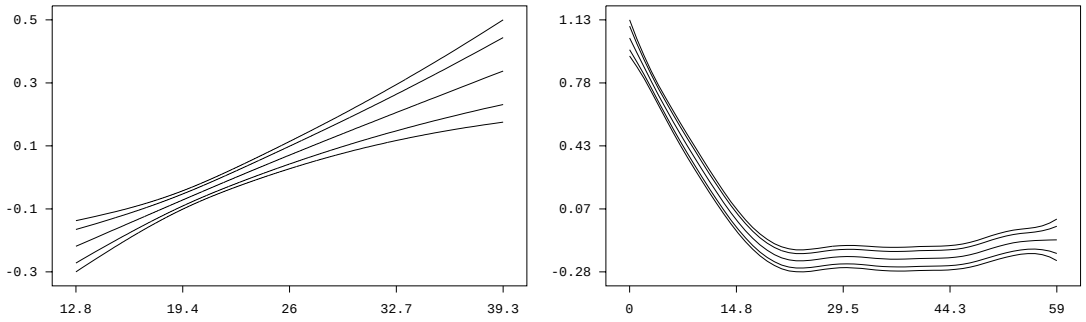


Figure 9.10: Effect of the body mass index of the child's mother and of the age of the child together with pointwise 80% and 95% credible intervals.

A plot may be stored in postscript format using the `outfile` option. Executing

```
> r.plotnonp 1, replace outfile = c:\data\f_bmi.ps
```

stores the plot for the estimated effect of `bmi` in the file `c:\data\f_bmi.ps`. Again, specifying `replace` allows *BayesX* to overwrite an existing file. Note, that *BayesX* does not display the graph on the screen if the option `outfile` is specified.

Estimation results for spatial effects are best visualized by drawing the respective map and coloring the regions of the map according to some characteristic of the posterior, e.g. the posterior mode. For the structured spatial effect this can be achieved using the post estimation command `drawmap`

```
> r.drawmap 3
```

which results in the graph shown in Figure 9.11.

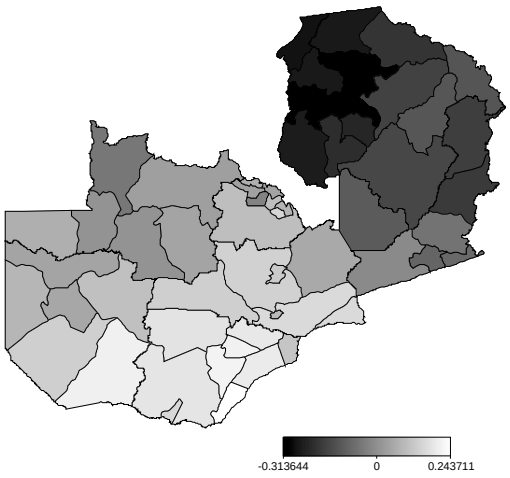


Figure 9.11: Posterior mode of the structured spatial effect.

Graph Objects The commands presented in the previous paragraph work only after having estimated a regression model in the current *BayesX* session but it may also be useful to visualize results of former analyzes. This can be achieved using *graph objects*. Note again, that *graph files* are also used in the batch file containing the commands to reproduce the automatically generated graphics. Therefore the purpose of this paragraph is also to enable the user to understand the content of this batch file.

First we read the estimation results into a *dataset object*. For example the estimation results for the effect of `bmi` can be read into *BayesX* by executing the commands

```
> dataset res
> res.infile using c:\data\r_f_bmi_pspline.res
```

Now the estimation results (or any content of a *dataset object*) may be visualized using a *graph object* which we create by typing

```
> graph g
```

The results stored in the *dataset object* `res` are now visualized using the `plot` command of *graph objects*. Executing

```
> g.plot bmi pmode ci95lower ci80lower ci80upper ci95upper using res
```

reproduces the graph in Figure 9.10.

Similar as for `plotnonp`, the direct usage of the `drawmap` command is only possible after executing a `regress` command. However, using *graph objects* again allows us to visualize results that have been stored in a file.

First we read the information contained in this file into a *dataset object*. For example the following command

```
> res.infile using c:\data\r_f_district_spatial.res
```

stores the estimation results for the structured spatial effect in the *dataset object* `res`. Now we can visualize the posterior mode using method `drawmap` of *graph objects* leading again to the graph shown in Figure 9.11:

```
> g.drawmap pmode district, map=m using res
```

Since – in contrast to a *remlreg object* – no *map object* is associated with a *graph object*, we explicitly have to specify the map that we want to use in the option list.

Using *graph objects* also allows us to plot other characteristics of the posterior than the posterior mode. For instance the posterior 95% probabilities may be visualized by

```
> g.drawmap pcat95 district, map=m using res
```

The result is shown in Figure 9.12.

A further advantage of *graph objects* is, that they allow to visualize the estimation results for the uncorrelated spatial effects. Since these are modelled as unstructured random effects, *BayesX* is unable to recognize them as spatial effects. However, proceeding as follows gives us the possibility to plot the unstructured spatial effect shown in Figure 9.13:

```
> res.infile using c:\data\r_f_district_random.res
> g.drawmap pmode district, map=m color swapcolors using res
```

9.4.1.5 Customizing graphics

This subsubsection describes how to customize graphics created in *BayesX*. All options are described for the usage with the post estimation commands but may be used with graph files as well. So the

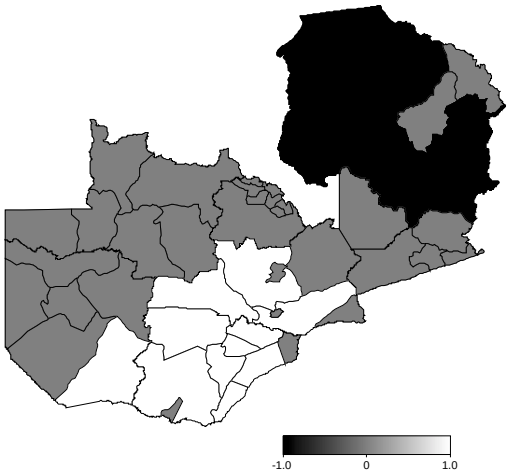


Figure 9.12: Posterior 95% probability of the structured spatial effect.

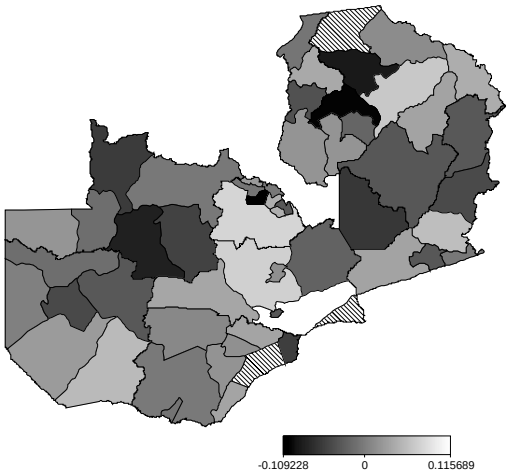


Figure 9.13: Posterior mode of the unstructured spatial effect.

options presented in this subsubsection also enable the user to modify the batch file containing the commands to reproduce the automatically generated graphics.

For the presentation of nonparametric effects it may be desirable to include only one of the credible interval into the plot. This is achieved by specifying the `levels` option. Possible values of this option are `1` and `2`, corresponding to the levels specified in the `regress` command (compare subsubsection 9.4.1.3). If the default values of `level1` and `level2` have been used, specifying `level=2` in the `plotnonp` command causes *BayesX* to plot the 80% credible interval only (Figure 9.14):

```
> r.plotnonp 1, levels=2
```

It may be useful to add some more information to the graphs of nonparametric effects to distinguish more obviously between different covariates. Ways to do so are the specification of a title or the specification of axis labels. Both possibilities are supported by *BayesX* as demonstrated in the following examples (compare Figure 9.15 for the resulting plots):

```
> r.plotnonp 1, title="Mother body mass index"
> r.plotnonp 1, xlab="bmi" ylab="f_bmi" title="Mother body mass index"
```

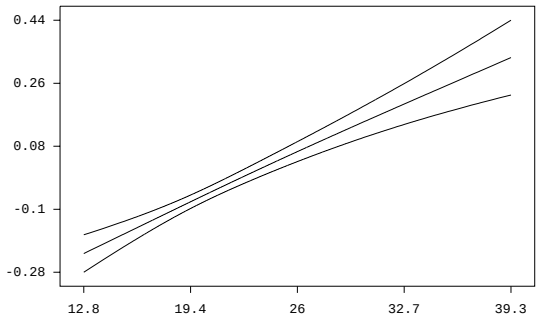


Figure 9.14: Effect of the body mass index of the child's mother with pointwise 80% credible intervals only.

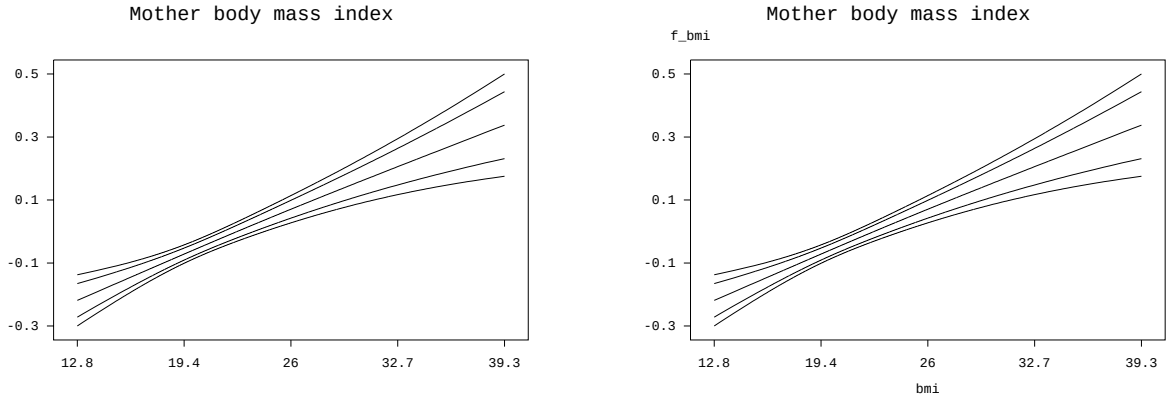


Figure 9.15: Specification of title and axis labels.

By default *BayesX* displays x- and y-axis with five equidistant ticks according to the range of the data that is to be visualized. These defaults may be overwritten using the options `xlimbottom`, `xlimtop` and `xstep` for the x-axis and `ylimbottom`, `ylimtop` and `ystep` for the y-axis, respectively. The usage of these options is more or less self-explanatory and is demonstrated in the following commands which lead to the graph shown in Figure 9.16.

```
> r.plotnonp 1, xlab="bmi" ylab="f_bmi" title="Mother body mass index"
  ylimbottom=-0.8 ylimtop=0.6 ystep=0.2 xlimbottom=12 xlimtop=40
```

Figure 9.16 also includes a graph for the effect of the age of the child that is customized in the same way as for the effect of `bmi`.

```
> r.plotnonp 2, xlab="age" ylab="f_age" title="Age of the child in months"
  ylimbottom=-0.3  ystep=0.3 xlimbottom=0 xlimtop=60 xstep=10
```

Now we turn to the options for method `drawmap`. By default `drawmap` uses grey scales to represent different values of the posterior mode. Using the option `color` forces *BayesX* to use different colors instead. Here the default would be to represent higher values through green colors and smaller values through red colors. Specifying `swapcolors` switches this definition. Therefore the following command

```
> r.drawmap 3, color swapcolors
```

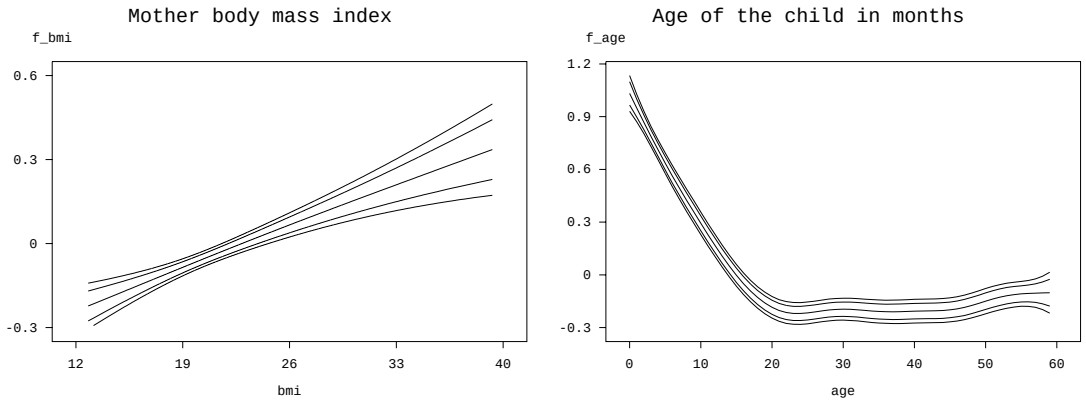


Figure 9.16: Re-defining x- and y-axis.

leads to the graph shown in Figure 9.17 with higher values being represented through red colors and smaller values through green colors.

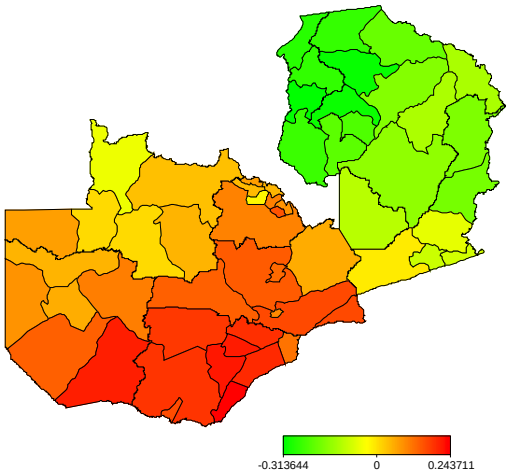


Figure 9.17: Posterior mode of the structured spatial effect in color.

Similar options as for the visualization of nonparametric effects exist for method `drawmap`. For example, a title may be included by specifying the option `title`

```
> r.drawmap 3, color swapcolors title="Structured spatial effect"
```

or the range of values to be displayed may be defined using the options `lowerlimit` and `upperlimit`:

```
> r.drawmap 3, color swapcolors title="Structured spatial effect" lowerlimit=-0.3
  upperlimit=0.3
```

The graph produced by the second command is shown in Figure 9.18.

9.5 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.

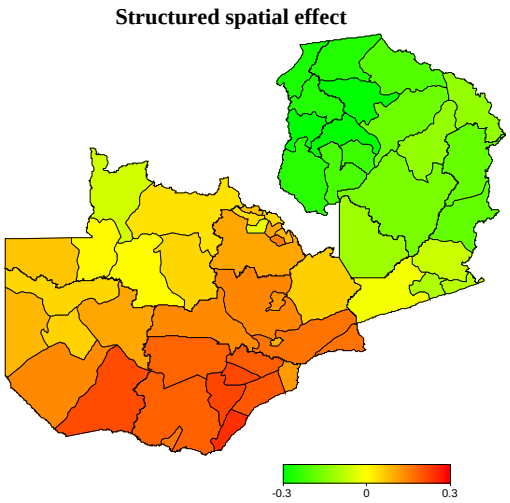


Figure 9.18: Specifying a title and the range of the plot for spatial effects.

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 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:

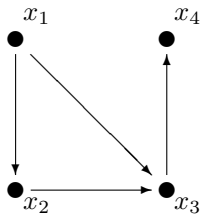
$$\begin{aligned} & p(\boldsymbol{x}, \boldsymbol{\beta}, \boldsymbol{\sigma}^2, d) \\ &= p(\boldsymbol{x} \mid \boldsymbol{\beta}, \boldsymbol{\sigma}^2, d)\, p(\boldsymbol{\beta} \mid \boldsymbol{\sigma}^2, d)\, p(\boldsymbol{\sigma}^2 \mid d)\, p(d) \\ &= \prod_{i=0}^{p-1} p(\boldsymbol{x}_i \mid \boldsymbol{x}_{pa(i)}, \boldsymbol{\beta}_{i|pa(i)}, \sigma^2_{i|pa(i)}) \prod_{i=0}^{p-1} p(\boldsymbol{\beta}_{i|pa(i)} \mid \sigma^2_{i|pa(i)}) \\ &\quad \prod_{i=0}^{p-1} p(\sigma^2_{i|pa(i)})\, p(d) \end{aligned}$$

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 subsection 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 2.4.1; 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 89 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.

  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

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.

- **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 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

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.

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 $\boldsymbol{\beta}_{i|-i}$, $i = 0, \ldots, p-1$, 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