BayesX

Software for Bayesian Inference in Structured Additive Regression Models

Version 3.0

Reference Manual

Developed by

Christiane Belitz
Andreas Brezger
Nadja Klein (University of Göttingen)
Thomas Kneib (University of Göttingen)
Stefan Lang (University of Innsbruck)
Nikolaus Umlauf (University of Innsbruck)

With contributions by

Daniel Adler
Jan Fahrenholz
Eva-Maria Fronk
Felix Heinzl
Andrea Hennerfeind
Manuela Hummel
Alexander Jerak
Susanne Konrath
Petra Kragler
Cornelia Oberhauser
Leyre Estíbaliz Osuna Echavarría
Daniel Sabanés Bové
Achim Zeileis

Supported by

Ludwig Fahrmeir (mentally)
Leo Held (mentally)
German Research Foundation (DFG)

Acknowledgements

The development of *BayesX* has been supported by grants from the German Research Foundation (DFG), Collaborative Research Center 386 "Statistical Analysis of Discrete Structures".

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

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

Licensing agreement

BayesX is available at http://www.bayesx.org

Contents

Chapter 1

What is BayesX?

General scope

BayesX is a software tool for estimating structured additive regression models. Structured additive regression embraces several well-known regression models such as generalized additive models (GAM), generalized additive mixed models (GAMM), generalized geoadditive mixed models (GGAMM), dynamic models, varying coefficient models, and geographically weighted regression within a unifying framework. Besides exponential family regression, *BayesX* also supports non-standard regression situations such as regression for categorical responses, hazard regression for continuous survival times, continuous time multi-state models, quantile regression, distributional regression models and multilevel models.

Inferential procedures

Estimation of regression models can be achieved based on four different inferential concepts that have been implemented in separate regression objects:

- **MCMC simulation techniques (bayesreg objects):** A fully Bayesian interpretation of structured additive regression models is obtained by specifying prior distributions for all unknown parameters. Estimation can be facilitated using Markov chain Monte Carlo simulation techniques, a general and versatile concept for Bayesian inference. Bayesreg objects provide numerically efficient implementations of MCMC schemes for structured additive regression models in case of exponential family responses, categorical responses, hazard regression and multi-state models. Suitable proposal densities have been developed to obtain rapidly mixing, well-behaved sampling schemes without the need for manual tuning.
- **MCMC simulation techniques (mcmcreg objects):** Mcmcreg objects provide similar functionality for fully Bayesian inference as bayesreg objects but implement distributional regression models for responses beyond simple exponential families (distributional regression), quantile regression and multilevel models. Generally, estimation is more efficient (in terms of computing time) than with bayesreg objects. Therefore mcmcreg objects should be preferred to bayesreg objects if possible.
- **Mixed model based estimation (remlreg objects):** An increasingly popular way to estimate semiparametric regression models is the representation of penalisation approaches as mixed models. Within *BayesX* this concept has been extended to structured additive regression models and several types of non-standard regression situations. The general idea is to take advantage of the close connection between penalty concepts and

corresponding random effects distributions. The smoothing parameters of the penalties then transform to variance components in the random effects (mixed) model. While the selection of smoothing parameters has been a difficult task for a long time, several estimation procedures for variance components in mixed models are already available since the 1970's. The most popular one is restricted maximum likelihood in Gaussian mixed models with marginal likelihood as the non-Gaussian counterpart. Remlreg objects employ mixed model methodology for the estimation of structured additive regression models. While regression coefficients are estimated based on penalised likelihood, restricted maximum likelihood or marginal likelihood estimation forms the basis for the determination of smoothing parameters. From a Bayesian perspective, this yields empirical Bayes / posterior mode estimates for the structured additive regression models. However, estimates can also merely be interpreted as penalised likelihood estimates from a frequentist perspective.

– **Penalized least squares including model selection (stepwisereg objects):** As a fourth alternative *BayesX* provides a penalized least squares (respectively penalized likelihood) approach for estimating structured additive regression tools. In addition to the previously described estimation alternatives, a powerful variable and model selection tool is included. Model choice and estimation of the parameters is done simultaneously. The algorithms are able to
  * decide whether a particular covariate enters the model,
  * decide whether a continuous covariate enters the model linearly or nonlinearly,
  * decide whether a spatial effect enters the model,
  * decide whether a unit- or cluster specific heterogeneity effect enters the model,
  * select complex interaction effects (two dimensional surfaces, varying coefficient terms),
  * select the degree of smoothness of nonlinear covariate, spatial or cluster specific heterogeneity effects.

  Inference is based on penalized likelihood in combination with fast algorithms for selecting relevant covariates and model terms. Different models are compared via various goodness of fit criteria, e.g. AIC, BIC, GCV and 5 or 10 fold cross validation.

Model classes and model terms

BayesX provides functionality for the following types of responses:

– **Univariate exponential family:** Supported response distributions are Gaussian, Poisson, Binomial and Gamma distribution.

– **Distributional regression:** A large number of univariate and multivariate continuous, discrete or mixed discrete-continuous responses can be treated within the framework of distributional regression. In this setting, potentially all parameters of these distributions can be related to structured additive predictors.

– **Quantile Regression:** Bayesian quantile regression allows to study specific quantiles of the response distribution without relying on a specific distributional assumption.

– **Categorical responses with unordered responses:** For categorical responses with unordered categories, *BayesX* supports multinomial logit and multinomial probit models. Both effects of category-specific and globally-defined covariates can be estimated. Category-specific offsets or non-availability indicators can be defined to account for varying choice sets.

- **Categorical responses with ordered responses:** For ordered categorical responses, ordinal as well as sequential models can be specified. Effects can be requested to be category-specific or to be constant over the categories. Supported response functions include the logit and the probit transformation.
- **Continuous time survival models:** *BayesX* supports Cox-type hazard regression models with structured additive predictor for continuous time survival analysis. In contrast to the Cox model, the baseline hazard rate is estimated jointly with the remaining effects based on penalized splines. Furthermore, both time-varying effects and time-varying covariates can be included in the predictor. Arbitrary combinations of right, left and interval censored as well as left truncated observations can be analysed.
- **Continuous time multi-state models:** Multi-state models form a general class for the analysis of the evolution of discrete phenomena in continuous time. Transition intensities between the discrete states are specified in analogy to the hazard rate in continuous time survival models.

Structured additive regression models can be build from arbitrary combinations of the following model terms:

- **Nonlinear effects:** Nonlinear effects can be estimated based on either penalised spline or random walk models.
- **Seasonal effects:** Specific autoregressive priors allow for the estimation of flexible, time-varying seasonal effects.
- **Spatial effects:** Spatial effects can be specified based on Markov random fields, stationary Gaussian random fields (kriging) or bivariate penalised splines. Both georeferenced regional data as well as point-referenced data based on coordinates are supported.
- **Interaction surfaces:** Bivariate extensions of penalised splines allow to estimate flexible interactions between continuous covariates. Stationary Gaussian random fields can also be considered a radial basis function approach and, hence, form a second possibility for the specification of interaction surfaces.
- **Varying coefficients:** Varying coefficient models with both continuous and spatial effect modifiers can be estimated. The latter case is also known as geographically weighted regression.
- **Cluster-specific random effects:** *BayesX* supports i.i.d. Gaussian random intercepts and random slopes.
- **Regularised high-dimensional effects:** High-dimensional vectors of regression coefficients can be assigned Bayesian regularisation priors. Available alternatives are ridge, lasso, and normal mixture of inverse gamma (spike and slab) priors.
- **Multilevel models:** In multilevel models, parameters of specific effects can themselves be assigned a structured additive predictor (e.g. in multilevel random effects specifications).

Note that parts of the functionality may be available for one of the regression objects only. For example, bayesreg objects do not support interval censored survival times while multinomial probit models can not be estimated with remlreg objects. Details can be found in the chapters corresponding to the specific object types.

Further functionality

- **Handling and manipulation of data sets:** *BayesX* provides a number of functions for handling and manipulating data sets, e.g. for reading ASCII data sets, creating new variables, obtaining summary statistics etc. Compare chapter 4 for details.

- **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. In *BayesX* version 1.5, a new color scheme based on HCL colors has been added to obtain a better representation of colored maps independent of the display device. Details can be found in 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.
- **Model selection for Gaussian and non-Gaussian dags:** This tool estimates Gaussian and non-Gaussian directed acyclical graphs (dag) via reversible jump MCMC. Details can be found in chapter 12.

Chapter 2

Getting started

This chapter provides some useful information for first-time users of *BayesX*: Which versions of BayesX are currently available (section 2.1), how is BayesX installed (section 2.2), what types of manuals exist (section 2.3), and how is the graphical user interface organized (section 2.4). Section 2.5 describes the general usage of *BayesX* and the structure of *BayesX* syntax. The final section contains a description of three data sets that will be used for demonstrating purposes in the later chapters.

2.1 Available versions of BayesX

Currently, *BayesX* is available in two different versions. The first one includes a graphical user interface that enables visualisation of estimated effects. While the computational kernel of *BayesX* has been implemented in C++, the graphical user interface has been implemented in Java.

The second version of *BayesX* is a command line version that is based purely on C++ and comes without any graphics facilities. While the GUI version is provided as a pre-compiled binary with an easy to use installer, the command line version is provided in terms of source code and has to be compiled on your system. A makefile is provided that assumes that the GNU C++ compiler is available on your system. Hence, the command line version is suitable for any operating system that supports the GNU compiler family and has been successfully tested on Windows, Linux and Mac OS. In addition, the cmake toolchain can be used to generate customized makefiles.

As a supplement to both versions, supplementing R packages are available from CRAN (http://www.r-project.org). The package *BayesX* provides functionality for reading, creating and manipulating maps in R as well as some customized visualisation tools. The packages *R2BayesX* and *BayesR* provide access to *BayesX* from within R such that models can be estimated in the usual R formula syntax. Note that not all features described in this manual will be supported by the two latter packages.

The current releases of both *BayesX* versions can be downloaded from http://www.bayesx.org.

Most of this manual applies for both versions of *BayesX*. However, those parts that deal with visualisation of data sets or estimation results are exclusively for the GUI version.

2.2 Installing BayesX

Depending on the version of *BayesX* you are planning to use, there are two different ways to install it on your computer. For the GUI version, you have to download the installation routine from the *BayesX* homepage and execute it (with administrator privileges for Windows Vista). The

installation routine will request all necessary information during the installation process. Note that it is recommended to install BayesX in a directory without spaces in the path name. When *BayesX* has been installed successfully, it can be started using the *Windows Start* button or the icon created on the desktop (depending on your specifications during the installation process).

For the command line version, you have to unpack the zip archive containing the source code of *BayesX*. If you have the make facility available, you can simply type `make BayesX` in the shell and *BayesX* will be compiled. Depending on your operating system, some minor modifications of the make file (for example relating to the version of the GNU compiler you have installed or the location of the readline library) may be necessary.

After a successful installation, there should be four subdirectories in the installation directors. The `doc` directory contains the program documentation, i.e. the three *BayesX* manuals (see the following subsection). The `examples` directory contains the three data sets, `credit.raw`, `rents.raw` and `zambia.raw`, which will be used for demonstrating purposes throughout the manual. A detailed description of these data sets is given in section 2.6. The `examples` directory also contains some tutorial programs that illustrate the usage of *BayesX*, see the tutorials manual. The `output` directory is the default directory for program output stored in files. Of course, the output window can be redefined by the user, compare section 7.4, section 8.2 and section 9.2. Finally, temporary files created when estimating regression models will be stored in the `temp` directory. Usually you will never use this directory.

The subdirectories and their content are briefly summarized in Table 2.1.

Directory	Content
`doc`	the *BayesX* manuals
`examples`	data set examples and tutorial programs
`output`	default directory for estimation output
`temp`	temporary files

Table 2.1: Subdirectories of the installation directory and their content.

2.3 Manuals

BayesX is shipped with three different manuals. The reference manual (i.e. the manual you are just reading) gives detailed information on the general usage of BayesX, the syntax of *BayesX* commands and the different objects used by *BayesX*. The methodology manual provides background information on the statistical methodology that is implemented in *BayesX*. In this manual, you will also find more references on the methodological background. The tutorial manual is intended to make new users familiar with the usage of *BayesX* by demonstrating examples. It contains three self-contained tutorials, describing how to perform semiparametric regression analyses using *BayesX*. All three manuals can be found in the `doc` directory (a subdirectory of the installation directory). In the GUI version of *BayesX*, the manuals are also available from the help menu.

2.4 Windows and buttons in BayesX

This section only applies to the GUI version of *BayesX*

After starting *BayesX* you will see a main window with a menu bar and four additional subwindows. 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 there is a menu bar containing the buttons BREAK, PAUSE and SUPPRESS OUTPUT, and the priority menu. Their functionality is described in subsection 2.4.5 and 2.4.6.

2.4.1 The command window

Allmost all *BayesX* commands are entered and executed in the *command window*. By default, a command will be executed if you press the return key. You can change this default delimiter using the `delimiter` command, see section 3.4.

2.4.2 The output window

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

The content 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 file save dialog will allow you to choose between two different file formats. The default is the rich-text format but it is also possible to store the *output window* in plain ASCII format. This, however, has the disadvantage that all text highlights (for example bold letters) will disappear in the saved file.

The *file menu* also allows to clear the *output window* (i.e. delete the content of the window) or to 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 varied in the *preferences menu*.

2.4.3 The review window

In many cases, subsequent commands change only slightly. The *review window* gives you convenient access to the last 100 past commands entered during a session. Double click on one of these past commands and it is automatically copied to the *command window*, where it can be modified and / or executed again.

2.4.4 The object browser

BayesX is object oriented, i.e. different types of objects are used to store data, estimate regression models, etc. The *object browser* provides an overview of the objects currently defined and about their contents. The *object browser* window is split into two parts. The left part displays the different object types currently supported by *BayesX* (*dataset objects*, *bayesreg objects*, *remlreg objects*, *map objects*, *dag objects* and *graph objects*)s. 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 and / or a short summary in the *output window*, depending on the object type. Double clicking on *dataset objects*, for example, will open a spreadsheet where the variables and the observations of the data set can be inspected. Clicking on *map objects* opens a window that contains a graphical representation of the map.

2.4.5 BREAK, PAUSE and SUPPRESS OUTPUT button

The *BayesX* button panel contains the BREAK button, the PAUSE button and the SUPPRESS OUTPUT button. The purpose of the BREAK button is to interrupt the process that is currently executed (this may take some time). Clicking on the PAUSE button interrupts the current process temporarily until the button is pressed again. If a process is paused, the button caption PAUSE

is replaced by CONTINUE, indicating that a second click on the button will continue the current process. Pausing a current process will increase the execution speed of other programs currently running on your computer. Clicking the SUPPRESS OUTPUT button suppresses printing of 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 increases the execution speed of *BayesX* and saves memory. Note, that you can store your output in a log-file even if printing of the output is suppressed (see section 3.2).

2.4.6 Priority menu

When running extensive computations, it may be desirable to reduce the priority of BayesX since otherwise all further programs may be executed very slowly. The priority menu allows you to change the priority of your computations from within BayesX. Usually there should be no need to increase the priority (although it is possible). To pause the current computations use the PAUSE button.

2.5 General usage of BayesX

2.5.1 Creating objects

BayesX is implemented in an object oriented way, although the object oriented concept does not go too far, i.e. inheritance or other concepts of object oriented programming languages such as S or C++ are not supported. As a consequence, the first thing to do during a session, is to create some objects. Currently, eight different object types are available: *dataset objects*, *bayesreg objects*, *mcmcreg objects*, *remlreg objects*, *stepwisereg objects*, *map objects*, *dag objects* and *graph objects*.

Dataset objects are used to store, handle, and manipulate data sets, see chapter 4 for details. *Map objects* are used to handle geographical information and are covered in more detail in chapter 5. The main purpose of *map objects* is to serve as auxiliary objects for regression objects when estimating spatial effects. *Graph objects* are used to visualize data (e.g. to create scatterplots or to color geographical maps according to some numerical characteristics), see chapter 6 for details.

The most important object types are *bayesreg objects*, *mcmcreg objects*, *remlreg objects* and *stepwisereg objects*. These objects are used to estimate Bayesian semiparametric regression models based on the different inferential approaches implemented in *BayesX* (see chapter 7 for *bayesreg objects*, chapter 8 for *remlreg objects*, chapter 9 for *stepwisereg objects* and chapter 10 for *mcmcreg objects*. *Dag objects* are used to estimate Gaussian or non-Gaussian DAGs (direct acyclic graphs) based on reversible jump MCMC simulation techniques (see chapter 12 for details).

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 some restrictions are imposed on the names of objects, i.e. not all object names are allowed. For example, object names have to begin with an uppercase or lowercase letter rather than a number. Section 4.13 discusses valid variable names but the same rules apply also to object names.

2.5.2 Applying methods to previously defined objects

When an object has been created successfully, you can apply methods to that particular object. For instance, *dataset objects* may be used to read 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 to 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 to
methodname	the name of the method
model	a model specification (for example a regression model)
`weight` *varname*	specifies *varname* as 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 is required to apply the particular method

Table 2.2: Parts of the general BayesX syntax.

Note that [...] indicates that this part of the syntax is optional and may be omitted. Moreover for most methods only some of the syntax parts above will be meaningful. 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 that a *dataset object* with name `mydata` has already been created and that some variables should be created. First of all, we have to tell *BayesX* how many observations we want to create. This can be done with the `set obs` command, see also section 4.10. For example

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

indicates that the data set `mydata` should have 1000 observations. In this case, 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. Similar as for the `set obs` command other syntax parts are not meaningful and therefore not allowed. If the negative values of `X` should be replaced with the constant 0, this can be achieved using the `replace` command:

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

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.

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

2.6 Description of data set examples

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

2.6.1 Rents for flats

According to the German rental law, owners of apartments or flats can base an increase in the amount that they charge for rent on 'average rents' for flats comparable in type, size, equipment, quality and location in a community. To provide information about these 'average rents', most of the larger cities publish 'rental guides', which can be based on regression analyses with rent as the dependent variable. The file `rent94.raw` (stored in the `examples` directory) is a subsample of data collected in 1994 for the Munich rental guide. 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 (see Table 2.3).

In addition to the data set, the `examples` directory contains a map of Munich in the file `munich.bnd`. This map will be useful for visualizing effects of the location `L`. See chapter 5 for a description on how to incorporate geographical maps into *BayesX*.

2.6.2 Credit scoring

The aim of credit scoring is to model and / or predict the probability that a client with certain covariates ('risk factors') will not pay back his credit. The data set contained in the file `credit.raw` consists of 1000 consumer credits from a bank in southern Germany. The response variable is 'creditability' in dichotomous form ($y = 0$ for creditworthy, $y = 1$ for not creditworthy). In addition, 20 covariates that are 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 & Tutz (2001, Ch. 2.1). Table 2.4 gives a description of the variables of the data set. Usually a binary logit model is applied to estimate the effect of the covariates on the probability of being not creditworthy. As in the case of the rents for flats example, this data set is used to demonstrate the usage of certain features of *BayesX*, see primarily subsection 7.6.1 for a Bayesian regression analysis of the data set.

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, continuous
amount	amount of credit in 1000 DM, continuous
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.

2.6.3 Childhood undernutrition in Zambia

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

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

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

This data set is used in the tutorial manual.

Variable	Description
hazstd	standardized Z-score for stunting
bmi	body mass index of the mother
agc	age of the child
district	district where the mother 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.

Chapter 3

Special Commands

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

3.1 Exiting BayesX

You can exit *BayesX* by typing either

```
> exit
```

or

```
> quit
```

in the *command window* / on the command line. Of course, the GUI version of *BayesX* can also be closed using the 'exit' entry from the file menu or by clicking on the cross in the upper right corner.

3.2 Opening and closing log files

Program output and commands entered by the user can automatically be stored in a log file to make them available for editing in your favorite text editor. Another important application of log files is the documentation of your work. A log file is opened by the command:

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

Afterwards all commands entered and all program output will be saved in the file *filename*. 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` has to 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 typing:

```
> logclose
```

Exiting *BayesX* automatically closes the current log file.

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. This is particularly useful for automatic

saving 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 stored.

Options

- `replace`
  By default an error will be raised if you try 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 in two different file types, namely rich text format (the default) and plain ASCII (requested by specifying `type=txt` as an option.

3.4 Changing the delimiter

By default, commands entered using the *command window* will be executed after pressing the return key. This can be inconvenient, in particular if your statements are long or in batch files. In this case it may be favorable to split a statement into several lines, and execute the command using a different delimiter.

You can change the delimiter using the `delimiter` command. The syntax is

`> delimiter` = *newdel*

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

```
> delimiter = ;
```

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

```
> delimiter = return;
```

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

Finally, note that it is not possible to change the delimiter for the command line. However, batch files (see the next section) with changed delimiter can also be used with the command line version of *BayesX*.

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.

Execution of a batch file is started by 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 currently running batch file.

Comments

Comments in batch files are indicated by a `%` sign, i.e. every line starting with `%` 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. By default this will cause errors, because *BayesX* interprets each line in your program as one statement. To overcome this problem one 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

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

`> dataset` *objectname*

where *objectname* is the name to be assigned to the data set. After the creation of a *dataset object* you can apply the methods discussed below.

Note that in the current version of *BayesX* only numerical variables are allowed. String valued variables, for example, are not yet supported by *BayesX* and the attempt to read such variables will raise an error message.

4.1 Method descriptive

Description

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

Syntax

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

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

Options

not allowed

Example

The statement

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

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

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

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

4.2 Method drop

Description

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

Syntax

> *objectname*.`drop` *varlist*

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

The first command is used to delete the variables specified in *varlist* from the data set. The second statement deletes certain observations, i.e. all observations for which *expression* is true will be deleted.

Options

not allowed

Example

The statement

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

deletes 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 deleted. The statement

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

will raise the error

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

because is not allowed to drop variables and 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 statements 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

Three different types of operators can be included in expressions: arithmetic, relational and logical operators.

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 undefined 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 (if the expression is true) or 0 (if the expression is false).

Example

Relational operators can for example be used to create indicator variables. The following statement generates a new variable `amountcat`, with value 1 if `amount<=10` and value 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 achieved 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 achieved 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 can be used to change the order of evaluation.

4.3.2 Functions

Functions are a further component of expressions. Any function is indicated by the function name followed by an opening and a closing parenthesis. Inside the parentheses one or more arguments may be specified. The argument(s) of a function may again be expressions, including calls to further functions. Multiple arguments of a function are separated by commas. All functions return missing values when given missing values as arguments or when the result is undefined.

All mathematical function currently available in *BayesX* are referenced in Table 4.1. Statistical functions can be found in Table 4.2.

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 does `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 can be used in expressions.

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`)`	empirical 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 ν is 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().
`poisson(`λ`)`	returns poisson distributed random numbers with parameter λ. If $\lambda \leq 0$, a missing value will be returned.
`uniform()`	uniform pseudo random number function; returns uniformly distributed pseudo-random numbers on the interval $(0, 1)$
`weibull(`α`,`λ`)`	returns weibull distributed random numbers with density $f(x) = \alpha\lambda^\alpha x^{\alpha-1} \exp(-\lambda x^\alpha)$. If $\alpha \leq 0$ and/or $\lambda \leq 0$, a missing value will be returned.

Table 4.2: List of statistical functions

Example

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

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

The command

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

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

4.3.4 Explicit subscribing

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

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

Example

Explicit subscribing combined with the variable `_n` (see Table 4.3) can be used to create lagged values of 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

Method `generate` is used to create a new variable in an existing *dataset object*.

Syntax

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

Method `generate` creates a new variable with name *newvar* (compare section 4.13 for valid variable names) The values of the new variable are specified by *expression*.

Options

not allowed

Example

The following command generates a new variable called `amount2` which contains the squared 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 can be changed with method `replace`, see section 4.9.

If you want to generate an indicator variable `largeamount` for amounts exceeding a certain value, say 3.5, you have to type

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

The variable `largeamount` takes the value 1 if `amount` is larger than 3.5 and 0 otherwise.

4.5 Method infile

Description

Reads data saved in an ASCII file.

Syntax

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

Reads the data stored in *filename*. The variable names can either be specified in *varlist* or be extracted from the first line of the file. To be more specific, if *varlist* is empty it is assumed that the first row of the file contains the variable names separated by blanks or tabs. The observations do not have to be 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 value of the first variable of the the first observation, the second value will be the value of the second variable of the first observation, and so on. An error message will occur if no values can be read for the last observation for some variables.

The period '.' or '`NA`' should be used to indicate missing values.

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

Options

- `missing =` *missingsigns*
  By default the period '.' or '`NA`' indicate missing values. If the missing values are indicated by a different sign, you can specify them in 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 in advance and therefore is unable to allocate enough memory. Therefore new memory has to be allocated whenever a certain amount of memory is exceeded. Specifying option `maxobs` allows to circumvent this problem and to considerably reduce the computing time. When `maxobs` is specified, *BayesX* can allocate enough memory to store at least *integer* observations before new memory must be reallocated. Suppose for example that your data set consists of approximately 100,000 observations. Then specifying `maxobs=105000` allocates enough memory to read in the data set quickly. Note that `maxobs=105000` does not require the data set to have exactly 105,000 observations. It only means that new memory will have to be allocated when the number of observations exceeds the 105,000 observations limit.

Example

Suppose we want to read a data set stored in `c:\data\testdata.raw` containing the 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 the data set, we have to create a new *dataset object* first, say `testdata`. We proceed by reading the data using the `infile` command:

```
> 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 has to be altered to:

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

If furthermore the data set is large with about 100,000 observations. In this 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 be much faster than the command without option `maxobs`.

4.6 Method outfile

Description

Method `outfile` writes data to a file in ASCII format.

Syntax

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

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

Options

- `header`
  Specifying the `header` option causes *BayesX* to write the variable names to 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 and the file specified in *filename* is already existing, an error will be raised. This prevents you from unintentionally overwriting an existing file.

Example

The statement

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

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

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

If you want to store only the two variables `y` and `amount`, you could type

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

This will raise the error message

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

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

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

4.7 Method pctile

Description

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

Syntax

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

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

Options

not allowed

Example

The statement

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

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

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

restricts the analysis to observations with x>0.

4.8 Method rename

Description

`rename` is used to change variable names.

Syntax

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

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

Options

not allowed

4.9 Method replace

Description

`replace` changes values of an existing variable.

Syntax

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

`replace` changes some or all values of the variable *varname* to the values specified in *expression*. If *varname* is not existing, an error will be raised. An optional `if` statement may be used to change the values of the variable only if the boolean expression *expression* 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 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*. The new number of observations has to be greater or equal than the current number to prevent you from unintentionally deleting parts of the data currently in memory. Observations may be eliminated using the `drop` statement, see section 4.2. The values of newly 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 tables.

Syntax

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

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

Options

not allowed

Example

The statement

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

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

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

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

4.13 Variable names

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

4.14 Examples: Working with datasets

This section contains two examples on how to work with *dataset objects*. The first example illustrates some of the methods described in this chapter using the credit scoring data set (see subsection 2.6.2 for a description). The second example demonstrates 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 either 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 be added to the model.

We start by creating the *dataset object* credit and proceed by reading the data using the infile command.

```
> dataset credit
> credit.infile using c:\bayesx\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 demonstrate how to simulate complex regression models. Suppose first that we want to simulate data according to the following Gaussian regression model:

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

First of all, we 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. We assume that the covariates are uniformly distributed between -3 and 3 and therefore enter the commands

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

Finally, we create the response variable by typing

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

Now we could estimate the regression model with the generated data set using one of the regression tools of *BayesX*, see chapter 7 or chapter 8.

Of course, more refined models can also be simulated. We may for example drop the assumption of a constant variance. Suppose the variance is heteroscedastic and growing with order $\log(i)$ where i is the observation index. Such a heteroscedastic model can be simulated by:

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

In this model the standard deviation is given by

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

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

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

To simulate this model we have to specify the linear predictor η_i and the number of replications n_i. We make use of 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 might therefore be preferred.

Chapter 5

map objects

The purpose of *map objects* is to provide functionality for handling and storing geographical maps. In addition, *map objects* serve as auxiliary objects for regression models with spatial effects, where the effect of a location variable is to be included via a spatially correlated prior. In this context, *map objects* store the neighborhood structure of a map and provide functionality to compute weights associated with this neighborhood structure.

The typical approach will be as follows: A *map object* is created, 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 weights, the user is given the choice between a couple of alternatives. Afterwards the *map object* can be passed to the `regress` function of a regression object to estimate regression models with spatial covariates, see chapter 7 and chapter 8, and in particular the subsections about spatial covariates therein.

The *BayesX* R package (see section 11.2) provides some functionality for creating and manipulating boundary information and for storing it in file formats that can be processed by *BayesX*.

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. Currently, two file formats are supported: *boundary files* and *graph files*. A *boundary file* contains information about the boundaries of the different regions of a map in terms of closed polygons, i.e. the boundary of each region is represented by a set of connected straight lines. A detailed description of the structure of boundary files is given below.

A *graph file* stores the nodes and edges of a graph representing the neighborhood structure of a map. In addition, weights associated with the edges of the graph can be specified. In terms of geographical maps, the nodes of the graph correspond to the regions of the map while the edges specify the neighborhood structure. As a consequence, the neighborhood structure of a geographical map is immediately available in a *graph file* while it has to be computed from the polygons in case of a *boundary file* (a task which may be time-consuming). Therefore an advisable strategy is to read a *boundary files* only once to compute the neighborhood structure and to store it as a *graph file* afterwards (using the `outfile` command, see section 5.2). This is particularly the case for geographical maps with a lot of regions. However, using *graph files* also has a disadvantage: Visualization of geographical information is only possible with boundary files since only these contain the information on the boundaries required for visual representation.

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.

Structure of a boundary file

A *boundary file* provides the boundary information of a geographical map in terms of closed polygons. For each region of the map, the boundary files contains a block of lines defining the name of the region, the number of lines the polygon consists of, and the polygons themselves. The first line of such a block contains the region code surrounded by quotation marks and the number of lines the polygon of the region consists of. The region 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 this part of the *boundary file* can be found in Figure 5.1. Note that the first and the last point of a polygon must be identical (see the example above) to obtain a closed polygon.

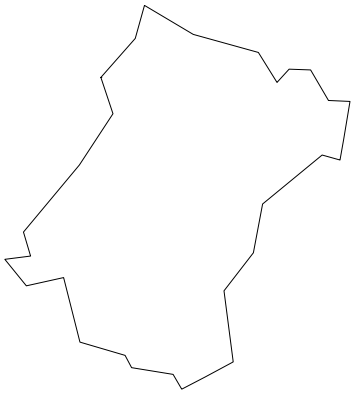


Figure 5.1: Corresponding graph of the section of the boundary file

In some cases a region may be divided into a number of subregions that are not connected. As an example, Figure 5.2 shows a region within Germany that is separated into 8 subregions. In such a case each subregion has to be included separately in the *boundary file* but all subregions should share the same region code. Note that the subregions can be placed anywhere in the *boundary file* and, in particular, do not have to follow each other.

Another special case that requires additional attention is illustrated in Figure 5.3: A region is completely surrounded by another region. In this case an additional line has to be added to the boundary description of the *surrounded* region. This additional line is placed between the first line and the polygons and has to contain the region code of the *surrounding* region. The full 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 of 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 corresponding spatial covariate have to match. Since only numerical variables allowed in *dataset objects*, the names of the regions in the *map object* have to be numbers, although, in principle, there is 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 & Liu (1981, Ch. 3) for a brief introduction to graph theory. A graph is a convenient way to represent the neighborhood structure of a geographical map, where the nodes of the graph correspond to the region codes while the neighborhood structure is represented by the edges of the graph. In some situations it can be useful to define weights associated with the edges of a graph which can be stored in the *graph file* as well.

In *BayesX*, the first line of a *graph file* has to contain the total number of nodes of the graph. Afterwards, each region (or node) is represented by a block of three lines. The first of these lines contains the name of the node (typically the name of the geographical region). The second line states the number of edges for that particular node. The third line contains the corresponding edges of the node, represented by the index of a neighboring node. Note that the index starts with zero, i.e. the first node is represented by the index 0, the second node by the index 1, and so on.

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

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

The first line specifies the total number of nodes, 327 in the present example. 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'.

Graph files also allow to specify weights associated with the edges of the nodes. Since no weights

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

have been explicitly specified in the preceding example, all weights are automatically defined to be equal to one. Nonequal weights can specified in the *graph file* by 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
      ⋮
```

In this case, 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* rather than a *boundary file*.

- `weightdef=adjacency` | `centroid` | `combnd`
  Option `weightdef` allows to request the computation of weights associated with each pair of neighbors. Currently there are three weight specifications available: `weightdef=adjacency`, `weightdef=centroid` and `weightdef=combnd`. For `weightdef=adjacency` all weights are set equal to one. This is the most common choice in spatial statistics and also the default weight method.

  Specifying `weightdef=centroid` results in weights which are inverse 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. In analogy to adjacency weights, the constant c is chosen in such a way that the total sum of weights is equal to the total number of neighbors.

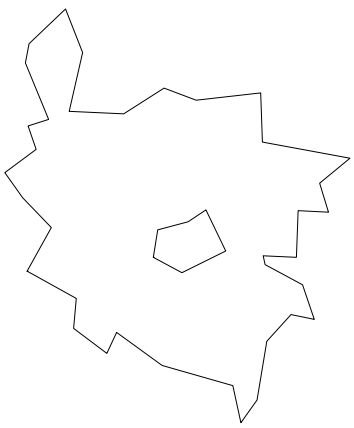


Figure 5.3: Example for a region that is totally surrounded by another region.

The third choice `weightdef=combnd` results in weights proportional to the length of the common boundary of two regions. 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. For *graph files* the option has no effect since the boundary information of regions is missing and the computation of weights is therefore not possible.

5.2 Method outfile

Description

Method `outfile` performs the reverse of the `infile` command, i.e. the current map information 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*. If `graph` is specified as an additional option, the file format will be a *graph file* and a *boundary file* otherwise.

Options

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

5.3 Method reorder

Description

Method `reorder` reorders the regions of a map to obtain an adjacency matrix with minimal envelope. A new map should always be reordered before using it with *bayesreg objects* since MCMC updates for spatial covariates will be much faster with an optimised envelope. *BayesX* uses the reverse Cuthill Mc-Kee algorithm to reorder maps, see George & Liu (1981, p. 58ff).

Syntax

> *objectname*.`reorder`

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

Options

not allowed.

Chapter 6

graph objects

Graph objects are used to visualize data or estimation results obtained with the regression objects in the GUI version of *BayesX*. No graph objects are available in the command line version of *BayesX*. However, the R package accompanying *BayesX* provides similar functionality for visualising data and estimation results as implemented for *graph objects*.

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 resulting plots are either printed on the screen or stored as postscript files 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 to store the map as a postscript file. See the options list below for more details.

Options

The most important option, which therefore is obligatory, is the `map` option. This option specified the name of the *map object* containing the boundary information to be drawn. Additional options for method `drawmap` (in alphabetical order) are:

- `color`

  The `color` option allows to choose between a grey scale and a colored scale. If the keyword `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 will probably be confusing in most cases). This can be achieved 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 (see option `titlesize`). The default is `fontsize=12`.

- `hcl`

  Requests that a color palette from the HCL color space should be used instead of an RGB palette. The HCL colors will be selected diverging from a neutral center (grey) to two different extreme colors (red and green) in contrast to the RGB colors diverging from yellow to red and green. HCL colors are particularly useful for electronic presentations since they are device-independent. The option `hcl` is only meaningful in combination with the option `color`.

- `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 to be drawn. This option is obligatory.

- `nolegend`

  By default a legend is drawn into the graph. Specifying the option `nolegend` will exclude the legend from the graph.

- `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 therefore the number of different colors). The maximum number of colors is 256, which is also the default value.

- `outfile =` *filename*

  If option `outfile` is specified the graph will be stored as a postscript file rather than being printed on the screen. The full path and the filename have to be specified in *filename*. By default, an error will be raised if the specified file is already existing or if the path is invalid. To overwrite an already existing file, option `replace` has to be specified in addition. This prevents you from overwriting your files unintentionally.

- `pcat`

  If you want to visualize posterior probabilities, it is convenient to specify `pcat`. In this case, method `drawmap` expects a variable containing only the values -1, 0 and 1. Of course you can achieve the same result by setting `nrcolors=3`, `lowerlimit=-1` and `upperlimit=1`.

- `replace`

  The `replace` option is only meanigful 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.

- `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"`).

- `titlesize =` *realvalue*

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

- `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.

Example

This example shows how to draw the map of Munich and how to color the subquarters in Munich according to some numerical characteristics. The boundary file of Munich (`munich.bnd`) as well as the data set `rent94means.raw` containing the distribution of the average rents across subquarters are included in the subfolder `examples` of the *BayesX* installation directory. In the following we assume that *BayesX* is installed in the folder `c:\bayesx`. We start by creating a *dataset object* `d` and a *map object* `m` and proceed with reading 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
```

Afterwards we create a *graph object* `g` and draw the map of Munich:

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

Figure 6.1: Map of Munich

The map of Munich appears on the screen in a separate window, compare 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 this 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 printed on the screen.

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

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

displays the distribution of the average rents `R` across subquarters `L`, 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. Specifying option `hcl` in addition yields the same information visualized in HCL colors (see Figure 6.3).

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

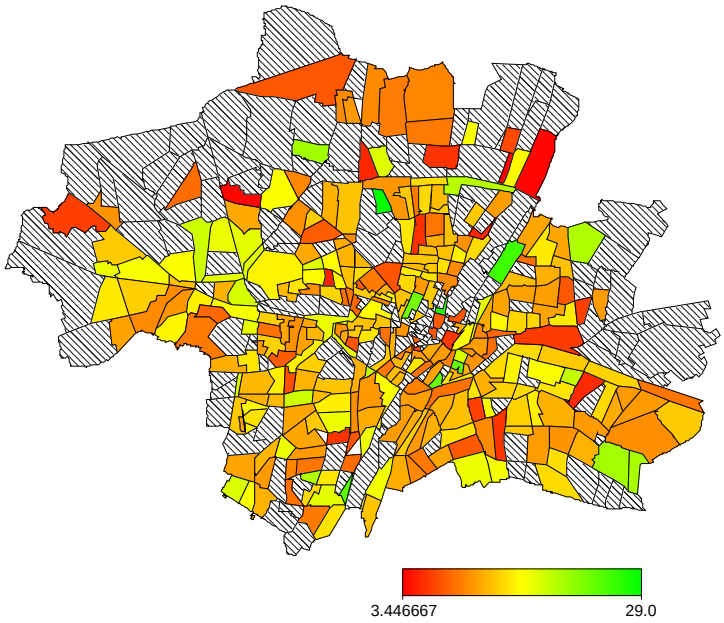


Figure 6.2: Distribution of the average rents per square meter in Munich visualized in RGB colors.

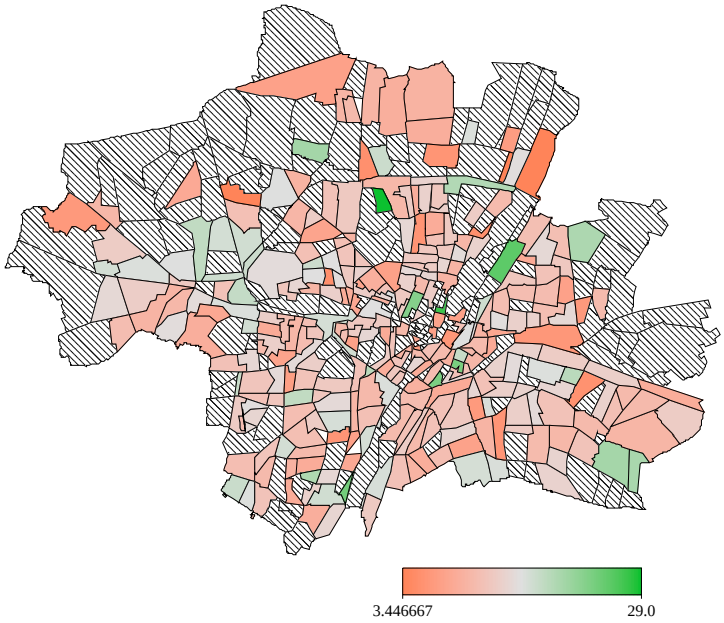


Figure 6.3: Distribution of the average rents per square meter in Munich visualized in HCL colors.

6.2 Method plot

Description

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

Syntax

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

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

Options

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

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

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

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

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

  ```
  connect=15
  ```

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

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

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

- `fontsize =` *integer*

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

- `height =` *integer*

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

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

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

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

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

```
linecolor = Bgr
```

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

- `linewidth =` *integer*

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

- `outfile =` *filename*

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

- `pointsize =` *integer*

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

- `replace`

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

- `title =` *characterstring*

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

- `titlesize =` *realvalue*

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

- `width =` *integer*

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

- `xlab =` *characterstring*

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

- `xlimbottom =` *realvalue*

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

- `xlimtop =` *realvalue*

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

- `xstart =` *realvalue*

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

- `xstep =` *realvalue*

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

- `ylab =` *characterstring*

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

- `ylimbottom =` *realvalue*

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

- `ylimtop =` *realvalue*

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

- `ystart =` *realvalue*

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

- `ystep =` *realvalue*

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

- Further options for representing dates

  In the following we describe options that may be useful if the variable on 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 might prefer that the x-axis is labelled in terms of dates rather than in the original coding (from 1 to 19). To achieve this, *BayesX* provides 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, respectively.

Example

We use the Munich rent data set `rent94.raw` to demonstrate the usage of method `plot`. The data set is included in the subfolder `examples` of the *BayesX* installation directory. In the following we assume that *BayesX* has been installed to the folder `c:\bayesx`. We start by reading the data by typing:

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

Then we 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 6.4 appears on the screen. The problem is that the points are connected by straight lines although 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. Typing

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

yields the second option mentioned above. The corresponding graphs are shown in Figure 6.5 and Figure 6.6, 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 6.7. 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
```

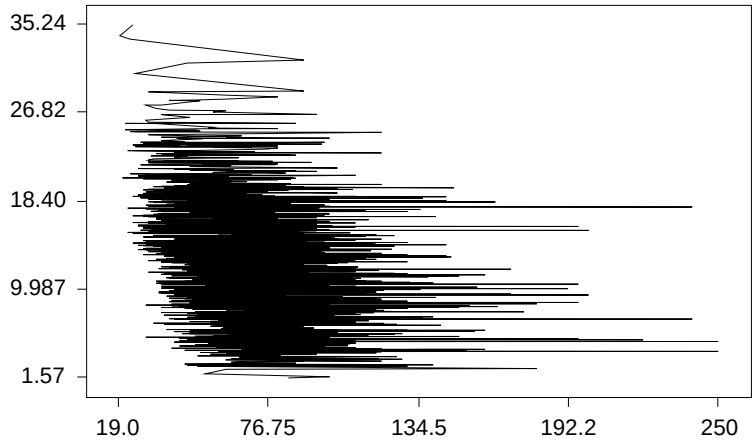


Figure 6.4: Scatterplot between floor space and rent per square meters (first try).

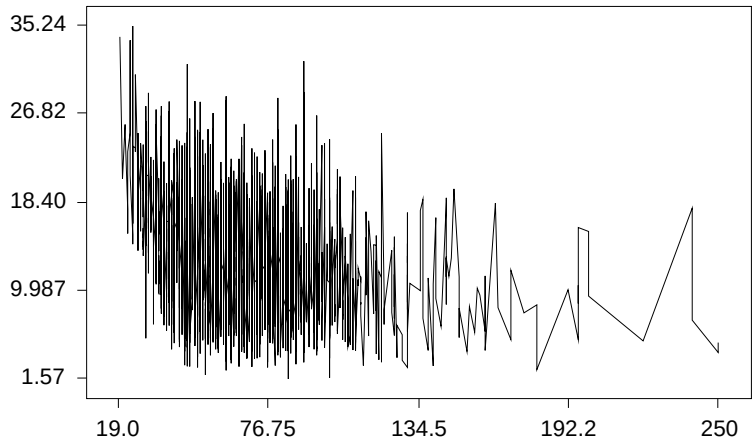


Figure 6.5: Scatterplot between floor space and rent per square meters (second try).

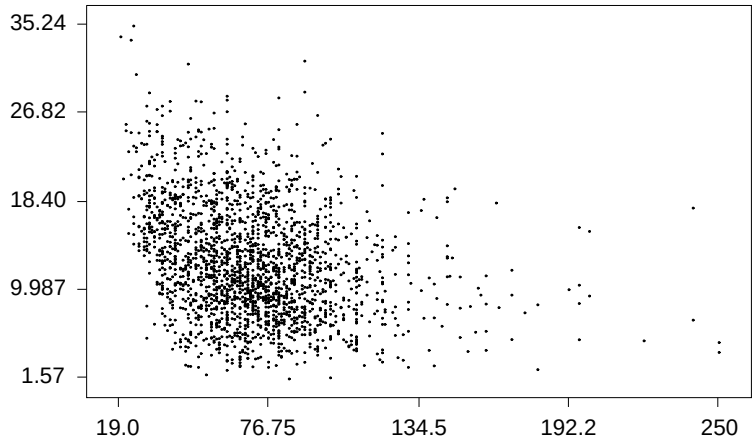


Figure 6.6: Scatterplot between floor space and rent per square meters (third try).

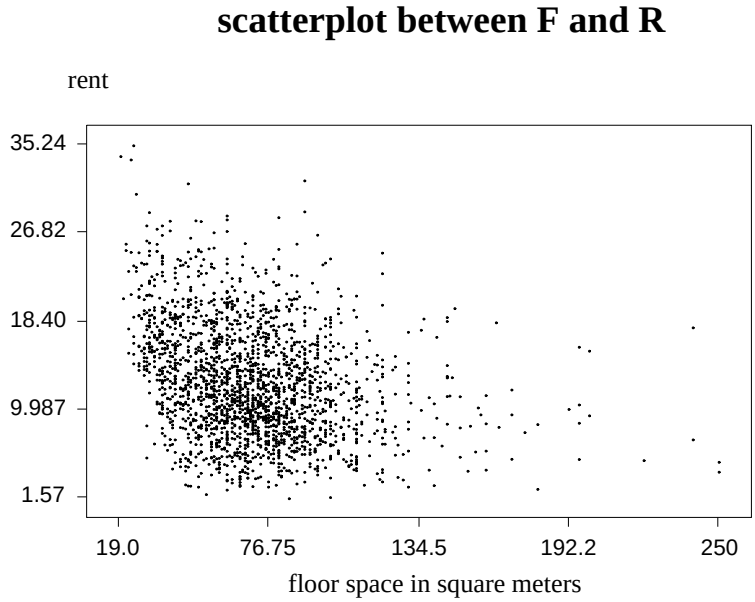


Figure 6.7: Scatterplot between floor space and rent per square meters (final try).

6.3 Method plotautocor

Description

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

Syntax

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

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

Options

- `mean`

  If option `mean` is specified, only minimum, mean and maximum autocorrelations are plotted for each lag number and model term. This typically leads 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 instead of being printed on the screen. The path and the filename have to be specified in *filename*. An error will be raised if the specified file is already existing and the `replace` option has not been specified.

- `replace`

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

6.4 Method plotsample

Description

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

Syntax

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

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

Options

- `outfile =` *filename*

  If option `outfile` is specified, the graph will be stored as a postscript file instead of being printed on the screen. The full path and the filename have to be specified in *filename*. An error will be raised if the specified file is already existing and the `replace` option has not been 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.

Chapter 7

bayesreg objects

bayesreg objects are used to fit (multivariate) exponential family, hazard rate or multi-state models with *structured additive predictor* subsumed in the class of *structured additive regression (STAR)* models, see Fahrmeir, Kneib & Lang (2004). Inference is based on fully Bayesian approach implemented via Markov Chain Monte Carlo (MCMC) simulation techniques. The methodology manual provides brief introduction to structured additive regression and MCMC-based inference. More details can be found in Fahrmeir & Lang (2001), Fahrmeir & Lang (2001), Lang & Brezger (2004), Brezger & Lang (2006), Fahrmeir & Osuna (2006) and Hennerfeind, Brezger & Fahrmeir (2006). Introductions to (Bayesian) generalized linear models as well as to semi- and nonparametric models are given in Fahrmeir et al. (2013). The paper of Chib & Greenberg (1995), the monograph *Markov Chain Monte Carlo in Practice* edited by Gilks, Richardson & Spiegelhalter (1996) and the introductory article by Green (2001) give a thorough overview over MCMC simulation techniques.

First steps with *bayesreg objects* can be done with the tutorial like example in chapter 1 of the tutorials manual which provides a self-contained demonstrating example.

7.1 Method regress

7.1.1 Description

Method `regress` estimates a structured additive regression model.

7.1.2 Syntax

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

Method `regress` estimates the regression model specified in *model* using the data specified in *dataset*. *dataset* has to be the name of a *dataset object* created before. The details of correct models are covered in subsubsection 7.1.2.2. The distribution of the response variable can be either Gaussian, gamma, binomial, multinomial or Poisson. In addition, *BayesX* supports continuous time survival and multi-state models, see also Table 7.6 for a more detailled overview. The response distribution is specified using option `family`, see subsubsection 7.1.2.4 below and the options list in subsection 7.1.3. The default value is `family=binomial` with a logit link. An `if` statement may be specified to analyze only parts of the data set, i.e. the observations where *expression* is true.

7.1.2.1 Optional weight variable

An optional weight variable *weightvar* may be specified to estimate weighted regression models. For Gaussian responses *BayesX* assumes that $y_r|\eta_r, \sigma^2 \sim N(\eta, \sigma^2/weightvar_r)$. Thus, for grouped Gaussian responses the weights must be the number of observations in the groups if the y_r's are the average of individual responses. If the y_r's are the sum of responses in every group, the weights have to 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 itself 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 have to 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. Similarly, for hazard regression and multi-state models, weighted regression is not implemented yet.

7.1.2.2 Syntax of possible model terms

The general syntax of models is:

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

where *depvar* specifies the dependent variable in the model and $term_1$,...,$term_r$ define the specific form of covariate effects on the dependent variable. The different terms have to be separated by '+' signs. A constant intercept is automatically included in the models and does not have to be requested by the user. This section reviews all possible model terms that are currently supported by *bayesreg objects* and provides some specific examples. Note that all described terms may be combined in arbitrary order. An overview about the capabilities of *bayesreg objects* is given in

Table 7.1. Table 7.2 shows how interactions between covariates are specified. Full details about all available options are given in subsubsection 7.1.2.3.

Throughout this section `Y` will denote the dependent variable.

Offset

Description: Adds an offset term to the predictor.

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

Syntax: `offs(offset)`

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 specified a model with q fixed (linear) effects:

```
Y = W1 + W2 + ··· + Wq
```

Shrinkage of fixed effects

Description: Defines a shrinkage-prior for the corresponding parameters γ_j, $j = 1, \ldots, q$, $q \geq 1$ of the linear effects `X1,...Xq`. There are three priors possible: ridge-, lasso- and Normal Mixture of inverse Gamma (NMIG)-prior.

Predictor: $\eta = \cdots + \gamma_1 X1 + \cdots + \gamma_q Xq + \cdots$

Syntax:

- Ridge-prior: `X1(ridge`[, *options*]`)`
- Lasso-prior: `X1(lasso`[, *options*]`)`
- NMIG-prior: `X1(nigmix`[, *options*]`)`

Example: The following model statement can be used to estimate a model with q lasso-penalized linear effects

```
Y = X1(lasso)+...+ Xq(lasso)
```

By default, the starting value of the shrinkage parameter in the Markov chain is set to 1 and the shrinkage parameter is estimated by the data. It is also possible to fix the shrinkage parameter through the iterations in order to use a prespecified amount for shrinkage. To do

so the the option `shrinkagefix` have to be set in the corresponding terms and this results in fixing the shrinkage parameter at the starting value assigned in the option `shrinkage`.

The following model term defines a lasso-penalty with shrinkage parameter fixed at the value 1.5:

```
Y = X1(lasso)+...+ Xq(lasso,shrinkage=1.5,shrinkagefix)
```

Full details about all possible options for shrinkage-effects are given in 7.1.2.3.

Important Remark: Except the option `tau2` for the variances of lasso or ridge (and resp. the options `I` and `t2` for nigmix), all the other possible options used in the shrinkage-methods are those which are specified in the first term of the corresponding penalty, e.g.

```
Y = X2(lasso,shrinkagepar=2,shrinkagefix)+ X1(lasso,shrinkagepar=1.5)
```

uses the options of `X2`. If the option `adaptive` is specified the options from each term are used.

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 that `X1` is a continuous covariate with possibly nonlinear effect. The following model statement defines a second order random walk prior for f_1:

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

The term `X1(rw2,a=0.001,b=0.001)` indicates, that the effect of `X1` should be included nonparametrically using a second order random walk prior. A first order random walk is obtained by modifying the first argument in `X1(rw2,a=0.001,b=0.001)` from `rw2` to `rw1` yielding `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 of the random walk. Besides the options `a` and `b`, some more options are available, see subsubsection 7.1.2.3 for details.

P-spline with first or second order random walk penalty

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

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

Syntax:

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

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

Example:

A P-spline with second order random walk penalty is obtained by:

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

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

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

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

Seasonal effect of a time scale

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 by

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

where 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 to incorporate spatial covariates with geographical information stored in the *map object* specified in option `map`.

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

Syntax:

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

Example:

For the specification of a Markov random field prior, `map` is an obligatory argument that represents the name of a *map object* (see chapter 5) containing 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 prior to estimation of a spatial effect to obtain a band matrix like precision matrix. This can be achieved using method `reorder` of *map objects*, see section 5.3 for details.

Two-dimensional P-spline with first order random walk penalty

Description:

Defines a two-dimensional P-spline for the spatial covariate `region` with a two-dimensional first order random walk penalty for the parameters of the spline. Estimation is based on the coordinates of the centroids of the regions. The centroids are computed using the geographical information stored in the *map object* specified in the option `map`.

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

Syntax:

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

Example:

For the specification of a two-dimensional P-spline (*geospline*) `map` is an obligatory argument indicating 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 two-dimensional 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:

Gaussian i.i.d. random effects allow 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 defined by

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

The inclusion of more than one random effect term in the model is possible, leading to 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 hazard regression or multi-state 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` is a continuous covariate. The following continuous time survival 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)
```

Similarly, baseline effects on the transition intensities can be specified in multi-state models.

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`. Therefore covariate `X2` is called the effect modifier. The smoothness prior for $f(X2)$ 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:

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`. The smoothness prior for $f(X2)$ 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:

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`. A seasonal prior is assumed for the effect of `time`.

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 a binary variable `X1`. In this case we can include an additional seasonal effects for observations with `X1`=1 by

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

Time-varying effects in hazard regression or multi-state 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` is assumed to have a 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 and `delta` is the vector of corresponding indicators of non-censoring. The following Cox model with hazard rate

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

is estimated by the model statement

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

Similarly, time-varying effects on the transition intensities can be specified in multi-state models.

Varying coefficients with spatial covariates as effect modifiers

Markov random field

Description:

Defines a varying coefficients 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}(`region`). The geographical information is assumed to be stored in the *map object* specified in the option `map`.

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

Syntax:

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

Example:

The statement

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

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

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

Unit- or cluster-specific unstructured effect

Description:

Defines a varying coefficient term where the effect of `X1` varies over the range of the group indicator `grvar`. Models of this type are usually referred to as models with random slopes. A Gaussian i.i.d. random effect with respect to grouping variable `grvar` is assumed for $f(grvar)$. A main effect $\gamma X1$ is additionally estimated using a diffuse prior for γ. Therefore 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 & Carlin (1996). 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 included.

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

Syntax:

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

Example:

A random slope with additional 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` has to be specified:

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

Surface estimators

Two-dimensional P-spline with first order random walk penalty

Description:

Defines a two-dimensional P-spline with a two-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 two-dimensional cubic P-spline with first order random walk penalty.

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

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

7.1.2.3 Description of additional options for terms of bayesreg objects

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

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

Note that all options may be specified in arbitrary order. Table 7.3 and Table 7.4 provide explanations and the default values of all possible options. All reasonable combinations of model terms and options can be found in Table 7.5.

Type	Syntax example	Description
Offset	`offs(offset)`	Variable `offs` is an offset term.
Linear effect	`W1`	Linear effect of `W1`.
Ridge effect	`X1(ridge)`	Linear effect of `X1` with ridge-penalty.
Lasso effect	`X1(lasso)`	Linear effect of `X1` with lasso-penalty.
NMIG effect	`X1(nigmix)`	Linear effect of `X1` with NMIG-penalty.
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)`	Time-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 are 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 obtained from the *map object* `m`.
Random intercept	`grvar(random)`	I.i.d. Gaussian (random) effect of the group indicator `grvar`, e.g. `grvar` may be an individual indicator when analyzing longitudinal data.
Baseline in Cox or multi-state models	`time(baseline)`	Nonlinear shape of the baseline effect $\lambda_0(time)$ of a Cox model. $\log(\lambda_0(time))$ is modelled by a P-spline with second order penalty.

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

Type of interaction	Syntax example	Description

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

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 or multi-state models	`X1*time(baseline)`	Nonlinear, time-varying effect of `X1`.

Table 7.2: Possible interaction terms for bayesreg objects.

Option	Description	Default
a,b	1. The options a and b specify the hyperparameters of the inverse Gamma prior for the variance τ^2. 2. The options a and b specify the hyperparameters of the inverse Gamma prior for the shrinkage parameter λ if the lasso- or the ridge-prior is used (**). 3. The options a and b specify the hyperparameters the inverse Gamma prior for the variance parameters t_j^2 if the NMIG-penalty is used.	a=0.001, b=0.001 a=0.001, b=0.001 a=5, b=50
adaptive	Specifies adaptive versions of the shrinkge priors.	-
aw,bw	The options aw and bw specify the hyperparameters of the Beta prior for the parameter ω if the NMIG-penalty is used (**).	a=0.001, b=0.001
contourprob	Forces the computation of contour probabilities for P-splines as described in Brezger & Lang (2008). For instance, contourprob=4 specifies that contour probabilities for difference orders zero to four are computed. Note that the global simple option approx may additionally be specified. In this case the computation of contour probabilities is based on stochastic approximations for quantiles as described in Tierney (1983).	contourprob=4
degree	Specifies the degree of B-spline basis functions.	degree=3
derivative	If specified, first order derivatives of the function estimate are computed (for P-splines only).	-
gridsize	The option gridsize can be used to restrict the number of points (on 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
I	Provides a starting value of for the indicator variable I_j of the corresponding effect if the NMIG-penalty is used. Values have to be 0 or 1 and have to be set in each shrinkage term.	I=1
lambda	Provides a starting value for the smoothing parameter λ.	lambda=0.1
min,max	The options min and max define the minimum and maximum block sizes of block move updates. In every iteration, *BayesX* randomly chooses the block size within this range. If omitted, the minimum and maximum block sizes are automatically determined during the burnin period such that the average acceptance rate is between 30% and 70%. The specification of minimum and maximum block sizes is only meaningful in combination with conditional prior proposals and therefore has no effect for Gaussian responses, categorical probit models, or if proposal=iwls or proposal=iwlsmode is specified.	automatic determination
monotone	Defines monotonicity constraints for P-splines. Specifying monotone=increasing yields increasing nonlinear functions and monotone=decreasing yields decreasing functions.	monotone = unrestricted
nofixed	The option nofixed suppresses the estimation of the main effect $\gamma X1$ for random slopes.	-
nrknots	Specifies the number of inner knots for a P-spline term.	nrknots=20

Table 7.3: Optional arguments for bayesreg object terms in alphabetical order (1).

Option	Description	Default
`w`	Specifies the starting value of the complexity parameter ω if the NMIG-penalty is used. Values have to be in the interval (0,1) (*).	`w=0.5`
`wfix`	Specifies if the complexity parameter ω should take the fixed value given in option `w` if the NMIG-penalty is used (**).	-
`period`	Period of the seasonal effect. The default is `period=12` which corresponds to monthly data.	`period=12`
`proposal`	Specifies the type of proposal. `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`
`shrinkagefix`	Specifies if the shrinkage parameter λ should take the fixed value given in option `shrinkage` (*).	-
`shrinkage`	Provides a starting value of for the shrinkage parameter λ if the lasso- or ridge -penalty is used. Values have to be positive (**).	`shrinkage=1`
`t2`	Provides a starting value of for the variance parameter t_j^2 of the corresponding effect if the NMIG-penalty is used. Values have to be positive and have to be set in each shrinkage term.	`t2=11`
`tau2`	Provides a starting value of for the variance parameter τ_j^2 of the corresponding effect if the lasso- or the ridge-prior is used. Values have to be positive and have to be set in each shrinkage term.	`tau2=0.1`
`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`
`v0`, `v1`	Hyperparameters that determine the modes of the marginal variances if the NMIG-penalty is used. Values have to be nonnegative (*).	`v0=0.005`, `v1=1`
	(*) these options are specified in the first covariate occurring in term of the corresponding penalty. (**) these options are specified in the first covariate occurring in term of the corresponding penalty in the nonadaptive case and in each covariate in the adaptive case.	

Table 7.4: Optional arguments for bayesreg object terms in alphabetical order (2).

	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	$\times$	$\times$	$\times$	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 7.5: Terms and options for bayesreg objects.

7.1.2.4 Specifying the response distribution

Supported univariate distributions are Gaussian, binomial (with logit or probit link), Poisson and gamma. 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. Continuous survival times as well as multi-state models can be analysed based on semiparametric models with Cox-type hazard rates, see subsections 7.1.2.5 and 7.1.2.6. An overview over the supported models is given in Table 7.6. The distribution of the response is specified by adding the additional option `family` to the (global) options list of the regression call. For instance, `family=gaussian` defines the response to be Gaussian distributed. In some cases, one or more additional options associated with the specified response distribution can 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 $y|\eta, \sigma^2 \sim N(\eta, \sigma^2 C^{-1})$, where $C = diag(weightvar_1, \ldots, weightvar_n)$ is a known weight matrix. Gaussian regression models are obtained by adding

```
family=gaussian
```

to the options list.

An optional weight variable *weightvar* can be specified to estimate weighted regression models, see subsubsection 10.1.2.1 for details. For grouped Gaussian responses, the weights represent the number of observations in the groups if the y_i's are the average of individual responses. If the y_is are the sum of responses in every group, the weights are given by 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 no weight variable is specified, *BayesX* assumes $weightvar_i = 1$, $i = 1, \ldots, n$.

For Gaussian responses, the additional parameter σ^2 for the error variancehas to be estimated. An additional inverse gamma prior with hyperparameters `aresp` and `bresp` is defined for σ^2. The default for the hyperparameters is `aresp=0.001` and `bresp=0.001`. The default values may be changed using the `aresp` and `bresp` option. For instance, adding

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

to the options list set both `a` and `b` equal 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

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

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

A second parameterization is typically employed for hyperparameters `a` and `b` of priors for variance parameters in the context of Bayesian hierarchical models. In this case, the density is given by

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

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

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

```
family=gamma
```

to the options list. An optional weight variable *weightvar* can be specified to estimate weighted regression models, see subsubsection 10.1.2.1 for details.

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

Updating of the scale parameter ν is implemented via 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 $Var(\nu^{prop})$ can be considered a tuning parameter. It is specified by the additional option `gammavar`. For example, when 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 deterministic scale parameter. The scale parameter is defined to be fixed 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 to be fixed at the value $\nu = 1$.

Binomial logit and probit models

A binomial logit model is requested by the option

```
family=binomial
```

while a probit model is obtained by adding

```
family=binomialprobit
```

to the option list.

For logit models a weight variable may be specified in addition, see subsubsection 10.1.2.1 for details. *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, while a multinomial probit model is requested by

`family=multinomialprobit`

A further option (`reference`) can be added to the options list to define the reference category. If the response variable has three categories 1, 2 and 3, the reference category can be set to 2, by adding

`reference=2`

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

Cumulative threshold models

So far, *bayesreg objects* support only cumulative probit models. Such a model is specified by adding

`family=cumprobit`

to the options list. The reference category will always be the largest value of the response and can not be changed by the user.

An important problem with Bayesian cumulative threshold models is the mixing and convergence of MCMC samples for the threshold parameters. Usually the mixing is relatively poor implying the necessity for 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 typically quite satisfactory. A description of this reparameterization can be found in Fahrmeir & Lang (2001) or in Chen & Dey (2000). However, parameter estimates are given in the original parameterization. To estimate three categorical response models without reparameterization the additional option `notransform` can be added to the options list (not recommended).

Poisson regression

A Poisson regression model is specified by adding

`family=poisson`

to the options list.

A weight variable may be specified in addition, see subsubsection 10.1.2.1 for details. 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.

7.1.2.5 Continuous time survival analysis

BayesX offers two alternatives of estimating continuous time survivals models with semiparametric predictor η, both of which are described in subsection 7.2 of the methodology manual. The first alternative is to assume that all time-dependent values are piecewise constant, leading to the so called *piecewise exponential model* (p.e.m.). The second alternative is to estimate the log-baseline effect $\log(\lambda_0(t)) = f_0(t)$ based on a P-spline with second order random walk penalty.

Piecewise exponential model (p.e.m.)

In subsection 7.2 of the methodology manual we demonstrated how continuous time survival data has to be manipulated to transform it to a Poisson for model estimation. Suppose that the following modified data set is available

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 7.2 of the methodology manual. Let `x1` be a covariate with linear effect and `x2` a continuous covariate with nonlinear effect. Then the correct syntax for estimating a p.e.m. with a *bayesreg object* named `b` would be 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 an additional covariate `X` may be estimated 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 a continuous time survival model with a P-spline prior with second order random walk penalty for the baseline effect,

```
family=cox
```

has to be specified in the options list. The number of knots and degree of the P-spline prior for $f_0(t)$ can be specified as additional options for the baseline term. Note that it is obligatory 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, the additional 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 continuous time survival model with a quadratic P-spline prior with 15 knots for the log-baseline would be estimated as follows:

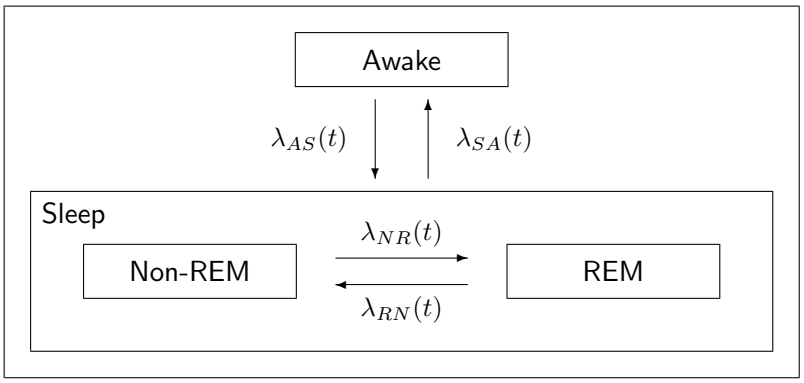


Figure 7.1: Schematic representation of sleep stages and transitions of interest.

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

Again a time-varying effect of a covariate `X` can be estimated by simply adding the term

`X*time(baseline)`

to the model statement.

7.1.2.6 Continuous time multi-state models

Multi-state models describe the temporal development of discrete phenomena in continuous time based on transition intensities for each of the observable transition types. Consider for example a multi-state model for human sleep as depicted in Figure 7.1 and that the transition intensities for the four possible transitions are specified as

$$\begin{aligned}
\lambda_{AS,i}(t) &= \exp\left[g_0^{(AS)}(t) + b_i^{(AS)}\right], \\
\lambda_{SA,i}(t) &= \exp\left[g_0^{(SA)}(t) + b_i^{(SA)}\right], \\
\lambda_{NR,i}(t) &= \exp\left[g_0^{(NR)}(t) + c_i(t)g_1^{(NR)}(t) + b_i^{(NR)}\right] \\
\lambda_{RN,i}(t) &= \exp\left[g_0^{(RN)}(t) + c_i(t)g_1^{(RN)}(t) + b_i^{(RN)}\right]
\end{aligned}$$

Each of the transitions is parameterised in terms of a baseline effect $g_0^{(h)}(t)$ and a transition specific frailty term (random effect) $b_i^{(h)}$. In addition, time-varying effects $g_1^{(h)}(t)$ of binary indicators $c_i(t)$ for a high blood level of cortisol are introduced for the transitions between REM and Non-REM.

The corresponding data set should be arranged as follows:

id	st	beg	end	tas	tsa	trn	tnr	cort	corthigh
1	2	0	1	0	1	0	0	52.6	0
1	1	1	5	1	0	0	0	52.6	0
1	2	5	8	0	1	0	0	52.6	0
1	1	8	10	1	0	0	0	52.6	0
1	2	10	36	0	0	0	0	52.6	0
1	2	36	76	0	0	0	0	46.9	0
1	2	76	108	0	0	0	1	47.5	0
1	3	108	109	0	0	1	0	47.5	0

```
1   2   109    110  0   0   0   1   47.5  0
1   3   110    111  0   0   1   0   47.5  0
1   2   111    115  0   0   0   1   47.5  0
1   3   115    116  0   0   0   0   47.5  0
1   3   116    126  0   0   1   0   37.4  0
.   .    .      .   .   .   .   .     .   .
.   .    .      .   .   .   .   .     .   .
.   .    .      .   .   .   .   .     .   .
2   2   0      12    0   1   0   0   22.5  0
2   1   12     15    1   0   0   0   22.5  0
2   1   15     28    0   1   0   0   88.6  1
.   .    .      .   .   .   .   .     .   .
.   .    .      .   .   .   .   .     .   .
.   .    .      .   .   .   .   .     .   .
```

Each path observed for the multi-state model is transformed into several lines in the data set, where `id` identifies the original paths. In the above example, parts of the first two observations are displayed. Each line of the data set represents a time interval identified by the variables `beg` and `end`. Variable `st` indicates the current state of the process. Note that the states have to be numbered consecutively from 1 to H. Since we are considering continuous time scales, an observation should start at $t = 0$ (unless the observation is left truncated) and the variables `beg` and `end` should be generated so that within each observation process `beg` equals the value of `end` in the previous row (unless observations are fragmentary only).

The variables `tas`, `tsa`, `trn` and `tnr` are binary indicators for the four transitions sleep $\rightarrow$ awake (`tsa`), awake $\rightarrow$ sleep (`tas`), Non-REM $\rightarrow$ REM (`tnr`) and REM $\rightarrow$ Non-REM (`trn`). Such an indicator equals one if the corresponding transition is observed at the end of the interval and zero otherwise. Note that there are lines in the data set, where none of the transitions is observed. These correspond to intervals where the value of the time-varying covariate `cort` (cortisol-level) changes. The variable `corthigh` is a dichotomized version of `cort` which indicates a high level of cortisol (`cort`>60).

The model specified above is estimated by entering the following command

```
> bayesreg msm
> msm.mregress tas = end(baseline) + id(random):
               tsa = end(baseline) + id(random):
               trn = end(baseline) + corthigh*end(baseline) + id(random):
               tnr = end(baseline) + corthigh*end(baseline) + id(random),
 family=multistate begin=beg state=st iterations=30000 burnin=10000
 step=20 using sleep
```

Note that a separate model equation has to specified for each transition with the binary transition indicator as response. Instead of method `regress`, method `mregress` has to be called since multiple model equations are combined. The right and the left boundary of the time intervals have to specified as covariate for the the baseline effect and as global option `begin`, respectively. Similarly, the state variable has to specified via the global option `state`.

More details about Bayesian semiparametric multi-state models including a detailed description of the human sleep data application can be found in Kneib & Hennerfeind (2006).

7.1.3 Options

Options for controlling MCMC simulations

Options for controlling MCMC simulations are listed in alphabetical order.

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

- `iterations =` *integer*
  Changes the number of MCMC iterations to *integer*, where *integer* must be a positive integer number. The number of iterations must be larger than the number of burn-in 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`

- `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 models with the log-link. For some distributions (e.g. multinomial) additional options may be specified to control MCMC inference. A detailed description on how to specify the distribution of the response is given in subsubsection 7.1.2.4. Table 7.6 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 either `family=multinomial` or `family=multinomialprobit` 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.

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=cox`	continuous-time survival data		`begin`
`family=multistate`	continuous-time multi-state data		`begin`, `state`

Table 7.6: Summary of supported response distributions.

Further options

Options are listed in alphabetical order:

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

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

`level1=99`

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

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

  `level2=70`

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

- `predict`
  Option `predict` may be specified to compute samples of the deviance D, the effective number of parameters p_D and the deviance information criterion 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 and $\theta = \mu$ for 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 criterion is defined as

$$DIC = \overline{D(\theta)} + p_D = D(\bar{\theta}) + 2p_D$$

  . *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. The individual saturated deviance D_i^{sat} can be used to compute deviance residuals. The deviance residuals are given by $r_i = sign(y_i - \mu_i)\sqrt{D_i^{sat}}$. The leverage statistics p_{D_i} is defined as the contribution of the ith observation to p_D. More details about the quantities discussed above can be found in Spiegelhalter et al. (2002). To clarify the computation of D, p_D, DIC etc. Table 7.7 provides formulas of the p.d.f. and the (unstandardized) deviance D for the different response distributions provided in *BayesX*.

7.1.4 Estimation output

The way the estimation output is presented depends on the estimated model. Estimation results of fixed effects are displayed in a tabular form in the *output window* and/or in a log file (if created before). Shown will be the posterior mean, the standard deviation, the 2.5% and the 97.5% quantiles. Other quantiles may be obtained by specifying the `level1` and/or `level2` option, see subsection 7.1.3 for details. Additionally a file is created where estimation results for fixed effects are replicated. The name of the file is given in the *output window* and/or in a log file. Estimation effects of nonlinear effects of continuous and spatial covariates as well as unstructured random

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

Table 7.7: 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.

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

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

8 -2.291905 -0.785899 -0.962212 -0.895262 -0.783646 -0.674511 -0.609532 -1 -1
9 -2.178096 -0.876173 -1.0428 -0.982029 -0.877516 -0.768126 -0.708452 -1 -1

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

The estimated nonlinear effects can be visualized by using either the graphics capabilities of *BayesX* or the *BayesX* R package, see section 11.1 and section 11.2, respectively. Of course, any other (statistics) software package with plotting facilities may be used as well.

7.1.5 Examples

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

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

Fixed effects

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

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

This model is estimated by typing:

```
> b.regress y = x1 + x2 + x3, iterations=12000 burnin=2000
  step=10 family=binomial 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 step=10 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 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 step=10 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. Accordingly, a multinomial probit model is estimated by typing

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

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

7.2 Method autocor

Description

This method is a post estimation command, i.e. its usage is meaningful only if method `regress` has been applied before. Method `autocor` computes the autocorrelation functions of all sampled (and stored) parameters. The computed functions will be written to an external file whose name and storing path is printed in the *output window* and/or an additional log

file. The computed autocorrelations can be visualized by using either method `plotautocor` of *graph objects* or the R function `plotautocor`.

Syntax

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

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

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

The computed autocorrelations can be visualized by using either method `plotautocor` of *graph objects* or the R function `plotautocor`. However, since the structure of the file is very simple, the visualization of the functions can be done with every software package which has graphics capabilities.

Options

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

Examples

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

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

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

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

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

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

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

The computed autocorrelation functions may now be visualized using the R 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 autocorrelation functions can be plotted directly in *BayesX* using method `plotautocor` of *graph objects*. 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*. Here autocorrelations are computed and plotted in one step.

7.3 Method getsample

Description

This method is a post estimation command, that 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* or by using the R 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 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
```

Alternatively we could use R function `plotsample` for visualizing sampling paths. We type, for instance

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

7.4 Global options

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

The syntax for specifying global options is

> *objectname.optionname* = *newvalue*

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

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

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

  <INSTALLDIRECTORY>`\output`.

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

<INSTALLDIRECTORY>`\output\b_nonpX.res`

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

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

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

`c:\data\res_nonpX.res`

- `iterationsprint =` *integer*

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

7.5 Visualizing estimation results

Visualization of estimation results is described in chapter 11.

7.6 Examples

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

7.6.1 Binary data: credit scoring

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

7.6.1.1 Reading the data into BayesX

In order to analyze the 'credit scoring' data set, we first have to load the data set into *BayesX*. For the rest of this section we assume that *BayesX* is installed in the directory `c:\bayes`. In this

case, the 'credit scoring' data set can be found in `c:\bayes\examples` under the name `credit.raw`. With the following two commands (entered in the *command window*) we first create a *dataset object* `credit` and afterwards load the data set into *BayesX* using the `infile` command:

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

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

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

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

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

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

7.6.1.2 Creating a bayesreg object

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

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

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

7.6.1.3 Probit models

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

We first estimate a model with fixed effects only:

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

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

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

```
SIMULATION TERMINATED
```

```
SIMULATION RUN TIME: 13 seconds


ESTIMATION RESULTS:

  Predicted values:

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

  Estimation results for the Deviance:

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

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

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

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

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

  Estimation results for the DIC:

  DIC based on the unstandardized Deviance

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

  DIC based on the saturated Deviance

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



  FixedEffects1
```

```
Acceptance rate:    100 %


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

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

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

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

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

```
f_duration_pspline


Acceptance rate:    100 %

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

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

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


f_duration_pspline_variance


Acceptance rate:    100 %

Estimation results for the variance component:

Mean:             0.00787338
Std. dev.:        0.0100301
2.5% Quantile:    0.00128376
10% Quantile:     0.00190016
50% Quantile:     0.00487717
90% Quantile:     0.0157856
97.5% Quantile:   0.0315107
```

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


f_amount_pspline


Acceptance rate:    100 %

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

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

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


f_amount_pspline_variance


Acceptance rate:    100 %

Estimation results for the variance component:

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

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

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

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

This produces the graphs (stored in postscript files) shown in Figure 7.2.

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

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

and obtain the improved graphs shown in Figure 7.3. 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 typing:

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

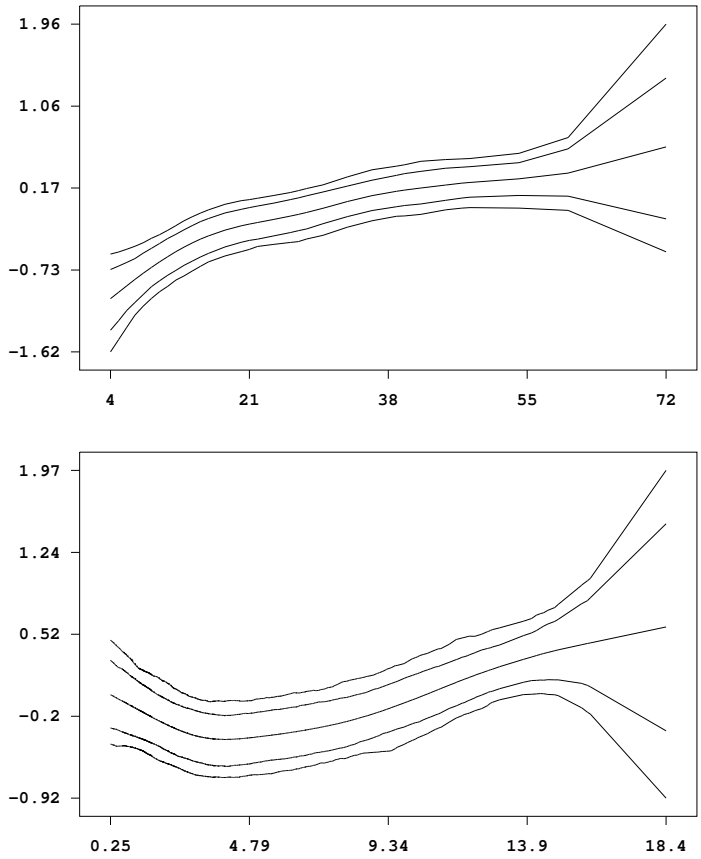


Figure 7.2: Estimated effects of `duration` *and* `amount` *of credit. Shown is the posterior mean within 80% and 95% credible regions.*

We obtain the file `c:\results\credit_autocor.ps` containing 9 pages of autocorrelation functions for all parameters in the model. The first page of this file is shown in Figure 7.4. We see that autocorrelations die off very quickly.

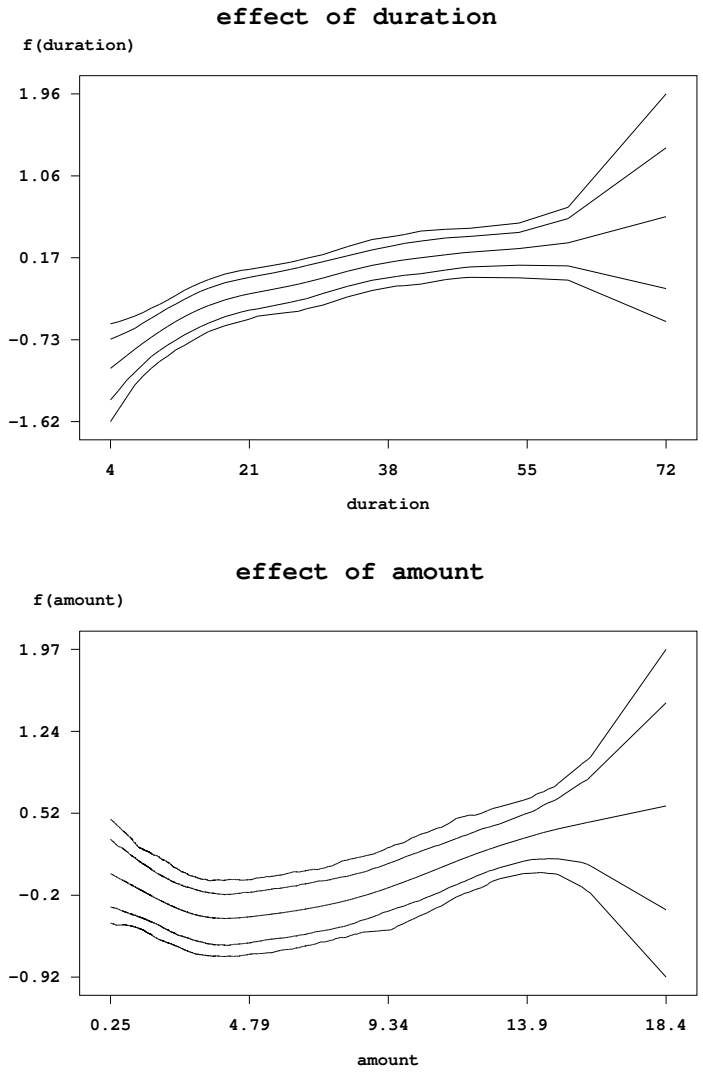


Figure 7.3: Improved plots of the effect of `duration` *and* `amount`.

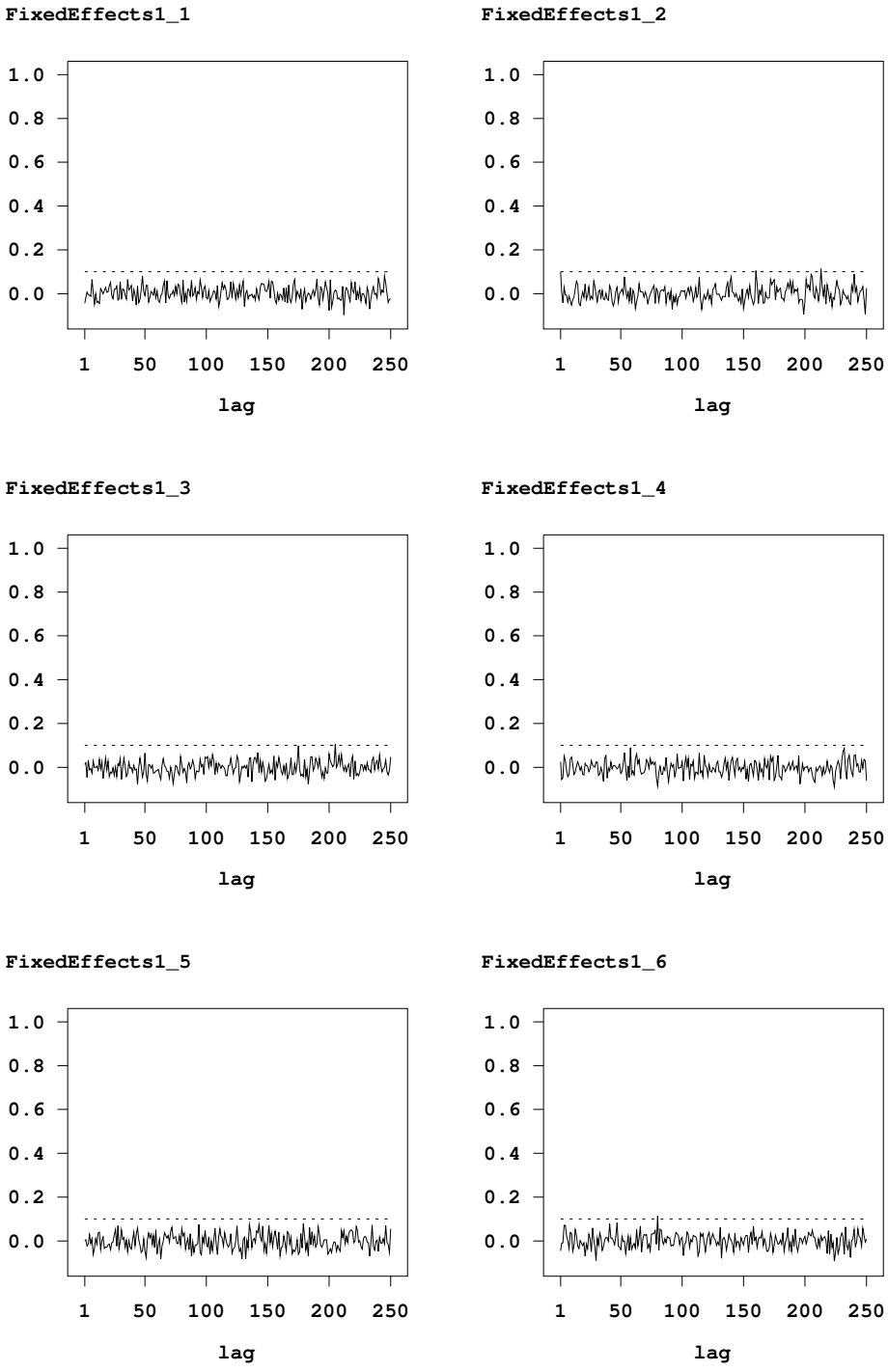


Figure 7.4: First page of the autocorrelation file.

7.6.1.4 Logit models

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

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

In contrast to binary probit models, the full conditionals for the regression coefficients are no longer Gaussian. *BayesX* offers 3 different types of proposal densities. These are iteratively weighted least squares (IWLS) proposals based either on the current state of the parameters or on the posterior modes as described in subsubsection 6.1.3 or Brezger & Lang (2006), and conditional prior proposals as described in Fahrmeir & Lang (2001). We recommend the usage of IWLS proposals, since no tuning is required and mixing properties are superior to those of conditional prior proposals. The default are IWLS proposals based on the current state of the parameters. The following statement causes *BayesX* to use IWLS proposals based on posterior modes, which usually yield even higher acceptance probabilities compared to ordinary IWLS proposals:

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

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

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

The resulting graphs are shown in Figure 7.5. As could have been expected only the scale of the estimated effects differs (because of the logit link).

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

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

The first page of the resulting postscript file is shown in Figure 7.6. As can be seen, the autocorrelations for the logit model with IWLS proposals are almost as low as for the probit model.

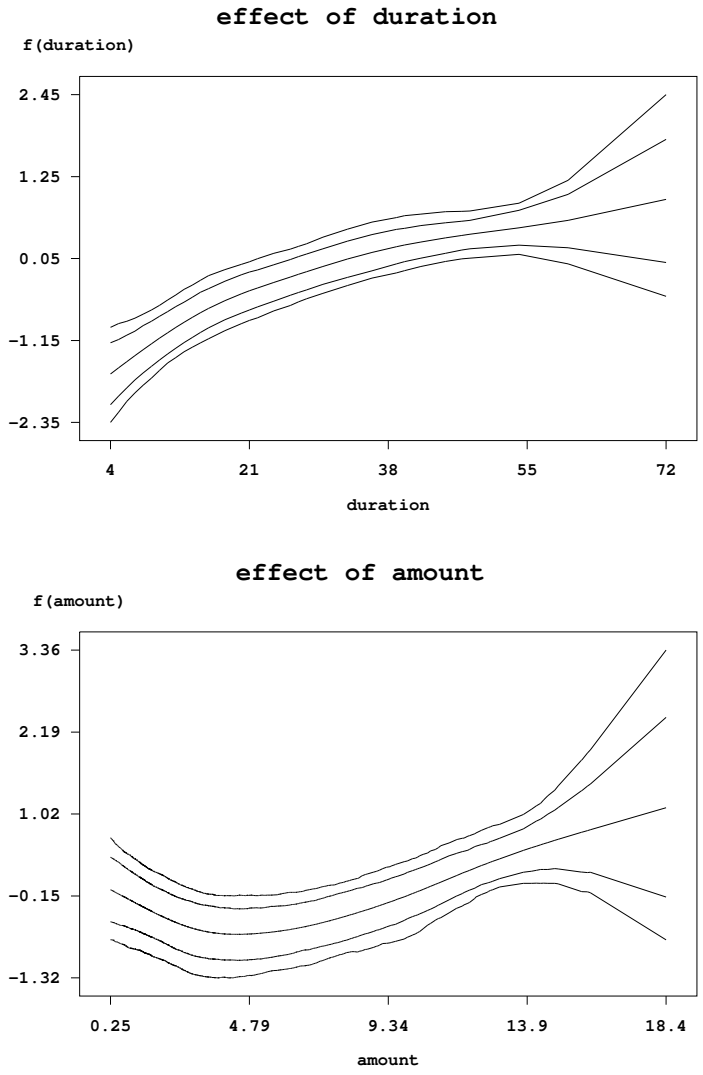


Figure 7.5: Effect of `duration` *and* `amount`*, if a logit model is estimated rather than a probit model.*

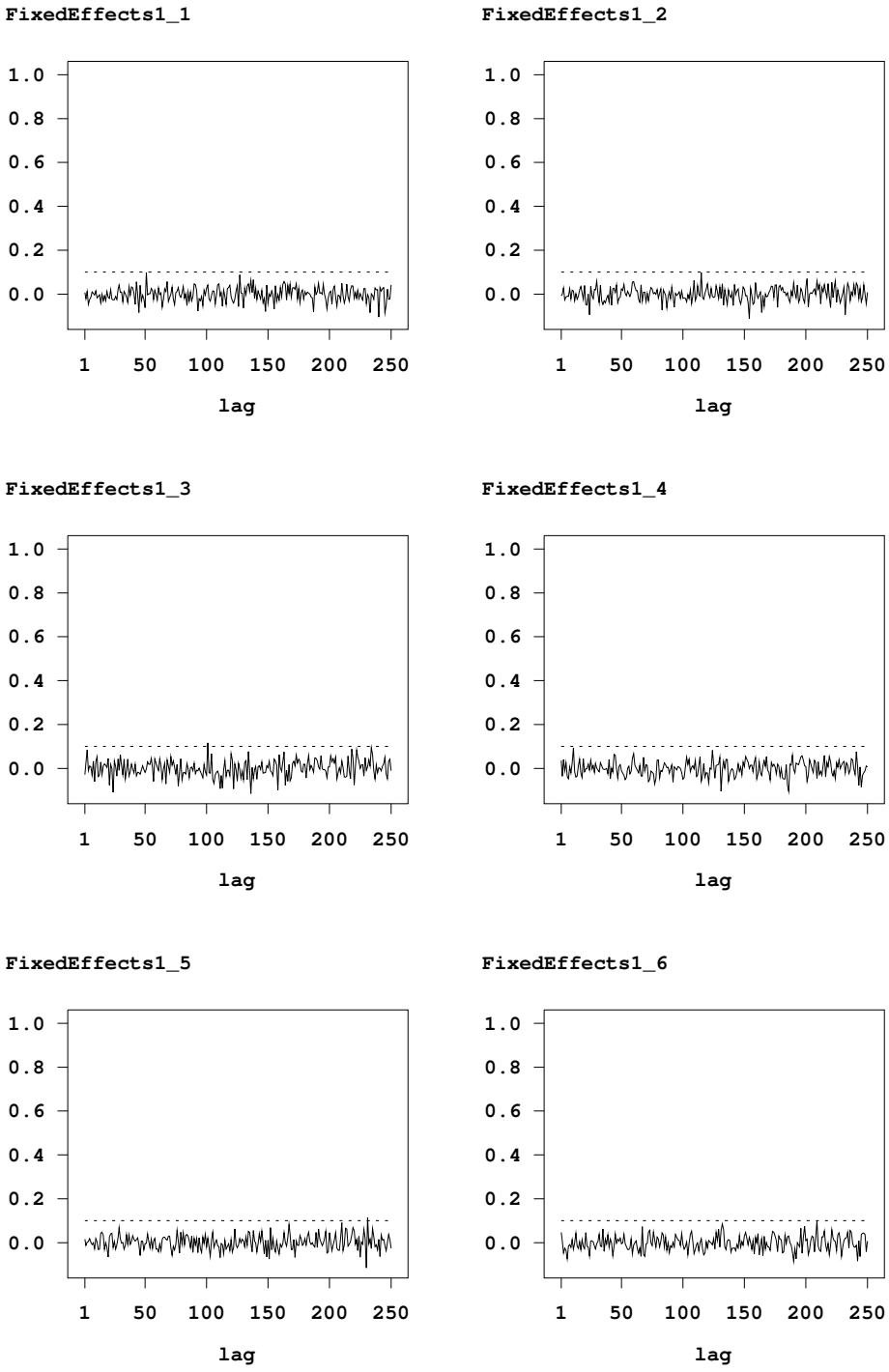


Figure 7.6: First page of the autocorrelation file, if a logit model is estimated.

7.6.1.5 Varying the hyperparameters

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

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

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

Figure 7.7 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`.

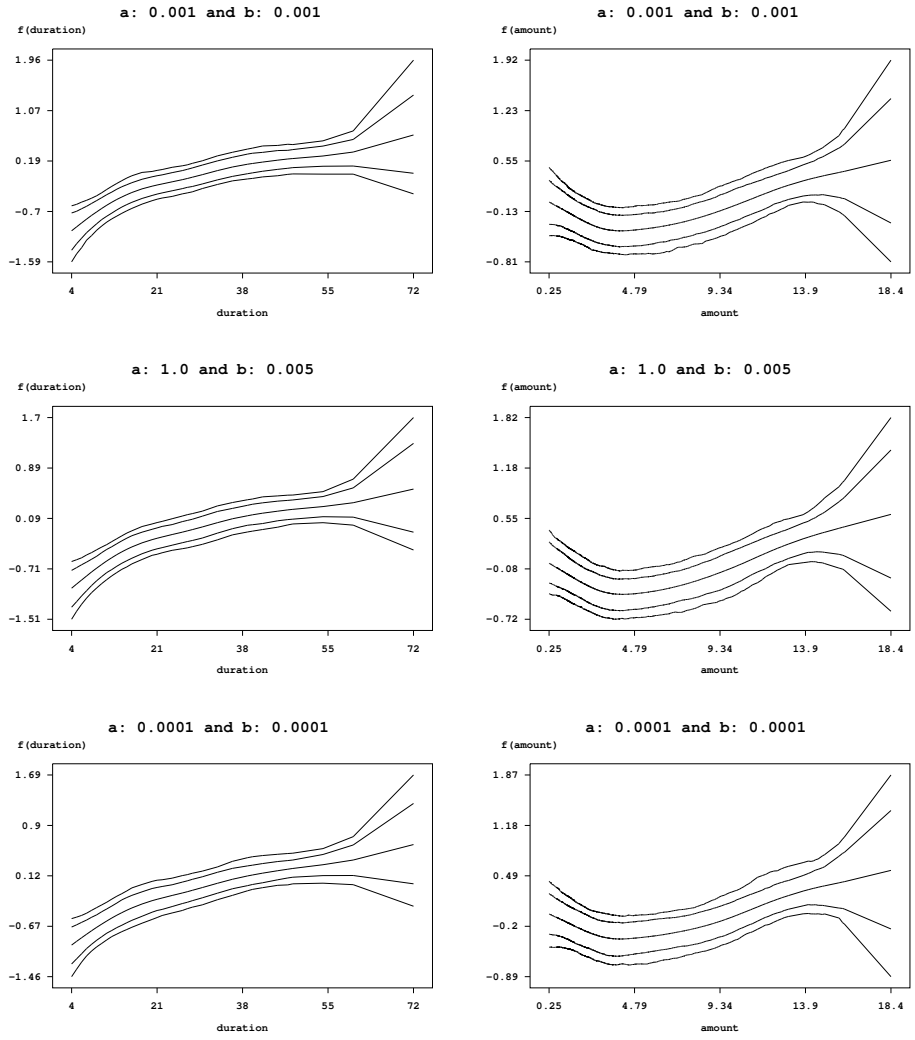


Figure 7.7: Results for the effect of `duration` *and* `amount` *for different values of the hyperparameters for the variances.*

Chapter 8

remlreg objects

Remlreg objects are used to fit (multivariate) exponential family, hazard rate or multi-state models with *structured additive predictor* subsumed in the class of *structured additive regression (STAR)* models, see Fahrmeir, Kneib & Lang (2004). Inference is based on a mixed model representation of the regression model and yields either penalised likelihood estimates (from a frequentist perspective) or empirical Bayes / posterior mode estimates (from a Bayesian perspective). The methodological background is provided in considerable detail in the methodology manual. More details on models for univariate responses can be found in Fahrmeir, Kneib & Lang (2004). Kneib & Fahrmeir (2006) describe models for categorical responses. Models for continuous time survival analysis based on structured hazard regression can be found in Kneib & Fahrmeir (2007). Interval censoring and some further extensions are discussed in Kneib (2006).

First steps with *remlreg objects* can be done with the example in chapter 2 of the tutorial manual which provides a self-contained demonstrating example.

8.1 Method regress

8.1.1 Syntax

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

Method `regress` estimates the regression model specified in *model* using the data specified in *dataset*. *dataset* has to be the name of a *dataset object* created before. The details of correct model specification are covered in subsubsection 8.1.1.2. The distribution of the response variable can be either Gaussian, binomial, multinomial, Poisson or gamma. In addition, *BayesX* supports continuous time survival and multi-state models, see also Table 7.6 for a more detailled overview.. The response distribution is specified using option `family`, see subsubsection 8.1.1.4 below. The default is `family=binomial` with a logit link. An `if` statement can be specified to analyze only parts of the data set, i.e. the observations where *expression* is true.

8.1.1.1 Optional weight variable

An optional weight variable *weightvar* can 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 represent the number of observations in the groups if the y_i's are the average of individual responses. If the y_is are the sum of responses in every group, the weights have to 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, the weight variable should correspond to the number of replications while the values of the response variable should represent 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 have to specify the number of observations in a group while the y_is are assumed to be the average of individual responses. Weights are not allowed for models with categorical response, continuous survival time models and multi-state models.

8.1.1.2 Syntax of possible model terms

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

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

depvar specifies the dependent variable whereas $term_1$,...,$term_r$ define the form of covariate influences. The different terms must be separated by '+' signs. A constant intercept is automatically included in the model and therefore has not to be specified by the user.

This section reviews all possible model terms supported in the current version of *remlreg objects* and provides some specific examples. Note that all described terms may be combined in arbitrary order. An overview about the capabilities of *remlreg objects* is given in Table 8.1. Table 8.2 shows how interactions between covariates are specified. Full details about all available options are given in subsubsection 8.1.1.3.

Throughout this section Y denotes the dependent variable.

Type	**Syntax example**	**Description**
Offset	`offs(offset)`	Variable `offs` is an offset term.
Linear effect	`W1`	Linear effect of `W1`.
Category-specific linear effect	`W1(catspecific)`	Category-specific linear effect of `W1` (in cumulative or sequential models only).
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. Gaussian (random) effect of the group indicator `grvar`, e.g. `grvar` may be an individual indicator when analyzing longitudinal data.
Baseline in Cox or multi-state 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 random walk penalty.

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

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 or multi-state models	`X1*time(baseline)`	Nonlinear, time-varying effect of `X1`.

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

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
```

Category-specific fixed effects

Description: In cumulative and sequential models for ordered categorical responses, fixed effects may either be defined globally or category-specific. To request the estimation of category-specific fixed effects, the keyword `catspecific` has to be specified. Category-specific effects can only be estimated for the response families `cumlogit`, `cumprobit`, `seqlogit`, and `seqprobit`.

Predictor: $\eta^{(j)} = \cdots + \gamma_1^{(j)} W1 + \cdots$

Syntax:

`W1(catspecific)`

Example:

The following model statement causes `regress` to estimate a model with category-specific effect of covariate `W1` and a global effect of covariate `W2`:

```
Y = W1(catspecific) + W2
```

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 that `X1` is a continuous covariate with possibly nonlinear effect. The following model statement defines a second order random walk prior for f_1:

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

The term `X1(rw2,a=0.001,b=0.001)` indicates, that the effect of `X1` should be incorporated nonparametrically using a second order random walk prior. A first order random walk can be requested by specifying `X1(rw1)` instead.

P-spline with first or second order random walk penalty

Description: Defines a P-spline with 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)
```

Seasonal component for time scales

Description: Defines a time-varying 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 by

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

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

Nonlinear baseline effect in continuous time survival or multi-state 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`) 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 covariate. The following Cox-type model with hazard rate λ and log-baseline effect $\log(\lambda_0(\texttt{time}))$

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

can be estimated by the model statement

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

Similarly, baseline effects for the transition intensities in multi-state models can be specified.

Spatial Covariates

Markov random field

Description:

Defines a Markov random field prior for the spatial covariate `region`. *BayesX* allows to incorporate spatial covariates with geographical information stored in the *map object* specified in option `map`.

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

Syntax:

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

Example:

For the specification of a Markov random field prior, `map` is an obligatory argument that represents the name of a *map object* (see chapter 5) containing 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.

Two-dimensional P-spline with first order random walk penalty

Description:

Defines a two-dimensional P-spline for the spatial covariate `region` with a two-dimensional first order random walk penalty for the parameters of the spline. Estimation is based on the coordinates of the centroids of the regions. The centroids are computed using the geographical information stored in the *map object* specified in the option `map`.

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

Syntax:

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

Example:

For the specification of a two-dimensional P-spline (*geospline*) `map` is an obligatory argument indicating 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 two-dimensional cubic P-spline with first order random walk penalty where the geographical information is stored in the *map object* `germany`.

Stationary Gaussian random field

Description:

Defines a stationary Gaussian random field for the spatial covariate `region`. 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 in option `map`.

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

Syntax:

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

Example:

For the specification of a stationary Gaussian random field (`geokriging`), `map` is an obligatory argument indicating the name of a *map object* (see chapter 5). The model term

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

specifies a stationary Gaussian random field 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:

Gaussian i.i.d. random effects allow 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 defined by

```
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 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`. Therefore covariate `X2` is called the effect modifier. The smoothness prior for $f(X2)$ 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`. 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)
```

If the effect of a covariate should vary according to different types of effect modifiers, this leads to similar identification problems as in usual additive models. To avoid such problems, option `center` can be specified to request the estimation of centered effects. For example, if both `X2` and `Z2` are assumed to modify the effect of `X1`, the specification of

```
Y = X1*X2(psplinerw2) + X1*Z2(psplinerw2)
```

yields a non-identifiable model. In contrast

```
Y = X1 + X1*X2(psplinerw2, center) + X1*Z2(psplinerw2, center)
```

is well-identified. Note that the main effect of `X1` has to be included separately. Equivalently, we could absorb the main effect into the first term, yielding

```
Y = X1*X2(psplinerw2) + X1*Z2(psplinerw2, center)
```

However, the former specification has the advantage that the model terms are clearly separated.

Models of the type just discussed arise for example if `X1` is a binary dummy-variable indicating two different groups of data. In this case the model

```
Y = X1 + X2(psplinerw2) + X1*X2(psplinerw2, center) + Z2(psplinerw2) + X1*Z2(psplinerw2,
```

assumes different effects of both `X2` and `Z2` in the groups.

Seasonal prior

Description:

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

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 a binary variable `X1`. In this case we can include an additional seasonal effects for observations with `X1`=1 by

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

Time-varying effects in continuous-time or multi-state 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` is assumed to have a 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 and `delta` is the vector of corresponding indicators of non-censoring. The following Cox model with hazard rate

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

is estimated by the model statement

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

Similarly, time-varying effects on the transition intensities in multi-state models can be specified.

Varying coefficients with spatial covariates as effect modifiers

Markov random field

Description:

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

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

Syntax:

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

Example:

For example the statement

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

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

Similarly as for varying coefficient terms with continuous effect modifiers, varying coefficients with spatial effect modifier can be centered to avoid identifiability problems:

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

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*, the main effects are *not* 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

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

would be

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

Surface estimators

Two-dimensional P-spline with first order random walk penalty

Description:

Defines a two-dimensional P-spline based on the tensor product of one-dimensional P-splines with a two-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 one-dimensional main effects of `X1` and `X2` into the models. In this case the two-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 two-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 options chosen, 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 in principle stationary Gaussian random fields can be used to estimate surfaces depending on arbitrary variables `X1` and `X2`, but 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 mentioned in section 4.2 of the methodology manual. If all distinct pairs of `X1` and `X2` shall be used as knots, we have to specify

```
Y = X1*X2(kriging,full)
```

Note, that the knots computed by the space filling algorithm will be stored in a file in the outfile directory of the *remlreg object* (the file name will be printed in the output window with the estimated effects). These knots can be read into a *dataset object* which may be passed to the kriging term if we want to use the same knots as in previous calls:

```
dataset kn
kn.infile using knotfile
Y = X1*X2(kriging,knotdata=kn)
```

To determine the actual number of knots, the options are interpreted in a specific sequence. If option `full` is specified, both `nrknots` and `knotdata` are ignored. Similarly, `nrknots` is ignored if `knotdata` is specified.

8.1.1.3 Description of additional options for terms of remlreg objects

All arguments described in this section are optional and may be omitted. Generally, options are specified by adding the option name to the specification of the model term type in the parentheses, separated by commas. All options may be specified in arbitrary order. Table 8.3 provides explanations and the default values of all possible options. All reasonable combinations of model terms and options can be found in Table 8.4.

optionname	description	default
`lambdastart`	Starting value for the smoothing parameter λ.	`lambdastart=10`
`degree`	Degree of B-spline basis functions.	`degree=3`
`nrknots`	Number of inner knots for a P-spline term or 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 in the kriging term.	-
`nu`	The smoothness parameter ν of the Matèrn correlation function for kriging terms.	`nu=1.5`
`maxdist`	Specifies the value c that is used to determine the scale parameter ρ of the Matèrn correlation function for kriging terms. Compare section 4.2 of the methodology manual.	default depends on `nu`
`p`	Parameter p of the coverage criterion for the space filling algorithm that determines the knots of a kriging term.	`p=-20`
`q`	Parameter q of the coverage criterion for the space filling algorithm that determines the knots of a kriging term.	`q=20`
`maxsteps`	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 and multi-state models. May be either '`quantiles`', '`equidistant`' or '`all`'.	`gridchoice=quantiles`
`tgrid`	Number of equidistant time points to be used for numerical integration in Cox and multi-state 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 and multi-state 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 and multi-state 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`	Period of a seasonal effect. The default (`period=12`) corresponds to monthly data.	`period=12`
`catspecific`	Requests that the corresponding effect should be modelled category-specific. Can only be used in cumulative and sequential models for categorical responses, i.e. with response families `cumlogit`, `cumprobit`, `seqlogit` and `seqprobit`.	-
`center`	For varying coefficient terms this option requests that the effect should be centered to avoid identifiability problems	-

Table 8.3: Optional arguments for remlreg object *terms.*

8.1.1.4 Specifying the response distribution

Supported univariate distributions are Gaussian, binomial (with logit, probit or cumulative log-log link), Poisson and gamma. For multivariate responses, *BayesX* supports multinomial logit models

	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*	×
`catspecific`	△	△	△	△	△	△	△	△	△	×
`center`	△	△	△	△	×	△	×	×	△	×

*	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`, `equidistant` and `all`

Table 8.4: Terms and options for remlreg objects.

for categorical responses with unordered categories and cumulative as well as sequential logit and probit models for categorical responses with ordered categories. Continuous survival times as well as multi-state models can be analysed based on semiparametric models with Cox-type hazard rates. An overview over the supported models is given in Table 8.5. The distribution of the response is specified by adding the additional option `family` to the (global) options list of the regression call. For instance, `family=gaussian` defines the response to be Gaussian distributed. In some cases, one or more additional options associated with the specified response distribution can 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.

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` `family=multinomialcatsp`	unordered multinomial unordered multinomial (with category-specific covariates)	logit logit	`reference` `reference`
`family=cumprobit` `family=cumlogit`	cumulative multinomial cumulative multinomial	probit logit	
`family=seqprobit` `family=seqlogit`	sequential multinomial sequential multinomial	probit logit	
`family=poisson`	Poisson	log	
`family=gamma`	gamma	log	
`family=cox`	continuous-time survival data		`leftint`, `lefttrunc`
`family=multistate`	continuous-time multi-state data		`state`, `lefttrunc`

Table 8.5: Summary of supported response distributions.

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 $y|\eta, \sigma^2 \sim N(\eta, \sigma^2 C^{-1})$, where $C = diag(weightvar_1, \ldots, weightvar_n)$ is a known weight matrix. Gaussian regression models are obtained by adding

`family=gaussian`

to the options list.

An optional weight variable *weightvar* can be specified to estimate weighted regression models, see subsubsection 10.1.2.1 for details. For grouped Gaussian responses, the weights represent the number of observations in the groups if the y_i's are the average of individual responses. If the y_is are the sum of responses in every group, the weights are given by 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 no weight variable is specified, *BayesX* assumes $weightvar_i = 1$, $i = 1, \ldots, n$.

Binomial logit, probit and complementary log-log models

A binomial logit model is requested by the option

```
family=binomial
```

while a probit model is obtained with

```
family=binomialprobit
```

and a complementary log-log model with

```
family=binomialcomploglog
```

A additional weight variable may be specified, see subsection 8.1.1 for the syntax. *BayesX* assumes that the weight variable corresponds to the number of replications and the response variable to the number of successes. If the weight variable is omitted, *BayesX* assumes that the number of replications is one, i.e. the values of the response must be either zero or one.

Multinomial logit models

So far, *remlreg objects* support only multinomial logit models and no probit models. A multinomial logit model without category-specific covariates is specified by adding the option

```
family=multinomial
```

to the options list.

If there are category-specific covariates, the option has to be altered to

```
family=multinomialcatsp
```

Category-specific covariates are included as follows: Suppose that covariate `x` has been observed for a response variable with the three categories 1, 2 and 3. Then you have to include the three variable `x1`, `x2` and `x3` into your dataset. Within the regression syntax, you have to specify

```
x_catspecific
```

to request a parametric effect of `x`

```
x_catspecific(psplinerw2)
```

for a nonparametric effect of `x` and

```
x_catspecific*id(psplinerw2)
```

to obtain a random effect of `x` with respect to the grouping variable `id`. Currently BayesX only supports these three term types for effects of category-specific effects.

For both `family=multinomial` and `family=multinomialcatsp` a second option (`reference`) can be added to the options list to define the reference category. If the response variable has three categories 1, 2 and 3, the reference category can be set to 2, by adding

```
reference=2
```

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

If some categories are not available for some observations, *BayesX* can account for this by including either category-specific offsets or non-availability indicators. Suppose again that the response has the three categories 1, 2 and 3. Then offset terms `o1`, `o2` and `o3` can be used to account for varying choice sets. If a category is available, the offset is simply set to zero, while a large negative value (i.e. -1000) has to be assigned to the offset term of categories which are not available. To account for the offset term,

```
o_catspecific(offset)
```

has to be added to the model specification. Of course, you can also assign different values to the

offsets, e.g. to account for a priori differences in the availability of some categories.

The usage of offset terms to account for non-availability may in some cases be numerically unstable (e.g. if several categories are not available or if the reference category is not available). Therefore an alternative possibility is to include non-availability indicators `na1`, `na2` and `na3`. Each of the indicators is assigned the value one if the corresponding category is not available and zero otherwise. Within the regression syntax, the non-availability indicator has to be specified as a global option, i.e.

```
... , family=multinomialcatsp naindicator=na_catspecific
```

Cumulative logit and probit models

A cumulative logit model is specified by adding

`family=cumlogit`

to the options list, a cumulative probit is obtained by

`family=cumprobit`

In both cases, 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 about 5 categories estimation may become rather computer intensive and time demanding (depending on the size of your data set).

By default, all effects in cumulative logit and probit models are considered to be defined globally. To obtain category-specific effects, the additional keyword `catspecific` has to be specified. For example, the specification of the predictor

```
Y = W1(catspecific) + W2 + X1(psplinerw2, catspecific) + X2(psplinerw2)
```

requests category-specific effects for the covariates `W1` and `X1`, and global effects for the covariates `W2` and `X2`. Note that complicated ordering restrictions have to be fulfilled for the covariate-dependent thresholds defined implicitly by category-specific effects. Therefore numerical problems are likely to be observed in models with sparse data or a lot of category-specific effects.

Sequential logit and probit models

A sequential logit model is specified by adding

`family=cumlogit`

to the options list, while a sequential probit can be requested by

`family=cumprobit`

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

Similar as in cumulative models, all effects in sequential logit and probit models are considered to be defined globally by default. To obtain category-specific effects, the additional keyword `catspecific` has to be specified. For example, the specification of the predictor

```
Y = W1(catspecific) + W2 + X1(psplinerw2, catspecific) + X2(psplinerw2)
```

requests category-specific effects for the covariates `W1` and `X1`, and global effects for the covariates `W2` and `X2`. In contrast to cumulative models no ordering restrictions are imposed in sequential models.

Poisson regression

A Poisson regression model is specified by adding

`family=poisson`

to the options list.

A weight variable may be specified in addition, see subsection 8.1.1 for the syntax. For grouped Poisson data, the weights must be the number of observations in a group and the responses are assumed to be the average of individual responses.

Gamma distributed responses

In the literature, the density function of the gamma distribution is parameterized in various ways. In the context of regression analysis, the density is usually parameterized in terms of the mean μ and the scale parameter s. Then, the density of a gamma distributed random 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 typically employed for hyperparameters `a` and `b` of priors for variance parameters in the context of Bayesian hierarchical models. In this case, the density is 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 variable is parameterised in the first form (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* can be specified to estimate weighted regression models, see subsection 8.1.1 for the syntax.

Continuous time survival analysis

BayesX offers two alternatives of estimating continuous time survivals models with semiparametric predictor η, both of which are described in subsection 7.2 of the methodology manual. The first alternative is to assume that all time-dependent values are piecewise constant, leading to the so called *piecewise exponential model* (p.e.m.). The second alternative is to estimate the log-baseline effect $\log(\lambda_0(t)) = f_0(t)$ based on a P-spline with second order random walk penalty.

Piecewise exponential model (p.e.m.)

In subsection 7.2 of the methodology manual we demonstrated how continuous time survival data has to be manipulated to transform it to a Poisson for model estimation. Suppose that the following modified data set is available

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 7.2 of the methodology manual. Let `x1` be a covariate with linear effect and `x2` a continuous covariate with nonlinear effect. Then the correct syntax for estimating a p.e.m. with a *remlreg object* named `r` is e.g. as follows:

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

or

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

Note that a time-varying effect of an additional covariate `X` may be estimated 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 a continuous time survival model with a P-spline prior with second order random walk penalty for the baseline effect,

```
family=cox
```

has to be specified in the options list. The number of knots and degree of the P-spline prior for $f_0(t)$ can be specified as additional options for the baseline term. Note that it is obligatory 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. In the example above with survival data

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

a continuous time survival 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 ...
```

Again a time-varying effect of a covariate `X` can be estimated by simply adding the term

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

to the model statement.

Interval censoring and left truncation

Interval censoring and left truncation can be incorporated using the additional options `leftint` and `lefttrunc` of *remlreg objects*. These two variables represent the lower interval boundary T_{lo} and the left truncation time T_{tr} as discussed in section 7.3 of the methodology manual. The time variable specified in the baseline statement corresponds to T_{up}, the upper boundary of the interval. In general an observation can now be described completely by the quadruple $(T_{tr}, T_{lo}, T_{up}, \delta)$, with

$$\begin{array}{ll} T_{lo} = T_{up},\ \delta = 1 & \text{if the observation is uncensored,} \\ T_{lo} = T_{up},\ \delta = 0 & \text{if the observation is right censored,} \\ T_{lo} < T_{up},\ \delta = 0 & \text{if the observation is interval censored.} \end{array}$$

For left truncated observations we have $T_{tr} > 0$ while $T_{tr} = 0$ for observations which are not truncated.

An example for a statement that estimates a model with left truncation and interval censoring is given by

```
> r.regress delta = tup(baseline)+ x1 + x2(psplinerw2), family=cox
  lefttrunc=ttr leftint=tlo ...
```

Continuous time multi-state models

Multi-state models describe the temporal development of discrete phenomena in continuous time based on transition intensities for each of the observable transition types. Consider for example a multi-state model for human sleep as depicted in Figure 8.1 and that the transition intensities for the four possible transitions are specified as

$$\begin{aligned} \lambda_{AS,i}(t) &= \exp\left[g_0^{(AS)}(t) + b_i^{(AS)}\right], \\ \lambda_{SA,i}(t) &= \exp\left[g_0^{(SA)}(t) + b_i^{(SA)}\right], \\ \lambda_{NR,i}(t) &= \exp\left[g_0^{(NR)}(t) + c_i(t) g_1^{(NR)}(t) + b_i^{(NR)}\right] \\ \lambda_{RN,i}(t) &= \exp\left[g_0^{(RN)}(t) + c_i(t) g_1^{(RN)}(t) + b_i^{(RN)}\right] \end{aligned}$$

Each of the transitions is parameterised in terms of a baseline effect $g_0^{(h)}(t)$ and a transition specific frailty term (random effect) $b_i^{(h)}$. In addition, time-varying effects $g_1^{(h)}(t)$ of binary indicators $c_i(t)$ for a high blood level of cortisol are introduced for the transitions between REM and Non-REM.

The corresponding data set should be arranged as follows:

id	st	beg	end	tas	tsa	trn	tnr	cort	corthigh
1	2	0	1	0	1	0	0	52.6	0
1	1	1	5	1	0	0	0	52.6	0
1	2	5	8	0	1	0	0	52.6	0
1	1	8	10	1	0	0	0	52.6	0
1	2	10	36	0	0	0	0	52.6	0
1	2	36	76	0	0	0	0	46.9	0
1	2	76	108	0	0	0	1	47.5	0
1	3	108	109	0	0	1	0	47.5	0
1	2	109	110	0	0	0	1	47.5	0
1	3	110	111	0	0	1	0	47.5	0

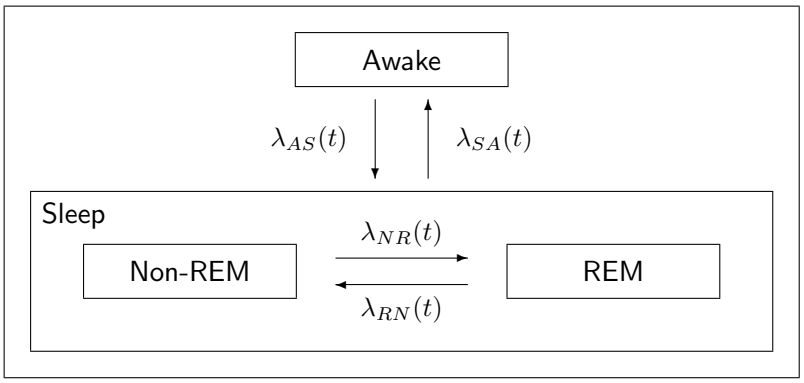


Figure 8.1: Schematic representation of sleep stages and transitions of interest.

```
1   2   111    115  0   0   0   1   47.5  0
1   3   115    116  0   0   0   0   47.5  0
1   3   116    126  0   0   1   0   37.4  0
.   .    .      .   .   .   .   .     .   .
.   .    .      .   .   .   .   .     .   .
.   .    .      .   .   .   .   .     .   .
2   2   0      12   0   1   0   0   22.5  0
2   1   12     15   1   0   0   0   22.5  0
2   1   15     28   0   1   0   0   88.6  1
.   .    .      .   .   .   .   .     .   .
.   .    .      .   .   .   .   .     .   .
.   .    .      .   .   .   .   .     .   .
```

Each path observed for the multi-state model is transformed into several lines in the data set, where `id` identifies the original paths. In the above example, parts of the first two observations are displayed. Each line of the data set represents a time interval identified by the variables `beg` and `end`. Variable `st` indicates the current state of the process. Note that the states have to be numbered consecutively from 1 to H. Since we are considering continuous time scales, an observation should start at $t = 0$ (unless the observation is left truncated) and the variables `beg` and `end` should be generated so that within each observation process `beg` equals the value of `end` in the previous row (unless observations are fragmentary only).

The variables `tas`, `tsa`, `trn` and `tnr` are binary indicators for the four transitions sleep → awake (`tsa`), awake → sleep (`tas`), Non-REM → REM (`tnr`) and REM → Non-REM (`trn`). Such an indicator equals one if the corresponding transition is observed at the end of the interval and zero otherwise. Note that there are lines in the data set, where none of the transitions is observed. These correspond to intervals where the value of the time-varying covariate `cort` (cortisol-level) changes. The variable `corthigh` is a dichotomized version of `cort` which indicates a high level of cortisol (`cort`>60).

The model specified above is estimated by entering the following command

```
> remlreg msm
> msm.mregress tas = end(baseline) + id(random):
               tsa = end(baseline) + id(random):
               trn = end(baseline) + corthigh*end(baseline) + id(random):
               tnr = end(baseline) + corthigh*end(baseline) + id(random),
```

```
family=multistate lefttrunc=beg state=st using sleep
```

Note that a separate model equation has to specified for each transition with the binary transition indicator as response. Instead of method `regress`, method `mregress` has to be called since multiple model equations are combined. The right and the left boundary of the time intervals have to specified as covariate for the the baseline effect and as global option `lefttrunc`, respectively. Similarly, the state variable has to specified via the global option `state`.

8.1.2 Options

Options for controlling the estimation process

- `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 close to the boundary of their parameter space, the usual Fisher-scoring algorithm for their determination has to be modified. If the fraction of the penalized part of an effect relative to the total effect is less than `lowerlim`, the estimation of the corresponding variance is stopped and the estimator is defined to be the current value of the variance (see section 6.2 of the methodology manual for details).
  DEFAULT: `lowerlim = 0.001`

- `maxit =` *integer*
  Defines the maximum number of iterations to be used in estimation. Since the estimation process will not necessarily converge, it may be useful to define an upper bound for the number of iterations. Note, that *BayesX* returns results based on the current values of all parameters even if no convergence could be achieved within `maxit` iterations, but a warning message will be printed in the *output window*.
  DEFAULT: `maxit=400`

- `maxchange =` *realvalue*
  Defines the maximum value that is allowed for relative changes in parameters in one iteration to prevent the program from crashing because of numerical problems. Note, that *BayesX* produces results based on the current values of all parameters even if the estimation procedure is stopped due to numerical problems, but an error message will be printed in the *output window*.
  DEFAULT: `maxchange=1000000`

Options for the analysis of survival times and multi-state models

- `leftint =` *variablename*
  Gives the name of the variable that contains the lower (left) boundary T_{lo} of the interval $[T_{lo}, T_{up}]$ for an interval censored observation. For right censored or uncensored observations we have to specify $T_{lo} = T_{up}$. If `leftint` is missing, all observations are assumed to be right censored or uncensored, depending on the corresponding value of the censoring indicator.

- `lefttrunc =` *variablename*
  Option `lefttrunc` specifies the name of the variable containing the left truncation time T_{tr}.

For observations that are not truncated, we have to specify $T_{tr} = 0$. If `lefttrunc` is missing, all observations are assumed to be not truncated. For multi-state models variable `lefttrunc` specifies the left endpoint of the corresponding time interval (compare page 125).

- `state =` *variablename*
  For multi-state models, `state` specifies the current state of the process (compare page 125).

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 options 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 (80%). Adding, for instance,

  ```
  level2=70
  ```

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

8.1.3 Estimation output

The way the estimation output is presented depends on the estimated model. Estimation results for fixed effects are displayed in a tabular form in the *output window* and/or in a log file (if created before). This table will contain the posterior mode, the standard deviation, p-values and an approximate 95% credible interval. Other credible intervals may be obtained by specifying the `level1` option, see subsection 8.1.2 for details. Additionally, a file replicating results for the fixed effects is created. The name of this file is supplied in the *output window* and/or in a log file.

Estimated nonparametric effects are presented in a different way. Here, results are stored in external ASCII-files that can be read into any general purpose statistics program (e.g. STATA, R, 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 names of the files and the storing directory are displayed in the *output window* and/or a log file. The files contain ten columns (for main effects) or eleven columns (for interaction effects). The first column contains a parameter index (starting with one), the second column (and the third column if the estimated effect is an interaction) contain the values of the covariate(s) whose effect has been 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 while a value of -1 to a strictly negative credible interval. A value of 0 indicates that the corresponding credible interval contains zero. Other credible intervals

and posterior probabilities may be obtained by specifying the `level1` and/or `level2` option, see subsection 8.1.2 for details. As an example, compare the following lines, which are the beginning of a file containing the results for a nonparametric effect of a particular covariate, x say:

intnr	x	pmode	ci95lower	ci80lower	std	ci80upper	ci95upper	pcat95	pcat80
1	-2.87694	-0.307921	-0.886815	-0.686408	0.295295	0.070567	0.270973	0	0
2	-2.86203	-0.320479	-0.885375	-0.689815	0.288154	0.0488558	0.244416	0	0
3	-2.8515	-0.329367	-0.88473	-0.69247	0.283292	0.0337362	0.225997	0	0
4	-2.85066	-0.330072	-0.884692	-0.692689	0.282913	0.0325457	0.224549	0	0
5	-2.82295	-0.3535	-0.884544	-0.700703	0.270887	-0.00629671	0.177545	0	-1
6	-2.79856	-0.37418	-0.886192	-0.708939	0.261178	-0.0394208	0.137832	0	-1
7	-2.79492	-0.377272	-0.886579	-0.710263	0.259798	-0.0442813	0.132035	0	-1
8	-2.79195	-0.379788	-0.886921	-0.711358	0.258689	-0.0482183	0.127345	0	-1
9	-2.78837	-0.382834	-0.887367	-0.712704	0.257363	-0.0529641	0.1217	0	-1

Note that the first row of the files always contains the names of the columns.

The estimated nonlinear effects can be visualized by using either the graphics capabilities of *BayesX* or the *BayesX* R package, see section 11.1 and section 11.2, respectively. Of course, any other (statistics) software package with plotting facilities may be used as well.

Estimation results for the variances and the smoothing parameters of nonparametric effects are printed in the *output window* and/or a log file. Additionally, a file is created containing the same information. For example, the file corresponding to the nonparametric effect presented above contains:

variance	smoothpar	stopped
0.0492324	20.3118	0

The value in the last row indicates whether the estimation of the variance has been stopped before convergence. A value of 1 corresponds to a 'stopped' variance.

8.1.4 Examples

Here we give only a few examples about the usage of method `regress`. A more detailed, tutorial like example can be found in chapter 2 of the tutorial manual.

Suppose that we have a data set `test` with a binary response variable `y`, and covariates `x1`, `x2`, `x3`, `t` and `region`, where `t` is assumed to be a time scale measured in months and `region` indicates the geographical region an observation belongs to. Suppose further that we have already created a *remlreg object* `r`.

Fixed effects

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

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

This model is estimated by typing:

```
> r.regress y = x1 + x2 + x3, family=binomial using test
```

By specifying option `family=binomial`, a binomial logit model is estimated. A probit model can be obtained 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

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

If the response is not binary but categorical with unordered categories 1, 2 and 3, we can estimate a multinomial logit model by typing:

```
> r.regress y = x1 + x2(psplinerw2) + x3(psplinerw2), family=multinomial
  reference=2 using test
```

In this case, `family=binomial` has to be altered to `family=multinomial`, and the option `reference=2` was added to define the value 2 as the reference category.

Time scales

In the 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 have to create a *map object* first. Afterwards we read 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 augment 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 achieved by

```
> r.regress y = x1 + x2(psplinerw2) + x3(psplinerw2) + t(psplinerw2)
  + t(season,period=12) + region(spatial,map=m) + region(random),
  family=binomial using test
```

8.2 Global options

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

The syntax for specifying global options is

objectname.optionname = *newvalue*

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

Currently only one global option is available for *remlreg objects*:

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

  <INSTALLDIRECTORY>`\output`

  The default file name is composed of the name of the *remlreg object* and the type of the file. For example, if you estimated a nonparametric effect for a covariate `X`, say, using a 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 achieved using the `outfile` option. Here, you have to specify the directory where the output should be stored and a base file name. This base file name should not be a complete file name. For example specifying

  ```
  outfile = c:\data\res1
  ```

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

  ```
  c:\data\res1_f_X_pspline.res
  ```

8.3 Visualizing estimation results

Visualization of estimation results is described in chapter 11

Chapter 9

stepwisereg objects

stepwisereg objects are used to fit models with *structured additive predictor* subsumed in the class of *structured additive regression (STAR)* models, see Belitz & Lang (2008) and Fahrmeir, Kneib & Lang (2004). In addition to *bayesreg* and *remlreg objects* described in the previous two chapters, *stepwisereg objects* are also able to perform model choice and variable selection. Model choice and estimation of the parameters is done simultaneously. The algorithms of *stepwisereg objects* are able to

- decide whether a particular covariate enters the model,
- decide whether a continuous covariate enters the model linearly or nonlinearly,
- decide whether a spatial effect enters the model,
- decide whether a unit- or cluster specific heterogeneity effect enters the model,
- select complex interaction effects (two dimensional surfaces, varying coefficient terms),
- select the degree of smoothness of nonlinear covariate, spatial or cluster specific heterogeneity effects.

Inference is based on penalized likelihood in combination with fast algorithms for selecting relevant covariates and model terms. Different models are compared via various goodness of fit criteria, e.g. AIC, BIC, GCV and 5 or 10 fold cross validation. Models with structured additive predictor and the algorithms for model choice and variable selection are described in considerable detail in the methodology manual. More details on the algorithms for model choice and variable selection are given in Belitz & Lang (2008) and Belitz (2007).

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* is the name of a *dataset object* created before. The details of correct model specification are covered in subsubsection 9.1.1.2. The distribution of the response variable can be either Gaussian, binomial, multinomial, gamma or Poisson. 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 can be specified to analyze only parts of the data set, i.e. the observations where *expression* is true.

9.1.1.1 Optional weight variable

An optional weight variable *weightvar* can 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 represent 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 have to 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, the weight variable should correspond to the number of replications while the values of the response variable should represent 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 specify the number of observations in a group while the y_i's are assumed to be the average of individual responses. Weights are not allowed for models with multicategorical responses.

9.1.1.2 Syntax of possible model terms

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

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

depvar specifies the dependent variable whereas $term_1$,...,$term_r$ define the form of covariate influences. The different terms must be separated by '+' signs. A constant intercept is automatically included in the model.

This section reviews all possible model terms supported in the current version of *stepwisereg objects* and provides some specific examples. Note that all terms may be combined in arbitrary order. An overview about the capabilities of *stepwisereg objects* is given in Table 9.1. Table 9.2 shows how interactions between covariates are specified. Full details about all available options for the different term types are given in subsubsection 9.1.1.3.

Throughout this section Y denotes the dependent variable.

Offset

Description: Adds an offset to the predictor.

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

Type	Syntax example	Description
offset	`offs(offset)`	Variable `offs` is an offset term.
linear effect	`W1`	Linear effect for `W1`.
factor	`F1(factor)`	Effect of categorical variable `F1`
P-spline	`X1(psplinerw1)` `X1(psplinerw2)`	Nonlinear effect of `X1`.
degree zero P-spline	`X1(rw1)` `X1(rw2)`	Nonlinear effect of `X1`.
seasonal prior	`time(season)`	Varying seasonal effect of `time`.
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 are stored in the *map object* `m` .
Two dimensional P-spline	`region(geosplinerw1,map=m)` `region(geosplinerw2,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. Gaussian random effect of the group indicator `grvar` , e.g. `grvar` may be an individuum indicator when analyzing longitudinal data.

Table 9.1: Overview over different model terms for stepwisereg objects.

Syntax:

```
offs(offset)
```

Example:

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

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

Linear effects

Description: Incorporates covariate `W1` as a linear 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 linear effects:

```
Y = W1 + W2 + ··· + Wq
```

Effects of categorical covariates

Description: Creates dummy (or effect coded) variables of the categorical covariate `W` and incorporates the dummies into the model. Since *stepwisereg objects* perform variable selection the dummies for `W` are either completely included in the model or completely excluded. Hence it is not possible that only a subset of the dummies enter the model. Usually interpretation of results is difficult if only a subset of a categorical covariate is included by the variable selection algorithm.

Type of interaction	Syntax example	Description

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

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)` `X1*X2(pspline2dimrw2)`	Two dimensional surface for the continuous covariates `X1` and `X2`.

ANOVA type interaction	`X1*X2(psplineinteract) +`
	`X1(psplinerw2) +`
	`X2(psplinerw2)`
		ANOVA type interaction for the continuous covariates `X1` and `X2`. Note that P-splines with first order difference penalty for the main effects are possible as well.

Table 9.2: Possible interaction terms for stepwisereg objects.

Predictor: $\eta = \cdots + \gamma_1 W1 + \gamma_2 W2 + \cdots + \gamma_q Wq + \cdots$
where $W1, \ldots, Wq$ are the dummies for W.

Syntax:

`W(factor`[, *options*]`)`

`W(factor`[, *options*]`)`

Example:

The following model statement causes `regress` to estimate a model with categorical covariate W:

```
Y = W(factor)
```

By default, the category with value 1 (if existing or not) is assumed to be the reference. Usually the reference category is different from 1 and must be explicitly specified. This is achieved with option `reference`. For instance,

```
Y = W(factor,reference=3)
```

assumes the category with value 3 as the reference.

Effect coding rather than dummy coding is obtained by

```
Y = W(factor,reference=3,coding=effect)
```

While both coding schemes produce equivalent models effect coding is favorable from a technical point of view. In particular, convergence of the algorithms for model choice and variable selection is considerably improved with effect coding. Hence, effect coding is strongly recommended.

Nonlinear effects of continuous covariates and time scales

P-spline with first or second order difference penalty

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

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

Syntax:

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

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

Examples:

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)
```

To perform model choice and variable selection an ordered list of smoothing parameters must be defined, see subsection 6.3 in the methodology manual. *BayesX* automatically defines an appropriate list of smoothing parameters, i.e. it is usually not necessary to specify the smoothing parameters. However, in some situations it may be favorable to define the list of smoothing parameters manually. *stepwisereg objects* provide three alternative ways for manually specifying the ordered list of smoothing parameters of a model term. They are specified via option `sp` and - depending on its value - a couple of accompanying options.

- *Degrees of freedom:* The first alternative is to define the list of smoothing parameters via the concept of equivalent degrees of freedom of a model term. Zero (covariate excluded from the model) and one (linear fit) degrees of freedom are always included in the list. The remaining smoothing parameters are chosen such that they correspond to an equidistant grid of degrees of freedom between `dfmin` and `dfmax`. The total number of smoothing parameters, respectively degrees of freedom (in addition to 0 and 1), is supplied via the option `number`. For instance,

  ```
  Y = X1(psplinerw2,sp=df,dfmin=2,dfmax=5,number=4,dfstart)
  ```

  defines the smoothing parameters such that they correspond to 0,1,2,3,4 and 5 degrees of freedom. Specifying

  ```
  Y = X1(psplinerw2,sp=df,dfmin=2,dfmax=5,number=8)
  ```

  results in $0, 1, 2, 2.5, 3, 3.5, \ldots, 5$ possible degrees of freedom.

  In general, it is advisable to compare the list of degrees of freedom with the degrees of freedom of the selected model. In particular if the degree of freedom of the selected best model is close to or even equal to `dfmax` the selection should be rerun with an increased value for `dfmax`.

  It is also possible to define the smoothing parameters of the start model. This is done in two steps. First, the global option `startmodel=userdefined` must be set. Second, the smoothing parameter of the start model is defined via option `dfstart`. For instance,

  ```
  Y = X1(psplinerw2,sp=df,dfmin=2,dfmax=5,number=4,dfstart=2)
  , startmodel=userdefined
  ```

  defines a P-spline with 2 degrees of freedom for $X1$ as the start model. Note, however, that the model selection algorithms are typically not sensitive to the choice of the start model. Hence, option `dfstart` will be rarely used.

- *Degrees of freedom, smoothing parameters on a log-scale:* This alternative also defines the list of smoothing parameters via the equivalent degrees of freedom. The difference

to the first alternative is that the corresponding smoothing parameters between `dfmin` and `dfmax` are chosen on a log-scale resulting in non-equidistant degrees of freedom. A log-scale is specified using the additional option `logscale`. An example is the term:

```
Y = X1(psplinerw2,sp=df,dfmin=2,dfmax=5,number=4,logscale)
```

– *Smoothing parameters on a log-scale:* The third alternative allows to specify the smoothing parameters directly using the options `sp`, `spmin`, `spmax`, `spstart` and `number`. The smoothing parameters are chosen between `spmin` and `spmax` on a log-scale. Again exclusion of the covariate from the model as well as a linear fit is always included in the list of modeling alternatives. As an example consider the term

```
Y = X1(psplinerw2,sp=direct,spmin=10,spmax=10000,number=10)
```

which defines a list of 10 smoothing parameters on a log-scale between 10 and 10000.

The smoothing parameters of the start model are specified by setting the global option `startmodel=userdefined` and the local option `spstart`.

Zero degree P-spline

Description: Defines a zero degree P-spline with first or second order difference penalty for the effect of `X1`. A zero degree P-spline typically estimates for *every* distinct covariate value in the data set a separate parameter. Usually there is no reason to prefer zero degree P-splines over higher order P-splines. An exception are ordinal covariates or continuous covariates with only a small number of different values. For ordinal covariates higher order P-splines are not meaningful while zero degree P-splines might be an alternative to modeling nonlinear relationships via a dummy approach with completely unrestricted regression parameters.

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

Syntax:

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

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

Example:

Suppose that `X1` is at least ordinal with possibly nonlinear effect. The following model statement defines zero degree P-splines with second order difference penalties for f_1:

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

First order differences are obtained by specifying `X1(rw1)`.

To perform model choice and variable selection an ordered list of smoothing parameters must be defined, see subsection 6.3 in the methodology manual. *BayesX* automatically defines an appropriate list of smoothing parameters, i.e. it is usually not necessary to specify the smoothing parameters. The list of smoothing parameters is specified manually in the same way as for P-splines, see the entry above (page 134).

Seasonal component for time scales

Description: Defines a time-varying 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 by

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

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

To perform model choice and variable selection an ordered list of smoothing parameters must be defined, see subsection 6.3 in the methodology manual. *BayesX* automatically defines an appropriate list of smoothing parameters, i.e. it is usually not necessary to specify the smoothing parameters. The list of smoothing parameters is specified manually in the same way as for P-splines, see the entry above (page 134).

Spatial Covariates

Pairwise difference penalty

Description:

Defines a pairwise difference penalty for the spatial covariate `region`. *BayesX* allows to incorporate spatial covariates with geographical information stored in the *map object* specified in option `map`.

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

Syntax:

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

Example:

For the specification of a pairwise difference penalty, `map` is an obligatory argument that represents the name of a *map object* (see chapter 5) containing 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 pairwise difference penalty for `region` where the geographical information is stored in the *map object* `germany`. An error will be raised if `germany` is not existing.

To perform model choice and variable selection an ordered list of smoothing parameters must be defined, see subsection 6.3 in the methodology manual. *BayesX* automatically defines an appropriate list of smoothing parameters, i.e. it is usually not necessary to specify the smoothing parameters. The list of smoothing parameters is specified manually in the same way as for P-splines, see the entry above (page 134).

Two-dimensional P-spline with first or second order random walk penalty

Description:

Defines a two-dimensional P-spline for the spatial covariate `region` with a two-dimensional first or second order difference penalty for the parameters of the spline. Estimation is based on the coordinates of the centroids of the regions. The centroids are computed using the geographical information stored in the *map object* specified in the option `map`.

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

Syntax:

`region(geosplinerw1,map=`*characterstring*[, *options*]`)`
`region(geosplinerw2,map=`*characterstring*[, *options*]`)`

Example:

For the specification of a two-dimensional P-spline (*geospline*) `map` is an obligatory argument indicating 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(geosplinerw1,map=germany)
```

specifies a two-dimensional cubic P-spline with second order difference penalty where the geographical information is stored in the *map object* `germany`.

To perform model choice and variable selection an ordered list of smoothing parameters must be defined, see subsection 6.3 in the methodology manual. *BayesX* automatically defines an appropriate list of smoothing parameters, i.e. it is usually not necessary to specify the smoothing parameters. The list of smoothing parameters is specified manually in the same way as for P-splines, see the entry above (page 134).

Unordered group indicators

Unit- or cluster specific unstructured effect

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

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

Syntax:

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

Example:

Gaussian i.i.d. random effects allow 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 defined by

```
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 *stepwisereg objects*.

To perform model choice and variable selection an ordered list of smoothing parameters must be defined, see subsection 6.3 in the methodology manual. *BayesX* automatically defines an appropriate list of smoothing parameters, i.e. it is usually not necessary to specify the smoothing parameters. The list of smoothing parameters is specified manually in the same way as for P-splines, see the entry above (page 134).

Varying coefficients with continuous covariates as effect modifier

P-spline with first or second order difference penalty

Description:

Defines a varying coefficient term, where the effect of `X1` varies smoothly over the range of `X2`. For f a P-spline with first or second order difference penalty is assumed.

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 difference penalty is defined as follows:

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

If the effect of a covariate should vary according to different types of effect modifiers, this leads to similar identification problems as in usual additive models. To avoid such problems, option `center` can be specified to request the estimation of centered effects. For example, if both `X2` and `Z2` are assumed to modify the effect of `X1`, the specification of

```
Y = X1*X2(psplinerw2) + X1*Z2(psplinerw2)
```

yields a non-identifiable model. In contrast

```
Y = X1 + X1*X2(psplinerw2, center) + X1*Z2(psplinerw2, center)
```

is well-identified. Note that the main effect of `X1` has to be included separately. Equivalently, we could absorb the main effect into the first term, yielding

```
Y = X1*X2(psplinerw2) + X1*Z2(psplinerw2, center)
```

However, the former specification has the advantage that the model terms are clearly separated.

Models of the type just discussed arise for example if `X1` is a binary dummy-variable indicating two different groups of data. In this case the model

```
Y = X1 + X2(psplinerw2) + X1*X2(psplinerw2, center) + Z2(psplinerw2) +
X1*Z2(psplinerw2, center)
```

assumes different effects of both `X2` and `Z2` in the groups.

To perform model choice and variable selection an ordered list of smoothing parameters must be defined, see subsection 6.3 in the methodology manual. *BayesX* automatically defines an appropriate list of smoothing parameters, i.e. it is usually not necessary to specify the smoothing parameters. The list of smoothing parameters is specified manually in the same way as for P-splines, see the entry above (page 134).

Zero degree P-spline

Description:

Defines a varying coefficient term, where the effect of `X1` varies smoothly over the range of `X2`. Therefore covariate `X2` is called the effect modifier of `X1`. The smoothness of $f(X2)$ is induced by defining a first or second order difference penalty.

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 zero degree P-spline and a second order difference penalty is defined as follows:

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

To perform model choice and variable selection an ordered list of smoothing parameters must be defined, see subsection 6.3 in the methodology manual. *BayesX* automatically defines an appropriate list of smoothing parameters, i.e. it is usually not necessary to specify the smoothing parameters. The list of smoothing parameters is specified manually in the same way as for P-splines, see the entry above (page 134).

Varying coefficients with spatial covariates as effect modifiers

Pairwise difference penalty

Description:

Defines a varying coefficient term where the effect of `X1` varies smoothly over the range of the spatial covariate `region`. A pairwise difference penalty is assumed 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 an unweighted pairwise difference penalty. Weighted difference penalties can be estimated by including an appropriate weight definition when creating the *map object* `germany` (see section 5.1).

Similarly as for varying coefficient terms with continuous effect modifiers, varying coefficients with spatial effect modifier can be centered to avoid identifiability problems:

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

To perform model choice and variable selection an ordered list of smoothing parameters must be defined, see subsection 6.3 in the methodology manual. *BayesX* automatically defines an appropriate list of smoothing parameters, i.e. it is usually not necessary to specify the smoothing parameters. The list of smoothing parameters is specified manually in the same way as for P-splines, see the entry above (page 134).

Two-dimensional P-spline with first or second order difference penalty

Description:

Defines a varying coefficient term where the effect of `X1` varies smoothly over the range of the spatial covariate `region`. A two-dimensional P-spline for the spatial covariate `region` with a two-dimensional first order difference penalty for the parameters of the spline are defined. Estimation is based on the coordinates of the centroids of the regions. The centroids are computed using the geographical information stored in the *map object* specified in the option `map`.

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

Syntax:

`X1*region(geosplinerw1,map=`*characterstring*[, *options*]`)`
`X1*region(geosplinerw2,map=`*characterstring*[, *options*]`)`

Example:

For the specification of a two-dimensional P-spline (*geospline*) `map` is an obligatory argument indicating 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 = X1*region(geosplinerw1,map=germany)
```

specifies a two-dimensional cubic P-spline with first order difference penalty where the geographical information is stored in the *map object* `germany`.

To perform model choice and variable selection an ordered list of smoothing parameters must be defined, see subsection 6.3 in the methodology manual. *BayesX* automatically defines an appropriate list of smoothing parameters, i.e. it is usually not necessary to specify the smoothing parameters. The list of smoothing parameters is specified manually in the same way as for P-splines, see the entry above (page 134).

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 (ridge type penalty) 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*, the main effects are *not* 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

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

would be

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

To perform model choice and variable selection an ordered list of smoothing parameters must be defined, see subsection 6.3 in the methodology manual. *BayesX* automatically defines an appropriate list of smoothing parameters, i.e. it is usually not necessary to specify the smoothing parameters. The list of smoothing parameters is specified manually in the same way as for P-splines, see the entry above (page 134).

Surface estimators

Two-dimensional P-spline with first or second order difference penalty

Description:

Defines a two-dimensional P-spline based on the tensor product of one-dimensional P-splines with a two-dimensional first or second order difference penalty for the parameters of the spline.

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

Syntax:

`X1*X2(pspline2dimrw1`[, *options*]`)`
`X1*X2(pspline2dimrw2`[, *options*]`)`

Example:

The model term

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

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

ANOVA type interaction

Description: In some applications it is desirable to decompose the effect of the two covariates $X1$ and $X2$ into two main effects modeled by one dimensional functions and a two dimensional interaction effect, i.e.

$$f(X1, X2) = f_1(X1) + f_2(X2) + f_{1|2}(X1, X2). \tag{9.1}$$

stepwisereg objects allow such decompositions using the approach described in section subsubsection 3.1.2 of the methodology manual.

Predictor: $\eta = \cdots + f_1(X1) + f_2(X2) + f_{1|2}(X1, X2) + \cdots$

Syntax:

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

Example:

The model term

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

specifies an ANOVA type interaction effect with second order difference penalty for the main effects.

To perform model choice and variable selection an ordered list of smoothing parameters must be defined, see autorefstepwiseest in the methodology manual. *BayesX* automatically defines an appropriate list of smoothing parameters, i.e. it is usually not necessary to specify the smoothing parameters. The list of smoothing parameters is specified manually in the same way as for P-splines, see the entry above (page 134).

9.1.1.3 Description of additional options for terms of stepwisereg 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 commas. All options may be specified in arbitrary order. Table 9.3 provides explanations and the default values of all possible options. All reasonable combinations of model terms and options can be found in Table 9.4.

local option	type	default	values	description
`sp`	string	automatic	automatic df direct	list of smoothing parameters are automatically specified smoothing parameters are directly specified by the user in terms of degrees of freedom use options `dfmin`, `dfmax`, `number`, `dfstart`, `logscale` for specification smoothing parameters are directly specified by the user use options `spmin`, `spmax`, `number`, `spstart`, `logscale` for specification
`dfmin`	numeric (real)	–	see page 134	minimum degree of freedom, see also the entry on P-splines at page 134
`dfmax`	numeric (real)	–	see page 134	maximum degree of freedom
`dfstart`	numeric (real)	1	$\{0, 1\} \cup [dfmin; dfmax]$	degree of freedom used in the start model
`logscale`	boolean	false	false true	equidistant degrees of freedom smoothing parameters on a logarithmic scale
`spmin`	numeric (real)	10^{-4}	$[10^{-6}; 10^{8}]$	minimum smoothing parameter
`spmax`	numeric (real)	10^{4}	$[10^{-6}; 10^{8}]$	maximum smoothing parameter
`spstart`	numeric (real)	–	$\{-1, 0\} \cup [10^{-6}; 10^{8}]$	smoothing parameter for the start model `spstart=0` excludes the term from the start model `spstart=-1` includes a linear effect
`number`	numeric (integer)	0	$\{0; 100\}$	number of different smoothing parameters
`forced_into`	boolean	false	false true	term may be excluded from the model term may not be excluded from the model
`nofixed`	boolean	false	false true	linear fit is allowed linear fit is not allowed
`center`	boolean	false	false true	varying coefficient term is not centered varying coefficient term is centered
`coding`	string	dummy	dummy effect	dummy coding of categorical variables effect coding of categorical variables
`reference`	numeric (real)	1	$(-100; 100)$	specifies the reference category (for categorical covariates)
`degree`	numeric (integer)	3	$\{0; \ldots; 5\}$	degree of B–spline basis functions
`nrknots`	numeric (integer)	20	$\{5; \ldots; 500\}$	number of inner knots for a P–spline term
`monotone`	string	unrestricted	unrestricted increasing decreasing convex concave	no constraint on the spline function monotonically increasing function monotonically decreasing function convex function, i.e. positive second derivative concave function, i.e. negative second derivative
`gridsize`	numeric (integer)	-1	$\{-1; 10; \ldots; 500\}$	May be used to restrict the number of points (on 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.
`period`	numeric (integer)	12	$\{2; \ldots; 72\}$	period for a seasonal effect

Table 9.3: Possible local options for stepwisereg objects. Note, that boolean options are specified without supplying a value.

	factor	rw1 rw2	season	psplinerw1 psplinerw2	spatial	random	geosplinerw1 geosplinerw2	pspline2dimrw1 pspline2dimrw2 psplineinteract
`dfmin`	—	real	real	real	real	real	real	real
`dfmax`	—	real	real	real	real	real	real	real
`dfstart`	integer	real	real	real	real	real	real	real
`logscale`	—	boolean	boolean	boolean	boolean	boolean	boolean	boolean
`sp`	string	string	string	string	string	string	string	string
`spmin`	—	real	real	real	real	real	real	real
`spmax`	—	real	real	real	real	real	real	real
`spstart`	0,1	real	real	real	real	real	real	real
`number`	—	integer	integer	integer	integer	integer	integer	integer
`forced_into`	boolean	boolean	boolean	boolean	boolean	boolean	boolean	boolean
`nofixed`	—-	boolean	—-	boolean	boolean	boolean	boolean	boolean
`center`	—	boolean	—	boolean	boolean	boolean	—	—
`coding`	string	—	—	—	—	—	—	—
`reference`	real	—	—	—	—	—	—	—
`degree`	—	—	—	integer	—	—	integer	integer
`nrknots`	—	—	—	integer	—	—	integer	integer
`monotone`	—	—	—	string	—	—	—	—
`gridsize`	—	—	—	integer	—	—	—	integer
`period`	—	—	integer	—	—	—	—	—
`map`	—	—	—	—	*map object*	—	*map object*	—

Table 9.4: Terms and options for stepwisereg objects. Note, that boolean options are specified without supplying a value.

9.1.1.4 Specifying the response distribution

Supported univariate distributions are Gaussian, binomial (with logit or probit link), Poisson and gamma. For multivariate responses, *stepwisereg* objects currently support only multinomial logit models for categorical responses with unordered categories. Continuous time survival models may be estimated by assuming a piecewise exponential model which is estimated via an equivalent Poisson model. An overview over the supported models is given in Table 9.5. The distribution of the response is specified by adding the additional option `family` to the (global) options list of the regression call. For instance, `family=gaussian` defines the response to be Gaussian. In some cases, one or more additional options associated with the specified response distribution can be specified. An example is the `reference` option for multinomial responses, which defines the reference category. In the following we give details on how to specify the models.

value of `family`	response distribution	link	options
`family=gaussian`	Gaussian	identity	
`family=binomial` `family=binomialprobit`	binomial binomial	logit probit	
`family=multinomial`	unordered multinomial	logit	`reference`
`family=poisson` `family=gamma`	Poisson gamma	log log	

Table 9.5: Summary of supported response distributions.

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 $y|\eta, \sigma^2 \sim N(\eta, \sigma^2 C^{-1})$, where $C = diag(weightvar_1, \ldots, weightvar_n)$ is a known weight matrix. Gaussian regression models are obtained by adding

```
family=gaussian
```

to the options list.

An optional weight variable *weightvar* can be specified to estimate weighted regression models, see subsubsection 9.1.1.1 for details. For grouped Gaussian responses, the weights represent 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 are given by 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 no weight variable is specified, *BayesX* assumes $weightvar_i = 1$, $i = 1, \ldots, n$.

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

$$p(y) \propto y^{s-1} \exp\left(-\frac{s}{\mu} y\right) \tag{9.2}$$

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 parametrization is typically employed for hyperparameters `a` and `b` of priors for variance parameters in the context of Bayesian hierarchical models. In this case, the density is given by

$$p(y) \propto y^{a-1} \exp(-by) \tag{9.3}$$

for $y > 0$. In this parametrization 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 variable is parameterized in the first form (Equation 9.2). 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* can be specified to estimate weighted regression models, see subsubsection 10.1.2.1 for details.

It is possible to assume a fixed scale parameter. The scale parameter is defined to be fixed 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 to be fixed at the value $\nu = 1$.

Binomial logit and probit models

A binomial logit model is requested by the option

```
family=binomial
```

while a probit model is obtained with

```
family=binomialprobit.
```

A additional weight variable may be specified, see subsubsection 9.1.1.1 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, *stepwisereg objects* support only multinomial logit models and no probit models. A multinomial logit model is specified by adding the option

```
family=multinomial
```

to the options list. A second option (`reference`) may be added to the options list to define the reference category. If the response variable has three categories 1, 2 and 3, the reference category can be set to 2, by adding

```
reference=2
```

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

Poisson regression

A Poisson regression model is specified by adding

`family=poisson`

to the options list.

A weight variable may be specified in addition, see subsubsection 9.1.1.1 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.

Piecewise exponential model (p.e.m.)

In subsection 7.2 of the methodology manual we demonstrated how continuous time survival data are manipulated to transform it to a Poisson model for estimation. Suppose that the following modified data set is available

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 7.2 of the methodology manual. Let `x1` be a covariate with linear effect and `x2` a continuous covariate with nonlinear effect. Then the correct syntax for estimating a p.e.m. with a *stepwisereg object* named `s` is e.g. as follows:

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

or

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

Note that a time-varying effect of an additional covariate `X` may be estimated by simply adding the term

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

to the model statement.

9.1.2 Options

Options for controlling the selection algorithms

- `algorithm = stringvalue`
  Specifies the selection algorithm. Possible values are `cdescent1` (adaptive algorithms in the methodology manual, see subsection 6.3), `cdescent2` (adaptive algorithms 1 and 2 with backfitting, see remarks 1 and 2 of section 3 in Belitz & Lang (2008)), `cdescent3` (search according to `cdescent1` followed by `cdescent2` using the selected model in the first step as

the start model) and `stepwise` (stepwise algorithm implemented in the gam routine of S-plus, see Chambers and Hastie, 1991). This option will rarely be specified by the user.
DEFAULT: `algorithm = cdescent1`

- `criterion = stringvalue`
  Specifies the goodness of fit criterion. Possible values are listed in table Table 9.6. If `criterion = MSEP` is specified the data are randomly divided into a test- and validation data set. The test data set is used to estimate the models and the validation data set is used to estimate the mean squared prediction error (MSEP) which serves as the goodness of fit criterion to compare different models. The proportion of data used for the test and validation sample can be specified using option `proportion`, see below. The default is to use 75% of the data for the training sample.
  DEFAULT: `criterion = AIC_imp`

- `proportion = realvalue`
  This option may be used in combination with option `criterion=MSEP`, see above. In this case the data are randomly divided into a training and a validation sample. `proportion` defines the fraction (between 0 and 1) of the original data used as training sample.
  DEFAULT: `proportion = 0.75`

- `startmodel = stringvalue`
  Defines the start model for variable selection. Possible values are listed in table Table 9.6.
  DEFAULT: `startmodel = empty`

- `trace=stringvalue`
  Specifies how detailed the output in the *output window* will be. Possible values are given in table Table 9.6.
  DEFAULT: `trace = trace_half`

- `steps=integervalue`
  Defines the maximum number of iterations. If the selection process has not converged after `steps` iterations the algorithm terminates and a warning is raised. Setting `steps=0` allows the user to estimate a certain model without any model choice. This option will rarely be specified by the user.
  DEFAULT: `steps = 100`

Options for computing confidence intervals

- `CI = stringvalue`
  By default confidence intervals for linear and nonlinear terms are not computed. Option `CI` allows to compute confidence intervals. *stepwisereg objects* provide two alternatives for confidence interval estimation. These are

  – confidence intervals conditional on the selected model, i.e. model uncertainty is not taken into account. The confidence intervals are based on MCMC simulations from the posterior. Pointwise (Bayesian) confidence intervals are simply obtained by computing the respective quantiles of simulated parameters and function evaluations. Conditional confidence intervals are specified by `CI = MCMCselect`.

  – unconditional confidence intervals where model uncertainty is taken into account. The computation is based on bootstrap confidence intervals proposed by Wood (2006b), see the methodology manual for details. Unconditional confidence intervals are specified

global option	type	default	values	description
`algorithm`	string	cdescent1	cdescent1	adaptive search
			cdescent2	exact search
			cdescent3	adaptive/exact search
			stepwise	stepwise algorithm
`criterion`	string	AIC_imp	GCV	Generalized Cross Validation based on deviance residuals, see e.g. Wood (2006a)
			GCVrss	Generalized Cross Validation based on residual sum of squares (for Gaussian responses GCV and GCVrss coincide), see e.g. Wood (2006a)
			AIC	Akaike Information Criterion, see e.g. Burnham & Anderson (1998)
			AIC_imp	improved AIC with bias correction for regression models, see e.g. Burnham & Anderson (1998)
			BIC	Bayesian information criterion, see e.g. Hastie, Tibshirani & Firedman (2001)
			MSEP	Mean Squared Error Prediction
			CV5	5–fold cross validation, see e.g. Hastie, Tibshirani & Firedman (2001)
			CV10	10–fold cross validation, see e.g. Hastie, Tibshirani & Firedman (2001)
			AUC	area under the ROC curve (binary response only)
`proportion`	numeric (real)	0.75	$(0; 1)$	in combination with `criterion=MSEP` (see description above)
`startmodel`	string	linear	linear	start model with degrees of freedom equal to one for model terms
			empty	empty model containing only an intercept
			full	most complex possible model
			userdefined	start model is specified by the user; otherwise the linear one
`trace`	string	trace_on	trace_on	output shows full selection path
			trace_half	output shows only the best model of each iteration
			trace_off	no output except start and final model
`steps`	numeric (integer)	1000	$\{0; 10000\}$	maximum number of iterations

Table 9.6: Global options controlling the selection algorithm.

by `CI = MCMCbootstrap`. The number of bootstrap samples is specified using option `bootstrapsamples`, see below.

Both alternatives are computer intensive. Conditional confidence intervals take much less computing time than unconditional intervals. The advantage of unconditional confidence intervals is that sampling distributions for the degrees of freedom or smoothing parameters are obtained.
DEFAULT: CI = none

- `bootstrapsamples = integervalue`
  Defines the number of bootstrap samples used for `CI=MCMCbootstrap`.
  DEFAULT: bootstrapsamples=99

- `iterations=integervalue`
  Defines the number of MCMC iterations used for `CI=MCMCselect` or `CI=MCMCbootstrap`. With `CI=MCMCbootstrap`, option `iterations` specifies the total number of iterations, i.e. the sum of iterations used for the individual conditional MCMC estimations. The number of `iterations` are then divided equally between the individual conditional estimations so that the number of iterations used for one model is `iterations / (bootstrapsamples + 1)`.

Typically 99 bootstrap samples (plus the original data set yields 100) are used to approximate the sampling distribution of the smoothing parameters. If we wish for every bootstrap replication 200 samples from the posterior `iterations=20000` is required.
DEFAULT: `iterations=20000`

- `step=integervalue`
  Defines the thinning parameter for MCMC simulation with `CI=MCMCselect` or `CI=MCMCbootstrap`. For example, `step = 20` means, that only every 20th sampled parameter will be stored and used to compute characteristics of the posterior distribution. The aim of thinning is to reach a considerable reduction of disk storing and computing time.
  DEFAULT: `step=20`

- `burnin = integervalue`
  Defines the number of MCMC iterations used for the burn–in iterations at the beginning of each conditional MCMC estimation. Usually a certain number of burn–in iterations are required to achieve convergence of the Markov chain towards its stationary (i.e. the posterior) distribution. In our case, the initial estimates for each conditional MCMC estimation are the posterior mode estimates, i.e. the Markov chain already starts in its stationary distribution. Hence, burn–in iterations are not necessarily needed here and we can define `burnin=0` which saves considerable computing time. Anyway, specifying this option is meaningful only in combination with `CI=MCMCbootstrap` or `CI=MCMCselect`.
  DEFAULT: `burnin=0`

- `level1 =` *integer*
  By default, *BayesX* computes confidence 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 leads to the computation of confidence intervals for a nominal level of 99% rather than 95%.
  DEFAULT: `level1 =` *95*

- `level2 =` *integer*
  By default, *BayesX* computes 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 leads to the computation of credible intervals for a nominal level of 70% rather than 80%.
  DEFAULT: `level2 =` *80*

Further options

- `family =` *stringvalue*
  Specifies the response distribution and link function, see subsubsection 9.1.1.4 for details.
  DEFAULT: `family =` *binomial*

- `predict`
  By specifying `predict` an additional file with ending `predictmean.raw` is created that contains for every observation the estimated predictor $\hat{\eta}_i$ and expectation $\hat{E}(y_i|\eta_i) = \hat{\mu}_i$ as well as the deviance D_i. If bootstrap replications are available (see option `CI` for details) model averaged estimates for η_i and μ_i are additionally computed.

- `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.
  DEFAULT: `reference = 1`

9.1.3 Estimation output

The estimation output depends on the estimated model. Estimation results for linear effects are displayed in a tabular form in the *output window* and/or in a log file (if created before). This table always contains the estimated coefficient. Standard deviations and 95% confidence intervals are available only if option `CI =` *MCMCbootstrap* or `CI =` *MCMCselect* have been specified. A different confidence level may be obtained by specifying the `level1` option, see section 9.1.2 for details. Additionally, a file replicating results for the fixed effects is created. The name of this file is supplied in the *output window* and/or in a log file.

Estimated nonparametric effects are presented in a different way. Here, results are stored in external ASCII-files that can be read into any general purpose statistics program (e.g. STATA, R, 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 names of the files and the storing directory are displayed in the *output window* and/or a log file. The files contain ten columns (for main effects) or eleven columns (for interaction effects). The first column contains a parameter index (starting with one), the second column (and the third column if the estimated effect is an interaction) contain the values of the covariate(s) whose effect has been estimated. In the following columns the estimation results are given in form of the point estimate, the lower boundaries of the 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 while 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 section 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	pmean	pqu2p5	pqu10	pmed	pqu90	pq97p5	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 credible intervals, standard deviations etc. are available only if option `CI =` *MCMCbootstrap* or `CI =` *MCMCselect* have been specified.

The estimated nonlinear effects can be visualized using either the graphics capabilities of *BayesX* or a couple of R functions, see section 11.1 and section 11.2, respectively. Of course, any other (statistics) software package with plotting facilities can 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.

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 chapter 3 of the tutorial manual.

Suppose that we have a data set `test` with a continuous response variable `y`, and covariates `x1`, `x2`, `x3`, and `region`, where `region` indicates the geographical location an observation belongs to. Suppose further that we have already created a *stepwisereg object* `s`.

Linear 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:

```
> s.regress y = x1 + x2 + x3, family=gaussian using test
```

By specifying option `family=gaussian`, a linear model with Gaussian errors is estimated and relevant variables are selected.

Suppose now that covariate `x1` is categorical with three categories 1,2 and 3. In this case `x1` should be incorporated as a factor variable. We obtain

```
> s.regress y = x1(factor,reference=2) + x2 + x3, family=gaussian using test
```

where the reference category for x1 is 2. *BayesX* automatically creates dummy variables for `x1`. If effect coding should be used instead of dummy coding we have to write

```
> s.regress y = x1(factor,reference=2,coding=effect) + x2 + x3, family=gaussian
 using test
```

Additive models

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

```
> s.regress y = x1(factor,reference=2) + x2(psplinerw2) + x3(psplinerw2),
 family=gaussian using test
```

which corresponds to the predictor

$$\eta = \gamma_0 + \gamma_1 x1\_1 + \gamma_2 x1\_3 + f_1(x2) + f_2(x3),$$

where `x1_1` and `x1_3` are dummy variables for `x1`

Spatial covariates

To incorporate a structured spatial effect, we first have to create a *map object*. Afterwards we read 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 augment our predictor with a spatial effect:

```
> s.regress y = x1(factor,referenc=2+ x2(psplinerw2) + x3(psplinerw2)+
  region(spatial,map=m), family=gaussian using test
```

9.2 Global options

The purpose of global options is to affect the global behavior of a *stepwisereg 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 *stepwisereg 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 *stepwisereg object* and the type of the file. For example, if you estimated a nonparametric effect for a covariate `X`, say, using a P-spline, then the estimation output will be written to

  <*INSTALLDIRECTORY*>`\output\r_f_X_pspline.res`

  where `r` is the name of the *stepwisereg 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 achieved using the `outfile` option. Here, you have to specify the directory where the output should be stored and a base file name. This base file name should not be a complete file name. For example specifying

  ```
  outfile = c:\data\res1
  ```

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

  ```
  c:\data\res1_f_X_pspline.res
  ```

9.3 Visualizing estimation results

Visualization of estimation results is described in chapter 11

Chapter 10

mcmcreg objects

mcmcreg objects are used to fit Bayesian quantile regression and (multivariate) distributional regression models with *structured additive predictor*, see Klein, Kneib & Lang (2014). Hierarchical data structures may be considered using hierarchical or multilevel structured additive predictors. For multilevel structured additive models see Lang et al. (2014). Inference is based on a fully Bayesian approach implemented via Markov Chain Monte Carlo (MCMC) simulation techniques. The methodology manual provides a brief introduction to (multilevel) structured additive regression and MCMC-based inference. More details can be found in the references cited above and in the book by Fahrmeir et al. (2013).

10.1 Method hregress

10.1.1 Description

Method `hregress` estimates (hierarchical) distributional structured additive regression models.

10.1.2 Syntax

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

Method `hregress` estimates the regression model specified in *model* using the data specified in *dataset*. *dataset* has to be the name of a *dataset object* created before. The details of correct models are covered in subsubsection 10.1.2.2. The distribution of the response variable can be chosen from a wide range of uni- and multivariate distributions. It is specified using options `family` and `equationtype`, see subsubsection 10.1.2.4 below and the options list in subsection 10.1.3. The default value is `family=gaussian` and `equationtype=mu` with an identity link. An `if` statement may be specified to analyze only parts of the data set, i.e. the observations where *expression* is true.

10.1.2.1 Optional weight variable

An optional weight variable *weightvar* may be specified to estimate weighted regression models.

10.1.2.2 Syntax of possible model terms

The general syntax of models is:

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

where *depvar* specifies the dependent variable in the model and $term_1$,...,$term_r$ define the specific form of covariate effects on the dependent variable. The different terms have to be separated by '+' signs. A constant intercept is (in contrast to *bayesreg objects*) NOT automatically included in the models and must be specified by the analyst using the term `const`. This section reviews all possible model terms that are currently supported by *mcmcreg objects* and provides some specific examples. Note that all described terms may be combined in arbitrary order. An overview about the capabilities of *mcmcreg objects* is given in Table 10.1. Table 10.2 shows how interactions between covariates are specified. Full details about all available options are given in subsubsection 10.1.2.3.

Throughout this section, `Y` will denote the dependent variable.

Offset

Description: Adds an offset term to the predictor.

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

Syntax: `offs(offset)`

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
```

Note that the offset variable is included on the level of the predictor.

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 specifies a model with q fixed (linear) effects and an intercept:

```
Y = const + W1 + W2 + ··· + Wq
```

Shrinkage of fixed effects

Description: Defines a shrinkage prior for the corresponding parameters γ_j, $j = 1, \ldots, q$, $q \geq 1$ of the linear effects `X1`,...,`Xq`. There are two priors possible: ridge and lasso type priors.

Predictor: $\eta = \cdots + \gamma_1 X1 + \cdots + \gamma_q Xq + \cdots$

Syntax:

- Ridge-prior: `X1(ridge`[, *options*]`)`
- Lasso-prior: `X1(lasso`[, *options*]`)`

Example: The following model statement can be used to estimate a model with q lasso-penalized linear effects

```
Y = X1(lasso)+...+ Xq(lasso)
```

By default, the starting value of the shrinkage parameter in the Markov chain is set to 1 and the shrinkage parameter is estimated by the data. It is also possible to fix the shrinkage parameter through the iterations in order to use a prespecified amount of shrinkage. To do so, the option `shrinkagefix` has to be set in the corresponding terms and this results in fixing the shrinkage parameter at the starting value assigned in the option `shrinkage`.

The following model term defines a lasso penalty with shrinkage parameter fixed at the value 1.5:

```
Y = X1(lasso)+...+ Xq(lasso,shrinkage=1.5,shrinkagefix)
```

Full details about all possible options for shrinkage effects are given in Section 7.1.2.3.

Important Remark: Except for the option `tau2` for the variances of lasso and ridge (and the options `I` and `t2` for nigmix), all the other possible options used in the shrinkage methods are those which are specified in the first term of the corresponding penalty, e.g.

```
Y = X2(lasso,shrinkagepar=2,shrinkagefix)+ X1(lasso,shrinkagepar=1.5)
```

uses the options of `X2`. If the option `adaptive` is specified the options from each term are used.

Nonlinear effects of continuous covariates and time scales

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

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

Syntax: `X1(pspline[, options])`

Example:

A P-spline with second order random walk prior is obtained by:

```
Y = X1(pspline)
```

By default, a second order random walk prior is used, 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 and a first order random walk prior:

```
Y = X1(pspline,degree=2,nrknots=30,difforder=1)
```

Full details about all possible options for P-splines are given in subsubsection 10.1.2.3.

Spatial Covariates

Description:

Defines a Markov random field prior for the spatial covariate `region`. *BayesX* allows to incorporate spatial covariates with geographical information stored in the *map object* specified in option `map`.

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

Syntax:

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

Example:

For the specification of a Markov random field prior, `map` is an obligatory argument that represents the name of a *map object* (see chapter 5) containing 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 prior to estimation of a spatial effect to obtain a band matrix like precision matrix. This can be achieved using method `reorder` of *map objects*, see section 5.3 for details.

Unordered group indicators

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

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

Syntax:

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

Example:

Gaussian i.i.d. random effects allow 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 defined by

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

Note that *BayesX* allows the specification of hierarchical or multilevel models that go far beyond the simple random intercept term described above, see subsubsection 10.1.2.5.

Varying coefficients with continuous covariates as effect modifier

Description:

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

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

Syntax:

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

Example:

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

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

Varying coefficients with spatial covariates as effect modifiers

Description:

Defines a varying coefficients 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}(`region`). The geographical information is assumed to be stored in the *map object* specified in the option `map`.

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

Syntax:

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

Example:

The statement

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

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

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

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 $f(grvar)$.

Syntax:

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

Example:

A random slope is specified as follows:

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

Note that *BayesX* allows the specification of hierarchical or multilevel models that go far beyond the simple random intercept term described above, see subsubsection 10.1.2.5.

Surface estimators

Two-dimensional kriging term

Description:

Defines a two dimensional Gaussian field (kriging term)

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

Syntax:

`X1*X2(kriging`[, *further options*]`)`

Example:

The model term

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

specifies a two-dimensional Gaussian field.

10.1.2.3 Description of additional options for terms of mcmcreg objects

All arguments described in this section are optional and can therefore be omitted. Generally, all options are specified by adding the option name to the specification of the model term type in the parentheses, separated by commas. Boolean options are specified by simply adding the option name. For example, a random intercept term with `a=b=0.001` as parameters for the inverse gamma prior of the variance parameter is specified as follows:

```
X1*grvar(random,a_re=0.001,b_re=0.001)
```

Note that all options may be specified in arbitrary order. Table 10.3 and Table 10.4 provide explanations and the default values of all possible options. All reasonable combinations of model terms and options can be found in Table 10.5.

Type	Syntax example	Description
Offset	`offs(offset)`	Variable `offs` is an offset term.
Linear effect	`W1`	Linear effect of `W1`.
Ridge effect	`X1(ridge)`	Linear effect of `X1` with ridge prior.
Lasso effect	`X1(lasso)`	Linear effect of `X1` with lasso prior.
P-spline	`X1(pspline)`	Nonlinear effect of `X1`.
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 are stored in the *map object* `m`.
Two dimensional kriging term	`region(geospline,map=m)`	Spatial effect of `region`. Estimates a two dimensional kriging term based on the centroids of the regions. The centroids are obtained from the *map object* `m`.
Random intercept	`grvar(random)`	I.i.d. Gaussian (random) effect of the group indicator `grvar`, e.g. `grvar` may be an individual indicator when analyzing longitudinal data.

Table 10.1: Overview over different model terms for mcmcreg objects.

Type of interaction	Syntax example	Description
Varying coefficient term	`X1*X2(pspline)`	Effect of `X1` varies smoothly over the range of the continuous covariate `X2` or `time`.
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 kriging term	`X1*X2(kriging)`	Two dimensional surface for the continuous covariates `X1` and `X2`.

Table 10.2: Possible interaction terms for mcmcreg objects.

Option	Description	Default
a, b	The options a and b specify the hyperparameters of the inverse Gamma prior for the variance τ^2 of nonlinear effects (e.g. P-splines or Markov random fields).	a=0.001, b=0.001
a_re, b_re	The options a_re and b_re specify the hyperparameters of the inverse Gamma prior for the variance τ^2 of random effects (random or hrandom).	a_re=0.001, b_re=0.001
binning	uses binning of the covariate according to Lang et al. (2014).	binning=-1 (no binning)
centermethod	Defines the method for centering nonlinear terms (e.g. P-splines). centermethods = meanf Centered sampling such that the sum of the $f(x)$ over *all* observations is zero. centermethods = meanfd Centered sampling such that the sum of the $f(x)$ over the *distinct* observations is zero. centermethods = meancoeff Centered sampling such that the sum of the regression coefficients is zero. centermethods = meansimple Center the parameters around zero (after they have been sampled).	centermethod=meanfd
constraints	Defines monotonicity constraints for P-splines. Specifying constraints=increasing yields increasing nonlinear functions and constraints=decreasing yields decreasing functions.	constraints = unrestricted
degree	Specifies the degree of B-spline basis functions for P-splines.	degree=3
derivative	If specified, first order derivatives of the function estimate are computed (for P-splines only).	-
difforder	Specifies the difference order (1 or 2) of random walks for P-spline priors.	2
lambda	Provides a starting value for the smoothing parameter λ. NOTE: In distributional regression models with more than one equation it may be better to specifiy lambda=100.	lambda=10
lambda_re	Provides a starting value for the smoothing parameter λ for random effects (random or hrandom).	lambda_re=0.1
nocenter	Indicates that a nonlinear term should not be centered.	-
nu	Specifies the parameter nu of the Matern family of covariance functions (kriging terms only).	nu=1.5
nrknots	Specifies the number of inner knots for a P-spline term.	nrknots=20
meaneffect	Indicates that mean effects (expected values) of the response in dependence of the covariate (with other covariates held fixed at mean values) should be computed.	-
round	Rounds the covariate before estimation, e.g. round=2 rounds to 2 digits after the decimal point.	round=-1 (no rounding)

Table 10.3: Optional arguments for mcmcreg object terms in alphabetical order (1).

Option	Description	Default
samplederivative	Indicates that samples of the derivatives should be stored. If this is the case, credible intervals, standard errors etc. for derivatives are computed in addition to mean estimates. This option is only meaningful in combination with option derivative.	-
samplef	Indicates that samples for the nonlinear functions in addition to samples of regression coefficients should be stored.	-
updatem	Specifies the method for updating regression coefficients in the MCMC sampler. update=direct uses direct updating by sampling from the full conditionals in case of gaussian responses or by IWLS proposals in case of non-gaussian responses , update=orthogonal uses orthogonal bases as described in Lang et al. (2014).	updatem=direct

Table 10.4: Optional arguments for mcmcreg object terms in alphabetical order (2).

	pspline	spatial	random	kriging	geokriging
`a`	realvalue	realvalue	×	realvalue	realvalue
`b`	realvalue	realvalue	×	realvalue	realvalue
`a_re`	×	×	realvalue	×	×
`b_re`	×	×	realvalue	×	×
`binning`	-1, integervalue	×	×	×	×
`centermethod`	∗	∗	×	×	×
`constraints`	•	×	×	×	×
`degree`	integer	×	×	×	×
`derivative`	△	×	×	×	×
`lambda`	realvalue	realvalue	realvalue	realvalue	realvalue
`nrknots`	integer	×	×	×	×
`nocenter`	△	△	×	×	×
`nu`	×	×	×	0.5,1.5,2.5,3.5	0.5,1.5,2.5,3.5
`map`	×	*map object*	×	×	*map object*
`maxdist`	×	×	×	realvalue	realvalue
`meaneffect`	△	△	△	△	△
`round`	realvalue	×	×	×	×
`sampledderivative`	△	×	×	×	×
`samplef`	△	△	△	△	△
`updatem`	○	○	○	○	○
×	not available				
∗	admissible values are `meanfd,meanf,meanfd,meancoeff,meansimple`				
○	admissible values are `direct,orthogonal`				
•	admissible values are `unrestricted,increasing,decreasing`				
△	available as boolean option (specified without supplying a value)				

Table 10.5: Terms and options for mcmcreg objects.

10.1.2.4 Specifying the response distribution

An overview of supported univariate distributions is given in Tables 10.6, 10.7 and 10.8. Supported multivariate distributions are presented in Table 10.9. *mcmcreg objects* allow to define for each parameter of a specific distribution a full STAR predictor. This is done by defining for each parameter of the distribution a separate model equation. To define the equation type, the user has to specify the (global) options `family` and `equationtype`. In some cases, one or more additional options associated with the specified response distribution can be specified. In the following, we give detailed instructions on how to specify the various models for two distributions, the Gaussian and Gamma distribution. For the other distributions the specification is analogous.

Gaussian responses

The classical Gaussian regression model is specified with homoscedastic variances, i.e. the responses y_i are Gaussian with expected value $\mu = \eta_i$ depending on covariates and homoscedastic variance σ^2 not depending on covariates. The homoscedastic Gaussian model is specified as

```
b.hregress depvar = const + term1 + ... , family=normal equationtype=mu using d
```

where `depvar` is the response variable, `term1` is a specific covariate effect to be specified as outlined in the previous section and `d` is the name of the dataset object where the data are stored. The dots indicate that there might be other covariate terms in the equation. Since we are dealing with homoscedastic models where only the mean μ depends on covariates the specification of the `equationtype` as done above is not necessary.

A Gaussian regression model with heteroscedastic variances σ_i^2 depending on covariates is specified as:

```
b.hregress depvar = const + s_term1 + ..., family=normal equationtype=sigma2
using d
```

```
b.hregress depvar = const + m_term1 + ..., family=normal equationtype=mu using d
```

The covariates and the respective terms can be completely different in the variance and mean equation. It is also possible to model the standard deviation σ rather than the variance σ^2 using the syntax:

```
b.hregress depvar = const + s_term1 + ..., family=normal2 equationtype=sigma
 using d
```

```
b.hregress depvar = const + m_term1 + ..., family=normal2 equationtype=mu using d
```

Note that we have to specify `family=normal2` rather than `family=normal` when using the standard deviation.

Gamma distributed responses

Gamma distributed responses are specified with the following two equations, one for the parameter σ and one for the expected value μ:

```
b.hregress depvar = const + s_term1 + ..., family=gamma equationtype=sigma using d
```

```
b.hregress depvar = const + m_term1 + ..., family=gamma equationtype=mu using d
```

A model with constant parameter σ not depending on covariates is given by:

```
b.hregress depvar = const  , family=gamma equationtype=sigma using d
```

```
b.hregress depvar = const + m_term1 + ... , family=gamma equationtype=mu using d
```

1. Continuous distributions on $\mathbb{R}$	Density	Parameter	family	equationtype
Normal	$p(y\|\mu,\sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(y-\mu)^2}{2\sigma^2}\right)$	$\sigma^2 > 0$	normal	sigma2
		$\mu \in \mathbb{R}$	normal	mu
Normal	$p(y\|\mu,\sigma) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(y-\mu)^2}{2\sigma^2}\right)$	$\sigma > 0$	normal2	sigma
		$\mu \in \mathbb{R}$	normal2	mu
t	$p(y\|\mu,\sigma^2,df) = \frac{\Gamma((df+1)/2)}{\Gamma(1/2)\Gamma(df/2)\sqrt{df\sigma^2}} \left(1+\frac{(y-\mu)^2}{df\sigma^2}\right)^{-\frac{df+1}{2}}$	$df > 0$	t	df
		$\sigma^2 > 0$	t	sigma2
		$\mu \in \mathbb{R}$	t	mu
2. Continuous distributions on $\mathbb{R}^+$				
Log-normal	$p(y\|\mu,\sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2 y}} \exp\left(-\frac{(\log(y)-\mu)^2}{2\sigma^2}\right)$	$\sigma^2 > 0$	lognormal	sigma2
		$\mu \in \mathbb{R}$	lognormal	mu
Log-normal	$p(y\|\mu,\sigma) = \frac{1}{\sqrt{2\pi\sigma^2 y}} \exp\left(-\frac{(\log(y)-\mu)^2}{2\sigma^2}\right)$	$\sigma > 0$	lognormal2	sigma
		$\mu \in \mathbb{R}$	lognormal2	mu
Inverse Gaussian	$p(y\|\mu,\sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2 y^{3/2}}} \exp\left(-\frac{(y-\mu)^2}{2y\mu^2\sigma^2}\right)$	$\sigma^2 > 0$	invgaussian	sigma2
		$\mu > 0$	invgaussian	mu
Gamma	$p(y\|\mu,\sigma) = \left(\frac{\sigma}{\mu}\right)^{\sigma} \frac{y^{\sigma-1}}{\Gamma(\sigma)} \exp\left(-\frac{\sigma}{\mu}y\right)$	$\sigma > 0$	gamma	sigma
		$\mu > 0$	gamma	mu
Weibull	$p(y\|\lambda,\alpha) = \alpha\lambda^{\alpha} y^{\alpha-1} \exp\left(-\left(\lambda y\right)^{\alpha}\right)$	$\alpha > 0$	weibull	alpha
		$\lambda > 0$	weibull	lambda
Pareto	$p(y\|b,p) = pb^p(y+p)^{-p-1}$	$p > 0$	pareto	p
		$b > 0$	pareto	b
Truncated normal	$p(y\|\mu,\sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(y-\mu)^2}{2\sigma^2}\right) \frac{1}{\sigma(-\Phi(-\mu/\sigma))}$	$\sigma^2 > 0$	truncnormal	sigma2
		$\mu \in \mathbb{R}$	truncnormal	mu
Truncated normal	$p(y\|\mu,\sigma) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(y-\mu)^2}{2\sigma^2}\right) \frac{1}{\sigma(-\Phi(-\mu/\sigma))}$	$\sigma > 0$	truncnormal2	sigma
		$\mu \in \mathbb{R}$	truncnormal2	mu
Generalized gamma	$p(y\|\mu,\sigma,\tau) = \left(\frac{\sigma}{\mu}\right)^{\sigma\tau} \frac{\tau y^{\sigma\tau-1}}{\Gamma(\sigma)} \exp\left(-\left(\frac{\sigma}{\mu}y\right)^{\tau}\right)$	$\tau > 0$	gengamma	tau
		$\sigma > 0$	gengamma	sigma
		$\mu > 0$	gengamma	mu
Dagum	$p(y\|a,b,p) = \frac{apy^{ap-1}}{b^{ap}(1+(y/b)^a)^{p+1}}$	$p > 0$	dagum	p
		$b > 0$	dagum	b
		$a > 0$	dagum	a

Table 10.6: List of possible response distributions in distributional regression.

3. Discrete distributions	Density	Parameter	`family`	`equationtype`
Poisson	$p(y\|\lambda) = \frac{\lambda^y \exp(-\lambda)}{y!}$	$\lambda > 0$	`poisson`	`lambda`
Negative binomial	$p(y\|\mu,\delta) = \frac{\Gamma(y+\delta)}{\Gamma(y+1)\Gamma(\delta)} \left(\frac{\delta}{\delta+\mu}\right)^\delta \left(\frac{\mu}{\delta+\mu}\right)^y$	$\delta > 0$	`negbin`	`delta`
		$\mu > 0$	`negbin`	`mu`
Zero-inflated Poisson	$p(y\|\lambda,\pi) = \pi \mathbb{1}_{\{0\}}(y) + (1-\pi)\frac{\lambda^y \exp(-\lambda)}{y!}$	$\pi \in (0,1)$	`zip`	`pi`
		$\lambda > 0$	`zip`	`lambda`
Zero-inflated negative binomial	$p(y\|\pi,\mu,\delta) = \pi \mathbb{1}_{\{0\}}(y) + \frac{(1-\pi)\Gamma(y+\delta)}{\Gamma(y+1)\Gamma(\delta)} \left(\frac{\delta}{\delta+\mu}\right)^\delta \left(\frac{\mu}{\delta+\mu}\right)^y$	$\delta > 0$	`zinb`	`delta`
		$\pi \in (0,1)$	`zinb`	`pi`
		$\mu > 0$	`zinb`	`mu`
Hurdle Poisson	$p(y\|\lambda,\pi) = \begin{cases} \pi & y = 0 \\ \frac{(1-\pi)}{1-\exp(-\lambda)} \frac{\lambda^y \exp(-\lambda)}{y!} & y > 0 \end{cases}$	$\pi \in (0,1)$	`hurdle`	`pi`
		$\lambda > 0$	`hurdle`	`delta`
				`lambda`
Hurdle negative binomial	$p(y\|\mu,\delta,\pi) = \begin{cases} \pi & y = 0 \\ \frac{(1-\pi)}{1-\left(\frac{\delta}{\delta+\mu}\right)^\delta} \frac{\Gamma(y+\delta)}{\Gamma(y+1)\Gamma(\delta)} \left(\frac{\delta}{\delta+\mu}\right)^\delta \left(\frac{\mu}{\delta+\mu}\right)^y & y > 0 \end{cases}$	$\pi \in (0,1)$	`hurdle`	`pi`
		$\delta > 0$	`hurdle`	`delta`
		$\mu > 0$	`hurdle`	`mu`
Binomial	$p(y\|\pi) = \begin{cases} \pi & y = 0 \\ (1-\pi) & y = 1 \end{cases}$	$\pi \in (0,1)$	`binomial`	`logit`
Cloglog	$p(y\|\pi) = \begin{cases} \pi & y = 0 \\ (1-\pi) & y = 1 \end{cases}$	$\pi \in (0,1)$	`binomialglog`	`cloglog`

Table 10.7: List of possible response distributions.

4. Mixed discrete-continuous distributions					
Zero-adjusted	$p(y\|\pi, g(y)) = \begin{cases} 1-\pi & y = 0 \\ \pi g(y) & y > 0 \end{cases}$ $g(y)$ a distribution from 2.			`zeroadjusted`	
5. Distributions with compact support					
Beta	$p(y\|\mu, \sigma^2) = \frac{y^{p-1}(1-y)^{q-1}}{\mathrm{B}(p,q)}$		$\sigma^2 \in (0,1)$	`beta`	`sigma2`
	$\mu = \frac{p}{p+q}$, $\sigma^2 = \frac{1}{p+q+1}$		$\mu \in (0,1)$	`beta`	`mu`

Zero-One-inflated Beta	$p(y\|\mu, \sigma^2, \upsilon, \tau) = \begin{cases} \frac{\upsilon}{1+\upsilon+\tau} & y = 0 \\ \left(1 - \frac{\upsilon+\tau}{1+\upsilon+\tau}\right) \frac{y^{p-1}(1-y)^{q-1}}{\mathrm{B}(p,q)} & y \in (0,1) \\ \frac{\tau}{1+\upsilon+\tau} & y = 1 \end{cases}$		$\sigma^2 \in (0,1)$
			$\mu \in (0,1)$
			$upsilon > 0$
			$\tau > 0$
				`betainf`
				`betainf`
				`betainf`
				`betainf`
					`sigma2`
					`mu`
					`nu`
					`tau`

Zero-inflated Beta	$p(y\|\mu, \sigma^2, \upsilon) = \begin{cases} \frac{\upsilon}{1+\upsilon} & y = 0 \\ \left(1 - \frac{\upsilon}{1+\upsilon}\right) \frac{y^{p-1}(1-y)^{q-1}}{\mathrm{B}(p,q)} & y \in (0,1) \end{cases}$		$\sigma^2 \in (0,1)$
			$\mu \in (0,1)$
			$upsilon > 0$
				`betainf0`
				`betainf0`
				`betainf0`
					`sigma2`
					`mu`
					`nu`

One-inflated Beta	$p(y\|\mu, \sigma^2, \tau) = \begin{cases} \left(1 - \frac{\tau}{1+\tau}\right) \frac{y^{p-1}(1-y)^{q-1}}{\mathrm{B}(p,q)} & y \in (0,1) \\ \frac{\tau}{1+\tau} & y = 1 \end{cases}$		$\sigma^2 \in (0,1)$
			$\mu \in (0,1)$
			$\tau > 0$
				`betainf1`
				`betainf1`
				`betainf1`
					`sigma2`
					`mu`
					`tau`

Table 10.8: List of possible response distributions.

6. Multivariate distributions	Density	Parameter	`family`	`equationtype`
Bivariate normal	$p(y_1, y_2) = \frac{1}{\sqrt{(2\pi)^2 \det(\boldsymbol{\Sigma})}} \exp\left(-\frac{1}{2}(\mathbf{y}-\boldsymbol{\mu})' \boldsymbol{\Sigma}^{-1}(\mathbf{y}-\boldsymbol{\mu})\right)$	$\rho \in [-1,1]$	`bivnormal`	`rho`
	$\mathbf{y} = (y_1, y_2)'$	$\sigma_2 > 0$	`bivnormal`	`sigma`
	$\boldsymbol{\mu} = (\mu_1, \mu_2)'$	$\sigma_1 > 0$	`bivnormal`	`sigma`
	$\boldsymbol{\Sigma} = \begin{pmatrix} \sigma_1^2 & \rho\sigma_1\sigma_2 \\ \rho\sigma_1\sigma_2 & \sigma_2^2 \end{pmatrix}$	$\mu_2 > 0$	`bivnormal`	`mu`
		$\mu_1 > 0$	`bivnormal`	`mu`
Bivariate normal (Fishers-z)	$p(y_1, y_2) = \frac{1}{\sqrt{(2\pi)^2 \det(\boldsymbol{\Sigma})}} \exp\left(-\frac{1}{2}(\mathbf{y}-\boldsymbol{\mu})' \boldsymbol{\Sigma}^{-1}(\mathbf{y}-\boldsymbol{\mu})\right)$	$\rho \in [-1,1]$	`bivnormal_fz`	`rho`
		$\sigma_2 > 0$	`bivnormal_fz`	`sigma`
		$\sigma_1 > 0$	`bivnormal_fz`	`sigma`
		$\mu_2 > 0$	`bivnormal_fz`	`mu`
		$\mu_1 > 0$	`bivnormal_fz`	`mu`
Bivariate t	$f(y_1, y_2) = \frac{\Gamma\left(\frac{df+D}{2}\right)}{\Gamma\left(\frac{df}{2}\right)(df\,\pi)} \left(\det(\boldsymbol{\Sigma})\right)^{-\frac{1}{2}} \left[1 + (\mathbf{y}-\boldsymbol{\mu})' \boldsymbol{\Sigma}^{-1}(\mathbf{y}-\boldsymbol{\mu})\right]^{-\frac{df+2}{2}}$	$df > 0$	`bivt`	`df`
		$\rho \in [-1,1]$	`bivt`	`rho`
		$\sigma_2 > 0$	`bivt`	`sigma`
		$\sigma_1 > 0$	`bivt`	`sigma`
		$\mu_2 > 0$	`bivt`	`mu`
		$\mu_1 > 0$	`bivt`	`mu`
Bivariate probit	$\mathbf{y}^* = \boldsymbol{\eta}^{\mu} + \boldsymbol{\epsilon}, \qquad \boldsymbol{\epsilon} \sim \mathrm{N}(\mathbf{0}, \boldsymbol{\Sigma}),\ \boldsymbol{\Sigma} = \begin{pmatrix} 1 & \rho \\ \rho & 1 \end{pmatrix}$	$\rho \in [-1,1]$	`bivprobit`	`rho`
	$y_d = 1 \Longleftrightarrow y_d^* > 0$	$\mu_2 > 0$	`bivprobit`	`mu`
	$d = 1, 2$	$\mu_1 > 0$	`bivprobit`	`mu`
Bivariate logit	$p(y\|p_1, p_2, \psi) = \begin{cases} p_{11} = \psi p_{01} p_{10} / p_{00} & y_1 = y_2 = 1 \\ p_{10} = p_1 - p_{11} & y_1 = y_2 = 0 \\ p_{01} = p_2 - p_{11} & y_0 = y_2 = 1 \\ p_{00} = 1 - p_{10} - p_{01} - p_{11} & y_0 = y_2 = 0 \end{cases}$	$\psi > 0$	`bivlogit`	`oddsratio`
		$p_2 \in (0,1)$	`bivlogit`	`mu`
		$p_1 > 0$	`bivlogit`	`mu`
Dirichlet	$p(y_1, \ldots, y_{D-1}) = \frac{1}{\mathrm{B}(\boldsymbol{\alpha})} \prod_{d=1}^{D} y_d^{\alpha_d - 1},\ \mathrm{B}(\boldsymbol{\alpha}) = \frac{\prod_{d=1}^{D} \Gamma(\alpha_d)}{\Gamma\left(\sum_{d=1}^{D} \alpha_d\right)}$	$\alpha_d > 0$	`dirichlet`	`alpha`
	$\sum_{d=1}^{D} y_d = 1$	$D \geq 2$		

Table 10.9: List of possible response distributions.

10.1.2.5 Hierarchical or multilevel models

Random intercept models

As already mentioned at various places in the manuals, BayesX is able to specify and estimate hierarchical or multilevel models, see section 8.3 in the methodology manual and in particular the paper by Lang et al. (2014). Suppose we want to estimate a homoscedastic Gaussian regression model with response variable y, P-spline effects $f_1(X1)$ and $f_2(X2)$ for the continuous covariates $X1$ and $X2$, and a random intercept for the cluster variable C. Assume further that $X1$ is an individual specific covariate whereas $X2$ is a cluster specific covariate according to the cluster variable C. Then the model can be specified as:

```
b.hregress y = const + X1(pspline)  + X2(pspline) + C(random) , family=normal
equationtype=mu using d
```

As an alternative to this specification, the model could be defined in hierarchical form. To do so we first need a second dataset dC, say, where the unique observations of the two cluster variables C and $x2$ are stored. The first 5 lines of this dataset in ASCII format may look like this:

```
C X2
1 -2.3
2 2.2
3 1.4
4 3
5 -1.6
```

Note that the values of the cluster variable C must be stored in increasing order! Now the model can be reformulated in hierarchical order as follows:

```
b.hregress C = X2(pspline) , family=gaussian_re equationtype=mu using dC
b.hregress y = const + X1(pspline)  + C(hrandom) , family=normal
equationtype=mu using d
```

In this form we have defined the random intercept with additional nonlinear covariate effect of the cluster specific variable $X2$ using a separate equation. While this specification is formally equivalent to the first model definition, estimation is now carried out in the hierarchical way as described in detail in Lang et al. (2014). The hierarchical formulation is usually more favorable as the computing time for every iteration is reduced and also the mixing of the MCMC sampler (sometimes dramatically) improves.

Random slope models

It is also possible estimate random slopes hierarchically. Suppose we want to estimate an interaction effect (random slope) between the individual specific covariate $X1$ and the cluster variable C. In non-hierarchical form we obtain:

```
b.hregress y = const + X1(pspline)  + X2(pspline) + C(random) + X1*C(random) ,
family=normal equationtype=mu using d
```

The same model in hierarchical form is given as:

```
b.hregress C = X2(pspline) , family=gaussian_re equationtype=mu using dC
dC.generate C2 = C
b.hregress C2 = const , family=gaussian_re equationtype=mu using dC
```

```
d.generate C2 = C
b.hregress y = const + X1(pspline)  + C(hrandom) + X1*C2(hrandom) ,
family=normal equationtype=mu using d
```

Note that we have to duplicate the cluster variable C in both datasets in order to distinguish the random slope equation from the random intercept equation.

Three level models

BayesX is able to estimate hierarchical models with arbitrary number of hierarchy levels. Suppose there is another cluster variable D which is nested in C. Suppose further that we have an additional covariate X3 at cluster level D. Then a three level random intercept model is defined hierarchically as follows:

```
b.hregress C = X3   , family=gaussian_re equationtype=mu using dD
b.hregress C = X2(pspline) + D(hrandom)  , family=gaussian_re
equationtype=mu using dC
b.hregress y = const + X1(pspline)  + C(hrandom) , family=normal
equationtype=mu using d
```

Here the cluster level dataset for variable D is dD. For the cluster level covariate X3 a linear effect is assumed.

Random slope models could be defined analogously.

Placing the overall intercept

Placing the overall intercept in hierarchical models is a delicate issue as it can have a strong impact on the mixing behaviour of the resulting Markov chains. Usually the best option is to place the overall intercept in the lowest level random intercept equation. In doing so our previous model becomes

```
b.hregress C = const + X3   , family=gaussian_re equationtype=mu using dD
b.hregress C = X2(pspline) + D(hrandom)  , family=gaussian_re
equationtype=mu using dC
b.hregress y = X1(pspline)  + C(hrandom) , family=normal equationtype=mu using d
```

10.1.3 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`). Option `burnin` must be specified in the first equation of the model, otherwise the option will be ignored.
  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 burn-in iterations. Option `iterations` must be specified in the first equation of the model, otherwise the option will be ignored.
  DEFAULT: `iterations=52000`

- `step =` *integer*
  Defines the thinning parameter for MCMC simulation. For example, `step = 50` means, that 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. Option `step` must be specified in the first equation of the model, otherwise the option will be ignored.
  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 with homoscedastic variance. *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 with homoscedastic variance. *realvalue* must be a positive real valued number.
  DEFAULT: `bresp=0.005`

- `equationtype =` *characterstring*
  Defines the type of equation in the model, e.g. `equationtype=sigma2` and `equationtype=mu` for Gaussian responses. The admittable equation types depend on the response distribution family, see Tables 10.6, 10.7, 10.8 and 10.9.
  DEFAULT: `equationtype=mu`

- `family =` *characterstring*
  Defines the distribution of the response variable in the model. Supported models can be found in Tables 10.6, 10.7, 10.8 and 10.9. For some distributions (e.g. multinomial) additional options may be specified to control MCMC inference. A list of distributions with associated additional options is given in Table 10.10.
  A more detailed description on how to specify the distribution of the response is given in subsubsection 10.1.2.4.
  DEFAULT: `family=normal`

- `reference =` *realvalue*
  Option `reference` is meaningful only if `family=multinom_probit` 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.

value of `family`	response distribution	link	additional options
`family=normal`	Gaussian	identity	`aresp`, `bresp`
`family=multinom_probit`	unordered multinomial	probit	`reference`

Table 10.10: Response distributions with additional options.

Further options

Options are listed in alphabetical order:

- `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 as well as simultaneous 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%. Option `level1` must be specified in the first equation of the model, otherwise the option will be ignored.

- `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 as well as simultaneous 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%. Option `level2` must be specified in the first equation of the model, otherwise the option will be ignored.

- `predict`
  Option `predict` may be specified to compute predicted values, samples of the deviance D, the deviance information criterion DIC, etc. The following specifications are possible:

  - `predict=no`
    This the default specification, i.e. predicted values are not computed.
  - `predict=full`
    This specification computes and reports the model deviance, the DIC, posterior means, medians, standard deviations and some quantiles of the predictor, the estimated parameter and the expected value. The deviance and DIC are reported in the regression output as well as in a file which ends with "predict_DIC.res". The other results are stored in a file which ends with "predict.res".

    Regarding the computation of the DIC and related quantities please take a look at page 85.

– `predict=fulls`
This specification provides the same output as `predict=full`. If method `getsample` is applied after method `hregress` then `predict=fulls` provides an additional file (which ends with "predict_sample.raw") with all samples of the computed quantities. Note that the size of the output file is huge even for very simple models because the number of sampled quantities is proportional to the number of observations. Usually it is not necessary to store all samples of the quantities obtained with `predict=full` or `predict=fulls`. Therefore the specification `predict=full` is favorable compared to `predict=fulls`.

10.1.4 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 10.1.3 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, R) 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 17 or 18 columns depending on whether the corresponding model term is an interaction effect. The first column contains an index (starting with one), the second column (and the third column if the estimated effect is an interaction effect) 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, standard deviations and the 2.5%, 10%, 50%, 90% and 97.5% pointwise credible intervals. Other quantiles may be obtained by specifying the `level1` and/or `level2` option, see subsection 10.1.3 for details. The next 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. The remaining 6 columns provide simultaneous credible intervals together with their corresponding posterior probabilities. The simultaneous credible intervals are computed according to a proposal by Krivobokova et al. (2010).

The estimated nonlinear effects can be visualized by using either the graphics capabilities of *BayesX* or the *BayesX* R package, see section 11.1 and section 11.2, respectively. Of course, any other (statistics) software package with plotting facilities may be used as well.

10.1.5 Examples

Here we give only a few examples about the usage of method `hregress`.

Suppose that we have a data set `test` with a continuous response variable `y`, and covariates `x1`, `x2` and `x3`. Suppose further that we have already created a *mcmcreg object* `b`.

10.1.5.1 Single equation and parameter models

We start by specifying Gaussian regression models for the response variable `y` with homoscedastic variance σ^2, i.e. we define only a predictor for the mean μ of the normal distribution.

Fixed effects

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

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

This model is estimated by typing:

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

Here, `step=10` defines the thinning parameter, i.e. 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=normal`, a Gaussian regression model is estimated.

Additive models

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

```
> b.hregress y = const + x1 + x2(pspline) + x3(pspline), iterations=12000
  burnin=2000 step=10 family=normal using test
```

which corresponds to the predictor

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

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.hregress y = const + x1 + x2(pspline) + x3(pspline) +
  region(spatial,map=m), iterations=12000 burnin=2000
  step=10 family=normal 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.hregress y = const + x1 + x2(psplinerw2) + x3(psplinerw2) +
   region(spatial,map=m) + region(random), iterations=12000
  burnin=2000 step=10 family=normal using test
```

10.1.5.2 Multiple equation and parameter models

We now add another equation for the variance of the Gaussian regression model. We assume the same complex predictor for the mean and variance equation as in the previous section and specify the two equations by:

```
> b.hregress y = const + x1 + x2(psplinerw2) + x3(psplinerw2) +
   region(spatial,map=m) + region(random), iterations=12000
  burnin=2000 step=10 family=normal equationtype=sigma2 using test
> b.hregress y = const + x1 + x2(psplinerw2) + x3(psplinerw2) +
   region(spatial,map=m) + region(random),
  family=normal equationtype=mu using test
```

Note that we additionally specify the option `equationtype` in both equations in order to distinguish between the variance and mean equation. Note also that MCMC specific details should only be specified in the first equation. In fact, a possible specification in the second or following equations will be ignored.

In both equations we estimate region specific random effects and a smooth spatial effect. It is therefore favorable to rewrite the model in multilevel form. This done using the following commands:

```
> b.hregress region = const  + region2(spatial,map=m) , iterations=12000
  burnin=2000 step=10 family=gaussian_re equationtype=sigma2 using regiondata

> b.hregress y =  x1 + x2(psplinerw2) + x3(psplinerw2) + region(hrandom),
  family=normal equationtype=sigma2 using test
> b.hregress region = const  + region2(spatial,map=m) ,
   family=gaussian_re equationtype=mu using regiondata

> b.hregress y = const + x1 + x2(psplinerw2) + x3(psplinerw2) +
  region(hrandom),  family=normal equationtype=mu using test
```

The model contains now two more equations for the random effects. We shifted the overall intercept into the random effects equations to improve the mixing of MCMC samples, see Lang et al. (2014) for an explanation. The smooth spatial effect is also contained in the random effects equations to improve mixing and speed up computations.

10.2 Method autocor

This method is a post estimation command, i.e. its usage is meaningful only if method `hregress` has been applied before. Method `autocor` computes the autocorrelation functions of all sampled (and stored) parameters.

The usage of method `autocor` is identical to its corresponding counterpart for *bayesreg objects*, see section 7.2 for details.

10.3 Method getsample

This method is a post estimation command, that is only meaningful if method `hregress` 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* or by using the R function `plotsample`. Of course, any other program with graphics capacities could be used as well.

The usage of method `getsample` is identical to its corresponding counterpart for *bayesreg objects*, see section 7.3 for details.

10.4 Global options

The purpose of global options is to affect the global behavior of a *mcmcreg 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 *mcmcreg objects*:

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

  <*INSTALLDIRECTORY*>`\output`.

  The default filename is composed of the name of the *mcmcreg 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 *mcmcreg 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.

10.5 Visualizing estimation results

Visualization of estimation results is described in chapter 11.

Chapter 11

Visualizing estimation results

In this chapter we show, how estimation results produced with one of the regression tools described in the two previous chapters can be visualized. In general, there are two possibilities to visualize results: Within *BayesX*, special functions can be applied to regression objects. Since all three regression tools provide almost the same possibilities to visualize results, we describe them simultaneously in the next section. Tools for the visualization of autocorrelations for MCMC samples are described in subsection 11.1.3. An alternative way to visualize results is to use the R package supplementing *BayesX*. Some further comments on this package are provided in section 11.2.

11.1 BayesX functions

BayesX allows to visualize estimation results immediately after estimation. The *output window* and/or the log file describe how to do this for a particular model term. Nonlinear effects of continuous covariates and time scales are plotted with method `plotnonp`. Spatial effects are visualized with method `drawmap`. When using *bayesreg objects*, autocorrelation functions of sampled parameters can be visualized with method `plotautocor`.

11.1.1 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. T

Syntax

> *objectname*.`plotnonp` *termnumber* [*, options*]

Plots the estimated effect with term number *termnumber*. The term number will be printed in the *output window* and/or an open log file. Several options are available for labelling axis, adding a title, etc., see the options list below. Note that method `plotnonp` can be applied only if random walks, P-splines or seasonal components are used as priors.

Options

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

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

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

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

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

  `connect=15`

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

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

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

- `fontsize =` *integer*

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

- `height =` *integer*

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

- `levels = all|1|2|none`

  By default, `plotnonp` plots the estimated nonlinear covariate effect together with the pointwise credible intervals based on nominal levels of 80% and 95% (the nominal levels may be changed using the options level1 and/or level2). Option `levels` allows to omit completely pointwise credible intervals in the graphs (`levels=none`), print only the 95% credible intervals (`levels=1`) or to print only the 80% credible intervals (`levels=2`).

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

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

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

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

  ```
  linecolor = Bgr
  ```

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

- `linewidth =` *integer*

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

- `outfile =` *characterstring*

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

- `pointsize =` *integer*

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

- `replace`

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

- `title =` *characterstring*

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

- `titlesize =` *realvalue*

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

- `width =` *integer*

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

- `xlab =` *characterstring*

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

- `xlimbottom =` *realvalue*

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

- `xlimtop =` *realvalue*

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

- `xstart =` *realvalue*

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

- `xstep =` *realvalue*

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

- `ylab =` *characterstring*

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

- `ylimbottom =` *realvalue*

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

- `ylimtop =` *realvalue*

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

- `ystart =` *realvalue*

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

- `ystep =` *realvalue*

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

Examples

Suppose we have already created a regression object `reg` and have estimated a regression model with Gaussian errors using a command like

```
> reg.regress Y = X(psplinerw2), family=gaussian using d
```

where `Y` is the response variable and `X` the only explanatory variable. The effect of `X` is modelled nonparametrically using Bayesian P-splines. In the *output window* we obtain the following estimation output for the effect of `X`:

```
f_x

Results are stored in file
c:\results\reg_f_x_pspline.res
Results may be visualized using method plotnonp
Type for example: objectname.plotnonp 0
```

The term number of the effect of X is 0, i.e. by typing

```
> reg.plotnonp 0
```

we obtain the plot shown in Figure 11.1.

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

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

we obtain the plot shown in Figure 11.2.

By default, the plots appear in an additional window on the screen. They can be directly stored in postscript format by adding option `outfile`. For example by typing

```
> reg.plotnonp 0 , title="my title" xlab="x axis" outfile="c:\results\result1.ps"
```

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

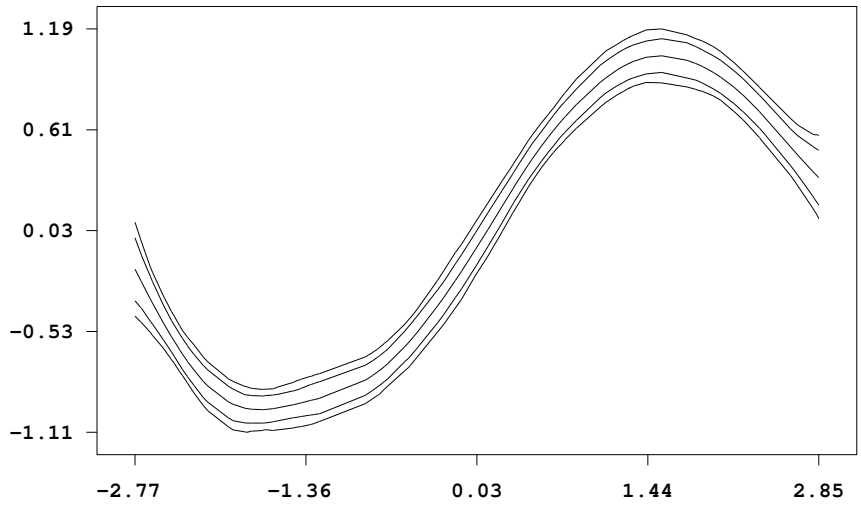


Figure 11.1: Illustration for the usage of method `plotnonp`

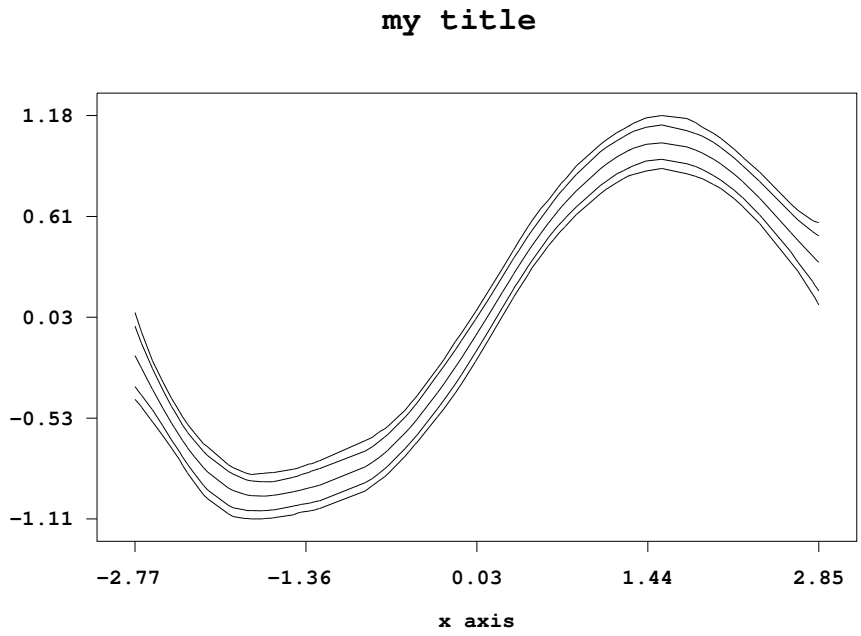


Figure 11.2: Second illustration for the usage of method `plotnonp`

11.1.2 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.

Syntax

> *objectname*.`drawmap` *termnumber* [, *options*]

Visualizes the effect of a spatial covariate by coloring the regions of the corresponding geographical map according to the estimated effect (or other characteristics of the posterior). The term number *termnumber* identifies the model term and can be found in the *output window* and/or an open log file. Several options are available for adding a title or changing the color scale etc., see the options list below. Note that method `drawmap` can be applied only if Markov random fields, geosplines or geokriging are used as priors.

Options

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

- `color`

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

- `drawnames`

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

- `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 (see option `titlesize`). The default is `fontsize=12`.

- `hcl`

  Requests that a color palette from the HCL color space should be used instead of an RGB palette. The HCL colors will be selected diverging from a neutral center (grey) to two different extreme colors (red and green) in contrast to the RGB colors diverging from yellow to red and green. HCL colors are particularly useful for electronic presentations since they are device-independent. The option `hcl` is only meaningful in combination with the option `color`.

- `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.

- `nolegend`

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

- `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.

- `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.

- `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`.

- `plotvar =` *variablename*

  By default, the regions of the map are colored according to the estimated spatial effect. Option `plotvar` allows to color the map according to other characteristics of the posterior by explicitly specifying the name of the variable to be plotted. Compare the header of the file containing the estimation results to see all variables available for plotting.

- `replace`

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

- `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"`).

- `titlesize =` *realvalue*

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

- `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.

Examples

Suppose we have already created a regression object `reg` and have estimated a regression model with Gaussian errors using something like

```
> map m
> m.infile using c:\maps\map1.bnd
> reg.regress Y = region(spatial,map=m), family=gaussian using d
```

where `Y` is the response variable and `region` the only explanatory variable. The effect of the spatial covariate `region` is modelled nonparametrically using a Markov random field. In the *output window* we obtain the following estimation output for the effect of `region`:

```
f_spat_region

Results are stored in file
c:\results\reg_f_region_spatial.res
Results may be visualized using method 'drawmap'
Type for example: objectname.drawmap 0
```

The term number of the effect of `region` is 0, i.e. by typing

```
> reg.drawmap 0
```

we obtain the map shown in Figure 11.3 where the regions are colored according to the estimated spatial effect.

By default the regions are colored in grey scale. A color scale is obtained by adding option `color`. A title can be added as well. For example by typing

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

we obtain the map shown in Figure 11.4.

By default, the maps appear in an additional window on the screen. They can be directly stored in postscript format by adding option `outfile`. For example by typing

```
> reg.drawmap 0 , color title="my title" outfile="c:\results\result1.ps"
```

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

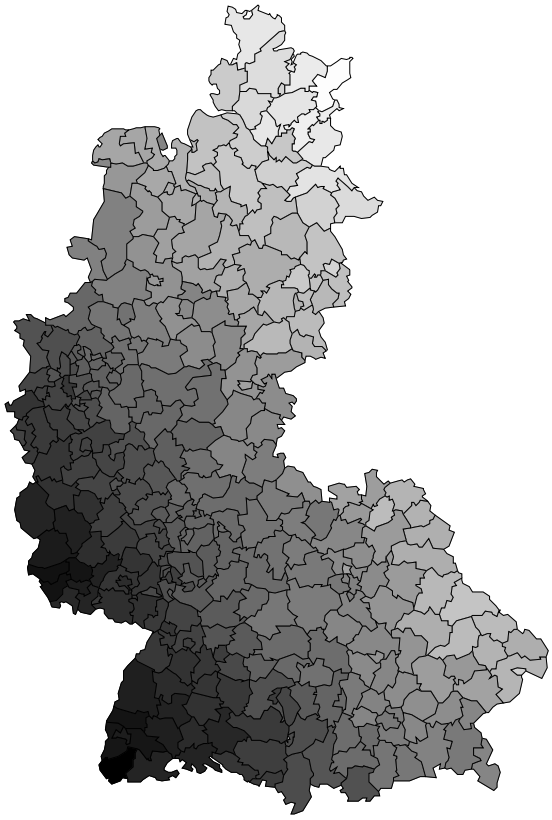


Figure 11.3: Illustration for the usage of method `drawmap`

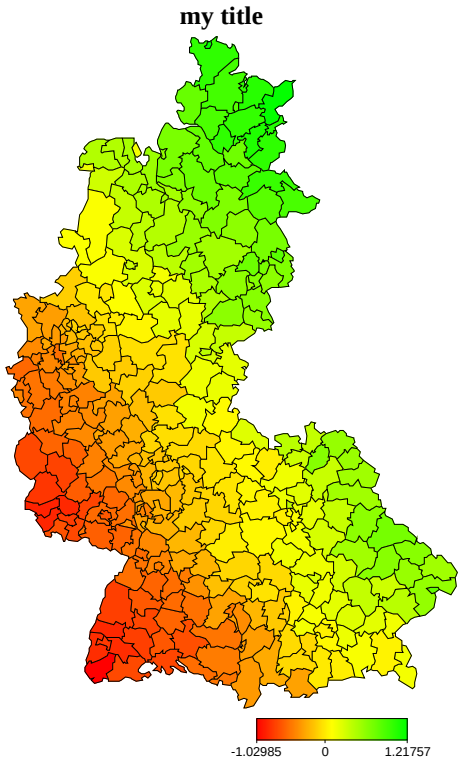


Figure 11.4: Second illustration for the usage of method `drawmap`

11.1.3 Method plotautocor

Description

Method `plotautocor` is a post estimation command, i.e. it is meaningful only if method `regress` has been applied before. Method `plotautocor` computes and visualizes the autocorrelation functions of the parameters in the model. This method is only applicable to *bayesreg objects*.

Syntax

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

Computes and visualizes the autocorrelation functions in the model. Several options are available for specifying the maximum lag for autocorrelations, storing the graphs in postscript format etc., see the options list below.

Options

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

- `maxlag =` *integer*

  Option `maxlag` may be used to specify the maximum lag for autocorrelations. The default is `maxlag=250`.

- `mean`

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

- `outfile =` *characterstring*

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

- `replace`

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

Examples

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

```
> reg.regress Y = X(psplinerw2), family=gaussian using d
```

where `Y` is the response variable and `X` the only explanatory variable. The effect of `X` is modelled nonparametrically using Bayesian P-splines. We can now check the mixing of sampled parameters by computing and drawing the autocorrelation functions up to a maximum lag of 150:

```
> reg.plotautocor , maxlag=150 outfile="c:\results\autocor.ps"
```

In this example the autocorrelation functions are not shown on the screen but stored in postscript format in the file `c:\results\autocor.ps`. If option `outfile` is omitted, the functions are plotted on the screen. The resulting file contains 5 pages. As an example, the first page of the file is shown in Figure 11.5.

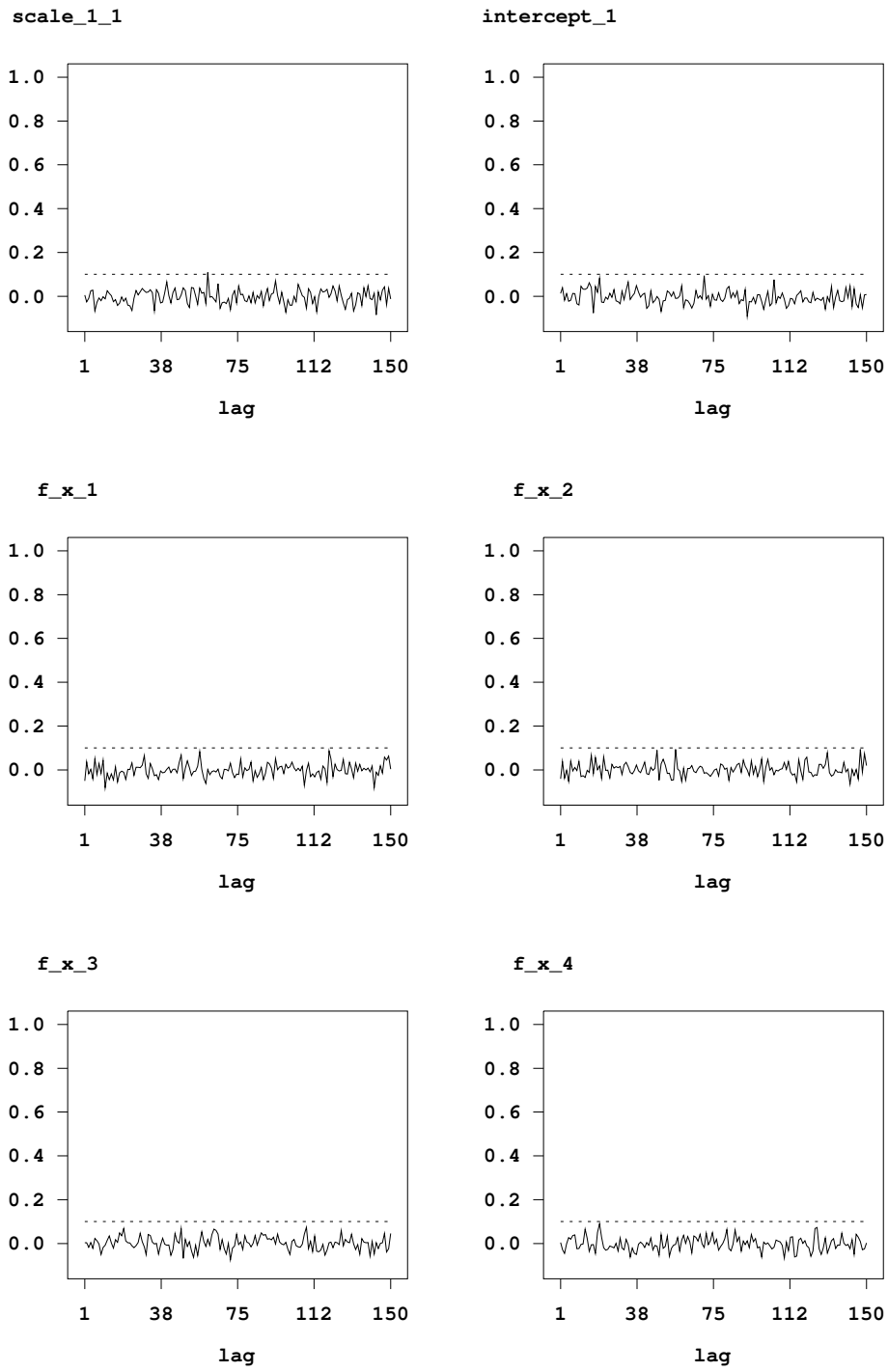


Figure 11.5: Illustration for the usage of method `plotautocor`

11.2 R package BayesX

To provide some convenient graphics facilities for the command line version of *BayesX*, an R package has been developed that contains virtually the same functionality as described in the preceding section. In addition, routines for manipulating geographical information are available. The R package *BayesX* is available from CRAN (http://www.r-project.org).

Chapter 12

DAG Objects

Author: Eva–Maria Fronk

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 & Heckerman (1994). We refer to Brooks (1998) or Gilks, Richardson & Spiegelhalter (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 & Green (1999). The following explanations to the statistical background of the program can be found in more detail in Fronk & Giudici (2004).

12.1 Method estimate

12.1.1 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 $\mathbf{X}_{pa(i)}$,

$$X_i \mid \mathbf{x}_{pa(i)}, \boldsymbol{\beta}_{i|pa(i)}, \sigma^2_{i|pa(i)}, d \sim \mathrm{N}(\beta_{i0} + \sum_{x_l \in pa(x_i)} \beta_{il} x_l, \sigma^2_{i|pa(i)}).$$

The joint distribution of all variables $\mathbf{X} = (X_0, \ldots, X_{p-1})'$ is then given by

$$p(\mathbf{x} \mid \boldsymbol{\beta}, \boldsymbol{\sigma}^2) = \prod_{i=0}^{p-1} p(x_i \mid \mathbf{x}_{pa(i)}, \boldsymbol{\beta}_{i|pa(i)}, \sigma^2_{i|pa(i)}),$$

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 $\mathbf{x}_{pa(i)}$. 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

$$p(\mathbf{x} \mid \boldsymbol{\beta}, \boldsymbol{\sigma}^2, d) \quad = \quad \prod_{i=0}^{p-1} p(x_i \mid \mathbf{x}_{pa(i)}, \boldsymbol{\beta}_{i|pa(i)}, \sigma^2_{i|pa(i)})$$

and the "global parameter independences"

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

(for a detailed description see Geiger & Heckerman (1999), we get the joint distribution:

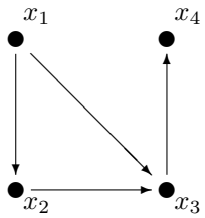
$$\begin{aligned} & p(\mathbf{x}, \boldsymbol{\beta}, \boldsymbol{\sigma}^2, d) \\ &= \quad p(\mathbf{x} \mid \boldsymbol{\beta}, \boldsymbol{\sigma}^2, d)\, p(\boldsymbol{\beta} \mid \boldsymbol{\sigma}^2, d)\, p(\boldsymbol{\sigma}^2 \mid d)\, p(d) \\ &= \quad \prod_{i=0}^{p-1} p(\mathbf{x}_i \mid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

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 12.1.



$$A = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

Figure 12.1: 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 $\pi(d, \boldsymbol{\mu}, \boldsymbol{\beta}, \boldsymbol{\sigma}^2 \mid \mathbf{x})$ 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 section 12.1.3.

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 $\mathbf{x}_{pa(i)}$ shall be given by a probit model, i.e.

$$p_i \;=\; E(X_i \mid \mathbf{x}_{pa(i)}) \;=\; \Phi(\mathbf{x}'_{pa(i)}\boldsymbol{\beta}_{i|pa(i)}, \sigma^2_{i|pa(i)}), \tag{12.1}$$

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 & 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 $\mathrm{N}(\mathbf{x}'_{pa(i)}\boldsymbol{\beta}_{i|pa(i)}, 1)$, 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 $\mathbf{x}_{pa(i)}$ 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 $f(x_i \mid \mathbf{x}_{pa(i)})$ are then CG regressions 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 $Z_{ki} \mid x_{ki}, \mathbf{x}_{kpa(i)}\boldsymbol{\beta}_{i|pa(i)}$

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 & Madigan (1997a), Andersson, Madigan & Perlman (1997b) and Chickering (1995).

12.1.2 Syntax

The creation of objects has been described in general in subsection 2.5.1; in the context of dag models the corresponding dag object is created by:

`dag` *objectname*,

To perform a model selection as described above call:

objectname.`estimate` *variables* [`if` *expression*], [*options*] `using` *dataset*

Then the method `estimate` estimates the dag for the variables given in *variables* which have to be defined in *dataset*. The parameters of the via the dag defined regression models are also estimated. An if-statement may be specified to analyze only a part of the data set, i.e. only those observations where *expression* is true. There are several facultative *options* concerning the (start) parameters of the algorithm or the kind of output at the end. They are listed in the next paragraphs.

12.1.3 Options

Options for controlling MCMC simulations

The following options correspond to those given on page 170 and are therefore only briefly explained.

- `burnin =` b
  Changes the number of burnin iterations from 2000 to b; it is a positive integer number with $0 < b < 500001$.
  DEFAULT: `burnin = 2000`
- `iterations =` i
  Changes the number of MCMC iterations from 52000 to i; it is a positive integer number with $0 < i < 10000000$.
  DEFAULT: `iterations = 52000`

- `step = s`
  Changes the thinning parameter of MCMC iterations from 50 to s; it is a positive integer number with $0 < s < 1000$.
  DEFAULT: `step = 50`

Options for initial values of algorithm

- **Changing hyperparameters of partial variances**
  As already mentioned we assume $\sigma^2_{i|pa(i)} \sim IG(\delta_{i|pa(i)}, \lambda_{i|pa(i)})$ for $i = 0, \ldots, p-1$. By the following two commands the values of the two hyperparameters $\delta_{i|pa(i)}$ and $\lambda_{i|pa(i)}$ can be freely chosen. If this is not done the default values correspond to a non-informative gamma distribution.

  - `delta = c`
    Specifies the first parameter of the inverse gamma distribution of the partial variances, $\delta_{i|pa(i)}$, is set equal to d. Otherwise it is equal to 1. The value c has to be of type realvalue with $0 < c < 20$.
    DEFAULT: `delta = 1`
  - `lambda = l`
    Specifies the second parameter of the inverse gamma distribution of the partial variances, $\lambda_{i|pa(i)}$, is set equal to l. Otherwise it is equal to 0.005. The value d has to be of type realvalue with $0 < d < 20$.
    DEFAULT: `lambda = 0.005`

- **Choosing special graph to start from**
  Usually the algorithm starts from the independent model, that means from a dag without any edges. This can be changed by the command

  `type = 0/1/2/3/4`
  DEFAULT: `type = 0`

  where the different values have the following meanings:

  - `type=0`
    Algorithm starts from an **independent** model with no edges.
  - `type=1`
    Algorithm starts from a **complete** model where all edges are directed from "lower" variables to "higher" ones, i.e. $x_i \to x_j$, $\forall i < j$.
  - `type=2`
    Algorithm starts from a **complete** model where all edges are directed from "higher" variables to "lower" ones, i.e. $x_j \to x_i$, $\forall i < j$.
  - `type=3`
    Algorithm starts from a model where there is an edge from each variable to the next "higher" one, like a **chain**, i.e. $x_i \to x_j$, $\forall i = j + 1$.
  - `type=4`
    Algorithm starts from a model where there is an edge from each variable to the next "lower" one, like a **chain**, i.e. $x_j \to x_i$, $\forall i = j + 1$.

Options concerning the way of model selection

- **Kind of switch step**
  There are three ways how the switch step can be carried out in the rj-algorithm. The first one is similar to the performance of a birth or death step (i.e. adding or deleting an edge): A proposal is made and then accepted by its corresponding acceptance ratio. As it may be very complicate to calculate a good proposal, a simplification can be achieved by the consideration if the switch step leads to an equivalent dag model. If this holds true, the given and the proposed dag should be statistically indistinguishable. The proposed dag can therefore be accepted with probability 0.5. The kind of switch step can be chosen by the command

  `switch = normal/equi/mix`,
  DEFAULT: `switch = normal`

  which differ in the following way:

  - `switch=normal`
    The switch step is carried out by proposing the new dag and accepting it with the corresponding acceptance probability. The transformation into equivalent model may occur very seldom and, consequently, the acceptance ratio very low.
  - `switch=equi`
    The switch step is only allowed if it results into an equivalent model. In this case, it is performed with a probability of 0.5. Transformations into a non-equivalent model can only occur by a birth or death step.
  - `switch=mix`
    This command causes a mixture of both procedures described above: If the proposed switch step leads to an equivalent model it is accepted with probability 0.5. If it results into a non-equivalent model a proposal is made and accepted by the corresponding acceptance ratio.

- **Kind of distribution family / interactions**
  There are three different types of data sets as they can consists of continuous, binary, or mixed variables which results in the assumption of a Gaussian, a multinomial, or a conditional Gaussian distribution. Dependent on the kind of data set the rj-algorithm for the model selection changes as described above.This can be indicated by the optional command

  `family = continuous/discrete/mixed`.

  In the case that the model selection for a binary data set shall be carried out accounting for interactions the command

  `family = discrete_ia`

  is needed instead of `family = discrete`. In this case, a special option concerning the output is given by the command `detail_ia` which is explained below.

- **Restriction to the search space**
  It is possible to restrict the search space, i.e. to state an (missing) edge as fix or determinate the orientation of an edge. This is done by writing the restrictions into a file *restrict* which is then read by the command

`fix_file =` *path_of_restrict*

The restriction is then given by a $p \times p$ matrix that lies under the path *path_of_restrict*. The matrix is allowed to have three possible entries, namely 0,1, and 2 which have the following meaning: An entry of 2 corresponds to no restriction of the corresponding edge, it may occur or not. An entry of 1 indicates that this edge has to exist in each graph of the Markov chain, whereas 0 denotes that the corresponding edge must not occur.

Options concerning the output

- **Estimated regression coefficients**
  As already mentioned, the parameters of each regression model are estimated in every iteration. Because of the fact that the qualitative structure of the dag is usually of greater interest than the quantitative estimations of the regression coefficients, in the standard output these estimated parameters are omitted. Nevertheless

  `print_dags`

  gives the mean, the 10%, 50% and the 90% quantile of every parameter of all regression models. As the model space for dags is very huge we abandon the possibility to store the estimated values for each dag. To perform the necessary calculation *BayesX* creates a temporary file under the device *c:\\...*. For this purpose, its important to ensure that a device with this name exists. Otherwise the user has to provide an alternative path for the storage file by the command

  `store_file =` *alternative_path*

- **Estimated coefficients of interactions**

- **Criteria for the listed models**
  As model selection for dags is performed in an extremely huge search space one might not want to get a list of all models which have been visited during MCMC estimation regardless of the relative frequency of their appearance. The option

  `print_models = all/prob/limit/normal`,
  DEFAULT: `print_models = normal`

  allows to focus on special criterions for the models printed in the output.

  - `print_models = all`
    All models which have been visited by the Markov chain are printed.
  - `print_models = prob`
    The most frequent models of the chain are printed except for those which are the less frequent ones and have altogether a probability of alpha=0.05. The value of alpha can be changed as it is explained a few lines below.
  - `print_models = limit`
    Here, the first 10 models with the highest frequencies are printed. The number of listed models can be made different from 10 as it is explained a few lines below.
  - `print_models = normal`
    The option `normal` is a mixture of `limit` and `prob`, as it chooses the one which produces less models. The default parameters are again alpha=0.05 and number=10.

Number of different dags visited by the algorithm: 16

********* DIFFERENT MODELS sorted by frequencies *********
********************* all models **********************

010	*000*	*000*	*1*	*3894*	*0.3890*
000	*100*	*000*	*1*	*3814*	*0.3810*
000	*100*	*100*	*2*	*806*	*0.0806*

⋮

Figure 12.2: Example for model listing in the output.

The default setting of `print_models` is `print_models=normal`.

- `number =` n
  Changes the number of printed models in the option `print_models = limit` to n. The variable number has to be of type realvalue with $0 \leq n \leq 10000$.
  DEFAULT = 10

- `alpha =` a
  Sets alpha in the option `print_models = prob` equal to a. That means when using `print_models = prob` the most frequent models which unify 1-a of the posterior probability are printed. The variable alpha has to be of type intvalue with $a \in [0, 1]$.
  DEFAULT = 0.05

- `printit =` p
  Prints every p-th iteration in the output window instead of every 100-th. The printing of the iterations can be suppressed by setting p higher than the number of iterations. The variable printit has to be of type intvalue with $0 < p < 10000001$.
  DEFAULT = 100

12.1.4 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 12.2 gives the information that the most frequent model is represented by the adjacency matrix $A = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$ and contains 1 edge. It occurred in the thinned out Markov chain for 3894 times which corresponds to a relative frequency of about 0.389. Notice, that different dags may represent the same statistical model as they may be Markov-equivalent.

- **Listing of essential graphs**
  Additional to the dags, the essential graphs are printed, too. I.e. those dags, which are

Number of different equivalent classes visited by the algorithm: 6

******* DIFFERENT EQUIVALENCE CLASSES sorted by frequencies ********
**************************** all models *******************************

Skeleton: 010 000 000
No immoralities.
Number of edges: 1 Abs.freq.: 7708 Rel.freq.: 0.771

Skeleton: 011 000 000
Immoralities: (0;1,2)
Number of edges: 2 Abs.freq.: 806 Rel.freq.: 0.0806

Skeleton: 011 000 000
No immoralities.
Number of edges: 2 Abs.freq.: 523 Rel.freq.: 0.0523

⋮

Figure 12.3: Example for listing of equivalence classes in the output.

equivalent to each other, are summarized and represented by their essential graph. The representation of the essential graph, which can contain undirected as well as directed edges, is as follows. First the underlying graph, the skeleton, is given by the adjacency matrix as described above. But now, the entries indicate always an undirected edge. (E.g. the undirected graph a–b of the two variables a and b is given by 01 00.) Then the immoralities of the essential graph are listed. Remember that within an essential graph an oriented edge can only occur as a part of an immorality $b \to a \leftarrow c$ which is here represented by the triple (a;b,c). The example output of Figure Figure 12.3 shows that the first two dag models of Figure 12.2 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.

Bibliography

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

Andersson, S. A., Madigan, D., & 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., & Perlman, M. D. (1997b). On the Markov equivalence of Chain Graphs, Undirected Graphs, and Acyclic Digraphs. *Scandinavian Journal of Statistics*, **24**, 81–102.

Belitz, C. (2007). *Model Selection in Generalized Structured Additive Regression Models.* PhD Thesis, University of Munich.

Belitz, C. & Lang, S. (2008). Simultaneous selection of variables and smoothing parameters in structured additive regression models. *Computational Statistics and Data Analysis*, **53** , 61-81.

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

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

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

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

Brezger, A. & Lang, S. (2006). Generalized additive regression based on Bayesian P-splines. *Computational Statistics and Data Analysis* **50**, 967–991.

Brezger, A. & Lang, S. (2008). Simultaneous Probability statements for Bayesian P-splines. *Statistical Modelling*, **8**, 141–168.

Brooks, S. P. (1998). Markov Chain Monte Carlo Method and its Application. *The Statistician*, **47**, 69–100.

Burnham, K. P. & Anderson, D. R. (1998). *Model Selection and Multimodel Inference.* Springer, New York.

Chambers, J. M. & Hastie, T. (1991). *Statistical Models in S.* Chapman and Hall.

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

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

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

Chickering, D. M. (1995). A Transformational Characterization of Equivalent Bayesian Network Structures. In: Besnard, P. & Hanks, S. (eds.), *Uncertainty in Artificial Intelligence, Proceedings of the Eleventh Conference*, 87–98. San Francisco: Morgan Kaufmann.

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

Fahrmeir, L., Kneib, T. & Lang, S. (2004). Penalized structured additive regression for space-time data: A Bayesian perspective. *Statistica Sinica*, **14**, 715–745.

Fahrmeir, L., Kneib, T., Lang, S. & Marx, B.(2013) *Regression: Models, Methods and Applications.* New York: Springer-Verlag.

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

Fahrmeir, L.& Osuna, L. (2006). Structured additive regression for overdispersed and zero-inflated count data. *Applied Stochastic Models in Business and Industry*, **22**, 351–369

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

Fronk, E.-M. & Giudici, P. (2004). Markov Chain Monte Carlo model selection for DAG Models. *Statistical Methods and Applications*, **13**, 259-273.

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

Geiger, D. & Heckerman, D. (1994). Learning Gaussian Networks. *Proceedings of the Tenth Conference on Uncertainty in Artificial Intelligence*, 235–243.

Geiger, D. & Heckerman, D. (1999). Parameter priors for directed acyclic graphical models and the characterisation of several probability distributions. Submitted for publication.

Gelfand, A. E., Sahu, S. K. & Carlin, B. P. (1996). Efficient Parametrizations for Generalized Linear Mixed Models. In: Bernardo, J. M., Berger, J. O., Dawid, A. P. & Smith, A. F. M. (eds.), *Bayesian Statistics 5*, 165–180. Oxford University Press.

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

Gilks, W. R., Richardson, S., & Spiegelhalter, D. J. (1996). *Markov Chain Monte Carlo in Practice.* Chapman and Hall, London.

Giudici, P. & 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.

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

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

Griffin, J. E., and Brown, P. J. (2005). Alternative prior distributions for variable selection with very many more variables than observations. Technical report, University of Warwick, Dept. of Statistics.

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

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

Hastie, T. & Tibshirani, R. (2000). Bayesian Backfitting. *Statistical Science*, **15**, 193–223.

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

Hennerfeind, A., Brezger, A. & Fahrmeir, L. (2006). Geoadditive survival models. *Journal of the American Statistical Association*, **101**, 1065–1075.

Ishwaran, H., and Rao, S. J. (2005). Spike and Slab Variable Selection: Frequentist and Bayesian Strategies. *The Annals of Statistics*, **33**, 730-773.

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.

Klein, N., Kneib, T. & Lang, S.(2014). Bayesian structured additive distributional regression. *Under revision for Annals of Applied Statistics.*

Kneib, T. (2006). Geoadditive hazard regression for interval censored survival times. *Computational Statistics and Data Analysis*, **51**, 777–792

Kneib, T. & Hennerfeind, A. (2006). Bayesian Semiparametric Multi-State Models. *Statistical Modelling*, **8**, 169–198.

Kneib, T. & Fahrmeir, L. (2006). Structured additive regression for categorical space-time data: A mixed model approach. *Biometrics*, **62**, 109–118.

Kneib, T. & Fahrmeir, L. (2007). A mixed model approach to structured hazard regression. *Scandinavian Journal of Statistics*, **34**, 207–228.

Kneib, T., Konrath, S. & Fahrmeir, L. (2009). High-dimensional Structured Additive Regression Models: Bayesian Regularisation, Smoothing and Predictive Performance. Department of Statistics, Technical Report No. 46, LMU Munich.

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

Konrath, S., Kneib, T., Fahrmeir, L. (2008). Bayesian Regularisation in Structured Additive Regression Models for Survival Data. Department of Statistics, Technical Report No.35, LMU Munich.

Krivobokova, T. & Kneib, T. & Claeskens, G. (2010). Simultaneous Confidence Bands for Penalized Spline Estimators. *Journal of the American Statistical Association*, **105**, 852–863.

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

Lang, S., Umlauf, N., Wechselberger, P., Harttgen, K. & Kneib, T. (2014). Multilevel Structured Additive Regression. *Statistics and Computing*, **24**, 223–238.

Lauritzen, S. L. (1996). *Graphical Models*, Clarendon Press, Oxford.

Lin, X. & 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. & Nelder, J. A. (1989). *Generalized Linear Models.* Chapman and Hall, London.

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

Park, T., and Casella, G. (2008). The Bayesian Lasso. *Journal of the American Statistical Association*, **482**, 681-686.

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

Tierney, L. (1983). A space-efficient recursive procedure for estimating a quantile of an unknown distribution. *SIAM Journal on Scientific and Statistical Computing*, **4**, 706-11.

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

Wood, S. N. (2006a). *Generalized Additive Models: An Introduction with R.* Chapman and Hall.

Wood, S. N. (2006b). On confidence intervals for GAMs based on penalized regression splines. *Australian and New Zealand Journal of Statistics*, **48**, 445–464.

Index