Structured Additive Regression Models: An R Interface to BayesX

Nikolaus Umlauf Universität Innsbruck
Thomas Kneib Universität Oldenburg
Stefan Lang Universität Innsbruck
Achim Zeileis Universität Innsbruck

Abstract

Structured additive regression (STAR) models provide a flexible framework for modeling possible nonlinear effects of covariates: They contain the well established frameworks of generalized linear models (GLM) and generalized additive models (GAM) as special cases but also allow a wider class of effects, e.g. for geographical or spatio-temporal data. This allows to specify complex and realistic models that can be estimated using Bayesian inference based on modern Markov chain Monte Carlo (MCMC) simulation techniques, based on a mixed model representation of STAR models, or with stepwise regression techniques combining penalized least squares estimation with model selection. Software for fitting STAR models is provided in the standalone software package **BayesX**: a comprehensive open-source regression toolbox written in C++ code. **BayesX** not only covers models for responses from univariate exponential families, but also models from less-standard regression situations such as models for multi-categorical responses with either ordered or unordered categories, continuous time survival data, or continuous time multi-state models. This paper presents the full interactive R interface **R2BayesX** to **BayesX**. With the new package, STAR models can be conveniently specified using R's formula language (with some extended terms), fitted using the **BayesX** binary, represented in R with objects of suitable classes, and finally printed/summarized/plotted. As a result of the superior graphics capabilities of R package **R2BayesX** greatly enhances the usability of **BayesX**. On the other hand some of the more complex models from the STAR class, especially models for multivariate responses, extend the already impressive capabilities for semiparametric regression in R. Moreover, **R2BayesX** is the most comprehensive package for simulation based Bayesian semiparametric regression in R.

Keywords:˜STAR models, MCMC, REML, stepwise, R.

1. Introduction

The free software **BayesX** (**?**) is a standalone program comprising powerful tools for Bayesian and mixed model based inference in complex semiparametric regression models with structured additive predictor (STAR, see Section˜**??**). To gain improved computational performance, the algorithms implemented utilize numerically efficient (sparse) matrix architectures and are written in a C++ environment. Besides exponential family regression, **BayesX** also supports models for multi-categorical responses, hazard regression for continuous survival times, and continuous time multi-state models.

In this article, we describe the full interactive R (**?**) interface to the command-line binary version of BayesX, which is an extension of package **BayesX** (**?**), that mainly pro-

vides functionality for exploring estimation results. The R package is called **R2BayesX** and has recently been added to the Comprehensive R Archive Network (CRAN, http://CRAN.R-project.org/package=R2BayesX). Within the new package, users are now provided with a typical R front end that may

- specify and estimate STAR models with **BayesX** directly from the console (function `bayesx()`, Section~**??**),
- apply a set of extractor functions and methods on **BayesX** fitted model objects, e.g. producing high level graphics of estimated effects, model diagnostic plots, summary statistics and more (Table~**??**),

In addition, users can

- run already existing **BayesX** input program files from R (function `run.bayesx()`, Section~**??**),
- automatically import **BayesX** output files into R (function `read.bayesx.output()`, Section~**??**).

Furthermore, the models supported by **BayesX** are conveniently specified using R's formula language definition, wherefore several special model term constructor functions, such as the main **R2BayesX** constructor functions `sx()` and `r()` for smooth and random effects, respectively, as well as the functions `s()` and `te()` from the **mgcv** package (**??**), facilitate a consistent way to translate R syntax into **BayesX** interpretable commands (see Section~**??**).

The remainder of this paper is as follows. The next section gives a first example of a common R session applying **R2BayesX** on a dataset on childhood malnutrition in Zambia. Section~**??** briefly discusses the methodological background of regression models with a structured additive predictor, before a description on implementation details and the user interface is given in Section~**??** and **??** respectively. In Section~**??**, the interface usability is further illustrated with the childhood malnutrition data and a dataset on forest health.

2. Motivating example

To give an introductory example of the various features of the interface, we estimate a Bayesian geoadditive regression model for the childhood malnutrition dataset in Zambia (see Section~**??**, **?**) using Markov chain Monte Carlo (MCMC) simulation.

The data consists of 4847 observations including 8 variables, both continuous and categorical. In this analysis, the main interest is on modeling the dependence of stunting (`stunting`), represented by anthropometric indicators of newborn children, on covariates including the age of the child (`age`), the body mass index of the mother (`mbmi`) and the district the child lives in (`district`). The model is given by

$$\texttt{stunting}_i = \gamma_0 + f_1(\texttt{agechild}_i) + f_2(\texttt{mbmi}_i) + f_{spat}(\texttt{district}_i) + \varepsilon_i, \qquad \varepsilon_i \sim N(0, \sigma^2),$$

where the functions f_1 and f_2 of continuous covariates `agechild` and `mbmi` have possible nonlinear effects on `stunting` and are modeled nonparametrically using P(enalized)-splines. Here, the spatially correlated effect f_{spat} of locational covariate `district` is modeled using

Kriging based on centroid coordinates (geokriging) of the districts in Zambia. To estimate the model with **BayesX** from R, the data together with a boundary map file of the districts in Zambia is loaded with

```
R> data("ZambiaNutrition", "ZambiaBnd", package = "R2BayesX")
```

Boundary files are basically shapefiles (see also package **shapefiles**, **?**) that only include information on the polygons that form the map and may be constructed using function `shp2bnd()` or imported with `read.bnd()`. The model formula is specified by

```
R> f <- stunting ~ sx(agechild) + sx(mbmi) +
+    sx(district, bs = "gk", map = ZambiaBnd, full = TRUE)
```

Finally, the model is fitted with the main model-fitting function `bayesx()`

```
R> b <- bayesx(f, family = "gaussian", method = "MCMC",
+    data = ZambiaNutrition)
```

Thereafter, a model summary is displayed by calling

```
R> summary(b)

Call:
bayesx(formula = f, data = ZambiaNutrition, family = "gaussian",
    method = "MCMC")

Fixed effects estimation results:

Parametric Coefficients:
               Mean      Sd    2.5%     50%  97.5%
(Intercept)  0.0274  0.0400 -0.0551  0.0271 0.1049

Smooth terms variances:
               Mean     Sd   2.5%    50%  97.5%    Min    Max
sx(agechild) 0.0068 0.0070 0.0013 0.0049 0.0229 0.0006 0.0892
sx(district) 0.0453 0.0182 0.0207 0.0410 0.0885 0.0127 0.1502
sx(mbmi)     0.0022 0.0034 0.0003 0.0012 0.0094 0.0002 0.0483

Scale estimate:
         Mean     Sd   2.5%    50%  97.5%
Sigma2 0.8174 0.0168 0.7847 0.8171 0.8514

N = 4847  burnin = 2000  DIC = 4887.417  pd = 38.27104
method = MCMC  family = gaussian  iterations = 12000  step = 10
```

A plot of the estimated effect for covariate `mbmi` may then be produced by typing

```
R> plot(b, term = "sx(mbmi)")
```

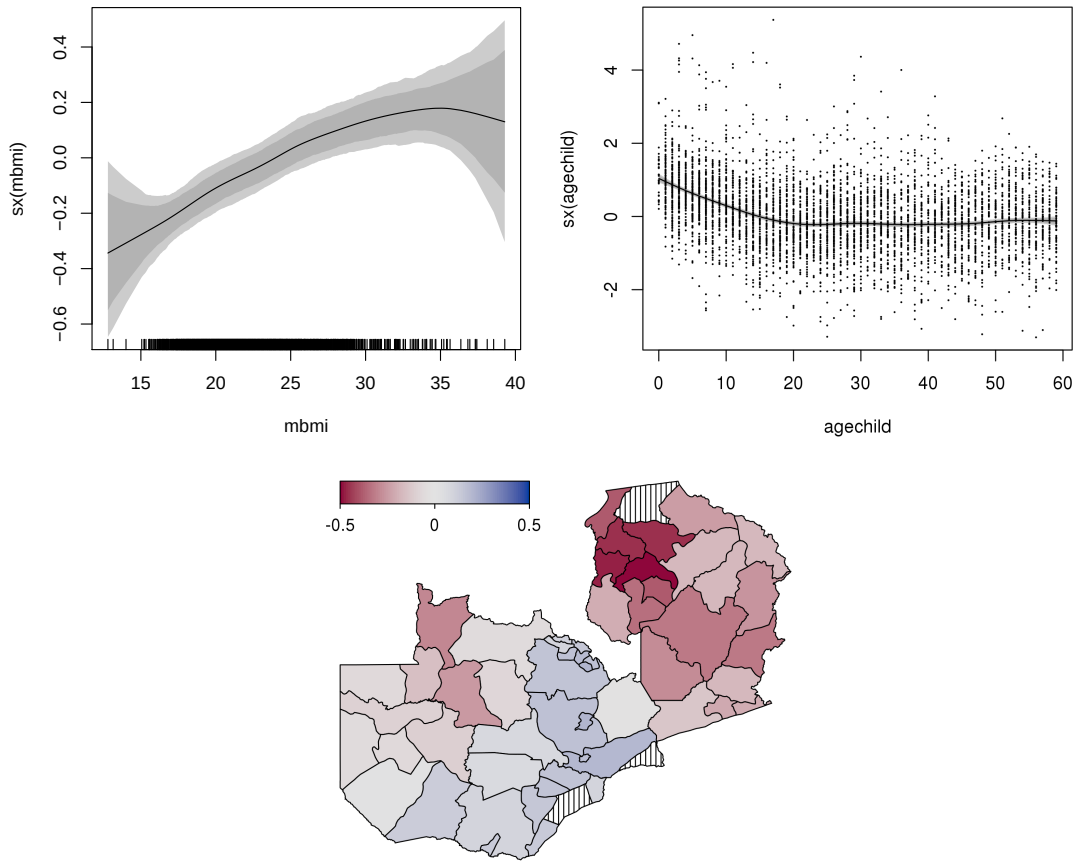

Figure 1: Visualization examples: Estimated effect for covariate `mbmi` (black line) together with 95% and 80% credible intervals (upper left panel). The upper right panel shows the estimated effect of `agechild` including partial residuals. The lower panel illustrates visualization of the estimated spatial effect for covariate `district` using a map effect plot.

and for covariate `agechild` including partial residuals by

```
R> plot(b, term = "sx(agechild)", residuals = TRUE)
```

The estimated effect of the correlated spatial effect of the districts in Zambia may e.g. be visualized using a map effect plot generated by

```
R> plot(b, term = "sx(district)", map = ZambiaBnd)
```

The plots are shown in Figure˜**??**.

3. STAR models

The STAR model class supported by **R2BayesX** is based on the framework of Bayesian generalized linear models (GLM) (e.g. see **?** and **?**). GLMs assume that, given covariates $\mathbf{x}$ and unknown parameters $\boldsymbol{\gamma}$, the distribution of the response variable y belongs to an exponential

family with mean $\mu = E(y|\mathbf{x}, \boldsymbol{\gamma})$ linked to a linear predictor η by

$$\mu = h^{-1}(\eta), \qquad \eta = \mathbf{x}'\boldsymbol{\gamma},$$

where h is a known link function and $\boldsymbol{\gamma}$ are unknown regression coefficients. In STAR models (**??**), the linear predictor is replaced by a more general and flexible, structured additive predictor

$$\eta = f_1(\mathbf{z}) + \ldots + f_p(\mathbf{z}) + \mathbf{x}'\boldsymbol{\gamma}, \tag{1}$$

with $\mu = E(y|\mathbf{x}, \mathbf{z}, \boldsymbol{\gamma}, \boldsymbol{\theta})$ and $\mathbf{z}$ represents a generic vector of all nonlinear modeled covariates. The vector $\boldsymbol{\theta}$ comprises all parameters of the functions $f_1, \ldots, f_p$. The functions f_j are possibly smooth functions comprising effects as e.g. given by

- nonlinear effects of continuous covariates: $f_j(\mathbf{z}) = f(z_1)$,
- two-dimensional surfaces: $f_j(\mathbf{z}) = f(z_1, z_2)$,
- spatially correlated effects: $f_j(\mathbf{z}) = f_{spat}(z_s)$,
- varying coefficients: $f_j(\mathbf{z}) = z_1 f(z_2)$,
- spatially varying effects: $f_j(\mathbf{z}) = z_1 f_{spat}(z_s)$ or $f_j(\mathbf{z}) = z_1 f(z_2, z_3)$,
- random intercepts with cluster index c: $f_j(\mathbf{z}) = \beta_c$,
- random slopes with cluster index c: $f_j(\mathbf{z}) = z_1 \beta_c$.

STAR models cover a number of well known model classes as special cases, including generalized additive models (GAM, **?**), generalized additive mixed models (GAMM, **?**), geoadditive models (**?**), varying coefficient models (**?**), and geographically weighted regression (**?**).

The unified representation of a STAR predictor arises from the fact that all functions f_j in (**??**) may be specified by a basis function approach, where the vector of function evaluations $\mathbf{f}_j = (f_j(\mathbf{z}_1), \ldots, f_j(\mathbf{z}_n))'$ of the $i = 1, \ldots, n$ observations can be written in matrix notation

$$\mathbf{f}_j = \mathbf{Z}_j \boldsymbol{\beta}_j,$$

where the design matrix $\mathbf{Z}_j$ depends on the specific term structure chosen for f_j and $\boldsymbol{\beta}_j$ are unknown regression coefficients to be estimated. Hence, the predictor (**??**) may be rewritten as

$$\boldsymbol{\eta} = \mathbf{Z}_1 \boldsymbol{\beta}_1 + \ldots + \mathbf{Z}_p \boldsymbol{\beta}_p + \mathbf{X}\boldsymbol{\gamma},$$

where $\mathbf{X}$ corresponds to the usual design matrix for the linear effects.

To ensure particular functional forms, prior distributions are assigned to the regression coefficients. The general form of the prior for $\boldsymbol{\beta}_j$ is

$$p(\boldsymbol{\beta}_j|\tau_j^2) \propto \exp\left(-\frac{1}{2\tau_j^2}\boldsymbol{\beta_j}'\mathbf{K}_j\boldsymbol{\beta}_j\right),$$

where $\mathbf{K}_j$ is a quadratic penalty matrix that shrinks parameters towards zero or penalizes too abrupt jumps between neighboring parameters. In most cases $\mathbf{K}_j$ will be rank deficient and the prior for $\boldsymbol{\beta}_j$ is partially improper.

The variance parameter τ_j^2 is equivalent to the inverse smoothing parameter in a frequentist approach and controls the trade off between flexibility and smoothness. For full Bayesian inference, weakly informative inverse Gamma hyperpriors $\tau_j^2 \sim IG(a_j, b_j)$ are assigned to τ_j^2, with $a_j = b_j = 0.001$ as a standard option. Small values for a_j and b_j correspond to an approximate uniform distribution for $\log \tau_j^2$. For empirical Bayes inference, τ_j^2 is considered an unknown constant which is determined via restricted maximum likelihood (REML).

In **BayesX**, estimation of regression parameters is based on three inferential concepts:

1. *Full Bayesian inference via MCMC*
   A fully Bayesian interpretation of STAR models is obtained by specifying prior distributions for all unknown parameters. Estimation is carried out using Markov chain Monte Carlo simulation techniques. **BayesX** provides numerically efficient implementations of MCMC schemes for structured additive regression models. Suitable proposal densities have been developed to obtain rapidly mixing, well-behaved sampling schemes without the need for manual tuning (**?**).

2. *Inference via a mixed model representation*
   Another concept used for estimation is based on mixed model methodology. The general idea is to take advantage of the close connection between penalty concepts and corresponding random effects distributions. The smoothing variances of the priors then transform to variance components in the random effects (mixed) model. While regression coefficients are estimated based on penalized 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 STAR models. However, estimates can also merely be interpreted as penalized likelihood estimates from a frequentist perspective (**?**).

3. *Penalized likelihood including variable selection*
   As a third alternative **BayesX** provides a penalized least squares (or penalized likelihood) approach for estimating STAR models. In addition, 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 (**?**).

A thorough introduction into the regression models supported by the program is also provided in the **BayesX** methodology manual (**?**).

4. Implementation in R

The design of the interface attempts to address the following major issues: First, the interfacing function calls should be as practical as possible and oriented on R's standards for regression model fitting functions, second, to provide a set of functions and methods for a convenient representation of fitted model objects, that would enhance the usability of **BayesX**.

In the developing process of package **R2BayesX** we first considered how to connect **BayesX** and R most appropriately. Because **BayesX** does not have an application programming interface (API), amongst others, embedding the sources in an R package seemed to be difficult since the development status of the project is very advanced. The effort of writing and reading data is comparatively small to the processing time needed for estimation of STAR models with **BayesX**. Therefore, we decided to build an interface that primarily sends input program files to the command-line binary version of **BayesX** and reads the returned output files back into R.

We also looked for existing R implementations to get an overview of best practices for software designs including GAM's. The following gives a short summary of some of the larger projects that have added to CRAN. One of the first implementations of GAM's in R is the **gam** package (**??**). The package is supporting local regression and smoothing splines in combination with a backfitting algorithm and is actually a version of the S-Plus routines for GAM's. The probably best-known and also recommended package is **mgcv** (**???**), which provides fast an stable algorithms for estimating GAM's based on GCV, REML and others. Vector generalized additive models (VGAM, **?**) for categorical responses are covered by package **VGAM** (**?**). Another comprehensive package for GAM's, accounting for responses that do not necessarily follow the exponential family and may exhibit heterogeneity, is **gamlss** (**??**). A package based on mixed model technologies is **SemiPar** (**??**) and its dependent version for adaptive splines **AdaptFit** (**?**). The package **spikeSlabGAM** applies Bayesian variable selection, model choice and regularization for GAMM's (**?**).

Within R, regression models are most conveniently specified using its formula language. By reviewing the above mentioned packages for GAM's, we searched for existing structures that on the hand could support a translation method for R commands to **BayesX** interpretable syntax, and on the other hand, functionality that is already well established, such that applying **R2BayesX** only requires little additional effort for the common R user. Interestingly, most of the packages use different standards for incorporating additive (smooth/special) terms in model formulas and the model frame building process, while a popular name convention for a model term constructor function is "`s`", as it is used in package **gam**, **mgcv** and **VGAM**. However, the implementations are again more or less different from each other, i.e. loading packages simultaneously will result in conflicts. The most flexible project seems to be package **mgcv**. Here, similar to package **gam** and **VGAM**, function `s()` does not evaluate design or penalty matrices, but it returns a smooth term definition object of class `"xx.smooth.spec"`, where `"xx"` may be specified by the user. To set up a model with a user defined smooth, a method for the S3 generic function `smooth.construct()`, that returns a design matrix etc., needs to be supplied. Since implementation of additional model terms is also a concern for

R2BayesX and function `s()` is a quite parsimonious solution, we adopted its functionality and developed methods for a new generic function `bayesx.construct()`, that returns the required command for a particular smooth term in **BayesX**. To give an example, we generate a call to function `s()` with some covariate `x` specifying a P-spline term and return the **BayesX** command with

```
R> bayesx.construct(s(x, bs = "ps"))

[1] "x(psplinerw2,nrknots=7,degree=4)"
```

Given an R model formula, the specified terms are translated one after another and finally merged into a complete program which may be sent to **BayesX**. Next to the integration of function `s()`, a more **BayesX** alike model term constructor function is provided. The advantage is that a common **BayesX** user may specifiy models in a more familiar way, which will make a change to **R2BayesX** more easy. The function is called `sx()` and is described in Section~**??**. To avoid package conflicts the new function simply interfaces function `s()`. Moreover, a special constructor function for random effect terms is provided.

The sequence of functions that are consecutively called using the interface is the following: A final program file is generated by first applying function `parse.bayesx.input()` on the R input parameters, including the model `formula`, `data`, etc. The returned object is then further processed with function `write.bayesx.input()`, utilizing the methods described above, as well as setting up the necessary data files (folders) to be used with **BayesX**. Afterwards, function `run.bayesx()` executes the program through a call to function `system()`. The output files returned by the binary are imported into R using function `read.bayesx.output()`. Hence, it is also possible to run and read already existing **BayesX** program and output files, see Section~**??** and the **R2BayesX** manuals for a detailed description. The object returned by function `read.bayesx.output()` is a list of class `"bayesx"`, wherefore a set of base R functions and methods described in Table~**??**, amongst others, is available. The returned fitted model term objects are also assigned with suitable classes which have plotting methods. Particular effort has been raised on the development of easy to use map effect plots using color legends, see also Section~**??** and Section~**??**.

Functions `parse.bayesx.input()`, `write.bayesx.input()`, `run.bayesx()` and `read.bayesx.output()` are combined in the front end main model fitting function `bayesx()` presented in Section~**??**.

5. User interface

5.1. Installing the BayesX binary from R

Before STAR models can be fitted with package **R2BayesX** from R, the binary command-line version of the program **BayesX** needs to be installed and linked to R. The recommended option on UNIX and Windows systems is to auto-compile/install the **BayesX** command-line binary within R by calling the function `install.bayesx()`. Therefore package **R2BayesX** needs to be loaded:

```
R> library("R2BayesX")
```

Installing BayesX from R then ideally only requires running function

```
install.bayesx(inst.dir = NULL, source.dir = NULL, type = NULL)
```

where argument `inst.dir` is a path to a valid installation directory with user writing permissions (e.g. for Windows systems `inst.dir = "C:/BayesX"`). If `source.dir = NULL`, the necessary installation files will automatically tried to be downloaded from the **BayesX** homepage (url: http://www.stat.uni-muenchen.de/~bayesx). Otherwise `source.dir` specifies the path where `install.bayesx()` may either find the packed sources `bayesxsource.zip` (url: http://www.stat.uni-muenchen.de/~bayesx/install/bayesxsource.zip), or using Windows, the installer `BayesX_windows.exe` (url: http://www.stat.uni-muenchen.de/~bayesx/install/BayesX_windows.exe). If the corresponding file is available, `install.bayesx()` will then attempt to execute the Windows installation process as described in the Appendix~**??**, or to compile the sources as shown in Appendix~**??**, respectively. If argument `type` is set to `type = "sources"`, compilation of the sources using the GNU Compiler Collection (GCC) C compiler is forced on any platform.

5.2. Linking the BayesX binary to R

After the successful compilation/installation, usually the full path to the command-line binary needs to be declared to R at the beginning of every new session by setting

```
R> options(bayesx.bin = "/path/to/BayesX")
```

where `"/path/to/BayesX"` is e.g. the path provided to argument `inst.dir` of function `install.bayesx()`, with the name of the **BayesX** binary at last. On Windows platforms the name of the binary is `"bayesx.exe"`, on all other platforms commonly `"BayesX"`. Hence, on Windows systems the user may specify e.g.

```
R> options(bayesx.bin = "C:/BayesX/commandline/bayesx.exe")
```

To avoid setting the path to the binary each time R is starting, it is suggested to add the code above to the R startup profile site, see also Appendix~**??**. Afterwards, the function call

```
R> check.install.bayesx()
```

will check if **BayesX** is available from R.

5.3. Processing BayesX from R

The main model-fitting function in the package **R2BayesX** is called `bayesx()`. The arguments of `bayesx()` are

```
bayesx(formula, data, weights = NULL, subset = NULL,
  offset = NULL, na.action = na.fail, contrasts = NULL,
  family = "gaussian", method = "MCMC", control = bayesx.control(...),
  ...)
```

family	Response distribution	Link	method
"binomial"	binomial	logit	"MCMC" "REML" "STEP"
"binomialprobit"	binomial	probit	"MCMC" "REML" "STEP"
"gamma"	gamma	log	"MCMC" "REML" "STEP"
"gaussian"	Gaussian	identity	"MCMC" "REML" "STEP"
"multinomial"	unordered multinomial	logit	"MCMC" "REML" "STEP"
"poisson"	Poisson	log-link	"MCMC" "REML" "STEP"
"cox"	continuous-time survival data		"MCMC" "REML"
"cumprobit"	cumulative threshold	probit	"MCMC" "REML"
"multistate"	continuous-time multistate data		"MCMC" "REML"
"binomialcomploglog"	binomial	compl. log-log	"REML"
"cumlogit"	cumulative multinomial	logit	"REML"
"multinomialcatsp"	unordered multinomial (with category-specific covariates)	logit	"REML"
"multinomialprobit"	unordered multinomial	probit	"MCMC"
"seqlogit"	sequential multinomial	logit	"REML"
"seqprobit"	sequential multinomial	probit	"REML"

Table 1: Distributions implemented for methods "MCMC", "REML" and "STEP".

where the first two lines basically represent the standard model frame specifications (see **?**). However, the object supplied to argument `data` is not necessarily an R data object, it is also possible to provide a character string with a path to a dataset stored on disc, which may be reasonable when using large datasets. An example is given in Section~**??**. Additional contrast specifications for factor variables can be passed to argument `contrasts`. Using factors, we recommend deviation or effect coding (see function `contr.sum()`) rather than the usual dummy coding of factors as it typically improves convergence of estimation algorithms used in **BayesX**. The distribution assigned to the response may be set with argument `family`, the default is `family = "gaussian"`. Note that family objects are currently not supported by BayesX. Argument `method` determines the inferential concept used for estimation. Options are: Markov chain Monte Carlo simulation - `"MCMC"`, mixed model based estimation using restricted maximum likelihood/marginal likelihood - `"REML"` and penalized likelihood including model selection - `"STEP"`. An overview of all available distributions for the different methods is given in Table~**??**. The last argument specifies several parameters controlling the processing of the **BayesX** binary that are arranged by function `bayesx.control()`. Note that all additional controlling arguments are automatically parsed within function `bayesx()` using the dot dot dot argument "`...`", which is sent to `bayesx.control()`. The most important parameters for the different methods are listed in Table~**??**.

The returned fitted model object is a list of class `"bayesx"`, which is supported by several standard extractor functions, such as `plot()` and `summary()`. For models estimated using method `"REML"`, function `summary()` generates summary statistics similar to objects returned

from the main model fitting function `gam()` of the **mgcv** package. For `"MCMC"` estimated models, the mean, standard deviation and quantiles of parameter samples are provided. Using `"STEP"`, the parametric part of the summary statistics is represented like `"MCMC"`, i.e. if computed, the confidence bands are based on an MCMC algorithm subsequent to the model selection, while the remaining summary is similar to `"REML"`. The implemented S3 methods for plotting fitted term objects are quite flexible, i.e., depending on the term structure, the generic function `plot()` calls one of the following functions: for 2d plots function `plot2d()` or `plotblock()` (for factors, unit- or cluster specific plots, draws a block for every estimated parameter including mean and credible intervals), for perspective or image and contour plots function `plot3d()`, map effects plots are produced by function `plotmap()`, with or without colorlegends drawn by function `colorlegend()`, amongst others. See Table~**??** in the Appendix for an overview of the most important arguments for the plotting functions. In some situations it may be useful to inspect the log file generated by the **BayesX** binary. The file can either be viewed directly during processing if argument `verbose` is set to `TRUE` in function `bayesx.control()`, or extracted from the fitted model object using function `bayesx_logfile()`. For MCMC post estimation diagnosis, besides the implemented trace and autocorrelation plots, samples of the parameters may also be extracted using function `samples()`. The sampling paths are provided as a data frame, and hence may e.g. be converted to objects of class `"mcmc"` using the **coda** package (**?**) for further analysis. In addition, an R script for the estimated model, including function calls for saving, loading, plotting of term effects and diagnostic plots, may be generated using function `getscript()`. The produced R script may be useful for less experienced users of the package to get a quick overview of post estimation commands. Moreover, the script facilitates the final preparation of plots and diagnostics to be included in publications. A list of all available functions and methods of package **R2BayesX** can be found in Table~**??**.

`method`	Parameter	Description
`"MCMC"`	`iterations`	integer, sets the number of iterations for the sampler, default: 12000.
	`burnin`	integer, sets the burn-in period of the sampler, default: 2000.
	`step`	integer, defines the thinning parameter for MCMC simulation. E.g.,`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, default: 10.
`"REML"`	`eps`	numeric, 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: 0.00001.
	`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.

continued on next page

continued from previous page

"STEP"	`algorithm`	character, specifies the selection algorithm. Possible values are `"cdescent1"` (adaptive algorithms see subsection 6.3 in **?**), `"cdescent2"` (adaptive algorithms 1 and 2 with backfitting, see remarks 1 and 2 of section 3 in **?**), `"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` function of S-plus, see **?**), default: `"cdescent1"`.
	`criterion`	character, specifies the goodness of fit criterion, possible criterions are: `"MSEP"` (divides the data randomly into a test- and validation dataset. The test dataset is used to estimate the models and the validation dataset is used to estimate the mean squared prediction error (MSEP) which serves as the goodness of fit criterion to compare different models), `"GCV"` (Generalized Cross Validation based on deviance residuals, see e.g. **?**), `"GCVrss"` (Generalized Cross Validation based on residual sum of squares, see e.g. **?**), `"AIC"` (Akaike Information Criterion, see e.g. **?**), `"AIC_imp"` (improved AIC with bias correction for regression models, see e.g. **?**), `"BIC"` (Bayesian information criterion, see e.g. **?**), `"MSEP"`, `"CV5"` (5–fold cross validation, see e.g **?**), `"CV10"` (5–fold cross validation, see e.g **?**) and `"AUC"` (area under the ROC curve, binary response only), default: `"AIC_imp"`.
	`startmodel`	character, defines the start model for variable selection. Options are `"linear"` (start model with degrees of freedom euqal to one for model terms), `"empty"` (empty model containing only an intercept), `"full"` (most complex possible model) and `"userdefined"` (start model is specified by the user), default: `"linear"`.

Table 2: Most important controlling parameters for the different methods using function `bayesx()`. (A detailed documentation is provided in the manual of function `bayesx.control()` of package **R2BayesX**.)

5.4. Available additive terms

In package **R2BayesX**, the main constructor function for specifying additive terms in STAR `formula`s is called `sx()`. The function is basically an interface to the term constructor function `s()` of package **mgcv**, also see Section~**??**.

The arguments of function `sx()` are

```
sx(x, z = NULL, bs = "ps", by = NA, ...)
```

where `x` represents the covariate that is used for univariate and `z` for bivariate model terms. Argument `bs` chooses the basis/type of the term, possible options are shown in Table~**??**.

Function	Description
`print()`	simple printed display of the initial call and some additional information of the fitted model.
`summary()`	returns an object of class `"summary.bayesx"` containing the relevant summary statistics (which has a `print()` method).
`coef()`	extracts coefficients of the linear modeled terms.
`confint()`	compute confidence intervals of linear modeled terms if `method = "REML"`, for `"MCMC"` the quantiles of the coefficient samples according to a specified probability level are computed.
`cprob()`	extract contour probabilities of a particular P-spline term, only meaningful if `method = "MCMC"` and argument `contourprob` is specified as an additional argument in the term constructor function `sx()`, or within argument `xt` in function `s()`. E.g. in the introductory example, contour probabilities for the term using covariate `mbmi` are estimated with `s(mbmi, bs = "ps", xt = list(contourprob = 4))` (see also Section~**??**).
`fitted()`	fitted values of either the mean and linear predictor, or a selected model term.
`bayesx_logfile()`, `bayesx_prgfile()`, `bayesx_runtime()`	extracts the internal **BayesX** log file, the program file and the overall runtime of the binary.
`residuals()`	extract model or partial residuals for a selected term.
`samples()`	extract samples of parameters from MCMC simulation.
`terms()`	extract terms of model components.
`model.frame()`	extract/generate a model frame.
`logLik()`	extract fitted log-likelihood, only if `method = "REML"`.
`plot()`	either model diagnostic plots or effect plots of particular terms.
`getscript()`	generate an R script for term effect, diagnostic plots and model summary statistics.
`AIC()`, `BIC()`, `DIC()`, `GCV()`	computes information criteria, availability is dependent on the `method` used.

Table 3: Functions and methods for objects of class `"bayesx"`. (A detailed documentation is provided in the manual of package **R2BayesX**.)

Note that it is possible to specify all term type versions presented in the table, the short and the original ones that are used within **BayesX**, e.g. `bs = "ps"` or `bs = "psplinerw2"` result in the same type of model term. A numeric or a factor variable can be provided to argument `by` to estimate varying coefficient terms, where the effect of the variable provided to `by` varies over the range of the covariate(s) of this term. The dot dot dot "`...`" argument is used to specify term specific control parameters or additional geographical information. In the example Section~**??**, to modify the degree and the inner knots for the P-spline term `sx(mbmi)`, the user may e.g. type `sx(mbmi, degree = 2, knots = 10)`. For supplying additional boundary or graph files (see function `read.bnd()`, `read.gra()`, `shp2bnd()` and `bnd2gra()` for importing and creation of spatial objects that may be sent to **BayesX**), that are used to compute suitable neighborhood penalty matrices for terms using Markov random field priors, or to calculate

bs	Description
"rw1", "rw2"	zero degree P-splines: defines a zero degree P-spline with first or second order difference penalty. A zero degree P-spline typically estimates for every distinct covariate value in the dataset 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.
"season"	seasonal effect of a time scale.
"ps", "psplinerw1", "psplinerw2"	P-spline with first or second order difference penalty.
"te", "pspline2dimrw1"	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.
"kr", "kriging"	kriging with stationary Gaussian random fields.
"gk", "geokriging"	geokriging with stationary Gaussian random fields: estimates a stationary Gaussian random field based on the centroids of a map object provided in boundary format (see function `read.bnd()` and `shp2bnd()`) as an additional argument named `map` within function `sx()`, or supplied within argument `xt` when using function `s()`, e.g. `xt = list(map = MapBnd)`.
"gs", "geospline"	geosplines based on two-dimensional P-splines with first order random walk penalty: 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 of a map object provided in boundary format (see function `read.bnd()` and `shp2bnd()`) as an additional argument named `map` within function `sx()`, or supplied within argument `xt` when using function `s()`, e.g. `xt = list(map = MapBnd)`.
"mrf", "spatial"	Markov random fields: defines a Markov random field prior for a spatial covariate, where geographical information is provided by a map object in boundary or graph file format (see function `read.bnd()`, `read.gra()` and `shp2bnd()`), as an additional argument named `map` within function `sx()`, or supplied within argument `xt` when using function `s()`, e.g. `xt = list(map = MapBndorGra)`.

continued on next page

continued from previous page	
`"bl"`, `"baseline"`	nonlinear baseline effect in hazard regression or multi-state models: defines a P-spline with second order random walk penalty for the parameters of the spline for the log-baseline effect $\log(\lambda(time))$.
`"factor"`	special **BayesX** specifier for factors, especially meaningful if `method = "STEP"`, since the factor term is then treated as a full term, which is either included or removed from the model.
`"ridge"`, `"lasso"`, `"nigmix"`	shrinkage of fixed effects: defines a shrinkage-prior for the corresponding parameters γ_j, $j = 1, \ldots, q$, $q \geq 1$ of the linear effects $x_1, \ldots, x_q$. There are three priors possible: ridge-, lasso- and Normal Mixture of inverse Gamma prior.

Table 4: Possible **BayesX** model terms within function `sx()` and `s()`.

the centroids of particular regions for geosplines and geokriging terms, an argument named `map` needs to be provided. For instance, the necessary boundary file `ZambiaBnd` for the geokriging term in Section~**??** is included with `sx(district, bs = "gk", map = ZambiaBnd)`. Information about all possible extra arguments for a particular term basis/type can be looked up using function `bayesx.term.options()`, e.g. possible options for P-splines using `"MCMC"` are printed to the console by

```
R> bayesx.term.options(bs = "ps", method = "MCMC")

possible options for 'bs = "ps"':

     degree: the degree of the B-spline basis functions.
           Default: integer, 'degree = 3'.

      knots: number of inner knots.
           Default: integer, 'knots = 20'.
```

For reasons of simplicity only the first two additional controlling arguments are shown.

Some care has to be taken with the identifiability of varying coefficients terms. The standard in **BayesX** is to center nonlinear main effects terms around zero whereas varying coefficient terms are not centered. This makes sense since main effects nonlinear terms are not identifiable and varying coefficients terms are usually identifiable. However, there are situations where a varying coefficients term is not identifiable. Then the term must be centered. Since centering is not automatically accomplished it has to be enforced by the user by adding option `center = TRUE` in function `sx()`. To give an example, the varying coefficient terms in $\eta = \ldots + g_1(z_1)z + g_2(z_2)z + \gamma_0 + \gamma_1 z + \ldots$ are not identified, whereas in $\eta = \ldots + g_1(z_1)z + \gamma_0 + \ldots$, the varying coefficient term is identifiable. In the first case, centering is necessary, in the second case, it is not.

As mentioned above, users may optionally call the constructor function `s()` directly. The usage of `s()` in **R2BayesX** is in principle similar to package **mgcv**. The applicable arguments

are

```
s(..., k = -1, bs = "ps", m = NA, by = NA, xt = NULL)
```

Within `s()`, the list of covariates used for the model term is set with argument "`...`". For instance, in the example of Section~**??**, the term for the body mass index of the mother may also be included in the model formula by `s(mbmi)`, a term with two covariates is specified e.g. with `s(mbmi, agechild)`. Here, the parameter `k` controls the dimension of the basis used for smooth terms. Setting argument `m` is only meaningful for P-splines, i.e. `bs = "ps"`, and controls the degree of the B-spline basis functions and the order of the difference penalty. E.g. a B-spline of degree 3 with a 2nd order difference penalty is set with `s(mbmi, bs = "ps", m = c(1, 2))` (note that argument `m` is slightly different than argument `degree` in function `sx()` using P-splines, see also the manual of `s()`). Argument `by` is used in the same way as for `sx()`. The additional parameters that are specified by argument "`...`" in function `sx()`, may be set in `s()` within argument `xt`, e.g. similar to the example above, the boundary object `ZambiaBnd` is supplied with `s(district, bs = "gk", xt = list(map = ZambiaBnd))`. Besides `s()`, **R2BayesX** also supports calls to the tensor product constructor function `te()`, however, only a small set of the features of this function is supported.

In addition to `sx()`, `s()` and `te()`, unit- or cluster specific unstructured (random) effects may be incorporated into the model `formula` using the random effects term constructor function `r()`. The arguments of `r()` are

```
r(id, by = NA, xt = NULL)
```

where `id` is the unit or cluster identification covariate the random effects should be estimated for. E.g. the model formula from the introduction could be extended by a random effects term for the districts in Zambia with `r(district)`. Argument `by` takes covariates for which random slopes may be estimated. Argument `xt` is used in the same way as described for function `s()`, i.e. similar to the above, e.g. hyperpriors a and b for the variance parameter are set with `r(district, xt = list(a = 0.0001, b = 0.0001))`.

5.5. Additional options

For most practical purposes fitting models with function `bayesx()` may be sufficient. However, the interfacing functions that are consecutively called within `bayesx()` can also be used independently. This could be meaningful for two reasons: First, users may want to use already existing **BayesX** program files wherefore a new setup within R is not required, and secondly, there might be a need for automated importing of **BayesX** output files into R for further analysis.

Function `run.bayesx()` is used to run an arbitrary **BayesX** program file. The arguments of `run.bayesx()` are

```
run.bayesx(dir, prg.name = "bayesx.estim.input.prg",
  verbose = TRUE, bin = getOption("bayesx.bin"))
```

where `dir` is a character string with the directory the program file is stored in and `prg.name` is the name of the corresponding program file. During processing of **BayesX** several informations

will be printed to the R console if `verbose = TRUE`. Argument `bin` specifies the location of the **BayesX** `bin`ary the program file is sent to, see also Section~**??**. The function returns a list including the log file returned by **BayesX** as well as information on the total runtime.

Model output files are imported using function

```
read.bayesx.output(dir, model.name = NULL)
```

Here, `dir` is again a directory and `model.name` the name of the model the files are imported for, also provided as character strings. Note that the function will search for all different BayesX estimated models in the declared directory if argument `model.name` is set to `NULL`. The returned object is also of class `"bayesx"`, i.e. all the functions and methods described in Table~**??** may be applied.

Another noteworthy feature of package **R2BayesX** is the internal handling of data. **BayesX** uses numerically efficient algorithms including sparse matrix computations which in principle allow to estimate models using large datasets. Moreover, the number of different observations for particular covariates is usually much smaller than the total number of observations. That is, the output files returned by the binary only include estimates for unique covariate values. Since these files typically reserve much less disc space, importing the fitted model objects into R using `read.bayesx.output()` is straightforward in most cases, whereas handling the complete dataset within R may be problematic. As mentioned in Section~**??**, users can exploit this by providing a character string to argument `data` in function `bayesx()`, which includes the path to a dataset instead of an R data object. As a consequence, this dataset will not be loaded within R and is only used internally by the **BayesX** binary. To give an example, we generate a large dataset that might produce problems with R's memory allocation using a model fitting function, especially if the model contains a large number of parameters. Therefore, we store the data on disc in the temporary folder of the running session with

```
R> set.seed(321)
R> file <- paste(tempdir(), "/data.raw", sep = "")
R> n <- 5e+06
R> dat <- data.frame(x = rep(runif(1000, -3, 3), length.out = n))
R> dat$y <- with(dat, sin(x) + rnorm(n, sd = 2))
R> write.table(dat, file = file, quote = FALSE, row.names = FALSE)
```

This produces a dataset of approximately 170Mb with only 1000 unique observations for covariate `x`. The path to the dataset is stored in object `file` and is provided to argument `data` in the function call

```
R> b <- bayesx(y ~ sx(x), family = "gaussian", method = "MCMC",
+    iterations = 3000, burnin = 1000, step = 2, predict = FALSE,
+    data = file, seed = 123)
```

For illustration purposes, the number of iterations is only set to 3000. Note that argument `predict` is set to `FALSE`, i.e. only output files of estimated effects will be returned, otherwise an expanded dataset using all observations would be written in the output directory, also containing the data used for estimation. The runtime of this example is about 4 1/2 hours

```
R> bayesx_runtime(b)

    user   system  elapsed
16442.12     7.56 16461.33
```

on a Linux system with an Intel 2.33GHz Dual Core processor, while the returned object `b` uses

```
R> print(object.size(b), units = "Mb")

0.4 Mb
```

of memory size.

6. STAR models in practice

The focus of this section is on demonstrating the various features of the **R2BayesX** package. Therefore, the examples provided are replicate analysis taken from **?** and **?**. The presented datasets have been added to package **R2BayesX**, ensuring straightforward traceability of the following code. In the first example, a Gaussian regression model is estimated using Markov chain Monte Carlo simulation. The second example covers estimation based on mixed model technology, where a cumulative threshold model is assigned for the response variable (see **?** and **?** for cumulative threshold models). The last example illustrates the approach of the stepwise algorithm for model and variable selection.

6.1. Childhood malnutrition in Zambia: Analysis with MCMC

This analysis has already been conducted by **?** and has also been used as a demonstrating example in **?**. Stunting is one of the leading drivers of a number of problems development countries are faced with, for instance, a direct consequence of stunting is a high mortality rate. Here, the primary interest is to model the dependence of stunting of newborn children, with an age ranging from 0 to 5 years, on covariates such as the body mass index of the mother, the age of the child and others presented in Table~**??**. The response `stunting` is standardized in terms of a reference population, i.e in this dataset stunting for child i is represented by

$$\texttt{stunting}_i = \frac{AI_i - MAI}{\sigma},$$

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

Following **?**, we estimate a structured additive regression model with predictor

$$\begin{aligned}\eta \;=\; & \gamma_0 + \gamma_1 \texttt{memploymentyes} + \gamma_2 \texttt{urbanno} + \gamma_3 \texttt{genderfemale} + \\ & \gamma_4 \texttt{meducationno} + \gamma_5 \texttt{meducationprimary} + \\ & f_1(\texttt{mbmi}) + f_2(\texttt{agechild}) + f_{str}(\texttt{district}) + f_{unstr}(\texttt{district}) \qquad (2)\end{aligned}$$

where `memploymentyes` is the deviation (effect) coded version of covariate `memployment`, generated with function `contr.sum()` by setting the contrasts argument of the factor variable,

Variable	Description
`stunting`	standardized Z-score for stunting.
`mbmi`	body mass index of the mother.
`agechild`	age of the child in months.
`district`	district where the mother lives.
`memployment`	mother's employment status with categories 'yes' and 'no'.
`meducation`	mother's educational status with categories for no education or incomplete primary 'no', complete primary but incomplete secondary 'primary' and complete secondary or higher 'secondary'.
`urban`	locality of the domicile with categories 'yes' and 'no'.
`gender`	gender of the child with categories 'male' and 'female'.

Table 5: Variables in the dataset on childhood malnutrition in Zambia.

i.e. `memploymentyes` contains of values -1, corresponding to 'yes', and 1, 'no' respectively, likewise for covariates `genderfemale`, `urbanno`, `meducationno` and `meducationprimary`. As mentioned in the introduction, functions f_1 and f_2 of the continuous covariates `agechild` and `mbmi` are assumed to have a possibly nonlinear effect on `stunting` and are therefore modeled with P-splines. Furthermore, the spatial effect is decomposed into a structured effect f_{str}, modeled by a Gaussian Markov random field, and an unstructured effect f_{unstr}, using a random effects term for the districts in Zambia.

The data for this analysis is provided in the **R2BayesX** package and can be loaded with

```
R> data("ZambiaNutrition", package = "R2BayesX")
```

Since function f_{str} is modeled by a Markov random fields term, **BayesX** needs information about the district neighborhood structure, which e.g. is enclosed in the file

```
R> data("ZambiaBnd", package = "R2BayesX")
```

The object `ZambiaBnd` has class `"bnd"` and is basically a `list()` of polygon matrices, with x- and y-coordinates of the boundary points in the first and second column respectively. To read in an arbitrary boundary file into R function `read.bnd()` can be used. With the information of the boundary file **BayesX** may compute an appropriate adjacency matrix, allowing for a smoothly varying effect of the neighboring regions. There is a generic plotting method implemented for objects of class `"bnd"`, which in principle calls function `plotmap()`. E.g. a simple map, as shown in Figure~**??**, of the districts in Zambia is drawn by typing

```
R> plot(ZambiaBnd)
```

Having loaded the necessary files, the model `formula` is specified with

```
R> f <- stunting ~ memployment + urban + gender + meducation +
+    sx(mbmi) + sx(agechild) + sx(district, bs = "mrf", map = ZambiaBnd) +
+    r(district)
```

The model is then fitted using MCMC by calling

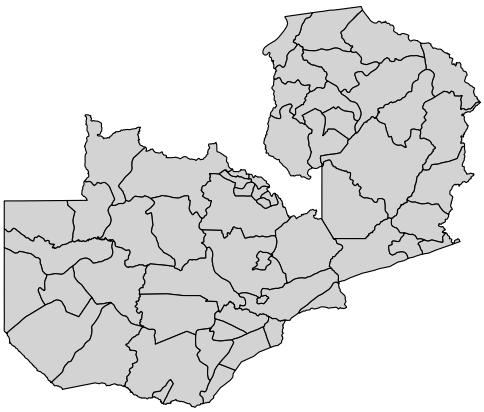

Figure 2: Example on childhood malnutrition: A simple map of the districts in Zambia.

```
R> zm <- bayesx(f, family = "gaussian", method = "MCMC",
+    iterations = 12000, burnin = 2000, step = 10,
+    seed = 123, data = ZambiaNutrition)
```

Argument `iterations`, `burnin` and `step` set the number of iterations of the MCMC simulation, the burnin period, which will be removed from the generated samples, and the step length for which samples should be stored, i.e. if `step = 10`, every 10th sampled parameter will be saved. In most applications 12000 iterations should be enough for a valid fit with sufficiently small autocorrelations of stored parameters, at least in the model building stage. However, it is absolutely necessary to take a look at sampled parameters and autocorrelation functions to check the mixing behavior (see below). Moreover, it is generally advisable to specify a higher number of iterations for the final model that appears in publications. Argument `seed` sets the state of the random number generator in **BayesX**, which is meaningful for exact replication of the examples.

After the model has been successfully fitted, summary statistics of the MCMC estimated model object may be printed with

```
R> summary(zm)

Call:
bayesx(formula = f, data = ZambiaNutrition, family = "gaussian",
    method = "MCMC", iterations = 12000, burnin = 2000, step = 10)

Fixed effects estimation results:

Parametric Coefficients:
                    Mean      Sd    2.5%     50%   97.5%
(Intercept)       0.0991  0.0475  0.0046  0.1018  0.1863
memploymentno    -0.0084  0.0135 -0.0359 -0.0084  0.0170
urbanno          -0.0895  0.0217 -0.1306 -0.0893 -0.0450
genderfemale      0.0582  0.0133  0.0320  0.0578  0.0850
meducationno     -0.1722  0.0269 -0.2248 -0.1719 -0.1163
```

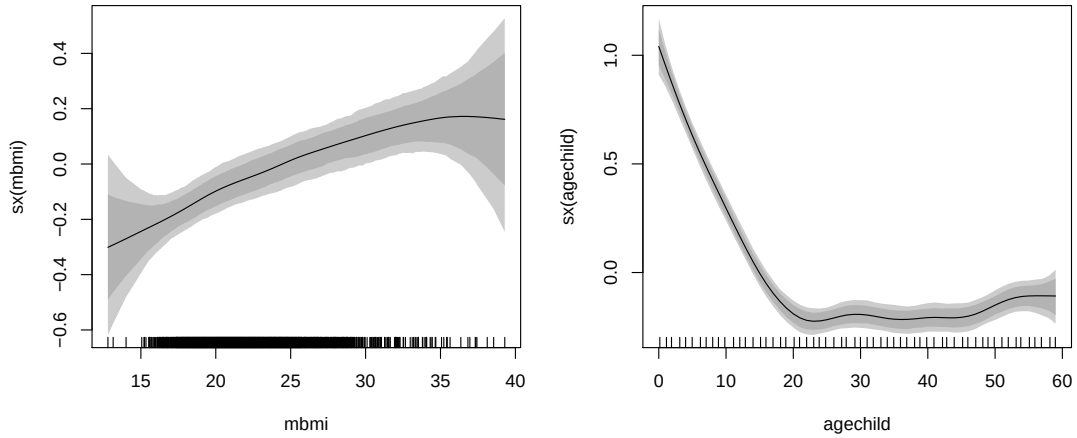

Figure 3: Example on childhood malnutrition: Effect of the body mass index of the child's mother and of the age of the child together with pointwise 80% and 95% credible intervals.

```
meducationprimary -0.0611  0.0262 -0.1115 -0.0614 -0.0091

Smooth terms variances:
               Mean     Sd   2.5%    50%  97.5%    Min    Max
sx(agechild) 0.0062 0.0060 0.0014 0.0042 0.0233 0.0007 0.0570
sx(district) 0.0360 0.0191 0.0094 0.0325 0.0813 0.0025 0.1784
sx(mbmi)     0.0019 0.0028 0.0003 0.0011 0.0081 0.0002 0.0468

Random effects variances:
              Mean     Sd   2.5%    50%  97.5%    Min    Max
r(district) 0.0076 0.0064 0.0008 0.0062 0.0226 0.0003 0.0701

Scale estimate:
         Mean     Sd   2.5%    50% 97.5%
Sigma2 0.8023 0.0163 0.7721 0.8017 0.836

N = 4847  burnin = 2000  DIC = 4899.506  pd = 50.41262
method = MCMC  family = gaussian  iterations = 12000  step = 10
```

which typically includes mean, standard deviation and quantiles of sampled linear effects, smooth terms variances and random effects variances, as well as goodness of fit criteria and some other information about the model. The estimated effects for covariates `agechild` and `mbmi` may then be visualized with

```
R> plot(zm, term = c("sx(mbmi)", "sx(agechild)"))
```

and are shown in Figure~**??**.

A visual representation of the structured and unstructured spatial effect can be obtained in two ways. Using the plain plot function

```
R> plot(zm, term = c("sx(district)", "r(district)"))
```

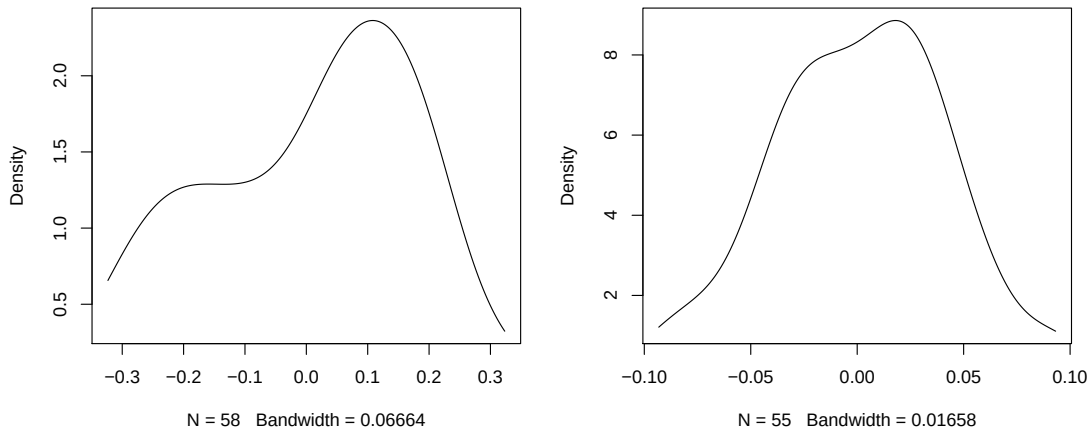

Figure 4: Example on childhood malnutrition: Kernel density estimates of the mean of the structured, left panel, and the unstructured spatial effect, right panel respectively.

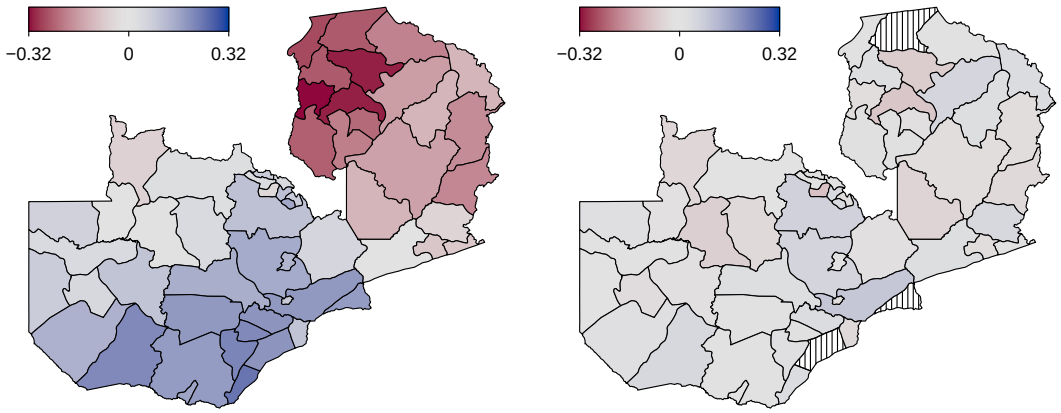

Figure 5: Example on childhood malnutrition: Estimated mean effect of the structured spatial effect (left panel), together with the unstructured spatial effect using the color and legend scaling of the structured effect (right panel).

produces a kernel density estimator of the posterior mean of the effects, see Figure **??**. The kernel densities reveal the general form of the random effects distributions. To view the spatial structure of the correlated effect we have to use the plot function in combination with the boundary object `ZambiaBnd`:

```
R> plot(zm, term = "sx(district)", map = ZambiaBnd)
```

The generated map effect plot is shown in Figure~**??**. As a default the districts of Zambia are colored in a symmetrical range within $+-max(|min(pmean)|, |max(pmean)|)$. In many situations the visual impression of the colored map is problematic. This is primarily the case if there are some districts with extraordinarily high posterior means compared to the rest of the districts. Then the map is dominated by the colors of these outlying districts. A more informative map may be obtained by restricting the range of the plotting area using the range option. For the Zambia data the corresponding random effects are comparably

symmetric and without outlying districts such that the plot function with default options produces fairly informative maps. To demonstrate the range option we draw the unstructered random effect and the legend range within the same range as the structured random effect

```
R> range <- lrange <- c(-0.32, 0.32)
R> plot(zm, term = "r(district)", map = ZambiaBnd,
+    range = range, lrange = lrange)
```

The resulting map is also shown in Figure~**??** (right panel). Using for both the structured and the unstructered effect the same scale is useful for comparison. In most cases one of the two effects clearly dominates the other. In our case the structured spatially correlated effect clearly exceeds the unstructured effect.

For MCMC post estimation diagnosis, it is also possible to extract sampling paths of parameters with function `samples()`, or to plot the samples directly. For instance, coefficient sampling paths for term `sx(mbmi)` are displayed with

```
R> plot(zm, term = "sx(mbmi)", which = "coef-samples")
```

see Figure~**??**. The plot of sampled parameters should ideally show white noise, i.e. more or less uncorrelated samples that show no particular pattern. In our case the samples are exactly as they should be. In addition, autocorrelation functions may be drawn, e.g. for the variance samples of term `sx(mbmi)`, by typing

```
R> plot(zm, term = "sx(mbmi)", which = "var-samples", acf = TRUE)
```

The maximum autocorrelation of all sampled parameters in the model are displayed with

```
R> plot(zm, which = "max-acf")
```

Autocorrelations for all lags should be close to zero as is mostly the case in our example. See Figure~**??**, for the autocorrelation plots. The plot of maximum autocorrelations over all model parameters suggests to use a larger number of iterations in a final run (e.g. 22000 ore even 32000 iterations).

In some situations problems may occur during processing of the **BayesX** binary, that are not automatically detected by the main model fitting function `bayesx()`. Therefore the user may inspect the log-file generated by the binary in two ways: Setting the option `verbose = TRUE` in `bayesx.control()` will print all information produced by **BayesX** simultaneously at runtime. The option is especially helpful if **BayesX** crashes. Another way to obtain the log-file is to use function `bayesx_logfile()` if **BayesX** successfully finished processing. In this example the log-file may be printed with

```
R> bayesx_logfile(zm)

> bayesreg b
> map ZambiaBnd
> ZambiaBnd.infile using /tmp/Rtmpa3Z6WF/bayesx/ZambiaBnd.bnd
NOTE: 57 regions read from file /tmp/Rtmpa3Z6WF/bayesx/ZambiaBnd.bnd
> dataset d
```

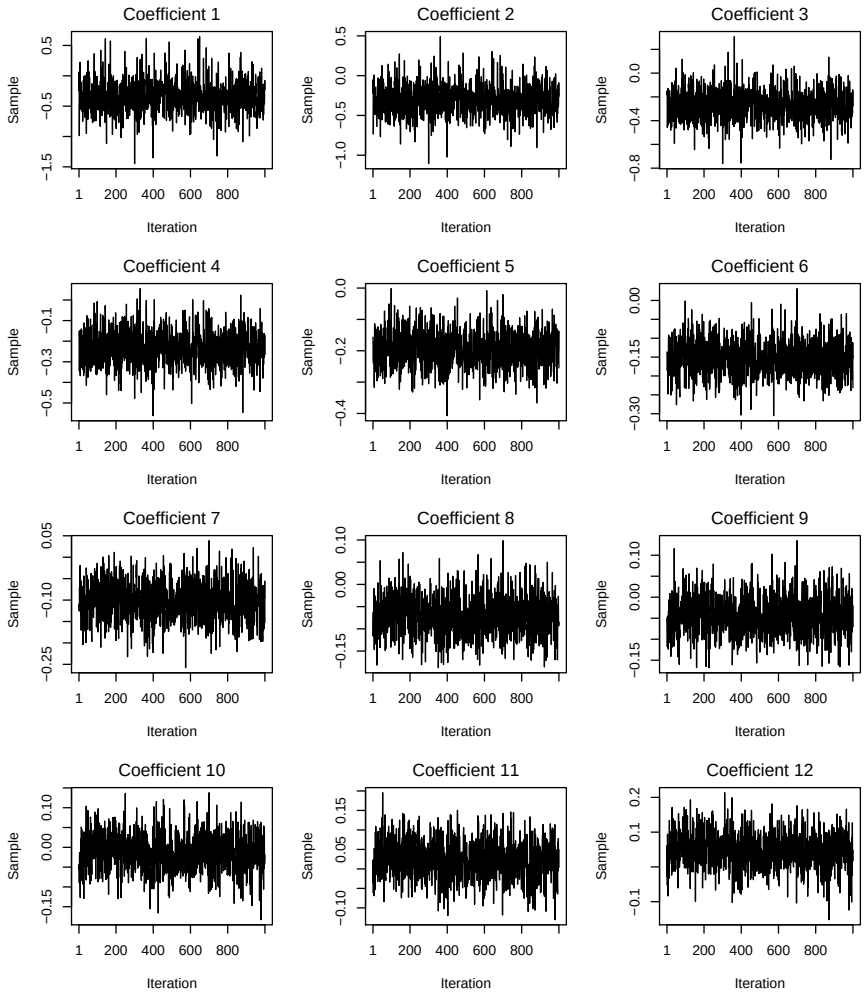

Figure 6: Example on childhood malnutrition: Sampling paths of the first 12 coefficients of term `sx(mbmi)`.

```
> d.infile using /tmp/Rtmpa3Z6WF/bayesx/bayesx.estim.data.raw
NOTE: 14 variables with 4847 observations read from file
/tmp/Rtmpa3Z6WF/bayesx/bayesx.estim.data.raw

> b.outfile = /tmp/Rtmpa3Z6WF/bayesx/bayesx.estim
> b.regress stunting = mbmi(psplinerw2,nrknots=20,degree=3) +
    agechild(psplinerw2,nrknots=20,degree=3) + district(spatial,map=ZambiaBnd) +
    district(random) + memploymentyes + urbanno + genderfemale + meducationno +
    meducationprimary, family=gaussian iterations=12000 burnin=2000 step=10
    setseed=123 predict using d
NOTE: no observations for region 11
NOTE: no observations for region 84
NOTE: no observations for region 96
```

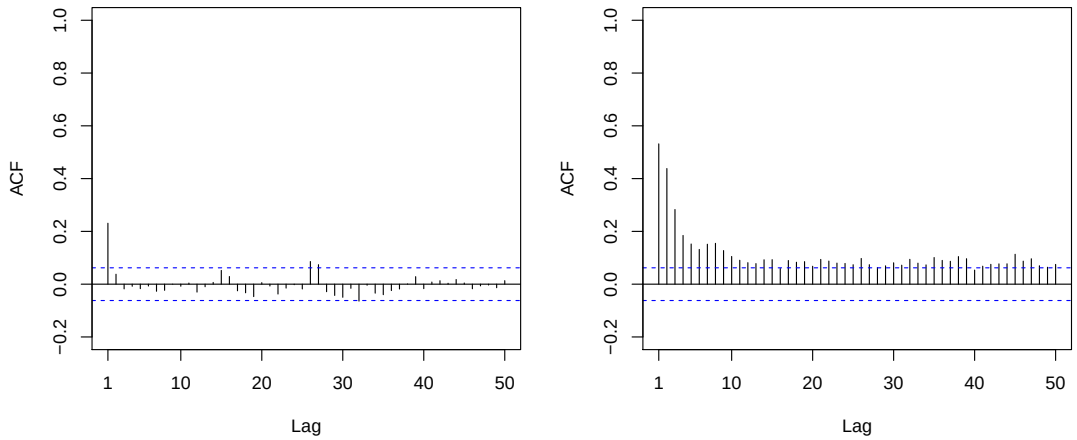

Figure 7: Example on childhood malnutrition: Autocorrelation function of the samples of the variance parameter of term `sx(mbmi)`, left panel, maximum autocorrelation of all parameters of the model, right panel respectively.

```
BAYESREG OBJECT b: regression procedure

GENERAL OPTIONS:

  Number of iterations:  12000
  Burn-in period:        2000
  Thinning parameter:    10


RESPONSE DISTRIBUTION:

  Family: Gaussian
  Number of observations: 4847
  Number of observations with positive weights: 4847
  Response function: identity
  Hyperparameter a: 0.001
  Hyperparameter b: 0.001
```

To simplify matters only a fragment of the log-file is shown in the above. The log-file typically provides information on the used data, model specifications, algorithms and possible error messages.

6.2. Forest health dataset: Analysis with REML

The dataset on forest health comprises information on the defoliation of beech trees, which serves as an indicator of overall forest health here. The data was collected annually from 1980 to 1997 during a project of visual inspection of trees around Rothenbuch, Germany, see **?**, and is discussed in detail in **?**. In this example, the percentage rate of defoliation of each tree is aggregated into three ordinal categories, which are modeled in terms of covariates

characterizing the stand and site of a tree. In addition, temporal and spatial information is available, see also Table~**??**.

Variable	Description
`id`	tree location identification number.
`year`	year of census.
`defoliation`	percentage of tree defoliation in three ordinal categories, 'defoliation $< 12.5\%$', '$12.5\% \leq$ defoliation $< 50\%$' and 'defoliation $\geq 50\%$'.
`age`	age of stands in years.
`canopy`	forest canopy density in percent.
`inclination`	slope inclination in percent.
`elevation`	elevation (meters above sea level).
`soil`	soil layer depth in cm.
`ph`	soil pH at 0-2cm depth.
`moisture`	soil moisture level with categories 'moderately dry', 'moderately moist' and 'moist or temporarily wet'.
`alkali`	proportion of base alkali-ions with categories 'very low', 'low', 'high' and 'very high'.
`humus`	humus layer thickness in cm.
`stand`	stand type with categories 'deciduous' and 'mixed'.
`fertilized`	fertilization applied with categories 'yes' and 'no'.

Table 6: Variables in the forest health dataset.

Similar to **?**, we start with a threshold model and cumulative logit link, with $P(\texttt{defoliation}_{it} \leq r)$ of tree i at time t, $r = 1, 2$, and the additive predictor

$$\eta_{it}^{(r)} = f_1(\texttt{age}_{it}) + f_2(\texttt{inclination}_i) + f_3(\texttt{canopy}_{it}) + f_4(\texttt{year}) + f_5(\texttt{elevation}_i) + \mathbf{x}_{it}'\boldsymbol{\gamma}$$

where $f_1, \ldots, f_5$ are possibly nonlinear smooth functions of the continuous covariates and $\mathbf{x}_{it}'\boldsymbol{\gamma}$ comprises covariates with parametric effects using deviation (effect) coding for factor covariates.

To estimate the model within R the data is loaded and the model formula specified with

```
R> data("ForestHealth", package = "R2BayesX")
R> f <- defoliation ~  stand + fertilized +
+    humus + moisture + alkali + ph + soil +
+    sx(age) + sx(inclination) + sx(canopy) +
+    sx(year) + sx(elevation)
```

The covariates entering nonlinearly are again modeled by P-splines. The model is then fitted applying REML by assigning a cumulative logit model and calling

```
R> fm1 <- bayesx(f, family = "cumlogit",
+    method = "REML", data = ForestHealth)
```

After the estimation process has converged, the estimated effects of the nonparametric modeled terms may be visualized by typing

```
R> plot(fm1, term = c("sx(age)", "sx(inclination)",
+    "sx(canopy)", "sx(year)", "sx(elevation)"))
```

and are shown in Figure~**??**. In this example some contradictory results occur. The effect of

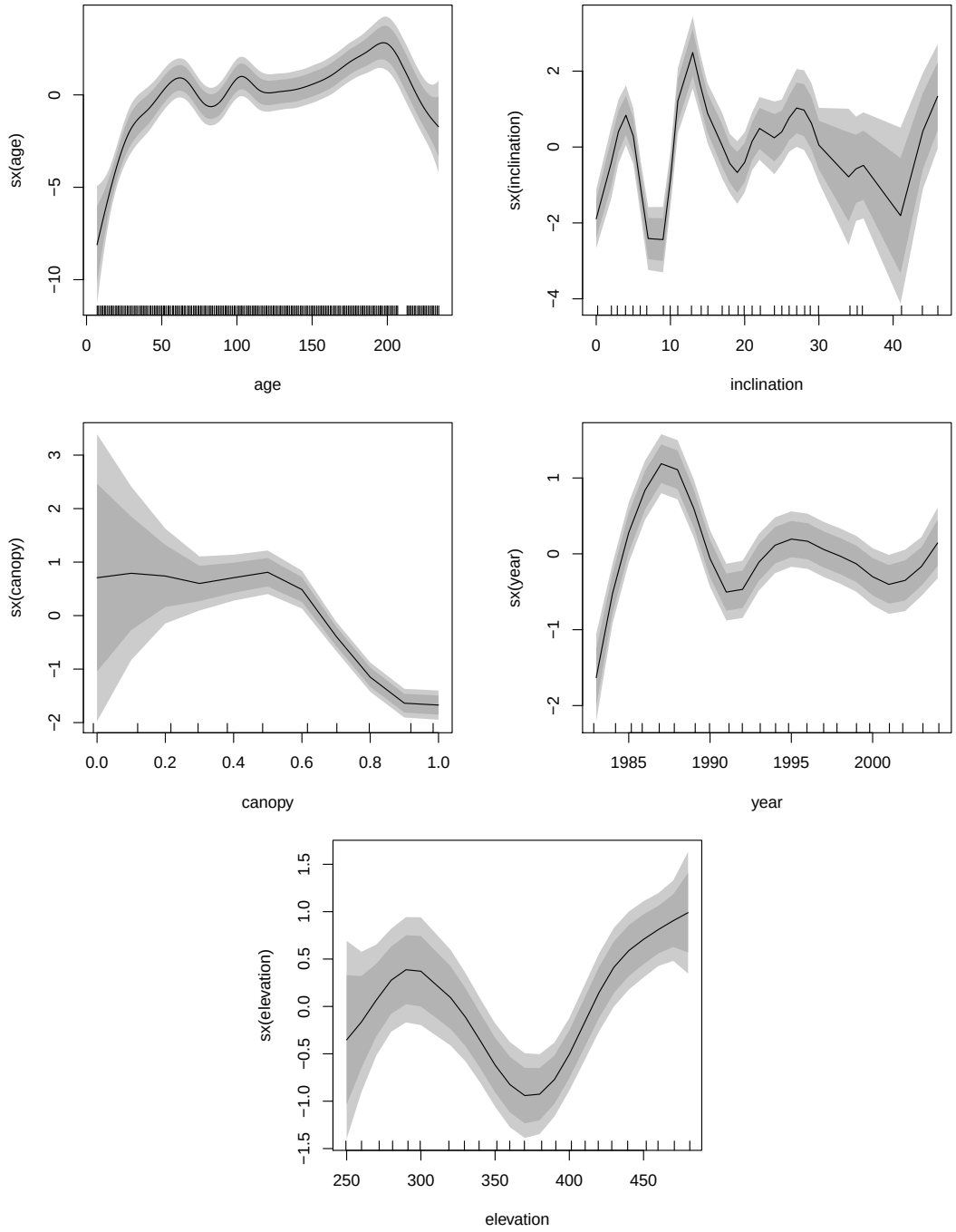

Figure 8: Forest damage: Estimates of nonparametric effects including 80% and 95% pointwise confidence intervals of the model without the spatial effect.

covariate `age` on the `defoliation` seems to decline for both, younger and older trees, which

intuitively should be a monotone increasing effect, this also holds for the effect of `elevation`. Moreover, the extremely wiggly estimate of `inclination` is hardly interpretable. Therefore, **?** extend the model by a spatial effect, which is modeled by a two dimensional geospline term of the tree locations. The tree x- and y-coordinates are calculated by the centroid positions of tree polygons given by the boundary map file, which may be loaded with

```
R> data("BeechBnd", package = "R2BayesX")
```

We now fit the model:

```
R> f <- update(f, ~ . + sx(id, bs = "gs", map = BeechBnd))
R> fm2 <- bayesx(f, family = "cumlogit",
+    method = "REML", data = ForestHealth)
```

Taking a look at model information criteria with

```
R> BIC(fm1, fm2)
```

```
         df     BIC
fm1 59.9714 2016.04
fm2 94.8222 1930.06
```

```
R> GCV(fm1, fm2)
```

```
         df      GCV
fm1 59.9714 0.816340
fm2 94.8222 0.610199
```

clearly indicates a better fit by modeling the spatial effect of tree locations. The summary statistics for both models gives:

```
R> summary(fm1)
```

```
Call:
bayesx(formula = f, data = ForestHealth, family = "cumlogit",
    method = "REML")

Fixed effects estimation results:

Parametric Coefficients:
                        Estimate Std. Error t value Pr(>|t|)
theta_1                  -4.3485     1.5039 -2.8914   0.0039 **
theta_2                   0.7500     1.5156  0.4948   0.6208
standmixed               -0.6175     0.1044 -5.9178   <2e-16 ***
fertilizedno              0.5362     0.1901  2.8208   0.0048 **
humus[0cm, 1cm]          -0.1407     0.1648 -0.8536   0.3934
humus(1cm, 2cm]           0.4421     0.1682  2.6289   0.0086 **
```

```
humus(2cm, 3cm]            0.0975     0.1793  0.5439   0.5866
humus(3cm, 4cm]            0.0771     0.2307  0.3341   0.7383
moisturemoderately dry    -0.7569     0.2088 -3.6246   0.0003 ***
moisturemoderately moist   0.3067     0.1418  2.1625   0.0307 *
alkalivery low             1.1612     0.2482  4.6793   <2e-16 ***
alkalilow                 -0.3889     0.1881 -2.0680   0.0388 *
alkalihigh                -0.9853     0.2242 -4.3957   <2e-16 ***
ph                        -0.8074     0.3021 -2.6728   0.0076 **
soil                      -0.0470     0.0104 -4.5008   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Smooth terms:
                Variance Smooth Par.      df Stopped
sx(age)           4.9911      0.2004 12.3322       0
sx(canopy)        0.0527     18.9743  4.7092       0
sx(elevation)     0.0668     14.9682  5.0563       0
sx(inclination)  25.8453      0.0387 14.4449       0
sx(year)          0.2971      3.3664  8.4287       0

N = 1793  df = 59.9714  AIC = 1686.69  BIC = 2016.04
logLik = -783.375  GCV = 0.81634  method = REML  family = cumlogit
```

R> summary(fm2)

```
Call:
bayesx(formula = defoliation ~ stand + fertilized + humus + moisture +
    alkali + ph + soil + sx(age) + sx(inclination) + sx(canopy) +
    sx(year) + sx(elevation) + sx(id, bs = "gs", map = BeechBnd),
    data = ForestHealth, family = "cumlogit", method = "REML")

Fixed effects estimation results:

Parametric Coefficients:
                        Estimate Std. Error t value Pr(>|t|)
theta_1                  -1.8244     2.0034 -0.9106   0.3626
theta_2                   4.5302     2.0421  2.2184   0.0267 *
standmixed               -0.1778     0.2269 -0.7835   0.4335
fertilizedno              0.5816     0.4977  1.1685   0.2428
humus[0cm, 1cm]          -0.3371     0.2004 -1.6817   0.0928 .
humus(1cm, 2cm]           0.2453     0.1951  1.2576   0.2087
humus(2cm, 3cm]           0.1656     0.2066  0.8014   0.4230
humus(3cm, 4cm]           0.2205     0.2578  0.8552   0.3926
moisturemoderately dry   -0.7054     0.5450 -1.2943   0.1957
moisturemoderately moist -0.0765     0.3899 -0.1961   0.8446
alkalivery low            0.9401     0.6297  1.4929   0.1357
alkalilow                -0.3564     0.4866 -0.7324   0.4640
```

```
alkalihigh                -0.3869     0.5608 -0.6899   0.4904
ph                        -0.3033     0.3611 -0.8399   0.4011
soil                      -0.0072     0.0281 -0.2553   0.7985
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Smooth terms:
                 Variance Smooth Par.        df Stopped
sx(age)            3.8455      0.2600   10.9703       0
sx(canopy)         0.0179     55.8909    3.2481       0
sx(elevation)      0.0002   5203.4900    1.0280       1
sx(id)            56.3986      0.0177   53.6092       0
sx(inclination)    0.0103     97.4621    1.8657       0
sx(year)           0.5220      1.9158    9.1008       0

N = 1793  df = 94.8222  AIC = 1409.33  BIC = 1930.06
logLik = -609.84  GCV = 0.610199  method = REML  family = cumlogit
```

Most of the parametric modeled terms in the second model now have an insignificant effect on tree defoliation, with similar findings for covariates `inclination` and `elevation` (where the pointwise 95% credible intervals cover the zero line). However, the estimate of the `age` effect seems to be improved in terms of monotonicity, see Figure~**??**.

A kernel density plot of the estimated spatial effect is then obtained by

```
R> plot(fm2, term = "sx(id)", map = FALSE)
```

The effect may also be visualized either using a 3d perspective plot, an image/contour plot or a map effect plot using the boundary file `BeechBnd` with

```
R> plot(fm2, term = "sx(id)", map = BeechBnd)
```

Both the kernel density and map effect plot are shown in Figure~**??**. In this example the coloring of the plot is strongly influenced by a few very high and low values. Therefore, it might be useful to restrict the color range e.g. according to the 10% and 90% quantiles of the kernel density estimate of the effect, by typing

```
R> plot(fm2, term = "sx(id)", map = BeechBnd, range = c(-3, 3))
```

the resulting map is shown in Figure~**??**. Trimming the color range of the plot now leads to a better representation of the effect.

Summarizing these results identify a strong influence of the spatial effect of the overall model fit, indicating that a clear splitting of locational specific covariates and the spatial effect is hardly possible in this example.

6.3. Childhood malnutrition in Zambia: Analysis with STEP

To illustrate the implemented methodology for simultaneous selection of variables and smoothing parameters, we proceed with the dataset on malnutrition in Zambia of Section~**??**. In this

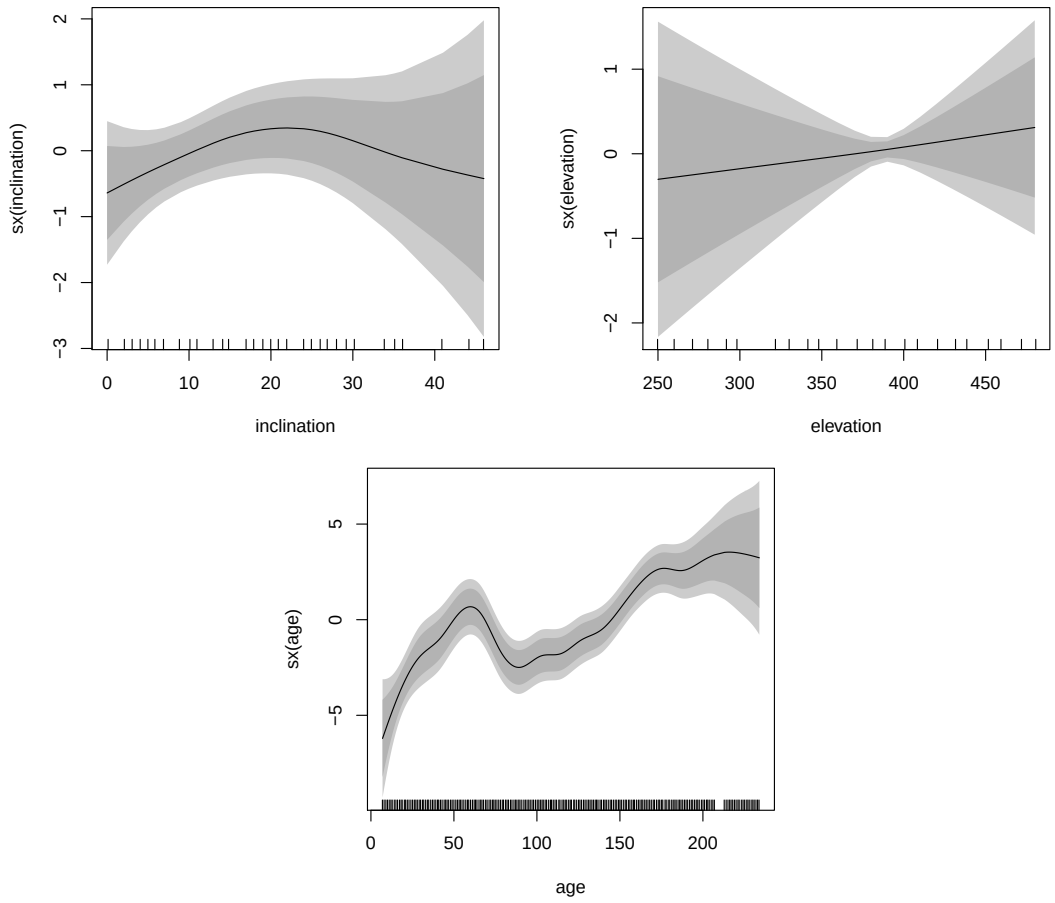

Figure 9: Forest damage: Estimated effects of covariates `inclination`, `elevation` and `age`, including 80% and 95% point-wise confidence intervals, of the model including the spatial effect.

example, the structured additive predictor˜(**??**) contains two continuous covariates `mbmi` and `agechild`, that are assumed to have a possibly nonlinear effect on the response `stunting` and are modeled with P-splines. However, a linear effect could be more appropriate and, hence, the linear effect is also considered using the selection algorithm in **BayesX**. Additionally, for each variable and function, the implemented procedures decide if a term is included or removed from the model. To estimate the model applying the option `method = "STEP"`, we use the same model formula of Section˜**??** and call

```
R> f <- stunting ~ memployment + urban + gender +
+    sx(meducation, bs = "factor") + sx(mbmi) + sx(agechild) +
+    sx(district, bs = "mrf", map = ZambiaBnd) + r(district)
R> zms <- bayesx(f, family = "gaussian", method = "STEP",
+    algorithm = "cdescent1", startmodel = "empty",
+    seed = 123, data = ZambiaNutrition)
```

where argument `algorithm` chooses the selection algorithm and `startmodel` the start model for variable selection, see also Table ˜**??** for all possible options. Usually the selected final

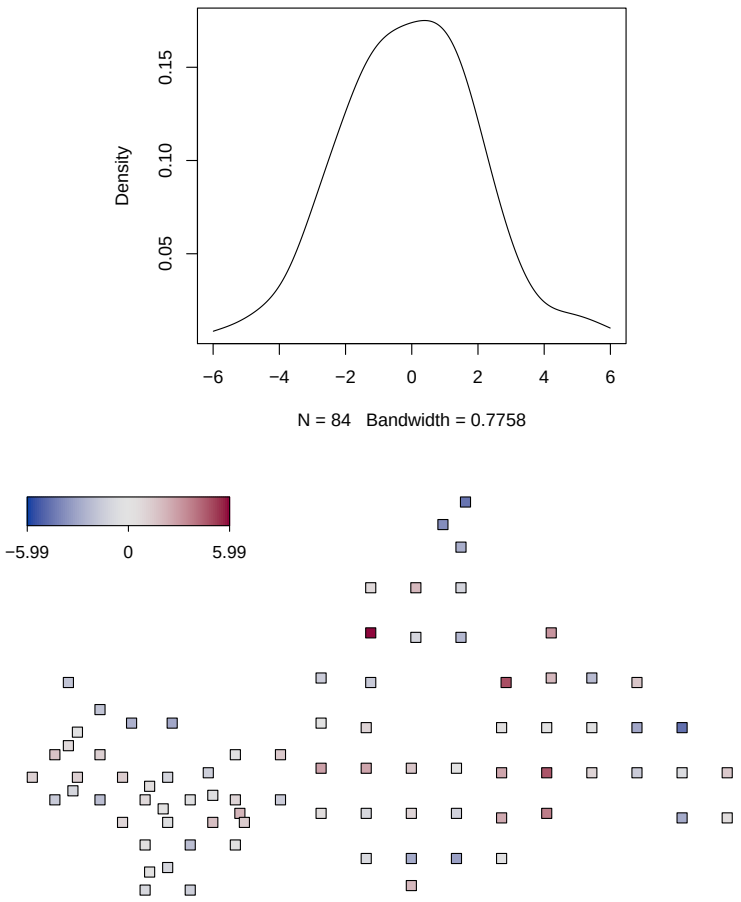

Figure 10: Forest damage: Kernel density estimate of the spatial effect, upper panel, together with a map effect plot, lower panel respectively.

model is unaffected by the selection algorithm and startmodel. However, it is generally of interest to assess the dependence of results on the selction algorithm and the startmodel. The summary statistics of the final selected model are then provided with

```
R> summary(zms)

Call:
bayesx(formula = f, data = ZambiaNutrition, family = "gaussian",
    method = "STEP", algorithm = "cdescent1", startmodel = "empty",
    seed = 123)

Fixed effects estimation results:

Parametric Coefficients:
               Mean      Sd    2.5%     50% 97.5%
(Intercept) -0.5946  0.0000  0.0000  0.0000     0
urbanno     -0.0945  0.0000  0.0000  0.0000     0
```

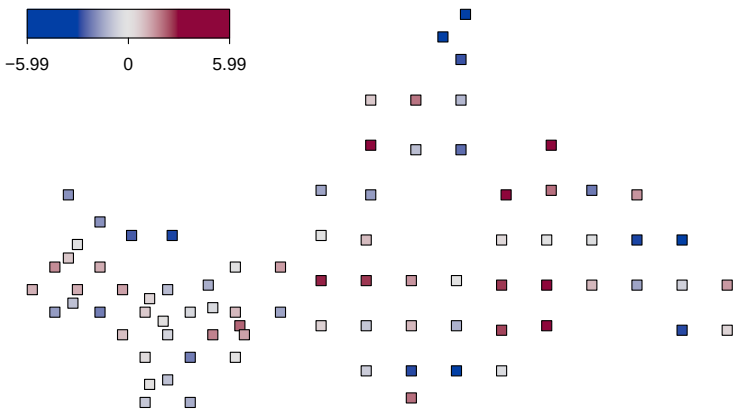

Figure 11: Forest damage: Estimate of the spatial effect with color scaling based on the 10% and 90% quantiles of the kernel density estimate of the effect.

```
genderfemale  0.0589  0.0000  0.0000  0.0000     0
meducation_2  0.1087  0.0000  0.0000  0.0000     0
meducation_3  0.4064  0.0000  0.0000  0.0000     0
mbmi          0.0209  0.0000  0.0000  0.0000     0

Smooth terms:
             lambda     df
f(agechild) 15.4071 10.959
f(district)  7.5775 24.366
r(district) 35.6851 17.872

Scale estimate: 0.7897

N = 4847  AIC_imp = -1024.35  method = STEP  family = gaussian
```

Note that variable `memployment` was removed from the model and variable `mbmi` is modeled by a linear effect. Moreover, the columns `sd`, `2.5%`, `50%` and `97.5%` contain no values, likewise for the estimated random and smooth effects. The posterior quantiles may be computed if argument `CI` in function `bayesx.control()` is specified. E.g. conditional confidence bands can be calculated conditional on the selected model, i.e. they are computed for selected variables and functions only. The computation of conditional confidence bands is based on an MCMC-algorithm subsequent to the selection procedure. For the selection of a model with an subsequent computation of conditional confidence bands the user may type

```
R> zmsccb <- bayesx(f, family = "gaussian", method = "STEP",
+    algorithm = "cdescent1", startmodel = "empty",
+    CI = "MCMCselect", iterations = 10000, step = 10,
+    seed = 123, data = ZambiaNutrition)
```

which results in the following summary

```
R> summary(zmsccb)

Call:
bayesx(formula = f, data = ZambiaNutrition, family = "gaussian",
    method = "STEP", algorithm = "cdescent1", startmodel = "empty",
    CI = "MCMCselect", iterations = 10000, step = 10, seed = 123)

Fixed effects estimation results:

Parametric Coefficients:
                Mean      Sd    2.5%     50%   97.5%
(Intercept)  -0.5906  0.0937 -0.7790 -0.5900 -0.4121
urbanno      -0.0944  0.0242 -0.1398 -0.0947 -0.0457
genderfemale  0.0591  0.0130  0.0337  0.0593  0.0849
meducation_2  0.1087  0.0291  0.0523  0.1083  0.1674
meducation_3  0.4080  0.0690  0.2808  0.4053  0.5448
mbmi          0.0207  0.0041  0.0125  0.0207  0.0289

Smooth terms:
             lambda     df
f(agechild) 15.4071 10.959
f(district)  7.5775 24.366
r(district) 35.6851 17.872

Scale estimate: 0.7897

N = 4847  DIC = 4965.945  pd = 59.33154  AIC_imp = -1024.35
method = STEP  family = gaussian  iterations = 10000  step = 10
```

It is also possible to obtain unconditional confidence bands by setting `CI = "MCMCbootstrap"`, which additionally considers the uncertainty due to model selection.

Another important feature of the stepwise procedure is the definition of a `startmodel`, the options are listed in Table~??. Besides the default of an `"empty" startmodel`, it may be reasonable to start with an userdefined model. Therefore, the starting values for the degrees of freedom of the P-spline, spatial and random effect terms need to be specified, by typing

```
R> f <- stunting ~ memployment + urban + gender +
+    sx(meducation, bs = "factor") + sx(mbmi, dfstart = 2) +
+    sx(district, bs = "mrf", map = ZambiaBnd, dfstart = 5) +
+    r(district, xt = list(dfstart = 5)) +
+    sx(agechild, dfstart = 2)
```

The model is then fitted by

```
R> zmsud <- bayesx(f, family = "gaussian", method = "STEP",
+    algorithm = "cdescent1", startmodel = "userdefined",
+    CI = "MCMCselect", iterations = 10000, step = 10,
+    seed = 123, data = ZambiaNutrition)
```

which actually produces the model output of the first model again.

A. Appendix

A.1. Installing the BayesX Windows binary

The installation routine `BayesX_windows.exe` (url: http://www.stat.uni-muenchen.de/~bayesx/install/BayesX_windows.exe) may be downloaded from the **BayesX** homepage and executed. The routine will request all necessary information during the installation process. It is recommended to install **BayesX** in a directory without spaces in the path name. This will install a pre-compiled computing kernel including a Java graphical interface, which automatically installs the necessary command-line version of **BayesX** for use within the R interface. The binary is named `bayesx.exe` and is stored in the `commandline` directory of the **BayesX** installation folder.

A.2. Installing the BayesX sources

The zip archive named `bayesxsource.zip` (url: http://www.stat.uni-muenchen.de/~bayesx/install/bayesxsource.zip), containing the C++ source code of **BayesX**, needs to be downloaded and unpacked. If the make facility is available, one can simply type make **BayesX** in the shell and **BayesX** will be compiled. Depending on the operating system, some minor modifications of the make file (for example relating to the version of the installed GNU compiler or the location of the readline library) may be necessary. For MAC OS, versions of an adjusted makefile and the main function that have been used for a sucessful compilation may be found at the FAQ site (http://www.stat.uni-muenchen.de/~bayesx/bayesxfaq.html).

A.3. Editing R's profile startup site

To permanently link to the **BayesX** command line binary, R's startup profile site may be edited. See the manual for R's startup mechanism with

```
R> help("Startup")
```

to find out how to edit the startup site on specific systems that is processed at the beginning of a new R session. Then, by adding the line

```
options(bayesx.bin = "path/to/BayesX")
```

to the startup site will tell R where to find the binary permanently. The string `"path/to/BayesX"` is the full path to the **BayesX** binary with the name of the binary at last. On Windows systems the binary name is `"bayesx.exe"`, on all other systems usually `"BayesX"`. Hence, for Windows platforms `bayesx.bin` may be specified e.g. with

```
options(bayesx.bin = "C:/BayesX/commandline/bayesx.exe")
```

depending on the installation location of the binary.

Furthermore, another possibility is to add the installation directory of the binary to the environment PATH variable of the operating system. Both options will automatically link the binary with R for the upcoming sessions.

A.4. Most important arguments used within function `plot.bayesx()`

Argument	Description
`term`	the term that should be plotted, either an integer or a character, e.g. `term = "sx(x)"`.
`which`	choose the type of plot that should be drawn, possible options are: `"effect"`, `"coef-samples"`, `"var-samples"`, `"intcpt-samples"`, `"hist-resid"`, `"qq-resid"`, `"scatter-resid"`, `"scale-resid"`, `"max-acf"`. Argument `which` may also be specified as integer, e.g. `which = 1`. The first three arguments are all model term specific. For the residual model diagnostic plot options `which` may be set with `which = 5:8`.
`residuals`	if set to `TRUE`, partial residuals may also be plotted if available.
`rug`	if set to `TRUE`, a `rug()` is added to the plot.
`jitter`	if set to `TRUE`, a `jitter()`ed `rug()` is added to the plot.
`col.surface`	the color of the surface, may also be a function, e.g. `col.surface = heat.colors`.
`grid`	the grid size of the surface(s).
`image`	if set to `TRUE`, an `image.plot()` is drawn.
`contour`	if set to `TRUE`, a `contour()` plot is drawn.
`map`	the map to be plotted, the map object must be a list of matrices with first column indicating the x-coordinate and second column the y-coordinate each, see also function `polygon()`.
`legend`	if set to `TRUE`, a legend will be shown.
`range`	specify the range of values the plot should be generated for, e.g. only values between -2 and 2 are of interest then `range = c(-2, 2)`.
`color`	the colors for the legend, may also be a function, e.g. `colors = heat.colors`.
`pos`	the position of the legend, either a numeric vector, e.g. `pos = c(0.1, 0.2)` will add the legend at the 10% point in the x-direction and at the 20% point in the y-direction of the plotting window, may also be negative, or one of the following: `"bottomleft"`, `"topleft"`, `"topright"` or `"bottomright"`. Using function `plotmap()` option `"right"` is also valid.
`lrange`	specifies the range of the legend.
`symmetric`	if set to `TRUE`, a symmetric legend will be drawn corresponding to the $+-max(\|x\|)$ of values x that are used for plotting.

Table 7: Most important plotting parameters using the generic function `plot.bayesx()`. The first block describes arguments of `plot.bayesx()`, subsequent blocks arguments that are passed to function `plot2d()`, `plot3d()`, `plotmap()` and `colorlegend()` respectively.

Affiliation:

Nikolaus Umlauf, Stefan Lang, Achim Zeileis
Department of Statistics
Universität Innsbruck
Universitätsstr.˜15
A-6020 Innsbruck, Austria
E-mail: Nikolaus.Umlauf@uibk.ac.at,
Stefan.Lang@uibk.ac.at,
Achim.Zeileis@R-project.org
URL: http://www.uibk.ac.at/statistics/personal/umlauf/,
http://www.uibk.ac.at/statistics/personal/lang/,
http://eeecon.uibk.ac.at/˜zeileis/

Thomas Kneib
Department of Mathematics
Universität Oldenburg
D-26111 Oldenburg, Germany
E-mail: Thomas.Kneib@uni-oldenburg.de
URL: http://www.staff.uni-oldenburg.de/thomas.kneib/