Structured Additive Regression Models: An R Interface to BayesX

Nikolaus Umlauf
Universität Innsbruck

Daniel Adler
Universität Göttingen

Thomas Kneib
Universität Göttingen

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, allowing for specification of complex and realistic models. **BayesX** is standalone software package providing software for fitting general class of STAR models. Based on a comprehensive open-source regression toolbox written in C++, **BayesX** uses Bayesian inference for estimating STAR models based on Markov chain Monte Carlo (MCMC) simulation techniques, a mixed model representation of STAR models, or stepwise regression techniques combining penalized least squares estimation with model selection. **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 a new fully interactive R interface to **BayesX**: the R package **R2BayesX**. 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. This makes **BayesX** much more accessible to users familiar with R and adds extensive graphics capabilities for visualizing fitted STAR models. Furthermore, **R2BayesX** complements the already impressive capabilities for semiparametric regression in R by a comprehensive toolbox comprising in particular more complex response types and alternative inferential procedures such as simulation-based Bayesian inference.

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

1. Introduction

The free software **BayesX** (see Brezger, Kneib, and Lang 2005) is a standalone program (current version 2.0.1, Belitz, Brezger, Kneib, and Lang 2009) comprising powerful tools for Bayesian, mixed-model-based and stepwise inference in complex semiparametric regression models with structured additive predictor. Besides exponential family regression, **BayesX** also supports models for multi-categorical responses, hazard regression for continuous survival

times, and continuous time multi-state models. The software is written in C++, utilizing numerically efficient (sparse) matrix architectures.

To facilitate usage of results from **BayesX** in subsequent analyses, specifically in explorations and visualizations of the fitted models, Kneib, Heinzl, Brezger, and Sabanes Bove (2011) provide a package for R (R Development Core Team 2011), also called **BayesX**, that can read and process output files from **BayesX**. However, in this approach the users still have to read their data into **BayesX**, fit the models of interest and obtain the corresponding output files. To alleviate this task, we introduce a new R package **R2BayesX** that provides a fully interactive R interface to **BayesX** that has the usual R modeling "look & feel" and obviates the tedious exercise of manually exporting data and fitting models in **BayesX**. Within the new package, users are now provided with the typical R modeling workflow namely:

- Specification and estimation of STAR models using `bayesx(formula, data, ...)` (which internally calls **BayesX** and reads its results).
- Methods and extractor functions for fitted "`bayesx`" model objects, e.g., producing high-level graphics of estimated effects, model diagnostic plots, summary statistics etc.

In addition, users can leverage the underlying infrastructure, i.e.:

- Run already existing **BayesX** input program files from R via `run.bayesx()`.
- Automatically import **BayesX** output files into R via `read.bayesx.output()`.

The formula interface of the `bayesx()` function uses several special model term constructor functions for the structured predictors: `sx()` and `r()` for smooth and random effects, respectively, as well as the functions `s()` and `te()` from the **mgcv** package (Wood 2012, 2006), facilitating a consistent way to translate R syntax into **BayesX**-interpretable commands.

The functionality is made available in package **R2BayesX**, available from the Comprehensive R Archive Network (CRAN) at `http://CRAN.R-project.org/package=R2BayesX`. It depends on the companion package **BayesXsrc** (also available from CRAN, see Adler, Kneib, Lang, Umlauf, and Zeileis 2012) that ships the **BayesX** C++ sources along with flexible `Makefile`s so that upon installation of the R package a suitable **BayesX** binary is produced on all platforms.

The remainder of this paper is as follows. Section 2 gives a first motivating example of an R session applying **R2BayesX** to a dataset on childhood malnutrition in Zambia. Subsequently, Section 3 briefly discusses the methodological background of structured additive regression models before Section 4 describes the implementation details and the user interface provided by **R2BayesX**. In Section 6, the versatility of **BayesX** and the flexibility of the **R2BayesX** interface are further illustrated with an extended analyses of the childhood malnutrition data and a dataset on forest health in Germany.

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 Kandala, Lang, Klasen, and Fahrmeir 2001 and also Section 6.1) 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 assessment of the determinats of stunting (`stunting`), represented by anthropometric indicators of newborn children. Covariates include the age of the children (`agechild`), the body mass index (BMI) of the mother (`mbmi`) and the district the children live 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 map of the districts in Zambia (see Section 5.2 for details of the map format) is loaded with

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

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)
```

The 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)  0.0275  0.0422 -0.0587  0.0284 0.1088

Smooth terms variances:
               Mean     Sd   2.5%    50%  97.5%    Min    Max
sx(agechild) 0.0065 0.0064 0.0012 0.0045 0.0232 0.0007 0.0721
sx(district) 0.0442 0.0175 0.0203 0.0411 0.0886 0.0133 0.1445
sx(mbmi)     0.0020 0.0025 0.0003 0.0012 0.0086 0.0001 0.0282

Scale estimate:
```

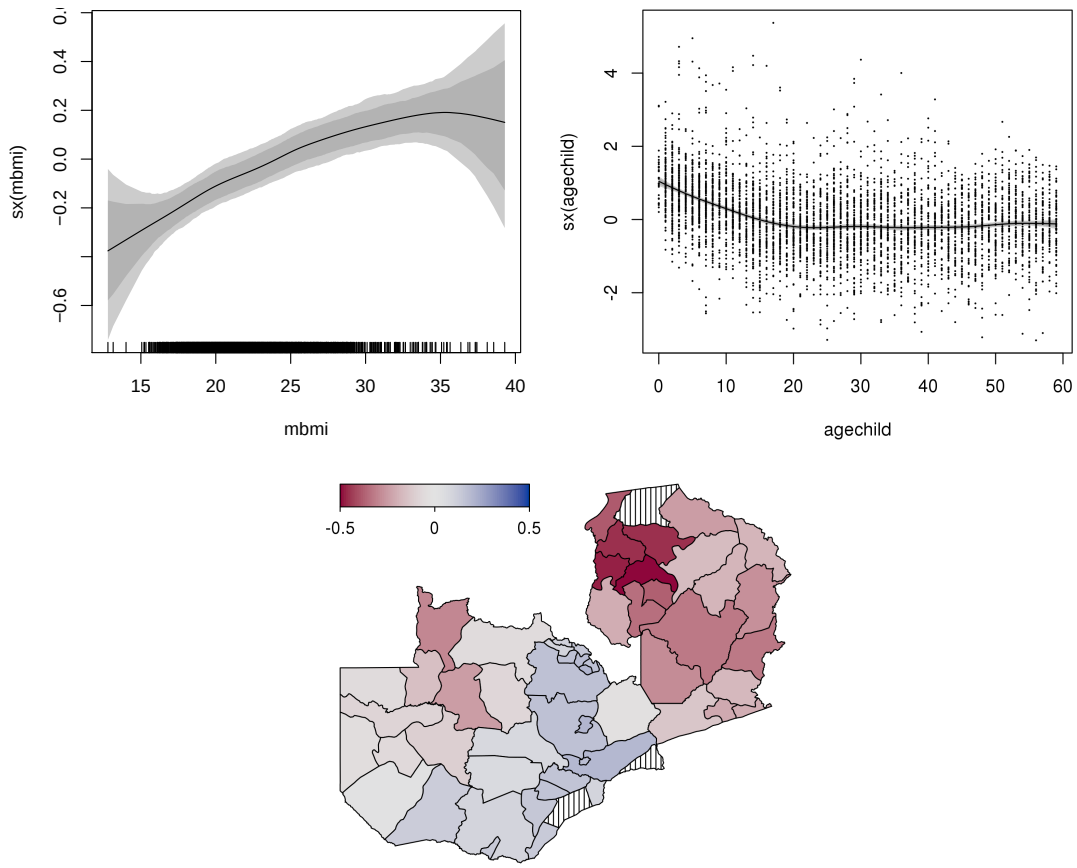

Figure 1: Visualization examples: Estimated effect for covariate `mbmi` (black line) together with 95% and 80% credible intervals (upper left panel). The upper right panel shows the estimated effect of `agechild` including partial residuals. The lower panel illustrates visualization of the estimated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

```
         Mean     Sd   2.5%    50%  97.5%
Sigma2 0.8178 0.0164 0.7856 0.8174 0.8515

N = 4847  burnin = 2000  DIC = 4884.843  pd = 37.6607
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)")
```

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 1, depicting the centered additive effects (i.e., each of the additive effects is zero on average). The map effect plot indicates pronounced stunting (i.e., low values of the response) in the northern parts of Zambia. Furthermore, stunting effects are lower (i.e., the response is higher) for children younger than 20 months of age, while the `agechild` effect is almost constant for ages above 20 months. Finally, the response increases almost linearly with increasing mother's BMI. In comparison, the effects of `mbmi` and the spatial effect seem to have a quite similar influence in absolute magnitude (indicated by the ranges of the respective axes), while the strongest driver of stunting appears to be covariate `agechild`. Extended analyses of the data are discussed in Sections 6.1 and 6.3.

3. STAR models

The STAR model class supported by **R2BayesX** is based on the framework of Bayesian generalized linear models (GLMs, see e.g., McCullagh and Nelder 1989 and Fahrmeir and Tutz 2001). 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

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

where h is a known link function and $\boldsymbol{\gamma}$ are unknown regression coefficients. In STAR models (Fahrmeir, Kneib, and Lang 2004; Brezger and Lang 2006), the linear predictor is replaced by a more general and flexible, structured additive predictor

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

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 encompassing various types of effects, e.g.:

- 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, Hastie and Tibshirani 1990), generalized additive mixed models (GAMM, Lin and Zhang 1999), geoadditive models (Kamman and Wand 2003), varying coefficient models (Hastie and Tibshirani 1993), and geographically weighted regression (Fotheringham, Brunsdon, and Charlton 2002).

The unified representation of a STAR predictor arises from the fact that all functions f_j in (1) may be specified by a basis function approach, where the vector of function evaluations $\mathbf{f}_j = (f_j(\mathbf{z}_1), \ldots, f_j(\mathbf{z}_n))^\top$ 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 (1) 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

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

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 MCMC 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 (Brezger and Lang 2006).

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 (Fahrmeir *et al.* 2004).

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., Akaike or Bayes information criterion (AIC or BIC), generalized cross-validation (GCV), or 5- or 10-fold cross-validation (Belitz and Lang 2008).

A thorough introduction into the regression models supported by the program is also provided in the **BayesX** methodology manual (Belitz *et al.* 2009).

4. Implementation of the R interface to BayesX

The design of the interface attempts to address the following major issues: First, the interface functions should follow R's conventions for regression model fitting functions so that they are easy to employ for R users. Second, the functions and methods for representing fitted model objects should reflect **BayesX** models to enhance their usability.

4.1. Interface approach

The first challenge in establishing a communication between R and **BayesX** is the question which interface to use. As **BayesX** is written in C++, one might expect that `.C()` or `.Call()` could be an option. However, as **BayesX** was designed as a standalone software it does not offer an application programming interface (API) and restructuring the mature and complex **BayesX** C++ code to obtain an API at this point is not straightforward. Hence, **R2BayesX** adopts the simpler approach of writing the data out from R, calling the **BayesX** binary with a suitable program file, and then collecting all output files and representing them in suitable R objects. This is straightforward and the additional computation effort (as compared to a direct call) is rather modest compared to time needed for carrying out the estimation of STAR models within **BayesX**.

Thus, for the interface adopted by **R2BayesX** a binary installation of **BayesX** is required. To make this easily available to R users in a standardized way, the **BayesX** C++ sources are encapsulated in R package **BayesXsrc** along with `Makefile`s for GNU/BSD and MinGW platforms that conform with R build shells. Consequently, upon installation of the **BayesXsrc**

package, the binary `BayesX` (or `BayesX.exe` on Windows) is created in the installed package. Package **BayesXsrc** is also available from CRAN at http://CRAN.R-project.org/package=BayesXsrc and some of its implementation details are discussed in Appendix A.

4.2. Model specification

The second challenge for the interface package **R2BayesX** is to employ an objects and methods interface that reflects the workflow typically adapted by R packages for fitting GAMs and related models. CRAN packages that implement such models include the following prominent ones: One of the first implementations of GAMs in R is the **gam** package (Hastie and Tibshirani 1990; Hastie 2011). 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 GAMs. The probably best-known and also recommended package is **mgcv** (Wood 2006, 2011, 2012), which provides fast and stable algorithms for estimating GAMs based on GCV, REML and others. Vector generalized additive models (VGAMs, Yee and Wild 1996) for categorical responses are covered by package **VGAM** (Yee 2010). Another comprehensive toolbox for GAMs, accounting for responses that do not necessarily follow the exponential family and may exhibit heterogeneity, is the **gamlss** suite of packages (Rigby and Stasinopoulos 2005; Stasinopoulos and Rigby 2007). A package based on mixed model technologies is **SemiPar** (Ruppert, Wand, and Carrol 2003; Wand 2010) and, building on top of this, the **AdaptFit** package for adaptive splines (Krivobokova 2009). The package **spikeSlabGAM** applies Bayesian variable selection, model choice and regularization for GAMMs (Scheipl 2011).

Most of these packages follow the common R paradigm of specifying regression models conveniently using its formula language (Chambers and Hastie 1992). However, the above-mentioned packages employ somewhat different approaches for representing smooth/special terms for GAMs in formulas and the subsequent building of model frames. A popular approach, though, is to use a model term constructor function "`s`", as used in packages **gam**, **mgcv**, and **VGAM**. As the implementation details are somewhat different across these packages, loading packages simultaneously may lead to conflicts. Therefore, **R2BayesX** follows the approach of the recommended package **mgcv** where `s()` does not evaluate design or penalty matrices, but simply 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 term, a method for the S3 generic function `smooth.construct()` needs to be supplied, that returns a design matrix etc. Since implementation of additional model terms is also a concern for **R2BayesX** and function `s()` is a very lean solution, we adopt its functionality and provide 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**. In addition to the support of **mgcv**'s `s()` function, a second smooth term constructor `sx()` is provided (described in detail in Section 5.2). This is essentially a convenience interface that calls `s()` but uses somewhat

different defaults and additional options specific to the terms supported by **BayesX**. Moreover, a special constructor function `r()` for random effect terms is provided.

4.3. Under the hood

The main user interface of **R2BayesX** is the function `bayesx()` (presented in detail in Section 5.1). Internally, this function employs the helper functions `parse.bayesx.input()`, `write.bayesx.input()`, `run.bayesx()`, and `read.bayesx.output()` in the following work sequence: First, a program file is generated by applying function `parse.bayesx.input()` to 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 temporary directories and data files to be used with **BayesX**. Afterwards, function `run.bayesx()` (provided in **BayesXsrc**) 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()`. Using these helper functions it is also possible to run and read already existing **BayesX** program and output files, see Section 5.4 and the **R2BayesX** manuals for a detailed description. The object returned by function `read.bayesx.output()` is a list of class "`bayesx`", for which a set of base R functions and methods described in Table 3, amongst others, is available. The returned fitted model term objects also have suitable classes along with corresponding plotting methods. Particular effort has been given on the development of easy-to-use map effect plots using color legends (by default employing HCL-based palettes, Zeileis, Hornik, and Murrell 2009, from the **colorspace** package, Ihaka, Murrell, Hornik, and Zeileis 2012). See also Section 5 for more details and Section 6 for some practical applications.

5. User interface

5.1. Calling 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 = NULL, contrasts = NULL,
  family = "gaussian", method = "MCMC", control = bayesx.control(...), ...)
```

where the first two lines basically represent the standard model frame specifications (see Chambers and Hastie 1992) and the third line collects the arguments specific to **BayesX**.

The data processing is carried out "as usual" as in `lm()` or `glm()` with the following additions: (1) The `data` can not only be provided as a "`data.frame`" but it is also possible to provide a character string with a path to a dataset stored on disc, which can be leveraged to avoid reading very large data files into R just to write them out again for **BayesX**. An example is given in Section 5.4. (2) 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**.

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	"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".

The **BayesX**-specific arguments comprise specification of the response distribution family, the estimation method and further control parameters collected in bayesx.control(). The default response distribution is family = "gaussian". Note that "family" objects (in the glm() sense) are currently not supported by **BayesX**. The inferential concepts that can be used as the estimation method comprise: "MCMC" for Markov chain Monte Carlo simulation, "REML" for mixed-model-based estimation using restricted maximum likelihood/marginal likelihood, and "STEP" for penalized likelihood including model selection. An overview of all available distributions for the different methods is given in Table 1.

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 2.

The returned fitted model object is a list of class "bayesx", which is supported by several standard methods and 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

method	Parameter	Description
"MCMC"	iterations	Integer number of iterations for the sampler, default: 12000.
	burnin	Integer 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.
"STEP"	algorithm	Character, specifies the selection algorithm. Possible values are "cdescent1" (adaptive algorithms see Section 6.3 in Belitz *et al.* 2009), "cdescent2" (adaptive algorithms 1 and 2 with backfitting, see remarks 1 and 2 of Section 3 in Belitz and 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 function of S-PLUS, see Chambers and Hastie 1992), 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 which serves as the goodness of fit criterion to compare different models), "GCV" (generalized cross-validation based on deviance residuals), "GCVrss" (GCV based on residual sum of squares), see e.g., Wood (2006), "AIC" (Akaike information criterion), "AIC_imp" (improved AIC with bias correction for regression models), see e.g., Burnham and Anderson (1998), "BIC" (Bayesian information criterion) "CV5" (5-fold cross validation) "CV10" (10-fold CV), see e.g., Hastie, Tibshirani, and Friedman (2009), 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" (model with degrees of freedom equal to one for model terms), "empty" (empty model containing only an intercept), "full" (most complex possible model) and "userdefined" (user-specified model), default: "linear".

Table 2: Most important controlling parameters for the different methods using function bayesx(). See ?bayesx.control for more details.

Function	Description
`print()`	Simple printed display of the initial call and some additional information of the fitted model.
`summary()`	Return an object of class "`summary.bayesx`" containing the relevant summary statistics (which has a `print()` method).
`coef()`	Extract coefficients of the linearly 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 `mbmi` are estimated with `s(mbmi, bs = "ps", xt = list(contourprob = 4))` (see also Section 5.2).
`fitted()`	Fitted values of either the mean and linear predictor, or a selected model term.
`residuals()`	Extract model or partial residuals for a selected term.
`samples()`	Extract samples of parameters from MCMC simulation.
`bayesx_logfile()`	Extract the internal **BayesX** log file.
`bayesx_prgfile()`	Extract the **BayesX** program file.
`bayesx_runtime()`	Extract the overall runtime of the **BayesX** binary.
`terms()`	Extract terms of model components.
`model.frame()`	Extract/generate the 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()`	Compute information criteria, availability is dependent on the `method` used.

Table 3: Functions and methods for objects of class "`bayesx`". More details are provided in the manual pages.

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 Appendix B for an overview of the most important arguments for the plotting functions. 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 easily be converted to objects of class "`mcmc`" using the **coda** package (Plummer, Best, Cowles, and Vines 2006) 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. 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 fitting process when setting `verbose = TRUE`, or it can be extracted from the fitted model object using function `bayesx_logfile()`. A list of all available functions and methods of package **R2BayesX** can be found in Table 3.

5.2. 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 4. The arguments of function `sx()` are

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

where `x` represents the covariate that is used for univariate terms and `z` is used additionally for bivariate model terms. Argument `bs` chooses the basis/type of the term, see Table 4 for possible options of `bs` (and note that some terms have equivalent short and long specifications, e.g., `bs = "ps"` or `bs = "psplinerw2"`). Argument `by` can be a numeric or a factor variable 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. Finally, the "`...`" argument is used to set term-specific control parameters or additional geographical information.

For example to modify the degree and the inner knots for the P-spline term `sx(mbmi)` from Section 2, `sx(mbmi, degree = 2, knots = 10)` could be used. 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 shown 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 simplicity, only the first two options are shown here.

For fitting geoadditive models utilizing spatial information – i.e., by computing suitable neighborhood penalty matrices for terms using Markov random field (MRF) priors, or by calculating the centroids of particular regions for geosplines and geokriging terms – an argument named `map` needs to be provided to `sx()`. For example, the map of Zambia in the geokriging term in Section 2 is included with `sx(district, bs = "gk", map = ZambiaBnd)`. The `map` argument can be an object of class "`SpatialPolygonsDataFrame`" (Pebesma and Bivand

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: Estimation is 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 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` (see above).
"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` (see above).
"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.
"re"	Gaussian i.i.d. Random effects of a unit or cluster identification covariate.

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

2005; Bivand, Pebesma, and Gómez-Rubio 2008) or an object of class "bnd". The latter is essentially a named list of the map's polygons which is the format required by **BayesX** for its computations. In case a "SpatialPolygonsDataFrame" is supplied it is transformed internally to such a polygon list which is employed for all further computations. Furthermore, "bnd" objects can be created directly using functions from the R package **BayesX** of Kneib *et al.* (2011): `read.bnd()` and `shp2bnd()` create "bnd" objects from text files or shapefiles (using package **shapefiles**, Stabler 2006), respectively. For MRF terms, it is possible to supply the whole map as outlined above but it suffices to supply the corresponding neighborhood information. Internally, **BayesX** uses a list specification of neighbors which is captured in objects of class "gra" that can be created by `read.gra()` and `bnd2gra()`. Improvements in the handling of spatial information – especially by leveraging more functionality from the **sp** family of packages – are planned for future versions of **R2BayesX**.

Some care is warranted for 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 (with an intercept in the model) 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.

5.3. Other additive term interfaces

As mentioned above, users may optionally call the constructor function `s()` directly, since `sx()` is only a wrapper function which calls `s()` in the end. The usage of `s()` in **R2BayesX** is in principle similar to package **mgcv**. However, the applicable arguments are limited to arguments corresponding to those also available in **BayesX**, i.e.,

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

Note that the defaults of `s()` (as provided by **mgcv**) are different from the values above and would not correspond to a spline basis available in **BayesX**. Hence, the new `sx()` function is introduced in **R2BayesX** as a more convenient interface with defaults as in **BayesX**.

Within `s()`, the list of covariates used for the model term is set with argument "`...`". For instance, in the example of Section 2, 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 by **BayesX**.

5.4. Additional options

For practical purposes fitting models with function `bayesx()` is typically sufficient. However, the interfacing functions that are called internally within `bayesx()` can also be used independently. This could be useful for two reasons: First, users may want to use already existing **BayesX** program files, and second, there might be a need for automated importing of previously generated **BayesX** output files into R for further analysis.

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

```
run.bayesx(prg = NULL, verbose = TRUE, ...)
```

where `prg` is a character string with the path to a program file to be executed. If argument `prg` is not provided **BayesX** will start in batch mode. During processing of **BayesX** several informations will be printed to the R console if `verbose = TRUE`. Further arguments may be passed to function `system()`, which calls the **BayesX** binary, using the "`...`" argument. 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 3 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 very 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 more burdensome when provided to model fitting functions that do not account for special matrix structures. As mentioned in Section 5.1, 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 less than half a megabyte of memory:

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

0.4 Mb
```

6. STAR models in practice

The focus of this section is on demonstrating the various features of the **R2BayesX** package. Therefore, the examples provided reconsider analyses from Brezger *et al.* (2005) and Fahrmeir, Kneib, and Lang (2009). The presented datasets have been added to package **R2BayesX**, ensuring straightforward reproducibility 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 employed for an ordered response variable (see Fahrmeir and Tutz 2001, and Kneib and Fahrmeir 2006 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 Kandala *et al.* (2001) and has also been used as a demonstrating example in Brezger *et al.* (2005). Stunting is one of the leading drivers of a

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.

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 5. The response `stunting` is standardized in terms of a reference population, i.e., in this dataset stunting for child i is represented by

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

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

Following Kandala *et al.* (2001), we estimate a structured additive regression model with predictor

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

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, 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")
```

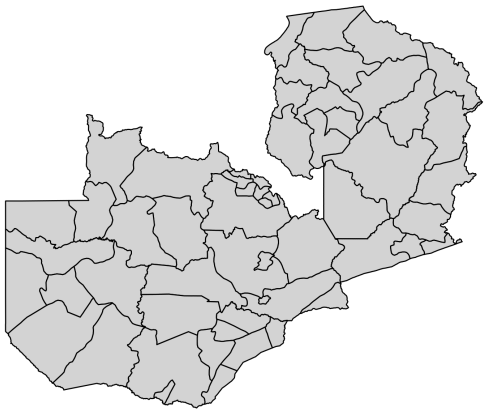

Figure 2: Example on childhood malnutrition: A simple map of the districts in Zambia.

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. With the information of the boundary file **BayesX** may compute an appropriate adjacency matrix, allowing for a smoothly varying effect of the neighboring regions. In addition, "`bnd`" objects can be used to calculate centroids of polygons to estimate smooth bivariate effects of the resulting coordinates (e.g., using the `"geokriging"` option in Section 2, also see Section 6.2 for another example). There is a generic plotting method implemented for objects of class "`bnd`", which essentially calls function `plotmap()`. E.g., a simple map, as shown in Figure 2, 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) +
+    sx(district, bs = "re")
```

As mentioned above, the structured spatial effect is now modeled as a Markov random field (option `"mrf"`), while in Section 2 we used the region centroids to model a smooth spatial effect applying (geo)kriging. The model is then fitted using MCMC by calling

```
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 crucial to inspect the 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** for exact reproducibility of the model fit.

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,
    seed = 123)

Fixed effects estimation results:

Parametric Coefficients:
                     Mean      Sd    2.5%     50%   97.5%
(Intercept)        0.1052  0.0497  0.0060  0.1059  0.1989
memploymentno     -0.0076  0.0139 -0.0356 -0.0071  0.0206
urbanno           -0.0897  0.0221 -0.1340 -0.0894 -0.0474
genderfemale       0.0583  0.0129  0.0338  0.0584  0.0833
meducationno      -0.1736  0.0277 -0.2256 -0.1731 -0.1203
meducationprimary -0.0614  0.0255 -0.1125 -0.0623 -0.0128

Smooth terms variances:
               Mean     Sd   2.5%    50%  97.5%    Min    Max
sx(agechild) 0.0059 0.0057 0.0012 0.0043 0.0193 0.0005 0.0701
sx(district) 0.0355 0.0185 0.0100 0.0319 0.0823 0.0035 0.1317
sx(mbmi)     0.0019 0.0025 0.0003 0.0011 0.0079 0.0002 0.0319

Random effects variances:
               Mean     Sd   2.5%    50%  97.5%    Min    Max
sx(district) 0.0073 0.0059 0.0006 0.0056 0.0215 0.0002 0.0374

Scale estimate:
         Mean     Sd   2.5%    50%  97.5%
Sigma2 0.8019 0.0165 0.7718 0.8017 0.8342

N = 4847  burnin = 2000  DIC = 4903.827  pd = 50.69312
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

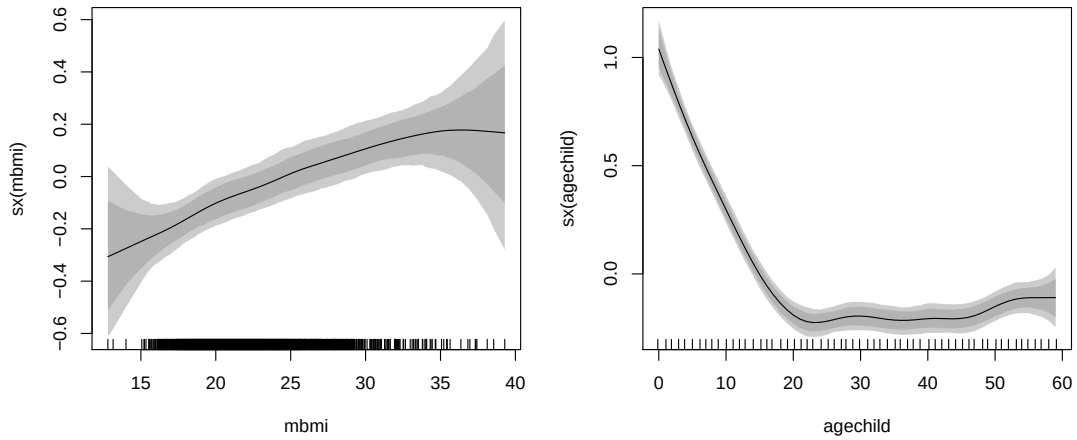

Figure 3: Example on childhood malnutrition: Effect of the body mass index of the child's mother and of the age of the child together with pointwise 80% and 95% credible intervals.

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 3. The interpretations of both terms are essentially unchanged compared to the simpler model considered in Section 2: The age of the child has a larger effect on stunting while mother's BMI could also be modeled appropriately by a linear term.

A visual representation of the posterior means for the structured and unstructured spatial effects, respectively, can be obtained in two ways: via kernel density estimates or using shaded maps. The former can be obtained be the plain plot function yielding both panels of Figure 4:

```
R> plot(zm, term = c("sx(district):mrf", "sx(district):re"))
```

Note that here the `term` labels have been extended by their respective basis specifications (`mrf` and `re`) to make the labels unique. Equivalently, `term` can also be specified by the corresponding index (based on the ordering in the model formula), e.g., `term = c(7, 8)`. In Figure 4, the kernel densities reveal the general form of the random effects distributions which are assumed to follow a Gaussian distribution. The range of the estimated random spatial effect is much smaller than the range of the structured spatial effect, indicating that model fit improvement by including random effects that account for unobserved spatial heterogeneity of the regions in Zambia, is relatively low. This is also supported by the comparatively low variance estimate of the random effects term given in the model summary above.

Alternatively, to view the spatial structure of the correlated effect the plot function can be used in combination with the boundary object `ZambiaBnd` yielding the map effect plot in the left panel of Figure 5:

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

As a default the districts of Zambia are colored in a symmetrical range within the estimated $\pm \max(|\text{posterior mean}|)$ of the corresponding effect. In many situations the visual impression

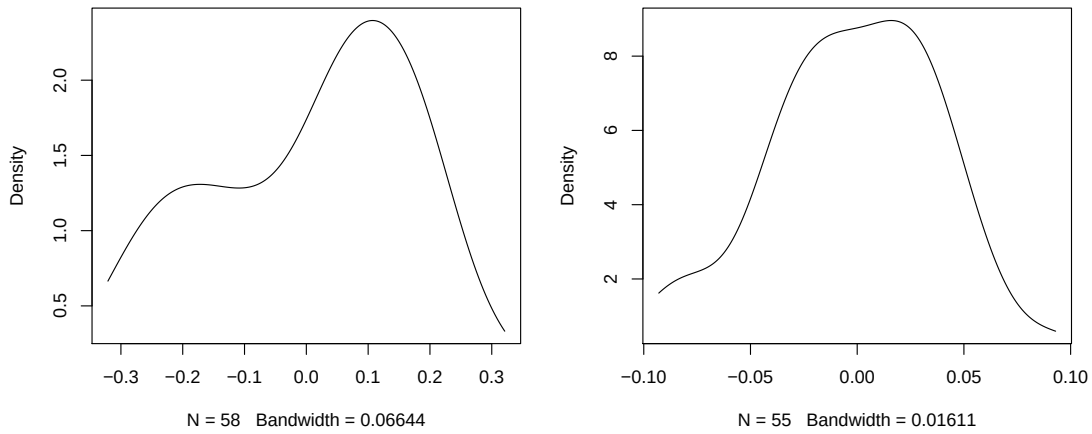

Figure 4: Example on childhood malnutrition: Kernel density estimates of the mean of the structured, left panel, and the unstructured spatial effect, right panel respectively.

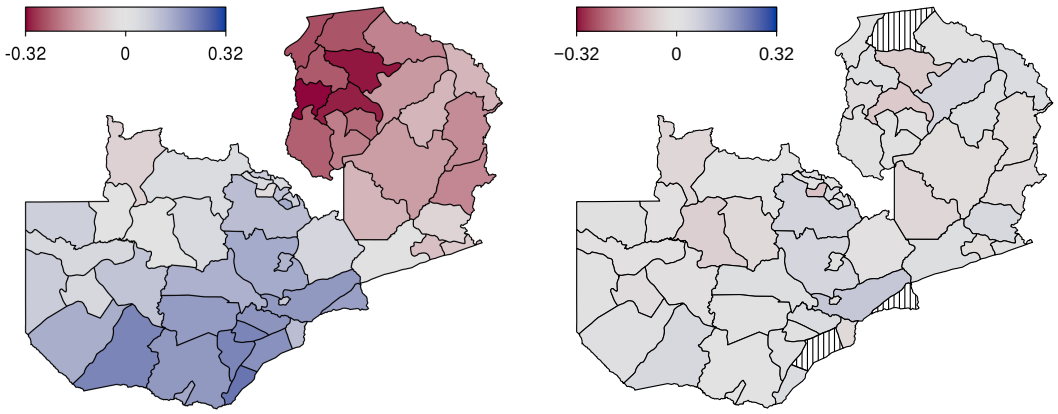

Figure 5: Example on childhood malnutrition: Estimated mean effect of the structured spatial effect (left panel), together with the unstructured spatial effect using the color and legend scaling of the structured effect (right panel).

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, yielding the right panel of Figure 5:

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

Using the same scale for both the structured and the unstructered effect is useful for comparison. In most cases one of the two effects clearly dominates the other. In our case the

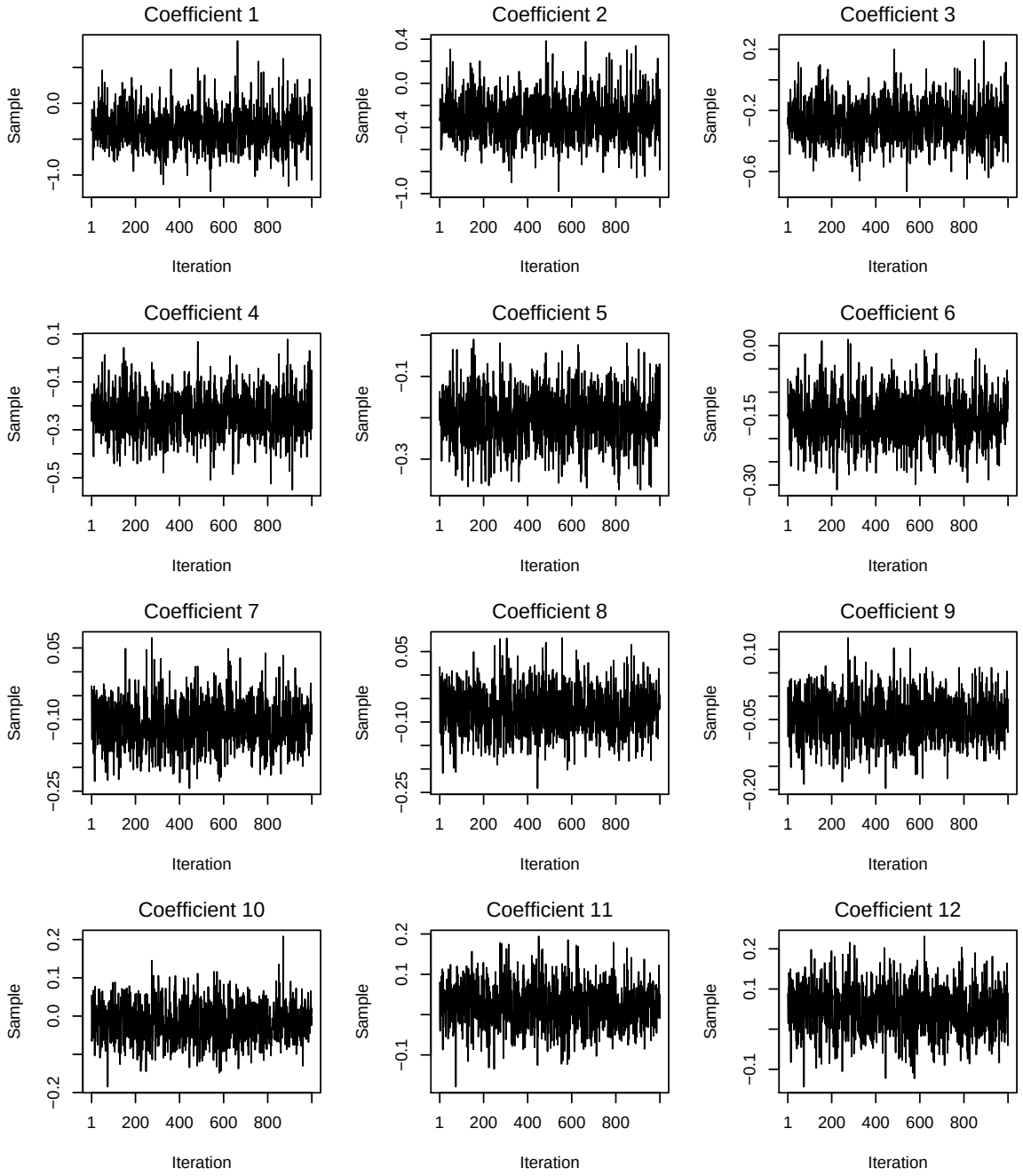

Figure 6: Example on childhood malnutrition: Sampling paths of the first 12 coefficients of term `sx(mbmi)`.

structured spatially correlated effect clearly exceeds the unstructured effect.

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)
```

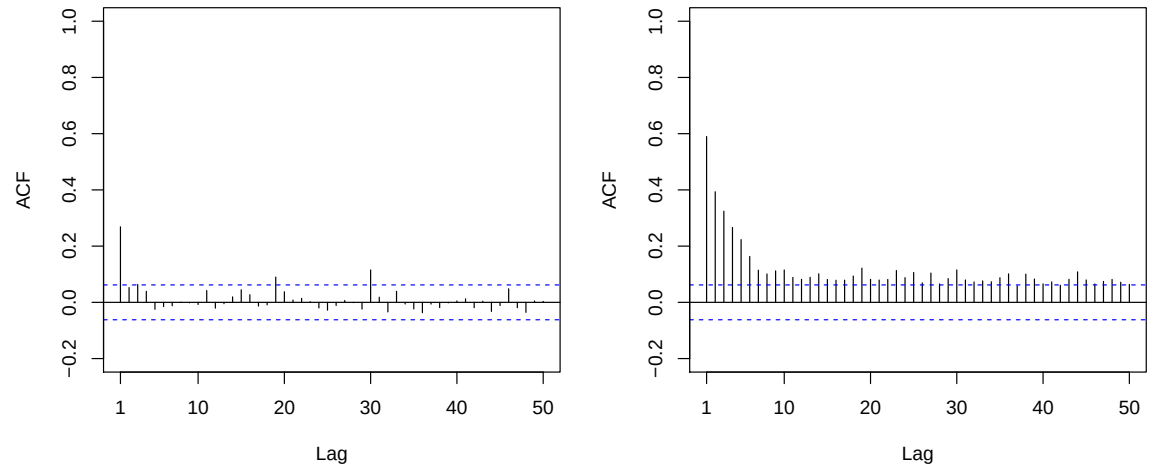

Figure 7: Example on childhood malnutrition: Autocorrelation function of the samples of the variance parameter of term `sx(mbmi)` (left panel) and maximum autocorrelation of all parameters of the model (right panel).

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 6. 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. 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 7, 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** fails in the estimation of the model. 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
> 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


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 were collected annually from 1980 to 1997 during a project of visual inspection of trees around Rothenbuch, Germany, see Göttlein and Pruscha (1996), and is discussed in detail in Fahrmeir *et al.* (2009). In this example, the percentage rate of defoliation of each tree is aggregated into three ordinal

Variable	Description
`id`	Tree location identification number.
`year`	Year of census.
`defoliation`	Percentage of tree defoliation in three ordinal categories: '$<$ 12.5%', '12.5% $\leq$ defoliation $<$ 50%', '$\geq$ 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.

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 6.

Similar to Fahrmeir *et al.* (2009), we start with a threshold model and cumulative logit link, with $P(\texttt{defoliation}_{it} \leq r)$ of tree i at time t, for response category $r = 1, 2$, and the additive predictor

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

where $f_1, \ldots, f_5$ are possibly nonlinear smooth functions of the continuous covariates and $\mathbf{x}_{it}^{\top}\boldsymbol{\gamma}$ 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

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

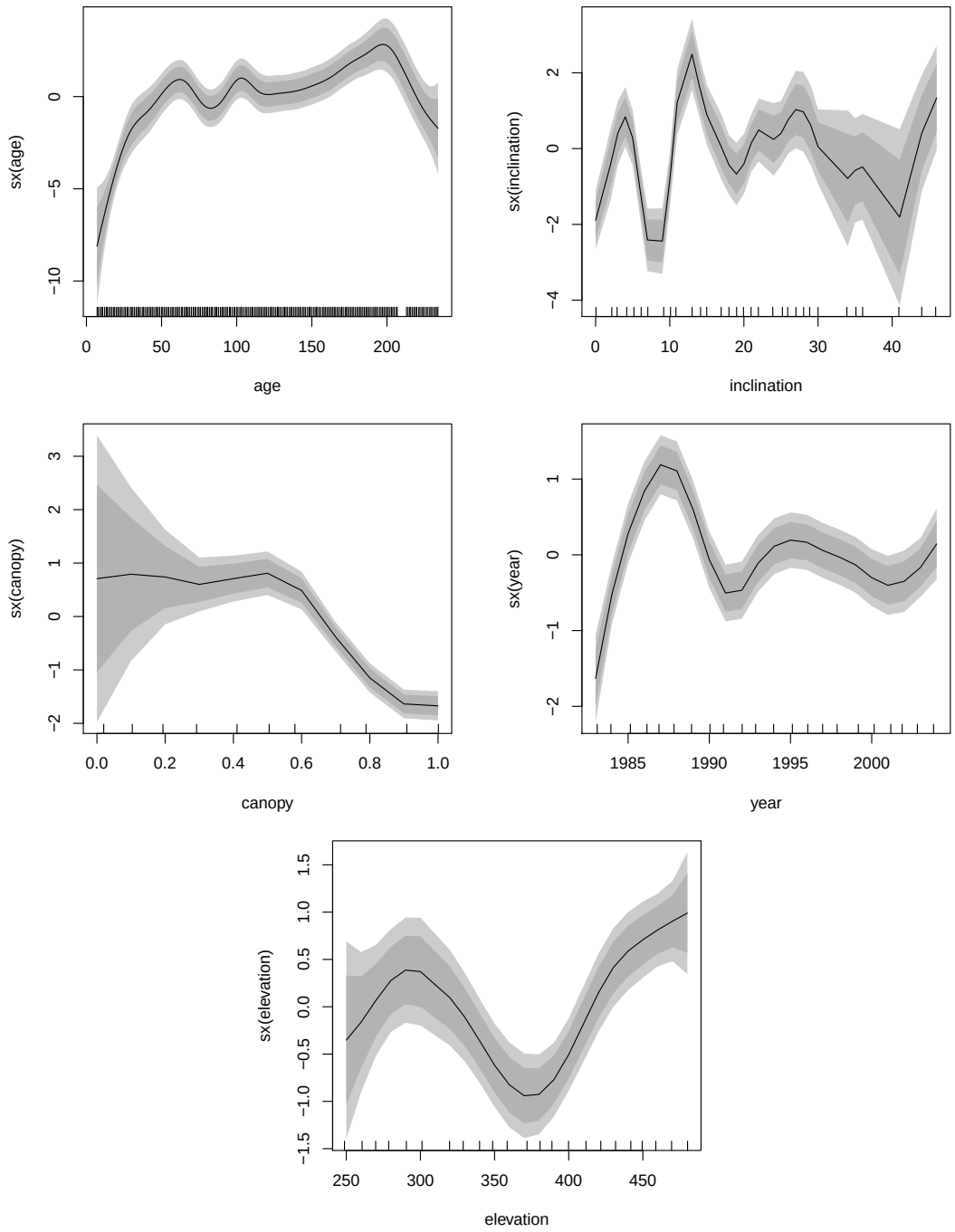

Figure 8: Forest damage: Estimates of nonparametric effects including 80% and 95% pointwise confidence intervals of the model without the spatial effect.

The results are shown in Figure 8 and appear to be rather unintuitive. In particular, the effect of the covariate `age` on `defoliation` seems to be non-monotonic with low defoliation levels for both younger and older trees. Similarly, the effect of `elevation` is very non-monotonic with high defoliation for both low and high elevations. Finally, the extremely wiggly estimate of `inclination` is hardly interpretable. Therefore, Göttlein and Pruscha (1996) 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 `BeechBnd`. We can update the model by adding a `"gs"` effect:

```
R> data("BeechBnd", package = "R2BayesX")
R> fm2 <- update(fm1, . ~ . + sx(id, bs = "gs", map = BeechBnd))
```

Taking a look at model information criteria

```
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
```

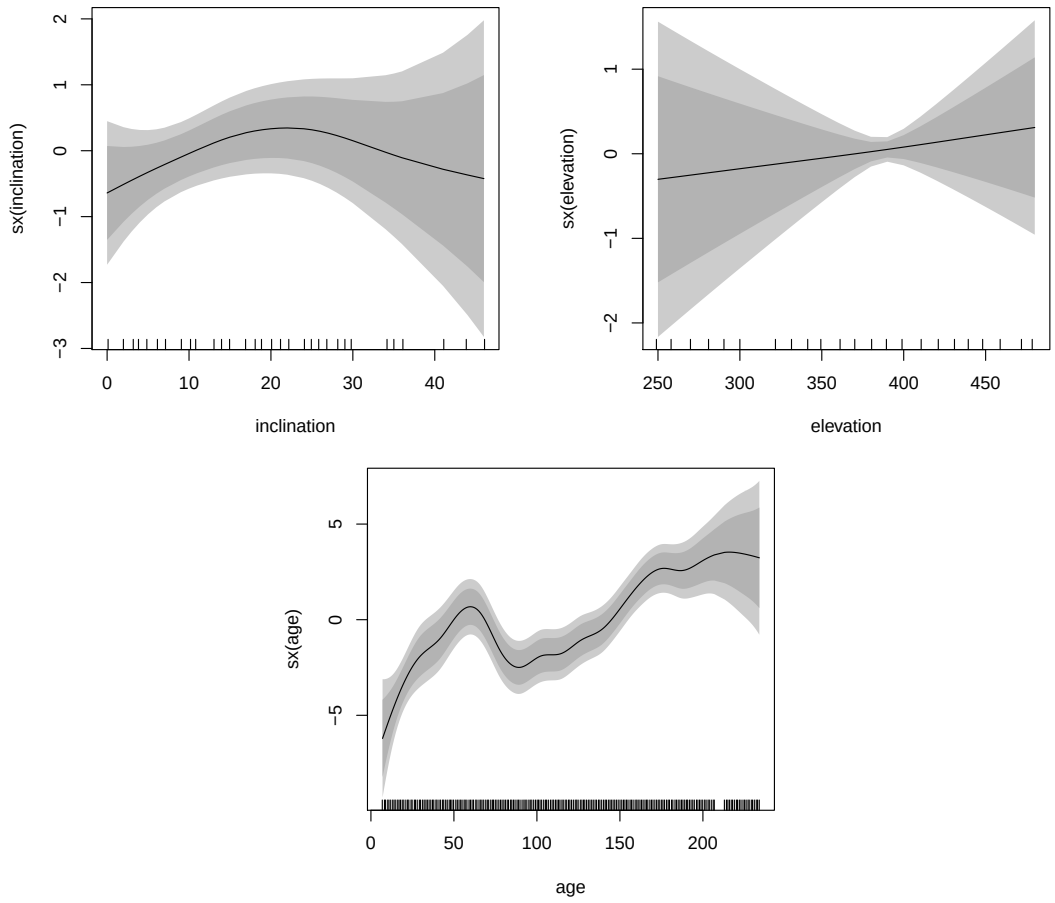

Figure 9: Forest damage: Estimated effects of covariates `inclination`, `elevation` and `age`, including 80% and 95% point-wise confidence intervals, of the model including the spatial effect.

```
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 9.

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 the first two panels of Figure 10. In this example the coloring of the plot is strongly influenced by a few very high and low values. Therefore, it is helpful to restrict the color range e.g., using the maximum shading for all absolute effects in excess of 3 (which roughly corresponds to the absolute values of the 10% and 90% quantiles of the kernel density estimate of the effect). The resulting map in the bottom panel of Figure 10 is created by:

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

Trimming the color range of the plot now leads to a better representation of the effect.

In summary, the results identify a strong influence of the spatial effect on the overall model fit, indicating that a clear splitting of location-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 6.1. In this example, the structured additive predictor (2) 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, to assess whether this is really necessary the corresponding 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 6.1 and call

```
R> f <- stunting ~ memployment + urban + gender +
+    sx(meducation, bs = "factor") + sx(mbmi) + sx(agechild) +
+    sx(district, bs = "mrf", map = ZambiaBnd) + sx(district, bs = "re")
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 2 for all possible options. Usually the selected final model is pretty much insensitive with respect to the selection algorithm and startmodel. However, it is generally of interest to assess the dependence of results on the selection algorithm and the startmodel. The summary statistics of the final selected model are then provided with

```
R> summary(zms)
```

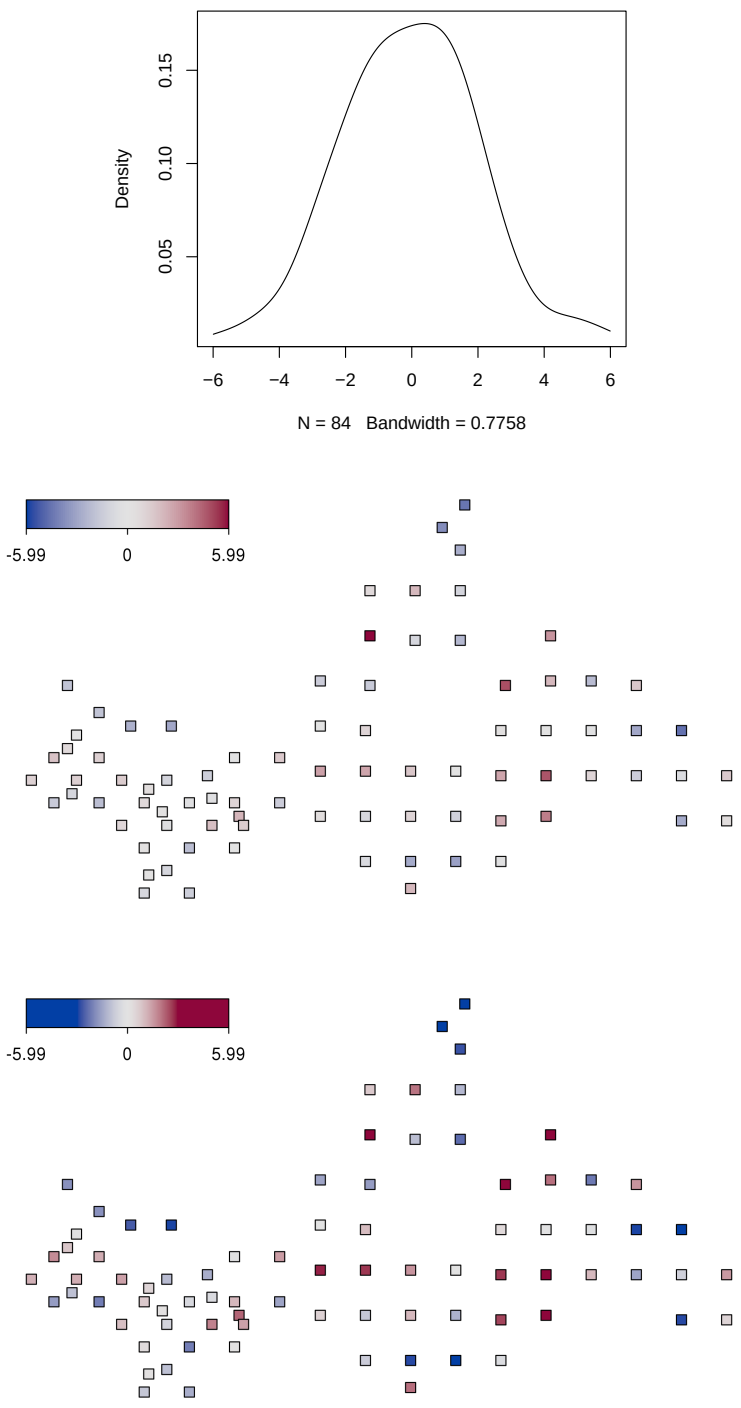

Figure 10: Forest damage: Kernel density estimate of the spatial effect (top panel), together with a map effect plot (middle panel), and map effect plot with modified color scaling (bottom panel). In the latter plot, the maximum shading is attained at -3 and 3, respectively, corresponding to the 10% and 90% quantiles of the kernel density estimate of the effect.

```
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)  -0.4863  0.0000  0.0000  0.0000     0
urbanno      -0.0945  0.0000  0.0000  0.0000     0
genderfemale  0.0589  0.0000  0.0000  0.0000     0
meducation_0 -0.1087  0.0000  0.0000  0.0000     0
meducation_2  0.2977  0.0000  0.0000  0.0000     0
mbmi          0.0209  0.0000  0.0000  0.0000     0

Smooth terms:
              lambda     df
f(agechild)  14.9489 11.011
f(district)   7.5775 24.366
re(district) 35.6851 17.872

Scale estimate: 0.7897

N = 4847  AIC_imp = -1024.37  method = STEP  family = gaussian
```

Thus, the results are similar to those from model `zm` in Section 6.1. However, the variable `memployment` is removed from the model and variable `mbmi` is modeled by a linear effect.

By default, the columns `sd`, `2.5%`, `50%` and `97.5%` from a `"STEP"` fit 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 a 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)  -0.4852  0.0976 -0.6748 -0.4836 -0.2885
urbanno      -0.0959  0.0242 -0.1415 -0.0966 -0.0456
genderfemale  0.0597  0.0130  0.0343  0.0601  0.0850
meducation_0 -0.1086  0.0290 -0.1643 -0.1077 -0.0493
meducation_2  0.2976  0.0675  0.1726  0.2976  0.4252
mbmi          0.0209  0.0042  0.0127  0.0208  0.0294

Smooth terms:
              lambda     df
f(agechild)  14.9489 11.011
f(district)   7.5775 24.366
re(district) 35.6851 17.872

Scale estimate: 0.7897

N = 4847  DIC = 4964.55  pd = 58.7637  AIC_imp = -1024.37
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 variation of this model would be to start from a `"userdefined"` instead of an `"empty"` `startmodel` (see also Table 2 for further options). In the `"userdefined"` case Besides the default of an `"empty"` `startmodel`, it may be reasonable to start with a the initial degrees of freedom (complexity or roughness) in the search for the nonlinearly modeled terms can be supplied. For example, the starting values for the degrees of freedom of the P-spline, spatial and random effect terms can be specified via

```
R> f <- stunting ~ memployment + urban + gender +
+    sx(meducation, bs = "factor") + sx(mbmi, dfstart = 2) +
+    sx(district, bs = "mrf", map = ZambiaBnd, dfstart = 5) +
+    sx(district, bs = "re", 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 (`zms`) again.

7. Summary

The R package **R2BayesX** provides an interface to the standalone software package **BayesX** for estimation of structured additive regression (STAR) models via MCMC, REML, or stepwise selection. The interface has the usual "look & feel" of regression modeling functions in R with a `formula`-based fitting function `bayesx()` along with suitable methods such as `summary()` and `plot()`. Adapting functionality from the **mgcv** package, the package allows for specification of regressions with smooth terms via the `s()` constuctor and adds its own smooth term constructor `sx()`, which facilitates specification of **BayesX**-specific terms along with corresponding optional control arguments. Moreover, the software design is modular enabling the import of already existing **BayesX** fitted-model files or the execution of previously generated **BayesX** program files from R. For post-estimation analysis and graphical inspection, the suite of methods allows for extraction of summary statistics and fitted model term objects as well as generation of 2d, 3d, image, and map effect plots, amongst others.

Acknowledgments

We would like to thank Fabian Scheipl for testing the **R2BayesX** package and giving fruitful feedback, and Uwe Ligges for sharing his expertise on compiling **BayesXsrc** package for Windows.

References

Adler D, Kneib T, Lang S, Umlauf N, Zeileis A (2012). ***BayesXsrc**: R Package Distribution of the **BayesX** C++ Sources*. R package version 2.0.1-1, URL http://CRAN.R-project.org/package=BayesXsrc.

Belitz C, Brezger A, Kneib T, Lang S (2009). ***BayesX** – Software for Bayesian Inference in Structured Additive Regression Models*. Version 2.0.1, URL http://www.BayesX.org/.

Belitz C, Lang S (2008). "Simultaneous Selection of Variables and Smoothing Parameters in Structured Additive Regression Models." *Computational Statistics & Data Analysis*, **53**, 61–81.

Bivand RS, Pebesma EJ, Gómez-Rubio V (2008). *Applied Spatial Data Analysis with R*. Springer-Verlag, New York. URL http://www.asdar-book.org/.

Brezger A, Kneib T, Lang S (2005). "**BayesX**: Analyzing Bayesian Structured Additive Regression Models." *Journal of Statistical Software*, **14**(11), 1–22. URL http://www.jstatsoft.org/v14/i11/.

Brezger A, Lang S (2006). "Generalized Structured Additive Regression Based on Bayesian P-Splines." *Computational Statistics & Data Analysis*, **50**, 947–991.

Burnham KP, Anderson DR (1998). *Model Selection and Multimodel Inference*. Springer-Verlag, New York.

Chambers JM, Hastie TJ (eds.) (1992). *Statistical Models in S*. Chapman & Hall, London.

Fahrmeir L, Kneib T, Lang S (2004). "Penalized Structured Additive Regression for Space Time Data: A Bayesian Perspective." *Statistica Sinica*, **14**, 731–761.

Fahrmeir L, Kneib T, Lang S (2009). *Regression – Modelle, Methoden und Anwendungen*. 2nd edition. Springer-Verlag, Berlin.

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

Fotheringham AS, Brunsdon C, Charlton ME (2002). *Geographically Weighted Regression: The Analysis of Spatially Varying Relationships*. John Wiley & Sons, Chichester.

Göttlein A, Pruscha H (1996). "Der Einfuss von Bestandskenngrössen, Topographie, Standord und Witterung auf die Entwicklung des Kronenzustandes im Bereich des Forstamtes Rothenbuch." *Forstwissenschaftliches Zentralblatt*, **114**, 146–162.

Hastie T (2011). ***gam**: Generalized Additive Models*. R package version 1.06.2, URL http://CRAN.R-project.org/package=gam.

Hastie T, Tibshirani R (1990). *Generalized Additive Models*. Chapman & Hall/CRC.

Hastie T, Tibshirani R (1993). "Varying-Coefficient Models." *Journal of the Royal Statistical Society B*, **55**, 757–796.

Hastie TJ, Tibshirani RJ, Friedman J (2009). *The Elements of Statistical Learning*. 2nd edition. Springer-Verlag, New York.

Ihaka R, Murrell P, Hornik K, Zeileis A (2012). ***colorspace**: Color Space Manipulation*. R package version 1.1-1, URL http://CRAN.R-project.org/package=colorspace.

Kamman EE, Wand MP (2003). "Geoadditive Models." *Journal of the Royal Statistical Society C*, **52**, 1–18.

Kandala NB, Lang S, Klasen S, 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.

Kneib T, Fahrmeir L (2006). "Structured Additive Regression for Multicategorical Space-Time Data: A Mixed Model Approach." *Biometrics*, **62**, 109–118.

Kneib T, Heinzl F, Brezger A, Sabanes Bove D (2011). ***BayesX**: R Utilities Accompanying the Software Package **BayesX***. R package version 0.2-5, URL http://CRAN.R-project.org/package=BayesX.

Krivobokova T (2009). ***AdaptFit**: Adaptive Semiparametic Regression*. R package version 0.2-2, URL http://CRAN.R-project.org/package=AdaptFit.

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 JA (1989). *Generalized Linear Models*. 2nd edition. Chapman & Hall, London.

Pebesma EJ, Bivand RS (2005). "Classes and Methods for Spatial Data in R." *R News*, **5**(2), 9–13. URL http://CRAN.R-project.org/doc/Rnews/.

Plummer M, Best N, Cowles K, Vines K (2006). "**coda**: Convergence Diagnosis and Output Analysis for MCMC." *R News*, **6**(1), 7–11. URL http://CRAN.R-project.org/doc/Rnews/.

R Development Core Team (2011). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-07-0, URL http://www.R-project.org/.

Rigby RA, Stasinopoulos DM (2005). "Generalized Additive Models for Location, Scale and Shape." *Journal of the Royal Statistical Society C*, **54**(3), 507–554.

Ruppert D, Wand MP, Carrol RJ (2003). *Semiparametric Regression*. Cambridge University Press, New York.

Scheipl F (2011). "**spikeSlabGAM**: Bayesian Variable Selection, Model Choice and Regularization for Generalized Additive Mixed Models in R." *Journal of Statistical Software*, **43**(14), 1–24. URL http://www.jstatsoft.org/v43/i14/.

Stabler B (2006). ***shapefiles**: Read and Write ESRI Shapefiles*. R package version 0.6, URL http://CRAN.R-project.org/package=shapefiles.

Stasinopoulos DM, Rigby RA (2007). "Generalized Additive Models for Location Scale and Shape (GAMLSS) in R." *Journal of Statistical Software*, **23**(7), 1–46. URL http://www.jstatsoft.org/v23/i07/.

Wand M (2010). ***SemiPar**: Semiparametic Regression*. R package version 1.0-3, URL http://CRAN.R-project.org/package=SemiPar.

Wood SN (2006). *Generalized Additive Models: An Introduction with R*. Chapman & Hall/CRC.

Wood SN (2011). "Fast Stable Restricted Maximum Likelihood and Marginal Likelihood Estimation of Semiparametric Generalized Linear Models." *Journal of the Royal Statistical Society B*, **73**(3), 3–36.

Wood SN (2012). ***mgcv**: GAMs with GCV/AIC/REML Smoothness Estimation and GAMMs by PQL*. R package version 1.7-13, URL http://CRAN.R-project.org/package=mgcv.

Yee TW (2010). "The **VGAM** Package for Categorical Data Analysis." *Journal of Statistical Software*, **32**(10), 1–34. URL http://www.jstatsoft.org/v32/i10/.

Yee TW, Wild CJ (1996). "Vector Generalized Additive Models." *Journal of the Royal Statistical Society B*, **58**(3), 481–493.

Zeileis A, Hornik K, Murrell P (2009). "Escaping RGBland: Selecting Colors for Statistical Graphics." *Computational Statistics & Data Analysis*, **53**, 3259–3270.

A. BayesXsrc: Packaging the BayesX C++ sources for R

BayesX was originally developed under the Borland C++ compiler and is distributed as a Windows application with a Java-based user interface. Since version 2.0, it is also offers a command line version comprising the interpreter and modules for computations. The sources have been modified to be compliant with the GNU Compiler Collection (GCC), and the software was ported to run on Linux, Mac OS X, Windows and several BSDs.

Our objective with the R package **BayesXsrc** is to offer R users a convenient way to download, build, and install the open-source **BayesX** software as if this were an ordinary R package, and for offering prebuilt binary versions of **BayesX** through the CRAN build servers for major R platforms.

To accomplish this goal, **BayesXsrc** comes with a tiny R package hull, within which the **BayesX** sources for the command line version are embedded. In order to compile the **BayesX** sources with the R build system (e.g., via `R CMD INSTALL`), `Makefile`s under `src/` are utilized to compile the sources stored at `src/bayesxsrc`. The current source tree of **BayesX** requires a slightly different setting of compile flags for Windows which is achieved by using the two standard locations for R `Makefile`s: `src/Makefile.win` for Windows and `src/Makefile` otherwise. Since R 2.13.1 the package installation was enhanced to support non-standard installation of compiled code via an R installation script. If an R script `src/install.libs.R` is found, it will be executed after successful compilation. We make use of this feature to copy the binary executable to the package installation directory in an architecture-specific subfolder to support multi-architecture installations using R as a cross-platform portable install shell:

```
binary <- if(WINDOWS) "BayesX.exe" else "BayesX"
if(file.exists(binary)) {
  libarch <- if (nzchar(R_ARCH)) paste("libs", R_ARCH, sep = "") else "libs"
  dest <- file.path(R_PACKAGE_DIR, libarch)
  dir.create(dest, recursive = TRUE, showWarnings = FALSE)
  file.copy(binary, dest, overwrite = TRUE)
}
```

Since the executable exists in a designated location within the installed **BayesXsrc** package, we can run the command line version from within R via a single front-end function `run.bayesx()`.

Note that the package hull of **BayesXsrc** (without `inst/bayesxsrc`) is maintained in a subversion (SVN) repository on R-Forge (at https://R-Forge.R-project.org/projects/bayesr/, along with **R2BayesX**). It enables simple creation of development snapshots of **BayesX**: The **BayesXsrc** sources in the SVN provide a top-level `bootstrap.sh` shell script that pulls the **BayesX** sources from its SVN repository (at http://svn.gwdg.de/svn/bayesx/) and stores them in the `src/bayesxsrc` directory. Subsequently, the R source package for **BayesXsrc** can be created as usual via `R CMD build`.

Although we do not effectively distribute an R package in the usual sense, we use the R package system as a cross-platform build system. The installation via an R package is an attractive alternative, in particular for software that aims to be embedded to R. Potential support for distribution and delivery of self-contained software in form of sources and prebuilt binaries via CRAN is very attractive to end users but also to smaller development teams (like us) that would otherwise have no resources for multi-platform builds and tests.

B. Options for the `plot()` method

Objects of class "`bayesx`" returned either from function `bayesx()` or `read.bayesx.output()` have a method for the `plot()` generic. Depending on the structure of the "`bayesx`" object, the method identifies the various types of inherent model terms and applies one of the following implemented plotting functions: `plot2d()`, `plot3d()` or `plotmap()`. Using the method with-

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 $\pm \max(\lvert x \rvert)$ of values x that are used for plotting.

Table 7: Most important arguments of the `plot()` method for "`bayesx`" objects. The first block describes arguments of the `plot()` method itself, subsequent blocks arguments that are passed to `plot2d()`, `plot3d()`, `plotmap()`, and `colorlegend()`, respectively.

out further specifications will produce a plot of all estimated effects. For individual effect plots argument `term` is used. For MCMC estimated models argument `which` is useful to inspect sampling paths of coefficients, but also to view basic residual diagnostic plots. To build map effect plots using `plotmap()`, a map needs to be supplied to argument `map`. The `map` must be an object of "`bnd`" or "`SpatialPolygonsDataFrame`". Per default, similar to 2d plots, map effect plots are colored using a diverging color legend where the `range` is set `symmetrical`, e.g. according to the $\pm$ max(|posterior mean|) of the effect. In this setting it is easier to distinguish between regions of large and no influence. The most important options of the plotting method are shown in Table 7, for a detailed description of all available arguments and options please see documentation of function `plot.bayesx()`, `plot2d()`, `plot3d()`, `plotmap()` and `colorlegend()`.

Affiliation:

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

Thomas Kneib, Daniel Adler
Department of Statistics and Econometrics
Georg-August-Universität Göttingen
Platz der Göttinger Sieben 5 (MZG)
37073 Göttingen, Germany
E-mail: tkneib@uni-goettingen.de
dadler@uni-goettingen.de,
URL: http://www.uni-goettingen.de/en/264255.html,
http://neoscientists.org/~dadler/