

Package ‘R2BayesX’

May 7, 2026

Version 1.1-6

Date 2025-03-25

Title Estimate Structured Additive Regression Models with 'BayesX'

Description An R interface to estimate structured additive regression (STAR) models with 'BayesX'.

Depends R (>= 2.13.0), BayesXsrc, colorspace, mgcv

Suggests interp, coda, maps, MBA, parallel, sf, shapefiles, sp, spdep,
splines, spData (>= 2.3.1), fields

Imports methods

License GPL-2 | GPL-3

LazyLoad yes

NeedsCompilation yes

Author Nikolaus Umlauf [aut, cre] (ORCID:

<<https://orcid.org/0000-0003-2160-9803>>),

Thomas Kneib [aut],

Stefan Lang [aut],

Achim Zeileis [aut] (ORCID: <<https://orcid.org/0000-0003-0918-3766>>)

Maintainer Nikolaus Umlauf <Nikolaus.Umlauf@uibk.ac.at>

Repository CRAN

Date/Publication 2025-03-25 17:50:05 UTC

Contents

R2BayesX-package	3
add.neighbor	3
bayesx	4
bayesx.construct	12
bayesx.control	16
bayesx.term.options	20
bayesx_logfile	22
bayesx_prgfile	23
bayesx_runtime	24

BeechBnd	25
BeechGra	26
bnd2gra	26
colorlegend	27
cprob	31
delete.neighbor	32
DIC	33
FantasyBnd	34
fitted.bayesx	35
ForestHealth	37
GAMart	39
GCV	40
GermanyBnd	41
get.neighbor	42
getscript	42
GRstats	44
Interface between nb and gra format	45
Interface between sp and bnd format	46
MunichBnd	48
parse.bayesx.input	48
plot.bayesx	49
plot2d	53
plot3d	56
plotblock	59
plotmap	62
plotsamples	67
predict.bayesx	69
read.bayesx.output	71
read.bnd	72
read.gra	74
samples	76
shp2bnd	77
sliceplot	79
summary.bayesx	81
sx	82
term.freqs	87
write.bayesx.input	88
write.bnd	89
write.gra	90
ZambiaBnd	91
ZambiaNutrition	92

Description

This package interfaces the **BayesX** (<https://www.uni-goettingen.de/de/bayesx/550513.html>) command-line binary from R. The main model fitting function is called `bayesx`.

Before STAR models can be estimated, the command-line version of **BayesX** needs to be installed, which is done by installing the R source code package **BayesXsrc**. Please see function `bayesx` and `bayesx.control` for more details on model fitting and controlling.

The package also provides functionality for high level graphics of estimated effects, see function `plot.bayesx`, `plot2d`, `plot3d`, `plotblock`, `plotmap`, `plotsamples` and `colorlegend`.

More standard extractor functions and methods for the fitted model objects may be applied, e.g., see function `summary.bayesx`, `fitted.bayesx`, `residuals.bayesx`, `samples`, `plot.bayesx`, as well as `AIC`, `BIC` etc., please see the examples of the help sites. Predictions for new data based on refitting with weights can be obtained by function `predict.bayesx`.

In addition, it is possible to run arbitrary **BayesX** program files using function `run.bayesx`. **BayesX** output files that are stored in a directory may be read into R calling function `read.bayesx.output`.

Author(s)

Nikolaus Umlauf, Thomas Kneib, Stefan Lang, Achim Zeileis.

Examples

```
## to see the package demos
demo(package = "R2BayesX")
```

Description

Adds a neighborhood relationship between two given regions to a map object in graph format.

Usage

```
add.neighbor(map, region1, region2)
```

Arguments

`map` map object in graph format that should be modified.
`region1, region2` character, names of the regions that should be connected as neighbors.

Value

Returns an adjacency matrix that represents the neighborhood structure of map plus the new neighborhood relation in graph format.

Author(s)

Felix Heinzl, Thomas Kneib.

See Also

[get.neighbor](#), [delete.neighbor](#), [read.gra](#), [write.gra](#), [bnd2gra](#).

Examples

```
## read the graph file
file <- file.path(find.package("R2BayesX"), "examples", "Germany.gra")
germany <- read.gra(file)

## add some neighbors
get.neighbor(germany, c("1001", "7339"))
germany <- add.neighbor(germany, "7339", "1001")
get.neighbor(germany, c("1001", "7339"))
```

bayesx

Estimate STAR Models with BayesX

Description

This is the documentation of the main model fitting function of the interface. Within function `bayesx`, three inferential concepts are available for estimation: Markov chain Monte Carlo simulation (MCMC), estimation based on mixed model technology and restricted maximum likelihood (REML), and a penalized least squares (respectively penalized likelihood) approach for estimating models using model selection tools (STEP).

Usage

```
bayesx(formula, data, weights = NULL, subset = NULL,
       offset = NULL, na.action = NULL, contrasts = NULL,
       control = bayesx.control(...), model = TRUE,
       chains = NULL, cores = NULL, ...)
```

Arguments

`formula` symbolic description of the model (of type $y \sim x$), also see [sx](#), [formula.gam](#) and [S](#).

data	a <code>data.frame</code> or <code>list</code> containing the model response variable and covariates required by the formula. By default the variables are taken from <code>environment(formula)</code> : typically the environment from which <code>bayesx</code> is called. Argument <code>data</code> may also be a character string defining the directory the data is stored, where the first row in the data set must contain the variable names and columns should be tab separated. Using this option will avoid loading the complete data into R, only the BayesX output files will be imported, which might be helpful using large datasets.
weights	prior weights on the data.
subset	an optional vector specifying a subset of observations to be used in the fitting process.
offset	can be used to supply a model offset for use in fitting.
na.action	a function which indicates what should happen when the data contain NA's. The default is set by the <code>na.action</code> setting of <code>options</code> , and is <code>na.omit</code> if set to NULL.
contrasts	an optional list. See the <code>contrasts.arg</code> of <code>model.matrix.default</code> .
control	specify several global control parameters for <code>bayesx</code> , see <code>bayesx.control</code> .
model	a logical value indicating whether <code>model.frame</code> should be included as a component of the returned value.
chains	integer. The number of sequential chains that should be run, the default is one chain if <code>chains = NULL</code> . For each chain a separate seed for the random number generator is used. The return value of <code>bayesx</code> is a list of class "bayesx", i.e. each list element represents a separate model, for which the user can e.g. apply all plotting methods or extractor functions. Convergence diagnostics can then be computed using function <code>GRstats</code> .
cores	integer. How many cores should be used? The default is one core if <code>cores = NULL</code> . The return value is again a list of class "bayesx", for which all plotting and extractor functions can be applied, see argument <code>chains</code> . Note that this option is not available on Windows systems, see the documentation of function <code>mclapply</code> .
...	arguments passed to <code>bayesx.control</code> , e.g. <code>family</code> and <code>method</code> , defaults are <code>family = "gaussian"</code> , <code>method = "MCMC"</code> .

Details

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

Full Bayesian inference via MCMC: A fully Bayesian interpretation of structured additive regression models is obtained by specifying prior distributions for all unknown parameters. Estimation can be facilitated using Markov chain Monte Carlo simulation techniques. **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.

Inference via a mixed model representation: The other concept used for estimation is based on mixed model methodology. Within **BayesX** this concept has been extended to structured additive regression models and several types of non-standard regression situations. The general idea is to

take advantage of the close connection between penalty concepts and corresponding random effects distributions. The smoothing parameters of the penalties then transform to variance components in the random effects (mixed) model. While the selection of smoothing parameters has been a difficult task for a long time, several estimation procedures for variance components in mixed models are already available since the 1970's. The most popular one is restricted maximum likelihood in Gaussian mixed models with marginal likelihood as the non-Gaussian counterpart. 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 structured additive regression models. However, estimates can also merely be interpreted as penalized likelihood estimates from a frequentist perspective.

Penalized likelihood including variable selection: As a third alternative **BayesX** provides a penalized least squares (respectively penalized likelihood) approach for estimating structured additive regression models. In addition, a powerful variable and model selection tool is included. Model choice and estimation of the parameters is done simultaneously. The algorithms are able to

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

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

Within the model fitting function `bayesx`, the different inferential concepts may be chosen by argument method of function `bayesx.control`. Options are "MCMC", "REML" and "STEP".

The wrapper function `bayesx` basically starts by setting up the necessary **BayesX** program file using function `bayesx.construct`, `parse.bayesx.input` and `write.bayesx.input`. Afterwards the generated program file is send to the command-line binary executable version of **BayesX** with `run.bayesx`. As a last step, function `read.bayesx.output` will read the estimated model object returned from **BayesX** back into R.

For estimation of STAR models, function `bayesx` uses formula syntax as provided in package `mgcv` (see `formula.gam`), i.e., models may be specified using the **R2BayesX** main model term constructor functions `sx` or the `mgcv` constructor functions `s`. For a detailed description of the model formula syntax used within `bayesx` models see also `bayesx.construct` and `bayesx.term.options`.

After the **BayesX** binary has successfully finished processing an object of class "bayesx" is returned, wherefore a set of standard extractor functions and methods is available, including methods to the generic functions `print`, `summary`, `plot`, `residuals` and `fitted`.

See `fitted.bayesx`, `plot.bayesx`, and `summary.bayesx` for more details on these methods.

Value

A list of class "bayesx", see function `read.bayesx.output`.

WARNINGS

For geographical effects, note that **BayesX** may crash if the region identification covariate is a [factor](#), it is recommended to code these variables as [integer](#), please see the example below.

Note

If a model is specified with a structured and an unstructured spatial effect, e.g. the model formula is something like $y \sim \text{sx}(\text{id}, \text{bs} = \text{"mrf"}, \text{map} = \text{MapBnd}) + \text{sx}(\text{id}, \text{bs} = \text{"re"})$, the model output contains of one additional total spatial effect, named with `"sx(id):total"`. Also see the last example.

Author(s)

Nikolaus Umlauf, Thomas Kneib, Stefan Lang, Achim Zeileis.

References

Belitz C, Brezger A, Kneib T, Lang S (2011). **BayesX** - Software for Bayesian Inference in Structured Additive Regression Models. Version 2.0.1. URL <https://www.uni-goettingen.de/de/bayesx/550513.html>.

Belitz C, Lang S (2008). Simultaneous selection of variables and smoothing parameters in structured additive regression models. *Computational Statistics & Data Analysis*, **53**, 61–81.

Brezger A, Kneib T, Lang S (2005). **BayesX**: Analyzing Bayesian Structured Additive Regression Models. *Journal of Statistical Software*, **14**(11), 1–22. URL <https://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.

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

Umlauf N, Adler D, Kneib T, Lang S, Zeileis A (2015). Structured Additive Regression Models: An R Interface to BayesX. *Journal of Statistical Software*, **63**(21), 1–46. <https://www.jstatsoft.org/v63/i21/>

See Also

[parse.bayesx.input](#), [write.bayesx.input](#), [run.bayesx](#), [read.bayesx.output](#), [summary.bayesx](#), [plot.bayesx](#), [fitted.bayesx](#), [bayesx.construct](#), [bayesx.term.options](#), [sx](#), [formula.gam](#), [s](#).

Examples

```
## generate some data
set.seed(111)
n <- 200

## regressor
dat <- data.frame(x = runif(n, -3, 3))

## response
```

```

dat$y <- with(dat, 1.5 + sin(x) + rnorm(n, sd = 0.6))

## estimate models with
## bayesx REML and MCMC
b1 <- bayesx(y ~ sx(x), method = "REML", data = dat)

## same using mgcv syntax
b1 <- bayesx(y ~ s(x, bs = "ps", k = 20), method = "REML", data = dat)

## now with MCMC
b2 <- bayesx(y ~ sx(x), method = "MCMC",
  iter = 1200, burnin = 200, data = dat)

## compare reported output
summary(c(b1, b2))

## plot the effect for both models
plot(c(b1, b2), residuals = TRUE)

## use confint
confint(b1, level = 0.99)
confint(b2, level = 0.99)

## Not run:
## more examples
set.seed(111)
n <- 500

## regressors
dat <- data.frame(x = runif(n, -3, 3), z = runif(n, -3, 3),
  w = runif(n, 0, 6), fac = factor(rep(1:10, n/10)))

## response
dat$y <- with(dat, 1.5 + sin(x) + cos(z) * sin(w) +
  c(2.67, 5, 6, 3, 4, 2, 6, 7, 9, 7.5)[fac] + rnorm(n, sd = 0.6))

## estimate models with
## bayesx MCMC and REML
## and compare with
## mgcv gam()
b1 <- bayesx(y ~ sx(x) + sx(z, w, bs = "te") + fac,
  data = dat, method = "MCMC")
b2 <- bayesx(y ~ sx(x) + sx(z, w, bs = "te") + fac,
  data = dat, method = "REML")
b3 <- gam(y ~ s(x, bs = "ps") + te(z, w, bs = "ps") + fac,
  data = dat)

## summary statistics
summary(b1)
summary(b2)
summary(b3)

## plot the effects

```

```

op <- par(no.readonly = TRUE)
par(mfrow = c(3, 2))
plot(b1, term = "sx(x)")
plot(b1, term = "sx(z,w)")
plot(b2, term = "sx(x)")
plot(b2, term = "sx(z,w)")
plot(b3, select = 1)
vis.gam(b3, c("z","w"), theta = 40, phi = 40)
par(op)

## combine models b1 and b2
b <- c(b1, b2)

## summary
summary(b)

## only plot effect 2 of both models
plot(b, term = "sx(z,w)")

## with residuals
plot(b, term = "sx(z,w)", residuals = TRUE)

## same model with kriging
b <- bayesx(y ~ sx(x) + sx(z, w, bs = "kr") + fac,
  method = "REML", data = dat)
plot(b)

## now a mrf example
## note: the regional identification
## covariate and the map regionnames
## should be coded as integer
set.seed(333)

## simulate some geographical data
data("MunichBnd")
N <- length(MunichBnd); n <- N*5

## regressors
dat <- data.frame(x1 = runif(n, -3, 3),
  id = as.factor(rep(names(MunichBnd), length.out = n)))
dat$sp <- with(dat, sort(runif(N, -2, 2), decreasing = TRUE)[id])

## response
dat$y <- with(dat, 1.5 + sin(x1) + sp + rnorm(n, sd = 1.2))

## estimate models with
## bayesx MCMC and REML
b1 <- bayesx(y ~ sx(x1) + sx(id, bs = "mrf", map = MunichBnd),
  method = "MCMC", data = dat)
b2 <- bayesx(y ~ sx(x1) + sx(id, bs = "mrf", map = MunichBnd),
  method = "REML", data = dat)

```

```
## summary statistics
summary(b1)
summary(b2)

## plot the spatial effects
plot(b1, term = "sx(id)", map = MunichBnd,
     main = "bayesx() MCMC estimate")
plot(b2, term = "sx(id)", map = MunichBnd,
     main = "bayesx() REML estimate")
plotmap(MunichBnd, x = dat$sp, id = dat$id,
        main = "Truth")

## try geosplines instead
b <- bayesx(y ~ sx(id, bs = "gs", map = MunichBnd) + sx(x1), data = dat)
summary(b)
plot(b, term = "sx(id)", map = MunichBnd)

## geokriging
b <- bayesx(y ~ sx(id, bs = "gk", map = MunichBnd) + sx(x1),
            method = "REML", data = dat)
summary(b)
plot(b, term = "sx(id)", map = MunichBnd)

## perspective plot of the effect
plot(b, term = "sx(id)")

## image and contour plot
plot(b, term = "sx(id)", image = TRUE,
     contour = TRUE, grid = 200)

## model with random effects
set.seed(333)
N <- 30
n <- N*10

## regressors
dat <- data.frame(id = sort(rep(1:N, n/N)), x1 = runif(n, -3, 3))
dat$re <- with(dat, rnorm(N, sd = 0.6)[id])

## response
dat$y <- with(dat, 1.5 + sin(x1) + re + rnorm(n, sd = 0.6))

## estimate model
b <- bayesx(y ~ sx(x1) + sx(id, bs = "re"), data = dat)
summary(b)
plot(b)

## extract estimated random effects
## and compare with true effects
plot(fitted(b, term = "sx(id)")$Mean ~ unique(dat$re))
```

```

## now a spatial example
## with structured and
## unstructured spatial
## effect
set.seed(333)

## simulate some geographical data
data("MunichBnd")
N <- length(MunichBnd); names(MunichBnd) <- 1:N
n <- N*5

## regressors
dat <- data.frame(id = rep(1:N, n/N), x1 = runif(n, -3, 3))
dat$sp <- with(dat, sort(runif(N, -2, 2), decreasing = TRUE)[id])
dat$re <- with(dat, rnorm(N, sd = 0.6)[id])

## response
dat$y <- with(dat, 1.5 + sin(x1) + sp + re + rnorm(n, sd = 0.6))

## estimate model
b <- bayesx(y ~ sx(x1) +
  sx(id, bs = "mrf", map = MunichBnd) +
  sx(id, bs = "re"), method = "MCMC", data = dat)
summary(b)

## plot all spatial effects
plot(b, term = "sx(id):mrf", map = MunichBnd,
  main = "Structured spatial effect")
plot(b, term = "sx(id):re", map = MunichBnd,
  main = "Unstructured spatial effect")
plot(b, term = "sx(id):total", map = MunichBnd,
  main = "Total spatial effect", digits = 4)

## some experiments with the
## stepwise algorithm
## generate some data
set.seed(321)
n <- 1000

## regressors
dat <- data.frame(x1 = runif(n, -3, 3), x2 = runif(n),
  x3 = runif(n, 3, 6), x4 = runif(n, 0, 1))

## response
dat$y <- with(dat, 1.5 + sin(x1) + 0.6 * x2 + rnorm(n, sd = 0.6))

## estimate model with STEP
b <- bayesx(y ~ sx(x1) + sx(x2) + sx(x3) + sx(x4),
  method = "STEP", algorithm = "cdescent1", CI = "MCMCselect",
  iter = 10000, step = 10, data = dat)
summary(b)
plot(b)

```

```

## a probit example
set.seed(111)
n <- 1000
dat <- data.frame(x <- runif(n, -3, 3))

dat$z <- with(dat, sin(x) + rnorm(n))
dat$y <- rep(0, n)
dat$y[dat$z > 0] <- 1

b <- bayesx(y ~ sx(x), family = "binomialprobit", data = dat)
summary(b)
plot(b)

## estimate varying coefficient models
set.seed(333)
n <- 1000
dat <- data.frame(x = runif(n, -3, 3), id = factor(rep(1:4, n/4)))

## response
dat$y <- with(dat, 1.5 + sin(x) * c(-1, 0.2, 1, 5)[id] + rnorm(n, sd = 0.6))

## estimate model
b <- bayesx(y ~ sx(x, by = id, center = TRUE),
  method = "REML", data = dat)
summary(b)
plot(b, resid = TRUE, cex.resid = 0.1)

## End(Not run)

```

 bayesx.construct

Construct BayesX Model Term Objects

Description

The function `bayesx.construct` is used to provide a flexible framework to implement new model term objects in [bayesx](#) within the **BayesX** syntax.

Usage

```
bayesx.construct(object, dir, prg, data)
```

Arguments

object	is a smooth, shrinkage or random specification object in a STAR formula, generated by the R2BayesX model term constructor functions <code>sx</code> (or using the constructor functions <code>s</code> and <code>te</code> of the mgcv package). Objects generated by these functions have class <code>"xx.smooth.spec"</code> where <code>"xx"</code> is determined by the <code>"bs"</code> argument of <code>sx</code> (and <code>s</code>).
--------	--

dir	character, a directory where possible data should be stored, e.g. in <code>bayesx</code> models, if <code>bs = "gk"</code> , <code>bs = "gs"</code> or <code>bs = "mrf"</code> is chosen, the corresponding map will be written as a "bnd" or "gra" file (see <code>read.bnd</code> and <code>read.gra</code>) to this directory, so BayesX can use this spatial object for estimation.
prg	if additional data handling must be applied, e.g. storing maps ("bnd") objects in the directory specified in <code>dir</code> , <code>write.bayesx.input</code> needs to write the extra commands in a program file provided with argument <code>prg</code> , i.e. this may all be handled within a <code>bayesx.construct</code> constructor function.
data	if additional data is needed to setup the BayesX term it is found here.

Details

The main idea of these constructor functions is to provide a flexible framework to implement new model term objects in the **BayesX** syntax within `bayesx`, i.e. for any smooth or random term in **R2BayesX** a constructor function like `bayesx.construct.ps.smooth.construct` may be provided to translate R specific syntax into **BayesX** readable commands. During processing with `write.bayesx.input` each model term is constructed with `bayesx.construct` after another, wrapped into a full formula, which may then be sent to the **BayesX** binary with function `run.bayesx`.

At the moment the following model terms are implemented:

- "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", "geokriling": Geokriling 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 $\log(\lambda(\text{time}))$.
- "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 $\gamma_j, j = 1, \dots, q, q \geq 1$ of the linear effects x_1, \dots, 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.

See function [sx](#) for a description of the main **R2BayesX** model term constructor functions.

Value

The model term syntax used within **BayesX** as a character string.

WARNINGS

If new `bayesx.construct` functions are implemented in future work, there may occur problems with reading the corresponding **BayesX** output files with `read.bayesx.output`, e.g., if the new objects do not have the structure as implemented with `bs = "ps"` etc., i.e. function `read.bayesx.output` must also be adapted in such cases.

Note

Using `sx` additional controlling arguments may be supplied within the dot dot dot "... " argument. Please see the help site for function `bayesx.term.options` for a detailed description of possible optional parameters.

Within the `xt` argument in function `s`, additional **BayesX** specific parameters may be also supplied, see the examples below.

Author(s)

Nikolaus Umlauf, Thomas Kneib, Stefan Lang, Achim Zeileis.

See Also

[sx](#), [bayesx.term.options](#), [s](#), [formula.gam](#), [read.bnd](#), [read.gra](#).

Examples

```
bayesx.construct(sx(x1, bs = "ps"))
bayesx.construct(sx(x1, x2, bs = "te"))

## now create BayesX syntax for smooth terms
## using mgcv constructor functions
bayesx.construct(s(x1, bs = "ps"))

## for tensor product P-splines,
```

```
bayesx.construct(s(x1, x2, bs = "te"))

## increase number of knots
## for a P-spline
bayesx.construct(sx(x1, bs = "ps", nrknots = 40))

## now with degree 2 and
## penalty order 1
bayesx.construct(sx(x1, bs = "ps", knots = 40, degree = 2, order = 1))
bayesx.construct(s(x1, bs = "ps", k = 41, m = c(0, 1)))

## random walks
bayesx.construct(sx(x1, bs = "rw1"))
bayesx.construct(sx(x1, bs = "rw2"))

## shrinkage priors
bayesx.construct(sx(x1, bs = "lasso"))
bayesx.construct(sx(x1, bs = "ridge"))
bayesx.construct(sx(x1, bs = "nigmix"))

## for cox models, baseline
bayesx.construct(sx(time, bs = "bl"))

## kriging
bayesx.construct(sx(x, z, bs = "kr"))

## seasonal
bayesx.construct(sx(x, bs = "season"))

## factors
bayesx.construct(sx(id, bs = "factor"))

## now with some geographical information
## note: maps must be either supplied in
## 'bnd' or 'gra' format, also see function
## read.bnd() or read.gra()
data("MunichBnd")
bayesx.construct(sx(id, bs = "mrf", map = MunichBnd))

## same with
bayesx.construct(s(id, bs = "mrf", xt = list(map = MunichBnd)))

bayesx.construct(sx(id, bs = "gk", map = MunichBnd))
bayesx.construct(sx(id, bs = "gs", map = MunichBnd))

## also vary number of knots
bayesx.construct(sx(id, bs = "gs", knots = 10, map = MunichBnd))
bayesx.construct(s(id, bs = "gs", k = 12, m = c(1, 1), xt = list(map = MunichBnd)))

## random effects
bayesx.construct(sx(id, bs = "re"))
bayesx.construct(sx(id, bs = "re", by = x1))
bayesx.construct(sx(id, bs = "re", by = x1, xt = list(nofixed=TRUE)))
```

```
## generic
## specifies some model term
## and sets all additional arguments
## within argument xt
## only for experimental use
bayesx.construct(sx(x, bs = "generic", dosomething = TRUE, a = 1, b = 2))
```

bayesx.control *Control Parameters for BayesX*

Description

Various parameters that control fitting of regression models using [bayesx](#).

Usage

```
bayesx.control(model.name = "bayesx.estim",
  family = "gaussian", method = "MCMC", verbose = FALSE,
  dir.rm = TRUE, outfile = NULL, replace = FALSE, iterations = 12000L,
  burnin = 2000L, maxint = NULL, step = 10L, predict = TRUE,
  seed = NULL, hyp.prior = NULL, distopt = NULL, reference = NULL,
  zipdistopt = NULL, begin = NULL, level = NULL, eps = 1e-05,
  lowerlim = 0.001, maxit = 400L, maxchange = 1e+06, leftint = NULL,
  lefttrunc = NULL, state = NULL, algorithm = NULL, criterion = NULL,
  proportion = NULL, startmodel = NULL, trace = NULL,
  steps = NULL, CI = NULL, bootstrapsamples = NULL, ...)
```

Arguments

model.name	character, specify a base name model output files are named with in outfile.
family	character, specify the distribution used for the model, options for all methods, "MCMC", "REML" and "STEP" are: "binomial", "binomialprobit", "gamma", "gaussian", "multinomial", "poisson". For "MCMC" and "REML" only: "cox", "cumprobit" and "multistate". For "REML" only use: "binomialcomploglog", "cumlogit", "multinomialcatsp", "multinomialprobit", "seqlogit", "seqprobit".
method	character, which method should be used for estimation, options are "MCMC", "HMC" (hierarchical MCMC), "REML" and "STEP".
verbose	logical, should output be printed to the R console during runtime of bayesx .
dir.rm	logical, should the the output files and directory removed after estimation?
outfile	character, specify a directory where bayesx should store all output files, all output files will be named with model.name as the base name.
replace	if set to TRUE, the files in the output directory specified in argument outfile will be replaced.
iterations	integer, sets the number of iterations for the sampler.

burnin	integer, sets the burn-in period of the sampler.
maxint	integer, if first or second order random walk priors are specified, in some cases the data will be slightly grouped: The range between the minimal and maximal observed covariate values will be divided into (small) intervals, and for each interval one parameter will be estimated. The grouping has almost no effect on estimation results as long as the number of intervals is large enough. With the maxint option the amount of grouping can be determined by the user. integer is the maximum number of intervals allowed. for equidistant data, the default maxint = 150 for example, means that no grouping will be done as long as the number of different observations is equal to or below 150. for non equidistant data some grouping may be done even if the number of different observations is below 150.
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. The aim of thinning is to reach a considerable reduction of disk storing and autocorrelations between sampled parameters.
predict	logical, option predict may be specified to compute samples of the deviance D, the effective number of parameters pD and the deviance information criterion DIC of the model. In addition, if predict = FALSE, 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. Hence, this option is useful when dealing with large data sets, that might cause memory problems if predict is set to TRUE.
seed	integer, set the seed of the random number generator in BayesX , usually set using function set.seed .
hyp.prior	numeric, defines the value of the hyper-parameters a and b for the inverse gamma prior of the overall variance parameter σ^2 , if the response distribution is Gaussian. numeric, must be a positive real valued number. The default is hyp.prior = c(1, 0.005).
distopt	character, defines the implemented formulation for the negative binomial model if the response distribution is negative binomial. The two possibilities are to work with a negative binomial likelihood (distopt = "nb") or to work with the Poisson likelihood and the multiplicative random effects (distopt = "poga").
reference	character, option reference is meaningful only if either family = "multinomial" or family = "multinomialprobit" is specified as the response distribution. In this case reference defines the reference category to be chosen. Suppose, for instance, that the response is three categorical with categories 1, 2 and 3. Then reference = 2 defines the value 2 to be the reference category.
zipdistopt	character, defines the zero inflated distribution for the regression analysis. The two possibilities are to work with a zero inflated Poisson distribution (zipdistopt = "zip") or to work with the zero inflated negative binomial likelihood (zipdistopt = "zinb").
begin	character, option begin is meaningful only if family = "cox" is specified as the response distribution. In this case begin specifies the variable that records when

the observation became at risk. This option can be used to handle left truncation and time-varying covariates. If `begin` is not specified, all observations are assumed to have become at risk at time 0.

<code>level</code>	integer, besides the posterior means and medians, BayesX provides point-wise posterior credible intervals for every effect in the model. In a Bayesian approach based on MCMC simulation techniques credible intervals are estimated by computing the respective quantiles of the sampled effects. By default, BayesX computes (point-wise) credible intervals for nominal levels of 80% and 95%. The option <code>level[1]</code> allows to redefine one of the nominal levels (95%). Adding, for instance, <code>level[1] = 99</code> to the options list computes credible intervals for a nominal level of 99% rather than 95%. Similar to argument <code>level[1]</code> the option <code>level[2]</code> allows to redefine one of the nominal levels (80%). Adding, for instance, <code>level[2] = 70</code> to the options list computes credible intervals for a nominal level of 70% rather than 80%.
<code>eps</code>	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 <code>eps</code> , the estimation process is assumed to have converged.
<code>lowerlim</code>	numeric, since small variances are close to the boundary of their parameter space, the usual fisher-scoring algorithm for their determination has to be modified. If the fraction of the penalized part of an effect relative to the total effect is less than <code>lowerlim</code> , the estimation of the corresponding variance is stopped and the estimator is defined to be the current value of the variance (see section 6.2 of the BayesX methodology manual for details).
<code>maxit</code>	integer, defines the maximum number of iterations to be used in estimation. Since the estimation process will not necessarily converge, it may be useful to define an upper bound for the number of iterations. Note, that BayesX returns results based on the current values of all parameters even if no convergence could be achieved within <code>maxit</code> iterations, but a warning message will be printed in the output window.
<code>maxchange</code>	numeric, defines the maximum value that is allowed for relative changes in parameters in one iteration to prevent the program from crashing because of numerical problems. Note, that BayesX produces results based on the current values of all parameters even if the estimation procedure is stopped due to numerical problems, but an error message will be printed in the output window.
<code>leftint</code>	character, gives the name of the variable that contains the lower (left) boundary T_{lo} of the interval $[T_{lo}, T_{up}]$ for an interval censored observation. for right censored or uncensored observations we have to specify $T_{lo} = T_{up}$. If <code>leftint</code> is missing, all observations are assumed to be right censored or uncensored, depending on the corresponding value of the censoring indicator.
<code>lefttrunc</code>	character, option <code>lefttrunc</code> specifies the name of the variable containing the left truncation time T_{tr} . For observations that are not truncated, we have to specify $T_{tr} = 0$. If <code>lefttrunc</code> is missing, all observations are assumed to be not truncated. for multi-state models variable <code>lefttrunc</code> specifies the left endpoint of the corresponding time interval.
<code>state</code>	character, for multi-state models, <code>state</code> specifies the current state variable of the process.

algorithm	character, specifies the selection algorithm. Possible values are "cdescent1" (adaptive algorithms in the methodology manual, see subsection 6.3), "cdescent2" (adaptive algorithms 1 and 2 with backfitting, see remarks 1 and 2 of section 3 in Belitz 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 routine of S-plus, see Chambers and Hastie, 1992). This option will rarely be specified by the user.
criterion	character, specifies the goodness of fit criterion. If criterion = "MSEP" is specified the data are randomly divided into a test- and validation data set. The test data set is used to estimate the models and the validation data set is used to estimate the mean squared prediction error (MSEP) which serves as the goodness of fit criterion to compare different models. The proportion of data used for the test and validation sample can be specified using option proportion, see below. The default is to use 75% of the data for the training sample.
proportion	numeric, this option may be used in combination with option criterion = "MSEP", see above. In this case the data are randomly divided into a training and a validation sample. proportion defines the fraction (between 0 and 1) of the original data used as training sample.
startmodel	character, defines the start model for variable selection. Options are "linear", "empty", "full" and "userdefined".
trace	character, specifies how detailed the output in the output window will be. Options are "trace_on", "trace_half" and "trace_off".
steps	integer, defines the maximum number of iterations. If the selection process has not converged after steps iterations the algorithm terminates and a warning is raised. Setting steps = 0 allows the user to estimate a certain model without any model choice. This option will rarely be specified by the user.
CI	character, compute confidence intervals for linear and nonlinear terms. Option CI allows to compute confidence intervals. Options are CI = "none", confidence intervals conditional on the selected model CI = "MCMCselect" and unconditional confidence intervals where model uncertainty is taken into account CI = "MCMCbootstrap". Both alternatives are computer intensive. Conditional confidence intervals take much less computing time than unconditional intervals. The advantage of unconditional confidence intervals is that sampling distributions for the degrees of freedom or smoothing parameters are obtained.
bootstrapsamples	integer, defines the number of bootstrap samples used for "CI = MCMCbootstrap".
...	not used

Value

A list with the arguments specified is returned.

Author(s)

Nikolaus Umlauf, Thomas Kneib, Stefan Lang, Achim Zeileis.

References

For methodological and reference details see the **BayesX** manuals available at: <https://www.uni-goettingen.de/de/bayesx/550513.html>.

Belitz C, Lang S (2008). Simultaneous selection of variables and smoothing parameters in structured additive regression models. *Computational Statistics & Data Analysis*, **53**, 61–81.

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

Umlauf N, Adler D, Kneib T, Lang S, Zeileis A (2015). Structured Additive Regression Models: An R Interface to BayesX. *Journal of Statistical Software*, **63**(21), 1–46. <https://www.jstatsoft.org/v63/i21/>

See Also

[bayesx](#).

Examples

```
bayesx.control()

## Not run:
set.seed(111)
n <- 500
## regressors
dat <- data.frame(x = runif(n, -3, 3))
## response
dat$y <- with(dat, 10 + sin(x) + rnorm(n, sd = 0.6))

## estimate models with
## bayesx MCMC and REML
b1 <- bayesx(y ~ sx(x), method = "MCMC", data = dat)
b2 <- bayesx(y ~ sx(x), method = "REML", data = dat)

## compare reported output
summary(b1)
summary(b2)

## End(Not run)
```

bayesx.term.options *Show BayesX Term Options*

Description

BayesX model terms specified using functions `sx` may have additional optional control arguments. Therefore function `bayesx.term.options` displays the possible additional controlling parameters for a particular model term.

Usage

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

Arguments

bs	character, the term specification for which controlling parameters should be shown.
method	character, for which method should additional arguments be shown, options are "MCMC", "REML" and "STEP".

Details

At the moment the following model terms are implemented, for which additional controlling parameters may be specified:

- "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 $\log(\lambda(\text{time}))$.
- "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 , $j = 1, \dots, q$, $q \geq 1$ of the linear effects x_1, \dots, 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.

Author(s)

Nikolaus Umlauf, Thomas Kneib, Stefan Lang, Achim Zeileis.

Examples

```
## show arguments for P-splines
bayesx.term.options(bs = "ps")
bayesx.term.options(bs = "ps", method = "REML")

## Markov random fields
bayesx.term.options(bs = "mrf")
```

bayesx_logfile	<i>BayesX Log-Files</i>
----------------	-------------------------

Description

Function to show the internal **BayesX** log-files.

Usage

```
bayesx_logfile(x, model = 1L)
```

Arguments

x	a fitted "bayesx" object.
model	integer, for which model the log-file should be printed, i.e. if x contains more than one estimated model.

Value

The log-file returned from **BayesX**.

Author(s)

Nikolaus Umlauf, Thomas Kneib, Stefan Lang, Achim Zeileis.

See Also

[bayesx](#).

Examples

```
## Not run:
## generate some data
set.seed(111)
n <- 500

## regressor
dat <- data.frame(x = runif(n, -3, 3))

## response
dat$y <- with(dat, 1.5 + sin(x) + rnorm(n, sd = 0.6))

## estimate model
b <- bayesx(y ~ sx(x), data = dat)

## now see the log-file
bayesx_logfile(b)

## End(Not run)
```

bayesx_prgfile

BayesX Program-Files

Description

Function to show the internal **BayesX** program-files.

Usage

```
bayesx_prgfile(x, model = 1L)
```

Arguments

x	a fitted "bayesx" object.
model	integer, for which model the program-file should be printed, i.e. if x contains more that one estimated model.

Value

The program file used for estimation with **BayesX**.

Author(s)

Nikolaus Umlauf, Thomas Kneib, Stefan Lang, Achim Zeileis.

See Also

[bayesx](#).

Examples

```
## Not run:
## generate some data
set.seed(111)
n <- 500

## regressor
dat <- data.frame(x = runif(n, -3, 3))

## response
dat$y <- with(dat, 1.5 + sin(x) + rnorm(n, sd = 0.6))

## estimate model
b <- bayesx(y ~ sx(x), data = dat)

## now see the prg-file
bayesx_prgfile(b)

## End(Not run)
```

bayesx_runtime

BayesX Program-Runtimes

Description

Function to extract running times of the **BayesX** binary.

Usage

```
bayesx_runtime(x, model = 1L)
```

Arguments

x	a fitted "bayesx" object.
model	integer, for which model the program-file should be printed, i.e. if x contains more that one estimated model.

Value

The runtime of the **BayesX** binary returned form [system.time](#).

Author(s)

Nikolaus Umlauf, Thomas Kneib, Stefan Lang, Achim Zeileis.

See Also

[bayesx](#).

Examples

```
## Not run:
## generate some data
set.seed(111)
n <- 500

## regressor
dat <- data.frame(x = runif(n, -3, 3))

## response
dat$y <- with(dat, 1.5 + sin(x) + rnorm(n, sd = 0.6))

## estimate model
b <- bayesx(y ~ sx(x), data = dat)

## now see the prg-file
bayesx_runtime(b)

## End(Not run)
```

BeechBnd

Beech Location Map

Description

This database produces a location map of beeches around Rothenbuch, Germany.

Usage

```
data("BeechBnd")
```

Format

A [list](#) of class "bnd" containing 83 polygon matrices with x-coordinates in the first and y-coordinates in the second column each.

Source

<https://www.uni-goettingen.de/de/bayesx/550513.html>.

See Also

[plotmap](#), [read.bnd](#), [write.bnd](#)

Examples

```
## load BeechBnd and plot it
data("BeechBnd")
plotmap(BeechBnd)
```

BeechGra

Beech Neighborhood Information

Description

This database produces a graph file including neighborhood information of the beech trees around Rothenbuch, Germany.

Usage

```
data("BeechGra")
```

Format

An adjacency matrix that represents the neighborhood structure defined in the graph file.

Source

<https://www.uni-goettingen.de/de/bayesx/550513.html>.

See Also

[read.gra](#), [bnd2gra](#)

Examples

```
## load BeechGra adjacency matrix
data("BeechGra")
print(BeechGra)
```

bnd2gra

Convert Boundary Format to Graph Format

Description

Converts a map in boundary format to a map in graph format.

Usage

```
bnd2gra(map, npoints = 2)
```

Arguments

`map` map in boundary format that should be converted.
`npoints` integer. How many points must be shared by two polygons to be a neighbor?

Value

Returns an adjacency matrix that represents the neighborhood structure of the map object in graph format.

Author(s)

Felix Heinzl, Thomas Kneib.

References

BayesX Reference Manual. Available at <https://www.uni-goettingen.de/de/bayesx/550513.html>.

See Also

[read.bnd](#), [read.gra](#), [write.bnd](#), [write.gra](#).

Examples

```
data("FantasyBnd")
plotmap(FantasyBnd, names = TRUE)
adjmat <- bnd2gra(FantasyBnd)
adjmat
```

colorlegend

Plot a Color Legend

Description

Function to generate a color legend, the legend may be added to an existing plot or drawn in a separate plotting window.

Usage

```
colorlegend (color = NULL, ncol = NULL, x = NULL,
  breaks = NULL, pos = "center", shift = 0.02, side.legend = 1L,
  side.ticks = 1L, range = NULL, lrange = NULL,
  width = 0.4, height = 0.06, scale = TRUE, xlim = NULL,
  ylim = NULL, plot = NULL, full = FALSE, add = FALSE,
  col.border = "black", lty.border = 1L, lwd.border = 1L,
  ticks = TRUE, at = NULL, col.ticks = "black", lwd.ticks = 1L,
  lty.ticks = 1L, length.ticks = 0.3, labels = NULL,
  distance.labels = 0.8, col.labels = "black", cex.labels = 1L,
  digits = 2L, swap = FALSE, symmetric = TRUE, xpd = NULL,
  title = NULL, side.title = 2, shift.title = c(0, 0), ...)
```

Arguments

color	character, integer. The colors for the legend, may also be a function, e.g. <code>colors = heat.colors</code> .
ncol	integer, the number of different colors that should be generated if color is a function.
x	numeric, values for which the color legend should be drawn.
breaks	numeric, a set of breakpoints for the colors: must give one more breakpoint than ncol.
pos	character, numeric. The position of the legend. Either a numeric vector, e.g. <code>pos = c(0.1, 0.2)</code> 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", "bottomright", "left", "right", "top", "bottom" and "center".
shift	numeric, if argument pos is a character, shift determines the distance of the legend from the plotting box.
side.legend	integer, if set to 2 the legend will be flipped by 90 degrees.
side.ticks	integer, if set to 2, the ticks and labels will be on the opposite site of the legend.
range	numeric, specifies a range for x values for which the legend should be drawn.
lrange	numeric, specifies the range of legend.
width	numeric, the width of the legend, if <code>scale = TRUE</code> the width is proportional to the x-limits of the plotting window.
height	numeric, the height of the legend, if <code>scale = TRUE</code> the height is proportional to the y-limits of the plotting window.
scale	logical, if set to TRUE, the width and height of the legend will be calculated proportional to the x- and y-limits of the plotting window.
xlim	numeric, the x-limits of the plotting window the legend should be added for, numeric vector, e.g., returned from function range .
ylim	numeric, the y-limits of the plotting window the legend should be added for, numeric vector, e.g., returned from function range .
plot	logical, if set to TRUE, the legend will be drawn in a separate plotting window.
full	logical, if set to TRUE, the legend will be drawn using the full window range.
add	logical, if set to TRUE, the legend will be added to an existing plot.
col.border	the color of the surrounding border line of the legend.
lty.border	the line type of the surrounding border line of the legend.
lwd.border	the line width of the surrounding border line of the legend.
ticks	logical, if set to TRUE, ticks will be added to the legend.
at	numeric, specifies at which locations ticks and labels should be added.
col.ticks	the colors of the ticks.
lwd.ticks	the line width of the ticks.
lty.ticks	the line type of the ticks.

<code>length.ticks</code>	numeric, the length of the ticks as percentage of the height or width of the colorlegend.
<code>labels</code>	character, specifies labels that should be added to the ticks.
<code>distance.labels</code>	numeric, the distance of the labels to the ticks, proportional to the length of the ticks.
<code>col.labels</code>	the colors of the labels.
<code>cex.labels</code>	text size of the labels.
<code>digits</code>	integer, the decimal places if labels are numerical.
<code>swap</code>	logical, if set to TRUE colors will be represented in reverse order.
<code>symmetric</code>	logical, if set to TRUE, a symmetric legend will be drawn corresponding to the $\pm \max(\text{abs}(x))$ value.
<code>xpd</code>	sets the <code>xpd</code> parameter in function <code>par</code> .
<code>title</code>	character, a title for the legend.
<code>side.title</code>	integer, 1 or 2. Specifies where the legend is placed, either on top if <code>side.title = 1</code> or at the bottom if <code>side.title = 2</code> .
<code>shift.title</code>	numeric vector of length 2. Specifies a possible shift of the title in either x- or y-direction.
<code>...</code>	other graphical parameters to be passed to function <code>text</code> .

Value

A named list with the colors generated, the breaks and the function map, which may be used for mapping of x values to the colors specified in argument `colors`, please see the examples below.

Author(s)

Nikolaus Umlauf, Thomas Kneib, Stefan Lang, Achim Zeileis.

Examples

```
## play with colorlegend
colorlegend()
colorlegend(side.legend = 2)
colorlegend(side.legend = 2, side.ticks = 2)
colorlegend(height = 2)
colorlegend(width = 1, height = 0.8, scale = FALSE,
  pos = c(0, 0.2), length.ticks = 0.5)
colorlegend(color = heat.colors, ncol = 9)
colorlegend(color = heat.colors, ncol = 9, swap = TRUE)
colorlegend(pos = "bottomleft")
colorlegend(pos = "topleft")
colorlegend(pos = "topright")
colorlegend(pos = "bottomright")

## take x values for the color legend
```

```

x <- runif(100, -2, 2)
colorlegend(color = diverge_hcl, x = x)
colorlegend(color = diverge_hcl, x = x, at = c(-1.5, 0, 1.5))
colorlegend(color = diverge_hcl, x = x, at = c(-1.5, 0, 1.5),
  labels = c("low", "middle", "high"))
colorlegend(color = rainbow_hcl, x = x, at = c(-1.5, 0, 1.5),
  labels = c("low", "middle", "high"), length.ticks = 1.5)
colorlegend(color = heat_hcl, x = x, at = c(-1.5, 0, 1.5),
  labels = c("low", "middle", "high"), length.ticks = 1.5,
  lwd.border = 2, lwd.ticks = 2, cex.labels = 1.5, font = 2)
colorlegend(color = topo.colors, x = x, at = c(-1.5, 0, 1.5),
  labels = c("low", "middle", "high"), length.ticks = 1.5,
  lwd.border = 2, lwd.ticks = 2, cex.labels = 1.5, font = 2,
  col.border = "green3", col.ticks = c(2, 5, 2),
  col.labels = c(6, 4, 3))
colorlegend(color = diverge_hsv, x = x, at = c(-1.5, 0, 1.5),
  labels = c("low", "middle", "high"), length.ticks = 1.5,
  lwd.border = 2, lwd.ticks = 2, cex.labels = 1.5, font = 2,
  col.border = "green3", col.ticks = c(2, 5, 2),
  col.labels = c(6, 4, 3), lty.border = 2, lty.ticks = c(2, 3, 2))
colorlegend(color = diverge_hsv, x = x, at = c(-1.5, 0, 1.5),
  labels = c("low", "middle", "high"), length.ticks = 1.5,
  lwd.border = 2, lwd.ticks = 2, cex.labels = 1.5, font = 2,
  col.border = "green3", col.ticks = c(2, 5, 2),
  col.labels = c(6, 4, 3), lty.border = 2, lty.ticks = c(2, 3, 2),
  ncol = 3)
colorlegend(color = c("red", "white", "red"), x = x, at = c(-1.5, 0, 1.5),
  labels = c("low", "middle", "high"), length.ticks = 1.5,
  lwd.border = 2, lwd.ticks = 2, cex.labels = 1.5, font = 2,
  col.border = "green3", col.ticks = c(2, 5, 2),
  col.labels = c(6, 4, 3), lty.border = 2, lty.ticks = c(2, 3, 2),
  ncol = 3, breaks = c(-2, -1, 1, 2))
colorlegend(color = diverge_hcl, x = x, range = c(-3, 3))
colorlegend(color = diverge_hcl, x = x, range = c(-3, 3), lrange = c(-6, 6))

## combine plot with color legend
n <- 100
x <- y <- seq(-3, 3, length.out = n)
z <- outer(sin(x), cos(x))
pal <- colorlegend(color = diverge_hcl, x = z, plot = FALSE)
op <- par(no.readonly = TRUE)
par(mar = c(4.1, 4.1, 1.1, 1.1))
layout(matrix(c(1, 2), nrow = 1), widths = c(1, 0.3))
image(x = x, y = y, z = z, col = pal$colors, breaks = pal$breaks)
par(mar = c(4.1, 0.1, 1.1, 3.1))
colorlegend(color = diverge_hcl, x = z, plot = TRUE, full = TRUE,
  side.legend = 2, side.ticks = 2)
par(op)

## another example with different plot
n <- 50

```

```
x <- sin(seq(-3, 3, length.out = n))
pal <- colorlegend(color = diverge_hcl, x = x, plot = FALSE)
op <- par(no.readonly = TRUE)
par(mar = c(7.1, 4.1, 1.1, 1.1))
barplot(x, border = "transparent", col = pal$map(x))
colorlegend(color = diverge_hcl, x = x, plot = FALSE, add = TRUE,
  xlim = c(0, 60), ylim = c(-1, 1), pos = c(0, -0.15), xpd = TRUE,
  scale = FALSE, width = 60, height = 0.15,
  at = seq(min(x), max(x), length.out = 9))
par(op)
```

cprob

Extract Contour Probabilities

Description

Function to extract estimated contour probabilities of a particular effect estimated with P-splines using Markov chain Monte Carlo (MCMC) estimation techniques. Note that, the contour probability option must be specified within function `sx`, see the example.

Usage

```
cprob(object, model = NULL, term = NULL, ...)
```

Arguments

object	an object of class "bayesx".
model	for which model the contour probabilities should be provided, either an integer or a character, e.g. <code>model = "mcmc.model"</code> .
term	if not NULL, the function will search for the term contour probabilities should be extracted for, either an integer or a character, eg <code>term = "s(x)"</code> .
...	not used.

Author(s)

Nikolaus Umlauf, Thomas Kneib, Stefan Lang, Achim Zeileis.

References

Brezger, A., Lang, S. (2008): Simultaneous probability statements for Bayesian P-splines. *Statistical Modeling*, **8**, 141–186.

See Also

[bayesx](#).

Examples

```
## Not run:
## generate some data
set.seed(111)
n <- 500

## regressor
dat <- data.frame(x = runif(n, -3, 3))

## response
dat$y <- with(dat, 1.5 + sin(x) + rnorm(n, sd = 0.6))

## estimate model
## need to set the contourprob option,
## otherwise BayesX will not calculate probabilities
## see also the reference manual of BayesX available
## at www.BayesX.org
b <- bayesx(y ~ sx(x), bs = "ps", contourprob = 4), data = dat)

## extract contour probabilities
cprob(b, term = "sx(x)")

## End(Not run)
```

delete.neighbor

Delete Neighborhood Relations

Description

Adds the neighborhood relationship between two given regions from a map object in graph format.

Usage

```
delete.neighbor(map, region1, region2)
```

Arguments

```
map          map object in graph format that should be modified.
region1, region2  names of the regions that should no longer be regarded as neighbors.
```

Value

Returns an adjacency matrix that represents the neighborhood structure of map minus the deleted neighborhood relation in graph format.

Author(s)

Felix Heinzl, Thomas Kneib.

See Also

[get.neighbor](#), [add.neighbor](#), [read.gra](#), [write.gra](#), [bnd2gra](#).

Examples

```
## read the graph file
file <- file.path(find.package("R2BayesX"), "examples", "Germany.gra")
germany <- read.gra(file)

## delete some neighbors
get.neighbor(germany, c("7339"))
germany <- delete.neighbor(germany, "7339", "7141")
get.neighbor(germany, c("7339"))
```

DIC

Deviance Information Criterion

Description

Generic function returning the deviance information criteriom of a fitted model object.

Usage

```
DIC(object, ...)
```

```
## S3 method for class 'bayesx'
DIC(object, ...)
```

Arguments

<code>object</code>	an object of class "bayesx".
<code>...</code>	specify for which model the criterion should be returned, e.g. type <code>model = 1</code> to obtain the value for the first model. Only meaningful if object contains of more than one model.

Author(s)

Nikolaus Umlauf, Thomas Kneib, Stefan Lang, Achim Zeileis.

See Also

[bayesx](#).

Examples

```
## Not run:
## generate some data
set.seed(121)
n <- 200

## regressors
dat <- data.frame(x = runif(n, -3, 3))

## generate response
dat$y <- with(dat, 1.5 + sin(x) + rnorm(n, sd = 0.6))

## estimate model
b <- bayesx(y ~ sx(x), data = dat, method = "MCMC")

## extract DIC
DIC(b)

## End(Not run)
```

FantasyBnd

Fantasy Map

Description

This database produces a fantasy map of 10 regions.

Usage

```
data("FantasyBnd")
```

Format

A [list](#) of class "bnd" containing 10 polygon matrices with x-coordinates in the first and y-coordinates in the second column each.

See Also

[plotmap](#), [read.bnd](#), [write.bnd](#)

Examples

```
## load FantasyBnd and plot it
data("FantasyBnd")
plotmap(FantasyBnd)
```

`fitted.bayesx`*Extract BayesX Fitted Values and Residuals*

Description

Extractor functions to the fitted values/model residuals of the estimated model with `bayesx` and fitted model term partial effects/residuals.

Usage

```
## S3 method for class 'bayesx'  
fitted(object, model = NULL, term = NULL, ...)
```

```
## S3 method for class 'bayesx'  
residuals(object, model = NULL, term = NULL, ...)
```

Arguments

<code>object</code>	an object of class "bayesx".
<code>model</code>	for which model the fitted values/residuals should be provided, either an integer or a character, e.g. <code>model = "mcmc.model"</code> .
<code>term</code>	if not NULL, the function will search for the term fitted values/residuals specified here, either an integer or a character, eg <code>term = "sx(x)"</code> .
<code>...</code>	not used.

Value

For `fitted.bayesx`, either the fitted linear predictor and mean or if e.g. `term = "sx(x)"`, an object with class "`xx.bayesx`", where "`xx`" is depending of the type of the term. In principle the returned term object is simply a `data.frame` containing the covariate(s) and its effects, depending on the estimation method, e.g. for MCMC estimated models, mean/median fitted values and other quantities are returned. Several additional informations on the term are provided in the `attributes` of the object. For all types of terms plotting functions are provided, see function `plot.bayesx`.

Using `residuals.bayesx` will either return the mean model residuals or the mean partial residuals of a term specified in argument `term`.

Author(s)

Nikolaus Umlauf, Thomas Kneib, Stefan Lang, Achim Zeileis.

See Also

[read.bayesx.output](#).

Examples

```
## Not run: ## generate some data
set.seed(121)
n <- 500

## regressors
dat <- data.frame(x = runif(n, -3, 3), z = runif(n, 0, 1),
  w = runif(n, 0, 3))

## generate response
dat$y <- with(dat, 1.5 + sin(x) + z -3 * w + rnorm(n, sd = 0.6))

## estimate model
b1 <- bayesx(y ~ sx(x) + z + w, data = dat)

## extract fitted values
fit <- fitted(b1)
hist(fit, freq = FALSE)

## now extract 1st model term
## and plot it
fx <- fitted(b1, term = "sx(x)")
plot(fx)

## extract model residuals
hist(residuals(b1))

## extract partial residuals for sx(x)
pres <- residuals(b1, term = "sx(x)")
plot(fx, ylim = range(pres[, 2]))
points(pres)

## now another example with
## use of read.bayesx.output
## load example data from
## package R2BayesX
dir <- file.path(find.package("R2BayesX"), "examples", "ex01")
b2 <- read.bayesx.output(dir)

## extract fitted values
hist(fitted(b2))

## extract model term of x
## and plot it
fx <- fitted(b2, term = "sx(x)")
plot(fx)

## have a look at the attributes
names(attributes(fx))

## extract the sampling path of the variance
```

```

spv <- attr(fx, "variance.sample")
plot(spv, type = "l")

## combine model objects
b <- c(b1, b2)

## extract fitted terms for second model
fit <- fitted(b, model = 2, term = 1:2)
names(fit)
plot(fit["sx(id)"])

## End(Not run)

```

ForestHealth

Forest Health Data

Description

The data set consists of 16 variables with 1796 observations on forest health to identify potential factors influencing the health status of trees and therefore the vital status of the forest. In addition to covariates characterizing a tree and its stand, the exact locations of the trees are known. The interest is on detecting temporal and spatial trends while accounting for further covariate effects in a flexible manner.

Usage

```
data("ForestHealth")
```

Format

A data frame containing 1793 observations on 16 variables.

id: tree location identification number.

year: year of census.

defoliation: percentage of tree defoliation in three ordinal categories, 'defoliation < 12.5%', '12.5% <= defoliation < 50%' and 'defoliation >= 50%'

x: x-coordinate of the tree location.

y: y-coordinate of the tree location.

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, categorical coded.

stand: stand type with categories ‘deciduous’ and ‘mixed’.

fertilized: fertilization applied with categories ‘yes’ and ‘no’.

Source

<https://www.uni-goettingen.de/de/bayesx/550513.html>.

References

Kneib, T. & Fahrmeir, L. (2010): A Space-Time Study on Forest Health. In: Chandler, R. E. & Scott, M. (eds.): Statistical Methods for Trend Detection and Analysis in the Environmental Sciences, Wiley.

Glottlein A, Pruscha H (1996). Der Einfluss von Bestandskenngrößen, Topographie, Standort und Witterung auf die Entwicklung des Kronenzustandes im Bereich des Forstamtes Rothenbuch. *Forstwissens. Zent.*, **114**, 146–162.

See Also

[bayesx](#)

Examples

```
## Not run:
## load zambia data and map
data("ForestHealth")
data("BeechBnd")

fm <- bayesx(defoliation ~ stand + fertilized +
  humus + moisture + alkali + ph + soil +
  sx(age) + sx(inclination) + sx(canopy) +
  sx(year) + sx(elevation),
  family = "cumlogit", method = "REML", data = ForestHealth)

summary(fm)
plot(fm, term = c("sx(age)", "sx(inclination)",
  "sx(canopy)", "sx(year)", "sx(elevation)"))

## End(Not run)
```

Description

This is an artificial data set mainly used to test the **R2BayesX** interfacing functions. The data includes three different types of response variables. One numeric, one binomial and a categorical response with 4 different levels. In addition, several numeric and factor covariates are provided. The data set is constructed such that the observations are based upon different locations (pixels in 'longitude' and 'latitude' coordinates) obtained from a regular grid.

Usage

```
data("GAMart")
```

Format

A data frame containing 500 observations on 12 variables.

num: numeric, response variable.

bin: factor, binomial response variable with levels "no" and "yes".

cat: factor, multi categorical response with levels "none", "low", "medium" and "high".

x1: numeric covariate.

x2: numeric covariate.

x3: numeric covariate.

fac: factor covariate with levels "low", "medium" and "high".

id: factor, pixel identification index.

long: numeric, the longitude coordinate of the pixel.

lat: numeric, the latitude coordinate of the pixel.

See Also

[bayesx](#)

Examples

```
## Not run:
data("GAMart")

## normal response
b <- bayesx(num ~ fac + sx(x1) + sx(x2) + sx(x3) +
  sx(long, lat, bs = "te") + sx(id, bs = "re"),
  data = GAMart)
summary(b)
plot(b)
```

```
## binomial response
b <- bayesx(bin ~ fac + sx(x1) + sx(x2) + sx(x3) +
  sx(long, lat, bs = "te") + sx(id, bs = "re"),
  data = GAMart, family = "binomial", method = "REML")
summary(b)
plot(b)

## categorical response
b <- bayesx(cat ~ fac + sx(x1) + sx(x2) + sx(x3) +
  sx(long, lat, bs = "te") + sx(id, bs = "re"),
  data = GAMart, family = "cumprobit", method = "REML")
summary(b)
plot(b)

## End(Not run)
```

GCV

Generalized Cross Validation Criterion

Description

Generic function returning the generalized cross validation criterium of a fitted model object.

Usage

```
GCV(object, ...)
```

```
## S3 method for class 'bayesx'
GCV(object, ...)
```

Arguments

object	an object of class "bayesx".
...	specify for which model the criterion should be returned, e.g. type model = 1 to obtain the value for the first model. Only meaningful if object contains of more than one model.

Author(s)

Nikolaus Umlauf, Thomas Kneib, Stefan Lang, Achim Zeileis.

See Also

[bayesx](#).

Examples

```
## Not run:
## generate some data
set.seed(121)
n <- 200

## regressors
dat <- data.frame(x = runif(n, -3, 3))

## generate response
dat$y <- with(dat, 1.5 + sin(x) + rnorm(n, sd = 0.6))

## estimate model
b <- bayesx(y ~ sx(x), data = dat, method = "REML")

## extract GCV
GCV(b)

## End(Not run)
```

GermanyBnd

Germany Map

Description

This database produces a map of Germany since 2001 containing 439 administrative districts.

Usage

```
data("GermanyBnd")
```

Format

A [list](#) of class "bnd" containing 466 polygon matrices with x-coordinates in the first and y-coordinates in the second column each.

Source

<https://www.uni-goettingen.de/de/bayesx/550513.html>.

See Also

[plotmap](#), [read.bnd](#), [write.bnd](#)

Examples

```
## load GermanyBnd and plot it
data("GermanyBnd")
plotmap(GermanyBnd)
```

`get.neighbor`*Obtain Neighbors of Given Regions*

Description

Extracts the neighbors of a number of regions from a map in graph format.

Usage

```
get.neighbor(map, regions)
```

Arguments

<code>map</code>	map object in graph format.
<code>regions</code>	vector of names of regions for which the neighbors should be extracted.

Value

A list of vectors containing the neighbors of the elements in regions.

Author(s)

Felix Heinzl, Thomas Kneib.

See Also

[add.neighbor](#), [delete.neighbor](#)

Examples

```
file <- file.path(find.package("R2BayesX"), "examples", "Germany.gra")
germany <- read.gra(file)
get.neighbor(germany, "1001")
get.neighbor(germany, c("1001", "7339"))
```

`getscript`*Generate an executable R fitted model script*

Description

The function generates an executable R script for obtaining summary statistics, visualization of model diagnostics and term effect plots of a fitted [bayesx](#) model object.

Usage

```
getscript(object, file = NULL, device = NULL, ...)
```

Arguments

object	an object of class "bayesx".
file	optional, an output file the script is written to.
device	a graphical device function, e.g. pdf , see the examples and the help site of Devices for all available devices. If set, the script will have extra calls to the specified devices that will generate graphics to the specified file. If file = NULL, the working directory is taken.
...	arguments passed to devices, e.g. height and width of a graphical device.

Author(s)

Nikolaus Umlauf, Thomas Kneib, Stefan Lang, Achim Zeileis.

See Also

[bayesx](#).

Examples

```
## Not run:
## generate some data
set.seed(111)
n <- 500

## regressor
dat <- data.frame(x = runif(n, -3, 3))

## response
dat$y <- with(dat, 1.5 + sin(x) + rnorm(n, sd = 0.6))

## estimate model
b <- bayesx(y ~ sx(x), data = dat)

## generate the R script
## and print it
script <- getsript(b)
script

## with a pdf device
script <- getsript(b, device = pdf, height = 5, width = 6)
script

## End(Not run)
```

GRstats	<i>Compute Gelman and Rubin's convergence diagnostics from multicore BayesX models.</i>
---------	---

Description

This function takes a fitted [bayesx](#) object estimated with multiple chains/cores and computes the Gelman and Rubin's convergence diagnostic of the model parameters using function [gelman.diag](#) provided in package [coda](#).

Usage

```
GRstats(object, term = NULL, ...)
```

Arguments

object	an object of class "bayesx", returned from the model fitting function bayesx using the multiple chain or core option.
term	character or integer. The term for which the diagnostics should be computed, see also function samples .
...	arguments passed to function gelman.diag .

Value

An object returned from [gelman.diag](#).

Author(s)

Nikolaus Umlauf, Thomas Kneib, Stefan Lang, Achim Zeileis.

See Also

[bayesx](#), [gelman.diag](#), [samples](#).

Examples

```
## Not run:
## generate some data
set.seed(111)
n <- 500

## regressors
dat <- data.frame(x = runif(n, -3, 3), z = runif(n, -3, 3),
  w = runif(n, 0, 6), fac = factor(rep(1:10, n/10)))

## response
dat$y <- with(dat, 1.5 + sin(x) + cos(z) * sin(w) +
  c(2.67, 5, 6, 3, 4, 2, 6, 7, 9, 7.5)[fac] + rnorm(n, sd = 0.6))
```

```

## estimate model
b <- bayesx(y ~ sx(x) + sx(z, w, bs = "te") + fac,
  data = dat, method = "MCMC", chains = 3)

## obtain Gelman and Rubin's convergence diagnostics
GRstats(b, term = c("sx(x)", "sx(z,w)"))
GRstats(b, term = c("linear-samples", "var-samples"))

## of all parameters
GRstats(b, term = c("sx(x)", "sx(z,w)",
  "linear-samples", "var-samples"))

## End(Not run)

```

Interface between nb and gra format

Convert nb and gra format into each other

Description

Convert neighborhood structure objects of class "nb" from R-package **spdep** to graph objects of class "gra" from R-package **R2BayesX** and vice versa.

Usage

```

nb2gra(nbObject)
gra2nb(graObject)

```

Arguments

nbObject	neighborhood structure object of class "nb"
graObject	graph object of class "gra"

Value

Equivalent object in the other format.

Author(s)

Daniel Sabanes Bove.

See Also

[sp2bnd](#), [bnd2sp](#) for conversion between the geographical information formats and [read.gra](#), [write.gra](#) for the interface to the **R2BayesX** files.

Examples

```

## Not run: ## first nb to gra:
if(requireNamespace("spdep") &
  requireNamespace("sf") &
  requireNamespace("spData")) {
  library("spdep")
  library("spData")
  library("sf")

  columbus <- as(st_read(system.file("shapes/columbus.shp", package="spData")[1]), "Spatial")

  colNb <- poly2nb(columbus)

  ## ... here manual editing is possible ...
  ## then export to graph format
  colGra <- nb2gra(colNb)

  ## and save in BayesX file
  graFile <- tempfile()
  write.gra(colGra, file=graFile)

  ## now back from gra to nb:
  colGra <- read.gra(graFile)
  newColNb <- gra2nb(colGra)
  newColNb

  ## compare this with the original
  colNb

  ## only the call attribute does not match (which is OK):
  all.equal(newColNb, colNb, check.attributes = FALSE)
  attr(newColNb, "call")
  attr(colNb, "call")
}
## End(Not run)

```

Interface between *sp* and *bnd* format

Convert sp and bnd format into each other

Description

Convert geographical information objects of class "SpatialPolygons" (or specializations) from R-package **sp** to objects of class "bnd" from R-package **R2BayesX** and vice versa.

Usage

```

sp2bnd(spObject, regionNames, height2width, epsilon)
bnd2sp(bndObject)

```

Arguments

<code>spObject</code>	object of class "SpatialPolygons" (or specializations).
<code>regionNames</code>	character vector of region names (parallel to the Polygons list in <code>spObject</code>), defaults to the IDs.
<code>height2width</code>	ratio of total height to width, defaults to the bounding box values.
<code>epsilon</code>	how much can two polygons differ (in maximum squared Euclidean distance) and still match each other?, defaults to machine precision.
<code>bndObject</code>	object of class "bnd".

Value

Equivalent object in the other format.

Author(s)

Daniel Sabanes Bove.

See Also

[nb2gra](#), [gra2nb](#) for conversion between the neighborhood structure formats and [read.bnd](#), [write.bnd](#) for the interface to the **R2BayesX** files.

Examples

```
## Not run: ## bnd to sp:
file <- file.path(find.package("R2BayesX"), "examples", "Germany.bnd")
germany <- read.bnd(file)
spGermany <- bnd2sp(germany)

## plot the result together with the neighborhood graph
if(requireNamespace("spdep")) {
  library("spdep")
  plot(spGermany)
  nbGermany <- poly2nb(spGermany)
  plot(nbGermany, coords = coordinates(spGermany), add = TRUE)

  ## example with one region inside another
  spExample <- spGermany[c("7231", "7235"), ]
  plot(spExample)
  plot(poly2nb(spExample), coords = coordinates(spExample), add = TRUE)

  ## now back from sp to bnd:
  bndGermany <- sp2bnd(spGermany)
  plotmap(bndGermany)

  ## compare names and number of polygons
  stopifnot(
    identical(names(bndGermany), names(germany)),
    identical(length(bndGermany), length(germany))
  )
}
```

```
}
## End(Not run)
```

MunichBnd

Munich Map

Description

This database produces a city map of Munich containing 105 administrative districts.

Usage

```
data("MunichBnd")
```

Format

A `list` of class "bnd" containing 106 polygon matrices with x-coordinates in the first and y-coordinates in the second column each.

Source

<https://www.uni-goettingen.de/de/bayesx/550513.html>.

See Also

`plotmap`, `read.bnd`, `write.bnd`

Examples

```
## load MunichBnd and plot it
data("MunichBnd")
plotmap(MunichBnd)
```

parse.bayesx.input

Parse BayesX Input

Description

Function to parse bayesx input parameters which are then send to `write.bayesx.input`.

Usage

```
parse.bayesx.input(formula, data, weights = NULL,
  subset = NULL, offset = NULL, na.action = na.fail,
  contrasts = NULL, control = bayesx.control(...), ...)
```

Arguments

formula	symbolic description of the model (of type $y \sim x$). For more details see bayesx and sx .
data	a data.frame or list containing the model response variable and covariates required by the formula. By default the variables are taken from <code>environment(formula)</code> : typically the environment from which <code>bayesx</code> is called. Argument <code>data</code> may also be a character string defining the directory the data is stored, where the first row in the data set must contain the variable names and columns should be tab separated.
weights	prior weights on the data.
subset	an optional vector specifying a subset of observations to be used in the fitting process.
offset	can be used to supply a model offset for use in fitting.
na.action	a function which indicates what should happen when the data contain NA's.
contrasts	an optional list. See the <code>contrasts.arg</code> of model.matrix.default .
control	specify several global control parameters for <code>bayesx</code> , see bayesx.control .
...	arguments passed to bayesx.control .

Value

Returns a list of class "bayesx.input" which is send to [write.bayesx.input](#) for processing within [bayesx](#).

Author(s)

Nikolaus Umlauf, Thomas Kneib, Stefan Lang, Achim Zeileis.

Examples

```
parse.bayesx.input(y ~ x1 + sx(x2), data = "")
```

plot.bayesx	<i>Default BayesX Plotting</i>
-------------	--------------------------------

Description

Generic functions for plotting objects of class "bayesx" and model term classes "geo.bayesx", "linear.bayesx", "mrf.bayesx", "random.bayesx" and "sm.bayesx".

Usage

```
## S3 method for class 'bayesx'
plot(x, model = NULL, term = NULL, which = 1L, ask = FALSE, ...)
```

Arguments

x	a fitted bayesx object.
model	for which model the plot should be provided, either an integer or a character, e.g. <code>model = "mcmc.model"</code> .
term	the term that should be plotted, either an integer or a character, e.g. <code>term = "sx(x)"</code> .
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. <code>which = 1</code> . The first three arguments are all model term specific. For the residual model diagnostic plot options which may be set with <code>which = 5:8</code> .
ask	...
...	other graphical parameters passed to plotblock , plotmap , plot2d , plot3d , acf and density .

Details

Depending on the class of the term that should be plotted, function `plot.bayesx` calls one of the following plotting functions in the end:

- [plotblock](#),
- [plotsamples](#),
- [plotmap](#),
- [plot2d](#),
- [plot3d](#),
- [acf](#),
- [density](#),

For details on argument specifications, please see the help sites for the corresponding function.

If argument `x` contains of more than one model and e.g. `term = 2`, the second terms of all models will be plotted

Note

If a model is specified with a structured and an unstructured spatial effect, e.g. the model formula is something like `y ~ sx(id, bs = "mrf", map = MapBnd) + sx(id, bs = "re")`, the model output contains of one additional total spatial effect, named with `"sx(id):total"`. Also see the last example.

Author(s)

Nikolaus Umlauf, Thomas Kneib, Stefan Lang, Achim Zeileis.

See Also

[plotblock](#), [plotsamples](#), [plotmap](#), [plot2d](#), [plot3d](#), [bayesx](#), [read.bayesx.output](#).

Examples

```
## Not run:
## generate some data
set.seed(111)
n <- 500

## regressors
dat <- data.frame(x = runif(n, -3, 3), z = runif(n, -3, 3),
  w = runif(n, 0, 6), fac = factor(rep(1:10, n/10)))

## response
dat$y <- with(dat, 1.5 + sin(x) + cos(z) * sin(w) +
  c(2.67, 5, 6, 3, 4, 2, 6, 7, 9, 7.5)[fac] + rnorm(n, sd = 0.6))

## estimate model
b1 <- bayesx(y ~ sx(x) + sx(z, w, bs = "te") + fac,
  data = dat, method = "MCMC")

## plot p-spline term
plot(b1, term = 1)
## same with
plot(b1, term = "sx(x)")

## with residuals
plot(b1, term = "sx(x)", residuals = TRUE)

## plot tensor term
plot(b1, term = "sx(z,w)")

## use other palette
plot(b1, term = "sx(z,w)", col.surface = heat.colors)

## swap colors
plot(b1, term = "sx(z,w)", col.surface = heat.colors, swap = TRUE)

## plot tensor term with residuals
plot(b1, term = "sx(z,w)", residuals = TRUE)

## plot image and contour
plot(b1, term = "sx(z,w)", image = TRUE)
plot(b1, term = "sx(z,w)", image = TRUE, contour = TRUE)

## increase the grid
plot(b1, term = "sx(z,w)", image = TRUE, contour = TRUE, grid = 100)

## plot factor term
plot(b1, term = "fac")

## plot factor term with residuals
plot(b1, term = "fac", resid = TRUE, cex = 0.5)

## plot residual diagnostics
```

```

plot(b1, which = 5:8)

## plot variance sampling path of term sx(x)
plot(b1, term = 1, which = "var-samples")

## plot coefficients sampling paths of term sx(x)
plot(b1, term = 1, which = "coef-samples")

## plot the sampling path of the intercept
par(mfrow = c(1, 1))
plot(b1, which = "intcpt-samples")

## plot the autocorrelation function
## of the sampled intercept
plot(b1, which = "intcpt-samples",
     acf = TRUE, lag.max = 50)

## increase lags
plot(b1, which = "intcpt-samples",
     acf = TRUE, lag.max = 200)

## plot maximum autocorrelation
## of all sampled parameters in b1
plot(b1, which = "max-acf")

## plot maximum autocorrelation of
## all sampled coefficients of term sx(x)
plot(b1, term = "sx(x)", which = "coef-samples",
     max.acf = TRUE, lag.max = 100)

## now a spatial example
set.seed(333)

## simulate some geographical data
data("MunichBnd")
N <- length(MunichBnd); names(MunichBnd) <- 1:N
n <- N*5

## regressors
dat <- data.frame(id = rep(1:N, n/N), x1 = runif(n, -3, 3))
dat$sp <- with(dat, sort(runif(N, -2, 2), decreasing = TRUE)[id])
dat$re <- with(dat, rnorm(N, sd = 0.6)[id])

## response
dat$y <- with(dat, 1.5 + sin(x1) + sp + re + rnorm(n, sd = 0.6))

## estimate model
b2 <- bayesx(y ~ sx(x1) + sx(id, bs = "mrf", map = MunichBnd) +
            sx(id, bs = "re"), method = "MCMC", data = dat)

## summary statistics
summary(b2)

```

```

## plot structured spatial effect
plot(b2, term = "sx(id)", map = MunichBnd)

## plot unstructured spatial effect
plot(b2, term = "sx(id):re", map = MunichBnd)

## now without map
## generates a kernel density plot
## of the effects
plot(b2, term = "sx(id):mrf", map = FALSE)
plot(b2, term = "sx(id):re", map = FALSE)

## with approximate quantiles of the
## kernel density estimate
plot(b2, term = "sx(id):re", map = FALSE,
     kde.quantiles = TRUE, probs = c(0.025, 0.5, 0.975))

## plot the total spatial effect
plot(b2, term = "sx(id):total")
plot(b2, term = "sx(id):total", map = MunichBnd)

## combine model objects
b <- c(b1, b2)

## plot first term of second model
plot(b, model = 2, term = 1)
plot(b, model = "b2", term = "sx(x1)")

## plot second term of both models
plot(b, term = 2, map = MunichBnd)

## plot everything
plot(b)

## End(Not run)

```

plot2d

2D Effect Plot

Description

Function to plot simple 2D graphics for univariate effects/functions, typically used for objects of class "linear.bayesx" and "sm.bayesx" returned from function [bayesx](#) and [read.bayesx.output](#).

Usage

```

plot2d(x, residuals = FALSE, rug = TRUE, jitter = TRUE,
       col.residuals = NULL, col.lines = NULL, col.polygons = NULL,
       col.rug = NULL, c.select = NULL, fill.select = NULL,

```

```
data = NULL, sep = "", month = NULL, year = NULL,
step = 12, shift = NULL, trans = NULL, ...)
```

Arguments

<code>x</code>	a matrix or data frame, containing the covariate for which the effect should be plotted in the first column and at least a second column containing the effect, typically the structure for univariate functions returned within <code>bayesx</code> and <code>read.bayesx.output</code> model term objects is used, also see <code>fitted.bayesx</code> . Another possibility is to specify the plot via a formula, e.g. $y \sim x$, also see the example. <code>x</code> may also be a character file path to the data to be used for plotting.
<code>residuals</code>	if set to TRUE, partial residuals may also be plotted if available.
<code>rug</code>	add a <code>rug</code> to the plot.
<code>jitter</code>	if set to TRUE a <code>jittered rug</code> plot is added.
<code>col.residuals</code>	the color of the partial residuals.
<code>col.lines</code>	the color of the lines.
<code>col.polygons</code>	specify the background color of polygons, if <code>x</code> has at least 3 columns, i.e. column 2 and 3 can form one polygon.
<code>col.rug</code>	specify the color of the rug representation.
<code>c.select</code>	integer vector of maximum length of columns of <code>x</code> , selects the columns of the resulting data matrix that should be used for plotting. E.g. if <code>x</code> has 5 columns, then <code>c.select = c(1, 2, 5)</code> will select column 1, 2 and 5 for plotting. Note that first element of <code>c.select</code> should always be the column that holds the variable for the x-axis.
<code>fill.select</code>	integer vector, select pairwise the columns of the resulting data matrix that should form one polygon with a certain background color specified in argument <code>col</code> . E.g. <code>x</code> has three columns, or is specified with formula $f1 + f2 \sim x$, then setting <code>fill.select = c(0, 1, 1)</code> will draw a polygon with <code>f1</code> and <code>f2</code> as boundaries. If <code>x</code> has five columns or the formula is e.g. $f1 + f2 + f3 + f4 \sim x$, then setting <code>fill.select = c(0, 1, 1, 2, 2)</code> , the pairs <code>f1, f2</code> and <code>f3, f4</code> are selected to form two polygons.
<code>data</code>	if <code>x</code> is a formula, a <code>data.frame</code> or <code>list</code> . By default the variables are taken from <code>environment(x)</code> : typically the environment from which <code>plot2d</code> is called. Note that <code>data</code> may also be a character file path to the data.
<code>sep</code>	the field separator character when <code>x</code> or <code>data</code> is a character, see function <code>read.table</code> .
<code>month, year, step</code>	provide specific annotation for plotting estimation results for temporal variables. <code>month</code> and <code>year</code> define the minimum time point whereas <code>step</code> specifies the type of temporal data with <code>step = 4</code> , <code>step = 2</code> and <code>step = 1</code> corresponding to quarterly, half yearly and yearly data.
<code>shift</code>	numeric. Constant to be added to the smooth before plotting.
<code>trans</code>	function to be applied to the smooth before plotting, e.g., to transform the plot to the response scale.
<code>...</code>	other graphical parameters, please see the details.

Details

For 2D plots the following graphical parameters may be specified additionally:

- `cex`: specify the size of partial residuals,
- `lty`: the line type for each column that is plotted, e.g. `lty = c(1, 2)`,
- `lwd`: the line width for each column that is plotted, e.g. `lwd = c(1, 2)`,
- `poly.lty`: the line type to be used for the polygons,
- `poly.lwd`: the line width to be used for the polygons,
- `density angle, border`: see [polygon](#),
- ...: other graphical parameters, see function [plot](#).

Author(s)

Nikolaus Umlauf, Thomas Kneib, Stefan Lang, Achim Zeileis.

See Also

[plot.bayesx](#), [bayesx](#), [read.bayesx.output](#), [fitted.bayesx](#).

Examples

```
## generate some data
set.seed(111)
n <- 500
## regressor
dat <- data.frame(x = runif(n,-3,3))

## response
dat$y <- with(dat, 10 + sin(x) + rnorm(n,sd=0.6))

## Not run:
## estimate model
b <- bayesx(y ~ sx(x), data = dat)
summary(b)

## plot estimated effect
plot(b, which = 1)
plot(b, which = 1, rug = FALSE)

## extract fitted effects
f <- fitted(b, term = "sx(x)")

## now use plot2d
plot2d(f)
plot2d(f, residuals = TRUE)
plot2d(f, residuals = TRUE, pch = 2, col.resid = "green3")
plot2d(f, col.poly = NA, lwd = 1, lty = 1)
plot2d(f, col.poly = NA, lwd = 1, lty = 1, col.lines = 4)
plot2d(f, col.poly = c(2, 3), lwd = 1, col.lines = 4, lty = 1)
plot2d(f, lwd = c(1, 3, 2, 2, 3), col.poly = NA, lty = 1)
```

```

plot2d(f, lwd = c(1, 3, 2, 2, 3), col.poly = NA, lty = 1, col.lines = 2:6)
plot2d(f, lwd = c(1, 3, 2, 2, 3), col.poly = NA, lty = 1, col.lines = 2:6,
  resid = TRUE, pch = 4, col.resid = 7)

## End(Not run)

## another variation
plot2d(sin(x) ~ x, data = dat)
dat$f <- with(dat, sin(dat$x))
plot2d(f ~ x, data = dat)
dat$f1 <- with(dat, f + 0.1)
dat$f2 <- with(dat, f - 0.1)
plot2d(f1 + f2 ~ x, data = dat)
plot2d(f1 + f2 ~ x, data = dat, fill.select = c(0, 1, 1), lty = 0)
plot2d(f1 + f2 ~ x, data = dat, fill.select = c(0, 1, 1), lty = 0,
  density = 20, poly.lty = 1, poly.lwd = 2)
plot2d(f1 + f + f2 ~ x, data = dat, fill.select = c(0, 1, 0, 1),
  lty = c(0, 1, 0), density = 20, poly.lty = 1, poly.lwd = 2)

```

plot3d

3D Effect Plot

Description

Function to plot 3D graphics or image and/or contour plots for bivariate effects/functions, typically used for objects of class "sm.bayesx" and "geo.bayesx" returned from function [bayesx](#) and [read.bayesx.output](#).

Usage

```

plot3d(x, residuals = FALSE, col.surface = NULL,
  ncol = 99L, swap = FALSE, col.residuals = NULL, col.contour = NULL,
  c.select = NULL, grid = 30L, image = FALSE, contour = FALSE,
  legend = TRUE, cex.legend = 1, breaks = NULL, range = NULL,
  digits = 2L, d.persp = 1L, r.persp = sqrt(3), outscale = 0,
  data = NULL, sep = "", shift = NULL, trans = NULL,
  type = "interp", linear = FALSE, extrap = FALSE,
  k = 40, ...)

```

Arguments

x a matrix or data frame, containing the covariates for which the effect should be plotted in the first and second column and at least a third column containing the effect, typically the structure for bivariate functions returned within [bayesx](#) and [read.bayesx.output](#) model term objects is used, also see [fitted.bayesx](#). Another possibility is to specify the plot via a formula, e.g. for simple plotting of bivariate surfaces $z \sim x + y$, also see the example. **x** may also be a character file path to the data to be used for plotting.

residuals	if set to TRUE, partial residuals may also be plotted if available.
col.surface	the color of the surface, may also be a function, e.g. <code>col.surface = heat.colors</code> .
ncol	the number of different colors that should be generated, if <code>col.surface</code> is a function.
swap	if set to TRUE colors will be represented in reverse order.
col.residuals	the color of the partial residuals, or if <code>contour = TRUE</code> the color of the contour lines.
col.contour	the color of the contour lines.
c.select	integer vector of maximum length of columns of <code>x</code> , selects the columns of the resulting data matrix that should be used for plotting. E.g. if <code>x</code> has 5 columns, then <code>c.select = c(1, 2, 5)</code> will select column 1, 2 and 5 for plotting. If <code>c.select = 95</code> or <code>c.select = 80</code> , function <code>plot3d</code> will search for the corresponding columns to plot a 95% or 80% confidence surfaces respectively. Note that if e.g. <code>c.select = c(1, 2)</code> , <code>plot3d</code> will use columns 1 + 2 and 2 + 2 for plotting.
grid	the grid size of the surface(s).
image	if set to TRUE, an <code>image.plot</code> is drawn.
contour	if set to TRUE, a <code>contour</code> plot is drawn.
legend	if <code>image = TRUE</code> an additional legend may be added to the plot.
cex.legend	the expansion factor for the legend text, see <code>text</code> .
breaks	a set of breakpoints for the colors: must give one more breakpoint than <code>ncol</code> .
range	specifies a certain range values should be plotted for.
digits	specifies the legend decimal places.
d.persp	see argument <code>d</code> in function <code>persp</code> .
r.persp	see argument <code>r</code> in function <code>persp</code> .
outscale	scales the outer ranges of <code>x</code> and <code>z</code> limits used for interpolation.
data	if <code>x</code> is a formula, a <code>data.frame</code> or <code>list</code> . By default the variables are taken from <code>environment(x)</code> : typically the environment from which <code>plot3d</code> is called. Note that <code>data</code> may also be a character file path to the data.
sep	the field separator character when <code>x</code> or <code>data</code> is a character, see function <code>read.table</code> .
shift	numeric. Constant to be added to the smooth before plotting.
trans	function to be applied to the smooth before plotting, e.g., to transform the plot to the response scale.
type	character. Which type of interpolation method should be used. The default is <code>type = "interp"</code> , see function <code>interp</code> . The two other options are <code>type = "mba"</code> , which calls function <code>mba.surf</code> of package MBA , or <code>type = "mgcv"</code> , which uses a spatial smoother withing package mgcv for interpolation. The last option is definitely the slowest, since a full regression model needs to be estimated.
linear	logical. Should linear interpolation be used withing function <code>interp</code> ?
extrap	logical. Should interpolations be computed outside the observation area (i.e., extrapolated)?

k integer. The number of basis functions to be used to compute the interpolated surface when `type = "mgcv"`.

... parameters passed to [colorlegend](#) if an image plot with legend is drawn, also other graphical parameters, please see the details.

Details

For 3D plots the following graphical parameters may be specified additionally:

- `cex`: specify the size of partial residuals,
- `col`: it is possible to specify the color for the surfaces if `se > 0`, then e.g. `col = c("green", "black", "red")`,
- `pch`: the plotting character of the partial residuals,
- `...`: other graphical parameters passed functions [persp](#), [image.plot](#) and [contour](#).

Author(s)

Nikolaus Umlauf, Thomas Kneib, Stefan Lang, Achim Zeileis.

See Also

[plot.bayesx](#), [bayesx](#), [read.bayesx.output](#), [fitted.bayesx](#), [colorlegend](#).

Examples

```
## generate some data
set.seed(111)
n <- 500

## regressors
dat <- data.frame(z = runif(n, -3, 3), w = runif(n, 0, 6))

## response
dat$y <- with(dat, 1.5 + cos(z) * sin(w) + rnorm(n, sd = 0.6))

## Not run:
## estimate model
b <- bayesx(y ~ sx(z, w, bs = "te", knots = 5), data = dat, method = "REML")
summary(b)

## plot estimated effect
plot(b, term = "sx(z,w)")

## extract fitted effects
f <- fitted(b, term = "sx(z,w)")

## now use plot3d
plot3d(f)
plot3d(f, swap = TRUE)
plot3d(f, residuals = TRUE)
plot3d(f, resid = TRUE, cex.resid = 0.1)
```

```

plot3d(f, resid = TRUE, pch = 2, col.resid = "green3")
plot3d(f, resid = TRUE, c.select = 95, cex.resid = 0.1)
plot3d(f, resid = TRUE, c.select = 80, cex.resid = 0.1)
plot3d(f, grid = 100, border = NA)
plot3d(f, c.select = 95, border = c("red", NA, "green"),
  col.surface = c(1, NA, 1), resid = TRUE, cex.resid = 0.2)

## now some image and contour
plot3d(f, image = TRUE, legend = FALSE)
plot3d(f, image = TRUE, legend = TRUE)
plot3d(f, image = TRUE, contour = TRUE)
plot3d(f, image = TRUE, contour = TRUE, swap = TRUE)
plot3d(f, image = TRUE, contour = TRUE, col.contour = "white")
plot3d(f, contour = TRUE)
op <- par(no.readonly = TRUE)
par(mfrow = c(1, 3))
plot3d(f, image = TRUE, contour = TRUE, c.select = 3)
plot3d(f, image = TRUE, contour = TRUE, c.select = "Estimate")
plot3d(f, image = TRUE, contour = TRUE, c.select = "97.5")
par(op)

## End(Not run)

## another variation
dat$f1 <- with(dat, sin(z) * cos(w))
with(dat, plot3d(cbind(z, w, f1)))

## same with formula
plot3d(sin(z) * cos(w) ~ z + w, zlab = "f(z,w)", data = dat)
plot3d(sin(z) * cos(w) ~ z + w, zlab = "f(z,w)", data = dat,
  ticktype = "detailed")

## play with palettes
plot3d(sin(z) * cos(w) ~ z + w, col.surface = heat.colors, data = dat)
plot3d(sin(z) * cos(w) ~ z + w, col.surface = topo.colors, data = dat)
plot3d(sin(z) * cos(w) ~ z + w, col.surface = cm.colors, data = dat)
plot3d(sin(z) * cos(w) ~ z + w, col.surface = rainbow, data = dat)
plot3d(sin(z) * cos(w) ~ z + w, col.surface = terrain.colors, data = dat)

plot3d(sin(z) * cos(w) ~ z + w, col.surface = rainbow_hcl, data = dat)
plot3d(sin(z) * cos(w) ~ z + w, col.surface = diverge_hcl, data = dat)
plot3d(sin(z) * cos(w) ~ z + w, col.surface = sequential_hcl, data = dat)

plot3d(sin(z) * cos(w) ~ z + w,
  col.surface = rainbow_hcl(n = 99, c = 300, l = 80, start = 0, end = 100),
  data = dat)
plot3d(sin(z) * cos(w) ~ z + w,
  col.surface = rainbow_hcl(n = 99, c = 300, l = 80, start = 0, end = 100),
  image = TRUE, grid = 200, data = dat)

```

Description

Function to plot effects for model terms including factor, or group variables for random effects, typically used for objects created within `bayesx` or `read.bayesx.output`.

Usage

```
plotblock(x, residuals = FALSE, range = c(0.3, 0.3),
  col.residuals = "black", col.lines = "black", c.select = NULL,
  fill.select = NULL, col.polygons = NULL, data = NULL,
  shift = NULL, trans = NULL, ...)
```

Arguments

<code>x</code>	either a list of length of the unique factors, where each list element contains the estimated effects for one factor as a matrix, see <code>fitted.bayesx</code> , or one data matrix with first column as the group or factor variable. Also formulas are accepted, e.g it is possible to specify the plot with $f \sim x$ or $f1 + f2 \sim x$. By convention, the covariate for which effects should be plotted, is always in the first column in the resulting data matrix, that is used for plotting, i.e. in the second formula example, the data matrix is <code>cbind(x, f1, f2)</code> , also see argument <code>c.select</code> and <code>fill.select</code> .
<code>residuals</code>	if set to <code>TRUE</code> , partial residuals will be plotted if available. Partial residuals may be set as an <code>attribute</code> of <code>x</code> named <code>"partial.resids"</code> , where the partial residuals must be a matrix with first column specifying the covariate, and second column the partial residuals that should be plotted.
<code>range</code>	numeric vector, specifying the left and right bound of the block.
<code>col.residuals</code>	the color of the partial residuals.
<code>col.lines</code>	vector of maximum length of columns of <code>x</code> minus 1, specifying the color of the lines.
<code>c.select</code>	integer vector of maximum length of columns of <code>x</code> , selects the columns of the resulting data matrix that should be used for plotting. E.g. if <code>x</code> has 5 columns, then <code>c.select = c(1, 2, 5)</code> will select column 1, 2 and 5 for plotting. Note that first element of <code>c.select</code> should always be 1, since this is the column of the covariate the effect is plotted for.
<code>fill.select</code>	integer vector, select pairwise the columns of the resulting data matrix that should form one polygon with a certain background color specified in argument <code>col</code> . E.g. <code>x</code> has three columns, or is specified with formula $f1 + f2 \sim x$, then setting <code>fill.select = c(0, 1, 1)</code> will draw a polygon with <code>f1</code> and <code>f2</code> as boundaries. If <code>x</code> has five columns or the formula is e.g. $f1 + f2 + f3 + f4 \sim x$, then setting <code>fill.select = c(0, 1, 1, 2, 2)</code> , the pairs <code>f1, f2</code> and <code>f3, f4</code> are selected to form two polygons.
<code>col.polygons</code>	specify the background color for the upper and lower confidence bands, e.g. <code>col = c("green", "red")</code> .
<code>data</code>	if <code>x</code> is a formula, a <code>data.frame</code> or <code>list</code> . By default the variables are taken from <code>environment(x)</code> : typically the environment from which <code>plotblock</code> is called.
<code>shift</code>	numeric. Constant to be added to the smooth before plotting.

trans function to be applied to the smooth before plotting, e.g., to transform the plot to the response scale.

... graphical parameters, please see the details.

Details

Function `plotblock` draws for every factor or group the effect as a "block" in one graphic, i.e. similar to boxplots, estimated fitted effects, e.g. containing quantiles for MCMC estimated models, are drawn as one block, where the upper lines represent upper quantiles, the middle line the mean or median, and lower lines lower quantiles, also see the examples. The following graphical parameters may be supplied additionally:

- `cex`: specify the size of partial residuals,
- `lty`: the line type for each column that is plotted, e.g. `lty = c(1, 2)`,
- `lwd`: the line width for each column that is plotted, e.g. `lwd = c(1, 2)`,
- `poly.lty`: the line type to be used for the polygons,
- `poly.lwd`: the line width to be used for the polygons,
- `density.angle`, `border`: see [polygon](#),
- ...: other graphical parameters, see function [plot](#).

Author(s)

Nikolaus Umlauf, Thomas Kneib, Stefan Lang, Achim Zeileis.

See Also

[plot.bayesx](#), [bayesx](#), [read.bayesx.output](#), [fitted.bayesx](#).

Examples

```
## generate some data
set.seed(111)
n <- 500

## regressors
dat <- data.frame(fac = factor(rep(1:10, n/10)))

## response
dat$y <- with(dat, 1.5 + c(2.67, 5, 6, 3, 4, 2, 6, 7, 9, 7.5)[fac] +
  rnorm(n, sd = 0.6))

## Not run:
## estimate model
b <- bayesx(y ~ fac, data = dat)
summary(b)

## plot factor term effects
plot(b, term = "fac")
```

```

## extract fitted effects
f <- fitted(b, term = "fac")

## now use plotblock
plotblock(f)

## some variations
plotblock(f, col.poly = c(2, 3))
plotblock(f, col.poly = NA, lwd = c(2, 1, 1, 1, 1))
plotblock(f, col.poly = NA, lwd = 3, range = c(0.5,0.5))
plotblock(f, col.poly = NA, lwd = 3, col.lines = 1:5, lty = 1)
plotblock(f, col.poly = NA, lwd = 3, col.lines = 1:5,
  lty = c(3, 1, 2, 2, 1))
plotblock(f, resid = TRUE)
plotblock(f, resid = TRUE, cex = 0.1)
plotblock(f, resid = TRUE, cex = 0.1, col.resid = 2)
plotblock(f, resid = TRUE, cex = 2, col.resid = 3, pch = 3)
plotblock(f, lty = 0, poly.lty = 1, density = c(5, 20))
plotblock(f, lty = 0, poly.lty = 1, density = c(5, 20),
  poly.lwd = c(1, 2))
plotblock(f, lty = 0, poly.lty = c(1, 2), density = c(5, 20))
plotblock(f, lty = 0, poly.lty = c(1, 2), density = c(5, 20),
  border = c("red", "green3"))
plotblock(f, lty = 0, poly.lty = c(1, 2), density = c(5, 20),
  border = c("red", "green3"), col.poly = c("blue", "yellow"))
plotblock(f, lty = c(1,0,0,0,0), poly.lty = c(1, 2),
  density = c(5, 20), border = c("red", "green3"),
  col.poly = c("blue", "yellow"))
plotblock(f, lty = c(1,0,0,0,0), poly.lty = c(1, 2),
  density = c(20, 20), border = c("red", "green3"),
  col.poly = c("blue", "yellow"), angle = c(10, 75))

## End(Not run)

## another example
plotblock(y ~ fac, data = dat, range = c(0.45, 0.45))

dat <- data.frame(fac = factor(rep(1:10, n/10)))
dat$y <- with(dat, c(2.67, 5, 6, 3, 4, 2, 6, 7, 9, 7.5)[fac])
plotblock(y ~ fac, data = dat)
plotblock(cbind(y - 0.1, y + 0.1) ~ fac, data = dat)
plotblock(cbind(y - 0.1, y + 0.1) ~ fac, data = dat,
  fill.select = c(0, 1, 1))
plotblock(cbind(y - 0.1, y + 0.1) ~ fac, data = dat,
  fill.select = c(0, 1, 1), poly.lty = 2, lty = 1,
  border = "grey5")

```

Description

The function takes a [list](#) polygons and draws the corresponding map. Different colors for each polygon can be used. Typically used for objects of class "mrf.bayesx" and "random.bayesx" returned from function [bayesx](#) and [read.bayesx.output](#).

Usage

```
plotmap(map, x = NULL, id = NULL, c.select = NULL, legend = TRUE,
        missing = TRUE, swap = FALSE, range = NULL, names = FALSE,
        values = FALSE, col = NULL, ncol = 100, breaks = NULL,
        cex.legend = 1, cex.names = 1, cex.values = cex.names, digits = 2L,
        mar.min = 2, add = FALSE, interp = FALSE, grid = 200,
        land.only = FALSE, extrap = FALSE, outside = FALSE, type = "interp",
        linear = FALSE, k = 40, p.pch = 15, p.cex = 1, shift = NULL,
        trans = NULL, ...)
```

Arguments

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, also see polygon .
x	a matrix or data frame with two columns, first column indicates the region and second column the the values which will define the background colors of the polygons, e.g. fitted values from bayesx . More columns are possible, e.g. quantiles, which can accessed with argument <code>se</code> .
id	if argument <code>x</code> is a vector, argument <code>id</code> should contain a character vector of the same length of <code>x</code> with entries indicating the polygon the i -th value of <code>x</code> belongs to, i.e. <code>id</code> must contain the same names as polygon names in <code>map</code> .
c.select	select the column of the data in <code>x</code> which should be used for plotting, may be an integer or character with the corresponding column name.
legend	if set to TRUE, a legend will be shown.
missing	should polygons be plotted for which no data is available in <code>x</code> ?
swap	if set to TRUE, colors will be represented in reverse order.
range	specify the range of values in <code>x</code> which should enter the plot, e.g. only values between -2 and 2 are of interest then <code>range = c(-2, 2)</code> .
names	if set to TRUE the name for each polygon will also be plotted at the centroids of the corresponding polygons.
values	if set to TRUE the corresponding values for each polygon will also be plotted at the centroids of the polygons.
col	the color of the surface, may also be a function, e.g. <code>col.surface = heat.colors</code> .
ncol	the number of different colors that should be generated if <code>col</code> is a function.
breaks	a set of breakpoints for the colors: must give one more breakpoint than <code>ncol</code> .
cex.legend	text size of the numbers in the legend.
cex.names	text size of the names if <code>names = TRUE</code> .

<code>cex.values</code>	text size of the names if <code>values = TRUE</code> .
<code>digits</code>	specifies the legend decimal places.
<code>mar.min</code>	Controls the definition of boundaries. Could be either <code>NULL</code> for individual settings of <code>mar</code> or a value which defines <code>mar</code> as follows: The boundaries will be calculated according to the height to width ratio of the map with minimal boundary <code>mar.min</code> .
<code>add</code>	if set to <code>TRUE</code> , the map will be added to an existing plot.
<code>interp</code>	logical. Should the values provided in argument <code>x</code> be interpolated to obtain a smooth colored map.
<code>grid</code>	integer. Defines the number of grid cells to be used for interpolation.
<code>land.only</code>	if set to <code>TRUE</code> , only interpolated pixels that cover land are drawn, see also function map.where .
<code>extrap</code>	logical. Should interpolations be computed outside the observation area (i.e., extrapolated)?
<code>outside</code>	logical. Should interpolated values outside the boundaries of the map be plotted.
<code>type</code>	character. Which type of interpolation method should be used. The default is <code>type = "interp"</code> , see function interp . The two other options are <code>type = "mba"</code> , which calls function <code>mba.surf</code> of package MBA , or <code>type = "mgcv"</code> , which uses a spatial smoother withing package mgcv for interpolation. The last option is definitely the slowest, since a full regression model needs to be estimated.
<code>linear</code>	logical. Should linear interpolation be used withing function interp ?
<code>k</code>	integer. The number of basis functions to be used to compute the interpolated surface when <code>type = "mgcv"</code> .
<code>p.pch</code>	numeric. The point size of the grid cells when using interpolation.
<code>p.cex</code>	numeric. The size of the grid cell points when using interpolation.
<code>shift</code>	numeric. Constant to be added to the smooth before plotting.
<code>trans</code>	function to be applied to the smooth before plotting, e.g., to transform the plot to the response scale.
<code>...</code>	parameters to be passed to colorlegend and others, e.g. change the border of the polygons and <code>density</code> , see polygon . Please see the examples.

Author(s)

Nikolaus Umlauf, Thomas Kneib, Stefan Lang, Achim Zeileis.

See Also

[plot.bayesx](#), [read.bnd](#), [colorlegend](#).

Examples

```

## load a sample map
data("FantasyBnd")

## plot the map
op <- par(no.readonly = TRUE)
plotmap(FantasyBnd, main = "Example of a plain map")
plotmap(FantasyBnd, lwd = 1, main = "Example of a plain map")
plotmap(FantasyBnd, lwd = 1, lty = 2)
plotmap(FantasyBnd, lwd = 1, lty = 2, border = "green3")
plotmap(FantasyBnd, lwd = 1, lty = 2, border = "green3",
        density = 50)
plotmap(FantasyBnd, lwd = 1, lty = 2,
        border = c("red", "green3"),
        density = c(10, 20), angle = c(5, 45))
plotmap(FantasyBnd, lwd = 1, lty = 2,
        border = c("red", "green3"),
        density = c(10, 20), angle = c(5, 45),
        col = c("blue", "yellow"))
plotmap(FantasyBnd, col = gray.colors(length(FantasyBnd)))

## add some values to the corresponding polygon areas
## note that the first column in matrix val contains
## the region identification index
x <- cbind(as.integer(names(FantasyBnd)), runif(length(FantasyBnd), -2, 2))
plotmap(FantasyBnd, x = x)

## now only plot values for some certain regions
set.seed(432)
samps <- sample(x[,1], 4)
nx <- x[samps,]
plotmap(FantasyBnd, x = nx, density = 20)

## play with legend
plotmap(FantasyBnd, x = x, names = TRUE, legend = FALSE)
plotmap(FantasyBnd, x = nx, density = 20, pos = c(0, 1))
plotmap(FantasyBnd, x = nx, density = 20, pos = c(0, 0.8),
        side.legend = 2)
plotmap(FantasyBnd, x = nx, density = 20, pos = c(0, 0.8),
        side.legend = 2, side.tick = 2)
plotmap(FantasyBnd, x = nx, density = 20, pos = c(0, 0.8),
        side.legend = 2, side.tick = 2, cex.legend = 0.5)
plotmap(FantasyBnd, x = x, values = TRUE,
        pos = c(-0.15, -0.12))
plotmap(FantasyBnd, x = nx, values = TRUE,
        pos = c(-0.07, -0.22), width = 2,
        at = nx[,2], side.legend = 2, distance.labels = 3,
        density = 20)
plotmap(FantasyBnd, x = nx, values = TRUE,
        pos = c(-0.07, -0.22), width = 2,
        at = nx[,2], side.legend = 2, distance.labels = 3,
        density = 20, symmetric = FALSE,

```

```

col = heat_hcl, swap = TRUE)
plotmap(FantasyBnd, x = nx, values = TRUE,
  pos = c(-0.07, -0.22), width = 2,
  at = nx[,2], side.legend = 2, distance.labels = 3,
  density = 20, symmetric = FALSE,
  col = heat_hcl, swap = TRUE, range = c(-5, 5))
plotmap(FantasyBnd, x = nx, values = TRUE,
  pos = c(-0.07, -0.22), width = 2,
  at = nx[,2], side.legend = 2, distance.labels = 3,
  density = 20, symmetric = FALSE,
  col = heat_hcl, swap = TRUE, lrange = c(-5, 5))
plotmap(FantasyBnd, x = nx, values = TRUE,
  pos = c(-0.07, -0.22), width = 2,
  at = nx[,2], side.legend = 2, distance.labels = 3,
  density = 20, symmetric = FALSE,
  col = heat_hcl, swap = TRUE,
  ncol = 4, breaks = seq(-2, 2, length = 5))

## more position options
plotmap(FantasyBnd, x = nx, density = 20, pos = "bottomleft")
plotmap(FantasyBnd, x = nx, density = 20, pos = "topleft")
plotmap(FantasyBnd, x = nx, density = 20, pos = "topright")
plotmap(FantasyBnd, x = nx, density = 20, pos = "bottomright")
plotmap(FantasyBnd, x = nx, density = 20, pos = "right")
par(op)

# load and plot a map from GermanyBnd
op <- par(no.readonly = TRUE)
data("GermanyBnd")
plotmap(GermanyBnd, main = "Map of GermanyBnd")
n <- length(GermanyBnd)

# add some colors
plotmap(GermanyBnd, col = rainbow(n))
plotmap(GermanyBnd, col = heat.colors(n))
plotmap(GermanyBnd, col = topo.colors(n))
plotmap(GermanyBnd, col = cm.colors(n))
plotmap(GermanyBnd, col = gray.colors(n))
plotmap(GermanyBnd, col = c("green", "green3"))
par(op)

## now with bayesx
set.seed(333)

## simulate some geographical data
data("MunichBnd")
N <- length(MunichBnd); names(MunichBnd) <- 1:N
n <- N*5

## regressors
dat <- data.frame(id = rep(1:N, n/N))
dat$sp <- with(dat, sort(runif(N, -2, 2), decreasing = TRUE)[id])

```

```

## response
dat$y <- with(dat, 1.5 + sp + rnorm(n, sd = 0.6))

## Not run:
## estimate model
b <- bayesx(y ~ sx(id, bs = "mrf", map = MunichBnd),
  method = "MCMC", data = dat)

## summary statistics
summary(b)

## plot spatial effect
op <- par(no.readonly = TRUE)
plot(b, map = MunichBnd)
plot(b, map = MunichBnd, c.select = "97.5")
plot(b, map = MunichBnd, c.select = "2.5")
plot(b, map = MunichBnd, c.select = "50")
plot(b, map = MunichBnd, names = TRUE,
  cex.names = 0.5, cex.legend = 0.8)
plot(b, map = MunichBnd, range = c(-0.5, 0.5))
plot(b, map = MunichBnd, range = c(-5, 5))
plot(b, map = MunichBnd, col = heat_hcl,
  swap = TRUE, symmetric = FALSE)
par(op)

## End(Not run)

```

plotsamples

Plot Sampling Path(s) of Coefficient(s) and Variance(s)

Description

This function plots the sampling paths of coefficient(s) and variance(s) stored in model term objects typically returned from function [bayesx](#) or [read.bayesx.output](#).

Usage

```
plotsamples(x, selected = "NA", acf = FALSE, var = FALSE,
  max.acf = FALSE, subset = NULL, ...)
```

Arguments

x	a vector or matrix, where each column represents a different sampling path to be plotted.
selected	a character string containing the term name the sampling paths are plotted for.
acf	if set to TRUE, the autocorrelation function for each sampling path is plotted.
var	indicates whether coefficient or variance sampling paths are displayed and simply changes the main title.

max.acf if set to TRUE, plotsamples will evaluate the maximum autocorrelation over all parameters of each sample.

subset integer. An index which selects the coefficients for which sampling paths should be plotted.

... other graphical parameters to be passed to [plot](#) and [acf](#), e.g. argument lag.max if acf = TRUE. An argument ask controls the display when more than 12 sampling paths should be plotted.

Author(s)

Nikolaus Umlauf, Thomas Kneib, Stefan Lang, Achim Zeileis.

See Also

[plot.bayesx](#), [bayesx](#), [read.bayesx.output](#).

Examples

```
## generate some data
set.seed(111)
n <- 500

## regressors
dat <- data.frame(x = runif(n, -3, 3))

## response
dat$y <- with(dat, 1.5 + sin(x) + rnorm(n, sd = 0.6))

## Not run:
## estimate model
b <- bayesx(y ~ sx(x), data = dat)
summary(b)

## plot sampling path for
## the variance
plot(b, term = "sx(x)", which = "var-samples")

## plot sampling paths for
## coefficients
plot(b, term = "sx(x)", which = "coef-samples")

## plot maximum autocorrelation of
## all sampled parameters of term s(x)
plot(b, term = "sx(x)", which = "coef-samples", max.acf = TRUE)

## extract samples of term sx(x)
sax <- as.matrix(samples(b, term = "sx(x)"))

## now use plotsamples
plotsamples(sax, selected = "sx(x)")
```

```
## some variations
plotsamples(sax, selected = "sx(x)", acf = TRUE)
plotsamples(sax, selected = "sx(x)", acf = TRUE, lag.max = 200)

## End(Not run)
```

predict.bayesx *Prediction from fitted BayesX objects*

Description

Takes a fitted "bayesx" object returned from `bayesx` and produces predictions by refitting the initial model with weights set to zero for new observations.

Usage

```
## S3 method for class 'bayesx'
predict(object, newdata, model = NULL,
        type = c("response", "link", "terms", "model"),
        na.action = na.pass, digits = 5, ...)
```

Arguments

<code>object</code>	an object of class "bayesx" or "bayesx.hpc".
<code>newdata</code>	a data frame or list containing the values of the model covariates at which predictions are required. If missing <code>newdata</code> is the <code>model.frame</code> of the provided model.
<code>model</code>	for which model should predictions be calculated, either an integer or a character, e.g. <code>model = "mcmc.model"</code> . Note that exactly one model must be selected within argument <code>model</code> to compute predicted values!
<code>type</code>	when <code>type = "response"</code> , the default, predictions on the scale of the response are returned, "link" returns the linear predictor. When <code>type = "terms"</code> , each component of the linear predictor is returned, but excludes any offset and intercept. If <code>type = "model"</code> , the full model returned from updating the initial model with weights, that is used for computing predictions, is returned.
<code>na.action</code>	function determining what should be done with missing values in <code>newdata</code> .
<code>digits</code>	predictions should usually be based on the new values provided in argument <code>newdata</code> . However, since this prediction method uses refitting of the model with weights, predictions for model terms need to be matched with the new observations. BayesX returns values with a lower precision than R, therefore argument <code>digits</code> is used to round values when <code>type = "terms"</code> , to find matching <code>newdata</code> pairs in the fitted objects returned from the refitted model and the new data. Note that this is a workaround and not 100% bulletproof. It is recommended to compute predictions for <code>type = "response"</code> or <code>type = "link"</code> .
<code>...</code>	not used.

Value

Depending on the specifications of argument type.

Note

This prediction method is based on refitting the initial model with weights, i.e., if new observations lie outside the domain of the respective covariate, the knot locations when using e.g. P-splines are calculated using the old and the new data. Hence, if there are large gaps between the old data domain and new observations, this could affect the overall fit of the estimated spline, i.e., compared to the initial model fit there will be smaller or larger differences depending on the newdata provided.

Author(s)

Nikolaus Umlauf, Thomas Kneib, Stefan Lang, Achim Zeileis.

See Also

[fitted.bayesx](#), [bayesx](#).

Examples

```
## Not run:
## generate some data
set.seed(121)
n <- 500

## regressors
dat <- data.frame(x = runif(n, -3, 3), z = runif(n, 0, 1),
  w = runif(n, 0, 3))

## generate response
dat$y <- with(dat, 1.5 + sin(x) + z -3 * w + rnorm(n, sd = 0.6))

## estimate model
b <- bayesx(y ~ sx(x) + z + w, data = dat)

## create some data for which predictions are required
nd <- data.frame(x = seq(2, 5, length = 100), z = 1, w = 0)

## prediction model from refitting with weights
nd$fit <- predict(b, newdata = nd)
plot(fit ~ x, type = "l", data = nd)

## End(Not run)
```

read.bayesx.output *Read BayesX Output from Directories*

Description

This function automatically reads in **BayesX** estimation output which is stored in an output directory.

Usage

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

Arguments

dir	a character string, specifies the directory file where BayesX output is stored.
model.name	a character string, specifies the base name of the model that should be read in, also see the examples. If not supplied read.bayesx.output tries to read in all existing model outputs in dir, every model is then stored as one element in the output list. By convention, read.bayesx.output searches for existing .tex output files, and others, to identify different models in the dir folder.

Details

The function searches for model term objects in the specified directory, which are then stored in a list. Each model term object will be of class xx.bayesx, so the generic functions described in [plot.bayesx](#) may be applied for visualizing the results. In addition summary statistics of the models may be printed to the R console with [summary.bayesx](#).

Value

read.bayesx.output typically returns a list of class "bayesx" with the first element containing a list with the following objects:

formula	the STAR formula used,
bayesx.setup	an object of class "bayesx.input", see parse.bayesx.input ,
bayesx.prg	a character containing the .prg file used for estimation with run.bayesx ,
bayesx.run	details on processing with run.bayesx ,
call	the original function call,
fitted.values	the fitted values of the estimated model,
residuals	the residuals of the estimated model,
effects	a list containing fitted effects of model terms, also see fitted.bayesx and samples ,
fixed.effects	a matrix with estimation results for fixed effects,
variance	estimation results for the variance parameter of the model,
smooth.hyp	a matrix with estimation results smooth terms,
model.fit	list containing additional information to be supplied to summary.bayesx .

Author(s)

Nikolaus Umlauf, Thomas Kneib, Stefan Lang, Achim Zeileis.

See Also

[summary.bayesx](#), [plot.bayesx](#), [samples](#).

Examples

```
## Not run: ## load example data from
## package example folder
dir <- file.path(find.package("R2BayesX"), "/examples/ex01")
b <- read.bayesx.output(dir)

## some model summaries
print(b)
summary(b)

## now plot estimated effects
plot(b)

## 2nd example
dir <- file.path(find.package("R2BayesX"), "/examples/ex02")
list.files(dir)

## dir contains of 2 different
## base names
## 01 only one nonparametric effect
b <- read.bayesx.output(dir, model.name = "nonparametric")
plot(b)

## 02 only one bivariate
## nonparametric effect
b <- read.bayesx.output(dir, model.name = "surface")
plot(b)

## End(Not run)
```

read.bnd

Read Geographical Information in Boundary Format

Description

Reads the geographical information provided in a file in boundary format and stores it in a map object.

Usage

```
read.bnd(file, sorted = FALSE)
```

Arguments

file	name of the boundary file to be read.
sorted	should the regions be ordered by the numbers specifying the region names (sorted = TRUE)?

Details

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

The following is an example of a boundary file as provided in file `Germany.bnd` in the examples folder of this package.

```
"1001",9
2534.64771,8409.77539
2554.54712,8403.92285
2576.78735,8417.96973
2592.00439,8366.46582
2560.39966,8320.81445
2507.72534,8319.64453
2496.02002,8350.07813
2524.11304,8365.29492
2534.64771,8409.77539
"1002",18
2987.64697,7774.17236
2954.87183,7789.38916
...
```

Hence, the region code of the first region is "1001" and contains of 9 points that form its polygon. The second region has region code "1002" and contains of 18 polygon points (note that only the first two points are shown).

Value

Returns a list of polygons that form the map. Additional attributes are

surrounding	Parallel list where for each polygon, the name of a possible surrounding region is saved.
height2width	Ratio between height and width of the map. Allows customised drawing and storage in files by specifying the appropriate height and width.

Author(s)

Daniel Sabanes Bove, Felix Heinzl, Thomas Kneib, Andreas Brezger.

References

BayesX Reference Manual. Available at <https://www.uni-goettingen.de/de/bayesx/550513.html>.

See Also

[write.bnd](#), [plotmap](#), [read.gra](#), [write.gra](#).

Examples

```
file <- file.path(find.package("R2BayesX"), "examples", "Germany.bnd")
germany <- read.bnd(file)
plotmap(germany)
```

read.gra

Read Geographical Information in Graph Format

Description

Reads the geographical information provided in a file in graph format and stores it in a map object.

Usage

```
read.gra(file, sorted = FALSE, sep = " ")
```

Arguments

file	the file path of the graph file to be read.
sorted	logical. Should the regions be ordered by the numbers specifying the region names (sorted = TRUE)?
sep	the field separator character. Values on each line of the file are separated by this character.

Details

A graph file stores the nodes and the edges of a graph and is a convenient way to represent the neighborhood structure of a geographical map. The structure of a graph file is given by:

- The first line of the graph file specifies the total number of nodes.
- The subsequent three lines correspond to the node with the name given in line 2, the number of neighbors in line 3 and the neighboring node identity numbers in line 4.

Note that the node identity numbering starts with 0. Example taken from the package example file `Germany.gra`:

```
309
1001
```

```

1
1
1059
3
0 3 4
1002
2
5 4
1051
3
4 1 9
1058
7
2 6 3 5 1 10 9
...

```

Hence, this graph file contains of 309 regions. The first region with name 1001 has 1 neighbor with neighboring node identity number 1. The last region in this example, region 1058, has 7 neighbors with neighboring node identity numbers 2 6 3 5 1 10 9.

In addition, graph files using the following format may be imported:

- The first line of the graph file specifies the total number of nodes.
- The subsequent lines start with the node name followed by the number of neighbors and the neighboring node identity numbers.

Example:

```

309
1001 1 2
1059 3 1 4 5
1002 2 6 5
1051 3 5 2 10
1058 7 3 7 4 6 2 11 10
...

```

Value

Returns an adjacency matrix that represents the neighborhood structure defined in the graph file. The diagonal elements of this matrix are the number of neighbors of each region. The off-diagonal elements are either -1 if regions are neighbors else 0.

Author(s)

Thomas Kneib, Felix Heinzl, rewritten by Nikolaus Umlauf.

References

BayesX Reference Manual, Chapter 5. Available at <https://www.uni-goettingen.de/de/bayesx/550513.html>.

See Also

[write.gra](#), [read.bnd](#), [write.bnd](#), [get.neighbor](#), [add.neighbor](#), [delete.neighbor](#).

Examples

```
file <- file.path(find.package("R2BayesX"), "examples", "Germany.gra")
germany <- read.gra(file)
```

samples

Extract Samples of Coefficients and Variances

Description

Function to extract the samples generated with Markov chain Monte Carlo simulation.

Usage

```
samples(object, model = NULL, term = NULL, coda = TRUE, acf = FALSE, ...)
```

Arguments

object	an object of class "bayesx".
model	for which model the samples should be provided, either an integer or a character, e.g. model = "mcmc.model".
term	character or integer , the term for which samples should be extracted. Also samples of linear effects may be returned if available and term = "linear-samples", or of the variance if term = "var-samples". If set to NULL, the samples of the linear effects will be returned.
acf	if set to TRUE, the autocorrelation function of the samples will be provided.
coda	if set to TRUE the function will return objects of class "mcmc" or "mcmc.list" as provided in the coda package.
...	further arguments passed to function acf , e.g. argument lag.max if acf = TRUE.

Value

A data.frame or an object of class "mcmc" or "mcmc.list", if argument coda = TRUE.

Author(s)

Nikolaus Umlauf, Thomas Kneib, Stefan Lang, Achim Zeileis.

See Also

[bayesx](#).

Examples

```
## Not run:
## generate some data
set.seed(111)
n <- 200

## regressor
dat <- data.frame(x = runif(n, -3, 3))

## response
dat$y <- with(dat, 1.5 + sin(x) + rnorm(n, sd = 0.6))

## estimate model
b <- bayesx(y ~ sx(x), data = dat)

## extract samples for the P-spline
sax <- samples(b, term = "sx(x)")
colnames(sax)

## plotting
plot(sax)

## linear effects samples
samples(b, term = "linear-samples")

## for acf, increase lag
sax <- samples(b, term = c("linear-samples", "var-samples", "sx(x)"),
  acf = TRUE, lag.max = 200, coda = FALSE)
names(sax)
head(sax)

## plot maximum autocorrelation
## of all parameters
sax <- samples(b, term = c("linear-samples", "var-samples", "sx(x)"),
  acf = TRUE, lag.max = 50, coda = FALSE)
names(sax)
matplot(y = apply(sax, 1, max), type = "h",
  ylab = "ACF", xlab = "lag")

## example using multiple chains
b <- bayesx(y ~ sx(x), data = dat, chains = 3)
sax <- samples(b, term = "sx(x)")
plot(sax)

## End(Not run)
```

Description

Converts the geographical information provided in a shape-file into a boundary object (see Chapter 5 of the **BayesX** Reference Manual)

Usage

```
shp2bnd(shpname, regionnames, check.is.in = TRUE)
```

Arguments

shpname	base filename of the shape-file (including path)
regionnames	either a vector of region names or the name of the variable in the dbf-file representing these names
check.is.in	test whether some regions are surrounded by other regions (FALSE speeds up the execution time but may result in a corrupted bnd-file)

Value

Returns a boundary object, i.e. a list of polygons that form the map. See [read.bnd](#) for more information on the format.

Author(s)

Felix Heinzl, Daniel Sabanes Bove, Thomas Kneib with contributions by Michael Hoehle and Frank Sagerer.

References

BayesX Reference Manual. Available at <https://www.uni-goettingen.de/de/bayesx/550513.html>.

See Also

[write.bnd](#), [read.bnd](#), [plotmap](#).

Examples

```
## read shapefile into bnd object
shpname <- file.path(find.package("R2BayesX"), "examples", "Northamerica")
north <- shp2bnd(shpname = shpname, regionnames = "COUNTRY")

## draw the map
plotmap(north)
```

Description

This function plots slices from user defined values of bivariate surfaces.

Usage

```
sliceplot(x, y = NULL, z = NULL, view = 1, c.select = NULL,
  values = NULL, probs = c(0.1, 0.5, 0.9), grid = 100,
  legend = TRUE, pos = "topright", digits = 2, data = NULL,
  rawdata = FALSE, type = "interp", linear = FALSE,
  extrap = FALSE, k = 40, rug = TRUE, rug.col = NULL,
  jitter = TRUE, ...)
```

Arguments

x	a matrix or data frame, containing the covariates for which the effect should be plotted in the first and second column and at least a third column containing the effect, typically the structure for bivariate functions returned within bayesx and read.bayesx.output model term objects is used, also see fitted.bayesx . Another possibility is to specify the plot via a formula, e.g. for simple plotting of bivariate surfaces $z \sim x + y$, also see the example.
y	if x is a vector the argument y and z must also be supplied as vectors.
z	if x is a vector the argument y and z must also be supplied as vectors, z defines the surface given by $z = f(x, y)$.
view	which variable should be used for the x-axis of the plot, the other variable will be used to compute the slices. May also be a character with the name of the corresponding variable.
c.select	integer, selects the column that is used in the resulting matrix to be used as the z argument.
values	the values of the x or y variable that should be used for computing the slices, if set to NULL, slices will be constructed according to the quantiles, see also argument probs.
probs	numeric vector of probabilities with values in [0,1] to be used within function quantile to compute the values for plotting the slices.
grid	the grid size of the surface where the slices are generated from.
legend	if set to TRUE, a legend with the values that were used for slicing will be added.
pos	the position of the legend, see also function legend .
digits	the decimal place the legend values should be rounded.
data	if x is a formula, a data.frame or list. By default the variables are taken from <code>environment(x)</code> : typically the environment from which <code>plot3d</code> is called.

rawdata	if set to TRUE, the data will not be interpolated, only raw data will be used. This is useful when displaying data on a regular grid.
type	character. Which type of interpolation method should be used. The default is type = "interp", see function interp . The two other options are type = "mba", which calls function mba.surf of package MBA , or type = "mgcv", which uses a spatial smoother withing package mgcv for interpolation. The last option is definitely the slowest, since a full regression model needs to be estimated.
linear	logical. Should linear interpolation be used withing function interp ?
extrap	logical. Should interpolations be computed outside the observation area (i.e., extrapolated)?
k	integer. The number of basis functions to be used to compute the interpolated surface when type = "mgcv".
rug	add a rug to the plot.
jitter	if set to TRUE a jittered rug plot is added.
rug.col	specify the color of the rug representation.
...	parameters passed to matplot and legend .

Details

Similar to function [plot3d](#), this function first applies bivariate interpolation on a regular grid, afterwards the slices are computed from the resulting surface.

Author(s)

Nikolaus Umlauf, Thomas Kneib, Stefan Lang, Achim Zeileis.

See Also

[plot.bayesx](#), [bayesx](#), [read.bayesx.output](#), [fitted.bayesx](#), [plot3d](#).

Examples

```
## generate some data
set.seed(111)
n <- 500

## regressors
dat <- data.frame(z = runif(n, -3, 3), w = runif(n, 0, 6))

## response
dat$y <- with(dat, 1.5 + cos(z) * sin(w) + rnorm(n, sd = 0.6))

## Not run:
## estimate model
b <- bayesx(y ~ sx(z, w, bs = "te", knots = 5), data = dat, method = "REML")
summary(b)
```

```

## plot estimated effect
plot(b, term = "sx(z,w)", sliceplot = TRUE)
plot(b, term = "sx(z,w)", sliceplot = TRUE, view = 2)
plot(b, term = "sx(z,w)", sliceplot = TRUE, view = "w")
plot(b, term = "sx(z,w)", sliceplot = TRUE, c.select = 4)
plot(b, term = "sx(z,w)", sliceplot = TRUE, c.select = 6)
plot(b, term = "sx(z,w)", sliceplot = TRUE, probs = seq(0, 1, length = 10))

## End(Not run)

## another variation
dat$f1 <- with(dat, sin(z) * cos(w))
sliceplot(cbind(z = dat$z, w = dat$w, f1 = dat$f1))

## same with formula
sliceplot(sin(z) * cos(w) ~ z + w, ylab = "f(z)", data = dat)

## compare with plot3d()
plot3d(sin(z) * 1.5 * w ~ z + w, zlab = "f(z,w)", data = dat)
sliceplot(sin(z) * 1.5 * w ~ z + w, ylab = "f(z)", data = dat)
sliceplot(sin(z) * 1.5 * w ~ z + w, view = 2, ylab = "f(z)", data = dat)

```

summary.bayesx

Bayesx Summary Statistics

Description

Takes an object of class "bayesx" and displays summary statistics.

Usage

```

## S3 method for class 'bayesx'
summary(object, model = NULL,
        digits = max(3, getOption("digits") - 3), ...)

```

Arguments

object	an object of class "bayesx".
model	for which model the plot should be provided, either an integer or a character, e.g. model = "mcmc.model".
digits	choose the decimal places of represented numbers in the summary statistics.
...	not used.

Details

This function supplies detailed summary statistics of estimated objects with **BayesX**, i.e. informations on smoothing parameters or variances are supplied, as well as random effects variances and parametric coefficients. Depending on the model estimated and the output provided, additional

model specific information will be printed, e.g. if method = "MCMC" was specified in `bayesx`, the number of iterations, the burnin and so forth is shown. Also goodness of fit statistics are provided if the object contains such informations.

Author(s)

Nikolaus Umlauf, Thomas Kneib, Stefan Lang, Achim Zeileis.

See Also

`bayesx`, `read.bayesx.output`.

Examples

```
## Not run:
## generate some data
set.seed(111)
n <- 500

## regressors
dat <- data.frame(x = runif(n, -3, 3), z = runif(n, -3, 3),
  w = runif(n, 0, 6), fac = factor(rep(1:10, n/10)))

## response
dat$y <- with(dat, 1.5 + sin(x) + cos(z) * sin(w) +
  c(2.67, 5, 6, 3, 4, 2, 6, 7, 9, 7.5)[fac] + rnorm(n, sd = 0.6))

## estimate model
b <- bayesx(y ~ sx(x) + sx(z, w, bs = "te") + fac,
  data = dat, method = "MCMC")

## now show summary statistics
summary(b)

## End(Not run)
```

sx

Construct BayesX Model Terms in A Formula

Description

Function `sx` is a model term constructor function for terms used within the formula argument of function `bayesx`. The function does not evaluate matrices etc., the behavior is similar to function `s` from package `mgcv`. It purely exists to build a basic setup for the model term which can be processed by function `bayesx.construct`.

Usage

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

Arguments

x	the covariate the model term is a function of.
z	a second covariate.
bs	a character string, specifying the basis/type which is used for this model term.
by	a numeric or factor variable of the same dimension as each covariate. In the numeric vector case the elements multiply the smooth, evaluated at the corresponding covariate values (a ‘varying coefficient model’ results). In the factor case the term is replicated for each factor level. Note that centering of the term may be needed, please see the notes.
...	special controlling arguments or objects used for the model term, see also the examples and function bayesx.term.options for all possible optional parameters.

Details

The following term types may be specified using argument bs:

- "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", "geokriling": Geokriling 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 $\log(\lambda(\text{time}))$.

- "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 $\gamma_j, j = 1, \dots, q, q \geq 1$ of the linear effects x_1, \dots, 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.

Value

A list of class "xx.smooth.spec", where "xx" is a basis/type identifying code given by the bs argument of f.

Note

Some care has to be taken with the identifiability of varying coefficients terms. The standard in **BayesX** is to center nonlinear main effects terms around zero whereas varying coefficient terms are not centered. This makes sense since main effects nonlinear terms are not identifiable and varying coefficients terms are usually identifiable. However, there are situations where a varying coefficients term is not identifiable. Then the term must be centered. Since centering is not automatically accomplished it has to be enforced by the user by adding option center = TRUE in function f. To give an example, the varying coefficient terms in $\eta = \dots + g_1(z_1)z + g_2(z_2)z + \gamma_0 + \gamma_1z + \dots$ are not identified, whereas in $\eta = \dots + g_1(z_1)z + \gamma_0 + \dots$, the varying coefficient term is identifiable. In the first case, centering is necessary, in the second case, it is not.

Author(s)

Nikolaus Umlauf, Thomas Kneib, Stefan Lang, Achim Zeileis.

See Also

[bayesx](#), [bayesx.term.options](#), [s](#), [bayesx.construct](#).

Examples

```
## funktion sx() returns a list
## which is then processed by function
## bayesx.construct to build the
## BayesX model term structure
sx(x)

bayesx.construct(sx(x))
bayesx.construct(sx(x, bs = "rw1"))
bayesx.construct(sx(x, bs = "factor"))
bayesx.construct(sx(x, bs = "offset"))
bayesx.construct(sx(x, z, bs = "te"))

## varying coefficients
bayesx.construct(sx(x1, by = x2))
bayesx.construct(sx(x1, by = x2, center = TRUE))
```

```
## using a map for markov random fields
data("FantasyBnd")
plot(FantasyBnd)
bayesx.construct(sx(id, bs = "mrf", map = FantasyBnd))

## random effects
bayesx.construct(sx(id, bs = "re"))

## examples using optional controlling
## parameters and objects
bayesx.construct(sx(x, bs = "ps", knots = 20))
bayesx.construct(sx(x, bs = "ps", nrknots = 20))
bayesx.construct(sx(x, bs = "ps", knots = 20, nocenter = TRUE))

## use of bs with original
## BayesX syntax
bayesx.construct(sx(x, bs = "psplinerw1"))
bayesx.construct(sx(x, bs = "psplinerw2"))
bayesx.construct(sx(x, z, bs = "pspline2dimrw2"))

bayesx.construct(sx(id, bs = "spatial", map = FantasyBnd))
bayesx.construct(sx(x, z, bs = "kriging"))
bayesx.construct(sx(id, bs = "geospline", map = FantasyBnd, nrknots = 5))
bayesx.construct(sx(x, bs = "catspecific"))

## Not run:
## generate some data
set.seed(111)
n <- 200

## regressor
dat <- data.frame(x = runif(n, -3, 3))

## response
dat$y <- with(dat, 1.5 + sin(x) + rnorm(n, sd = 0.6))

## estimate models with
## bayesx REML and MCMC
b1 <- bayesx(y ~ sx(x), method = "REML", data = dat)

## increase inner knots
## decrease degree of the P-spline
b2 <- bayesx(y ~ sx(x, knots = 30, degree = 2), method = "REML", data = dat)

## compare reported output
summary(c(b1, b2))

## plot the effect for both models
plot(c(b1, b2), residuals = TRUE)
```

```

## more examples
set.seed(111)
n <- 500

## regressors
dat <- data.frame(x = runif(n, -3, 3), z = runif(n, -3, 3),
  w = runif(n, 0, 6), fac = factor(rep(1:10, n/10)))

## response
dat$y <- with(dat, 1.5 + sin(x) + cos(z) * sin(w) +
  c(2.67, 5, 6, 3, 4, 2, 6, 7, 9, 7.5)[fac] + rnorm(n, sd = 0.6))

## estimate model
b <- bayesx(y ~ sx(x) + sx(z, w, bs = "te") + fac,
  data = dat, method = "MCMC")

summary(b)
plot(b)

## now a mrf example
## note: the regional identification
## covariate and the map regionnames
## should be coded as integer
set.seed(333)

## simulate some geographical data
data("MunichBnd")
N <- length(MunichBnd); n <- N*5
names(MunichBnd) <- 1:N

## regressors
dat <- data.frame(x1 = runif(n, -3, 3),
  id = as.factor(rep(names(MunichBnd), length.out = n)))
dat$sp <- with(dat, sort(runif(N, -2, 2), decreasing = TRUE)[id])

## response
dat$y <- with(dat, 1.5 + sin(x1) + sp + rnorm(n, sd = 1.2))

## estimate models with
## bayesx MCMC and REML
b <- bayesx(y ~ sx(x1) + sx(id, bs = "mrf", map = MunichBnd),
  method = "REML", data = dat)

## summary statistics
summary(b)

## plot the effects
op <- par(no.readonly = TRUE)
par(mfrow = c(1,2))
plot(b, term = "sx(id)", map = MunichBnd,
  main = "bayesx() estimate")

```

```

plotmap(MunichBnd, x = dat$sp, id = dat$id,
  main = "Truth")
par(op)

## model with random effects
set.seed(333)
N <- 30
n <- N*10

## regressors
dat <- data.frame(id = sort(rep(1:N, n/N)), x1 = runif(n, -3, 3))
dat$re <- with(dat, rnorm(N, sd = 0.6)[id])

## response
dat$y <- with(dat, 1.5 + sin(x1) + re + rnorm(n, sd = 0.6))

## estimate model
b <- bayesx(y ~ sx(x1, bs = "psplinerw1") + sx(id, bs = "re"), data = dat)
summary(b)
plot(b)

## extract estimated random effects
## and compare with true effects
plot(fitted(b, term = "sx(id)")$Mean ~ unique(dat$re))

## End(Not run)

```

term.freqs

Extract model term selection frequencies.

Description

This function takes a fitted [bayesx](#) object and returns selection frequency tables of model terms. These tables are only returned using the stepwise procedure in combination with the bootstrap confidence intervals, see function [bayesx.control](#).

Usage

```
term.freqs(object, model = NULL, term = NULL, ...)
```

Arguments

object	an object of class "bayesx".
model	for which model the tables should be provided, either an integer or a character, e.g. model = "mcmc.model".
term	character or integer. The term for which the frequency table should be extracted.
...	not used.

Author(s)

Nikolaus Umlauf, Thomas Kneib, Stefan Lang, Achim Zeileis.

See Also

[bayesx](#), [bayesx.control](#).

Examples

```
## Not run:
## generate some data
set.seed(111)
n <- 500

## regressors
dat <- data.frame(x = runif(n, -3, 3), z = runif(n, -1, 1),
  w = runif(n, 0, 1), fac = factor(rep(1:10, n/10)))

## response
dat$y <- with(dat, 1.5 + sin(x) + rnorm(n, sd = 0.6))

## estimate model
b <- bayesx(y ~ sx(x) + sx(z) + sx(w) + sx(fac, bs = "re"),
  method = "STEP", CI = "MCMCbootstrap", bootstrapsamples = 99,
  data = dat)
summary(b)

## extract frequency tables
term.freqs(b)

## End(Not run)
```

write.bayesx.input *Write the BayesX Program*

Description

Function `write.bayesx.input` takes an object from [parse.bayesx.input](#) and translates the input to an executable program file which may be send to the **BayesX** binary.

Usage

```
write.bayesx.input(object)
```

Arguments

`object` An object of class "bayesx.input", see [parse.bayesx.input](#)

Details

This function translates the model specified in the formula within `parse.bayesx.input` or `bayesx` into a **BayesX** executable program file, secondly the function writes a data file into the specified directory chosen in `bayesx.control`, `parse.bayesx.input` or `bayesx`, where **BayesX** will find the necessary variables for estimation.

Value

Function returns a list containing a character string with all commands used within the executable of **BayesX**, the program name, model name and the file directory where the program file is stored.

Author(s)

Nikolaus Umlauf, Thomas Kneib, Stefan Lang, Achim Zeileis.

Examples

```
## generate some data
set.seed(111)
n <- 500

## regressors
dat <- data.frame(x = runif(n, -3, 3), z = runif(n, -3, 3),
  w = runif(n, 0, 6), fac = factor(rep(1:10, n/10)))

## response
dat$y <- with(dat, 1.5 + sin(x) + cos(z) * sin(w) +
  c(2.67, 5, 6, 3, 4, 2, 6, 7, 9, 7.5)[fac] + rnorm(n, sd = 0.6))

## create BayesX .prg
pars <- parse.bayesx.input(y ~ sx(x) + sx(z, w, bs = "te") + fac,
  data = dat)
prg <- write.bayesx.input(pars)
print(prg)

## have a look at the generated files
## which are used within BayesX
print(list.files(paste(tempdir(), "/bayesx", sep = "")))
```

write.bnd

Saving Maps in Boundary Format

Description

Writes the information of a map object to a file (in boundary format)

Usage

```
write.bnd(map, file, replace = FALSE)
```

Arguments

map	map object to be saved (should be in boundary format).
file	name of the file to write to
replace	should an existing file be overwritten with the new version?

Author(s)

Thomas Kneib, Felix Heinzl.

References

BayesX Reference Manual. Available at <https://www.uni-goettingen.de/de/bayesx/550513.html>.

See Also

[read.bnd](#), [write.gra](#), [read.gra](#).

Examples

```
data("FantasyBnd")
tfile <- tempfile()
write.bnd(FantasyBnd, file = tfile)
cat(readLines(tfile), sep = "\n")
unlink(tfile)
```

write.gra

Saving Maps in Graph Format

Description

Writes the information of a map object to a file (in graph format).

Usage

```
write.gra(map, file, replace = FALSE)
```

Arguments

map	map object to be saved (should be in graph format, see bnd2gra for the conversion of boundary format to graph format).
file	name of the file to write to
replace	should an existing file be overwritten with the new version?

Author(s)

Thomas Kneib, Felix Heinzl.

References

BayesX Reference Manual. Available at <https://www.uni-goettingen.de/de/bayesx/550513.html>.

See Also

[read.gra](#), [read.bnd](#), [write.bnd](#).

Examples

```
data("FantasyBnd")
tfile <- tempfile()
write.gra(bnd2gra(FantasyBnd), file = tfile)
cat(readLines(tfile), sep = "\n")
unlink(tfile)
```

ZambiaBnd

Zambia Map

Description

This database produces a map of Zambia containing 57 districts.

Usage

```
data("ZambiaBnd")
```

Format

A [list](#) of class "bnd" containing 57 polygon matrices with x-coordinates in the first and y-coordinates in the second column each.

Source

<https://www.uni-goettingen.de/de/bayesx/550513.html>.

See Also

[plotmap](#), [read.bnd](#), [write.bnd](#)

Examples

```
## load ZambiaBnd and plot it
data("ZambiaBnd")
plotmap(ZambiaBnd)
```

Description

The Demographic Health Surveys (DHS) of Zambia was conducted 1992. The survey is produced jointly by Macro International, a USAIDfunded firm specializing in demographic research, and the national statistical agency of the country.

Malnutrition among children is usually determined by assessing an anthropometric status of the children relative to a reference standard. In our example, malnutrition is measured by stunting or insufficient height for age, indicating chronic malnutrition. Stunting for a child i is determined using a Z -score defined as

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

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

The main interest is on modeling the dependence of malnutrition on covariates including the age of the child, the body mass index of the child's mother, the district the child lives in and some further categorical covariates.

Usage

```
data("ZambiaNutrition")
```

Format

A data frame containing 4847 observations on 8 variables.

stunting: standardised Z -score for stunting.

mbmi: body mass index of the mother.

agechild: age of the child in months.

district: district where the mother lives.

memloyment: mother's employment status with categories 'working' and 'not working'.

meducation: mother's educational status with categories for complete primary but incomplete secondary 'no/incomplete', complete secondary or higher 'minimum primary' and no education or incomplete primary 'minimum secondary'.

urban: locality of the domicile with categories 'yes' and 'no'.

gender: gender of the child with categories 'male' and 'female'.

Source

<https://www.uni-goettingen.de/de/bayesx/550513.html>.

References

Kandala, N. B., 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.

See Also

[bayesx](#)

Examples

```
## Not run:
## load zambia data and map
data("ZambiaNutrition")
data("ZambiaBnd")

## estimate model
zm <- bayesx(stunting ~ memployment + meducation + urban + gender +
  sx(mbmi) + sx(agechild) + sx(district, bs = "mrf", map = ZambiaBnd) +
  sx(district, bs = "re"), iter = 12000, burnin = 2000, step = 10,
  data = ZambiaNutrition)

summary(zm)

## plot smooth effects
plot(zm, term = c("sx(bmi)", "sx(agechild)", "sx(district)"), map = ZambiaBnd)

## for more examples
demo("zambia")

## End(Not run)
```

Index

* datasets

- BeechBnd, 25
- BeechGra, 26
- FantasyBnd, 34
- ForestHealth, 37
- GAMart, 39
- GermanyBnd, 41
- MunichBnd, 48
- ZambiaBnd, 91
- ZambiaNutrition, 92

* hplot

- plot.bayesx, 49
- plot2d, 53
- plot3d, 56
- plotblock, 60
- plotmap, 62
- plotsamples, 67
- sliceplot, 79

* package

- R2BayesX-package, 3

* regression

- bayesx, 4
- bayesx.construct, 12
- bayesx.control, 16
- bayesx.term.options, 20
- bayesx_logfile, 22
- bayesx_prgfile, 23
- bayesx_runtime, 24
- colorlegend, 27
- cprob, 31
- DIC, 33
- fitted.bayesx, 35
- GCV, 40
- getscript, 42
- GRstats, 44
- parse.bayesx.input, 48
- predict.bayesx, 69
- R2BayesX-package, 3
- read.bayesx.output, 71

- samples, 76
- summary.bayesx, 81
- sx, 82
- term.freqs, 87
- write.bayesx.input, 88

* spatial

- add.neighbor, 3
- bnd2gra, 26
- delete.neighbor, 32
- get.neighbor, 42
- Interface between nb and gra
 - format, 45
- Interface between sp and bnd
 - format, 46
- read.bnd, 72
- read.gra, 74
- shp2bnd, 77
- write.bnd, 89
- write.gra, 90

- acf, 50, 68, 76

- add.neighbor, 3, 33, 42, 76

- AIC, 3

- attr, 60

- attributes, 35

- bayesx, 3, 4, 12, 13, 16, 20, 22–24, 31, 33, 35, 38–40, 42–44, 49, 50, 53–56, 58, 60, 61, 63, 67–70, 76, 79, 80, 82, 84, 87–89, 93

- bayesx.construct, 6, 7, 12, 82, 84

- bayesx.control, 3, 5, 6, 16, 49, 87–89

- bayesx.term.options, 6, 7, 14, 20, 83, 84

- bayesx_logfile, 22

- bayesx_prgfile, 23

- bayesx_runtime, 24

- BeechBnd, 25

- BeechGra, 26

- BIC, 3

- bnd2gra, 4, 26, 26, 33, 90

- bnd2sp, [45](#)
- bnd2sp(Interface between sp and bnd format), [46](#)
- bxopts (bayesx.term.options), [20](#)
- character, [76](#), [83](#)
- colorlegend, [3](#), [27](#), [58](#), [64](#)
- contour, [57](#), [58](#)
- cprob, [31](#)
- data.frame, [5](#), [35](#), [49](#)
- delete.neighbor, [4](#), [32](#), [42](#), [76](#)
- density, [50](#)
- Devices, [43](#)
- DIC, [33](#)
- drawmap (plotmap), [62](#)
- factor, [7](#), [83](#)
- FantasyBnd, [34](#)
- fitted, [6](#)
- fitted.bayesx, [3](#), [6](#), [7](#), [35](#), [54–56](#), [58](#), [60](#), [61](#), [70](#), [71](#), [79](#), [80](#)
- forest (ForestHealth), [37](#)
- ForestHealth, [37](#)
- formula.gam, [4](#), [6](#), [7](#), [14](#)
- GAMart, [39](#)
- gamart (GAMart), [39](#)
- GCV, [40](#)
- gelman.diag, [44](#)
- GermanyBnd, [41](#)
- get.neighbor, [4](#), [33](#), [42](#), [76](#)
- getsript, [42](#)
- gra2nb, [47](#)
- gra2nb(Interface between nb and gra format), [45](#)
- GRstats, [5](#), [44](#)
- image.plot, [57](#), [58](#)
- integer, [7](#), [76](#)
- Interface between nb and gra format, [45](#)
- Interface between sp and bnd format, [46](#)
- interp, [57](#), [64](#), [80](#)
- jitter, [54](#), [80](#)
- legend, [79](#), [80](#)
- list, [5](#), [25](#), [34](#), [41](#), [48](#), [49](#), [63](#), [91](#)
- logfile (bayesx_logfile), [22](#)
- map.where, [64](#)
- matplot, [80](#)
- mba.surf, [57](#), [64](#), [80](#)
- mclapply, [5](#)
- mgcv, [6](#), [12](#), [82](#)
- model.frame, [5](#)
- model.matrix.default, [5](#), [49](#)
- MunichBnd, [48](#)
- na.omit, [5](#)
- nb2gra, [47](#)
- nb2gra(Interface between nb and gra format), [45](#)
- numeric, [83](#)
- options, [5](#)
- par, [29](#)
- parse.bayesx.input, [6](#), [7](#), [48](#), [71](#), [88](#), [89](#)
- pdf, [43](#)
- persp, [57](#), [58](#)
- plot, [6](#), [55](#), [61](#), [68](#)
- plot.bayesx, [3](#), [6](#), [7](#), [35](#), [49](#), [55](#), [58](#), [61](#), [64](#), [68](#), [71](#), [72](#), [80](#)
- plot.bnd (plotmap), [62](#)
- plot2d, [3](#), [50](#), [53](#)
- plot3d, [3](#), [50](#), [56](#), [80](#)
- plotblock, [3](#), [50](#), [59](#)
- plotmap, [3](#), [25](#), [34](#), [41](#), [48](#), [50](#), [62](#), [74](#), [78](#), [91](#)
- plotnonp (plot2d), [53](#)
- plotsamples, [3](#), [50](#), [67](#)
- plotsurf (plot3d), [56](#)
- polygon, [55](#), [61](#), [63](#), [64](#)
- predict.bayesx, [3](#), [69](#)
- prgfile (bayesx_prgfile), [23](#)
- print, [6](#)
- quantile, [79](#)
- R2BayesX, [6](#), [12](#), [14](#)
- R2BayesX (R2BayesX-package), [3](#)
- r2bayesx (R2BayesX-package), [3](#)
- R2BayesX-package, [3](#)
- r2bayesx-package (R2BayesX-package), [3](#)
- range, [28](#)
- read.bayesx.output, [3](#), [6](#), [7](#), [14](#), [35](#), [50](#), [53–56](#), [58](#), [60](#), [61](#), [63](#), [67](#), [68](#), [71](#), [79](#), [80](#), [82](#)
- read.bnd, [13](#), [14](#), [21](#), [25](#), [27](#), [34](#), [41](#), [47](#), [48](#), [64](#), [72](#), [76](#), [78](#), [83](#), [90](#), [91](#)

read.gra, [4](#), [13](#), [14](#), [21](#), [26](#), [27](#), [33](#), [45](#), [74](#), [74](#),
[83](#), [90](#), [91](#)
read.table, [54](#), [57](#)
residuals, [6](#)
residuals.bayesx, [3](#)
residuals.bayesx (fitted.bayesx), [35](#)
rug, [54](#), [80](#)
run.bayesx, [3](#), [6](#), [7](#), [13](#), [71](#)
runtime (bayesx_runtime), [24](#)

s, [4](#), [6](#), [7](#), [12–14](#), [21](#), [82–84](#)
samples, [3](#), [44](#), [71](#), [72](#), [76](#)
set.seed, [17](#)
shp2bnd, [13](#), [21](#), [77](#), [83](#)
sliceplot, [79](#)
sp2bnd, [45](#)
sp2bnd (Interface between sp and bnd
format), [46](#)
summary, [6](#)
summary.bayesx, [3](#), [6](#), [7](#), [71](#), [72](#), [81](#)
sx, [4](#), [6](#), [7](#), [12–14](#), [20](#), [21](#), [31](#), [49](#), [82](#), [83](#)
system.time, [24](#)

te, [12](#)
term.freqs, [87](#)
text, [29](#), [57](#)

write.bayesx.input, [6](#), [7](#), [13](#), [48](#), [49](#), [88](#)
write.bnd, [25](#), [27](#), [34](#), [41](#), [47](#), [48](#), [74](#), [76](#), [78](#),
[89](#), [91](#)
write.gra, [4](#), [27](#), [33](#), [45](#), [74](#), [76](#), [90](#), [90](#)

zambia (ZambiaNutrition), [92](#)
ZambiaBnd, [91](#)
ZambiaNutrition, [92](#)