Package 'npreg'

July 21, 2022

Type Package

Title Nonparametric Regression via Smoothing Splines

Version 1.0-9

Date 2022-07-20

Author Nathaniel E. Helwig <helwig@umn.edu>

Maintainer Nathaniel E. Helwig <helwig@umn.edu>

Description Multiple and generalized nonparametric regression using smoothing spline ANOVA models and generalized additive models, as described in Helwig (2020) <doi:10.4135/9781526421036885885>. Includes support for Gaussian and non-Gaussian responses, smoothers for multiple types of predictors, interactions between smoothers of mixed types, eight different methods for smoothing parameter selection, and flexible tools for prediction and inference.

Suggests parallel, SuppDists

License GPL (>= 2)

NeedsCompilation no

Repository CRAN

Date/Publication 2022-07-20 23:00:05 UTC

R topics documented:

`bin.sample` *Bin Sample a Vector, Matrix, or Data Frame*

Description

Bin elements of a vector (or rows of a matrix/data frame) and randomly sample a specified number of elements from each bin. Returns sampled data and (optionally) indices of sampled data and/or breaks for defining bins.

Usage

```
bin.sample(x, nbin = 5, size = 1, equidistant = FALSE,
           index.return = FALSE, breaks.return = FALSE)
```

Arguments

`x`	Vector, matrix, or data frame to bin sample. Factors are allowed.
`nbin`	Number of bins for each variable (defaults to 5 bins for each dimension of `x`). If `length(bins) != ncol(x)`, then `nbin[1]` is used for each variable.
`size`	Size of sample to randomly draw from each bin (defaults to 1).

equidistant	Should bins be defined equidistantly for each predictor? If FALSE (default), sample quantiles define bins for each predictor. If length(equidistant) != ncol(x), then equidistant[1] is used for each variable.
index.return	If TRUE, returns the (row) indices of the bin sampled observations.
breaks.return	If TRUE, returns the (lower bounds of the) breaks for the binning.

Details

For a single variable, the unidimensional bins are defined using the .bincode function. For multiple variables, the multidimensional bins are defined using the algorithm described in the appendix of Helwig et al. (2015), which combines the unidimensional bins (calculated via .bincode) into a multidimensional bin code.

Value

If index.return = FALSE and breaks.return = FALSE, returns the bin sampled x observations.

If index.return = TRUE and/or breaks.return = TRUE, returns a list with elements:

x	bin sampled x observations.
ix	row indices of bin sampled observations (if index.return = TRUE).
bx	lower bounds of breaks defining bins (if breaks.return = TRUE).

Note

For factors, the number of bins is automatically defined to be the number of levels.

Author(s)

Nathaniel E. Helwig <helwig@umn.edu>

References

Helwig, N. E., Gao, Y., Wang, S., & Ma, P. (2015). Analyzing spatiotemporal trends in social media data via smoothing spline analysis of variance. *Spatial Statistics, 14*(C), 491-504. doi:10.1016/j.spasta.2015.09.002

See Also

.bincode for binning a numeric vector

Examples

```
##########   EXAMPLE 1   ##########
### unidimensional binning

# generate data
x <- seq(0, 1, length.out = 101)

# bin sample (default)
```

```
set.seed(1)
bin.sample(x)

# bin sample (return indices)
set.seed(1)
xs <- bin.sample(x, index.return = TRUE)
xs$x             # sampled data
x[xs$ix]         # indexing sampled data

# bin sample (return indices and breaks)
set.seed(1)
xs <- bin.sample(x, index.return = TRUE, breaks.return = TRUE)
xs$x             # sampled data
x[xs$ix]         # indexing sampled data
xs$bx            # breaks



##########   EXAMPLE 2   ##########
### bidimensional binning

# generate data
x <- expand.grid(x1 = seq(0, 1, length.out = 101),
                 x2 = seq(0, 1, length.out = 101))

# bin sample (default)
set.seed(1)
bin.sample(x)

# bin sample (return indices)
set.seed(1)
xs <- bin.sample(x, index.return = TRUE)
xs$x             # sampled data
x[xs$ix,]        # indexing sampled data

# bin sample (return indices and breaks)
set.seed(1)
xs <- bin.sample(x, index.return = TRUE, breaks.return = TRUE)
xs$x             # sampled data
x[xs$ix,]        # indexing sampled data
xs$bx            # breaks

# plot breaks and 25 bins
plot(xs$bx, xlim = c(0, 1), ylim = c(0, 1),
     xlab = "x1", ylab = "x2", main = "25 bidimensional bins")
grid()
text(xs$bx + 0.1, labels = 1:25)
```

---

boot *Bootstrap a Fit Smooth*

Description

Bootstraps a fit nonparametric regression model to form confidence intervals (BCa or percentile) and standard error estimates.

Usage

```
## S3 method for class 'ss'
boot(object, statistic, ..., R = 9999, level = 0.95, bca = TRUE,
     method = c("cases", "resid", "param"), fix.lambda = TRUE, cov.mat = FALSE,
     boot.dist = FALSE, verbose = TRUE, parallel = FALSE, cl = NULL)

## S3 method for class 'sm'
boot(object, statistic, ..., R = 9999, level = 0.95, bca = TRUE,
     method = c("cases", "resid", "param"), fix.lambda = TRUE,
     fix.thetas = TRUE, cov.mat = FALSE, boot.dist = FALSE,
     verbose = TRUE, parallel = FALSE, cl = NULL)

## S3 method for class 'gsm'
boot(object, statistic, ..., R = 9999, level = 0.95, bca = TRUE,
     method = c("cases", "resid", "param"), fix.lambda = TRUE,
     fix.thetas = TRUE, cov.mat = FALSE, boot.dist = FALSE,
     verbose = TRUE, parallel = FALSE, cl = NULL)
```

Arguments

`object`	a fit from `ss` (smoothing spline), `sm` (smooth model), or `gsm` (generalized smooth model)
`statistic`	a function to compute the statistic (see Details)
`...`	additional arguments to `statistic` function (optional)
`R`	number of bootstrap resamples used to form bootstrap distribution
`level`	confidence level for bootstrap confidence intervals
`bca`	logical indicating whether to calculate BCa (default) or percentile intervals
`method`	resampling method used to form bootstrap distribution
`fix.lambda`	logical indicating whether the smoothing parameter should be fixed (default) or re-estimated for each bootstrap sample
`fix.thetas`	logical indicating whether the "extra" smoothing parameters should be fixed (default) or re-estimated for each bootstrap sample. Only applicable to `sm` and `gsm` objects with multiple penalized terms.
`cov.mat`	logical indicating whether the bootstrap estimate of the covariance matrix should be returned
`boot.dist`	logical indicating whether the bootstrap distribution should be returned
`verbose`	logical indicating whether the bootstrap progress bar should be printed
`parallel`	logical indicating if the `parallel` package should be used for parallel computing (of the bootstrap distribution). Defaults to FALSE, which implements sequential computing.

cl	cluster for parallel computing, which is used when parallel = TRUE. Note that if parallel = TRUE and cl = NULL, then the cluster is defined as makeCluster(detectCores()).

Details

The statistic function must satisfy the following two requirements:

(1) the first input must be the object of class ss, sm, or gsm

(2) the output must be a scalar or vector calculated from the object

In most applications, the statistic function will be the model predictions at some user-specified newdata, which can be passed to statistic using the ... argument.

If statistic is not provided, then the function is internally defined to be the model predictions at an equidistance sequence (for ss objects) or the training data predictor scores (for sm and gsm objects).

Value

Produces an object of class 'boot.ss', 'boot.sm', or 'boot.gsm', with the following elements:

t0	Observed statistic, computed using statistic(object, ...)
se	Bootstrap estimate of the standard error
bias	Bootstrap estimate of the bias
cov	Bootstrap estimate of the covariance (if cov.mat = TRUE)
ci	Bootstrap estimate of the confidence interval
boot.dist	Bootstrap distribution of statistic (if boot.dist = TRUE)
bias.correct	Bias correction factor for BCa confidence interval.
acceleration	Acceleration parameter for BCa confidence interval.

The output list also contains the elements object, R, level, bca, method, fix.lambda, and fix.thetas, all of which are the same as the corresponding input arguments.

Note

For gsm objects, requesting method = "resid" uses a variant of the one-step technique described in Moulton and Zeger (1991), which forms the bootstrap estimates of the coefficients without refitting the model.

As a result, when bootstrapping gsm objects with method = "resid":

(1) it is necessary to set fix.lambda = TRUE and fix.thetas = TRUE

(2) any logical statistic must depend on the model coefficients, e.g., through the model predictions.

Author(s)

Nathaniel E. Helwig <helwig@umn.edu>

References

Davison, A. C., & Hinkley, D. V. (1997). *Bootstrap Methods and Their Application*. Cambridge University Press. doi:10.1017/CBO9780511802843

Efron, B., & Tibshirani, R. J. (1994). *An Introduction to the Boostrap*. Chapman & Hall/CRC. doi:10.1201/9780429246593

Moulton, L. H., & Zeger, S. L. (1991). Bootstrapping generalized linear models. *Computational Statistics & Data Analysis, 11*(1), 53-63. doi:10.1016/01679473(91)900524

See Also

ss for fitting "ss" (smoothing spline) objects

sm for fitting "sm" (smooth model) objects

gsm for fitting "gsm" (generalized smooth model) objects

Examples

```
## Not run:

##########   EXAMPLE 1   ##########
### smoothing spline

# generate data
set.seed(1)
n <- 100
x <- seq(0, 1, length.out = n)
fx <- 2 + 3 * x + sin(2 * pi * x)
y <- fx + rnorm(n, sd = 0.5)

# fit smoothing spline
ssfit <- ss(x, y, nknots = 10)

# nonparameteric bootstrap cases
set.seed(0)
boot.cases <- boot(ssfit)

# nonparameteric bootstrap residuals
set.seed(0)
boot.resid <- boot(ssfit, method = "resid")

# parameteric bootstrap residuals
set.seed(0)
boot.param <- boot(ssfit, method = "param")

# plot results
par(mfrow = c(1, 3))
plot(boot.cases, main = "Cases")
plot(boot.resid, main = "Residuals")
plot(boot.param, main = "Parametric")
```

```
##########   EXAMPLE 2   ##########
### smooth model

# generate data
set.seed(1)
n <- 100
x <- seq(0, 1, length.out = n)
fx <- 2 + 3 * x + sin(2 * pi * x)
y <- fx + rnorm(n, sd = 0.5)

# fit smoothing spline
smfit <- sm(y ~ x, knots = 10)

# define statistic (to be equivalent to boot.ss default)
newdata <- data.frame(x = seq(0, 1, length.out = 201))
statfun <- function(object, newdata) predict(object, newdata)

# nonparameteric bootstrap cases
set.seed(0)
boot.cases <- boot(smfit, statfun, newdata = newdata)

# nonparameteric bootstrap residuals
set.seed(0)
boot.resid <- boot(smfit, statfun, newdata = newdata, method = "resid")

# parameteric bootstrap residuals (R = 99 for speed)
set.seed(0)
boot.param <- boot(smfit, statfun, newdata = newdata, method = "param")

# plot results
par(mfrow = c(1, 3))
plotci(newdata$x, boot.cases$t0, ci = boot.cases$ci, main = "Cases")
plotci(newdata$x, boot.resid$t0, ci = boot.resid$ci, main = "Residuals")
plotci(newdata$x, boot.param$t0, ci = boot.param$ci, main = "Parametric")



##########   EXAMPLE 3   ##########
### generalized smooth model

# generate data
set.seed(1)
n <- 100
x <- seq(0, 1, length.out = n)
fx <- 2 + 3 * x + sin(2 * pi * x)
y <- fx + rnorm(n, sd = 0.5)

# fit smoothing spline
gsmfit <- gsm(y ~ x, knots = 10)

# define statistic (to be equivalent to boot.ss default)
newdata <- data.frame(x = seq(0, 1, length.out = 201))
```

```
statfun <- function(object, newdata) predict(object, newdata)

# nonparameteric bootstrap cases
set.seed(0)
boot.cases <- boot(gsmfit, statfun, newdata = newdata)

# nonparameteric bootstrap residuals
set.seed(0)
boot.resid <- boot(gsmfit, statfun, newdata = newdata, method = "resid")

# parameteric bootstrap residuals
set.seed(0)
boot.param <- boot(gsmfit, statfun, newdata = newdata,  method = "param")

# plot results
par(mfrow = c(1, 3))
plotci(newdata$x, boot.cases$t0, ci = boot.cases$ci, main = "Cases")
plotci(newdata$x, boot.resid$t0, ci = boot.resid$ci, main = "Residuals")
plotci(newdata$x, boot.param$t0, ci = boot.param$ci, main = "Parametric")

## End(Not run)
```

---

coef *Extract Smooth Model Coefficients*

Description

Extracts basis function coefficients from a fit smoothing spline (fit by ss), smooth model (fit by sm), or generalized smooth model (fit by gsm).

Usage

```
## S3 method for class 'gsm'
coef(object, ...)

## S3 method for class 'sm'
coef(object, ...)

## S3 method for class 'ss'
coef(object, ...)
```

Arguments

object	an object of class "gsm" output by the gsm function, "sm" output by the sm function, or "ss" output by the ss function
...	other arugments (currently ignored)

Details

For "ss" objects, the coefficient vector will be of length $m + q$ where `m` is the dimension of the null space and q is the number of knots used for the fit.

For "sm" and "gsm" objects, the coefficient vector will be of length $m + q$ if the `tprk = TRUE` (default). Otherwise the length will depend on the model formula and marginal knot placements.

Value

Coefficients extracted from the model `object`.

Author(s)

Nathaniel E. Helwig <helwig@umn.edu>

References

Chambers, J. M. and Hastie, T. J. (1992) *Statistical Models in S*. Wadsworth & Brooks/Cole.

Helwig, N. E. (2020). Multiple and Generalized Nonparametric Regression. In P. Atkinson, S. Delamont, A. Cernat, J. W. Sakshaug, & R. A. Williams (Eds.), SAGE Research Methods Foundations. doi:10.4135/9781526421036885885

See Also

`ss`, `sm`, `gsm`

`model.matrix`, `fitted.values`, `residuals`

Examples

```
# generate data
set.seed(1)
n <- 100
x <- seq(0, 1, length.out = n)
fx <- 2 + 3 * x + sin(2 * pi * x)
y <- fx + rnorm(n, sd = 0.5)

# smoothing spline
mod.ss <- ss(x, y, nknots = 10)
fit.ss <- fitted(mod.ss)
coef.ss <- coef(mod.ss)
X.ss <- model.matrix(mod.ss)
mean((fit.ss - X.ss %*% coef.ss)^2)

# smooth model
mod.sm <- sm(y ~ x, knots = 10)
fit.sm <- fitted(mod.sm)
coef.sm <- coef(mod.sm)
X.sm <- model.matrix(mod.sm)
mean((fit.sm - X.sm %*% coef.sm)^2)

# generalized smooth model (family = gaussian)
```

```
mod.gsm <- gsm(y ~ x, knots = 10)
fit.gsm <- fitted(mod.gsm)
coef.gsm <- coef(mod.gsm)
X.gsm <- model.matrix(mod.gsm)
mean((fit.gsm - X.gsm %*% coef.gsm)^2)
```

`deviance` *Smooth Model Deviance*

Description

Returns the deviance from a fit smoothing spline (fit by `ss`), smooth model (fit by `sm`), or generalized smooth model (fit by `gsm`).

Usage

```
## S3 method for class 'gsm'
deviance(object, ...)

## S3 method for class 'sm'
deviance(object, ...)

## S3 method for class 'ss'
deviance(object, ...)
```

Arguments

`object`	an object of class "gsm" output by the `gsm` function, "sm" output by the `sm` function, or "ss" output by the `ss` function
`...`	other arugments (currently ignored)

Details

For `ss` and `sm` objects, the deviance is caculated assuming iid Gaussian errors.

For `gsm` objects, the deviance is calculated by summing the squared deviance residuals, which are calculated using `family(object)$dev.resid`

Value

Deviance of the model `object`.

Author(s)

Nathaniel E. Helwig <helwig@umn.edu>

References

Chambers, J. M. and Hastie, T. J. (1992) *Statistical Models in S*. Wadsworth & Brooks/Cole.

Helwig, N. E. (2020). Multiple and Generalized Nonparametric Regression. In P. Atkinson, S. Delamont, A. Cernat, J. W. Sakshaug, & R. A. Williams (Eds.), SAGE Research Methods Foundations. doi:10.4135/9781526421036885885

See Also

ss, sm, gsm

fitted.values and residuals

Examples

```
## for 'ss' and 'sm' objects, this function is defined as
function(object, ...){
    sum(weighted.residuals(object)^2, na.rm = TRUE)
  }

## for 'gsm' objects, this function is defined as
function(object, ...){
    object$deviance
  }
```

---

diagnostic.plots *Plot Nonparametric Regression Diagnostics*

Description

Six regression diagnostic plots for a fit smoothing spline (fit by ss), smooth model (fit by sm), or generalized smooth model (fit by gsm).

Usage

```
diagnostic.plots(x, which = c(1, 2, 3, 5),
           caption = list("Residuals vs Fitted",
                          "Normal Q-Q", "Scale-Location",
                          "Cook's distance", "Residuals vs Leverage",
                          "Cook's dist vs Variance ratio"),
           panel = if (add.smooth) function(x, y, ...)
                      panel.smooth(x, y, iter = iter.smooth, ...)
                   else points,
           sub.caption = NULL, main = "",
           ask = prod(par("mfcol")) < length(which) && dev.interactive(),
         ..., id.n = 3, labels.id = names(residuals(x)), cex.id = 0.75, cex.pt = 1,
        qqline = TRUE, cook.levels = c(0.5, 1), add.smooth = getOption("add.smooth"),
          iter.smooth = if (isGlm) 0 else 3, label.pos = c(4, 2), cex.caption = 1,
         cex.oma.main = 1.25, cex.lab = 1, line.lab = 3, xlim = NULL, ylim = NULL)
```

Arguments

x	an object of class "gsm" output by the gsm function, "sm" output by the sm function, or "ss" output by the ss function
which	subset of the integers 1:6 indicating which plots to produce
caption	captions to appear above the plots
panel	panel function (panel.smooth or points?)
sub.caption	common title (for use above multiple figures)
main	title to each plot (in addition to caption)
ask	if TRUE, the user is asked before each plot
...	other parameters to be passed through to plotting functions
id.n	number of points to be labeled in each plot, starting with the most extreme
labels.id	vector of labels for extreme observations (NULL uses the observation numbers)
cex.id	magnification of point labels
cex.pt	magnification of points
qqline	logical indicating if a qqline should be added to the normal Q-Q plot
cook.levels	levels of Cook's distance at which to draw contours
add.smooth	logical indicating if a smoother should be added to most plots
iter.smooth	the number of robustness iterations, the argument iter in panel.smooth
label.pos	positioning of the labels, for the left hald and right half of the graph respectively, for plots 1-3, 5, and 6
cex.caption	controls the size of the caption
cex.oma.main	controls the size of the sub.caption only if that is above the figures (when there is more than one figure)
cex.lab	character expansion factor for axis labels
line.lab	on which margin line should the axis labels be drawn?
xlim	Limits for x-axis. If length(which) == 1, a vector of the form c(xmin, xmax). Otherwise a list the same length as which such that each list entry gives the x-axis limits for the corresponding plot.
ylim	Limits for y-axis. If length(which) == 1, a vector of the form c(ymin, ymax). Otherwise a list the same length as which such that each list entry gives the y-axis limits for the corresponding plot.

Details

This function is modeled after the plot.lm function. The structure of the arguments, as well as the internal codes, mimics the plot.lm function whenever possible. By default, only plots 1-3 and 5 are provided, but any subset of plots can be requested using the which argument.

The six plots include: (1) residuals versus fitted values, (2) normal Q-Q plot, (3) scale-location plot of $\sqrt{|residuals|}$ versus fitted values, (4) Cook's distances, (5) residuals versus leverages, and (6) Cook's distance versus variance ratio = leverage/(1-leverage).

Author(s)

Nathaniel E. Helwig <helwig@umn.edu>

References

Belsley, D. A., Kuh, E. and Welsch, R. E. (1980). Regression Diagnostics. New York: Wiley.

Cook, R. D. and Weisberg, S. (1982). Residuals and Influence in Regression. London: Chapman and Hall.

McCullagh, P. and Nelder, J. A. (1989). Generalized Linear Models. London: Chapman and Hall.

See Also

ss, sm, gsm

smooth.influence.measures and smooth.influence

Examples

```
# generate data
set.seed(1)
n <- 100
x <- seq(0, 1, length.out = n)
fx <- 2 + 3 * x + sin(2 * pi * x)
y <- fx + rnorm(n, sd = 0.5)

# smoothing spline
mod.ss <- ss(x, y, nknots = 10)
diagnostic.plots(mod.ss)

# smooth model
mod.sm <- sm(y ~ x, knots = 10)
diagnostic.plots(mod.sm)

# generalized smooth model (family = gaussian)
mod.gsm <- gsm(y ~ x, knots = 10)
diagnostic.plots(mod.gsm)
```

---

fitted *Extract Smooth Model Fitted Values*

Description

Extracts the fitted values from a fit smoothing spline (fit by ss), smooth model (fit by sm), or generalized smooth model (fit by gsm).

Usage

```
## S3 method for class 'ss'
fitted(object, ...)

## S3 method for class 'sm'
fitted(object, ...)

## S3 method for class 'gsm'
fitted(object, ...)
```

Arguments

object	an object of class "gsm" output by the gsm function, "sm" output by the sm function, or "ss" output by the ss function
...	other arugments (currently ignored)

Details

For objects of class ss, fitted values are predicted via `predict(object, object$data$x)$y`

For objects of class sm, fitted values are extracted via `object$fitted.values`

For objects of class gsm, fitted values are computed via `ginv(object$linear.predictors)` where `ginv = object$family$linkinv`

Value

Fitted values extracted (or predicted) from `object`

Author(s)

Nathaniel E. Helwig <helwig@umn.edu>

References

Chambers, J. M. and Hastie, T. J. (1992) *Statistical Models in S*. Wadsworth & Brooks/Cole.

Helwig, N. E. (2020). Multiple and Generalized Nonparametric Regression. In P. Atkinson, S. Delamont, A. Cernat, J. W. Sakshaug, & R. A. Williams (Eds.), SAGE Research Methods Foundations. doi:10.4135/9781526421036885885

See Also

ss, sm, gsm

Examples

```
# generate data
set.seed(1)
n <- 100
x <- seq(0, 1, length.out = n)
fx <- 2 + 3 * x + sin(2 * pi * x)
```

```
y <- fx + rnorm(n, sd = 0.5)

# smoothing spline
mod.ss <- ss(x, y, nknots = 10)
fit.ss <- fitted(mod.ss)

# smooth model
mod.sm <- sm(y ~ x, knots = 10)
fit.sm <- fitted(mod.sm)

# generalized smooth model (family = gaussian)
mod.gsm <- gsm(y ~ x, knots = 10)
fit.gsm <- fitted(mod.gsm)

# compare fitted values
mean((fit.ss - fit.sm)^2)
mean((fit.ss - fit.gsm)^2)
mean((fit.sm - fit.gsm)^2)
```

---

gsm *Fit a Generalized Smooth Model*

Description

Fits a generalized semi- or nonparametric regression model with the smoothing parameter selected via one of seven methods: GCV, OCV, GACV, ACV, PQL, AIC, or BIC.

Usage

```
gsm(formula, family = gaussian, data, weights, types = NULL, tprk = TRUE,
    knots = NULL, skip.iter = TRUE, spar = NULL, lambda = NULL, control = list(),
    method = c("GCV", "OCV", "GACV", "ACV", "PQL", "AIC", "BIC"),
    xrange = NULL, thetas = NULL, mf = NULL)

## S3 method for class 'gsm'
family(object, ...)
```

Arguments

	Arguments for gsm:
	Object of class "formula" (or one that can be coerced to that class): a symbolic description of the model to be fitted. Uses the same syntax as lm and glm.
family	Description of the error distribution and link function to be used in the model. This can be a character string naming a family function, a family function, or the result of a call to a family function. See the "Family Objects" section for details.

`data` Optional data frame, list or environment (or object coercible by `as.data.frame` to a data frame) containing the variables in the model. If not found in data, the variables are taken from `environment(formula)`, typically the environment from which `sm` is called.

`weights` Optional vector of weights to be used in the fitting process. If provided, weighted (penalized) likelihood estimation is used. Defaults to all 1.

`types` Named list giving the type of smooth to use for each predictor. If `NULL`, the type is inferred from the data. See "Types of Smooths" section for details.

`tprk` Logical specifying how to parameterize smooth models with multiple predictors. If `TRUE` (default), a **t**ensor **p**roduct **r**eproducing **k**ernel function is used to represent the function. If `FALSE`, a tensor product of marginal kernel functions is used to represent the function. See the "Multiple Smooths" section for details.

`knots` Spline knots for the estimation of the nonparametric effects. For models with multiple predictors, the knot specification will depend on the `tprk` input. See the "Choosing Knots" section for details

`skip.iter` Set to `FALSE` for deep tuning of the hyperparameters. Only applicable when multiple smooth terms are included. See the "Parameter Tuning" section for details.

`spar` Smoothing parameter. Typically (but not always) in the range $(0, 1]$. If specified `lambda = 256^(3*(spar-1))`.

`lambda` Computational smoothing parameter. This value is weighted by n to form the penalty coefficient (see Details). Ignored if `spar` is provided.

`control` Optional list with named components that control the optimization specs for the smoothing parameter selection routine.

Note that spar is only searched for in the interval $[lower, upper]$.

lower: lower bound for spar; defaults to 0.

upper: upper bound for spar; defaults to 1.

tol: the absolute precision (**tol**erance) used by `optimize`; defaults to 1e-8.

iterlim: the iteration limit used by `nlm`; defaults to 5000.

print.level: the print level used by `nlm`; defaults to 0 (no printing).

epsilon: relative convergence tolerance for IRPLS algorithm; defaults to 1e-8

maxit: maximum number of iterations for IRPLS algorithm; defaults to 25

epsilon.out: relative convergence tolerance for iterative NegBin update; defaults to 1e-6

maxit.out: maximum number of iterations for iterative NegBin update; defaults to 10

`method` Method for selecting the smoothing parameter. Ignored if `lambda` is provided.

`xrange` Optional named list containing the range of each predictor. If `NULL`, the ranges are calculated from the input `data`.

`thetas` Optional vector of hyperparameters to use for smoothing. If `NULL`, these are tuned using the requested `method`.

`mf` Optional model frame constructed from `formula` and `data` (and potentially `weights`).

Note: the last two arguments are not intended to be called by the typical user of this function. These arguments are included primarily for internal usage by the `boot.gsm` function.

Arguments for `family.gsm`:

`object`	an object of class "gsm"
`...`	additional arguments (currently ignored)

Details

Letting $\eta_i = \eta(x_i)$ with $x_i = (x_{i1}, \ldots, x_{ip})$, the function is represented as

$$\eta = X\beta + Z\alpha$$

where the basis functions in X span the null space (i.e., parametric effects), and Z contains the kernel function(s) of the contrast space (i.e., nonparametric effects) evaluated at all combinations of observed data points and knots. The vectors β and α contain unknown basis function coefficients.

Let $\mu_i = E(y_i)$ denote the mean of the i-th response. The unknown function is related to the mean μ_i such as

$$g(\mu_i) = \eta_i$$

where $g()$ is a known link function. Note that this implies that $\mu_i = g^{-1}(\eta_i)$ given that the link function is assumed to be invertible.

The penalized likelihood estimation problem has the form

$$-\sum_{i=1}^{n}[y_i\xi_i - b(\xi_i)] + n\lambda\alpha' Q\alpha$$

where ξ_i is the canonical parameter, $b()$ is a known function that depends on the chosen family, and Q is the penalty matrix. Note that $\xi_i = g_0(\mu_i)$ where g_0 is the canonical link function. This implies that $\xi_i = \eta_i$ when the chosen link function is canonical, i.e., when $g = g_0$.

Value

An object of class "gsm" with components:

`linear.predictors`	the linear fit on link scale. Use `fitted.gsm` to obtain the fitted values on the response scale.
`se.lp`	the standard errors of the linear predictors.
`deviance`	up to a constant, minus twice the maximized log-likelihood. Where sensible, the constant is chosen so that a saturated model has deviance zero.
`cv.crit`	the cross-validation criterion.
`nsdf`	the degrees of freedom (Df) for the null space.
`df`	the estimated degrees of freedom (Df) for the fit model.
`df.residual`	the residual degrees of freedom = `nobs - df`
`r.squared`	the squared correlation between response and fitted values.

`dispersion`	the estimated dispersion parameter.
`logLik`	the log-likelihood.
`aic`	Akaike's Information Criterion.
`bic`	Bayesian Information Criterion.
`spar`	the value of `spar` computed or given, i.e., $s = 1 + \log_{256}(\lambda)/3$
`lambda`	the value of λ corresponding to `spar`, i.e., $\lambda = 256^{3*(s-1)}$.
`penalty`	the smoothness penalty $\alpha' Q \alpha$.
`coefficients`	the basis function coefficients used for the fit model.
`cov.sqrt`	the square-root of the covariance matrix of `coefficients`. Note: `tcrossprod(cov.sqrt)` reconstructs the covariance matrix.
`specs`	a list with information used for prediction purposes: **knots** the spline knots used for each predictor. **thetas** the "extra" tuning parameters used to weight the penalties. **xrng** the ranges of the predictor variables. **xlev** the factor levels of the predictor variables (if applicable). **tprk** logical controlling the formation of tensor product smooths.
`data`	the data used to fit the model.
`types`	the type of smooth used for each predictor.
`terms`	the terms included in the fit model.
`method`	the `method` used for smoothing parameter selection. Will be `NULL` if `lambda` was provided.
`formula`	the formula specifying the fit model.
`weights`	the weights used for fitting (if applicable)
`call`	the matched call.
`family`	the input family evaluated as a function using .
`iter`	the number of iterations of IRPLS used.
`residuals`	the working (IRPLS) residuals from the fitted model.
`null.deviance`	the deviance of the null model (i.e., intercept only).

Family Objects

Supported families and links include:

`family`	`link`
binomial	logit, probit, cauchit, log, cloglog
gaussian	identity, log, inverse
Gamma	inverse, identity, log
inverse.gaussian	1/mu^2, inverse, identity, log
poisson	log, identity, sqrt
NegBin	log, identity, sqrt

See NegBin for information about the Negative Binomial family.

Methods

The smoothing parameter can be selected using one of seven methods:
Generalized Cross-Validation (GCV)
Ordinary Cross-Validation (OCV)
Generalized Approximate Cross-Validation (GACV)
Approximate Cross-Validation (ACV)
Penalized Quasi-Likelihood (PQL)
Akaike's Information Criterion (AIC)
Bayesian Information Criterion (BIC)

Types of Smooths

The following codes specify the spline types:

par	Parametric effect (factor, integer, or numeric).
nom	Nominal smoothing spline (unordered factor).
ord	Ordinal smoothing spline (ordered factor).
lin	Linear smoothing spline (integer or numeric).
cub	Cubic smoothing spline (integer or numeric).
qui	Quintic smoothing spline (integer or numeric).
per	Periodic smoothing spline (integer or numeric).
sph	Spherical spline (matrix with $d = 2$ columns: lat, long).
tps	Thin plate spline (matrix with $d \geq 1$ columns).

For finer control of some specialized spline types:

per.lin	Linear periodic spline ($m = 1$).
per.cub	Cubic periodic spline ($m = 2$).
per.qui	Quintic periodic spline ($m = 3$).
sph.2	2nd order spherical spline ($m = 2$).
sph.3	3rd order spherical spline ($m = 3$).
sph.4	4th order spherical spline ($m = 4$).
tps.lin	Linear thin plate spline ($m = 1$).
tps.cub	Cubic thin plate spline ($m = 2$).
tps.qui	Quintic thin plate spline ($m = 3$).

For details on the spline kernel functions, see basis.nom (nominal), basis.ord (ordinal), basis.poly (polynomial), basis.sph (spherical), and basis.tps (thin plate).

Choosing Knots

If tprk = TRUE, the four options for the knots input include:

1. a scalar giving the total number of knots to sample
2. a vector of integers indexing which rows of data are the knots

3. a list with named elements giving the marginal knot values for each predictor (to be combined via `expand.grid`)
4. a list with named elements giving the knot values for each predictor (requires the same number of knots for each predictor

If `tprk = FALSE`, the three options for the `knots` input include:

1. a scalar giving the common number of knots for each continuous predictor
2. a list with named elements giving the number of marginal knots for each predictor
3. a list with named elements giving the marginal knot values for each predictor

Multiple Smooths

Suppose `formula = y ~ x1 + x2` so that the model contains additive effects of two predictor variables.

The k-th predictor's marginal effect can be denoted as

$$f_k = X_k\beta_k + Z_k\alpha_k$$

where X_k is the n by m_k null space basis function matrix, and Z_k is the n by r_k contrast space basis function matrix.

If `tprk = TRUE`, the null space basis function matrix has the form $X = [1, X_1, X_2]$ and the contrast space basis function matrix has the form

$$Z = \theta_1 Z_1 + \theta_2 Z_2$$

where the θ_k are the "extra" smoothing parameters. Note that Z is of dimension n by $r = r_1 = r_2$.

If `tprk = FALSE`, the null space basis function matrix has the form $X = [1, X_1, X_2]$, and the contrast space basis function matrix has the form

$$Z = [\theta_1 Z_1, \theta_2 Z_2]$$

where the θ_k are the "extra" smoothing parameters. Note that Z is of dimension n by $r = r_1 + r_2$.

Parameter Tuning

When multiple smooth terms are included in the model, there are smoothing (hyper)parameters that weight the contribution of each combination of smooth terms. These hyperparameters are distinct from the overall smoothing parameter `lambda` that weights the contribution of the penalty.

`skip.iter = TRUE` (default) estimates the smoothing hyperparameters using Algorithm 3.2 of Gu and Wahba (1991), which typically provides adequate results when the model form is correctly specified. The `lambda` parameter is tuned via the specified smoothing parameter selection `method`.

`skip.iter = FALSE` uses Algorithm 3.2 as an initialization, and then the `nlm` function is used to tune the hyperparameters via the specified smoothing parameter selection `method`. Setting `skip.iter = FALSE` can (substantially) increase the model fitting time, but should produce better results—particularly if the model `formula` is misspecified.

Author(s)

Nathaniel E. Helwig <helwig@umn.edu>

References

Berry, L. N., & Helwig, N. E. (2021). Cross-validation, information theory, or maximum likelihood? A comparison of tuning methods for penalized splines. *Stats, 4*(3), 701-724. doi:10.3390/stats4030042

Craven, P. and Wahba, G. (1979). Smoothing noisy data with spline functions: Estimating the correct degree of smoothing by the method of generalized cross-validation. *Numerische Mathematik, 31*, 377-403. doi:10.1007/BF01404567

Gu, C. (2013). Smoothing spline ANOVA models, 2nd edition. New York: Springer. doi:10.1007/9781461453697

Gu, C. and Wahba, G. (1991). Minimizing GCV/GML scores with multiple smoothing parameters via the Newton method. *SIAM Journal on Scientific and Statistical Computing, 12(2)*, 383-398. doi:10.1137/0912021

Helwig, N. E. (2020). Multiple and Generalized Nonparametric Regression. In P. Atkinson, S. Delamont, A. Cernat, J. W. Sakshaug, & R. A. Williams (Eds.), SAGE Research Methods Foundations. doi:10.4135/9781526421036885885

Helwig, N. E. (2021). Spectrally sparse nonparametric regression via elastic net regularized smoothers. *Journal of Computational and Graphical Statistics, 30*(1), 182-191. doi:10.1080/10618600.2020.1806855

See Also

Related Modeling Functions:

`ss` for fitting a smoothing spline with a single predictor (Gaussian response).

`sm` for fitting smooth models with multiple predictors of mixed types (Gaussian response).

S3 Methods and Related Functions for "gsm" Objects:

`boot.gsm` for bootstrapping `gsm` objects.

`coef.gsm` for extracting coefficients from `gsm` objects.

`cooks.distance.gsm` for calculating Cook's distances from `gsm` objects.

`cov.ratio` for computing covariance ratio from `gsm` objects.

`deviance.gsm` for extracting deviance from `gsm` objects.

`dfbeta.gsm` for calculating DFBETA from `gsm` objects.

`dfbetas.gsm` for calculating DFBETAS from `gsm` objects.

`diagnostic.plots` for plotting regression diagnostics from `gsm` objects.

`family.gsm` for extracting `family` from `gsm` objects.

`fitted.gsm` for extracting fitted values from `gsm` objects.

`hatvalues.gsm` for extracting leverages from `gsm` objects.

`model.matrix.gsm` for constructing model matrix from `gsm` objects.

`predict.gsm` for predicting from `gsm` objects.

`residuals.gsm` for extracting residuals from `gsm` objects.

`rstandard.gsm` for computing standardized residuals from `gsm` objects.

`rstudent.gsm` for computing studentized residuals from `gsm` objects.

smooth.influence for calculating basic influence information from gsm objects.

smooth.influence.measures for convenient display of influential observations from gsm objects.

summary.gsm for summarizing gsm objects.

vcov.gsm for extracting coefficient covariance matrix from gsm objects.

weights.gsm for extracting prior weights from gsm objects.

Examples

```
##########   EXAMPLE 1   ##########
### 1 continuous predictor

# generate data
n <- 1000
x <- seq(0, 1, length.out = n)
fx <- 3 * x + sin(2 * pi * x) - 1.5

# gaussian (default)
set.seed(1)
y <- fx + rnorm(n, sd = 1/sqrt(2))
mod <- gsm(y ~ x, knots = 10)
mean((mod$linear.predictors - fx)^2)

# compare to result from sm (they are identical)
mod.sm <- sm(y ~ x, knots = 10)
mean((mod$linear.predictors - mod.sm$fitted.values)^2)

# binomial (no weights)
set.seed(1)
y <- rbinom(n = n, size = 1, p = 1 / (1 + exp(-fx)))
mod <- gsm(y ~ x, family = binomial, knots = 10)
mean((mod$linear.predictors - fx)^2)

# binomial (w/ weights)
set.seed(1)
w <- as.integer(rep(c(10,20,30,40,50), length.out = n))
y <- rbinom(n = n, size = w, p = 1 / (1 + exp(-fx))) / w
mod <- gsm(y ~ x, family = binomial, weights = w, knots = 10)
mean((mod$linear.predictors - fx)^2)

# poisson
set.seed(1)
y <- rpois(n = n, lambda = exp(fx))
mod <- gsm(y ~ x, family = poisson, knots = 10)
mean((mod$linear.predictors - fx)^2)

# negative binomial (known theta)
set.seed(1)
y <- rnbinom(n = n, size = 1/2, mu = exp(fx))
mod <- gsm(y ~ x, family = NegBin(theta = 1/2), knots = 10)
mean((mod$linear.predictors - fx)^2)
mod$family$theta    # fixed theta
```

```
# negative binomial (unknown theta)
set.seed(1)
y <- rnbinom(n = n, size = 1/2, mu = exp(fx))
mod <- gsm(y ~ x, family = NegBin, knots = 10)
mean((mod$linear.predictors - fx)^2)
mod$family$theta    # estimated theta

# gamma
set.seed(1)
y <- rgamma(n = n, shape = 2, scale = (1 / (2 + fx)) / 2)
mod <- gsm(y ~ x, family = Gamma, knots = 10)
mean((mod$linear.predictors - fx - 2)^2)

# inverse.gaussian (not run; requires statmod)
##set.seed(1)
##y <- statmod::rinvgauss(n = n, mean = sqrt(1 / (2 + fx)), shape = 2)
##mod <- gsm(y ~ x, family = inverse.gaussian, knots = 10)
##mean((mod$linear.predictors - fx - 2)^2)



##########   EXAMPLE 2   ##########
### 1 continuous and 1 nominal predictor
### additive model

# generate data
n <- 1000
x <- seq(0, 1, length.out = n)
z <- factor(sample(letters[1:3], size = n, replace = TRUE))
fun <- function(x, z){
  mu <- c(-2, 0, 2)
  zi <- as.integer(z)
  fx <- mu[zi] + 3 * x + sin(2 * pi * x) - 1.5
}
fx <- fun(x, z)

# define marginal knots
probs <- seq(0, 0.9, by = 0.1)
knots <- list(x = quantile(x, probs = probs),
              z = letters[1:3])

# gaussian (default)
set.seed(1)
y <- fx + rnorm(n, sd = 1/sqrt(2))
mod <- gsm(y ~ x + z, knots = knots)
mean((mod$linear.predictors - fx)^2)

# compare to result from sm (they are identical)
mod.sm <- sm(y ~ x + z, knots = knots)
mean((mod$linear.predictors - mod.sm$fitted.values)^2)

# binomial (no weights)
```

```
set.seed(1)
y <- rbinom(n = n, size = 1, p = 1 / (1 + exp(-fx)))
mod <- gsm(y ~ x + z, family = binomial, knots = knots)
mean((mod$linear.predictors - fx)^2)

# binomial (w/ weights)
set.seed(1)
w <- as.integer(rep(c(10,20,30,40,50), length.out = n))
y <- rbinom(n = n, size = w, p = 1 / (1 + exp(-fx))) / w
mod <- gsm(y ~ x + z, family = binomial, weights = w, knots = knots)
mean((mod$linear.predictors - fx)^2)

# poisson
set.seed(1)
y <- rpois(n = n, lambda = exp(fx))
mod <- gsm(y ~ x + z, family = poisson, knots = knots)
mean((mod$linear.predictors - fx)^2)

# negative binomial (known theta)
set.seed(1)
y <- rnbinom(n = n, size = 1/2, mu = exp(fx))
mod <- gsm(y ~ x + z, family = NegBin(theta = 1/2), knots = knots)
mean((mod$linear.predictors - fx)^2)
mod$family$theta    # fixed theta

# negative binomial (unknown theta)
set.seed(1)
y <- rnbinom(n = n, size = 1/2, mu = exp(fx))
mod <- gsm(y ~ x + z, family = NegBin, knots = knots)
mean((mod$linear.predictors - fx)^2)
mod$family$theta    # estimated theta

# gamma
set.seed(1)
y <- rgamma(n = n, shape = 2, scale = (1 / (4 + fx)) / 2)
mod <- gsm(y ~ x + z, family = Gamma, knots = knots)
mean((mod$linear.predictors - fx - 4)^2)

# inverse.gaussian (not run; requires statmod)
##set.seed(1)
##y <- statmod::rinvgauss(n = n, mean = sqrt(1 / (4 + fx)), shape = 2)
##mod <- gsm(y ~ x + z, family = inverse.gaussian, knots = knots)
##mean((mod$linear.predictors - fx - 4)^2)
```

`model.matrix` *Construct Design Matrix for Fit Model*

Description

model.matrix returns the design (or model) matrix used by the input object to produce the fitted values (for objects of class ss or sm) or the linear predictors (for objects of class gsm).

Usage

```
## S3 method for class 'ss'
model.matrix(object, ...)

## S3 method for class 'sm'
model.matrix(object, ...)

## S3 method for class 'gsm'
model.matrix(object, ...)
```

Arguments

object	an object of class ss, sm, or gsm
...	additional arguments (currently ignored)

Details

For ss objects, the basis.poly function is used to construct the design matrix.

For sm objects, the predict.sm function with option design = TRUE is used to construct the design matrix.

For gsm objects, the predict.gsm function with option design = TRUE is used to construct the design matrix.

Value

The design matrix that is post-multiplied by the coefficients to produce the fitted values (or linear predictors).

Author(s)

Nathaniel E. Helwig <helwig@umn.edu>

References

Chambers, J. M. (1992) Data for models. Chapter 3 of *Statistical Models in S* eds J. M. Chambers and T. J. Hastie, Wadsworth & Brooks/Cole.

Helwig, N. E. (2020). Multiple and Generalized Nonparametric Regression. In P. Atkinson, S. Delamont, A. Cernat, J. W. Sakshaug, & R. A. Williams (Eds.), SAGE Research Methods Foundations. doi:10.4135/9781526421036885885

See Also

basis.poly for the smoothing spline basis

predict.sm for predicting from smooth models

predict.gsm for predicting from generalized smooth models

Examples

```
# generate data
set.seed(1)
n <- 100
x <- seq(0, 1, length.out = n)
fx <- 2 + 3 * x + sin(2 * pi * x)
y <- fx + rnorm(n, sd = 0.5)

# smoothing spline
mod.ss <- ss(x, y, nknots = 10)
X.ss <- model.matrix(mod.ss)
mean((mod.ss$y - X.ss %*% mod.ss$fit$coef)^2)


# smooth model
mod.sm <- sm(y ~ x, knots = 10)
X.sm <- model.matrix(mod.sm)
mean((mod.sm$fitted.values - X.sm %*% mod.sm$coefficients)^2)

# generalized smooth model (family = gaussian)
mod.gsm <- gsm(y ~ x, knots = 10)
X.gsm <- model.matrix(mod.gsm)
mean((mod.gsm$linear.predictors - X.gsm %*% mod.gsm$coefficients)^2)
```

---

msqrt *Matrix (Inverse?) Square Root*

Description

Stable computation of the square root (or inverse square root) of a positive semi-definite matrix.

Usage

```
msqrt(x, inverse = FALSE, symmetric = FALSE,
      tol = .Machine$double.eps, checkx = TRUE)
```

Arguments

x	positive semi-definite matrix
inverse	compute inverse square root?
symmetric	does the square root need to be symmetric? See Details.

tol	tolerance for detecting linear dependencies in x
checkx	should x be checked for symmetry using isSymmetric?

Details

If symmetric = FALSE, this function computes the matrix z such that x = tcrossprod(z)

If symmetric = TRUE, this function computes the matrix z such that x = crossprod(z) = tcrossprod(z)

If inverse = TRUE, the matrix x is replaced by the pseudo-inverse of x in these equations (see psolve)

Value

The matrix z that gives the (inverse?) square root of x. See Details.

Note

The matrix (inverse?) square root is calculated by (inverting and) square rooting the eigenvalues that are greater than the first value multiplied by tol * nrow(x)

Author(s)

Nathaniel E. Helwig <helwig@umn.edu>

See Also

psolve

Examples

```
# generate x
set.seed(0)
x <- crossprod(matrix(rnorm(100), 20, 5))

# asymmetric square root (default)
xsqrt <- msqrt(x)
mean(( x - crossprod(xsqrt) )^2)
mean(( x - tcrossprod(xsqrt) )^2)

# symmetric square root
xsqrt <- msqrt(x, symmetric = TRUE)
mean(( x - crossprod(xsqrt) )^2)
mean(( x - tcrossprod(xsqrt) )^2)

# asymmetric inverse square root (default)
xsqrt <- msqrt(x, inverse = TRUE)
mean(( solve(x) - crossprod(xsqrt) )^2)
mean(( solve(x) - tcrossprod(xsqrt) )^2)

# symmetric inverse square root
xsqrt <- msqrt(x, inverse = TRUE, symmetric = TRUE)
mean(( solve(x) - crossprod(xsqrt) )^2)
```

```
mean(( solve(x) - tcrossprod(xsqrt) )^2)
```

NegBin *Family Function for Negative Binomial*

Description

Creates the functions needed to fit a Negative Binomial generalized smooth model via gsm with or without a known theta parameter. Adapted from the negative.binomial function in the **MASS** package.

Usage

```
NegBin(theta = NULL, link = "log")
```

Arguments

theta	the size parameter for the Negative Binomial distribution. Default of NULL indicates that theta should be estimated from the data.
link	the link function. Must be log, sqrt, identity, or an object of class link-glm (as generated by make.link).

Details

The Negative Binomial distribution has mean μ and variance $\mu + \mu^2/\theta$, where the size parameter θ is the inverse of the dispersion parameter. See NegBinomial for details.

Value

An object of class "family" with the functions and expressions needed to fit the gsm. In addition to the standard values (see family), this also produces the following:

logLik	function to evaluate the log-likelihood
canpar	function to compute the canonical parameter
cumulant	function to compute the cumulant function
theta	the specified theta parameter
fixed.theta	logical specifying if theta was provided

Author(s)

Nathaniel E. Helwig <helwig@umn.edu>

References

Venables, W. N. and Ripley, B. D. (1999) Modern Applied Statistics with S-PLUS. Third Edition. Springer.

https://www.rdocumentation.org/packages/MASS/versions/7.3-51.6/topics/negative.binomial

https://www.rdocumentation.org/packages/stats/versions/3.6.2/topics/NegBinomial

See Also

gsm for fitting generalized smooth models with Negative Binomial responses

theta.mle for maximum likelihood estimation of theta

Examples

```
# generate data
n <- 1000
x <- seq(0, 1, length.out = n)
fx <- 3 * x + sin(2 * pi * x) - 1.5

# negative binomial (size = 1/2, log link)
set.seed(1)
y <- rnbinom(n = n, size = 1/2, mu = exp(fx))

# fit model (known theta)
mod <- gsm(y ~ x, family = NegBin(theta = 1/2), knots = 10)
mean((mod$linear.predictors - fx)^2)
mod$family$theta   # fixed theta

# fit model (unknown theta)
mod <- gsm(y ~ x, family = NegBin, knots = 10)
mean((mod$linear.predictors - fx)^2)
mod$family$theta   # estimated theta
```

---

nominal *Nominal Smoothing Spline Basis and Penalty*

Description

Generate the smoothing spline basis and penalty matrix for a nominal spline. This basis and penalty are for an unordered factor.

Usage

```
basis.nom(x, knots, K = NULL, intercept = FALSE, ridge = FALSE)

penalty.nom(x, K = NULL)
```

Arguments

x	Predictor variable (basis) or spline knots (penalty). Factor or integer vector of length n.
knots	Spline knots. Factor or integer vector of length r.
K	Number of levels of x. If NULL, this argument is defined as K = length(unique(x)).
intercept	If TRUE, the first column of the basis will be a column of ones.
ridge	If TRUE, the basis matrix is post-multiplied by the inverse square root of the penalty matrix. See Note and Examples.

Details

Generates a basis function or penalty matrix used to fit nominal smoothing splines.

With an intercept included, the basis function matrix has the form

$$X = [X_0, X_1]$$

where matrix X_0 is an n by 1 matrix of ones, and X_1 is a matrix of dimension n by r.

The X_0 matrix contains the "parametric part" of the basis (i.e., the intercept). The matrix X_1 contains the "nonparametric part" of the basis, which consists of the *reproducing kernel* function

$$\rho(x, y) = \delta_{xy} - 1/K$$

evaluated at all combinations of x and knots. The notation δ_{xy} denotes Kronecker's delta function.

The penalty matrix consists of the *reproducing kernel* function

$$\rho(x, y) = \delta_{xy} - 1/K$$

evaluated at all combinations of x.

Value

Basis: Matrix of dimension c(length(x), df) where df = length(knots) + intercept.

Penalty: Matrix of dimension c(r, r) where r = length(x) is the number of knots.

Note

If the inputs x and knots are factors, they should have the same levels.

If the inputs x and knots are integers, the knots should be a subset of the input x.

If ridge = TRUE, the penalty matrix is the identity matrix.

Author(s)

Nathaniel E. Helwig <helwig@umn.edu>

References

Gu, C. (2013). Smoothing Spline ANOVA Models. 2nd Ed. New York, NY: Springer-Verlag. doi:10.1007/9781461453697

Helwig, N. E. (2017). Regression with ordered predictors via ordinal smoothing splines. *Frontiers in Applied Mathematics and Statistics, 3*(15), 1-13. doi:10.3389/fams.2017.00015

Helwig, N. E. (2020). Multiple and Generalized Nonparametric Regression. In P. Atkinson, S. Delamont, A. Cernat, J. W. Sakshaug, & R. A. Williams (Eds.), *SAGE Research Methods Foundations.* doi:10.4135/9781526421036885885

Helwig, N. E., & Ma, P. (2015). Fast and stable multiple smoothing parameter selection in smoothing spline analysis of variance models with large samples. *Journal of Computational and Graphical Statistics, 24*(3), 715-732. doi:10.1080/10618600.2014.926819

See Also

See `ordinal` for a basis and penalty for ordered factors.

Examples

```
################   standard parameterization   ################

# generate data
set.seed(0)
n <- 101
x <- factor(sort(rep(LETTERS[1:4], length.out = n)))
knots <- LETTERS[1:3]
eta <- 1:4
y <- eta[x] + rnorm(n, sd = 0.5)

# nominal smoothing spline basis
X <- basis.nom(x, knots, intercept = TRUE)

# nominal smoothing spline penalty
Q <- penalty.nom(knots, K = 4)

# pad Q with zeros (for intercept)
Q <- rbind(0, cbind(0, Q))

# define smoothing parameter
lambda <- 1e-5

# estimate coefficients
coefs <- solve(crossprod(X) + n * lambda * Q) %*% crossprod(X, y)

# estimate eta
yhat <- X %*% coefs

# check rmse
sqrt(mean((eta[x] - yhat)^2))
```

```
################   ridge parameterization   ######**********

# generate data
set.seed(0)
n <- 101
x <- factor(sort(rep(LETTERS[1:4], length.out = n)))
knots <- LETTERS[1:3]
eta <- 1:4
y <- eta[x] + rnorm(n, sd = 0.5)

# nominal smoothing spline basis
X <- basis.nom(x, knots, intercept = TRUE, ridge = TRUE)

# nominal smoothing spline penalty (ridge)
Q <- diag(rep(c(0, 1), times = c(1, ncol(X) - 1)))

# define smoothing parameter
lambda <- 1e-5

# estimate coefficients
coefs <- solve(crossprod(X) + n * lambda * Q) %*% crossprod(X, y)

# estimate eta
yhat <- X %*% coefs

# check rmse
sqrt(mean((eta[x] - yhat)^2))
```

`number2color` *Map Numbers to Colors*

Description

Each of the n elements of a numeric vector is mapped onto one of the m specified colors.

Usage

```
number2color(x, colors, ncol = 21, equidistant = TRUE, xmin = min(x), xmax = max(x))
```

Arguments

`x`	numeric vector of observations that should be mapped to colors
`colors`	an optional vector of colors (see Note for default colors)
`ncol`	number of colors m used for mapping
`equidistant`	if `TRUE` (default), the breaks used for binning are an equidistant seqeunce of values spanning the range of `x`. Otherwise sample quantiles of `x` are used to define the bin breaks.

xmin	minimum x value to use when defining breaks
xmax	maximum x value to use when defining breaks

Details

Elements of a numeric vector are binned using either an equidistant sequence (default) or sample quantiles. Each bin is associated with a unique color, so binning the observations is equivalent to mapping the numbers to colors. The colors are input to the colorRampPalette function to create a color palette with length specified by the ncol argument.

Value

Returns of vector of colors the same length as x

Note

If colors is missing, the default color palette is defined as

```
colors <- c("darkblue", rainbow(12)[c(9, 8, 7, 5, 3, 2, 1)], "darkred")
```

which is a modified version of the rainbow color palette.

Author(s)

Nathaniel E. Helwig <helwig@umn.edu>

See Also

.bincode is used to bin the data

Examples

```
x <- 1:100
xcol <- number2color(x)
plot(x, col = xcol)
```

---

ordinal *Ordinal Smoothing Spline Basis and Penalty*

Description

Generate the smoothing spline basis and penalty matrix for an ordinal spline. This basis and penalty are for an ordered factor.

Usage

```
basis.ord(x, knots, K = NULL, intercept = FALSE, ridge = FALSE)

penalty.ord(x, K = NULL, xlev = NULL)
```

Arguments

x	Predictor variable (basis) or spline knots (penalty). Ordered factor or integer vector of length n.
knots	Spline knots. Ordered factor or integer vector of length r.
K	Number of levels of x. If NULL, this argument is defined as K = length(unique(x)).
xlev	Factor levels of x (for penalty). If NULL, the levels are defined as levels(as.ordered(x)).
intercept	If TRUE, the first column of the basis will be a column of ones.
ridge	If TRUE, the basis matrix is post-multiplied by the inverse square root of the penalty matrix. See Note and Examples.

Details

Generates a basis function or penalty matrix used to fit ordinal smoothing splines.

With an intercept included, the basis function matrix has the form

$$X = [X_0, X_1]$$

where matrix X_0 is an n by 1 matrix of ones, and X_1 is a matrix of dimension n by r. The X_0 matrix contains the "parametric part" of the basis (i.e., the intercept). The matrix X_1 contains the "nonparametric part" of the basis, which consists of the *reproducing kernel* function

$$\rho(x, y) = 1 - (x \vee y) + (1/2K) * (x(x-1) + y(y-1)) + c$$

evaluated at all combinations of x and knots. The notation $(x \vee y)$ denotes the maximum of x and y, and the constant is $c = (K-1)(2K-1)/(6K)$.

The penalty matrix consists of the *reproducing kernel* function

$$\rho(x, y) = 1 - (x \vee y) + (1/2K) * (x(x-1) + y(y-1)) + c$$

evaluated at all combinations of x.

Value

Basis: Matrix of dimension c(length(x), df) where df = length(knots) + intercept.

Penalty: Matrix of dimension c(r, r) where r = length(x) is the number of knots.

Note

If the inputs x and knots are factors, they should have the same levels.

If the inputs x and knots are integers, the knots should be a subset of the input x.

If ridge = TRUE, the penalty matrix is the identity matrix.

Author(s)

Nathaniel E. Helwig <helwig@umn.edu>

References

Gu, C. (2013). Smoothing Spline ANOVA Models. 2nd Ed. New York, NY: Springer-Verlag. doi:10.1007/9781461453697

Helwig, N. E. (2017). Regression with ordered predictors via ordinal smoothing splines. *Frontiers in Applied Mathematics and Statistics, 3*(15), 1-13. doi:10.3389/fams.2017.00015

Helwig, N. E. (2020). Multiple and Generalized Nonparametric Regression. In P. Atkinson, S. Delamont, A. Cernat, J. W. Sakshaug, & R. A. Williams (Eds.), *SAGE Research Methods Foundations.* doi:10.4135/9781526421036885885

See Also

See `nominal` for a basis and penalty for unordered factors.

See `polynomial` for a basis and penalty for numeric variables.

Examples

```
################   standard parameterization   ################

# generate data
set.seed(0)
n <- 101
x <- factor(sort(rep(LETTERS[1:4], length.out = n)))
knots <- LETTERS[1:3]
eta <- 1:4
y <- eta[x] + rnorm(n, sd = 0.5)

# ordinal smoothing spline basis
X <- basis.ord(x, knots, intercept = TRUE)

# ordinal smoothing spline penalty
Q <- penalty.ord(knots, K = 4)

# pad Q with zeros (for intercept)
Q <- rbind(0, cbind(0, Q))

# define smoothing parameter
lambda <- 1e-5

# estimate coefficients
coefs <- solve(crossprod(X) + n * lambda * Q) %*% crossprod(X, y)

# estimate eta
yhat <- X %*% coefs

# check rmse
sqrt(mean((eta[x] - yhat)^2))



################   ridge parameterization   ################
```

```
# generate data
set.seed(0)
n <- 101
x <- factor(sort(rep(LETTERS[1:4], length.out = n)))
knots <- LETTERS[1:3]
eta <- 1:4
y <- eta[x] + rnorm(n, sd = 0.5)

# ordinal smoothing spline basis
X <- basis.ord(x, knots, intercept = TRUE, ridge = TRUE)

# ordinal smoothing spline penalty (ridge)
Q <- diag(rep(c(0, 1), times = c(1, ncol(X) - 1)))

# define smoothing parameter
lambda <- 1e-5

# estimate coefficients
coefs <- solve(crossprod(X) + n * lambda * Q) %*% crossprod(X, y)

# estimate eta
yhat <- X %*% coefs

# check rmse
sqrt(mean((eta[x] - yhat)^2))
```

`plot.ss` *Plot method for Smoothing Spline Fit and Bootstrap*

Description

Default plotting methods for `ss` and `boot.ss` objects.

Usage

```
## S3 method for class 'ss'
plot(x, n = 201, ci = TRUE, xseq = NULL, ...)

## S3 method for class 'boot.ss'
plot(x, n = 201, ci = TRUE, xseq = NULL, ...)
```

Arguments

`x`	an object of class 'ss' or 'boot.ss'
`n`	number of points used to plot smoothing spline estimate
`ci`	logical indicating whether to include a confidence interval
`xseq`	ordered sequence of points at which to plot smoothing spline estimate
`...`	optional additional argument for the `plotci` function, e.g., `level`, `col`, etc.

Details

Unless a sequence of points is provided via the `xseq` arugment, the plots are created by evaluating the smoothing spline fit at an equidistant sequence of `n` values that span the range of the training data.

Value

Plot of the function estimate and confidence interval with the title displaying the effective degrees of freedom.

Note

The `plot.ss` and `plot.boot.ss` functions produce plots that only differ in terms of their confidence intervals: `plot.ss` uses the Bayesian CIs, whereas `plot.boot.ss` uses the bootstrap CIs.

Author(s)

Nathaniel E. Helwig <helwig@umn.edu>

See Also

`ss` and `boot.ss`

Examples

```
# generate data
set.seed(1)
n <- 100
x <- seq(0, 1, length.out = n)
fx <- 2 + 3 * x + sin(2 * pi * x)
y <- fx + rnorm(n, sd = 0.5)

# fit smoothing spline
ssfit <- ss(x, y, nknots = 10)

# plot smoothing spline fit
plot(ssfit)

## Not run:

# bootstrap smoothing spline
ssfitboot <- boot(ssfit)

# plot smoothing spline bootstrap
plot(ssfitboot)

## End(Not run)
```

plotci *Generic X-Y Plotting with Confidence Intervals*

Description

Modification to the plot function that adds confidence intervals. The CIs can be plotted using polygons (default) or error bars.

Usage

```
plotci(x, y, se, level = 0.95, crit.val = NULL,
       add = FALSE, col = NULL, col.ci = NULL,
       alpha = NULL, bars = NULL, bw = 0.05,
       linkinv = NULL, ci = NULL, ...)
```

Arguments

x	a vector of 'x' values (n by 1). If y is missing, the x input can be a list or matrix containing the x, y, and se arguments.
y	a vector of 'y' values (n by 1).
se	a vector of standard error values (n by 1).
level	confidence level for the intervals (between 0 and 1).
crit.val	an optional critical value for the intervals. If provided, the level input is ignored. See Details.
add	a switch controlling whether a new plot should be created (via a call to plot) or if the plot should be added to the current plot (via a call to lines).
col	a character specifying the color for plotting the lines/points.
col.ci	a character specifying the color for plotting the intervals.
alpha	a scalar between 0 and 1 controlling the transparency of the intervals.
bars	a switch controlling whether the intervals should be plotted as bars or polygons.
bw	a positive scalar controlling the bar width. Ignored if bars = FALSE.
linkinv	an inverse link function for the plotting. If provided, the function plots x versus linkinv(y) and the intervals are similarly transformed.
ci	an optional matrix if dimension $nx2$ giving the confidence interval lower and upper bounds: ci = cbind(lwr, upr)
...	extra arguments passed to the plot or lines function.

Details

This function plots x versus y with confidence intervals. Unless ci is provided, the CIs have the form
lwr = y - crit.val * se

upr = y + crit.val * se
where crit.val is the critical value.

If crit.val = NULL, the critival value is determined from the level input as
crit.val <- qnorm(1-(1-level)/2)
where qnorm is the quantile function for the standard normal distribution.

Author(s)

Nathaniel E. Helwig <helwig@umn.edu>

See Also

This function is used by plot.ss to plot smoothing spline fits.

Examples

```
# generate data
set.seed(1)
n <- 100
x <- seq(0, 1, length.out = n)
fx <- 2 + 3 * x + sin(2 * pi * x)
y <- fx + rnorm(n, sd = 0.5)

# fit smooth model
smod <- sm(y ~ x, knots = 10)

# plot fit with 95% CI polygon
plotci(x, smod$fitted.values, smod$se.fit)

# plot fit with 95% CI bars
plotci(x, smod$fitted.values, smod$se.fit, bars = TRUE)

# plot fit +/- 1 SE
plotci(x, smod$fitted.values, smod$se.fit, crit.val = 1, bars = TRUE)
```

---

polynomial *Polynomial Smoothing Spline Basis and Penalty*

Description

Generate the smoothing spline basis and penalty matrix for a polynomial spline. Derivatives of the smoothing spline basis matrix are supported.

Usage

```
basis.poly(x, knots, m = 2, d = 0, xmin = min(x), xmax = max(x),
           periodic = FALSE, rescale = FALSE, intercept = FALSE,
           bernoulli = TRUE, ridge = FALSE)

penalty.poly(x, m = 2, xmin = min(x), xmax = max(x),
             periodic = FALSE, rescale = FALSE, bernoulli = TRUE)
```

Arguments

x	Predictor variable (basis) or spline knots (penalty). Numeric or integer vector of length n.
knots	Spline knots. Numeric or integer vector of length r.
m	Penalty order. "m=1" for linear smoothing spline, "m=2" for cubic, and "m=3" for quintic.
d	Derivative order. "d=0" for smoothing spline basis, "d=1" for 1st derivative of basis, and "d=2" for 2nd derivative of basis.
xmin	Minimum value of "x".
xmax	Maximum value of "x".
periodic	If TRUE, the smoothing spline basis is periodic w.r.t. the interval [xmin, xmax].
rescale	If TRUE, the nonparametric part of the basis is divided by the average of the reproducing kernel function evaluated at the knots.
intercept	If TRUE, the first column of the basis will be a column of ones.
bernoulli	If TRUE, scaled Bernoulli polynomials are used for the basis and penalty functions.
ridge	If TRUE, the basis matrix is post-multiplied by the inverse square root of the penalty matrix. See Note and Examples.

Details

Generates a basis function or penalty matrix used to fit linear, cubic, and quintic smoothing splines (or evaluate their derivatives).

For non-periodic smoothing splines, the basis function matrix has the form

$$X = [X_0, X_1]$$

where the matrix X_0 is of dimension n by $m - 1$ (plus 1 if an intercept is included), and X_1 is a matrix of dimension n by r.

The X_0 matrix contains the "parametric part" of the basis, which includes polynomial functions of x up to degree $m - 1$.

The matrix X_1 contains the "nonparametric part" of the basis, which consists of the *reproducing kernel* function

$$\rho(x, y) = \kappa_m(x)\kappa_m(y) + (-1)^{m-1}\kappa_{2m}(|x - y|)$$

evaluated at all combinations of x and knots. The κ_v functions are scaled Bernoulli polynomials.

For periodic smoothing splines, the X_0 matrix only contains the intercept column and the modified reproducing kernel function

$$\rho(x, y) = (-1)^{m-1}\kappa_{2m}(|x - y|)$$

is evaluated for all combinations of `x` and `knots`.

For non-periodic smoothing splines, the penalty matrix consists of the *reproducing kernel* function

$$\rho(x, y) = \kappa_m(x)\kappa_m(y) + (-1)^{m-1}\kappa_{2m}(|x - y|)$$

evaluated at all combinations of `x`. For periodic smoothing splines, the modified reproducing kernel function

$$\rho(x, y) = (-1)^{m-1}\kappa_{2m}(|x - y|)$$

is evaluated for all combinations of `x`.

If `bernoulli = FALSE`, the reproducing kernel function is defined as

$$\rho(x, y) = (1/(m-1)!)^2 \int_0^1 (x - u)_+^{m-1}(y - u)_+^{m-1} du$$

where $(.)_+ = \max(., 0)$. This produces the "classic" definition of a smoothing spline, where the function estimate is a piecewise polynomial function with pieces of degree $2m - 1$.

Value

Basis: Matrix of dimension `c(length(x), df)` where `df >= length(knots)`. If the smoothing spline basis is not periodic (default), then the number of columns is `df = length(knots) + m - !intercept`. For periodic smoothing splines, the basis has `m` fewer columns.

Penalty: Matrix of dimension `c(r, r)` where `r = length(x)` is the number of knots.

Note

Inputs `x` and `knots` should be within the interval [`xmin`, `xmax`].

If `ridge = TRUE`, the penalty matrix is the identity matrix.

Author(s)

Nathaniel E. Helwig <helwig@umn.edu>

References

Gu, C. (2013). Smoothing Spline ANOVA Models. 2nd Ed. New York, NY: Springer-Verlag. doi:10.1007/9781461453697

Helwig, N. E. (2017). Regression with ordered predictors via ordinal smoothing splines. *Frontiers in Applied Mathematics and Statistics, 3*(15), 1-13. doi:10.3389/fams.2017.00015

Helwig, N. E. (2020). Multiple and Generalized Nonparametric Regression. In P. Atkinson, S. Delamont, A. Cernat, J. W. Sakshaug, & R. A. Williams (Eds.), *SAGE Research Methods Foundations.* doi:10.4135/9781526421036885885

Helwig, N. E., & Ma, P. (2015). Fast and stable multiple smoothing parameter selection in smoothing spline analysis of variance models with large samples. *Journal of Computational and Graphical Statistics, 24*(3), 715-732. doi:10.1080/10618600.2014.926819

See Also

See thinplate for a thin plate spline basis and penalty.

See ordinal for a basis and penalty for ordered factors.

Examples

```
################   standard parameterization   ################

# generate data
set.seed(0)
n <- 101
x <- seq(0, 1, length.out = n)
knots <- seq(0, 0.95, by = 0.05)
eta <- 1 + 2 * x + sin(2 * pi * x)
y <- eta + rnorm(n, sd = 0.5)

# cubic smoothing spline basis
X <- basis.poly(x, knots, intercept = TRUE)

# cubic smoothing spline penalty
Q <- penalty.poly(knots, xmin = min(x), xmax = max(x))

# pad Q with zeros (for intercept and linear effect)
Q <- rbind(0, 0, cbind(0, 0, Q))

# define smoothing parameter
lambda <- 1e-5

# estimate coefficients
coefs <- solve(crossprod(X) + n * lambda * Q) %*% crossprod(X, y)

# estimate eta
yhat <- X %*% coefs

# check rmse
sqrt(mean((eta - yhat)^2))

# plot results
plot(x, y)
lines(x, yhat)



################   ridge parameterization   ##################

# generate data
set.seed(0)
n <- 101
x <- seq(0, 1, length.out = n)
knots <- seq(0, 0.95, by = 0.05)
eta <- 1 + 2 * x + sin(2 * pi * x)
y <- eta + rnorm(n, sd = 0.5)
```

```
# cubic smoothing spline basis
X <- basis.poly(x, knots, intercept = TRUE, ridge = TRUE)

# cubic smoothing spline penalty (ridge)
Q <- diag(rep(c(0, 1), times = c(2, ncol(X) - 2)))

# define smoothing parameter
lambda <- 1e-5

# estimate coefficients
coefs <- solve(crossprod(X) + n * lambda * Q) %*% crossprod(X, y)

# estimate eta
yhat <- X %*% coefs

# check rmse
sqrt(mean((eta - yhat)^2))

# plot results
plot(x, y)
lines(x, yhat)
```

`predict.gsm` *Predict method for Generalized Smooth Model Fits*

Description

`predict` method for class "gsm".

Usage

```
## S3 method for class 'gsm'
predict(object, newdata = NULL, se.fit = FALSE,
        type = c("link", "response", "terms"),
        terms = NULL, na.action = na.pass,
        intercept = NULL, combine = TRUE, design = FALSE,
        check.newdata = TRUE, ...)
```

Arguments

`object`	a fit from `gsm`.
`newdata`	an optional list or data frame in which to look for variables with which to predict. If omitted, the original data are used.
`se.fit`	a switch indicating if standard errors are required.
`type`	type of prediction (link, response, or model term). Can be abbreviated.

terms	which terms to include in the fit. The default of NULL uses all terms. This input **is** used regardless of the type of prediction.
na.action	function determining what should be done with missing values in newdata. The default is to predict NA.
intercept	a switch indicating if the intercept should be included in the prediction. If NULL (default), the intercept is included in the fit only when type = "r" and terms includes all model terms.
combine	a switch indicating if the parametric and smooth components of the prediction should be combined (default) or returned separately.
design	a switch indicating if the model (design) matrix for the prediction should be returned.
check.newdata	a switch indicating if the newdata should be checked for consistency (e.g., class and range). Ignored if newdata is not provided.
...	additional arguments affecting the prediction produced (currently ignored).

Details

Inspired by the predict.glm function in R's **stats** package.

Produces predicted values, obtained by evaluating the regression function in the frame newdata (which defaults to model.frame(object)). If the logical se.fit is TRUE, standard errors of the predictions are calculated.

If newdata is omitted the predictions are based on the data used for the fit. Regardless of the newdata argument, how cases with missing values are handled is determined by the na.action argument. If na.action = na.omit omitted cases will not appear in the predictions, whereas if na.action = na.exclude they will appear (in predictions and standard errors), with value NA.

Similar to the glm function, setting type = "terms" returns a matrix giving the predictions for each of the requested model terms. Unlike the glm function, this function allows for predictions using any subset of the model terms. Specifically, the predictions (on both the link and response scale) will only include the requested terms, which makes it possible to obtain estimates (and standard errors) for subsets of model terms. In this case, the newdata only needs to contain data for the subset of variables that are requested in terms.

Value

Default use returns a vector of predictions. Otherwise the form of the output will depend on the combination of argumments: se.fit, type, combine, and design.

type = "link":
When se.fit = FALSE and design = FALSE, the output will be the predictions on the link scale. When se.fit = TRUE or design = TRUE, the output is a list with components fit, se.fit (if requested), and X (if requested).

type = "response":
When se.fit = FALSE and design = FALSE, the output will be the predictions on the data scale. When se.fit = TRUE or design = TRUE, the output is a list with components fit, se.fit (if requested), and X (if requested).

type = "terms":
When se.fit = FALSE and design = FALSE, the output will be the predictions for each term on

the link scale. When `se.fit = TRUE` or `design = TRUE`, the output is a list with components `fit`, `se.fit` (if requested), and `X` (if requested).

Regardless of the `type`, setting `combine = FALSE` decomposes the requested result(s) into the **p**arametric and **s**mooth contributions.

Author(s)

Nathaniel E. Helwig <helwig@umn.edu>

References

https://stat.ethz.ch/R-manual/R-devel/library/stats/html/predict.glm.html

Craven, P. and Wahba, G. (1979). Smoothing noisy data with spline functions: Estimating the correct degree of smoothing by the method of generalized cross-validation. *Numerische Mathematik, 31*, 377-403. doi:10.1007/BF01404567

Gu, C. (2013). Smoothing spline ANOVA models, 2nd edition. New York: Springer. doi:10.1007/9781461453697

Helwig, N. E. (2020). Multiple and Generalized Nonparametric Regression. In P. Atkinson, S. Delamont, A. Cernat, J. W. Sakshaug, & R. A. Williams (Eds.), *SAGE Research Methods Foundations.* doi:10.4135/9781526421036885885

See Also

`gsm`

Examples

```
# generate data
set.seed(1)
n <- 1000
x <- seq(0, 1, length.out = n)
z <- factor(sample(letters[1:3], size = n, replace = TRUE))
fun <- function(x, z){
  mu <- c(-2, 0, 2)
  zi <- as.integer(z)
  fx <- mu[zi] + 3 * x + sin(2 * pi * x + mu[zi]*pi/4)
}
fx <- fun(x, z)
y <- rbinom(n = n, size = 1, p = 1 / (1 + exp(-fx)))

# define marginal knots
probs <- seq(0, 0.9, by = 0.1)
knots <- list(x = quantile(x, probs = probs),
              z = letters[1:3])

# fit gsm with specified knots (tprk = TRUE)
gsm.ssa <- gsm(y ~ x * z, family = binomial, knots = knots)
pred <- predict(gsm.ssa)
term <- predict(gsm.ssa, type = "terms")
mean((gsm.ssa$linear.predictors - pred)^2)
```

```
mean((gsm.ssa$linear.predictors - rowSums(term) - attr(term, "constant"))^2)

# fit gsm with specified knots (tprk = FALSE)
gsm.gam <- gsm(y ~ x * z, family = binomial, knots = knots, tprk = FALSE)
pred <- predict(gsm.gam)
term <- predict(gsm.gam, type = "terms")
mean((gsm.gam$linear.predictors - pred)^2)
mean((gsm.gam$linear.predictors - rowSums(term) - attr(term, "constant"))^2)
```

`predict.sm` *Predict method for Smooth Model Fits*

Description

`predict` method for class "sm".

Usage

```
## S3 method for class 'sm'
predict(object, newdata = NULL, se.fit = FALSE,
        interval = c("none", "confidence", "prediction"),
        level = 0.95, type = c("response", "terms"),
        terms = NULL, na.action = na.pass,
        intercept = NULL, combine = TRUE, design = FALSE,
        check.newdata = TRUE, ...)
```

Arguments

`object`	a fit from `sm`.
`newdata`	an optional list or data frame in which to look for variables with which to predict. If omitted, the original data are used.
`se.fit`	a switch indicating if standard errors are required.
`interval`	type of interval calculation. Can be abbreviated.
`level`	tolerance/confidence level.
`type`	type of prediction (response or model term). Can be abbreviated.
`terms`	which terms to include in the fit. The default of `NULL` uses all terms. This input **is** used regardless of the `type` of prediction.
`na.action`	function determining what should be done with missing values in `newdata`. The default is to predict `NA`.
`intercept`	a switch indicating if the intercept should be included in the prediction. If `NULL` (default), the intercept is included in the fit only when `type = "r"` and `terms` includes all model terms.
`combine`	a switch indicating if the parametric and smooth components of the prediction should be combined (default) or returned separately.

design	a switch indicating if the model (design) matrix for the prediction should be returned.
check.newdata	a switch indicating if the newdata should be checked for consistency (e.g., class and range). Ignored if newdata is not provided.
...	additional arguments affecting the prediction produced (currently ignored).

Details

Inspired by the predict.lm function in R's **stats** package.

Produces predicted values, obtained by evaluating the regression function in the frame newdata (which defaults to model.frame(object)). If the logical se.fit is TRUE, standard errors of the predictions are calculated. Setting intervals specifies computation of confidence or prediction (tolerance) intervals at the specified level, sometimes referred to as narrow vs. wide intervals.

If newdata is omitted the predictions are based on the data used for the fit. Regardless of the newdata argument, how cases with missing values are handled is determined by the na.action argument. If na.action = na.omit omitted cases will not appear in the predictions, whereas if na.action = na.exclude they will appear (in predictions, standard errors or interval limits), with value NA.

Similar to the lm function, setting type = "terms" returns a matrix giving the predictions for each of the requested model terms. Unlike the lm function, this function allows for predictions using any subset of the model terms. Specifically, when type = "response" the predictions will only include the requested terms, which makes it possible to obtain estimates (and standard errors and intervals) for subsets of model terms. In this case, the newdata only needs to contain data for the subset of variables that are requested in terms.

Value

Default use returns a vector of predictions. Otherwise the form of the output will depend on the combination of argumments: se.fit, interval, type, combine, and design.

type = "response":
When se.fit = FALSE and design = FALSE, the output will be the predictions (possibly with lwr and upr interval bounds). When se.fit = TRUE or design = TRUE, the output is a list with components fit, se.fit (if requested), and X (if requested).

type = "terms":
When se.fit = FALSE and design = FALSE, the output will be the predictions for each term (possibly with lwr and upr interval bounds). When se.fit = TRUE or design = TRUE, the output is a list with components fit, se.fit (if requested), and X (if requested).

Regardless of the type, setting combine = FALSE decomposes the requested result(s) into the **p**arametric and **s**mooth contributions.

Author(s)

Nathaniel E. Helwig <helwig@umn.edu>

References

https://stat.ethz.ch/R-manual/R-devel/library/stats/html/predict.lm.html

Craven, P. and Wahba, G. (1979). Smoothing noisy data with spline functions: Estimating the correct degree of smoothing by the method of generalized cross-validation. *Numerische Mathematik, 31*, 377-403. doi:10.1007/BF01404567

Gu, C. (2013). Smoothing spline ANOVA models, 2nd edition. New York: Springer. doi:10.1007/9781461453697

Helwig, N. E. (2020). Multiple and Generalized Nonparametric Regression. In P. Atkinson, S. Delamont, A. Cernat, J. W. Sakshaug, & R. A. Williams (Eds.), *SAGE Research Methods Foundations.* doi:10.4135/9781526421036885885

See Also

sm

Examples

```
# generate data
set.seed(1)
n <- 100
x <- seq(0, 1, length.out = n)
z <- factor(sample(letters[1:3], size = n, replace = TRUE))
fun <- function(x, z){
  mu <- c(-2, 0, 2)
  zi <- as.integer(z)
  fx <- mu[zi] + 3 * x + sin(2 * pi * x + mu[zi]*pi/4)
}
fx <- fun(x, z)
y <- fx + rnorm(n, sd = 0.5)

# define marginal knots
probs <- seq(0, 0.9, by = 0.1)
knots <- list(x = quantile(x, probs = probs),
              z = letters[1:3])

# fit sm with specified knots
smod <- sm(y ~ x * z, knots = knots)

# get model "response" predictions
fit <- predict(smod)
mean((smod$fitted.values - fit)^2)

# get model "terms" predictions
trm <- predict(smod, type = "terms")
attr(trm, "constant")
head(trm)
mean((smod$fitted.values - rowSums(trm) - attr(trm, "constant"))^2)

# get predictions with "newdata" (= the original data)
fit <- predict(smod, newdata = data.frame(x = x, z = z))
```

```
mean((fit - smod$fitted.values)^2)

# get predictions and standard errors
fit <- predict(smod, se.fit = TRUE)
mean((fit$fit - smod$fitted.values)^2)
mean((fit$se.fit - smod$se.fit)^2)

# get 99% confidence interval
fit <- predict(smod, interval = "c", level = 0.99)
head(fit)

# get 99% prediction interval
fit <- predict(smod, interval = "p", level = 0.99)
head(fit)

# get predictions only for x main effect
fit <- predict(smod, newdata = data.frame(x = x),
               se.fit = TRUE, terms = "x")
plotci(x, fit$fit, fit$se.fit)

# get predictions only for each group
fit.a <- predict(smod, newdata = data.frame(x = x, z = "a"), se.fit = TRUE)
fit.b <- predict(smod, newdata = data.frame(x = x, z = "b"), se.fit = TRUE)
fit.c <- predict(smod, newdata = data.frame(x = x, z = "c"), se.fit = TRUE)

# plot results (truth as dashed line)
plotci(x = x, y = fit.a$fit, se = fit.a$se.fit,
       col = "red", col.ci = "pink", ylim = c(-6, 6))
lines(x, fun(x, rep(1, n)), lty = 2, col = "red")
plotci(x = x, y = fit.b$fit, se = fit.b$se.fit,
       col = "blue", col.ci = "cyan", add = TRUE)
lines(x, fun(x, rep(2, n)), lty = 2, col = "blue")
plotci(x = x, y = fit.c$fit, se = fit.c$se.fit,
       col = "darkgreen", col.ci = "lightgreen", add = TRUE)
lines(x, fun(x, rep(3, n)), lty = 2, col = "darkgreen")

# add legends
legend("bottomleft", legend = c("Truth", "Estimate", "CI"),
       lty = c(2, 1, NA), lwd = c(1, 2, NA),
       col = c("black", "black","gray80"),
       pch = c(NA, NA, 15), pt.cex = 2, bty = "n")
legend("bottomright", legend = letters[1:3],
       lwd = 2, col = c("red", "blue", "darkgreen"), bty = "n")
```

`predict.ss` *Predict method for Smoothing Spline Fits*

Description

`predict` method for class "ss".

Usage

```
## S3 method for class 'ss'
predict(object, x, deriv = 0, se.fit = TRUE, ...)
```

Arguments

object	a fit from ss.
x	the new values of x.
deriv	integer; the order of the derivative required.
se.fit	a switch indicating if standard errors are required.
...	additional arguments affecting the prediction produced (currently ignored).

Details

Inspired by the predict.smooth.spline function in R's **stats** package.

Value

A list with components

x	The input x.
y	The fitted values or derivatives at x.
se	The standard errors of the fitted values or derivatives (if requested).

Author(s)

Nathaniel E. Helwig <helwig@umn.edu>

References

https://stat.ethz.ch/R-manual/R-devel/library/stats/html/predict.smooth.spline.html

Craven, P. and Wahba, G. (1979). Smoothing noisy data with spline functions: Estimating the correct degree of smoothing by the method of generalized cross-validation. *Numerische Mathematik, 31*, 377-403. doi:10.1007/BF01404567

Gu, C. (2013). Smoothing spline ANOVA models, 2nd edition. New York: Springer. doi:10.1007/9781461453697

Helwig, N. E. (2020). Multiple and Generalized Nonparametric Regression. In P. Atkinson, S. Delamont, A. Cernat, J. W. Sakshaug, & R. A. Williams (Eds.), *SAGE Research Methods Foundations.* doi:10.4135/9781526421036885885

See Also

ss

Examples

```
# generate data
set.seed(1)
n <- 100
x <- seq(0, 1, length.out = n)
fx <- 2 + 3 * x + sin(2 * pi * x)
y <- fx + rnorm(n, sd = 0.5)

# GCV selection (default)
ss.GCV <- ss(x, y, nknots = 10)

# get predictions and SEs (at design points)
fit <- predict(ss.GCV, x = x)
head(fit)

# compare to original fit
mean((fit$y - ss.GCV$y)^2)

# plot result (with default 95% CI)
plotci(fit)

# estimate first derivative
d1 <- 3 + 2 * pi * cos(2 * pi * x)
fit <- predict(ss.GCV, x = x, deriv = 1)
head(fit)

# plot result (with default 95% CI)
plotci(fit)
lines(x, d1, lty = 2)   # truth
```

`psolve` *Pseudo-Solve a System of Equations*

Description

This generic function solves the equation `a %*% x = b` for `x`, where `b` can be either a vector or a matrix. This implementation is similar to `solve`, but uses a pseudo-inverse if the system is computationally singular.

Usage

```
psolve(a, b, tol)
```

Arguments

`a`	a rectangular numeric matrix containing the coefficients of the linear system.

b	a numeric vector or matrix giving the right-hand side(s) of the linear system. If missing, b is taken to be an identity matrix and solve will return the (pseudo-)inverse of a.
tol	the tolerance for detecting linear dependencies in the columns of a. The default is .Machine$double.eps.

Details

If a is a symmetric matrix, eigen is used to compute the (pseudo-)inverse. This assumes that a is a positive semi-definite matrix. Otherwise svd is used to compute the (pseudo-)inverse for rectangular matrices.

Value

If b is missing, returns the (pseudo-)inverse of a. Otherwise returns psolve(a) %*% b.

Note

The pseudo-inverse is calculated by inverting the eigen/singular values that are greater than the first value multiplied by tol * min(dim(a)).

Author(s)

Nathaniel E. Helwig <helwig@umn.edu>

References

Moore, E. H. (1920). On the reciprocal of the general algebraic matrix. *Bulletin of the American Mathematical Society, 26*, 394-395. doi:10.1090/S0002990419200332 27

Penrose, R. (1955). A generalized inverse for matrices. *Mathematical Proceedings of the Cambridge Philosophical Society, 51(3)*, 406-413. doi:10.1017/S0305004100030401

See Also

msqrt

Examples

```
# generate X
set.seed(0)
X <- matrix(rnorm(100), 20, 5)
X <- cbind(X, rowSums(X))

# pseudo-inverse of X  (dim = 6 by 20)
Xinv <- psolve(X)

# pseudo-inverse of crossprod(X)  (dim = 6 by 6)
XtXinv <- psolve(crossprod(X))
```

`residuals` *Extract Model Residuals*

Description

Extracts the residuals from a fit smoothing spline ("ss"), smooth model ("sm"), or generalized smooth model ("gsm") object.

Usage

```
## S3 method for class 'ss'
residuals(object, type = c("working", "response", "deviance",
                           "pearson", "partial"), ...)

## S3 method for class 'sm'
residuals(object, type = c("working", "response", "deviance",
                           "pearson", "partial"), ...)

## S3 method for class 'gsm'
residuals(object, type = c("deviance", "pearson", "working",
                           "response", "partial"), ...)
```

Arguments

`object`	an object of class "ss", "sm", or "gsm"
`type`	type of residuals
`...`	other arugments (currently ignored)

Details

For objects of class `ss` and `sm`
* the working and response residuals are defined as 'observed - fitted'
* the deviance and Pearson residuals multiply the working residuals by `sqrt(weights(object))`

For objects of class `gsm`, the residual types are the same as those produced by the `residuals.glm` function

Value

Residuals from `object`

Author(s)

Nathaniel E. Helwig <helwig@umn.edu>

References

Chambers, J. M. and Hastie, T. J. (1992) *Statistical Models in S*. Wadsworth & Brooks/Cole.

Helwig, N. E. (2020). Multiple and Generalized Nonparametric Regression. In P. Atkinson, S. Delamont, A. Cernat, J. W. Sakshaug, & R. A. Williams (Eds.), SAGE Research Methods Foundations. doi:10.4135/9781526421036885885

See Also

ss, sm, gsm

Examples

```
# generate data
set.seed(1)
n <- 100
x <- seq(0, 1, length.out = n)
fx <- 2 + 3 * x + sin(2 * pi * x)
y <- fx + rnorm(n, sd = 0.5)

# smoothing spline
mod.ss <- ss(x, y, nknots = 10)
res.ss <- residuals(mod.ss)

# smooth model
mod.sm <- sm(y ~ x, knots = 10)
res.sm <- residuals(mod.sm)

# generalized smooth model (family = gaussian)
mod.gsm <- gsm(y ~ x, knots = 10)
res.gsm <- residuals(mod.gsm)

# y = fitted + residuals
mean((y - fitted(mod.ss) - res.ss)^2)
mean((y - fitted(mod.sm) - res.sm)^2)
mean((y - fitted(mod.gsm) - res.gsm)^2)
```

---

sm *Fit a Smooth Model*

Description

Fits a semi- or nonparametric regression model with the smoothing parameter(s) selected via one of eight methods: GCV, OCV, GACV, ACV, REML, ML, AIC, or BIC.

Usage

```
sm(formula, data, weights, types = NULL, tprk = TRUE, knots = NULL,
   skip.iter = TRUE, df, spar = NULL, lambda = NULL, control = list(),
   method = c("GCV", "OCV", "GACV", "ACV", "REML", "ML", "AIC", "BIC"),
   xrange = NULL, thetas = NULL, mf = NULL)
```

Arguments

formula	Object of class "formula" (or one that can be coerced to that class): a symbolic description of the model to be fitted. Uses the same syntax as lm and glm.
data	Optional data frame, list or environment (or object coercible by as.data.frame to a data frame) containing the variables in the model. If not found in data, the variables are taken from environment(formula), typically the environment from which sm is called.
weights	Optional vector of weights to be used in the fitting process. If provided, weighted least squares is used. Defaults to all 1.
types	Named list giving the type of smooth to use for each predictor. If NULL, the type is inferred from the data. See "Types of Smooths" section for details.
tprk	Logical specifying how to parameterize smooth models with multiple predictors. If TRUE (default), a **t**ensor **p**roduct **r**eproducing **k**ernel function is used to represent the function. If FALSE, a tensor product of marginal kernel functions is used to represent the function. See the "Multiple Smooths" section for details.
knots	Spline knots for the estimation of the nonparametric effects. For models with multiple predictors, the knot specification will depend on the tprk input. See the "Choosing Knots" section for details
skip.iter	Set to FALSE for deep tuning of the hyperparameters. Only applicable when multiple smooth terms are included. See the "Parameter Tuning" section for details.
df	Equivalent degrees of freedom (trace of the smoother matrix). Must be in $[m, n]$ where m is the number of columns of the null space basis function matrix X, and n is the number of observations. Will be approximate if skip.iter = FALSE.
spar	Smoothing parameter. Typically (but not always) in the range $(0, 1]$. If specified lambda = 256^(3*(spar-1)).
lambda	Computational smoothing parameter. This value is weighted by n to form the penalty coefficient (see Details). Ignored if spar is provided.
control	Optional list with named components that control the optimization specs for the smoothing parameter selection routine. **Note** that spar is only searched for in the interval $[lower, upper]$. **lower:** lower bound for spar; defaults to -1.5 **upper:** upper bound for spar; defaults to 1.5 **tol:** the absolute precision (**tol**erance) used by optimize and nlm; defaults to 1e-8. **iterlim:** the iteration limit used by nlm; defaults to 5000. **print.level:** the print level used by nlm; defaults to 0 (no printing).
method	Method for selecting the smoothing parameter. Ignored if lambda is provided and skip.iter = TRUE.
xrange	Optional named list containing the range of each predictor. If NULL, the ranges are calculated from the input data.
thetas	Optional vector of hyperparameters to use for smoothing. If NULL, these are tuned using the requested method.

`mf` Optional model frame constructed from `formula` and `data` (and potentially `weights`).

Note: the last two arguments are not intended to be called by the typical user of this function. These arguments are included primarily for internal usage by the `boot.sm` function.

Details

Letting $f_i = f(x_i)$ with $x_i = (x_{i1}, \ldots, x_{ip})$, the function is represented as

$$f = X\beta + Z\alpha$$

where the basis functions in X span the null space (i.e., parametric effects), and Z contains the kernel function(s) of the contrast space (i.e., nonparametric effects) evaluated at all combinations of observed data points and knots. The vectors β and α contain unknown basis function coefficients.

Letting $M = (X, Z)$ and $\gamma = (\beta', \alpha')'$, the penalized least squares problem has the form

$$(y - M\gamma)'W(y - M\gamma) + n\lambda\alpha' Q\alpha$$

where W is a diagonal matrix containg the weights, and Q is the penalty matrix. The optimal coefficients are the solution to

$$(M'WM + n\lambda P)\gamma = M'Wy$$

where P is the penalty matrix Q augmented with zeros corresponding to the β in γ.

Value

An object of class "sm" with components:

`fitted.values`	the fitted values, i.e., predictions.
`se.fit`	the standard errors of the fitted values.
`sse`	the sum-of-squared errors.
`cv.crit`	the cross-validation criterion.
`nsdf`	the degrees of freedom (Df) for the null space.
`df`	the estimated degrees of freedom (Df) for the fit model.
`df.residual`	the residual degrees of freedom = `nobs - df`
`r.squared`	the observed coefficient of multiple determination.
`sigma`	the estimate of the error standard deviation.
`logLik`	the log-likelihood (if `method` is REML or ML).
`aic`	Akaike's Information Criterion (if `method` is AIC).
`bic`	Bayesian Information Criterion (if `method` is BIC).
`spar`	the value of `spar` computed or given, i.e., $s = 1 + \log_{256}(\lambda)/3$
`lambda`	the value of λ corresponding to `spar`, i.e., $\lambda = 256^{3(s-1)}$.
`penalty`	the smoothness penalty $\alpha' Q\alpha$.
`coefficients`	the basis function coefficients used for the fit model.

`cov.sqrt`	the square-root of the covariance matrix of `coefficients`. Note: `tcrossprod(cov.sqrt)` reconstructs the covariance matrix.
`iter`	the number of iterations used by `nlm` (if applicable).
`specs`	a list with information used for prediction purposes:
	knots the spline knots used for each predictor.
	thetas the "extra" tuning parameters used to weight the penalties.
	xrng the ranges of the predictor variables.
	xlev the factor levels of the predictor variables (if applicable).
	tprk logical controlling the formation of tensor product smooths.
	skip.iter logical controlling the parameter tuning (same as input).
	control the `control` options use for tuning.
`data`	the data used to fit the model.
`types`	the type of smooth used for each predictor.
`terms`	the terms included in the fit model.
`method`	the `method` used for smoothing parameter selection. Will be `NULL` if `lambda` was provided.
`formula`	the formula specifying the fit model.
`weights`	the weights used for fitting (if applicable)
`call`	the matched call.

Methods

The smoothing parameter can be selected using one of eight methods:
Generalized Cross-Validation (GCV)
Ordinary Cross-Validation (OCV)
Generalized Approximate Cross-Validation (GACV)
Approximate Cross-Validation (ACV)
Restricted Maximum Likelihood (REML)
Maximum Likelihood (ML)
Akaike's Information Criterion (AIC)
Bayesian Information Criterion (BIC)

Types of Smooths

The following codes specify the spline types:

par	Parametric effect (factor, integer, or numeric).
nom	Nominal smoothing spline (unordered factor).
ord	Ordinal smoothing spline (ordered factor).
lin	Linear smoothing spline (integer or numeric).
cub	Cubic smoothing spline (integer or numeric).
qui	Quintic smoothing spline (integer or numeric).
per	Periodic smoothing spline (integer or numeric).
sph	Spherical spline (matrix with $d = 2$ columns: lat, long).
tps	Thin plate spline (matrix with $d \geq 1$ columns).

For finer control of some specialized spline types:

per.lin	Linear periodic spline ($m = 1$).
per.cub	Cubic periodic spline ($m = 2$).
per.qui	Quintic periodic spline ($m = 3$).
sph.2	Linear spherical spline ($m = 2$).
sph.3	Cubic spherical spline ($m = 3$).
sph.4	Quintic spherical spline ($m = 4$).
tps.lin	Linear thin plate spline ($m = 1$).
tps.cub	Cubic thin plate spline ($m = 2$).
tps.qui	Quintic thin plate spline ($m = 3$).

For details on the spline kernel functions, see `basis.nom` (nominal), `basis.ord` (ordinal), `basis.poly` (polynomial), `basis.sph` (spherical), and `basis.tps` (thin plate).

Choosing Knots

If `tprk = TRUE`, the four options for the `knots` input include:

1. a scalar giving the total number of knots to sample
2. a vector of integers indexing which rows of data are the knots
3. a list with named elements giving the marginal knot values for each predictor (to be combined via `expand.grid`)
4. a list with named elements giving the knot values for each predictor (requires the same number of knots for each predicto

If `tprk = FALSE`, the three options for the `knots` input include:

1. a scalar giving the common number of knots for each continuous predictor
2. a list with named elements giving the number of marginal knots for each predictor
3. a list with named elements giving the marginal knot values for each predictor

Multiple Smooths

Suppose `formula = y ~ x1 + x2` so that the model contains additive effects of two predictor variables.

The k-th predictor's marginal effect can be denoted as

$$f_k = X_k\beta_k + Z_k\alpha_k$$

where X_k is the n by m_k null space basis function matrix, and Z_k is the n by r_k contrast space basis function matrix.

If `tprk = TRUE`, the null space basis function matrix has the form $X = [1, X_1, X_2]$ and the contrast space basis function matrix has the form

$$Z = \theta_1 Z_1 + \theta_2 Z_2$$

where the θ_k are the "extra" smoothing parameters. Note that Z is of dimension n by $r = r_1 = r_2$.

If `tprk = FALSE`, the null space basis function matrix has the form $X = [1, X_1, X_2]$, and the contrast space basis function matrix has the form

$$Z = [\theta_1 Z_1, \theta_2 Z_2]$$

where the θ_k are the "extra" smoothing parameters. Note that Z is of dimension n by $r = r_1 + r_2$.

Parameter Tuning

When multiple smooth terms are included in the model, there are smoothing (hyper)parameters that weight the contribution of each combination of smooth terms. These hyperparameters are distinct from the overall smoothing parameter `lambda` that weights the contribution of the penalty.

`skip.iter = TRUE` (default) estimates the smoothing hyperparameters using Algorithm 3.2 of Gu and Wahba (1991), which typically provides adequate results when the model form is correctly specified. The `lambda` parameter is tuned via the specified smoothing parameter selection `method`.

`skip.iter = FALSE` uses Algorithm 3.2 as an initialization, and then the `nlm` function is used to tune the hyperparameters via the specified smoothing parameter selection `method`. Setting `skip.iter = FALSE` can (substantially) increase the model fitting time, but should produce better results—particularly if the model `formula` is misspecified.

Author(s)

Nathaniel E. Helwig <helwig@umn.edu>

References

Berry, L. N., & Helwig, N. E. (2021). Cross-validation, information theory, or maximum likelihood? A comparison of tuning methods for penalized splines. *Stats, 4*(3), 701-724. doi:10.3390/stats4030042

Craven, P. and Wahba, G. (1979). Smoothing noisy data with spline functions: Estimating the correct degree of smoothing by the method of generalized cross-validation. *Numerische Mathematik, 31*, 377-403. doi:10.1007/BF01404567

Gu, C. (2013). Smoothing spline ANOVA models, 2nd edition. New York: Springer. doi:10.1007/9781461453697

Gu, C. and Wahba, G. (1991). Minimizing GCV/GML scores with multiple smoothing parameters via the Newton method. *SIAM Journal on Scientific and Statistical Computing, 12(2)*, 383-398. doi:10.1137/0912021

Helwig, N. E. (2020). Multiple and Generalized Nonparametric Regression. In P. Atkinson, S. Delamont, A. Cernat, J. W. Sakshaug, & R. A. Williams (Eds.), SAGE Research Methods Foundations. doi:10.4135/9781526421036885885

Helwig, N. E. (2021). Spectrally sparse nonparametric regression via elastic net regularized smoothers. *Journal of Computational and Graphical Statistics, 30*(1), 182-191. doi:10.1080/10618600.2020.1806855

See Also

Related Modeling Functions:

`ss` for fitting a smoothing spline with a single predictor (Gaussian response).

gsm for fitting generalized smooth models with multiple predictors of mixed types (non-Gaussian response).

S3 Methods and Related Functions for "sm" Objects:

boot.sm for bootstrapping sm objects.

coef.sm for extracting coefficients from sm objects.

cooks.distance.sm for calculating Cook's distances from sm objects.

cov.ratio for computing covariance ratio from sm objects.

deviance.sm for extracting deviance from sm objects.

dfbeta.sm for calculating DFBETA from sm objects.

dfbetas.sm for calculating DFBETAS from sm objects.

diagnostic.plots for plotting regression diagnostics from sm objects.

fitted.sm for extracting fitted values from sm objects.

hatvalues.sm for extracting leverages from sm objects.

model.matrix.sm for constructing model matrix from sm objects.

predict.sm for predicting from sm objects.

residuals.sm for extracting residuals from sm objects.

rstandard.sm for computing standardized residuals from sm objects.

rstudent.sm for computing studentized residuals from sm objects.

smooth.influence for calculating basic influence information from sm objects.

smooth.influence.measures for convenient display of influential observations from sm objects.

summary.sm for summarizing sm objects.

vcov.sm for extracting coefficient covariance matrix from sm objects.

weights.sm for extracting prior weights from sm objects.

Examples

```
##########   EXAMPLE 1   ##########
### 1 continuous predictor

# generate data
set.seed(1)
n <- 100
x <- seq(0, 1, length.out = n)
fx <- 2 + 3 * x + sin(2 * pi * x)
y <- fx + rnorm(n, sd = 0.5)

# fit sm with 10 knots (tprk = TRUE)
sm.ssa <- sm(y ~ x, knots = 10)

# fit sm with 10 knots (tprk = FALSE)
sm.gam <- sm(y ~ x, knots = 10, tprk = FALSE)
```

```
# print both results (note: they are identical)
sm.ssa
sm.gam

# summarize both results (note: they are identical)
summary(sm.ssa)
summary(sm.gam)

# compare true MSE values (note: they are identical)
mean( ( fx - sm.ssa$fit )^2 )
mean( ( fx - sm.gam$fit )^2 )



##########   EXAMPLE 2   ##########
### 1 continuous and 1 nominal predictor
### additive model

# generate data
set.seed(1)
n <- 100
x <- seq(0, 1, length.out = n)
z <- factor(sample(letters[1:3], size = n, replace = TRUE))
fun <- function(x, z){
  mu <- c(-2, 0, 2)
  zi <- as.integer(z)
  fx <- mu[zi] + 3 * x + sin(2 * pi * x)
}
fx <- fun(x, z)
y <- fx + rnorm(n, sd = 0.5)

# define marginal knots
probs <- seq(0, 0.9, by = 0.1)
knots <- list(x = quantile(x, probs = probs),
              z = letters[1:3])

# fit sm with specified knots (tprk = TRUE)
sm.ssa <- sm(y ~ x + z, knots = knots)

# fit sm with specified knots (tprk = FALSE)
sm.gam <- sm(y ~ x + z, knots = knots, tprk = FALSE)

# print both results (note: they are identical)
sm.ssa
sm.gam

# summarize both results (note: they are almost identical)
summary(sm.ssa)
summary(sm.gam)

# compare true MSE values (note: they are identical)
mean( ( fx - sm.ssa$fit )^2 )
mean( ( fx - sm.gam$fit )^2 )
```

```
##########   EXAMPLE 3   ##########
### 1 continuous and 1 nominal predictor
### interaction model

# generate data
set.seed(1)
n <- 100
x <- seq(0, 1, length.out = n)
z <- factor(sample(letters[1:3], size = n, replace = TRUE))
fun <- function(x, z){
  mu <- c(-2, 0, 2)
  zi <- as.integer(z)
  fx <- mu[zi] + 3 * x + sin(2 * pi * x + mu[zi]*pi/4)
}
fx <- fun(x, z)
y <- fx + rnorm(n, sd = 0.5)

# define marginal knots
probs <- seq(0, 0.9, by = 0.1)
knots <- list(x = quantile(x, probs = probs),
              z = letters[1:3])

# fit sm with specified knots (tprk = TRUE)
sm.ssa <- sm(y ~ x * z, knots = knots)

# fit sm with specified knots (tprk = FALSE)
sm.gam <- sm(y ~ x * z, knots = knots, tprk = FALSE)

# print both results (note: they are slightly different)
sm.ssa
sm.gam

# summarize both results (note: they are slightly different)
summary(sm.ssa)
summary(sm.gam)

# compare true MSE values (note: they are slightly different)
mean( ( fx - sm.ssa$fit )^2 )
mean( ( fx - sm.gam$fit )^2 )



##########   EXAMPLE 4   ##########
### 4 continuous predictors
### additive model

# generate data
set.seed(1)
n <- 100
fun <- function(x){
```

```
  sin(pi*x[,1]) + sin(2*pi*x[,2]) + sin(3*pi*x[,3]) + sin(4*pi*x[,4])
}
data <- as.data.frame(replicate(4, runif(n)))
colnames(data) <- c("x1v", "x2v", "x3v", "x4v")
fx <- fun(data)
y <- fx + rnorm(n)

# define marginal knots
knots <- list(x1v = quantile(data$x1v, probs = seq(0, 1, length.out = 10)),
              x2v = quantile(data$x2v, probs = seq(0, 1, length.out = 10)),
              x3v = quantile(data$x3v, probs = seq(0, 1, length.out = 10)),
              x4v = quantile(data$x4v, probs = seq(0, 1, length.out = 10)))

# define ssa knot indices
knots.indx <- c(bin.sample(data$x1v, nbin = 10, index.return = TRUE)$ix,
                bin.sample(data$x2v, nbin = 10, index.return = TRUE)$ix,
                bin.sample(data$x3v, nbin = 10, index.return = TRUE)$ix,
                bin.sample(data$x4v, nbin = 10, index.return = TRUE)$ix)

# fit sm with specified knots (tprk = TRUE)
sm.ssa <- sm(y ~ x1v + x2v + x3v + x4v, data = data, knots = knots.indx)

# fit sm with specified knots (tprk = FALSE)
sm.gam <- sm(y ~ x1v + x2v + x3v + x4v, data = data, knots = knots, tprk = FALSE)

# print both results (note: they are slightly different)
sm.ssa
sm.gam

# summarize both results (note: they are slightly different)
summary(sm.ssa)
summary(sm.gam)

# compare true MSE values (note: they are slightly different)
mean( ( fx - sm.ssa$fit )^2 )
mean( ( fx - sm.gam$fit )^2 )
```

`smooth.influence` *Nonparametric Regression Diagnostics*

Description

These functions provide the basic quantities that are used to form a variety of diagnostics for checking the quality of a fit smoothing spline (fit by `ss`), smooth model (fit by `sm`), or generalized smooth model (fit by `gsm`).

Usage

```
## S3 method for class 'ss'
```

```
influence(model, do.coef = TRUE, ...)
## S3 method for class 'sm'
influence(model, do.coef = TRUE, ...)
## S3 method for class 'gsm'
influence(model, do.coef = TRUE, ...)

smooth.influence(model, do.coef = TRUE)
```

Arguments

model	an object of class "gsm" output by the gsm function, "sm" output by the sm function, or "ss" output by the ss function
do.coef	logical indicating if the changed coefficients are desired (see Details).
...	additional arguments (currently ignored)

Details

Inspired by influence and lm.influence functions in R's **stats** package.

The functions documented in smooth.influence.measures provide a more user-friendly way of computing a variety of regression diagnostics.

For non-Gaussian gsm objects, these regression diagnostics are based on one-step approximations, which may be inadequate if a case has high influence.

For all models, the diagostics are computed assuming that the smoothing parameters are fixed at the given values.

Value

A list with the components

hat	a vector containing the leverages, i.e., the diagonals of the smoothing matrix
coefficients	if do.coef is true, a matrix whose i-th row contains the change in the estimated coefficients which results when the i-th case is excluded from the fitting.
deviance	a vector whose i-th entry contains the deviance which results when the i-th case is excluded from the fitting.
df	a vector whose i-th entry contains the effective degrees-of-freedom which results when the i-th case is excluded from the fitting.
sigma	a vector whose i-th element contains the estimate of the residual standard deviation obtained when the i-th case is excluded from the fitting.
wt.res	a vector of *weighted* (or for class gsm rather *deviance*) residuals.

Warning

The approximations used for gsm objects can result in sigma estimates being NaN.

Note

The `coefficients` returned by `smooth.influence` (and the corresponding functions S3 `influence` methods) are the *change* in the coefficients which result from dropping each case, i.e., $\theta - \theta_i$, where θ are the original coefficients obtained from the full sample of n observations and θ_i are the coefficients that result from dropping the i-th case.

Author(s)

Nathaniel E. Helwig <helwig@umn.edu>

References

See the list in the documentation for `influence.measures`

Chambers, J. M. (1992) *Linear models.* Chapter 4 of *Statistical Models in S* eds J. M. Chambers and T. J. Hastie, Wadsworth & Brooks/Cole.

See Also

`ss`, `sm`, `gsm` for modeling functions

`smooth.influence.measures` for convenient summary

`diagnostic.plots` for regression diagnostic plots

Examples

```
# generate data
set.seed(1)
n <- 100
x <- seq(0, 1, length.out = n)
fx <- 2 + 3 * x + sin(2 * pi * x)
y <- fx + rnorm(n, sd = 0.5)

# fit models
mod.ss <- ss(x, y, nknots = 10)
mod.sm <- sm(y ~ x, knots = 10)
mod.gsm <- gsm(y ~ x, knots = 10)

# calculate influence
infl.ss <- influence(mod.ss)
infl.sm <- influence(mod.sm)
infl.gsm <- influence(mod.gsm)

# compare hat
mean((infl.ss$hat - infl.sm$hat)^2)
mean((infl.ss$hat - infl.gsm$hat)^2)
mean((infl.sm$hat - infl.gsm$hat)^2)

# compare deviance
mean((infl.ss$deviance - infl.sm$deviance)^2)
mean((infl.ss$deviance - infl.gsm$deviance)^2)
mean((infl.sm$deviance - infl.gsm$deviance)^2)
```

```
# compare df
mean((infl.ss$df - infl.sm$df)^2)
mean((infl.ss$df - infl.gsm$df)^2)
mean((infl.sm$df - infl.gsm$df)^2)

# compare sigma
mean((infl.ss$sigma - infl.sm$sigma)^2)
mean((infl.ss$sigma - infl.gsm$sigma)^2)
mean((infl.sm$sigma - infl.gsm$sigma)^2)

# compare residuals
mean((infl.ss$wt.res - infl.sm$wt.res)^2)
mean((infl.ss$wt.res - infl.gsm$dev.res)^2)
mean((infl.sm$wt.res - infl.gsm$dev.res)^2)

# NOTE: ss() coef only comparable to sm() and gsm() after rescaling
scale.sm <- rep(c(1, mod.sm$specs$thetas), times = c(2, 10))
scale.gsm <- rep(c(1, mod.gsm$specs$thetas), times = c(2, 10))
mean((coef(mod.ss) / scale.sm - coef(mod.sm))^2)
mean((coef(mod.ss) / scale.gsm - coef(mod.gsm))^2)
mean((coef(mod.sm) - coef(mod.gsm))^2)

# infl.ss$coefficients are *not* comparable to others
mean((infl.ss$coefficients - infl.sm$coefficients)^2)
mean((infl.ss$coefficients - infl.gsm$coefficients)^2)
mean((infl.sm$coefficients - infl.gsm$coefficients)^2)
```

`smooth.influence.measures`

Nonparametric Regression Deletion Diagnostics

Description

These functions compute several regression (leave-one-out deletion) diagnostics for a fit smoothing spline (fit by `ss`), smooth model (fit by `sm`), or generalized smooth model (fit by `gsm`).

Usage

```
smooth.influence.measures(model, infl = smooth.influence(model))

## S3 method for class 'ss'
rstandard(model, infl = NULL, sd = model$sigma,
          type = c("sd.1", "predictive"), ...)
## S3 method for class 'sm'
rstandard(model, infl = NULL, sd = model$sigma,
          type = c("sd.1", "predictive"), ...)
## S3 method for class 'gsm'
```

```
rstandard(model, infl = NULL,
          type = c("deviance", "pearson"), ...)

## S3 method for class 'ss'
rstudent(model, infl = influence(model, do.coef = FALSE),
         res = infl$wt.res, ...)
## S3 method for class 'sm'
rstudent(model, infl = influence(model, do.coef = FALSE),
         res = infl$wt.res, ...)
## S3 method for class 'gsm'
rstudent(model, infl = influence(model, do.coef = FALSE), ...)

## S3 method for class 'ss'
dfbeta(model, infl = NULL, ...)
## S3 method for class 'sm'
dfbeta(model, infl = NULL, ...)
## S3 method for class 'gsm'
dfbeta(model, infl = NULL, ...)

## S3 method for class 'ss'
dfbetas(model, infl = smooth.influence(model, do.coef = TRUE), ...)
## S3 method for class 'sm'
dfbetas(model, infl = smooth.influence(model, do.coef = TRUE), ...)
## S3 method for class 'gsm'
dfbetas(model, infl = smooth.influence(model, do.coef = TRUE), ...)

cov.ratio(model, infl = smooth.influence(model, do.coef = FALSE),
          res = weighted.residuals(model))

## S3 method for class 'ss'
cooks.distance(model, infl = NULL, res = weighted.residuals(model),
               sd = model$sigma, hat = hatvalues(model), ...)
## S3 method for class 'sm'
cooks.distance(model, infl = NULL, res = weighted.residuals(model),
               sd = model$sigma, hat = hatvalues(model), ...)
## S3 method for class 'gsm'
cooks.distance(model, infl = NULL, res = residuals(model, type = "pearson"),
               dispersion = model$dispersion, hat = hatvalues(model), ...)

## S3 method for class 'ss'
hatvalues(model, ...)
## S3 method for class 'sm'
hatvalues(model, ...)
## S3 method for class 'gsm'
hatvalues(model, ...)
```

Arguments

model	an object of class "gsm" output by the gsm function, "sm" output by the sm function, or "ss" output by the ss function
infl	influence structure as returned by smooth.influence
res	(possibly weighted) residuals with proper defaults
sd	standard deviation to use, see defaults
dispersion	dispersion (for gsm objects) to use, see defaults
hat	hat values S_{ii}, see defaults
type	type of residuals for rstandard
...	additional arguments (currently ignored)

Details

Inspired by influence.measures and related functions in R's **stats** package.

The function smooth.influence.measures produces a class "infl" object, which displays the DFBETAS for each coefficient, DFFITS, covariance ratios, Cook's distance, and the diagonals of the smoothing matrix. Cases which are influential with respect to any of these measures are marked with an asterisk.

The S3 methods dfbetas, dffits, covratio, and cooks.distance provide direct access to the corresponding diagnostic quantities. The S3 methods rstandard and rstudent give the standardized and Studentized residuals, respectively. (These re-normalize the residuals to have unit variance, using an overall and leave-one-out measure of the error variance, respectively.)

Values for generalized smoothing models are approximations, as described in Williams (1987) (except that Cook's distances are scaled as F rather than chi-square values). THe approximations can be poor when some cases have large influence.

The optional infl, res, and sd arguments are there to encourage the use of these direct access functions in situations where the underlying basic influence measures, e.g., from smooth.influence, are already available.

For ss and sm objects, the code rstandard(*, type = "predictive") returns the leave-one-out (ordinary) cross-validation residuals, and the PRESS (PREdictive Sum of Squares) statistic is defined as

```
PRESS <- sum(rstandard(model, type = "predictive")^2)
```

Note that OCV = PRESS / n, where OCV = ordinary cross-validation criterion

Note

Note: the dffits function in R's **stats** package can be used with the following syntax

```
dffits(model, infl = smooth.influence(model, do.coef = FALSE), res = weighted.residuals(model))
```

Author(s)

Nathaniel E. Helwig <helwig@umn.edu>

References

See references listed in influence.measures

Williams, D. A. (1987). Generalized linear model diagnostics using the deviance and single case deletions. *Applied Statistics, 36*, 181-191. doi:10.2307/2347550

See Also

ss, sm, gsm for modeling functions

smooth.influence for some basic influence information

diagnostic.plots for regression diagnostic plots

Examples

```
# generate data
set.seed(1)
n <- 100
x <- seq(0, 1, length.out = n)
fx <- 2 + 3 * x + sin(2 * pi * x)
y <- fx + rnorm(n, sd = 0.5)

# fit models
mod.ss <- ss(x, y, nknots = 10)
mod.sm <- sm(y ~ x, knots = 10)
mod.gsm <- gsm(y ~ x, knots = 10)

# calculate influence
infl.ss <- smooth.influence.measures(mod.ss)
infl.sm <- smooth.influence.measures(mod.sm)
infl.gsm <- smooth.influence.measures(mod.gsm)

# standardized residuals
rstan.ss <- rstandard(mod.ss)
rstan.sm <- rstandard(mod.sm)
rstan.gsm <- rstandard(mod.gsm)

# studentized residuals
rstud.ss <- rstudent(mod.ss)
rstud.sm <- rstudent(mod.sm)
rstud.gsm <- rstudent(mod.gsm)
```

---

spherical *Spherical Spline Basis and Penalty*

Description

Generate the smoothing spline basis and penalty matrix for a spherical spline. This basis is designed for predictors where the values are points on a sphere.

Usage

```
basis.sph(x, knots, m = 2, intercept = FALSE, ridge = FALSE)

penalty.sph(x, m = 2)
```

Arguments

x	Predictor variables (basis) or spline knots (penalty). Matrix of dimension n by 2. Column 1 is latitude (-90 to 90 deg) and column 2 is longitude (-180 to 180 deg).
knots	Spline knots. Matrix of dimension r by 2. Column 1 is latitude (-90 to 90 deg) and column 2 is longitude (-180 to 180 deg).
m	Penalty order. "m=2" for 2nd order spherical spline, "m=3" for 3rd order, and "m=4" for 4th order.
intercept	If TRUE, the first column of the basis will be a column of ones.
ridge	If TRUE, the basis matrix is post-multiplied by the inverse square root of the penalty matrix. See Note and Examples.

Details

Generates a basis function or penalty matrix used to fit spherical splines of order 2, 3, or 4.

With an intercept included, the basis function matrix has the form

$$X = [X_0, X_1]$$

where matrix X_0 is an n by 1 matrix of ones, and X_1 is a matrix of dimension n by r.

The X_0 matrix contains the "parametric part" of the basis (i.e., the intercept).

The matrix X_1 contains the "nonparametric part" of the basis, which consists of the *reproducing kernel* function

$$\rho(x, y) = [q_{2m-2}(x.y) - \alpha]/\beta$$

evaluated at all combinations of x and knots. Note that $\alpha = 1/(2m-1)$ and $\beta = 2\pi(2m-2)!$ are constants, $q_{2m-2}(.)$ is the spherical spline semi-kernel function, and $x.y$ denotes the cosine of the angle between x and y (see References).

The penalty matrix consists of the *reproducing kernel* function

$$\rho(x, y) = [q_{2m-2}(x.y) - \alpha]/\beta$$

evaluated at all combinations of x.

Value

Basis: Matrix of dimension c(length(x), df) where df = nrow(knots) + intercept.

Penalty: Matrix of dimension c(r, r) where r = nrow(x) is the number of knots.

Note

The inputs x and knots must have the same dimension.

If ridge = TRUE, the penalty matrix is the identity matrix.

Author(s)

Nathaniel E. Helwig <helwig@umn.edu>

References

Gu, C. (2013). Smoothing Spline ANOVA Models. 2nd Ed. New York, NY: Springer-Verlag. doi:10.1007/9781461453697

Wahba, G (1981). Spline interpolation and smoothing on the sphere. *SIAM Journal on Scientific Computing, 2*(1), 5-16. doi:10.1137/0902002

See Also

See `thinplate` for a thin plate spline basis and penalty.

Examples

```
################   standard parameterization   ########**########

# function with spherical predictors
set.seed(0)
n <- 1000
myfun <- function(x){
  sin(pi*x[,1]) + cos(2*pi*x[,2]) + cos(pi*x[,3])
  }
x3d <- cbind(runif(n), runif(n), runif(n)) - 0.5
x3d <- t(apply(x3d, 1, function(x) x / sqrt(sum(x^2))))
eta <- myfun(x3d)
y <- eta + rnorm(n, sd = 0.5)

# convert x latitude and longitude
x <- cbind(latitude = acos(x3d[,3]) - pi/2,
           longitude = atan2(x3d[,2], x3d[,1])) * (180 / pi)

# select first 100 points as knots
knots <- x[1:100,]

# cubic spherical spline basis
X <- basis.sph(x, knots, intercept = TRUE)

# cubic spherical spline penalty
Q <- penalty.sph(knots)

# pad Q with zeros (for intercept)
Q <- rbind(0, cbind(0, Q))

# define smoothing parameter
lambda <- 1e-5

# estimate coefficients
coefs <- psolve(crossprod(X) + n * lambda * Q) %*% crossprod(X, y)
```

```
# estimate eta
yhat <- X %*% coefs

# check rmse
sqrt(mean((eta - yhat)^2))



#################   ridge parameterization   ##################

# function with spherical predictors
set.seed(0)
n <- 1000
myfun <- function(x){
  sin(pi*x[,1]) + cos(2*pi*x[,2]) + cos(pi*x[,3])
  }
x3d <- cbind(runif(n), runif(n), runif(n)) - 0.5
x3d <- t(apply(x3d, 1, function(x) x / sqrt(sum(x^2))))
eta <- myfun(x3d)
y <- eta + rnorm(n, sd = 0.5)

# convert x latitude and longitude
x <- cbind(latitude = acos(x3d[,3]) - pi/2,
           longitude = atan2(x3d[,2], x3d[,1])) * (180 / pi)

# select first 100 points as knots
knots <- x[1:100,]

# cubic spherical spline basis
X <- basis.sph(x, knots, intercept = TRUE, ridge = TRUE)

# cubic spherical spline penalty (ridge)
Q <- diag(rep(c(0, 1), times = c(1, ncol(X) - 1)))

# define smoothing parameter
lambda <- 1e-5

# estimate coefficients
coefs <- psolve(crossprod(X) + n * lambda * Q) %*% crossprod(X, y)

# estimate eta
yhat <- X %*% coefs

# check rmse
sqrt(mean((eta - yhat)^2))
```

---

ss *Fit a Smoothing Spline*

Description

Fits a smoothing spline with the smoothing parameter selected via one of eight methods: GCV, OCV, GACV, ACV, REML, ML, AIC, or BIC.

Usage

```
ss(x, y = NULL, w = NULL, df, spar = NULL, lambda = NULL,
   method = c("GCV", "OCV", "GACV", "ACV", "REML", "ML", "AIC", "BIC"),
   m = 2L, periodic = FALSE, all.knots = FALSE, nknots = .nknots.smspl,
   knots = NULL, keep.data = TRUE, df.offset = 0, penalty = 1,
   control.spar = list(), tol = 1e-6 * IQR(x), bernoulli = TRUE,
   xmin = NULL, xmax = NULL, homosced = TRUE, iter.max = 1)
```

Arguments

x	Predictor vector of length n. Can also input a list or a two-column matrix specifying x and y.
y	Response vector of length n. If y is missing or NULL, the responses are assumed to be specified by x, with x the index vector.
w	Weights vector of length n. Defaults to all 1.
df	Equivalent degrees of freedom (trace of the smoother matrix). Must be in $[m, nx]$, where nx is the number of unique x values, see below.
spar	Smoothing parameter. Typically (but not always) in the range $(0, 1]$. If specified `lambda = 256^(3*(spar-1))`.
lambda	Computational smoothing parameter. This value is weighted by n to form the penalty coefficient (see Details). Ignored if spar is provided.
method	Method for selecting the smoothing parameter. Ignored if spar or lambda is provided.
m	Penalty order (integer). The penalty functional is the integrated squared m-th derivative of the function. Defaults to $m = 2$, which is a cubic smoothing spline. Set $m = 1$ for a linear smoothing spline or $m = 3$ for a quintic smoothing spline.
periodic	Logical. If TRUE, the estimated function $f(x)$ is constrained to be periodic, i.e., $f(a) = f(b)$ where $a = \min(x)$ and $b = \max(x)$.
all.knots	If TRUE, all distinct points in x are used as knots. If FALSE (default), a sequence knots is placed at the quantiles of the unique x values; in this case, the input nknots specifies the number of knots in the sequence. Ignored if the knot values are input using the knots argument.
nknots	Positive integer or function specifying the number of knots. Ignored if either all.knots = TRUE or the knot values are input using the knots argument.
knots	Vector of knot values for the spline. Should be unique and within the range of the x values (to avoid a warning).
keep.data	Logical. If TRUE, the original data as a part of the output object.
df.offset	Allows the degrees of freedom to be increased by df.offset in the GCV criterion.

penalty	The coefficient of the penalty for degrees of freedom in the GCV criterion.
control.spar	Optional list with named components controlling the root finding when the smoothing parameter spar is computed, i.e., missing or NULL, see below. **Note** that spar is only searched for in the interval $[lower, upper]$. **lower:** lower bound for spar; defaults to -1.5 **upper:** upper bound for spar; defaults to 1.5 **tol:** the absolute precision (**tol**erance) used by optimize; defaults to 1e-8.
tol	Tolerance for same-ness or uniqueness of the x values. The values are binned into bins of size tol and values which fall into the same bin are regarded as the same. Must be strictly positive (and finite).
bernoulli	If TRUE, scaled Bernoulli polynomials are used for the basis and penalty functions. If FALSE, produces the "classic" definition of a smoothing spline, where the function estimate is a piecewise polynomial function with pieces of degree $2m - 1$. See polynomial for details.
xmin	Minimum x value used to transform predictor scores to [0,1]. If NULL, xmin = min(x).
xmax	Maximum x value used to transform predictor scores to [0,1]. If NULL, xmax = max(x).
homosced	Are error variances homoscedastic? If FALSE, variance weights are (iteratively?) estimated from the data.
iter.max	Maximum number of iterations for variance weight estimation. Ignored if homosced = TRUE.

Details

Inspired by the smooth.spline function in R's **stats** package.

Neither x nor y are allowed to containing missing or infinite values.

The x vector should contain at least $2m$ distinct values. 'Distinct' here is controlled by tol: values which are regarded as the same are replaced by the first of their values and the corresponding y and w are pooled accordingly.

Unless lambda has been specified instead of spar, the computational λ used (as a function of spar) is $\lambda = 256^{3(s-1)}$, where $s =$ spar.

If spar and lambda are missing or NULL, the value of df is used to determine the degree of smoothing. If df is missing as well, the specified method is used to determine λ.

Letting $f_i = f(x_i)$, the function is represented as

$$f = X\beta + Z\alpha$$

where the basis functions in X span the null space (i.e., functions with m-th derivative of zero), and Z contains the reproducing kernel function of the contrast space evaluated at all combinations of observed data points and knots, i.e., $Z[i, j] = R(x_i, k_j)$ where R is the kernel function and k_j is the j-th knot. The vectors β and α contain unknown basis function coefficients. Letting $M = (X, Z)$ and $\gamma = (\beta', \alpha')'$, the penalized least squares problem has the form

$$(y - M\gamma)'W(y - M\gamma) + n\lambda\alpha' Q\alpha$$

where W is a diagonal matrix containg the weights, and Q is the penalty matrix. Note that $Q[i,j] = R(k_i, k_j)$ contains the reproducing kernel function evaluated at all combinations of knots. The optimal coefficients are the solution to

$$(M'WM + n\lambda P)\gamma = M'Wy$$

where P is the penalty matrix Q augmented with zeros corresponding to the β in γ.

Value

An object of class "ss" with components:

`x`	the distinct `x` values in increasing order; see Note.
`y`	the fitted values corresponding to `x`.
`w`	the weights used at the unique values of `x`.
`yin`	the `y` values used at the unique `y` values.
`tol`	the `tol` argument (whose default depends on `x`).
`data`	only if keep.data = TRUE: itself a list with components `x`, `y` and `w` (if applicable). These are the original $(x_i, y_i, w_i), i = 1, \ldots, n$, values where `data$x` may have repeated values and hence be longer than the above `x` component; see details.
`lev`	leverages, the diagonal values of the smoother matrix.
`cv.crit`	cross-validation score.
`pen.crit`	the penalized criterion, a non-negative number; simply the (weighted) residual sum of squares (RSS).
`crit`	the criterion value minimized in the underlying `df2lambda` function. When `df` is provided, the criterion is $[tr(S_\lambda) - df]^2$.
`df`	equivalent degrees of freedom used.
`df.residual`	the residual degrees of freedom = `nobs - df`
`spar`	the value of `spar` computed or given, i.e., $s = 1 + \log_{256}(\lambda)/3$
`lambda`	the value of λ corresponding to `spar`, i.e., $\lambda = 256^{3(s-1)}$.
`fit`	list for use by `predict.ss`, with components **n:** number of observations. **knot:** the knot sequence. **nk:** number of coefficients (# knots plus m). **coef:** coefficients for the spline basis used. **min, range:** numbers giving the corresponding quantities of `x` **m:** spline penalty order (same as input `m`) **periodic:** is spline periodic? **cov.sqrt** square root of covariance matrix of `coef` such that `tcrossprod(coef)` reconstructs the covariance matrix. **weighted** were weights `w` used in fitting? **df.offset** same as input **penalty** same as input

control.spar control parameters for smoothing parameter selection
bernoulli were Bernoulli polynomials used in fitting?

`call`	the matched call.
`sigma`	estimated error standard deviation.
`logLik`	log-likelihood (if `method` is REML or ML).
`aic`	Akaike's Information Criterion (if `method` is AIC).
`bic`	Bayesian Information Criterion (if `method` is BIC).
`penalty`	smoothness penalty $\alpha' Q \alpha$, which is the integrated squared m-th derivative of the estimated function $f(x)$.
`method`	smoothing parameter selection method. Will be `NULL` if `df`, `spar`, or `lambda` is provided.

Methods

The smoothing parameter can be selected using one of eight methods:
Generalized Cross-Validation (GCV)
Ordinary Cross-Validation (OCV)
Generalized Approximate Cross-Validation (GACV)
Approximate Cross-Validation (ACV)
Restricted Maximum Likelihood (REML)
Maximum Likelihood (ML)
Akaike's Information Criterion (AIC)
Bayesian Information Criterion (BIC)

Note

The number of unique x values, nx, are determined by the tol argument, equivalently to

```
nx <- sum(!duplicated( round((x - mean(x)) / tol) ))
```

In this case where not all unique x values are used as knots, the result is not a smoothing spline in the strict sense, but very close unless a small smoothing parameter (or large `df`) is used.

Author(s)

Nathaniel E. Helwig <helwig@umn.edu>

References

https://stat.ethz.ch/R-manual/R-devel/library/stats/html/smooth.spline.html

Berry, L. N., & Helwig, N. E. (2021). Cross-validation, information theory, or maximum likelihood? A comparison of tuning methods for penalized splines. *Stats, 4*(3), 701-724. doi:10.3390/stats4030042

Craven, P. and Wahba, G. (1979). Smoothing noisy data with spline functions: Estimating the correct degree of smoothing by the method of generalized cross-validation. *Numerische Mathematik, 31*, 377-403. doi:10.1007/BF01404567

Gu, C. (2013). Smoothing spline ANOVA models, 2nd edition. New York: Springer. doi:10.1007/9781461453697

Helwig, N. E. (2020). Multiple and Generalized Nonparametric Regression. In P. Atkinson, S. Delamont, A. Cernat, J. W. Sakshaug, & R. A. Williams (Eds.), *SAGE Research Methods Foundations.* doi:10.4135/9781526421036885885

Helwig, N. E. (2021). Spectrally sparse nonparametric regression via elastic net regularized smoothers. *Journal of Computational and Graphical Statistics, 30*(1), 182-191. doi:10.1080/10618600.2020.1806855

Wahba, G. (1985). A comparison of GCV and GML for choosing the smoothing parameters in the generalized spline smoothing problem. *The Annals of Statistics, 4*, 1378-1402. doi:10.1214/aos/1176349743

See Also

Related Modeling Functions:

`sm` for fitting smooth models with multiple predictors of mixed types (Gaussian response).

`gsm` for fitting generalized smooth models with multiple predictors of mixed types (non-Gaussian response).

S3 Methods and Related Functions for "ss" Objects:

`boot.ss` for bootstrapping `ss` objects.

`coef.ss` for extracting coefficients from `ss` objects.

`cooks.distance.ss` for calculating Cook's distances from `ss` objects.

`cov.ratio` for computing covariance ratio from `ss` objects.

`deviance.ss` for extracting deviance from `ss` objects.

`dfbeta.ss` for calculating DFBETA from `ss` objects.

`dfbetas.ss` for calculating DFBETAS from `ss` objects.

`diagnostic.plots` for plotting regression diagnostics from `ss` objects.

`fitted.ss` for extracting fitted values from `ss` objects.

`hatvalues.ss` for extracting leverages from `ss` objects.

`model.matrix.ss` for constructing model matrix from `ss` objects.

`plot.ss` for plotting predictions from `ss` objects.

`plot.boot.ss` for plotting `boot.ss` objects.

`predict.ss` for predicting from `ss` objects.

`residuals.ss` for extracting residuals from `ss` objects.

`rstandard.ss` for computing standardized residuals from `ss` objects.

`rstudent.ss` for computing studentized residuals from `ss` objects.

`smooth.influence` for calculating basic influence information from `ss` objects.

`smooth.influence.measures` for convenient display of influential observations from `ss` objects.

`summary.ss` for summarizing `ss` objects.

`vcov.ss` for extracting coefficient covariance matrix from `ss` objects.

`weights.ss` for extracting prior weights from `ss` objects.

Examples

```
# generate data
set.seed(1)
n <- 100
x <- seq(0, 1, length.out = n)
fx <- 2 + 3 * x + sin(2 * pi * x)
y <- fx + rnorm(n, sd = 0.5)

# GCV selection (default)
ss.GCV <- ss(x, y, nknots = 10)
ss.GCV

# OCV selection
ss.OCV <- ss(x, y, method = "OCV", nknots = 10)
ss.OCV

# GACV selection
ss.GACV <- ss(x, y, method = "GACV", nknots = 10)
ss.GACV

# ACV selection
ss.ACV <- ss(x, y, method = "ACV", nknots = 10)
ss.ACV

# ML selection
ss.ML <- ss(x, y, method = "ML", nknots = 10)
ss.ML

# REML selection
ss.REML <- ss(x, y, method = "REML", nknots = 10)
ss.REML

# AIC selection
ss.AIC <- ss(x, y, method = "AIC", nknots = 10)
ss.AIC

# BIC selection
ss.BIC <- ss(x, y, method = "BIC", nknots = 10)
ss.BIC

# compare results
mean( ( fx - ss.GCV$y )^2 )
mean( ( fx - ss.OCV$y )^2 )
mean( ( fx - ss.GACV$y )^2 )
mean( ( fx - ss.ACV$y )^2 )
mean( ( fx - ss.ML$y )^2 )
mean( ( fx - ss.REML$y )^2 )
mean( ( fx - ss.AIC$y )^2 )
mean( ( fx - ss.BIC$y )^2 )

# plot results
plot(x, y)
```

```
rlist <- list(ss.GCV, ss.OCV, ss.GACV, ss.ACV,
              ss.REML, ss.ML, ss.AIC, ss.BIC)
for(j in 1:length(rlist)){
   lines(rlist[[j]], lwd = 2, col = j)
}
```

---

summary *Summary methods for Fit Models*

Description

summary methods for object classes "gsm", "sm", and "ss".

Usage

```
## S3 method for class 'gsm'
summary(object, ...)

## S3 method for class 'sm'
summary(object, ...)

## S3 method for class 'ss'
summary(object, ...)

## S3 method for class 'summary.gsm'
print(x, digits = max(3, getOption("digits") - 3),
      signif.stars = getOption("show.signif.stars"), ...)

## S3 method for class 'summary.sm'
print(x, digits = max(3, getOption("digits") - 3),
      signif.stars = getOption("show.signif.stars"), ...)

## S3 method for class 'summary.ss'
print(x, digits = max(3, getOption("digits") - 3),
      signif.stars = getOption("show.signif.stars"), ...)
```

Arguments

object	an object of class "gsm" output by the gsm function, "sm" output by the sm function, or "ss" output by the ss function
x	an object of class "summary.gsm" output by the summary.gsm function, "summary.sm" output by the summary.sm function, or "summary.ss" output by the summary.ss function.
digits	the minimum number of significant digits to be printed in values.
signif.stars	logical. If TRUE, 'significance stars' are printed for each coefficient.
...	additional arguments affecting the summary produced (currently ignored).

Details

Summary includes information for assessing the statistical and practical significance of the model terms.

Statistical inference is conducted via (approximate) frequentist chi-square tests using the Bayesian interpretation of a smoothing spline (Nychka, 1988; Wahba, 1983).

With multiple smooth terms included in the model, the inferential results may (and likely will) differ slightly depending on the `tprk` argument (when using the `gsm` and `sm` functions).

If significance testing is of interest, the `tprk = FALSE` option may be desirable, given that this allows for unique basis function coefficients for each model term.

In all cases, the inferential results are based on a (pseudo) F or chi-square statistic which fails to consider the uncertainty of the smoothing parameter estimation.

Value

`residuals`	the deviance residuals.
`fstatistic`	the F statistic for testing all effects (parametric and smooth).
`dev.expl`	the explained deviance.
`p.table`	the coefficient table for (approximate) inference on the parametric terms.
`s.table`	the coefficient table for (approximate) inference on the smooth terms.
`dispersion`	the estimate of the dispersion parameter.
`r.squared`	the observed coefficient of multiple determination.
`adj.r.squared`	the adjusted coefficient of multiple determination.
`kappa`	the collinearity indices, i.e., square-roots of the variance inflation factors (see `varinf`). A value of 1 indicates no collinearity, and higher values indicate more collinearity of a given term with other model terms.
`pi`	the importance indices. Larger values indicate more importance, and the values satisfy `sum(pi) = 1`. Note that elements of `pi` can be negative.
`call`	the original function call.
`family`	the specified family (for gsm objects).

Author(s)

Nathaniel E. Helwig <helwig@umn.edu>

References

Helwig, N. E. (2020). Multiple and Generalized Nonparametric Regression. In P. Atkinson, S. Delamont, A. Cernat, J. W. Sakshaug, & R. A. Williams (Eds.), *SAGE Research Methods Foundations.* doi:10.4135/9781526421036885885

Nychka, D. (1988). Bayesian confience intervals for smoothing splines. *Journal of the American Statistical Association, 83(404)*, 1134-1143. doi:10.2307/2290146

Wahba, G. (1983). Bayesian "confidence intervals" for the cross-validated smoothing spline. *Journal of the Royal Statistical Society. Series B, 45(1)*, 133-150. doi:10.1111/j.25176161.1983.tb01239.x

See Also

gsm, sm, and ss

Examples

```
### Example 1: gsm

# generate data
set.seed(1)
n <- 1000
x <- seq(0, 1, length.out = n)
z <- factor(sample(letters[1:3], size = n, replace = TRUE))
fun <- function(x, z){
  mu <- c(-2, 0, 2)
  zi <- as.integer(z)
  fx <- mu[zi] + 3 * x + sin(2 * pi * x + mu[zi]*pi/4)
}
fx <- fun(x, z)
y <- rbinom(n = n, size = 1, p = 1 / (1 + exp(-fx)))

# define marginal knots
probs <- seq(0, 0.9, by = 0.1)
knots <- list(x = quantile(x, probs = probs),
              z = letters[1:3])

# fit sm with specified knots (tprk = TRUE)
gsm.ssa <- gsm(y ~ x * z, family = binomial, knots = knots)
summary(gsm.ssa)

# fit sm with specified knots (tprk = FALSE)
gsm.gam <- gsm(y ~ x * z, family = binomial, knots = knots, tprk = FALSE)
summary(gsm.gam)


### Example 2: sm

# generate data
set.seed(1)
n <- 100
x <- seq(0, 1, length.out = n)
z <- factor(sample(letters[1:3], size = n, replace = TRUE))
fun <- function(x, z){
  mu <- c(-2, 0, 2)
  zi <- as.integer(z)
  fx <- mu[zi] + 3 * x + sin(2 * pi * x + mu[zi]*pi/4)
}
fx <- fun(x, z)
y <- fx + rnorm(n, sd = 0.5)

# define marginal knots
probs <- seq(0, 0.9, by = 0.1)
knots <- list(x = quantile(x, probs = probs),
```

```
              z = letters[1:3])

# fit sm with specified knots (tprk = TRUE)
sm.ssa <- sm(y ~ x * z, knots = knots)
summary(sm.ssa)

# fit sm with specified knots (tprk = FALSE)
sm.gam <- sm(y ~ x * z, knots = knots, tprk = FALSE)
summary(sm.gam)


### Example 3: ss

# generate data
set.seed(1)
n <- 100
x <- seq(0, 1, length.out = n)
fx <- 2 + 3 * x + sin(2 * pi * x)
y <- fx + rnorm(n, sd = 0.5)

# regular smoothing spline
ss.reg <- ss(x, y, nknots = 10)
summary(ss.reg)
```

`theta.mle` *MLE of Theta for Negative Binomial*

Description

Computes the maximum likelihood estimate of the size (theta) parameter for the Negative Binomial distribution via a Newton-Raphson algorithm.

Usage

```
theta.mle(y, mu, theta, wt = 1,
          maxit = 100, maxth = .Machine$double.xmax,
          tol = .Machine$double.eps^0.5)
```

Arguments

`y`	response vector
`mu`	mean vector
`theta`	initial theta (optional)
`wt`	weight vector
`maxit`	max number of iterations
`maxth`	max possible value of `theta`
`tol`	convergence tolerance

Details

Based on the `glm.nb` function in the **MASS** package. If `theta` is missing, the initial estimate of theta is given by

`theta <- 1 / mean(wt * (y / mu - 1)^2)`

which is motivated by the method of moments estimator for the dispersion parameter in a quasi-Poisson model.

Value

Returns estimated theta with attributes

`SE`	standard error estimate
`iter`	number of iterations

Author(s)

Nathaniel E. Helwig <helwig@umn.edu>

References

Venables, W. N. and Ripley, B. D. (1999) Modern Applied Statistics with S-PLUS. Third Edition. Springer.

https://www.rdocumentation.org/packages/MASS/versions/7.3-51.6/topics/negative.binomial

https://www.rdocumentation.org/packages/MASS/versions/7.3-51.6/topics/glm.nb

See Also

`NegBin` for details on the Negative Binomial distribution

Examples

```
# generate data
n <- 1000
x <- seq(0, 1, length.out = n)
fx <- 3 * x + sin(2 * pi * x) - 1.5
mu <- exp(fx)

# simulate negative binomial data
set.seed(1)
y <- rnbinom(n = n, size = 1/2, mu = mu)

# estimate theta
theta.mle(y, mu)
```

`thinplate` *Thin Plate Spline Basis and Penalty*

Description

Generate the smoothing spline basis and penalty matrix for a thin plate spline.

Usage

```
basis.tps(x, knots, m = 2, rk = TRUE, intercept = FALSE, ridge = FALSE)

penalty.tps(x, m = 2, rk = TRUE)
```

Arguments

`x`	Predictor variables (basis) or spline knots (penalty). Numeric or integer vector of length n, or matrix of dimension n by p.
`knots`	Spline knots. Numeric or integer vector of length r, or matrix of dimension r by p.
`m`	Penalty order. "m=1" for linear thin plate spline, "m=2" for cubic, and "m=3" for quintic. Must satisfy $2m > p$.
`rk`	If true (default), the reproducing kernel parameterization is used. Otherwise, the classic thin plate basis is returned.
`intercept`	If `TRUE`, the first column of the basis will be a column of ones.
`ridge`	If `TRUE`, the basis matrix is post-multiplied by the inverse square root of the penalty matrix. Only applicable if `rk = TRUE`. See Note and Examples.

Details

Generates a basis function or penalty matrix used to fit linear, cubic, and quintic thin plate splines.

The basis function matrix has the form

$$X = [X_0, X_1]$$

where the matrix `X_0` is of dimension n by $M - 1$ (plus 1 if an intercept is included) where $M = \binom{p+m-1}{p}$, and `X_1` is a matrix of dimension n by r.

The `X_0` matrix contains the "parametric part" of the basis, which includes polynomial functions of the columns of `x` up to degree $m - 1$ (and potentially interactions).

The matrix `X_1` contains the "nonparametric part" of the basis.

If `rk = TRUE`, the matrix `X_1` consists of the *reproducing kernel* function

$$\rho(x, y) = (I - P_x)(I - P_y)E(|x - y|)$$

evaluated at all combinations of `x` and `knots`. Note that P_x and P_y are projection operators, $|.|$ denotes the Euclidean distance, and the TPS semi-kernel is defined as

$$E(z) = \alpha z^{2m-p} \log(z)$$

if p is even and

$$E(z) = \beta z^{2m-p}$$

otherwise, where α and β are positive constants (see References).

If `rk = FALSE`, the matrix `X_1` contains the TPS semi-kernel $E(.)$ evaluated at all combinations of `x` and `knots`. Note: the TPS semi-kernel is *not* positive (semi-)definite, but the projection is.

If `rk = TRUE`, the penalty matrix consists of the *reproducing kernel* function

$$\rho(x, y) = (I - P_x)(I - P_y)E(|x - y|)$$

evaluated at all combinations of `x`. If `rk = FALSE`, the penalty matrix contains the TPS semi-kernel $E(.)$ evaluated at all combinations of `x`.

Value

Basis: Matrix of dimension `c(length(x), df)` where `df = nrow(as.matrix(knots)) + choose(p + m - 1, p) - !intercept` and `p = ncol(as.matrix(x))`.

Penalty: Matrix of dimension `c(r, r)` where `r = nrow(as.matrix(x))` is the number of knots.

Note

The inputs `x` and `knots` must have the same dimension.

If `rk = TRUE` and `ridge = TRUE`, the penalty matrix is the identity matrix.

Author(s)

Nathaniel E. Helwig <helwig@umn.edu>

References

Gu, C. (2013). Smoothing Spline ANOVA Models. 2nd Ed. New York, NY: Springer-Verlag. doi:10.1007/9781461453697

Helwig, N. E. (2017). Regression with ordered predictors via ordinal smoothing splines. *Frontiers in Applied Mathematics and Statistics, 3*(15), 1-13. doi:10.3389/fams.2017.00015

Helwig, N. E., & Ma, P. (2015). Fast and stable multiple smoothing parameter selection in smoothing spline analysis of variance models with large samples. *Journal of Computational and Graphical Statistics, 24*(3), 715-732. doi:10.1080/10618600.2014.926819

See Also

See `polynomial` for a basis and penalty for numeric variables.

See `spherical` for a basis and penalty for spherical variables.

Examples

```
#################   standard parameterization   #################

# generate data
set.seed(0)
n <- 101
x <- seq(0, 1, length.out = n)
knots <- seq(0, 0.95, by = 0.05)
eta <- 1 + 2 * x + sin(2 * pi * x)
y <- eta + rnorm(n, sd = 0.5)

# cubic thin plate spline basis
X <- basis.tps(x, knots, intercept = TRUE)

# cubic thin plate spline penalty
Q <- penalty.tps(knots)

# pad Q with zeros (for intercept and linear effect)
Q <- rbind(0, 0, cbind(0, 0, Q))

# define smoothing parameter
lambda <- 1e-5

# estimate coefficients
coefs <- psolve(crossprod(X) + n * lambda * Q) %*% crossprod(X, y)

# estimate eta
yhat <- X %*% coefs

# check rmse
sqrt(mean((eta - yhat)^2))

# plot results
plot(x, y)
lines(x, yhat)



#################   ridge parameterization   ##################

# generate data
set.seed(0)
n <- 101
x <- seq(0, 1, length.out = n)
knots <- seq(0, 0.95, by = 0.05)
eta <- 1 + 2 * x + sin(2 * pi * x)
y <- eta + rnorm(n, sd = 0.5)

# cubic thin plate spline basis
X <- basis.tps(x, knots, intercept = TRUE, ridge = TRUE)

# cubic thin plate spline penalty (ridge)
```

```
Q <- diag(rep(c(0, 1), times = c(2, ncol(X) - 2)))

# define smoothing parameter
lambda <- 1e-5

# estimate coefficients
coefs <- psolve(crossprod(X) + n * lambda * Q) %*% crossprod(X, y)

# estimate eta
yhat <- X %*% coefs

# check rmse
sqrt(mean((eta - yhat)^2))

# plot results
plot(x, y)
lines(x, yhat)
```

`varimp` *Variable Importance Indices*

Description

Computes variable importance indices for terms of a smooth model.

Usage

```
varimp(object, newdata = NULL, combine = TRUE)
```

Arguments

`object`	an object of class "sm" output by the `sm` function or an object of class "gsm" output by the `gsm` function.
`newdata`	the data used for variable importance calculation (if `NULL` training data are used).
`combine`	a switch indicating if the parametric and smooth components of the importance should be combined (default) or returned separately.

Details

Suppose that the function can be written as

$$\eta = \eta_0 + \eta_1 + \eta_2 + ... + \eta_p$$

where η_0 is a constant (intercept) term, and η_j denotes the j-th effect function, which is assumed to have mean zero. Note that η_j could be a main or interaction effect function for all $j = 1, ..., p$.

The variable importance index for the j-th effect term is defined as

$$\pi_j = (\eta_j^\top \eta_*)/(\eta_*^\top \eta_*)$$

where $\eta_* = \eta_1 + \eta_2 + ... + \eta_p$. Note that $\sum_{j=1}^{p} \pi_j = 1$ but there is no guarantee that $\pi_j > 0$.

If all π_j are non-negative, then π_j gives the proportion of the model's R-squared that can be accounted for by the j-th effect term. Thus, values of π_j closer to 1 indicate that η_j is more important, whereas values of π_j closer to 0 (including negative values) indicate that η_j is less important.

Value

If `combine = TRUE`, returns a named vector containing the importance indices for each effect function (in `object$terms`).

If `combine = FALSE`, returns a data frame where the first column gives the importance indices for the parametric components and the second column gives the importance indices for the smooth (nonparametric) components.

Note

When `combine = FALSE`, importance indices will be equal to zero for non-existent components of a model term. For example, a `nominal` effect does not have a parametric component, so the `$p` component of the importance index for a nominal effect will be zero.

Author(s)

Nathaniel E. Helwig <helwig@umn.edu>

References

Gu, C. (2013). Smoothing spline ANOVA models, 2nd edition. New York: Springer. doi:10.1007/9781461453697

Helwig, N. E. (2020). Multiple and Generalized Nonparametric Regression. In P. Atkinson, S. Delamont, A. Cernat, J. W. Sakshaug, & R. A. Williams (Eds.), SAGE Research Methods Foundations. doi:10.4135/9781526421036885885

See Also

See `summary.sm` for more thorough summaries of smooth models.

See `summary.gsm` for more thorough summaries of generalized smooth models.

Examples

```
##########   EXAMPLE 1   ##########
### 1 continuous and 1 nominal predictor

# generate data
set.seed(1)
n <- 100
x <- seq(0, 1, length.out = n)
z <- factor(sample(letters[1:3], size = n, replace = TRUE))
fun <- function(x, z){
  mu <- c(-2, 0, 2)
```

```
  zi <- as.integer(z)
  fx <- mu[zi] + 3 * x + sin(2 * pi * x)
}
fx <- fun(x, z)
y <- fx + rnorm(n, sd = 0.5)

# define marginal knots
probs <- seq(0, 0.9, by = 0.1)
knots <- list(x = quantile(x, probs = probs),
              z = letters[1:3])

# fit correct (additive) model
sm.add <- sm(y ~ x + z, knots = knots)

# fit incorrect (interaction) model
sm.int <- sm(y ~ x * z, knots = knots)

# true importance indices
eff <- data.frame(x = 3 * x + sin(2 * pi * x), z = c(-2, 0, 2)[as.integer(z)])
eff <- scale(eff, scale = FALSE)
fstar <- rowSums(eff)
colSums(eff * fstar) / sum(fstar^2)

# estimated importance indices
varimp(sm.add)
varimp(sm.int)



##########   EXAMPLE 2   ##########
### 4 continuous predictors
### additive model

# generate data
set.seed(1)
n <- 100
fun <- function(x){
  sin(pi*x[,1]) + sin(2*pi*x[,2]) + sin(3*pi*x[,3]) + sin(4*pi*x[,4])
}
data <- as.data.frame(replicate(4, runif(n)))
colnames(data) <- c("x1v", "x2v", "x3v", "x4v")
fx <- fun(data)
y <- fx + rnorm(n)

# define ssa knot indices
knots.indx <- c(bin.sample(data$x1v, nbin = 10, index.return = TRUE)$ix,
                bin.sample(data$x2v, nbin = 10, index.return = TRUE)$ix,
                bin.sample(data$x3v, nbin = 10, index.return = TRUE)$ix,
                bin.sample(data$x4v, nbin = 10, index.return = TRUE)$ix)

# fit correct (additive) model
sm.add <- sm(y ~ x1v + x2v + x3v + x4v, data = data, knots = knots.indx)
```

```
# fit incorrect (interaction) model
sm.int <- sm(y ~ x1v * x2v + x3v + x4v, data = data, knots = knots.indx)

# true importance indices
eff <- data.frame(x1v = sin(pi*data[,1]), x2v = sin(2*pi*data[,2]),
                  x3v = sin(3*pi*data[,3]), x4v = sin(4*pi*data[,4]))
eff <- scale(eff, scale = FALSE)
fstar <- rowSums(eff)
colSums(eff * fstar) / sum(fstar^2)

# estimated importance indices
varimp(sm.add)
varimp(sm.int)
```

`varinf` *Variance Inflation Factors*

Description

Computes variance inflation factors for terms of a smooth model.

Usage

```
varinf(object, newdata = NULL)
```

Arguments

`object`	an object of class "sm" output by the `sm` function or an object of class "gsm" output by the `gsm` function.
`newdata`	the data used for variance inflation calculation (if `NULL` training data are used).

Details

Let κ_j^2 denote the VIF for the j-th model term.

Values of κ_j^2 close to 1 indicate no multicollinearity issues for the j-th term. Larger values of κ_j^2 indicate that η_j has more collinearity with other terms.

Thresholds of $\kappa_j^2 > 5$ or $\kappa_j^2 > 10$ are typically recommended for determining if multicollinearity is too much of an issue.

To understand these thresholds, note that

$$\kappa_j^2 = \frac{1}{1 - R_j^2}$$

where R_j^2 is the R-squared for the linear model predicting η_j from the remaining model terms.

Value

a named vector containing the variance inflation factors for each effect function (in `object$terms`).

Note

Suppose that the function can be written as

$$\eta = \eta_0 + \eta_1 + \eta_2 + ... + \eta_p$$

where η_0 is a constant (intercept) term, and η_j denotes the j-th effect function, which is assumed to have mean zero. Note that η_j could be a main or interaction effect function for all $j = 1, ..., p$.

Defining the $p \times p$ matrix C with entries

$$C_{jk} = \cos(\eta_j, \eta_k)$$

where the cosine is defined with respect to the training data, i.e.,

$$\cos(\eta_j, \eta_k) = \frac{\sum_{i=1}^{n} \eta_j(x_i)\eta_k(x_i)}{\sqrt{\sum_{i=1}^{n} \eta_j^2(x_i)}\sqrt{\sum_{i=1}^{n} \eta_k^2(x_i)}}$$

The variane inflation factors are the diagonal elements of C^{-1}, i.e.,

$$\kappa_j^2 = C^{jj}$$

where κ_j^2 is the VIF for the j-th term, and C^{jj} denotes the j-th diagonal element of the matrix C^{-1}.

Author(s)

Nathaniel E. Helwig <helwig@umn.edu>

References

Gu, C. (2013). Smoothing spline ANOVA models, 2nd edition. New York: Springer. doi:10.1007/9781461453697

Helwig, N. E. (2020). Multiple and Generalized Nonparametric Regression. In P. Atkinson, S. Delamont, A. Cernat, J. W. Sakshaug, & R. A. Williams (Eds.), SAGE Research Methods Foundations. doi:10.4135/9781526421036885885

See Also

See `summary.sm` for more thorough summaries of smooth models.

See `summary.gsm` for more thorough summaries of generalized smooth models.

Examples

```
##########   EXAMPLE 1   ##########
### 4 continuous predictors
### no multicollinearity

# generate data
set.seed(1)
n <- 100
fun <- function(x){
```

```
  sin(pi*x[,1]) + sin(2*pi*x[,2]) + sin(3*pi*x[,3]) + sin(4*pi*x[,4])
}
data <- as.data.frame(replicate(4, runif(n)))
colnames(data) <- c("x1v", "x2v", "x3v", "x4v")
fx <- fun(data)
y <- fx + rnorm(n)

# fit model
mod <- sm(y ~ x1v + x2v + x3v + x4v, data = data, tprk = FALSE)

# check vif
varinf(mod)


##########   EXAMPLE 2   ##########
### 4 continuous predictors
### multicollinearity

# generate data
set.seed(1)
n <- 100
fun <- function(x){
  sin(pi*x[,1]) + sin(2*pi*x[,2]) + sin(3*pi*x[,3]) + sin(3*pi*x[,4])
}
data <- as.data.frame(replicate(3, runif(n)))
data <- cbind(data, c(data[1,2], data[2:n,3]))
colnames(data) <- c("x1v", "x2v", "x3v", "x4v")
fx <- fun(data)
y <- fx + rnorm(n)

# check collinearity
cor(data)
cor(sin(3*pi*data[,3]), sin(3*pi*data[,4]))

# fit model
mod <- sm(y ~ x1v + x2v + x3v + x4v, data = data, tprk = FALSE)

# check vif
varinf(mod)
```

---

vcov *Calculate Variance-Covariance Matrix for a Fitted Smooth Model*

Description

Returns the variance-covariance matrix for the basis function coefficients from a fit smoothing spline (fit by ss), smooth model (fit by sm), or generalized smooth model (fit by gsm).

Usage

```
## S3 method for class 'ss'
vcov(object, ...)

## S3 method for class 'sm'
vcov(object, ...)

## S3 method for class 'gsm'
vcov(object, ...)
```

Arguments

object	an object of class "gsm" output by the gsm function, "sm" output by the sm function, or "ss" output by the ss function
...	other arugments (currently ignored)

Details

The variance-covariance matrix is calculated using the Bayesian interpretation of a smoothing spline. Unlike the classic treatments (e.g., Wahba, 1983; Nychka, 1988), which interpret the smoothing spline as a Bayesian estimate of a Gaussian process, this treatment applies the Bayesian interpretation directly on the coefficient vector. More specifically, the smoothing spline basis function coefficients are interpreted as Bayesian estimates of the basis function coefficients (see Helwig, 2020).

Value

Returns the (symmetric) matrix such that cell (i, j) contains the covariance between the i-th and j-th elements of the coefficient vector.

Author(s)

Nathaniel E. Helwig <helwig@umn.edu>

References

Helwig, N. E. (2020). Multiple and Generalized Nonparametric Regression. In P. Atkinson, S. Delamont, A. Cernat, J. W. Sakshaug, & R. A. Williams (Eds.), SAGE Research Methods Foundations. doi:10.4135/9781526421036885885

Nychka, D. (1988). Bayesian confience intervals for smoothing splines. *Journal of the American Statistical Association, 83(404)*, 1134-1143. doi:10.2307/2290146

Wahba, G. (1983). Bayesian "confidence intervals" for the cross-validated smoothing spline. *Journal of the Royal Statistical Society. Series B, 45(1)*, 133-150. doi:10.1111/j.25176161.1983.tb01239.x

See Also

ss, sm, gsm for model fitting

boot.ss, boot.sm, boot.gsm for bootstrapping

Examples

```
## for 'ss' objects this function is defined as
  function(object, ...){
    Sigma <- tcrossprod(object$fit$cov.sqrt)
    rownames(Sigma) <- colnames(Sigma) <- names(object$fit$coef)
    Sigma
  }

## for 'sm' and 'gsm' objects this function is defined as
  function(object, ...){
    Sigma <- tcrossprod(object$cov.sqrt)
    rownames(Sigma) <- colnames(Sigma) <- names(object$coefficients)
    Sigma
  }
```

---

weights *Extract Smooth Model Weights*

Description

Extracts prior weights from a fit smoothing spline (fit by ss), smooth model (fit by sm), or generalized smooth model (fit by gsm).

Usage

```
## S3 method for class 'ss'
weights(object, ...)

## S3 method for class 'sm'
weights(object, ...)

## S3 method for class 'gsm'
weights(object, ...)
```

Arguments

object	an object of class "gsm" output by the gsm function, "sm" output by the sm function, or "ss" output by the ss function
...	other arugments (currently ignored)

Details

Returns the "prior weights", which are user-specified via the w argument (of the ss function) or the weights argument (of the sm and gsm functions). If no prior weights were supplied, returns the (default) unit weights, i.e., rep(1, nobs).

Value

Prior weights extracted from object

Author(s)

Nathaniel E. Helwig <helwig@umn.edu>

References

Chambers, J. M. and Hastie, T. J. (1992) *Statistical Models in S*. Wadsworth & Brooks/Cole.

Helwig, N. E. (2020). Multiple and Generalized Nonparametric Regression. In P. Atkinson, S. Delamont, A. Cernat, J. W. Sakshaug, & R. A. Williams (Eds.), SAGE Research Methods Foundations. doi:10.4135/9781526421036885885

See Also

ss, sm, gsm

Examples

```
# generate weighted data
set.seed(1)
n <- 100
x <- seq(0, 1, length.out = n)
w <- rep(5:15, length.out = n)
fx <- 2 + 3 * x + sin(2 * pi * x)
y <- fx + rnorm(n, sd = 0.5 / sqrt(w))

# smoothing spline
mod.ss <- ss(x, y, w, nknots = 10)
w.ss <- weights(mod.ss)

# smooth model
mod.sm <- sm(y ~ x, weights = w, knots = 10)
w.sm <- weights(mod.sm)

# generalized smooth model (family = gaussian)
mod.gsm <- gsm(y ~ x, weights = w, knots = 10)
w.gsm <- weights(mod.gsm)

# note: weights are internally rescaled such as
w0 <- w / mean(w)
max(abs(w0 - w.ss))
max(abs(w0 - w.sm))
max(abs(w0 - w.gsm))
```

---

wtd.mean *Weighted Arithmetic Mean*

Description

Generic function for calculating the weighted (and possibly trimmed) arithmetic mean.

Usage

```
wtd.mean(x, weights, trim = 0, na.rm = FALSE)
```

Arguments

x	Numerical or logical vector.
weights	Vector of non-negative weights.
trim	Fraction [0, 0.5) of observations trimmed from each end before calculating mean.
na.rm	Logical indicating whether NA values should be removed before calculation.

Details

If weights are missing, the weights are defined to be a vector of ones (which is the same as the unweighted arithmetic mean).

If trim is non-zero, then trim observations are deleted from each end before the (weighted) mean is computed. The quantiles used for trimming are defined using the wtd.quantile function.

Value

Returns the weighted and/or trimmed arithmetic mean.

Note

The weighted (and possible trimmed) mean is defined as:

sum(weights * x) / sum(weights)

where x is the (possibly trimmed version of the) input data.

Author(s)

Nathaniel E. Helwig <helwig@umn.edu>

See Also

wtd.var for weighted variance calculations

wtd.quantile for weighted quantile calculations

Examples

```
# generate data and weights
set.seed(1)
x <- rnorm(10)
w <- rpois(10, lambda = 10)

# weighted mean
wtd.mean(x, w)
sum(x * w) / sum(w)
```

```
# trimmed mean
q <- quantile(x, probs = c(0.1, 0.9), type = 4)
i <- which(x < q[1] | x > q[2])
mean(x[-i])
wtd.mean(x, trim = 0.1)

# weighted and trimmed mean
q <- wtd.quantile(x, w, probs = c(0.1, 0.9))
i <- which(x < q[1] | x > q[2])
wtd.mean(x[-i], w[-i])
wtd.mean(x, w, trim = 0.1)
```

`wtd.quantile` *Weighted Quantiles*

Description

Generic function for calculating weighted quantiles.

Usage

```
wtd.quantile(x, weights, probs = seq(0, 1, 0.25),
             na.rm = FALSE, names = TRUE)
```

Arguments

`x`	Numerical or logical vector.
`weights`	Vector of non-negative weights.
`probs`	Numeric vector of probabilities with values in [0,1].
`na.rm`	Logical indicating whether `NA` values should be removed before calculation.
`names`	Logical indicating if the result should have names corresponding to the probabilities.

Details

If `weights` are missing, the weights are defined to be a vector of ones (which is the same as the unweighted quantiles).

The weighted quantiles are computed by linearly interpolating the empirical cdf via the `approx` function.

Value

Returns the weighted quantiles corresponding to the input probabilities.

Note

If the weights are all equal (or missing), the resulting quantiles are equivalent to those produced by the quantile function using the 'type = 4' argument.

Author(s)

Nathaniel E. Helwig <helwig@umn.edu>

See Also

wtd.mean for weighted mean calculations

wtd.var for weighted variance calculations

Examples

```
# generate data and weights
set.seed(1)
x <- rnorm(10)
w <- rpois(10, lambda = 10)

# unweighted quantiles
quantile(x, probs = c(0.1, 0.9), type = 4)
wtd.quantile(x, probs = c(0.1, 0.9))

# weighted quantiles
sx <- sort(x, index.return = TRUE)
sw <- w[sx$ix]
ecdf <- cumsum(sw) / sum(sw)
approx(x = ecdf, y = sx$x, xout = c(0.1, 0.9), rule = 2)$y
wtd.quantile(x, w, probs = c(0.1, 0.9))
```

---

wtd.var *Weighted Variance and Standard Deviation*

Description

Generic function for calculating weighted variance or standard deviation of a vector.

Usage

```
wtd.var(x, weights, na.rm = FALSE)

wtd.sd(x, weights, na.rm = FALSE)
```

Arguments

x	Numerical or logical vector.
weights	Vector of non-negative weights.
na.rm	Logical indicating whether NA values should be removed before calculation.

Details

The weighted variance is defined as

```
(n / (n - 1)) * sum(weights * (x - xbar)^2) / sum(weights)
```

where n is the number of observations with non-zero weights, and xbar is the weighted mean computed via the wtd.mean function.

The weighted standard deviation is the square root of the weighted variance.

Value

Returns the weighted variance or standard deviation.

Note

If weights are missing, the weights are defined to be a vector of ones (which is the same as the unweighted variance or standard deviation).

Author(s)

Nathaniel E. Helwig <helwig@umn.edu>

See Also

wtd.mean for weighted mean calculations

wtd.quantile for weighted quantile calculations

Examples

```
# generate data and weights
set.seed(1)
x <- rnorm(10)
w <- rpois(10, lambda = 10)

# weighted mean
xbar <- wtd.mean(x, w)

# weighted variance
wtd.var(x, w)
(10 / 9) * sum(w * (x - xbar)^2) / sum(w)

# weighted standard deviation
wtd.sd(x, w)
sqrt((10 / 9) * sum(w * (x - xbar)^2) / sum(w))
```

Index