Package ‘scam’

June 14, 2019

Version 1.2-4

Author Natalya Pya <nat.pya@gmail.com>

Maintainer Natalya Pya <nat.pya@gmail.com>

Title Shape Constrained Additive Models

Date 2019-06-12

Description Routines for generalized additive modelling under shape constraints on the component functions of the linear predictor (Pya and Wood, 2015) <doi:10.1007/s11222-013-9448-7>. Models can contain multiple shape constrained (univariate and/or bivariate) and unconstrained terms. The routines of gam() in package 'mgcv' are used for setting up the model matrix, printing and plotting the results. Penalized likelihood maximization based on Newton-Raphson method is used to fit a model with multiple smoothing parameter selection by GCV or UBRE/AIC.

Depends R (>= 2.15.0), mgcv (>= 1.8-2)

Imports methods, stats, graphics, Matrix, splines

Suggests nlme

License GPL (>= 2)

LazyLoad yes

NeedsCompilation no

Repository CRAN

Date/Publication 2019-06-14 11:40:07 UTC

R topics documented:

`scam-package` *Shape Constrained Additive Models*

Description

`scam` provides functions for generalized additive modelling under shape constraints on the component functions of the linear predictor of the GAM. Models can contain multiple shape constrained and unconstrained terms as well as bivariate smooths with double or single monotonicity. Univariate smooths under eight possible shape constraints such as monotonically increasing/decreasing, convex/concave, increasing/decreasing and convex, increasing/decreasing and concave, are available as model terms.

The model set up is the same as in `gam()` in package `mgcv` with the added shape constrained smooths, so the unconstrained smooths can be of more than one variable, and other user defined smooths can be included. Penalized log likelihood maximization is used to fit the model together with the automatic smoothness selection.

Details

Package:	scam
Version:	1.2-4
Author:	Natalya Pya <nat.pya@gmail.com>
Maintainer:	Natalya Pya <nat.pya@gmail.com>
Title:	Shape Constrained Additive Models
Date:	2019-06-12
Description:	Routines for generalized additive modelling under shape constraints on the component functions of the linear p
Depends:	R (>= 2.15.0), mgcv (>= 1.8-2)
Imports:	methods, stats, graphics, Matrix, splines
Suggests:	nlme
License:	GPL (>= 2)
LazyLoad:	yes

The package provides generalized additive modelling under shape constraints on the component functions of the linear predictor. `scam` and `plot.scam` functions are based on the functions of the unconstrained GAM `gam()` and `plot.gam()` in package `mgcv` and similar in use. `summary.scam` allows to extract the results of the model fitting in the same way as in `summary.gam`. A Bayesian approach is used to obtain a covariance matrix of the model coefficients and credible intervals for each smooth.

Author(s)

Natalya Pya <nat.pya@gmail.com> based partly on `mgcv` by Simon Wood

Maintainer: Natalya Pya <nat.pya@gmail.com>

References

Pya, N. and Wood, S.N. (2015) Shape constrained additive models. Statistics and Computing, 25(3), 543-559

Pya, N. (2010) Additive models with shape constraints. PhD thesis. University of Bath. Department of Mathematical Sciences

Wood S.N. (2006) Generalized Additive Models: An Introduction with R. Chapman and Hall/CRC Press.

Wood, S.N. (2008) Fast stable direct fitting and smoothness selection for generalized additive models. Journal of the Royal Statistical Society (B) 70(3):495-518

Wood, S.N. (2011) Fast stable restricted maximum likelihood and marginal likelihood estimation of semiparametric generalized linear models. Journal of the Royal Statistical Society (B) 73(1):3-36

The package was part supported by EPSRC grants EP/I000917/1, EP/K005251/1 and the Science Committee of the Ministry of Science and Education of the Republic of Kazakhstan grant #2532/GF3.

Examples

```
## see examples for scam
```

`anova.scam` *Approximate hypothesis tests related to SCAM fits*

Description

Performs hypothesis tests relating to one or more fitted `scam` objects. The function is a clone of `anova.gam` of the `mgcv` package.

The documentation below is similar to that of object `anova.gam`.

Usage

```
## S3 method for class 'scam'
anova(object, ..., dispersion = NULL, test = NULL,
                    freq = FALSE,p.type=0)
## S3 method for class 'anova.scam'
print(x, digits = max(3, getOption("digits") - 3),...)
```

Arguments

`object,...`	fitted model objects of class `scam` as produced by `scam()`.
`x`	an `anova.scam` object produced by a single model call to `anova.scam()`.
`dispersion`	a value for the dispersion parameter: not normally used.
`test`	what sort of test to perform for a multi-model call. One of `"Chisq"`, `"F"` or `"Cp"`.

freq	whether to use frequentist or Bayesian approximations for parametric term p-values. See summary.gam for details.
p.type	selects exact test statistic to use for single smooth term p-values. See summary.scam for details.
digits	number of digits to use when printing output.

Details

see anova.gam for details.

Value

In the multi-model case anova.scam produces output identical to anova.glm, which it in fact uses.

In the single model case an object of class anova.scam is produced, which is in fact an object returned from summary.scam.

print.anova.scam simply produces tabulated output.

WARNING

If models 'a' and 'b' differ only in terms with no un-penalized components then p values from anova(a,b) are unreliable, and usually much too low.

Default P-values will usually be wrong for parametric terms penalized using 'paraPen': use freq=TRUE to obtain better p-values when the penalties are full rank and represent conventional random effects.

For a single model, interpretation is similar to drop1, not anova.lm.

Author(s)

Simon N. Wood <simon.wood@r-project.org>

References

Scheipl, F., Greven, S. and Kuchenhoff, H. (2008) Size and power of tests for a zero random effect variance or polynomial regression in additive and linear mixed models. Comp. Statist. Data Anal. 52, 3283-3299

Wood, S.N. (2013a) On p-values for smooth components of an extended generalized additive model. Biometrika 100:221-228

Wood, S.N. (2013b) A simple test for random effects in regression models. Biometrika 100:1005-1010

See Also

scam, predict.scam, scam.check, summary.scam, anova.gam

Examples

```
library(scam)
set.seed(0)
fac <- rep(1:4,20)
x1 <- runif(80)*5
x2 <- runif(80,-1,2)
x3 <- runif(80, 0, 1)
y <- fac+log(x1)/5
y <- y + exp(-1.3*x2) +rnorm(80)*0.1
fac <- factor(fac)
b <- scam(y ~ fac+s(x1,bs="mpi")+s(x2,bs="mpd")+s(x3))

b1 <- scam(y ~ fac+s(x1,bs="mpi")+s(x2,bs="mpd"))
anova(b,b1,test="F")

## b2 <- scam(y ~ fac +s(x1)+s(x2)+te(x1,x2))
```

---

bfgs_gcv.ubre — *Multiple Smoothing Parameter Estimation by GCV/UBRE*

Description

Function to efficiently estimate smoothing parameters of SCAM by GCV/UBRE score optimization. The procedure is outer to the model fitting by the Newton-Raphson method. The function uses the BFGS method where the Hessian matrix is updated iteratively at each step. Backtracking is included to satisfy the sufficient decrease condition.

The function is not normally called directly, but rather service routines for `scam`.

Usage

```
bfgs_gcv.ubre(fn=gcv.ubre_grad, rho, ini.fd=TRUE, G, gamma=1, env,
        n.pen=length(rho), typx=rep(1,n.pen), typf=1,
        control)
```

Arguments

`fn`	GCV/UBRE Function which returs the GCV/UBRE value and its derivative wrt log smoothing parameter.
`rho`	log of the initial values of the smoothing parameters.
`ini.fd`	If TRUE, a finite difference to the Hessian is used to find the initial inverse Hessian, otherwise the initial inverse Hessian is a diagonal matrix ‘100*I’.
`G`	A list of items needed to fit a SCAM.
`gamma`	An ad hoc parametrer of the GCV/UBRE score.
`env`	Get the enviroment for the model coefficients, their derivatives and the smoothing parameter.

n.pen	Smoothing parameter dimension.
typx	A vector whose component is a positive scalar specifying the typical magnitude of sp.
typf	A positive scalar estimating the magnitude of the gcv near the minimum.
control	Control option list as returned by scam.control.

Value

A list is returned with the following items:

gcv.ubre	The optimal value of GCV/UBRE.
rho	The best value of the log smoothing parameter.
dgcv.ubre	The gradient of the GCV/UBRE.
iterations	The number of iterations taken until convergence.
conv.bfgs	Convergence information indicating why the BFGS terminated (given below).
termcode	An integer code indicating why the optimization process terminated. 1: relative gradient is close to zero, current iterate probably is a solution. 2: scaled distance between last two steps less than 'steptol', current iterate probably is a local minimizer, but it's possible that the algorithm is making very slow progress, or 'steptol' is too large. 3: last global step failed to locate a point lower than estimate. Either estimate is an approximate local minimum of the function or steptol is too small. 4: iteration limit exceeded. 5: five consecutive steps of length maxNstep have been taken, it's possible that 'maxstep' is too small.
object	A list of elements returned by the fitting procedure scam.fit for an optimal value of the smoothing parameter.
dgcv.ubre.check	If check.analytical=TRUE this is the finite-difference approximation of the gradient calculated by gcv.ubre_grad, otherwise NULL.
check.grad	If check.analytical=TRUE this is the relative difference (in and finite differenced derivatives calculated by gcv.ubre_grad, otherwise NULL.

Author(s)

Natalya Pya <nat.pya@gmail.com>

References

Pya, N. and Wood, S.N. (2015) Shape constrained additive models. Statistics and Computing, 25(3), 543-559

Pya, N. (2010) Additive models with shape constraints. PhD thesis. University of Bath. Department of Mathematical Sciences

Wood, S.N. (2011) Fast stable restricted maximum likelihood and marginal likelihood estimation of semiparametric generalized linear models. Journal of the Royal Statistical Society: Series B. 73(1): 1–34

See Also

scam

---

check.analytical *Checking the analytical gradient of the GCV/UBRE score*

Description

This function calculates the finite-difference approximation of the GCV/UBRE gradient for the fitted model and compares it with the analytical gradient.

Usage

```
check.analytical(object, data, del=1e-6,control)
```

Arguments

object	A fitted scam object.
data	An original data frame or list containing the model response variable and covariates.
del	A positive scalar (default is 1e-6) giving an increment for finite difference approximation.
control	Control option list as returned by scam.control.

Value

A list is returned with the following items:

dgcv.ubre.fd	The finite-difference approximation of the gradient.
check.grad	The relative difference in percentage between the analytical and finite differenced derivatives.

Author(s)

Natalya Pya <nat.pya@gmail.com>

See Also

scam

derivative.scam *Derivative of the univariate smooth model terms*

Description

Function to get derivatives of the smooth model terms (currently only of the univariate smooths). Analytical derivatives for SCOP-splines, finite difference approximation is used for all others

Usage

```
derivative.scam(object,smooth.number=1,deriv=1)
```

Arguments

object	fitted scam object
smooth.number	ordered number of the smooth model term (1,2,...), ordered as in the formula, which derivative is needed to be calculated.
deriv	either 1 if the 1st derivative is required, or 2 if the 2nd

Value

d	values of the derivative of the smooth term.
se.d	standard errors of the derivative.

Author(s)

Natalya Pya <nat.pya@gmail.com>

References

Pya, N. and Wood, S.N. (2015) Shape constrained additive models. Statistics and Computing, 25(3), 543-559

See Also

scam

Examples

```
 set.seed(2)
n <- 200
x1 <- runif(n)*4-1;
f1 <- exp(4*x1)/(1+exp(4*x1)) # monotone increasing smooth
x2 <- sort(runif(n)*3-1)      # decreasing smooth
f2 <- exp(-1.3*x2)
f <- f1+ f2
y <- f+ rnorm(n)*0.2
```

```
## fit model, results, and plot...
b <- scam(y~ s(x1,k=20,bs="mpi")+s(x2,k=15,bs="mpd"),family=gaussian)

d1 <- derivative.scam(b,smooth.number=1,deriv=1)

par(mfrow=c(1,2))

xx <- sort(x1,index=TRUE)
plot(xx$x,d1$d[xx$ix],type="l",xlab=expression(x[1]),
     ylab=expression(df[1]/dx[1]))

d2 <- derivative.scam(b,smooth.number=2,deriv=1)

xx <- sort(x2,index=TRUE)
plot(xx$x,d2$d[xx$ix],type="l",xlab=expression(x[2]),
     ylab=expression(df[2]/dx[2]))
```

`formula.scam` *SCAM formula*

Description

Description of `scam` formula (see `gam` of the `mgcv` package for Details), and how to extract it from a fitted `scam` object.

The function is a clone of `formula.gam` of the `mgcv` package.

Usage

```
## S3 method for class 'scam'
formula(x,...)
```

Arguments

`x`	fitted model objects of class `scam` as produced by `scam()`.
`...`	un-used in this case

Details

see `formula.gam` for details.

Value

Returns the model formula, `x$formula`. Provided so that `anova` methods print an appropriate description of the model.

See Also

`scam`

`gcv.ubre_grad` *The GCV/UBRE score value and its gradient*

Description

For the estimation of the SCAM smoothing parameters the GCV/UBRE score is optimized outer to the Newton-Raphson procedure of the model fitting. This function returns the value of the GCV/UBRE score and calculates its first derivative with respect to the log smoothing parameter using the method of Wood (2009).

The function is not normally called directly, but rather service routines for `bfgs_gcv.ubre`.

Usage

```
gcv.ubre_grad(rho, G, gamma, env, check.analytical=FALSE, del, maxit, maxHalf.fit,
              devtol.fit, steptol.fit)
```

Arguments

`rho`	log of the initial values of the smoothing parameters.
`G`	a list of items needed to fit a SCAM.
`gamma`	A constant multiplier to inflate the model degrees of freedom in the GCV or UBRE/AIC score.
`env`	Get the enviroment for the model coefficients, their derivatives and the smoothing parameter.
`check.analytical`	If this is `TRUE` then finite difference derivatives of GCV/UBRE score will be calculated, otherwise `NULL`.
`del`	A positive scalar (default is 1e-4) giving an increment for finite difference approximation when `check.analytical=TRUE`, otherwise `NULL`.
`maxit`	Maximum number of IRLS iterations to perform used in `scam.fit`.
`maxHalf.fit`	If a step of the Newton-Raphson optimization method leads to a worse penalized deviance, then the step length of the model coefficients is halved. This is the number of halvings to try before giving up used in `scam.fit`.
`devtol.fit`	A positive scalar giving the convergence control for the model fitting algorithm in `scam.fit`.
`steptol.fit`	A positive scalar giving the tolerance at which the scaled distance between two successive iterates is considered close enough to zero to terminate the model fitting algorithm in `scam.fit`.

Value

A list is returned with the following items:

`dgcv.ubre`	The value of GCV/UBRE gradient.

gcv.ubre
: The GCV/UBRE score value.

scale.est
: The value of the scale estimate.

object
: The elements of the fitting procedure monogam.fit for a given value of the smoothing parameter.

dgcv.ubre.check
: If check.analytical=TRUE this returns the finite-difference approximation of the gradient.

check.grad
: If check.analytical=TRUE this returns the relative difference (in and finite differenced derivatives.

Author(s)

Natalya Pya <nat.pya@gmail.com>

References

Pya, N. and Wood, S.N. (2015) Shape constrained additive models. Statistics and Computing, 25(3), 543-559

Pya, N. (2010) Additive models with shape constraints. PhD thesis. University of Bath. Department of Mathematical Sciences

Wood S.N. (2006) Generalized Additive Models: An Introduction with R. Chapman and Hall/CRC Press.

Wood, S.N. (2011) Fast stable restricted maximum likelihood and marginal likelihood estimation of semiparametric generalized linear models. Journal of the Royal Statistical Society: Series B. 73(1): 1–34

See Also

scam, scam.fit, bfgs_gcv.ubre

---

linear.functional.terms *Linear functionals of a smooth in GAMs*

Description

Since scam uses the model setup of gam of the mgcv package, in the same way as in gam scam allows the response variable to depend on linear functionals of smooth terms in the s with additional shape constraints.

See linear.functional.terms(mgcv).

Examples

```
 ## Not run:
###########################################
## similar to a "signal" regression
## example from mgcv() ...
###########################################
library(scam)
## decreasing smooth...
set.seed(4)
rf <- function(x=seq(-1,3,length=100)) {
## generates random functions...
  m <- ceiling(runif(1)*5) ## number of components
  f <- x*0;
  mu <- runif(m,min(x),max(x));sig <- (runif(m)+.5)*(max(x)-min(x))/10
  for (i in 1:m) f <- f+ dnorm(x,mu[i],sig[i])
  f
}

## simulate 200 functions and store in rows of L...
L <- matrix(NA,200,100)
for (i in 1:200) L[i,] <- rf()  ## simulate the functional predictors

x <- seq(-1,3,length=100) ## evaluation points
f2 <- function(x) { ## the coefficient function
     -4*exp(4*x)/(1+exp(4*x))
}
f <- f2(x)
plot(x,f ,type="l")
y <- L%*%f + rnorm(200)*20 ## simulated response data
X <- matrix(x,200,100,byrow=TRUE)

b <- scam(y~s(X,by=L,k=20,bs="mpd"))
par(mfrow=c(1,1))
plot(b,shade=TRUE);lines(x,f,col=2);
## compare with gam() of mgcv package...
g <- gam(y~s(X,by=L,k=20))
x11()
par(mfrow=c(1,1))
plot(g,shade=TRUE);lines(x,f,col=2)


## increasing smooth....
x <- seq(-1,3,length=100) ## evaluation points
f2 <- function(x) { ## the coefficient function
     4*exp(4*x)/(1+exp(4*x))
}
f <- f2(x)
plot(x,f ,type="l")
y <- L%*%f + rnorm(200)*20 ## simulated response data
X <- matrix(x,200,100,byrow=TRUE)

b <- scam(y~s(X,by=L,k=20,bs="mpi"))
```

```
par(mfrow=c(1,1))
plot(b,shade=TRUE);lines(x,f,col=2);

g <- gam(y~s(X,by=L,k=20))
x11()
par(mfrow=c(1,1))
plot(g,shade=TRUE);lines(x,f,col=2)


## End(Not run)
```

---

logLik.scam *Log likelihood for a fitted SCAM, for AIC*

Description

Function to extract the log-likelihood for a fitted `scam` model (fitted by penalized likelihood maximization). Used by `AIC`.

The function is a clone of `logLik.gam` of the `mgcv` package.

The documentation below is similar to that of object `logLik.gam`.

Usage

```
## S3 method for class 'scam'
logLik(object,...)
```

Arguments

`object`	fitted model objects of class `scam` as produced by `scam()`.
`...`	unused in this case

Details

see `logLik.gam` for details.

Value

Standard `logLik` object: see `logLik`.

References

Hastie and Tibshirani, 1990, Generalized Additive Models.

Wood, S.N. (2008) Fast stable direct fitting and smoothness selection for generalized additive models. J.R.Statist. Soc. B 70(3):495-518

See Also

`AIC`

`marginal.matrices.tescv.ps` *Constructs marginal model matrices for "tescv" and "tescx" bivariate smooths in case of B-splines basis functions for both unconstrained marginal smooths*

Description

This function returns the marginal model matrices and the list of penalty matrices for the tensor product bivariate smooth with the single concavity or convexity restriction along the second covariate. The marginal smooth functions of both covariates are constructed using the B-spline basis functions.

Usage

```
marginal.matrices.tescv.ps(x, z, xk, zk, m, q1, q2)
```

Arguments

`x`	A numeric vector of the values of the first covariate at which to evaluate the B-spline marginal functions. The values in `x` must be between `xk[m[1]+2]` and `xk[length(xk) - m[1] - 1]`.
`z`	A numeric vector of the values of the second covariate at which to evaluate the B-spline marginal functions. The values in `z` must be between `zk[m[2]+2]` and `zk[length(zk) - m[2] - 1]`.
`xk`	A numeric vector of knot positions for the first covariate, `x`, with non-decreasing values.
`zk`	A numeric vector of knot positions for the second covariate,z, with non-decreasing values.
`m`	A pair of two numbers where `m[i]+1` denotes the order of the basis of the i^{th} marginal smooth (e.g. `m[i] = 2` for a cubic spline.)
`q1`	A number denoting the basis dimension of the first marginal smooth.
`q2`	A number denoting the basis dimension of the second marginal smooth.

Details

The function is not called directly, but is rather used internally by the constructor `smooth.construct.tescv.smooth.spec` and `smooth.construct.tescx.smooth.spec` .

Value

`X1`	Marginal model matrix for the first unconstrained marginal smooth.
`X2`	Marginal model matrix for the second monotonic marginal smooth.
`S`	A list of penalty matrices for this tensor product smooth.

Author(s)

Natalya Pya <nat.pya@gmail.com>

References

Pya, N. and Wood, S.N. (2015) Shape constrained additive models. Statistics and Computing, 25(3), 543-559

See Also

smooth.construct.tescv.smooth.spec, smooth.construct.tescx.smooth.spec, marginal.matrices.tesmi1.ps, smooth.construct.tesmd1.smooth.spec, smooth.construct.tesmd2.smooth.spec

---

marginal.matrices.tesmi1.ps

Constructs marginal model matrices for "tesmi1" and "tesmd1" bivariate smooths in case of B-splines basis functions for both unconstrained marginal smooths

---

Description

This function returns the marginal model matrices and the list of penalty matrices for the tensor product bivariate smooth with the single monotone increasing or decreasing restriction along the first covariate. The marginal smooth functions of both covariates are constructed using the B-spline basis functions.

Usage

```
marginal.matrices.tesmi1.ps(x, z, xk, zk, m, q1, q2)
```

Arguments

x	A numeric vector of the values of the first covariate at which to evaluate the B-spline marginal functions. The values in x must be between xk[m[1]+2] and xk[length(xk) - m[1] - 1].
z	A numeric vector of the values of the second covariate at which to evaluate the B-spline marginal functions. The values in z must be between zk[m[2]+2] and zk[length(zk) - m[2] - 1].
xk	A numeric vector of knot positions for the first covariate, x, with non-decreasing values.
zk	A numeric vector of knot positions for the second covariate,z, with non-decreasing values.
m	A pair of two numbers where m[i]+1 denotes the order of the basis of the i^{th} marginal smooth (e.g. m[i] = 2 for a cubic spline.)
q1	A number denoting the basis dimension of the first marginal smooth.
q2	A number denoting the basis dimension of the second marginal smooth.

Details

The function is not called directly, but is rather used internally by the constructor `smooth.construct.tesmi1.smooth.spec` and `smooth.construct.tesmd1.smooth.spec` .

Value

`X1`	Marginal model matrix for the first monotonic marginal smooth.
`X2`	Marginal model matrix for the second unconstrained marginal smooth.
`S`	A list of penalty matrices for this tensor product smooth.

Author(s)

Natalya Pya <nat.pya@gmail.com>

References

Pya, N. and Wood, S.N. (2015) Shape constrained additive models. Statistics and Computing, 25(3), 543-559

Pya, N. (2010) Additive models with shape constraints. PhD thesis. University of Bath. Department of Mathematical Sciences

Wood S.N. (2006) Generalized Additive Models: An Introduction with R. Chapman and Hall/CRC Press.

See Also

`smooth.construct.tesmi1.smooth.spec`, `smooth.construct.tesmi2.smooth.spec`, `marginal.matrices.tesmi2.ps`, `smooth.construct.tesmd1.smooth.spec`, `smooth.construct.tesmd2.smooth.spec`

`marginal.matrices.tesmi2.ps`
Constructs marginal model matrices for "tesmi2" and "tesmd2" bivariate smooths in case of B-splines basis functions for both unconstrained marginal smooths

---

Description

This function returns the marginal model matrices and the list of penalty matrices for the tensor product bivariate smooth with the single monotone increasing or decreasing restriction along the second covariate. The marginal smooth functions of both covariates are constructed using the B-spline basis functions.

Usage

```
marginal.matrices.tesmi2.ps(x, z, xk, zk, m, q1, q2)
```

Arguments

- `x` A numeric vector of the values of the first covariate at which to evaluate the B-spline marginal functions. The values in `x` must be between `xk[m[1]+2]` and `xk[length(xk) - m[1] - 1]`.
- `z` A numeric vector of the values of the second covariate at which to evaluate the B-spline marginal functions. The values in `z` must be between `zk[m[2]+2]` and `zk[length(zk) - m[2] - 1]`.
- `xk` A numeric vector of knot positions for the first covariate, `x`, with non-decreasing values.
- `zk` A numeric vector of knot positions for the second covariate,z, with non-decreasing values.
- `m` A pair of two numbers where `m[i]+1` denotes the order of the basis of the i^{th} marginal smooth (e.g. `m[i] = 2` for a cubic spline.)
- `q1` A number denoting the basis dimension of the first marginal smooth.
- `q2` A number denoting the basis dimension of the second marginal smooth.

Details

The function is not called directly, but is rather used internally by the constructor `smooth.construct.tesmi2.smooth.spec` and `smooth.construct.tesmd2.smooth.spec` .

Value

- `X1` Marginal model matrix for the first unconstrained marginal smooth.
- `X2` Marginal model matrix for the second monotonic marginal smooth.
- `S` A list of penalty matrices for this tensor product smooth.

Author(s)

Natalya Pya <nat.pya@gmail.com>

References

Pya, N. and Wood, S.N. (2015) Shape constrained additive models. Statistics and Computing, 25(3), 543-559

Pya, N. (2010) Additive models with shape constraints. PhD thesis. University of Bath. Department of Mathematical Sciences

Wood S.N. (2006) Generalized Additive Models: An Introduction with R. Chapman and Hall/CRC Press.

See Also

`smooth.construct.tesmi1.smooth.spec`, `smooth.construct.tesmi2.smooth.spec`, `marginal.matrices.tesmi1.ps`, `smooth.construct.tesmd1.smooth.spec`, `smooth.construct.tesmd2.smooth.spec` plot.scam *SCAM plotting*

Description

The function is a clone of the `plot.gam` of the `mgcv` package with the differences in the construction of the Bayesian confidence intervals of the shape constrained smooth terms. The function takes a fitted `scam` object produced by `scam()` and plots the component smooth functions that make it up, on the scale of the linear predictor. Optionally produces term plots for parametric model components as well.

Usage

```
## S3 method for class 'scam'
plot(x,residuals=FALSE,rug=TRUE,se=TRUE,pages=0,select=NULL,scale=-1,
        n=100,n2=40,pers=FALSE,theta=30,phi=30,jit=FALSE,xlab=NULL,
        ylab=NULL,main=NULL,ylim=NULL,xlim=NULL,too.far=0.1,
        all.terms=FALSE,shade=FALSE,shade.col="gray80",
        shift=0,trans=I,seWithMean=FALSE,unconditional = FALSE,
        by.resids = FALSE,scheme=0,...)
```

Arguments

The list of the arguments is the same as in `plot.gam` of the `mgcv` package.

a fitted `gam` object as produced by `gam()`.

`residuals` If `TRUE` then partial residuals are added to plots of 1-D smooths. If `FALSE` then no residuals are added. If this is an array of the correct length then it is used as the array of residuals to be used for producing partial residuals. If `TRUE` then the residuals are the working residuals from the IRLS iteration weighted by the IRLS weights. Partial residuals for a smooth term are the residuals that would be obtained by dropping the term concerned from the model, while leaving all other estimates fixed (i.e. the estimates for the term plus the residuals).

`rug` when TRUE (default) then the covariate to which the plot applies is displayed as a rug plot at the foot of each plot of a 1-d smooth, and the locations of the covariates are plotted as points on the contour plot representing a 2-d smooth.

`se` when TRUE (default) upper and lower lines are added to the 1-d plots at 2 standard errors above and below the estimate of the smooth being plotted while for 2-d plots, surfaces at +1 and -1 standard errors are contoured and overlayed on the contour plot for the estimate. If a positive number is supplied then this number is multiplied by the standard errors when calculating standard error curves or surfaces. See also `shade`, below.

`pages` (default 0) the number of pages over which to spread the output. For example, if `pages=1` then all terms will be plotted on one page with the layout performed automatically. Set to 0 to have the routine leave all graphics settings as they are.

`select`	Allows the plot for a single model term to be selected for printing. e.g. if you just want the plot for the second smooth term set `select=2`.
`scale`	set to -1 (default) to have the same y-axis scale for each plot, and to 0 for a different y axis for each plot. Ignored if `ylim` supplied.
`n`	number of points used for each 1-d plot - for a nice smooth plot this needs to be several times the estimated degrees of freedom for the smooth. Default value 100.
`n2`	Square root of number of points used to grid estimates of 2-d functions for contouring.
`pers`	Set to `TRUE` if you want perspective plots for 2-d terms.
`theta`	One of the perspective plot angles.
`phi`	The other perspective plot angle.
`jit`	Set to TRUE if you want rug plots for 1-d terms to be jittered.
`xlab`	If supplied then this will be used as the x label for all plots.
`ylab`	If supplied then this will be used as the y label for all plots.
`main`	Used as title (or z axis label) for plots if supplied.
`ylim`	If supplied then this pair of numbers are used as the y limits for each plot.
`xlim`	If supplied then this pair of numbers are used as the x limits for each plot.
`too.far`	If greater than 0 then this is used to determine when a location is too far from data to be plotted when plotting 2-D smooths. This is useful since smooths tend to go wild away from data. The data are scaled into the unit square before deciding what to exclude, and `too.far` is a distance within the unit square.
`all.terms`	if set to `TRUE` then the partial effects of parametric model components are also plotted, via a call to `termplot`. Only terms of order 1 can be plotted in this way.
`shade`	Set to `TRUE` to produce shaded regions as confidence bands for smooths (not avaliable for parametric terms, which are plotted using `termplot`).
`shade.col`	define the color used for shading confidence bands.
`shift`	constant to add to each smooth (on the scale of the linear predictor) before plotting. Can be useful for some diagnostics, or with `trans`.
`trans`	function to apply to each smooth (after any shift), before plotting. `shift` and `trans` are occasionally useful as a means for getting plots on the response scale, when the model consists only of a single smooth.
`seWithMean`	if `TRUE` the component smooths are shown with confidence intervals that include the uncertainty about the overall mean. If `FALSE` then the uncertainty relates purely to the centred smooth itself. An extension of the argument presented in Nychka (1988) suggests that `TRUE` results in better coverage performance, and this is also suggested by simulation.
`unconditional`	if `TRUE` then the smoothing parameter uncertainty corrected covariance matrix is used to compute uncertainty bands, if available. Otherwise the bands treat the smoothing parameters as fixed.
`by.resids`	Should partial residuals be plotted for terms with `by` variables? Usually the answer is no, they would be meaningless.

scheme — Integer (0,1 or 2) or integer vector selecting a plotting scheme for each plot. `scheme == 0` produces a smooth curve with dashed curves indicating 2 standard error bounds. `scheme == 1` illustrates the error bounds using a shaded region. For `scheme==0`, contour plots are produced for 2-d smooths with the x-axes labelled with the first covariate name and the y axis with the second covariate name. For 2-d smooths `scheme==1` produces a perspective plot, while `scheme==2` produces a heatmap, with overlaid contours.

... — other graphics parameters to pass on to plotting commands.

Value

The function generates plots.

Author(s)

Natalya Pya <nat.pya@gmail.com> based on the `plot.gam` of the `mgcv` by Simon Wood

References

Pya, N. and Wood, S.N. (2015) Shape constrained additive models. Statistics and Computing, 25(3), 543-559

Pya, N. (2010) Additive models with shape constraints. PhD thesis. University of Bath. Department of Mathematical Sciences

Wood S.N. (2006) Generalized Additive Models: An Introduction with R. Chapman and Hall/CRC Press.

See Also

scam

Examples

```
## simulating data...
n <- 200
set.seed(1)
x0 <- rep(1:4,50)
x1 <- runif(n)*6-3
f1 <- 3*exp(-x1^2) # unconstrained smooth term
x2 <- runif(n)*4-1;
f2 <- exp(4*x2)/(1+exp(4*x2)) # monotone increasing smooth
x3 <- runif(n)*5;
f3 <- -log(x3)/5  # monotone decreasing smooth
f <- f1+f2+f3
y <- 2*x0 + f + rnorm(n)*0.3
x0 <- factor(x0)

## fit the model and plot ...
b <- scam(y~x0+s(x1,k=15,bs="cr")+s(x2,k=30,bs="mpi")+s(x3,k=30,bs="mpd"))
plot(b,pages=1,residuals=TRUE,all.terms=TRUE,shade=TRUE,shade.col=3)
```

```
## Not run:
 ## example with 2-d plots...
 ## simulating data...
   set.seed(2)
   n <- 30
   x0 <- rep(1:9,100)
   x1 <- sort(runif(n)*4-1)
   x2 <- sort(runif(n))
   x3 <- runif(n*n, 0, 1)
   f <- matrix(0,n,n)
   for (i in 1:n) for (j in 1:n)
       { f[i,j] <- -exp(4*x1[i])/(1+exp(4*x1[i]))+2*sin(pi*x2[j])}
   f1 <- as.vector(t(f))
   f2 <- x3*0
   e <- rnorm(length(f1))*0.1
   y <- 2*x0 + f1 + f2 + e
   x0 <- factor(x0)
   x11 <-  matrix(0,n,n)
   x11[,1:n] <- x1
   x11 <- as.vector(t(x11))
   x22 <- rep(x2,n)
   dat <- list(x0=x0,x1=x11,x2=x22,x3=x3,y=y)
## fit model  and plot ...
   b <- scam(y~x0+s(x1,x2,k=c(10,10),bs=c("tesmd1","ps"),m=2)+s(x3),data=dat)
   op <- par(mfrow=c(2,2))
   plot(b,all.terms=TRUE)
   plot(y,b$fitted.values,xlab="Simulated data",ylab="Fitted data",pch=19,cex=.3)
   par(op)

## and use of schemes...
   op <- par(mfrow=c(2,2))
   plot(b,all.terms=TRUE,scheme=1)
   par(op)
   op <- par(mfrow=c(2,2))
   plot(b,all.terms=TRUE,scheme=c(2,1))
   par(op)


## End(Not run)
```

Predict.matrix.mpi.smooth

Predict matrix method functions for SCAMs

Description

The various built in smooth classes for use with scam have associate Predict.matrix method functions to enable prediction from the fitted model.

Usage

```
## S3 method for class 'mpi.smooth'
Predict.matrix(object, data)
## S3 method for class 'mpd.smooth'
Predict.matrix(object, data)
## S3 method for class 'cv.smooth'
Predict.matrix(object, data)
## S3 method for class 'cx.smooth'
Predict.matrix(object, data)
## S3 method for class 'micx.smooth'
Predict.matrix(object, data)
## S3 method for class 'micv.smooth'
Predict.matrix(object, data)
## S3 method for class 'mdcx.smooth'
Predict.matrix(object, data)
## S3 method for class 'mdcv.smooth'
Predict.matrix(object, data)
## S3 method for class 'po.smooth'
Predict.matrix(object, data)
## S3 method for class 'tedmd.smooth'
Predict.matrix(object, data)
## S3 method for class 'tedmi.smooth'
Predict.matrix(object, data)
## S3 method for class 'tesmd1.smooth'
Predict.matrix(object, data)
## S3 method for class 'tesmd2.smooth'
Predict.matrix(object, data)
## S3 method for class 'tesmi1.smooth'
Predict.matrix(object, data)
## S3 method for class 'tesmi2.smooth'
Predict.matrix(object, data)
## S3 method for class 'temicx.smooth'
Predict.matrix(object, data)
## S3 method for class 'temicv.smooth'
Predict.matrix(object, data)
## S3 method for class 'tedecx.smooth'
Predict.matrix(object, data)
## S3 method for class 'tedecv.smooth'
Predict.matrix(object, data)
## S3 method for class 'tescx.smooth'
Predict.matrix(object, data)
## S3 method for class 'tescv.smooth'
Predict.matrix(object, data)
## S3 method for class 'tecvcv.smooth'
Predict.matrix(object, data)
## S3 method for class 'tecxcv.smooth'
Predict.matrix(object, data)
## S3 method for class 'tecxcx.smooth'
```

```
Predict.matrix(object, data)
```

Arguments

- `object` A smooth object, usually generated by a `smooth.construct` method having processed a smooth specification object generated by an `s` term in a `scam` formula.
- `data` A data frame containing the values of the named covariates at which the smooth term is to be evaluated.

Value

A matrix mapping the coefficients for the smooth term to its values at the supplied data values.

Author(s)

Natalya Pya <nat.pya@gmail.com>

References

Pya, N. and Wood, S.N. (2015) Shape constrained additive models. Statistics and Computing, 25(3), 543-559

Pya, N. (2010) Additive models with shape constraints. PhD thesis. University of Bath. Department of Mathematical Sciences

Wood S.N. (2006) Generalized Additive Models: An Introduction with R. Chapman and Hall/CRC Press.

`predict.scam` *Prediction from fitted SCAM model*

Description

This function is a clone of the `mgcv` library code `predict.gam` with some modifications to adopt shape preserving smooth terms. It takes a fitted `scam` object produced by `scam()` and produces predictions given a new set of values for the model covariates or the original values used for the model fit. Predictions can be accompanied by standard errors, based on the posterior distribution of the model coefficients.

It now alows prediction outside the range of knots, and use linear extrapolation in this case.

Usage

```
## S3 method for class 'scam'
predict(object,newdata,type="link",se.fit=FALSE,terms=NULL,
    block.size=1000,newdata.guaranteed=FALSE,na.action=na.pass,...)
```

Arguments

`object`	a fitted `scam` object as produced by `scam()`.
`newdata`	A data frame or list containing the values of the model covariates at which predictions are required. If this is not provided then predictions corresponding to the original data are returned. If `newdata` is provided then it should contain all the variables needed for prediction: a warning is generated if not.
`type`	When this has the value `"link"` (default) the linear predictor (possibly with associated standard errors) is returned. When `type="terms"` each component of the linear predictor is returned seperately (possibly with standard errors): this includes parametric model components, followed by each smooth component, but excludes any offset and any intercept. `type="iterms"` is the same, except that any standard errors returned for unconstrained smooth components will include the uncertainty about the intercept/overall mean. When `type="response"` predictions on the scale of the response are returned (possibly with approximate standard errors). When `type="lpmatrix"` then a matrix is returned which yields the values of the linear predictor (minus any offset) when postmultiplied by the parameter vector (in this case `se.fit` is ignored). The latter option is most useful for getting variance estimates for quantities derived from the model: for example integrated quantities, or derivatives of smooths. A linear predictor matrix can also be used to implement approximate prediction outside `R` (see example code, below).
`se.fit`	when this is TRUE (not default) standard error estimates are returned for each prediction.
`terms`	if `type=="terms"` then only results for the terms given in this array will be returned.
`block.size`	maximum number of predictions to process per call to underlying code: larger is quicker, but more memory intensive. Set to < 1 to use total number of predictions as this.
`newdata.guaranteed`	Set to `TRUE` to turn off all checking of `newdata` except for sanity of factor levels: this can speed things up for large prediction tasks, but `newdata` must be complete, with no `NA` values for predictors required in the model.
`na.action`	what to do about `NA` values in `newdata`. With the default `na.pass`, any row of `newdata` containing `NA` values for required predictors, gives rise to `NA` predictions (even if the term concerned has no `NA` predictors). `na.exclude` or `na.omit` result in the dropping of `newdata` rows, if they contain any `NA` values for required predictors. If `newdata` is missing then `NA` handling is determined from `object$na.action`.
`...`	other arguments.

Details

See `predict.gam` for details.

Value

If type=="lpmatrix" then a matrix is returned which will give a vector of linear predictor values (minus any offest) at the supplied covariate values, when applied to the model coefficient vector. Otherwise, if se.fit is TRUE then a 2 item list is returned with items (both arrays) fit and se.fit containing predictions and associated standard error estimates, otherwise an array of predictions is returned. The dimensions of the returned arrays depends on whether type is "terms" or not: if it is then the array is 2 dimensional with each term in the linear predictor separate, otherwise the array is 1 dimensional and contains the linear predictor/predicted values (or corresponding s.e.s). The linear predictor returned termwise will not include the offset or the intercept.

newdata can be a data frame, list or model.frame: if it's a model frame then all variables must be supplied.

Author(s)

Natalya Pya <nat.pya@gmail.com> based partly on mgcv by Simon Wood

References

Chambers and Hastie (1993) Statistical Models in S. Chapman & Hall.

Wood S.N. (2006) Generalized Additive Models: An Introduction with R. Chapman and Hall/CRC Press.

Pya, N. and Wood, S.N. (2015) Shape constrained additive models. Statistics and Computing, 25(3), 543-559

Pya, N. (2010) Additive models with shape constraints. PhD thesis. University of Bath. Department of Mathematical Sciences

See Also

scam, plot.scam

Examples

```
## Not run:
library(scam)
set.seed(2)
n <- 200
x1 <- runif(n)*6-3
f1 <- 3*exp(-x1^2) # unconstrained term
x2 <- runif(n)*4-1;
f2 <- exp(4*x2)/(1+exp(4*x2)) # monotone increasing smooth
f <- f1+f2
y <- f+rnorm(n)*0.2
dat <- data.frame(x1=x1,x2=x2,y=y)
b <- scam(y~s(x1,k=15,bs="cr",m=2)+s(x2,k=30,bs="mpi",m=2),
    family=gaussian(link="identity"),data=dat)

newd <- data.frame(x1=seq(-3,3,length.out=20),x2=seq(-1,3,length.out=20))
pred <- predict(b,newd)
pred
```

```
predict(b,newd,type="terms",se=TRUE)


## linear extrapolation with predict.scam()...
set.seed(3)
n <- 100
x <- sort(runif(n)*3-1)
f <- exp(-1.3*x)
y <- rpois(n,exp(f))
dat <- data.frame(x=x,y=y)
b <- scam(y~s(x,k=15,bs="mpd",m=2),family=poisson(link="log"),data=dat)
newd <- data.frame(x=c(2.3,2.7,3.2))
fe <- predict(b,newd,type="link",se=TRUE)
ylim<- c(min(y,exp(fe$fit)),max(y,exp(fe$fit)))
plot(c(x,newd[[1]]),c(y,NA,NA,NA),ylim=ylim)
lines(c(x,newd[[1]]),c(b$fitted,exp(fe$fit)),col=3)

## Gaussian model ....
## simulating data...
 set.seed(2)
 n <- 200
 x <- sort(runif(n)*4-1)
 f <- exp(4*x)/(1+exp(4*x)) # monotone increasing smooth
 y <- f+rnorm(n)*0.1
 dat <- data.frame(x=x,y=y)
 b <- scam(y~ s(x,k=25,bs="mpi",m=2),family=gaussian,data=dat)
 newd <- data.frame(x=c(3.2,3.3,3.6))
 fe <- predict(b,newd)
 plot(c(x,newd[[1]]),c(y,NA,NA,NA))
 lines(c(x,newd[[1]]),c(b$fitted,fe),col=3)

### passing observed data + new data...
   newd <- data.frame(x=c(x,3.2,3.3,3.6))
   fe <- predict(b,newd,se=TRUE)
   plot(newd[[1]],c(y,NA,NA,NA))
   lines(newd[[1]],fe$fit,col=2)
   lines(newd[[1]],fe$fit+2*fe$se.fit,col=3)
   lines(newd[[1]],fe$fit-2*fe$se.fit,col=4)

## prediction with CI...
   newd <- data.frame(x=seq(-1.2,3.5,length.out=100))
   fe <- predict(b,newd,se=TRUE)
   ylim<- c(min(y,fe$se.fit),max(y,fe$se.fit))
   plot(newd[[1]],fe$fit,type="l",ylim=ylim)
   lines(newd[[1]],fe$fit+2*fe$se.fit,lty=2)
   lines(newd[[1]],fe$fit-2*fe$se.fit,lty=2)


## bivariate example...
   n <- 30
   x1 <- sort(runif(n));  x2 <- sort(runif(n)*4-1)
   f1 <- matrix(0,n,n)
   for (i in 1:n) for (j in 1:n)
```

```
      { f1[i,j] <- 2*sin(pi*x1[i]) +exp(4*x2[j])/(1+exp(4*x2[j]))}
   f1 <- as.vector(t(f1));   f <- (f1-min(f1))/(max(f1)-min(f1))
   y <- f+rnorm(length(f))*0.1
   x11 <-  matrix(0,n,n); x11[,1:n] <- x1;  x11 <- as.vector(t(x11))
   x22 <- rep(x2,n)
   dat <- list(x1=x11,x2=x22,y=y)
   b <- scam(y~s(x1,x2,k=c(10,10),bs="tesmi2"),family=gaussian, data=dat)
   par(mfrow=c(2,2),mar=c(4,4,2,2))
   plot(b,se=TRUE);   plot(b,pers=TRUE,theta = 80, phi = 40)

   n.out <- 20
   xp <- seq(0,1.4,length.out=n.out)
   zp <- seq(-1,3.4,length.out=n.out)
   xp1 <-  matrix(0,n.out,n.out);   xp1[,1:n.out] <- xp
   xp1 <- as.vector(t(xp1));   xp2 <- rep(zp,n.out)
   newd <- data.frame(x1=xp1,x2=xp2)
   fe <- predict(b,newd)
   fc <- t(matrix(fe,n.out,n.out))
   persp(xp,zp,fc,expand= 0.85,ticktype = "simple",xlab="x1",
     ylab="x2",zlab="f^",main="d", theta = 80, phi = 40)



## obtaining a 'prediction matrix'...

newd <- data.frame(x1=c(-2,-1),x2=c(0,1))
Xp <- predict(b,newdata=newd,type="lpmatrix")
fv <- Xp
fv

## End(Not run)
```

---

print.scam *Print a SCAM object*

Description

The default print method for a scam object. The code is a clone of print.gam of the mgcv package with a slight simplification since only two methods of smoothing parameter selection (by GCV or UBRE) was implemented for scam.

Usage

```
## S3 method for class 'scam'
print(x,...)
```

Arguments

x	fitted model objects of class scam as produced by scam().
...	other arguments.

Details

As for mgcv(gam) prints out the family, model formula, effective degrees of freedom for each smooth term, and optimized value of the smoothness selection criterion used.

Author(s)

Natalya Pya <nat.pya@gmail.com>

References

Wood S.N. (2006) Generalized Additive Models: An Introduction with R. Chapman and Hall/CRC Press.

See Also

scam, summary.scam

---

residuals.scam *SCAM residuals*

Description

This function is a clone of the mgcv library code residuals.gam. It returns residuals for a fitted scam model object. Pearson, deviance, working and response residuals are available.

Usage

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

Arguments

object	a scam fitted model object.
type	the type of residuals wanted.
...	other arguments.

Details

See residuals.gam for details.

Value

An array of residuals.

Author(s)

Natalya Pya <nat.pya@gmail.com>

See Also

scam

---

scam *Shape constrained additive models (SCAM) and integrated smoothness selection*

Description

This function fits a SCAM to data. Univariate smooths subject to monotonicity, convexity, or monotonicity plus convexity are available as model terms, as well as bivariate smooths with double or single monotonicity. Smoothness selection is estimated as part of the fitting. Confidence/credible intervals are available for each smooth term.

All the shaped constrained smooths have been added to the gam() in package mgcv setup using the smooth.construct function. The routine calls a gam() function for the model set up, but there are separate functions for the model fitting, scam.fit, and smoothing parameter selection, bfgs_gcv.ubre. Any unconstrained smooth available in gam can be taken as model terms.

Usage

```
scam(formula, family = gaussian(), data = list(), gamma = 1,
      sp = NULL, weights = NULL, offset = NULL,
      optimizer="bfgs", optim.method=c("Nelder-Mead","fd"),
      scale = 0, knots=NULL, not.exp=FALSE, start= NULL, etastart,
      mustart,control=list())
```

Arguments

formula	A SCAM formula. This is exactly like the formula for a GAM (see formula.gam of the mgcv library) except that monotone smooth terms, can be added in the expression of the form s(x1,k=12,bs="mpi",by=z), where bs indicates the basis to use for the constrained smooth: the built in options for the monotonic smooths are described in shape.constrained.smooth.terms,
family	A family object specifying the distribution and link to use in fitting etc. See glm and family for more details.
data	A data frame or list containing the model response variable and covariates required by the formula. By default the variables are taken from environment(formula): typically the environment from which gam is called.
gamma	A constant multiplier to inflate the model degrees of freedom in the GCV or UBRE/AIC score.
sp	A vector of smoothing parameters can be provided here. Smoothing parameters must be supplied in the order that the smooth terms appear in the model formula. The default sp=NULL indicates that smoothing parameters should be estimated. If length(sp) does not correspond to the number of underlying smoothing parameters or negative values supplied then the vector is ignored and all the smoothing parameters will be estimated.

weights
: Prior weights on the data.

offset
: Used to supply a model offset for use in fitting. Note that this offset will always be completely ignored when predicting, unlike an offset included in `formula`. This conforms to the behaviour of `lm`, `glm` and `gam`.

optimizer
: The numerical optimization method to use to optimize the smoothing parameter estimation criterion. "bfgs" for the built in to `scam` package routine `bfgs_gcv.ubre`, "optim", "nlm", "nlm.fd" (based on finite-difference approximation of the derivatives).

optim.method
: In case of `optimizer="optim"` this specifies the numerical method to be used in `optim` in the first element, the second element of `optim.method` indicates whether the finite difference approximation should be used ("fd") or analytical gradient ("grad"). The default is `optim.method=c("Nelder-Mead","fd")`.

scale
: If this is positive then it is taken as the known scale parameter of the exponential family distribution. Negative value indicates that the scale paraemter is unknown. 0 indicates that the scale parameter is 1 for Poisson and binomial and unknown otherwise. This conforms to the behaviour of `gam`.

knots
: An optional list containing user specified knot values to be used for basis construction. Different terms can use different numbers of knots.

not.exp
: if `TRUE` then `notExp()` function will be used in place of `exp` (default value) in positivity ensuring beta parameters re-parameterization.

start
: Initial values for the model coefficients.

etastart
: Initial values for the linear predictor.

mustart
: Initial values for the expected values.

control
: A list of fit control parameters to replace defaults returned by `scam.control`. Values not set assume default values.

Details

A shape constrained additive model (SCAM) is a generalized linear model (GLM) in which the linear predictor is given by strictly parametric components plus a sum of smooth functions of the covariates where some of the functions are assumed to be shape constrained. For example,

$$\log(E(Y_i)) = X_i^* b + f_1(x_{1i}) + m_2(x_{2i}) + f_3(x_{3i})$$

where the independent response variables Y_i follow Poisson distribution with `log` link function, f_1, m_2, and f_3 are smooth functions of the corresponding covariates, and m_2 is subject to monotone increasing constraint.

All available shape constrained smooths are decsribed in `shape.constrained.smooth.terms`.

Value

The function returns an object of class `"scam"` with the following elements (this agrees with `gamObject`):

aic
: AIC of the fitted model: the degrees of freedom used to calculate this are the effective degrees of freedom of the model, and the likelihood is evaluated at the maximum of the penalized likelihood, not at the MLE.

assign — Array whose elements indicate which model term (listed in `pterms`) each parameter relates to: applies only to non-smooth terms.

bfgs.info — If `optimizer="bfgs"`, a list of convergence diagnostics relating to the BFGS method of smoothing parameter selection. The items are: `conv`, indicates why the BFGS algorithm of the smoothness selection terminated; `iter`, number of iterations of BFGS taken to get convergence; `grad`, the gradient of the GCV/UBRE score at convergence.

call — the matched call.

coefficients — the coefficients of the fitted model. Parametric coefficients are first, followed by coefficients for each spline term in turn.

coefficients.t — the parametrized coefficients of the fitted model (exponentiated for the monotonic smooths).

conv — indicates whether or not the iterative fitting method converged.

CPU.time — indicates the real and CPU time (in seconds) taken by the fitting process in case of unknown smoothing parameters

data — the original supplied data argument. Only included if the `scam` argument `keepData` is set to `TRUE` (default is `FALSE`).

deviance — model deviance (not penalized deviance).

df.null — null degrees of freedom.

df.residual — effective residual degrees of freedom of the model.

edf — estimated degrees of freedom for each model parameter. Penalization means that many of these are less than 1.

edf1 — alternative estimate of edf.

family — family object specifying distribution and link used.

fitted.values — fitted model predictions of expected value for each datum.

formula — the model formula.

gcv.ubre — the minimized GCV or UBRE score.

dgcv.ubre — the gradient of the GCV or UBRE score.

iter — number of iterations of the Newton-Raphson method taken to get convergence.

linear.predictors — fitted model prediction of link function of expected value for each datum.

method — `"GCV"` or `"UBRE"`, depending on the fitting criterion used.

min.edf — Minimum possible degrees of freedom for whole model.

model — model frame containing all variables needed in original model fit.

nlm.info — If `optimizer="nlm"` or `optimizer="nlm.fd"`, a list of convergence diagnostics relating to the BFGS method of smoothing parameter selection. The items are: `conv`, indicates why the BFGS algorithm of the smoothness selection terminated; `iter`, number of iterations of BFGS taken to get convergence; `grad`, the gradient of the GCV/UBRE score at convergence.

not.exp — if `TRUE` then `notExp()` function will be used in place of `exp` (default value) in positivity ensuring beta parameters re-parameterization.

`nsdf` number of parametric, non-smooth, model terms including the intercept.

`null.deviance` deviance for single parameter model.

`offset` model offset.

`optim.info` If `optimizer="optim"`, a list of convergence diagnostics relating to the BFGS method of smoothing parameter selection. The items are: `conv`, indicates why the BFGS algorithm of the smoothness selection terminated; `iter`, number of iterations of BFGS taken to get convergence; `optim.method`, the numerical optimization method used.

`prior.weights` prior weights on observations.

`pterms` `terms` object for strictly parametric part of model.

`R` Factor R from QR decomposition of weighted model matrix, unpivoted to be in same column order as model matrix.

`residuals` the working residuals for the fitted model.

`scale.estimated`
`TRUE` if the scale parameter was estimated, `FALSE` otherwise.

`sig2` estimated or supplied variance/scale parameter.

`smooth` list of smooth objects, containing the basis information for each term in the model formula in the order in which they appear. These smooth objects are returned by the `smooth.construct` objects.

`sp` estimated smoothing parameters for the model. These are the underlying smoothing parameters, subject to optimization.

`termcode` an integer indicating why the optimization process of the smoothness selection terminated (see `bfgs_gcv.ubre`).

`terms` `terms` object of `model` model frame.

`trA` trace of the influence matrix, total number of the estimated degrees of freedom (`sum(edf)`).

`var.summary` A named list of summary information on the predictor variables. See `gamObject`.

`Ve` frequentist estimated covariance matrix for the parameter estimators.

`Vp` estimated covariance matrix for the parameters. This is a Bayesian posterior covariance matrix that results from adopting a particular Bayesian model of the smoothing process.

`Ve.t` frequentist estimated covariance matrix for the reparametrized parameter estimators obtained using the delta method. Particularly useful for testing whether terms are zero. Not so useful for CI's as smooths are usually biased.

`Vp.t` estimated covariance matrix for the reparametrized parameters obtained using the delta method. Paricularly useful for creating credible/confidence intervals.

`weights` final weights used in the Newton-Raphson iteration.

`cmX` column means of the model matrix (with elements corresponding to smooths set to zero).

`y` response data.

Author(s)

Natalya Pya <nat.pya@gmail.com> based partly on mgcv by Simon Wood

References

Pya, N. and Wood, S.N. (2015) Shape constrained additive models. Statistics and Computing, 25(3), 543-559

Pya, N. (2010) Additive models with shape constraints. PhD thesis. University of Bath. Department of Mathematical Sciences

Wood, S.N. (2011) Fast stable restricted maximum likelihood and marginal likelihood estimation of semiparametric generalized linear models. Journal of the Royal Statistical Society (B) 73(1):3-36

Wood S.N. (2006a) *Generalized Additive Models: An Introduction with R*. Chapman and Hall/CRC Press.

Wood, S.N. (2006b) On confidence intervals for generalized additive models based on penalized regression splines. Australian and New Zealand Journal of Statistics. 48(4): 445-464.

See Also

scam-package, shape.constrained.smooth.terms, gam, s, plot.scam, summary.scam, scam.check, predict.scam

Examples

```
##************************************
## Gaussian model, 2 terms, 1 monotonic increasing....
   ## simulating data...
require(scam)

set.seed(0)
n <- 200
x1 <- runif(n)*6-3
f1 <- 3*exp(-x1^2) # unconstrained term
f1 <- (f1-min(f1))/(max(f1)-min(f1)) # function scaled to have range [0,1]
x2 <- runif(n)*4-1;
f2 <- exp(4*x2)/(1+exp(4*x2)) # monotone increasing smooth
f2 <- (f2-min(f2))/(max(f2)-min(f2)) # function scaled to have range [0,1]
f <- f1+f2
y <- f+rnorm(n)*0.1
dat <- data.frame(x1=x1,x2=x2,y=y)
  ## fit model, results, and plot...
b <- scam(y~s(x1,k=15,bs="cr",m=2)+s(x2,k=25,bs="mpi",m=2),
    family=gaussian(link="identity"),data=dat,not.exp=FALSE)
print(b)
summary(b)
plot(b,pages=1)


##*************************************
## Gaussian model, 2 terms, increasing + decreasing convex ....
   ## simulating data...
```

```
set.seed(2)
n <- 200
x1 <- runif(n)*4-1;
f1 <- exp(4*x1)/(1+exp(4*x1)) # monotone increasing smooth
x2 <- runif(n)*3-1;
f2 <- exp(-3*x2)/15  # monotone decreasing and convex smooth
f <- f1+f2
y <- f+ rnorm(n)*0.2
dat <- data.frame(x1=x1,x2=x2,y=y)
  ## fit model, results, and plot...
b <- scam(y~ s(x1,k=25,bs="mpi",m=2)+s(x2,k=25,bs="mdcx",m=2),
    family=gaussian(link="identity"),data=dat)
print(b)
summary(b)
plot(b,pages=1,scale=0)

##***********************************
## Not run:
## using optim() for smoothing parameter selection...
b1 <- scam(y~ s(x1,k=25,bs="mpi",m=2)+s(x2,k=25,bs="mdcx",m=2),
    family=gaussian(link="identity"),data=dat, optimizer="optim")
summary(b1)

b2 <- scam(y~ s(x1,k=25,bs="mpi",m=2)+s(x2,k=25,bs="mdcx",m=2),
    family=gaussian(link="identity"),data=dat, optimizer="optim",
    optim.method=c("BFGS","fd"))
summary(b2)

## using nlm()...
b3 <- scam(y~ s(x1,k=25,bs="mpi",m=2)+s(x2,k=25,bs="mdcx",m=2),
    family=gaussian(link="identity"),data=dat, optimizer="nlm")
summary(b3)

## End(Not run)


##************************************
## Poisson model ....
   ## simulating data...
set.seed(2)
n <- 200
x1 <- runif(n)*6-3
f1 <- 3*exp(-x1^2) # unconstrained term
x2 <- runif(n)*4-1;
f2 <- exp(4*x2)/(1+exp(4*x2)) # monotone increasing smooth
f <- f1+f2
y <- rpois(n,exp(f))
dat <- data.frame(x1=x1,x2=x2,y=y)
  ## fit model, results, and plot...
b <- scam(y~s(x1,k=15,bs="cr",m=2)+s(x2,k=30,bs="mpi",m=2),
      family=poisson(link="log"),data=dat,optimizer="nlm.fd")
print(b)
```

```
summary(b)
plot(b,pages=1)
scam.check(b)

## Gamma model...
   ## simulating data...
set.seed(3)
n <- 200
x1 <- runif(n)*6-3
f1 <- 1.5*sin(1.5*x1) # unconstrained term
x2 <- runif(n)*4-1;
f2 <- 1.5/(1+exp(-10*(x2+0.75)))+1.5/(1+exp(-5*(x2-0.75))) # monotone increasing smooth
x3 <- runif(n)*6-3;
f3 <- 3*exp(-x3^2)  # unconstrained term
f <- f1+f2+f3
y <- rgamma(n,shape=1,scale=exp(f))
dat <- data.frame(x1=x1,x2=x2,x3=x3,y=y)
   ## fit model, results, and plot...
b <- scam(y~s(x1,k=15,bs="ps",m=2)+s(x2,k=30,bs="mpi",m=2)+
            s(x3,k=15,bs="ps",m=2),family=Gamma(link="log"),
            data=dat,optimizer="nlm")
print(b)
summary(b)
par(mfrow=c(2,2))
plot(b)

## Not run:
## bivariate example...
 ## simulating data...
   set.seed(2)
   n <- 30
   x1 <- sort(runif(n)*4-1)
   x2 <- sort(runif(n))
   f1 <- matrix(0,n,n)
   for (i in 1:n) for (j in 1:n)
       { f1[i,j] <- -exp(4*x1[i])/(1+exp(4*x1[i]))+2*sin(pi*x2[j])}
   f <- as.vector(t(f1))
   y <- f+rnorm(length(f))*0.1
   x11 <-  matrix(0,n,n)
   x11[,1:n] <- x1
   x11 <- as.vector(t(x11))
   x22 <- rep(x2,n)
   dat <- list(x1=x11,x2=x22,y=y)
## fit model  and plot ...
   b <- scam(y~s(x1,x2,k=c(10,10),bs=c("tesmd1","ps"),m=2),
            family=gaussian(link="identity"), data=dat,sp=NULL)
   summary(b)
   par(mfrow=c(2,2),mar=c(4,4,2,2))
   plot(b,se=TRUE)
   plot(b,pers=TRUE,theta = 30, phi = 40)
   plot(y,b$fitted.values,xlab="Simulated data",ylab="Fitted data",pch=".",cex=3)

## example with random effect smoother...
```

```
    set.seed(2)
    n <- 200
    x1 <- runif(n)*6-3
    f1 <- 3*exp(-x1^2) # unconstrained term
    f1 <- (f1-min(f1))/(max(f1)-min(f1))
    x2 <- runif(n)*4-1;
    f2 <- exp(4*x2)/(1+exp(4*x2)) # monotone increasing smooth
    f2 <- (f2-min(f2))/(max(f2)-min(f2))
    f <- f1+f2
    a <- factor(sample(1:10,200,replace=TRUE))
    Xa <- model.matrix(~a-1)    ## random main effects
    y <- f + Xa%*%rnorm(length(levels(a)))*.5 + rnorm(n)*0.1
    dat <- data.frame(x1=x1,x2=x2,y=y,a=a)
    ## fit model and plot...
    b <- scam(y~s(x1,k=15,bs="cr",m=2)+s(x2,k=25,bs="mpi",m=2)+s(a,bs="re"), data=dat)
    summary(b)
    scam.check(b)
    plot(b,pages=1)


## End(Not run)
```

---

scam.check *Some diagnostics for a fitted scam object*

Description

Takes a fitted scam object produced by scam() and produces some diagnostic information about the fitting procedure and results. This function is almost the same as gam.check of the mgcv library.

Usage

```
scam.check(b,rl.col=3,... )
```

Arguments

b	a fitted scam object as produced by scam().
rl.col	color for the reference line on the quantile-quantile plot.
...	extra graphics parameters to pass to plotting functions.

Details

As for mgcv(gam) plots 4 standard diagnostic plots, and some other convergence diagnostics. The printed information relates to the optimization process used to select smoothing parameters.

Author(s)

Natalya Pya <nat.pya@gmail.com> based partly on mgcv by Simon Wood

References

Wood S.N. (2006) Generalized Additive Models: An Introduction with R. Chapman and Hall/CRC Press.

See Also

scam

Examples

```
 ## Not run:
library(scam)
set.seed(2)
n <- 200
x1 <- runif(n)*4-1;
f1 <- exp(4*x1)/(1+exp(4*x1)) # monotone increasing smooth
x2 <- runif(n)*3-1;
f2 <- exp(-3*x2)/15  # monotone decreasing and convex smooth
f <- f1+f2
y <- f+ rnorm(n)*0.2
dat <- data.frame(x1=x1,x2=x2,y=y)
b <- scam(y~ s(x1,k=25,bs="mpi",m=2)+s(x2,k=25,bs="mdcx",m=2),
    family=gaussian(link="identity"),data=dat)
plot(b,pages=1)
scam.check(b,pch=19,cex=.3)

## End(Not run)
```

---

scam.control *Setting GAM fitting defaults*

Description

This is an internal function of package scam which allows control of the numerical options for fitting a SCAM.

Usage

```
scam.control(maxit = 200, maxHalf.fit=40, devtol.fit=1e-8,steptol.fit=1e-8,
              keepData=FALSE,nlm=list(),optim=list(),bfgs=list())
```

Arguments

- `maxit` Maximum number of IRLS iterations to perform used in scam.fit.
- `maxHalf.fit` If a step of the Newton-Raphson optimization method leads to a worse penalized deviance, then the step length of the model coefficients is halved. This is the number of halvings to try before giving up used in scam.fit.

`devtol.fit`	A positive scalar giving the convergence control for the model fitting algorithm in `scam.fit`.
`steptol.fit`	A positive scalar giving the tolerance at which the scaled distance between two successive iterates is considered close enough to zero to terminate the model fitting algorithm in `scam.fit`.
`nlm`	list of control parameters to pass to `nlm` if this is used for outer estimation of smoothing parameters (not default).
`optim`	list of control parameters to pass to `optim` if this is used for outer estimation of smoothing parameters (not default).
`bfgs`	list of control parameters to pass to default BFGS optimizer used for outer estimation of log smoothing parameters.
`keepData`	Should a copy of the original `data` argument be kept in the `scam` object?

Details

Outer iteration is used to estimate smoothing parameters of SCAM by GCV/UBRE score optimization. The default procedure is the built-in BFGS method which is controlled by the list `bfgs` with the following elements: `steptol.bfgs` (default 1e-7) is the relative convergence tolerance; `gradtol.bfgs` (default 6.0554*1e-6) is a tolerance at which the gradient is considered to be close enougth to 0 to terminate the BFGS algorithm; `maxNstep` is a positive scalar which gives the maximum allowable step length (default 5); `maxHalf` gives the maximum number of step halving in "backtracking" to permit before giving up(default 30); `check.analytical` is logical whether the analytical gradient of GCV/UBRE should be checked numerically (default `FALSE`); `del` is an increment for finite differences when checking analytical gradients (default 1e-4).

If outer iteration using `nlm` is used for fitting, then the control list `nlm` stores control arguments for calls to routine `nlm`. As in `gam.control` the list has the following named elements: `ndigit` is the number of significant digits in the GCV/UBRE score; `gradtol` is the tolerance used to judge convergence of the gradient of the GCV/UBRE score to zero (default 1e-6); `stepmax` is the maximum allowable log smoothing parameter step (default 2); `steptol` is the minimum allowable step length (default 1e-4); `iterlim` is the maximum number of optimization steps allowed (default 200); `check.analyticals` indicates whether the built in exact derivative calculations should be checked numerically (default `FALSE`). Any of these which are not supplied and named in the list are set to their default values.

Outer iteration using `optim` is controlled using list `optim`, which currently has one element: `factr` which takes default value 1e7.

Author(s)

Natalya Pya Arnqvist <`nat.pya@gmail.com`> based partly on `gam.control` by Simon Wood

References

Pya, N. and Wood, S.N. (2015) Shape constrained additive models. Statistics and Computing, 25(3), 543-559

Pya, N. (2010) Additive models with shape constraints. PhD thesis. University of Bath. Department of Mathematical Sciences

Wood, S.N. (2011) Fast stable restricted maximum likelihood and marginal likelihood estimation of semiparametric generalized linear models. Journal of the Royal Statistical Society (B) 73(1):3-36

See Also

scam, scam.fit, gam.control

---

scam.fit *Newton-Raphson method to fit SCAM*

Description

This routine estimates SCAM coefficients given log smoothing parameters using the Newton-Raphson method. The estimation of the smoothing parameters by the GCV/UBRE score optimization is outer to the model fitting. Routine gcv.ubre_grad evaluates the first derivatives of the smoothness selection scores with respect to the log smoothing parameters. Routine bfgs_gcv.ubre estimates the smoothing parameters using the BFGS method.

The function is not normally called directly, but rather service routines for scam.

Usage

```
scam.fit(G, sp, maxit=200, maxHalf.fit=40, devtol.fit=1e-8, steptol.fit=1e-8,
                 gamma=1, start=NULL, etastart=NULL, mustart=NULL, env=env)
```

Arguments

G	A list of items needed to fit a SCAM.
sp	The vector of smoothing parameters.
maxit	Maximum iterations in the Newton-Raphson procedure.
maxHalf.fit	If a step of the Newton-Raphson optimization method leads to a worse penalized deviance, then the step length of the model coefficients is halved. This is the number of halvings to try before giving up.
devtol.fit	A positive scalar giving the tolerance at which the scaled distance between two successive penalized deviances is considered close enough to zero to terminate the algorithm.
steptol.fit	A positive scalar giving the tolerance at which the scaled distance between two successive iterates is considered close enough to zero to terminate the algorithm.
gamma	This constant allows to inflate the model degrees of freedom in the GCV or UBRE/AIC score.
start	Initial values for the model coefficients.
etastart	Initial values for the linear predictor.
mustart	Initial values for the expected values.
env	Get the enviroment for the model coefficients, their derivatives and the smoothing parameter.

Details

The routine applies step halving to any step that increases the penalized deviance substantially.

Author(s)

Natalya Pya <nat.pya@gmail.com>

References

Pya, N. and Wood, S.N. (2015) Shape constrained additive models. Statistics and Computing, 25(3), 543-559

Pya, N. (2010) Additive models with shape constraints. PhD thesis. University of Bath. Department of Mathematical Sciences

Wood, S.N. (2008) Fast stable direct fitting and smoothness selection for generalized additive models. Journal of the Royal Statistical Society (B) 70(3):495-518

Wood, S.N. (2011) Fast stable restricted maximum likelihood and marginal likelihood estimation of semiparametric generalized linear models. Journal of the Royal Statistical Society (B) 73(1):3-36

See Also

`scam`

---

shape.constrained.smooth.terms

Shape preserving smooth terms in SCAM

Description

As in `mgcv(gam)`, shape preserving smooth terms are specified in a `scam` formula using `s` terms. All the shape constrained smooth terms are constructed using the B-splines basis proposed by Eilers and Marx (1996) with a discrete penalty on the basis coefficients.

The univariate single penalty built in shape constrained smooth classes are summarized as follows

- Monotone increasing P-splines `bs="mpi"`. To achieve monotone increasing smooths these reparametrize the coefficients so that they form an increasing sequence. For details see `smooth.construct.mpi.smooth.spec`.
- Monotone decreasing P-splines `bs="mpd"`. To achieve monotone decreasing smooths these reparametrize the coefficients so that they form a decreasing sequence. A first order difference penalty applied to the basis coefficients starting with the second is used for the monotone increasing and decreasing cases.
- Convex P-splines `bs="cx"`. These reparametrize the coefficients so that the second order differences of the basis coefficients are greater than zero. For details see `smooth.construct.cx.smooth.spec`.
- Concave P-splines `bs="cv"`. These reparametrize the coefficients so that the second order differences of the basis coefficients are less than zero. For details see `smooth.construct.cv.smooth.spec`.

- Monotone increasing and convex P-splines bs="micx". These reparametrize the coefficients so that the first and the second order differences of the basis coefficients are greater than zero. For details see smooth.construct.micx.smooth.spec.
- Monotone increasing and concave P-splines bs="micv". These reparametrize the coefficients so that the first order differences of the basis coefficients are greater than zero while the second order difference are less than zero.
- Monotone decreasing and convex P-splines bs="mdcx". These reparametrize the coefficients so that the first order differences of the basis coefficients are less than zero while the second order difference are greater. For details see smooth.construct.mdcx.smooth.spec.
- Monotone decreasing and concave P-splines bs="mdcv". These reparametrize the coefficients so that the first and the second order differences of the basis coefficients are less than zero.

  For all four types of the mixed constrained smoothing a first order difference penalty applied to the basis coefficients starting with the third one is used.

Using the concept of the tensor product spline bases bivariate smooths under monotonicity constraint where monotonicity may be assumed on only one of the covariates (single monotonicity) or both of them (double monotonicity) are added as the smooth terms of the SCAM. Bivariate B-spline is constructed by expressing the coefficients of one of the marginal univariate B-spline bases as the B-spline of the other covariate. Double or single monotonicity is achieved by the corresponding re-parametrization of the bivariate basis coefficients to satisfy the sufficient conditions formulated in terms of the first order differences of the coefficients. The following explains the built in bivariate monotonic smooth classes.

- Double monotone increasing P-splines bs="tedmi". See smooth.construct.tedmi.smooth.spec for details.
- Double monotone decreasing P-splines bs="tedmd".
- Single monotone increasing P-splines along the first covariate direction bs="tesmi1".
- Single monotone increasing P-splines along the second covariate direction bs="tesmi2".
- Single monotone decreasing P-splines along the first covariate direction bs="tesmd1".
- Single monotone decreasing P-splines along the second covariate direction bs="tesmd2".

  Double penalties for the monotonic tensor product smooths are obtained from the penalties of the marginal smooths.

Author(s)

Natalya Pya <nat.pya@gmail.com>

References

Pya, N. and Wood, S.N. (2015) Shape constrained additive models. Statistics and Computing, 25(3), 543-559

Pya, N. (2010) Additive models with shape constraints. PhD thesis. University of Bath. Department of Mathematical Sciences

Eilers, P.H.C. and B.D. Marx (1996) Flexible Smoothing with B-splines and Penalties. Statistical Science, 11(2):89-121

Wood S.N. (2006a) Generalized Additive Models: An Introduction with R. Chapman and Hall/CRC Press.

Wood, S.N. (2006b) Low rank scale invariant tensor product smooths for generalized additive mixed models. Biometrics 62(4):1025-1036

See Also

`s`, `smooth.construct.mpi.smooth.spec`, `smooth.construct.mpd.smooth.spec`, `smooth.construct.cx.smooth.spec`, `smooth.construct.cv.smooth.spec`, `smooth.construct.micx.smooth.spec`, `smooth.construct.micv.smooth.spec`, `smooth.construct.mdcx.smooth.spec`, `smooth.construct.mdcv.smooth.spec`, `smooth.construct.tedmi.smooth.sp` `smooth.construct.tedmd.smooth.spec`, `smooth.construct.tesmi1.smooth.spec`, `smooth.construct.tesmi2.smoo` `smooth.construct.tesmd1.smooth.spec`, `smooth.construct.tesmd2.smooth.spec`

Examples

```
## see examples for scam
```

---

smooth.construct.cv.smooth.spec
Constructor for concave P-splines in SCAMs

Description

This is a special method function for creating smooths subject to concavity constraint which is built by the `mgcv` constructor function for smooth terms, `smooth.construct`. It is constructed using concave P-splines. This smooth is specified via model terms such as `s(x,k,bs="cv",m=2)`, where `k` denotes the basis dimension and `m+1` is the order of the B-spline basis.

Usage

```
## S3 method for class 'cv.smooth.spec'
smooth.construct(object, data, knots)
```

Arguments

`object`	A smooth specification object, generated by an `s` term in a GAM formula.
`data`	A data frame or list containing the data required by this term, with names given by `object$term`. The `by` variable is the last element.
`knots`	An optional list containing the knots supplied for basis setup. If it is `NULL` then the knot locations are generated automatically.

Value

An object of class `"cv.smooth"`.

Author(s)

Natalya Pya <nat.pya@gmail.com>

References

Pya, N. and Wood, S.N. (2015) Shape constrained additive models. Statistics and Computing, 25(3), 543-559

Pya, N. (2010) Additive models with shape constraints. PhD thesis. University of Bath. Department of Mathematical Sciences

See Also

smooth.construct.cx.smooth.spec, smooth.construct.mpi.smooth.spec, smooth.construct.mdcv.smooth.spec, smooth.construct.mdcx.smooth.spec, smooth.construct.micx.smooth.spec, smooth.construct.mpd.smooth.spec

Examples

```
 ## Not run:
## Concave P-splines example
  ## simulating data...
   set.seed(1)
   n <- 100
   x <- sort(2*runif(n)-1)
   f <- -4*x^2
   y <- f + rnorm(n)*0.45
   dat <- data.frame(x=x,y=y)
   b <- scam(y~s(x,k=15,bs="cv",m=2),family=gaussian,data=dat,not.exp=FALSE)
   # UNCONSTRAINED FIT *****************
   b1 <- scam(y~s(x,k=15,bs="cr",m=2),family=gaussian, data=dat,not.exp=FALSE)

## plot results ...
   plot(x,y,xlab="x",ylab="y")
   lines(x,f)      ## the true function
   lines(x,b$fitted,col=2)  ## constrained fit
   lines(x,b1$fitted,col=3) ## unconstrained fit

## Poisson version...
   y <- rpois(n,15*exp(f))
   dat <- data.frame(x=x,y=y)
 ## fit model ...
   b <- scam(y~s(x,k=15,bs="cv",m=2),family=poisson(link="log"),data=dat,not.exp=FALSE)

# UNCONSTRAINED FIT *****************
  b1 <- scam(y~s(x,k=15,bs="cr",m=2),family=poisson(link="log"), data=dat,not.exp=FALSE)

## plot results ...
   plot(x,y,xlab="x",ylab="y")
   lines(x,15*exp(f))      ## the true function
   lines(x,b$fitted,col=2)  ## constrained fit
   lines(x,b1$fitted,col=3) ## unconstrained fit

## plotting on log scale...
   plot(x,log(15*exp(f)),type="l")      ## the true function
   lines(x,log(b$fitted),col=2)  ## constrained fit
   lines(x,log(b1$fitted),col=3) ## unconstrained fit
```

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

---

smooth.construct.cx.smooth.spec
Constructor for convex P-splines in SCAMs

Description

This is a special method function for creating smooths subject to convexity constraint which is built by the mgcv constructor function for smooth terms, smooth.construct. It is constructed using the mixed constrained P-splines. This smooth is specified via model terms such as s(x,k,bs="cx",m=2), where k denotes the basis dimension and m+1 is the order of the B-spline basis.

Usage

```
## S3 method for class 'cx.smooth.spec'
smooth.construct(object, data, knots)
```

Arguments

object	A smooth specification object, generated by an s term in a GAM formula.
data	A data frame or list containing the data required by this term, with names given by object$term. The by variable is the last element.
knots	An optional list containing the knots supplied for basis setup. If it is NULL then the knot locations are generated automatically.

Value

An object of class "cx.smooth".

Author(s)

Natalya Pya <nat.pya@gmail.com>

References

Pya, N. and Wood, S.N. (2015) Shape constrained additive models. Statistics and Computing, 25(3), 543-559

Pya, N. (2010) Additive models with shape constraints. PhD thesis. University of Bath. Department of Mathematical Sciences

See Also

smooth.construct.cv.smooth.spec, smooth.construct.mpi.smooth.spec, smooth.construct.mdcv.smooth.spec, smooth.construct.mdcx.smooth.spec, smooth.construct.micv.smooth.spec, smooth.construct.mpd.smooth.spec

Examples

```
 ## Not run:

## Convex P-splines example
  ## simulating data...
   set.seed(1)
   n <- 100
   x <- sort(2*runif(n)-1)
   f <- 4*x^2
   y <- f + rnorm(n)*0.4
   dat <- data.frame(x=x,y=y)
   b <- scam(y~s(x,k=15,bs="cx",m=2),family=gaussian,data=dat,not.exp=FALSE)
   # UNCONSTRAINED FIT *****************
   b1 <- scam(y~s(x,k=15,bs="cr",m=2),family=gaussian, data=dat,not.exp=FALSE)

## plot results ...
   plot(x,y,xlab="x",ylab="y")
   lines(x,f)      ## the true function
   lines(x,b$fitted,col=2)  ## constrained fit
   lines(x,b1$fitted,col=3) ## unconstrained fit

## Poisson version...
   y <- rpois(n,exp(f))
   dat <- data.frame(x=x,y=y)
 ## fit model ...
   b <- scam(y~s(x,k=15,bs="cx",m=2),family=poisson(link="log"),data=dat,not.exp=FALSE)

# UNCONSTRAINED FIT *****************
  b1 <- scam(y~s(x,k=15,bs="cr",m=2),family=poisson(link="log"), data=dat,not.exp=FALSE)

## plot results ...
   plot(x,y,xlab="x",ylab="y")
   lines(x,exp(f))      ## the true function
   lines(x,b$fitted,col=2)  ## constrained fit
   lines(x,b1$fitted,col=3) ## unconstrained fit


## End(Not run)
```

---

smooth.construct.mdcv.smooth.spec

Constructor for monotone decreasing and concave P-splines in SCAMs

---

Description

This is a special method function for creating smooths subject to both monotone decreasing and concavity constraints which is built by the mgcv constructor function for smooth terms, `smooth.construct`. It is constructed using mixed constrained P-splines. This smooth is specified via model terms such as s(x,k,bs="mdcv",m=2), where k denotes the basis dimension and m+1 is the order of the B-spline basis.

Usage

```
## S3 method for class 'mdcv.smooth.spec'
smooth.construct(object, data, knots)
```

Arguments

object	A smooth specification object, generated by an s term in a GAM formula.
data	A data frame or list containing the data required by this term, with names given by object$term. The by variable is the last element.
knots	An optional list containing the knots supplied for basis setup. If it is NULL then the knot locations are generated automatically.

Value

An object of class "mdcv.smooth".

Author(s)

Natalya Pya <nat.pya@gmail.com>

References

Pya, N. and Wood, S.N. (2015) Shape constrained additive models. Statistics and Computing, 25(3), 543-559

See Also

smooth.construct.mpi.smooth.spec, smooth.construct.mpd.smooth.spec, smooth.construct.cx.smooth.spec, smooth.construct.cv.smooth.spec, smooth.construct.mdcx.smooth.spec, smooth.construct.micx.smooth.spec, smooth.construct.micv.smooth.spec

Examples

```
  ## Not run:
## Monotone decreasing and concave P-splines example
  ## simulating data...
   set.seed(2)
   n <- 100
   x <- sort(runif(n))
   f <- -x^4
   y <- f+rnorm(n)*0.10
   dat <- data.frame(x=x,y=y)
 ## fit model ...
   b <- scam(y~s(x,k=15,bs="mdcv",m=2),family=gaussian(link="identity"),data=dat)

# UNCONSTRAINED FIT *****************
   b1 <- scam(y~s(x,k=15,bs="ps",m=2),family=gaussian(link="identity"),data=dat)
```

```
## plot results ...
   plot(x,y,xlab="x",ylab="y")
   lines(x,f)          ## the true function
   lines(x,b$fitted.values,col=2) ## mixed constrained fit
   lines(x,b1$fitted.values,col=3) ## unconstrained fit

## End(Not run)
```

---

smooth.construct.mdcx.smooth.spec

Constructor for monotone decreasing and convex P-splines in SCAMs

---

Description

This is a special method function for creating smooths subject to both monotone decreasing and convexity constraints which is built by the `mgcv` constructor function for smooth terms, `smooth.construct`. It is constructed using mixed constrained P-splines. This smooth is specified via model terms such as `s(x,k,bs="mdcx",m=2)`, where `k` denotes the basis dimension and `m+1` is the order of the B-spline basis.

Usage

```
## S3 method for class 'mdcx.smooth.spec'
smooth.construct(object, data, knots)
```

Arguments

`object`	A smooth specification object, generated by an `s` term in a GAM formula.
`data`	A data frame or list containing the data required by this term, with names given by `object$term`. The `by` variable is the last element.
`knots`	An optional list containing the knots supplied for basis setup. If it is `NULL` then the knot locations are generated automatically.

Value

An object of class `"mdcx.smooth"`.

Author(s)

Natalya Pya <nat.pya@gmail.com>

References

Pya, N. and Wood, S.N. (2015) Shape constrained additive models. Statistics and Computing, 25(3), 543-559

Pya, N. (2010) Additive models with shape constraints. PhD thesis. University of Bath. Department of Mathematical Sciences

See Also

smooth.construct.mpi.smooth.spec, smooth.construct.mpd.smooth.spec, smooth.construct.cx.smooth.spec, smooth.construct.cv.smooth.spec, smooth.construct.mdcv.smooth.spec, smooth.construct.micx.smooth.spec, smooth.construct.micv.smooth.spec

Examples

```
 ## Not run:
## Monotone decreasing and convex P-splines example
  ## simulating data...
   set.seed(2)
   n <- 100
   x <- sort(runif(n)*3-1)
   f1 <- (x-3)^6  # monotone decreasing and convex smooth
   f <- (f1-min(f1))/(max(f1)-min(f1))
   y <- f+rnorm(n)*0.20
   dat <- data.frame(x=x,y=y)
 ## fit model ...
   b <- scam(y~s(x,k=15,bs="mdcx",m=2),family=gaussian(link="identity"),data=dat)

# UNCONSTRAINED FIT *****************
   b1 <- scam(y~s(x,k=15,bs="ps",m=2),family=gaussian(link="identity"),data=dat)

## plot results ...
   plot(x,y,xlab="x",ylab="y")
   lines(x,f)          ## the true function
   lines(x,b$fitted.values,col=2)  ## mixed constrained fit
   lines(x,b1$fitted.values,col=3) ## unconstrained fit

## End(Not run)
```

---

smooth.construct.micv.smooth.spec

Constructor for monotone increasing and concave P-splines in SCAMs

---

Description

This is a special method function for creating smooths subject to both monotone increasing and concavity constraints which is built by the mgcv constructor function for smooth terms, smooth.construct. It is constructed using mixed constrained P-splines. This smooth is specified via model terms such as s(x,k,bs="micv",m=2), where k denotes the basis dimension and m+1 is the order of the B-spline basis.

Usage

```
## S3 method for class 'micv.smooth.spec'
smooth.construct(object, data, knots)
```

Arguments

`object`	A smooth specification object, generated by an `s` term in a GAM formula.
`data`	A data frame or list containing the data required by this term, with names given by `object$term`. The `by` variable is the last element.
`knots`	An optional list containing the knots supplied for basis setup. If it is `NULL` then the knot locations are generated automatically.

Value

An object of class `"micv.smooth"`.

Author(s)

Natalya Pya <nat.pya@gmail.com>

References

Pya, N. and Wood, S.N. (2015) Shape constrained additive models. Statistics and Computing, 25(3), 543-559

Pya, N. (2010) Additive models with shape constraints. PhD thesis. University of Bath. Department of Mathematical Sciences

See Also

`smooth.construct.mpi.smooth.spec`, `smooth.construct.cx.smooth.spec`, `smooth.construct.cv.smooth.spec`, `smooth.construct.mdcv.smooth.spec`, `smooth.construct.mdcx.smooth.spec`, `smooth.construct.micx.smooth.spe`
`smooth.construct.mpd.smooth.spec`

Examples

```
 ## Not run:
## Monotone increasing and concave P-splines example
  ## simulating data...
   set.seed(3)
   n <- 100
   x <- sort(runif(n)*99+1)
   f1 <- log(x)
   f <- (f1-min(f1))/(max(f1)-min(f1))
   y <- f+rnorm(n)*0.10
   dat <- data.frame(x=x,y=y)
 ## fit model ...
   b <- scam(y~s(x,k=15,bs="micv",m=2), data=dat)

# UNCONSTRAINED FIT *****************
   b1 <- scam(y~s(x,k=15,bs="ps",m=2),data=dat)

## plot results ...
   plot(x,y,xlab="x",ylab="y")
   lines(x,f)           ## the true function
   lines(x,b$fitted.values,col=2)  ## mixed constrained fit
```

```
    lines(x,b1$fitted.values,col=3) ## unconstrained fit

## End(Not run)
```

---

smooth.construct.micx.smooth.spec
Constructor for monotone increasing and convex P-splines in SCAMs

---

Description

This is a special method function for creating smooths subject to both monotone increasing and convexity constraints which is built by the mgcv constructor function for smooth terms, `smooth.construct`. It is constructed using the mixed constrained P-splines. This smooth is specified via model terms such as `s(x,k,bs="micx",m=2)`, where `k` denotes the basis dimension and `m+1` is the order of the B-spline basis.

Usage

```
## S3 method for class 'micx.smooth.spec'
smooth.construct(object, data, knots)
```

Arguments

`object`	A smooth specification object, generated by an `s` term in a GAM formula.
`data`	A data frame or list containing the data required by this term, with names given by `object$term`. The `by` variable is the last element.
`knots`	An optional list containing the knots supplied for basis setup. If it is `NULL` then the knot locations are generated automatically.

Value

An object of class `"micx.smooth"`.

Author(s)

Natalya Pya <nat.pya@gmail.com>

References

Pya, N. and Wood, S.N. (2015) Shape constrained additive models. Statistics and Computing, 25(3), 543-559

Pya, N. (2010) Additive models with shape constraints. PhD thesis. University of Bath. Department of Mathematical Sciences

See Also

smooth.construct.mpi.smooth.spec, smooth.construct.cx.smooth.spec, smooth.construct.cv.smooth.spec, smooth.construct.mdcv.smooth.spec, smooth.construct.mdcx.smooth.spec, smooth.construct.micv.smooth.spec, smooth.construct.mpd.smooth.spec

Examples

```
 ## Not run:
## Monotone increasing and convex P-splines example
  ## simulating data...
   set.seed(22)
   n <- 100
   x <- runif(n)*2
   f <- x^2
   y <- rpois(n,exp(f))
   dat <- data.frame(x=x,y=y)
 ## fit model ...
   b <- scam(y~s(x,k=15,bs="micx",m=2),family=poisson(link="log"),
       data=dat,sp=NULL,not.exp=FALSE)

# UNCONSTRAINED FIT *****************
   b1 <- scam(y~s(x,k=15,bs="cr",m=2),family=poisson(link="log"),
         data=dat,sp=NULL)

## plot results ...
   plot(x,y,xlab="x",ylab="y")
   x1 <- sort(x,index=TRUE)
   lines(x1$x,exp(f)[x1$ix])      ## the true function
   lines(x1$x,b$fitted.values[x1$ix],col=2)  ## mixed constrained fit
   lines(x1$x,b1$fitted.values[x1$ix],col=3) ## unconstrained fit

## End(Not run)
```

---

smooth.construct.mpd.smooth.spec

Constructor for monotone decreasing P-splines in SCAMs

Description

This is a special method function for creating smooths subject to monotone decreasing constraints which is built by the mgcv constructor function for smooth terms, smooth.construct. It is constructed using monotonic P-splines. This smooth is specified via model terms such as s(x,k,bs="mpd",m=2), where k denotes the basis dimension and m+1 is the order of the B-spline basis.

Usage

```
## S3 method for class 'mpd.smooth.spec'
smooth.construct(object, data, knots)
```

Arguments

`object`	A smooth specification object, generated by an `s` term in a GAM formula.
`data`	A data frame or list containing the data required by this term, with names given by `object$term`. The `by` variable is the last element.
`knots`	An optional list containing the knots supplied for basis setup. If it is `NULL` then the knot locations are generated automatically.

Value

An object of class `"mpd.smooth"`.

Author(s)

Natalya Pya <nat.pya@gmail.com>

References

Pya, N. and Wood, S.N. (2015) Shape constrained additive models. Statistics and Computing, 25(3), 543-559

Pya, N. (2010) Additive models with shape constraints. PhD thesis. University of Bath. Department of Mathematical Sciences

See Also

`smooth.construct.mpi.smooth.spec`, `smooth.construct.cx.smooth.spec`, `smooth.construct.cv.smooth.spec`, `smooth.construct.mdcv.smooth.spec`, `smooth.construct.mdcx.smooth.spec`, `smooth.construct.micv.smooth.spe`
`smooth.construct.micx.smooth.spec`

Examples

```
  ## Not run:
## Monotone decreasing P-splines example
  ## simulating data...
   set.seed(3)
   n <- 100
   x <- runif(n)*3-1
   f <- exp(-1.3*x)
   y <- rpois(n,exp(f))
   dat <- data.frame(x=x,y=y)
 ## fit model ...
   b <- scam(y~s(x,k=15,bs="mpd",m=2),family=poisson(link="log"),
       data=dat,sp=NULL)

# UNCONSTRAINED FIT *****************
   b1 <- scam(y~s(x,k=15,bs="ps",m=2),family=poisson(link="log"),
         data=dat,sp=NULL)

## plot results ...
   plot(x,y,xlab="x",ylab="y")
   x1 <- sort(x,index=TRUE)
```

```
    lines(x1$x,exp(f)[x1$ix])      ## the true function
    lines(x1$x,b$fitted.values[x1$ix],col=2)  ## monotone fit
    lines(x1$x,b1$fitted.values[x1$ix],col=3) ## unconstrained fit

## End(Not run)
```

---

smooth.construct.mpi.smooth.spec
Constructor for monotone increasing P-splines in SCAMs

Description

This is a special method function for creating smooths subject to a monotone increasing constraint which is built by the mgcv constructor function for smooth terms, `smooth.construct`. It is constructed using monotonic P-splines. This smooth is specified via model terms suach as `s(x,k,bs="mpi",m=2)`, where `k` denotes the basis dimension and `m+1` is the order of the B-spline basis.

Usage

```
## S3 method for class 'mpi.smooth.spec'
smooth.construct(object, data, knots)
```

Arguments

`object`	A smooth specification object, generated by an `s` term in a GAM formula.
`data`	A data frame or list containing the data required by this term, with names given by `object$term`. The `by` variable is the last element.
`knots`	An optional list containing the knots supplied for basis setup. If it is `NULL` then the knot locations are generated automatically.

Details

The constructor is not called directly, but as with `gam(mgcv)` is used internally.

If the knots of the spline are not supplied, then they are placed evenly throughout the covariate values. If the knots are supplied, then the number of supplied knots should be `k+m+2`, and the range of the middle `k-m` knots must include all the covariate values.

Value

An object of class `"mpi.smooth"`.

Author(s)

Natalya Pya <nat.pya@gmail.com>

References

Pya, N. and Wood, S.N. (2015) Shape constrained additive models. Statistics and Computing, 25(3), 543-559

Pya, N. (2010) Additive models with shape constraints. PhD thesis. University of Bath. Department of Mathematical Sciences

See Also

smooth.construct.mpd.smooth.spec, smooth.construct.cv.smooth.spec, smooth.construct.cx.smooth.spec, smooth.construct.mdcv.smooth.spec, smooth.construct.mdcx.smooth.spec, smooth.construct.micv.smooth.spec, smooth.construct.micx.smooth.spec

Examples

```
## Monotone increasing P-splines example
  ## simulating data...
   set.seed(12)
   n <- 100
   x <- runif(n)*4-1
   f <- 4*exp(4*x)/(1+exp(4*x))
   y <- rpois(n,exp(f))
   dat <- data.frame(x=x,y=y)
 ## fit model ...
   b <- scam(y~s(x,k=15,bs="mpi",m=2),family=poisson(link="log"),
       data=dat,sp=NULL)

# UNCONSTRAINED FIT *****************
   b1 <- scam(y~s(x,k=15,bs="ps",m=2),family=poisson(link="log"),
         data=dat,sp=NULL)

## plot results ...
   plot(x,y,xlab="x",ylab="y")
   x1 <- sort(x,index=TRUE)
   lines(x1$x,exp(f)[x1$ix])      ## the true function
   lines(x1$x,b$fitted.values[x1$ix],col=2)  ## monotone fit
   lines(x1$x,b1$fitted.values[x1$ix],col=3) ## unconstrained fit

## example with supplied knots...
   knots <- list(x=c (-1.5,  -1.2, -.99, -.97, -.7, -.5, -.3, 0, 0.7,
           0.9,1.1, 1.22,1.5,2.2,2.77,2.93,2.99, 3.2,3.6))
   b2 <- scam(y~s(x,k=15,bs="mpi",m=2),knots=knots,
          family=poisson(link="log"), data=dat)
   summary(b2)
   plot(b2)

## Not run:
## example with two terms...
   set.seed(0)
   n <- 200
   x1 <- runif(n)*6-3
```

```
   f1 <- 3*exp(-x1^2) # unconstrained term
   x2 <- runif(n)*4-1;
   f2 <- exp(4*x2)/(1+exp(4*x2)) # monotone increasing smooth
   f <- f1+f2
   y <- f+rnorm(n)*0.1
   dat <- data.frame(x1=x1,x2=x2,y=y)
   knots <- list(x1=c(-4,-3.5,-2.99,-2.7,-2.5,-1.9,-1.1,-.9,-.3,0.3,.8,1.2,1.9,2.3,
2.7,2.99,3.5,4.1), x2=c(-1.5,-1.2,-1.1, -.89,-.69,-.5,-.3,0,0.7,
0.9,1.1,1.22,1.5,2.2,2.77,2.99,3.1, 3.2,3.6))
   b3 <- scam(y~s(x1,k=15,bs="bs",m=2)+s(x2,bs="mpi", k=15,m=2),
         knots=knots,data=dat)

## setting knots for f(x2) only...
   knots <- list(x2=c(-1.5,-1.2,-1.1, -.89,-.69,-.5,-.3,
   0,0.7,0.9,1.1,1.22,1.5,2.2,2.77,2.99,3.1, 3.2,3.6))
   b4 <- scam(y~s(x1,k=15,bs="bs")+s(x2,bs="mpi", k=15),
       knots=knots,data=dat)

## End(Not run)
```

`smooth.construct.po.smooth.spec`

Constructor for monotone increasing P-splines in SCAMs

Description

This is a special method function for creating univariate smooths subject to a positivity constraint which is built by the `mgcv` constructor function for smooth terms, `smooth.construct`. It is constructed using monotonic P-splines. This smooth is specified via model terms suach as `s(x,k,bs="po",m=2)`, where `k` denotes the basis dimension and `m+1` is the order of the B-spline basis.

Note: currently this smooth can work only with models with no intercept. See examples below.

Usage

```
## S3 method for class 'po.smooth.spec'
smooth.construct(object, data, knots)
```

Arguments

`object`	A smooth specification object, generated by an `s` term in a GAM formula.
`data`	A data frame or list containing the data required by this term, with names given by `object$term`. The `by` variable is the last element.
`knots`	An optional list containing the knots supplied for basis setup. If it is `NULL` then the knot locations are generated automatically.

Value

An object of class `"po.smooth"`.

Author(s)

Natalya Pya <nat.pya@gmail.com>

References

Pya, N. and Wood, S.N. (2015) Shape constrained additive models. Statistics and Computing, 25(3), 543-559

Pya, N. (2010) Additive models with shape constraints. PhD thesis. University of Bath. Department of Mathematical Sciences

See Also

smooth.construct.mpd.smooth.spec, smooth.construct.cv.smooth.spec, smooth.construct.cx.smooth.spec, smooth.construct.mdcv.smooth.spec, smooth.construct.mdcx.smooth.spec, smooth.construct.micv.smooth.spec, smooth.construct.micx.smooth.spec

Examples

```
##  SCOP-splines example with positivity constraint...
  ## simulating data...
## Not run:
   set.seed(3)
   n <- 100
   x <- seq(-3,3,length.out=100)
   f <- dnorm(x)
   y <- f + rnorm(n)*0.1
   b <- scam(y~s(x,bs="po")-1)

   b1 <- scam(y~s(x)) ## unconstrained model
   plot(x,y)
   lines(x,f)
   lines(x,fitted(b),col=2)
   lines(x,fitted(b1),col=3)

  ## two-term example...
  set.seed(8)
  n <- 200
  x1 <- seq(-3,3,length.out=n)
  f1 <- 3*exp(-x1^2) ## positively constrained smooth
  x2 <- runif(n)*3-1;
  f2 <- exp(4*x2)/(1+exp(4*x2)) # monotone increasing smooth
  f <- f1+f2
  y <- f+rnorm(n)*0.3
  dat <- data.frame(x1=x1,x2=x2,y=y)
  ## fit model, results, and plot...
  b2 <- scam(y~s(x1,bs="po")+s(x2,k=15,bs="mpi")-1,data=dat)
  summary(b2)
  plot(b2,pages=1)

  b3 <- scam(y~s(x1,bs="ps")+s(x2,bs="ps"),data=dat) ## unconstrained model
```

```
  summary(b3)
  plot(b3,pages=1)

## End(Not run)
```

---

smooth.construct.tecvcv.smooth.spec
Tensor product smoothing constructor for bivariate function subject to double concavity constraint

---

Description

This is a special method function for creating tensor product bivariate smooths subject to double concavity constraint, i.e. concavity constraint wrt both the first and the second covariates. This is built by the mgcv constructor function for smooth terms, smooth.construct. It is constructed from a pair of single penalty marginal smooths which are represented using the B-spline basis functions. This tensor product is specified by model terms such as s(x1,x2,k=c(q1,q2),bs="tecvcv",m=c(2,2)), where q1 and q2 denote the basis dimensions for the marginal smooths.

Usage

```
## S3 method for class 'tecvcv.smooth.spec'
smooth.construct(object, data, knots)
```

Arguments

object	A smooth specification object, generated by an s term in a GAM formula.
data	A data frame or list containing the values of the elements of object$term, with names given by object$term.
knots	An optional list containing the knots corresponding to object$term. If it is NULL then the knot locations are generated automatically.

Value

An object of class "tecvcv.smooth". In addition to the usual elements of a smooth class documented under smooth.construct of the mgcv library, this object contains:

p.ident	A vector of 0's and 1's for model parameter identification: 1's indicate parameters which will be exponentiated, 0's - otherwise.
Zc	A matrix of identifiability constraints.

Author(s)

Natalya Pya <nat.pya@gmail.com>

References

Pya, N. and Wood, S.N. (2015) Shape constrained additive models. Statistics and Computing, 25(3), 543-559

See Also

smooth.construct.tedmd.smooth.spec smooth.construct.temicx.smooth.spec smooth.construct.tedecx.smooth
smooth.construct.tecxcx.smooth.spec smooth.construct.tecxcv.smooth.spec

Examples

```
 ## Not run:
## tensor product `tecvcv' example
  ## simulating data...
   set.seed(3)
   n <- 30
   x1 <- sort(2*runif(n)-1)
   x2 <- sort(2*runif(n)-1)
   f1 <- matrix(0,n,n)
   for (i in 1:n) for (j in 1:n)
        f1[i,j] <- -4*(x1[i]^2+x2[j]^2)
   f <- as.vector(t(f1))
   y <- f+rnorm(length(f))*.05
   x11 <-  matrix(0,n,n)
   x11[,1:n] <- x1
   x11 <- as.vector(t(x11))
   x22 <- rep(x2,n)
   dat <- list(x1=x11,x2=x22,y=y)
## fit model ...
   b <- scam(y~s(x1,x2,k=c(10,10),bs="tecvcv"), data=dat)
## plot results ...
   par(mfrow=c(2,2),mar=c(4,4,2,2))
   plot(b,se=TRUE)
   plot(b,pers=TRUE,theta = 30, phi = 40)
   plot(y,b$fitted.values,xlab="Simulated data",ylab="Fitted data")
   x11()
   vis.scam(b, theta=30,phi=40)
   ## plotting the truth...
   x11()
   x1 <- seq(min(x1),max(x1),length.out=30)
   x2 <- seq(min(x2),max(x2),length.out=30)
   f1 <- matrix(0,n,n)
   for (i in 1:n) for (j in 1:n) f1[i,j] <- -4*(x1[i]^2+x2[j]^2)
   persp(x1,x2,f1,theta = 30, phi = 40)

## End(Not run)
```

smooth.construct.tecxcv.smooth.spec

Tensor product smoothing constructor for bivariate function subject to mixed constraints: convexity constraint wrt the first covariate and concavity wrt the second one

Description

This is a special method function for creating tensor product bivariate smooths subject to mixed constraints, convexity constraint wrt the first covariate and concavity wrt the second one. This is built by the `mgcv` constructor function for smooth terms, `smooth.construct`. It is constructed from a pair of single penalty marginal smooths which are represented using the B-spline basis functions. This tensor product is specified by model terms such as `s(x1,x2,k=c(q1,q2),bs="tecxcv",m=c(2,2))`, where `q1` and `q2` denote the basis dimensions for the marginal smooths.

Usage

```
## S3 method for class 'tecxcv.smooth.spec'
smooth.construct(object, data, knots)
```

Arguments

`object`	A smooth specification object, generated by an `s` term in a GAM formula.
`data`	A data frame or list containing the values of the elements of `object$term`, with names given by `object$term`.
`knots`	An optional list containing the knots corresponding to `object$term`. If it is `NULL` then the knot locations are generated automatically.

Value

An object of class `"tecxcv.smooth"`. In addition to the usual elements of a smooth class documented under `smooth.construct` of the `mgcv` library, this object contains:

`p.ident`	A vector of 0's and 1's for model parameter identification: 1's indicate parameters which will be exponentiated, 0's - otherwise.
`Zc`	A matrix of identifiability constraints.

Author(s)

Natalya Pya <nat.pya@gmail.com>

References

Pya, N. and Wood, S.N. (2015) Shape constrained additive models. Statistics and Computing, 25(3), 543-559

See Also

smooth.construct.tedmd.smooth.spec smooth.construct.tedecv.smooth.spec smooth.construct.tedecx.smooth.spec
smooth.construct.tecvcv.smooth.spec smooth.construct.tecxcx.smooth.spec

Examples

```
 ## Not run: 
## tensor product `tecxcv' example
  ## simulating data...
   set.seed(5)
   n <- 30
   x1 <- sort(2*runif(n)-1)
   x2 <- sort(2*runif(n)-1)
   f1 <- matrix(0,n,n)
   for (i in 1:n) for (j in 1:n)
        f1[i,j] <- 2*x1[i]^2 - 4*x2[j]^2
   f <- as.vector(t(f1))
   y <- f+rnorm(length(f))*.05
   x11 <-  matrix(0,n,n)
   x11[,1:n] <- x1
   x11 <- as.vector(t(x11))
   x22 <- rep(x2,n)
   dat <- list(x1=x11,x2=x22,y=y)
## fit model ...
   b <- scam(y~s(x1,x2,k=c(10,10),bs="tecxcv"), data=dat)
## plot results ...
   par(mfrow=c(2,2),mar=c(4,4,2,2))
   plot(b,se=TRUE)
   plot(b,pers=TRUE,theta = 30, phi = 40)
   plot(y,b$fitted.values,xlab="Simulated data",ylab="Fitted data")
   x11()
   vis.scam(b,theta=30,phi=40)
   ## plotting the truth...
   x11()
   x1 <- seq(min(x1),max(x1),length.out=30)
   x2 <- seq(min(x2),max(x2),length.out=30)
   f1 <- matrix(0,n,n)
   for (i in 1:n) for (j in 1:n) f1[i,j] <- 2*x1[i]^2 - 4*x2[j]^2
   persp(x1,x2,f1,theta = 30, phi = 40)

## End(Not run)
```

---

smooth.construct.tecxcx.smooth.spec
Tensor product smoothing constructor for bivariate function subject to double convexity constraint

---

Description

This is a special method function for creating tensor product bivariate smooths subject to double convexity constraint, convexity constraint wrt both the first and the second covariates. This is built by the `mgcv` constructor function for smooth terms, `smooth.construct`. It is constructed from a pair of single penalty marginal smooths which are represented using the B-spline basis functions. This tensor product is specified by model terms such as `s(x1,x2,k=c(q1,q2),bs="tecxcx",m=c(2,2))`, where `q1` and `q2` denote the basis dimensions for the marginal smooths.

Usage

```
## S3 method for class 'tecxcx.smooth.spec'
smooth.construct(object, data, knots)
```

Arguments

`object`	A smooth specification object, generated by an `s` term in a GAM formula.
`data`	A data frame or list containing the values of the elements of `object$term`, with names given by `object$term`.
`knots`	An optional list containing the knots corresponding to `object$term`. If it is `NULL` then the knot locations are generated automatically.

Value

An object of class `"tecxcx.smooth"`. In addition to the usual elements of a smooth class documented under `smooth.construct` of the `mgcv` library, this object contains:

`p.ident`	A vector of 0's and 1's for model parameter identification: 1's indicate parameters which will be exponentiated, 0's - otherwise.
`Zc`	A matrix of identifiability constraints.

Author(s)

Natalya Pya <nat.pya@gmail.com>

References

Pya, N. and Wood, S.N. (2015) Shape constrained additive models. Statistics and Computing, 25(3), 543-559

See Also

`smooth.construct.tedmd.smooth.spec` `smooth.construct.tedecv.smooth.spec` `smooth.construct.tedecx.smooth`
`smooth.construct.tecvcv.smooth.spec` `smooth.construct.tecxcv.smooth.spec`

Examples

```
 ## Not run:
## tensor product `tecxcx' example
  ## simulating data...
   set.seed(1)
   n <- 30
   x1 <- sort(2*runif(n)-1)
   x2 <- sort(2*runif(n)-1)
   f1 <- matrix(0,n,n)
   for (i in 1:n) for (j in 1:n)
       { f1[i,j] <- 2*(x1[i]^2 + x2[j]^2)}
   f <- as.vector(t(f1))
   y <- f+rnorm(length(f))*.05
   x11 <-  matrix(0,n,n)
   x11[,1:n] <- x1
   x11 <- as.vector(t(x11))
   x22 <- rep(x2,n)
   dat <- list(x1=x11,x2=x22,y=y)
## fit model ...
   b <- scam(y~s(x1,x2,k=c(10,10),bs="tecxcx"), data=dat)
## plot results ...
   par(mfrow=c(2,2),mar=c(4,4,2,2))
   plot(b,se=TRUE)
   plot(b,pers=TRUE,theta = 30, phi = 40)
   plot(y,b$fitted.values,xlab="Simulated data",ylab="Fitted data")
   x11()
   vis.scam(b,theta=20,phi=20)
   ## plotting the truth...
   x11()
   x1 <- seq(min(x1),max(x1),length.out=30)
   x2 <- seq(min(x2),max(x2),length.out=30)
   f1 <- matrix(0,n,n)
   for (i in 1:n) for (j in 1:n) f1[i,j] <- 2*(x1[i]^2 + x2[j]^2)
   persp(x1,x2,f1,theta = 30, phi = 40)

## End(Not run)
```

`smooth.construct.tedecv.smooth.spec`

Tensor product smoothing constructor for bivariate function subject to mixed constraints: monotone decreasing constraint wrt the first covariate and concavity wrt the second one

---

Description

This is a special method function for creating tensor product bivariate smooths subject to mixed constraints, monotone decreasing constraint wrt the first covariate and concavity wrt the second one, which is built by the mgcv constructor function for smooth terms, smooth.construct. It is constructed from a pair of single penalty marginal smooths which are represented using the B-spline basis functions. This tensor product is specified by model terms such as s(x1,x2,k=c(q1,q2),bs="tedecv",m=c(2,2)), where q1 and q2 denote the basis dimensions for the marginal smooths.

Usage

```
## S3 method for class 'tedecv.smooth.spec'
smooth.construct(object, data, knots)
```

Arguments

object	A smooth specification object, generated by an s term in a GAM formula.
data	A data frame or list containing the values of the elements of object$term, with names given by object$term.
knots	An optional list containing the knots corresponding to object$term. If it is NULL then the knot locations are generated automatically.

Value

An object of class "tedecv.smooth". In addition to the usual elements of a smooth class documented under smooth.construct of the mgcv library, this object contains:

p.ident	A vector of 0's and 1's for model parameter identification: 1's indicate parameters which will be exponentiated, 0's - otherwise.
Zc	A matrix of identifiability constraints.

Author(s)

Natalya Pya <nat.pya@gmail.com>

References

Pya, N. and Wood, S.N. (2015) Shape constrained additive models. Statistics and Computing, 25(3), 543-559

See Also

smooth.construct.tedmd.smooth.spec smooth.construct.temicx.smooth.spec smooth.construct.tedecx.smooth

Examples

```
 ## Not run:
## tensor product `tedecv' example
  ## simulating data...

   set.seed(2)
   n <- 30
   x1 <- sort(runif(n)*4-1)
   x2 <- sort(2*runif(n)-1)
```

```
    f1 <- matrix(0,n,n)
    for (i in 1:n) for (j in 1:n)
        { f1[i,j] <- -exp(4*x1[i])/(1+exp(4*x1[i]))- 4*x2[j]^2}
    f <- as.vector(t(f1))
    y <- f+rnorm(length(f))*0.1
    x11 <-  matrix(0,n,n)
    x11[,1:n] <- x1
    x11 <- as.vector(t(x11))
    x22 <- rep(x2,n)
    dat <- list(x1=x11,x2=x22,y=y)
 ## fit model ...
    b <- scam(y~s(x1,x2,k=c(10,10),bs="tedecv",m=2), data=dat)
 ## plot results ...
    par(mfrow=c(2,2),mar=c(4,4,2,2))
    plot(b,se=TRUE)
    plot(b,pers=TRUE,theta = 30, phi = 40)
    plot(y,b$fitted.values,xlab="Simulated data",ylab="Fitted data")
    x11()
    vis.scam(b, theta=30)
    ## plotting the truth...
    x11()
    x1 <- seq(min(x1),max(x1),length.out=30)
    x2 <- seq(min(x2),max(x2),length.out=30)
    f1 <- matrix(0,n,n)
    for (i in 1:n) for (j in 1:n) f1[i,j] <- -exp(4*x1[i])/(1+exp(4*x1[i]))- 4*x2[j]^2
    persp(x1,x2,f1,theta = 30, phi = 40)

 ## End(Not run)
```

`smooth.construct.tedecx.smooth.spec`

Tensor product smoothing constructor for bivariate function subject to mixed constraints: monotone decreasing constraint wrt the first covariate and convexity wrt the second one

---

Description

This is a special method function for creating tensor product bivariate smooths subject to mixed constraints, monotone decreasing constraint wrt the first covariate and convexity wrt the second one, which is built by the `mgcv` constructor function for smooth terms, `smooth.construct`. It is constructed from a pair of single penalty marginal smooths which are represented using the B-spline basis functions. This tensor product is specified by model terms such as `s(x1,x2,k=c(q1,q2),bs="tedecx",m=c(2,2))`, where `q1` and `q2` denote the basis dimensions for the marginal smooths.

Usage

```
## S3 method for class 'tedecx.smooth.spec'
smooth.construct(object, data, knots)
```

Arguments

object — A smooth specification object, generated by an s term in a GAM formula.

data — A data frame or list containing the values of the elements of object$term, with names given by object$term.

knots — An optional list containing the knots corresponding to object$term. If it is NULL then the knot locations are generated automatically.

Value

An object of class "tedecx.smooth". In addition to the usual elements of a smooth class documented under smooth.construct of the mgcv library, this object contains:

p.ident — A vector of 0's and 1's for model parameter identification: 1's indicate parameters which will be exponentiated, 0's - otherwise.

Zc — A matrix of identifiability constraints.

Author(s)

Natalya Pya <nat.pya@gmail.com>

References

Pya, N. and Wood, S.N. (2015) Shape constrained additive models. Statistics and Computing, 25(3), 543-559

See Also

smooth.construct.tedmd.smooth.spec smooth.construct.tedecv.smooth.spec

Examples

```
 ## Not run:
## tensor product `tedecx' example
  ## simulating data...

   set.seed(2)
   n <- 30
   x1 <- sort(runif(n)*4-1)
   x2 <- sort(2*runif(n)-1)
   f1 <- matrix(0,n,n)
   for (i in 1:n) for (j in 1:n)
        f1[i,j] <- -exp(4*x1[i])/(1+exp(4*x1[i])) + 2*x2[j]^2
   f <- as.vector(t(f1))
   y <- f+rnorm(length(f))*0.05
   x11 <-  matrix(0,n,n)
   x11[,1:n] <- x1
   x11 <- as.vector(t(x11))
   x22 <- rep(x2,n)
   dat <- list(x1=x11,x2=x22,y=y)
## fit model ...
```

```
   b <- scam(y~s(x1,x2,k=c(10,10),bs="tedecx",m=2), not.exp=TRUE, data=dat)
## b1 <- scam(y~s(x1,bs="mpd",m=2)+s(x2,bs="cx",m=2), data=dat)
## plot results ...
   par(mfrow=c(2,2),mar=c(4,4,2,2))
   plot(b,se=TRUE)
   plot(b,pers=TRUE,theta = 30, phi = 40)
   plot(y,b$fitted.values,xlab="Simulated data",ylab="Fitted data")
   x11()
   vis.scam(b,theta=20,phi=20)
   ## plotting the truth...
   x11()
   x1 <- seq(min(x1),max(x1),length.out=30)
   x2 <- seq(min(x2),max(x2),length.out=30)
   f1 <- matrix(0,n,n)
   for (i in 1:n) for (j in 1:n) f1[i,j] <- -exp(4*x1[i])/(1+exp(4*x1[i])) + 2*x2[j]^2
   persp(x1,x2,f1,theta = 30, phi = 40)

## End(Not run)
```

`smooth.construct.tedmd.smooth.spec`
Tensor product smoothing constructor for bivariate function subject to double monotone decreasing constraint

---

Description

This is a special method function for creating tensor product bivariate smooths subject to double monotone decreasing constraints which is built by the mgcv constructor function for smooth terms, `smooth.construct`. It is constructed from a pair of single penalty marginal smooths which are represented using the B-spline basis functions. This tensor product is specified by model terms such as `s(x1,x2,k=c(q1,q2),bs="tedmd",m=c(2,2))`, where `q1` and `q2` denote the basis dimensions for the marginal smooths.

Usage

```
## S3 method for class 'tedmd.smooth.spec'
smooth.construct(object, data, knots)
```

Arguments

`object`	A smooth specification object, generated by an `s` term in a GAM formula.
`data`	A data frame or list containing the values of the elements of `object$term`, with names given by `object$term`.
`knots`	An optional list containing the knots corresponding to `object$term`. If it is `NULL` then the knot locations are generated automatically.

Value

An object of class "tedmd.smooth". In addition to the usual elements of a smooth class documented under smooth.construct of the mgcv library, this object contains:

p.ident
: A vector of 0's and 1's for model parameter identification: 1's indicate parameters which will be exponentiated, 0's - otherwise.

Zc
: A matrix of identifiability constraints.

Author(s)

Natalya Pya <nat.pya@gmail.com>

References

Pya, N. and Wood, S.N. (2015) Shape constrained additive models. Statistics and Computing, 25(3), 543-559

Pya, N. (2010) Additive models with shape constraints. PhD thesis. University of Bath. Department of Mathematical Sciences

See Also

smooth.construct.tedmi.smooth.spec

Examples

```
 ## Not run:
## tensor product `tedmd' example
  ## simulating data...
   set.seed(2)
   n <- 30
   x1 <- sort(runif(n)*4-1)
   x2 <- sort(runif(n))
   f1 <- matrix(0,n,n)
   for (i in 1:n) for (j in 1:n)
       { f1[i,j] <- -exp(4*x1[i])/(1+exp(4*x1[i]))-2*exp(x2[j]-0.5)}
   f <- as.vector(t(f1))
   y <- f+rnorm(length(f))*0.1
   x11 <-  matrix(0,n,n)
   x11[,1:n] <- x1
   x11 <- as.vector(t(x11))
   x22 <- rep(x2,n)
   dat <- list(x1=x11,x2=x22,y=y)
## fit model ...
   b <- scam(y~s(x1,x2,k=c(10,10),bs="tedmd",m=2),
            family=gaussian(link="identity"), data=dat)
## plot results ...
   par(mfrow=c(2,2),mar=c(4,4,2,2))
   plot(b,se=TRUE)
   plot(b,pers=TRUE,theta = 80, phi = 40)
   plot(y,b$fitted.values,xlab="Simulated data",ylab="Fitted data")
```

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

---

smooth.construct.tedmi.smooth.spec
Tensor product smoothing constructor for bivariate function subject to double monotone increasing constraint

---

Description

This is a special method function for creating tensor product bivariate smooths subject to double monotone increasing constraints which is built by the mgcv constructor function for smooth terms, smooth.construct. It is constructed from a pair of single penalty marginal smooths which are represented using the B-spline basis functions. This tensor product is specified by model terms such as s(x1,x2,k=c(q1,q2),bs="tedmi",m=c(2,2)), where q1 and q2 denote the basis dimensions for the marginal smooths.

Usage

```
## S3 method for class 'tedmi.smooth.spec'
smooth.construct(object, data, knots)
```

Arguments

object	A smooth specification object, generated by an s term in a GAM formula.
data	A data frame or list containing the values of the elements of object$term, with names given by object$term.
knots	An optional list containing the knots corresponding to object$term. If it is NULL then the knot locations are generated automatically.

Value

An object of class "tedmi.smooth". In addition to the usual elements of a smooth class documented under smooth.construct of the mgcv library, this object contains:

p.ident	A vector of 0's and 1's for model parameter identification: 1's indicate parameters which will be exponentiated, 0's - otherwise.
Zc	A matrix of identifiability constraints.

Author(s)

Natalya Pya <nat.pya@gmail.com>

References

Pya, N. and Wood, S.N. (2015) Shape constrained additive models. Statistics and Computing, 25(3), 543-559

Pya, N. (2010) Additive models with shape constraints. PhD thesis. University of Bath. Department of Mathematical Sciences

See Also

smooth.construct.tedmd.smooth.spec

Examples

```
 ## Not run:
## tensor product `tedmi' example
  ## simulating data...
   set.seed(1)
   n <- 30
   x1 <- sort(runif(n)*4-1)
   x2 <- sort(runif(n))
   f1 <- matrix(0,n,n)
   for (i in 1:n) for (j in 1:n)
       { f1[i,j] <- exp(4*x1[i])/(1+exp(4*x1[i]))+2*exp(x2[j]-0.5)}
   f <- as.vector(t(f1))
   y <- f+rnorm(length(f))*0.1
   x11 <-  matrix(0,n,n)
   x11[,1:n] <- x1
   x11 <- as.vector(t(x11))
   x22 <- rep(x2,n)
   dat <- list(x1=x11,x2=x22,y=y)
## fit model ...
   b <- scam(y~s(x1,x2,k=c(10,10),bs="tedmi",m=2),family=gaussian, data=dat)
## plot results ...
   par(mfrow=c(2,2),mar=c(4,4,2,2))
   plot(b,se=TRUE)
   plot(b,pers=TRUE,theta = 30, phi = 40)
   plot(y,b$fitted.values,xlab="Simulated data",ylab="Fitted data")

## End(Not run)
```

---

smooth.construct.temicv.smooth.spec

Tensor product smoothing constructor for bivariate function subject to mixed constraints: monotone increasing constraint wrt the first covariate and concavity wrt the second one

---

Description

This is a special method function for creating tensor product bivariate smooths subject to mixed constraints, monotone increasing constraint wrt the first covariate and concavity wrt the second one, which is built by the mgcv constructor function for smooth terms, smooth.construct. It is constructed from a pair of single penalty marginal smooths which are represented using the B-spline basis functions. This tensor product is specified by model terms such as s(x1,x2,k=c(q1,q2),bs="temicv",m=c(2,2)), where q1 and q2 denote the basis dimensions for the marginal smooths.

Usage

```
## S3 method for class 'temicv.smooth.spec'
smooth.construct(object, data, knots)
```

Arguments

object	A smooth specification object, generated by an s term in a GAM formula.
data	A data frame or list containing the values of the elements of object$term, with names given by object$term.
knots	An optional list containing the knots corresponding to object$term. If it is NULL then the knot locations are generated automatically.

Value

An object of class "temicv.smooth". In addition to the usual elements of a smooth class documented under smooth.construct of the mgcv library, this object contains:

p.ident	A vector of 0's and 1's for model parameter identification: 1's indicate parameters which will be exponentiated, 0's - otherwise.
Zc	A matrix of identifiability constraints.

Author(s)

Natalya Pya <nat.pya@gmail.com>

References

Pya, N. and Wood, S.N. (2015) Shape constrained additive models. Statistics and Computing, 25(3), 543-559

See Also

smooth.construct.tedmd.smooth.spec smooth.construct.temicx.smooth.spec

Examples

```
 ## Not run:
## tensor product `temicv' example
  ## simulating data...
   set.seed(2)
   n <- 30
   x1 <- sort(runif(n)*4-1)
   x2 <- sort(2*runif(n)-1)
   f1 <- matrix(0,n,n)
   for (i in 1:n) for (j in 1:n)
        f1[i,j] <- exp(4*x1[i])/(1+exp(4*x1[i])) - 4*x2[j]^2
   f <- as.vector(t(f1))
   y <- f+rnorm(length(f))*0.1
   x11 <-  matrix(0,n,n)
   x11[,1:n] <- x1
```

```
    x11 <- as.vector(t(x11))
    x22 <- rep(x2,n)
    dat <- list(x1=x11,x2=x22,y=y)
 ## fit model ...
    b <- scam(y~s(x1,x2,k=c(10,10),bs="temicv",m=2), data=dat)
 ## plot results ...
    par(mfrow=c(2,2),mar=c(4,4,2,2))
    plot(b,se=TRUE)
    plot(b,pers=TRUE,theta = 30, phi = 40)
    plot(y,b$fitted.values,xlab="Simulated data",ylab="Fitted data")
    x11()
    vis.scam(b, theta=30, phi = 40)
    ## plotting the truth...
    x11()
    x1 <- seq(min(x1),max(x1),length.out=30)
    x2 <- seq(min(x2),max(x2),length.out=30)
    f1 <- matrix(0,n,n)
    for (i in 1:n) for (j in 1:n) f1[i,j] <- exp(4*x1[i])/(1+exp(4*x1[i])) - 4*x2[j]^2
    persp(x1,x2,f1,theta = 30, phi = 40)

 ## End(Not run)
```

`smooth.construct.temicx.smooth.spec`

Tensor product smoothing constructor for bivariate function subject to mixed constraints: monotone increasing constraint wrt the first covariate and convexity wrt the second one

---

Description

This is a special method function for creating tensor product bivariate smooths subject to mixed constraints, monotone increasing constraint wrt the first covariate and convexity wrt the second one, which is built by the `mgcv` constructor function for smooth terms, `smooth.construct`. It is constructed from a pair of single penalty marginal smooths which are represented using the B-spline basis functions. This tensor product is specified by model terms such as `s(x1,x2,k=c(q1,q2),bs="temicx",m=c(2,2))`, where `q1` and `q2` denote the basis dimensions for the marginal smooths.

Usage

```
## S3 method for class 'temicx.smooth.spec'
smooth.construct(object, data, knots)
```

Arguments

`object`	A smooth specification object, generated by an `s` term in a GAM formula.
`data`	A data frame or list containing the values of the elements of `object$term`, with names given by `object$term`.
`knots`	An optional list containing the knots corresponding to `object$term`. If it is `NULL` then the knot locations are generated automatically.

Value

An object of class "temicx.smooth". In addition to the usual elements of a smooth class documented under smooth.construct of the mgcv library, this object contains:

p.ident A vector of 0's and 1's for model parameter identification: 1's indicate parameters which will be exponentiated, 0's - otherwise.

Zc A matrix of identifiability constraints.

Author(s)

Natalya Pya <nat.pya@gmail.com>

References

Pya, N. and Wood, S.N. (2015) Shape constrained additive models. Statistics and Computing, 25(3), 543-559

See Also

smooth.construct.tedmd.smooth.spec

Examples

```
 ## Not run:
## tensor product `temicx' example
  ## simulating data...

   set.seed(1)
   n <- 30
   x1 <- sort(runif(n)*4-1)
   x2 <- sort(2*runif(n)-1)
   f1 <- matrix(0,n,n)
   for (i in 1:n) for (j in 1:n)
        f1[i,j] <- exp(4*x1[i])/(1+exp(4*x1[i])) + 2*x2[j]^2
   f <- as.vector(t(f1))
   y <- f+rnorm(length(f))*0.1
   x11 <-  matrix(0,n,n)
   x11[,1:n] <- x1
   x11 <- as.vector(t(x11))
   x22 <- rep(x2,n)
   dat <- list(x1=x11,x2=x22,y=y)
## fit model ...
   b <- scam(y~s(x1,x2,k=c(10,10),bs="temicx",m=2), data=dat)
## plot results ...
   par(mfrow=c(2,2),mar=c(4,4,2,2))
   plot(b,se=TRUE)
   plot(b,pers=TRUE,theta = 30, phi = 40)
   plot(y,b$fitted.values,xlab="Simulated data",ylab="Fitted data")
   x11()
   vis.scam(b,theta = 30, phi = 40)
   ## plotting the truth...
```

```
x11()
x1 <- seq(min(x1),max(x1),length.out=30)
x2 <-  seq(min(x2),max(x2),length.out=30)
f1 <- matrix(0,n,n)
for (i in 1:n) for (j in 1:n) f1[i,j] <- exp(4*x1[i])/(1+exp(4*x1[i])) + 2*x2[j]^2
persp(x1,x2,f1,theta = 30, phi = 40)

## End(Not run)
```

`smooth.construct.tescv.smooth.spec`

Tensor product smoothing constructor for a bivariate function concave in the second covariate

---

Description

This is a special method function for creating tensor product bivariate smooths concave in the second covariate which is built by the `mgcv` constructor function for smooth terms, `smooth.construct`. It is constructed from a pair of single penalty marginal smooths. This tensor product is specified by model terms such as `s(x1,x2,k=c(q1,q2),bs="tescv",m=c(2,2))`, where the basis for the first marginal smooth is specified in the second element of `bs`.

Usage

```
## S3 method for class 'tescv.smooth.spec'
smooth.construct(object, data, knots)
```

Arguments

`object`	A smooth specification object, generated by an `s` term in a GAM formula.
`data`	A data frame or list containing the values of the elements of `object$term`, with names given by `object$term`.
`knots`	An optional list containing the knots corresponding to `object$term`. If it is `NULL` then the knot locations are generated automatically.

Value

An object of class `"tescv.smooth"`. In addition to the usual elements of a smooth class documented under `smooth.construct` of the `mgcv` library, this object contains:

`p.ident`	A vector of 0's and 1's for model parameter identification: 1's indicate parameters which will be exponentiated, 0's - otherwise.
`Zc`	A matrix of identifiability constraints.
`margin.class`	The class of the first unconstrained marginal smooth.

Author(s)

Natalya Pya <nat.pya@gmail.com>

References

Pya, N. and Wood, S.N. (2015) Shape constrained additive models. Statistics and Computing, 25(3), 543-559

See Also

smooth.construct.temicv.smooth.spec smooth.construct.temicx.smooth.spec smooth.construct.tedecv.smoo
smooth.construct.tedecx.smooth.spec smooth.construct.tescx.smooth.spec

Examples

```
 ## Not run:
## tensor product `tescv' example
  ## simulating data...
   set.seed(5)
   n <- 30
   x1 <- sort(runif(n))
   x2 <- sort(2*runif(n)-1)
   f1 <- matrix(0,n,n)
   for (i in 1:n) for (j in 1:n)
        f1[i,j] <- sin(2*x1[i]) - 4*x2[j]^2
   f1 <- as.vector(t(f1))
   f <- (f1-min(f1))/(max(f1)-min(f1))
   y <- f+rnorm(length(f))*0.1
   x11 <-  matrix(0,n,n)
   x11[,1:n] <- x1
   x11 <- as.vector(t(x11))
   x22 <- rep(x2,n)
   dat <- list(x1=x11,x2=x22,y=y)
## fit model ...
   b <- scam(y~s(x1,x2,k=c(10,10),bs="tescv",m=2),
            family=gaussian(), data=dat)
## plot results ...
   par(mfrow=c(2,2),mar=c(4,4,2,2))
   plot(b,se=TRUE)
   plot(b,pers=TRUE, theta = 50, phi = 20)
   plot(y,b$fitted.values,xlab="Simulated data",ylab="Fitted data")
   x11()
   vis.scam(b, theta = 50, phi = 20)
   ## plotting the truth...
   x11()
   x1 <- seq(min(x1),max(x1),length.out=30)
   x2 <-  seq(min(x2),max(x2),length.out=30)
   f1 <- matrix(0,n,n)
   for (i in 1:n) for (j in 1:n) f1[i,j] <- sin(2*x1[i]) - 4*x2[j]^2
   persp(x1,x2,f1,theta = 50, phi = 20)

## End(Not run)
```

`smooth.construct.tescx.smooth.spec`

Tensor product smoothing constructor for a bivariate function convex in the second covariate

Description

This is a special method function for creating tensor product bivariate smooths convex in the second covariate which is built by the `mgcv` constructor function for smooth terms, `smooth.construct`. It is constructed from a pair of single penalty marginal smooths. This tensor product is specified by model terms such as `s(x1,x2,k=c(q1,q2),bs="tescx",m=c(2,2))`, where the basis for the first marginal smooth is specified in the second element of `bs`.

Usage

```
## S3 method for class 'tescx.smooth.spec'
smooth.construct(object, data, knots)
```

Arguments

`object`	A smooth specification object, generated by an `s` term in a GAM formula.
`data`	A data frame or list containing the values of the elements of `object$term`, with names given by `object$term`.
`knots`	An optional list containing the knots corresponding to `object$term`. If it is `NULL` then the knot locations are generated automatically.

Value

An object of class `"tescx.smooth"`. In addition to the usual elements of a smooth class documented under `smooth.construct` of the `mgcv` library, this object contains:

`p.ident`	A vector of 0's and 1's for model parameter identification: 1's indicate parameters which will be exponentiated, 0's - otherwise.
`Zc`	A matrix of identifiability constraints.
`margin.class`	The class of the first unconstrained marginal smooth.

Author(s)

Natalya Pya <nat.pya@gmail.com>

References

Pya, N. and Wood, S.N. (2015) Shape constrained additive models. Statistics and Computing, 25(3), 543-559

See Also

smooth.construct.temicv.smooth.spec smooth.construct.temicx.smooth.spec smooth.construct.tedecv.smoo
smooth.construct.tedecx.smooth.spec smooth.construct.tescv.smooth.spec

Examples

```
 ## Not run:
## tensor product `tescx' example
  ## simulating data...
   set.seed(2)
   n <- 30
   x1 <- sort(runif(n))
   x2 <- sort(2*runif(n)-1)
   f1 <- matrix(0,n,n)
   for (i in 1:n) for (j in 1:n)
        f1[i,j] <- sin(x1[i]) + 2*x2[j]^2
   f1 <- as.vector(t(f1))
   f <- (f1-min(f1))/(max(f1)-min(f1))
   y <- f+rnorm(length(f))*0.1
   x11 <-  matrix(0,n,n)
   x11[,1:n] <- x1
   x11 <- as.vector(t(x11))
   x22 <- rep(x2,n)
   dat <- list(x1=x11,x2=x22,y=y)
   ## fit model ...
   b <- scam(y~s(x1,x2,k=c(10,10),bs="tescx",m=2),
            family=gaussian(), data=dat)
   ## plot results ...
   par(mfrow=c(2,2),mar=c(4,4,2,2))
   plot(b,se=TRUE)
   plot(b,pers=TRUE, theta = 50, phi = 20)
   plot(y,b$fitted.values,xlab="Simulated data",ylab="Fitted data")
   x11()
   vis.scam(b, theta = 50, phi = 20)
   ## plotting the truth...
   x11()
   x1 <- seq(min(x1),max(x1),length.out=30)
   x2 <-  seq(min(x2),max(x2),length.out=30)
   f1 <- matrix(0,n,n)
   for (i in 1:n) for (j in 1:n) f1[i,j] <- sin(x1[i]) + 2*x2[j]^2
   persp(x1,x2,f1,theta = 50, phi = 20)

## End(Not run)
```

---

smooth.construct.tesmd1.smooth.spec
Tensor product smoothing constructor for a bivariate function monotone decreasing in the first covariate

---

Description

This is a special method function for creating tensor product bivariate smooths monotone decreasing in the first covariate which is built by the mgcv constructor function for smooth terms, smooth.construct. It is constructed from a pair of single penalty marginal smooths. This tensor product is specified by model terms such as s(x1,x2,k=c(q1,q2),bs="tesmd1",m=2).

Usage

```
## S3 method for class 'tesmd1.smooth.spec'
smooth.construct(object, data, knots)
```

Arguments

object	A smooth specification object, generated by an s term in a GAM formula.
data	A data frame or list containing the values of the elements of object$term, with names given by object$term.
knots	An optional list containing the knots corresponding to object$term. If it is NULL then the knot locations are generated automatically.

Value

An object of class "tesmd1.smooth". In addition to the usual elements of a smooth class documented under smooth.construct of the mgcv library, this object contains:

p.ident	A vector of 0's and 1's for model parameter identification: 1's indicate parameters which will be exponentiated, 0's - otherwise.
Zc	A matrix of identifiability constraints.
margin.class	The class of the second unconstrained marginal smooth.

Author(s)

Natalya Pya <nat.pya@gmail.com>

References

Pya, N. and Wood, S.N. (2015) Shape constrained additive models. Statistics and Computing, 25(3), 543-559

Pya, N. (2010) Additive models with shape constraints. PhD thesis. University of Bath. Department of Mathematical Sciences

See Also

smooth.construct.tesmd2.smooth.spec

Examples

```
 ## Not run:
## tensor product `tesmd1' example
  ## simulating data...
   set.seed(2)
   n <- 30
   x1 <- sort(runif(n)*4-1)
   x2 <- sort(runif(n))
   f1 <- matrix(0,n,n)
   for (i in 1:n) for (j in 1:n)
       { f1[i,j] <- -exp(4*x1[i])/(1+exp(4*x1[i]))+2*sin(pi*x2[j])}
   f <- as.vector(t(f1))
   y <- f+rnorm(length(f))*0.1
   x11 <-  matrix(0,n,n)
   x11[,1:n] <- x1
   x11 <- as.vector(t(x11))
   x22 <- rep(x2,n)
   dat <- list(x1=x11,x2=x22,y=y)
## fit model ...
   b <- scam(y~s(x1,x2,k=c(10,10),bs="tesmd1",m=2),data=dat,sp=NULL)
## plot results ...
   par(mfrow=c(2,2),mar=c(4,4,2,2))
   plot(b,se=TRUE)
   plot(b,pers=TRUE,theta = 30, phi = 40)
   plot(y,b$fitted.values,xlab="Simulated data",ylab="Fitted data")

## End(Not run)
```

---

smooth.construct.tesmd2.smooth.spec

Tensor product smoothing constructor for a bivariate function monotone decreasing in the second covariate

---

Description

This is a special method function for creating tensor product bivariate smooths monotone decreasing in the second covariate which is built by the `mgcv` constructor function for smooth terms, `smooth.construct`. It is constructed from a pair of single penalty marginal smooths. This tensor product is specified by model terms such as `s(x1,x2,k=c(q1,q2),bs="tesmd2",m=c(2,2))`, where the basis for the first marginal smooth is specified in the second element of `bs`.

Usage

```
## S3 method for class 'tesmd2.smooth.spec'
smooth.construct(object, data, knots)
```

Arguments

object	A smooth specification object, generated by an s term in a GAM formula.
data	A data frame or list containing the values of the elements of object$term, with names given by object$term.
knots	An optional list containing the knots corresponding to object$term. If it is NULL then the knot locations are generated automatically.

Value

An object of class "tesmd2.smooth". In addition to the usual elements of a smooth class documented under smooth.construct of the mgcv library, this object contains:

p.ident	A vector of 0's and 1's for model parameter identification: 1's indicate parameters which will be exponentiated, 0's - otherwise.
Zc	A matrix of identifiability constraints.
margin.class	The class of the first unconstrained marginal smooth.

Author(s)

Natalya Pya <nat.pya@gmail.com>

References

Pya, N. and Wood, S.N. (2015) Shape constrained additive models. Statistics and Computing, 25(3), 543-559

Pya, N. (2010) Additive models with shape constraints. PhD thesis. University of Bath. Department of Mathematical Sciences

See Also

smooth.construct.tesmd1.smooth.spec

Examples

```
 ## Not run:
## tensor product `tesmd2' example
  ## simulating data...
   set.seed(2)
   n <- 30
   x1 <- sort(runif(n)*4-1)
   x2 <- sort(runif(n))
   f1 <- matrix(0,n,n)
   for (i in 1:n) for (j in 1:n)
       { f1[i,j] <- exp(4*x1[i])/(1+exp(4*x1[i])) - 2*exp(x2[j]-0.5)}
   f <- as.vector(t(f1))
   y <- f+rnorm(length(f))*0.1
   x11 <-  matrix(0,n,n)
   x11[,1:n] <- x1
   x11 <- as.vector(t(x11))
```

```
   x22 <- rep(x2,n)
   dat <- list(x1=x11,x2=x22,y=y)
## fit model ...
   b <- scam(y~s(x1,x2,k=c(10,10),bs="tesmd2",m=2),data=dat,sp=NULL)
## plot results ...
   par(mfrow=c(2,2),mar=c(4,4,2,2))
   plot(b,se=TRUE)
   plot(b,pers=TRUE,theta = 90, phi = 60)
   plot(y,b$fitted.values,xlab="Simulated data",ylab="Fitted data")
   x11()
   vis.scam(b,theta = 90, phi = 60)

## End(Not run)
```

`smooth.construct.tesmi1.smooth.spec`
Tensor product smoothing constructor for a bivariate function monotone increasing in the first covariate

---

Description

This is a special method function for creating tensor product bivariate smooths monotone increasing in the first covariate which is built by the `mgcv` constructor function for smooth terms, `smooth.construct`. It is constructed from a pair of single penalty marginal smooths. This tensor product is specified by model terms such as `s(x1,x2,k=c(q1,q2),bs="tesmi1",m=2)`, where the basis for the second marginal smooth is specified in the second element of `bs`.

Usage

```
## S3 method for class 'tesmi1.smooth.spec'
smooth.construct(object, data, knots)
```

Arguments

`object`	A smooth specification object, generated by an `s` term in a GAM formula.
`data`	A data frame or list containing the values of the elements of `object$term`, with names given by `object$term`.
`knots`	An optional list containing the knots corresponding to `object$term`. If it is `NULL` then the knot locations are generated automatically.

Value

An object of class `"tesmi1.smooth"`. In addition to the usual elements of a smooth class documented under `smooth.construct` of the `mgcv` library, this object contains:

`p.ident`	A vector of 0's and 1's for model parameter identification: 1's indicate parameters which will be exponentiated, 0's - otherwise.
`Zc`	A matrix of identifiability constraints.
`margin.class`	The class of the second unconstrained marginal smooth.

Author(s)

Natalya Pya <nat.pya@gmail.com>

References

Pya, N. and Wood, S.N. (2015) Shape constrained additive models. Statistics and Computing, 25(3), 543-559

Pya, N. (2010) Additive models with shape constraints. PhD thesis. University of Bath. Department of Mathematical Sciences

See Also

smooth.construct.tesmi2.smooth.spec

Examples

```
  ## Not run:
## tensor product `tesmi1' example
  ## simulating data...
   set.seed(2)
   n <- 30
   x1 <- sort(runif(n)*4-1)
   x2 <- sort(runif(n))
   f1 <- matrix(0,n,n)
   for (i in 1:n) for (j in 1:n)
       { f1[i,j] <- exp(4*x1[i])/(1+exp(4*x1[i]))+2*sin(pi*x2[j])}
   f <- as.vector(t(f1))
   y <- f+rnorm(length(f))*0.1
   x11 <-  matrix(0,n,n)
   x11[,1:n] <- x1
   x11 <- as.vector(t(x11))
   x22 <- rep(x2,n)
   dat <- list(x1=x11,x2=x22,y=y)
## fit model ...
   b <- scam(y~s(x1,x2,k=c(10,10),bs="tesmi1",m=2), data=dat,sp=NULL)
## plot results ...
   par(mfrow=c(2,2))
   plot(b,se=TRUE)
   plot(b,pers=TRUE,theta = 30, phi = 40)
   plot(y,b$fitted.values,xlab="Simulated data",ylab="Fitted data")

## End(Not run)
```

---

smooth.construct.tesmi2.smooth.spec

Tensor product smoothing constructor for a bivariate function monotone increasing in the second covariate

---

Description

This is a special method function for creating tensor product bivariate smooths monotone increasing in the second covariate which is built by the `mgcv` constructor function for smooth terms, `smooth.construct`. It is constructed from a pair of single penalty marginal smooths. This tensor product is specified by model terms such as `s(x1,x2,k=c(q1,q2),bs="tesmi2",m=c(2,2))`, where the basis for the first marginal smooth is specified in the second element of `bs`.

Usage

```
## S3 method for class 'tesmi2.smooth.spec'
smooth.construct(object, data, knots)
```

Arguments

`object`	A smooth specification object, generated by an `s` term in a GAM formula.
`data`	A data frame or list containing the values of the elements of `object$term`, with names given by `object$term`.
`knots`	An optional list containing the knots corresponding to `object$term`. If it is `NULL` then the knot locations are generated automatically.

Value

An object of class `"tesmi2.smooth"`. In addition to the usual elements of a smooth class documented under `smooth.construct` of the `mgcv` library, this object contains:

`p.ident`	A vector of 0's and 1's for model parameter identification: 1's indicate parameters which will be exponentiated, 0's - otherwise.
`Zc`	A matrix of identifiability constraints.
`margin.class`	The class of the first unconstrained marginal smooth.

Author(s)

Natalya Pya <nat.pya@gmail.com>

References

Pya, N. and Wood, S.N. (2015) Shape constrained additive models. Statistics and Computing, 25(3), 543-559

Pya, N. (2010) Additive models with shape constraints. PhD thesis. University of Bath. Department of Mathematical Sciences

See Also

`smooth.construct.tesmi1.smooth.spec`

Examples

```
 ## Not run:
## tensor product `tesmi2' example
  ## simulating data...
   set.seed(2)
   n <- 30
   x1 <- sort(runif(n))
   x2 <- sort(runif(n)*4-1)
   f1 <- matrix(0,n,n)
   for (i in 1:n) for (j in 1:n)
       { f1[i,j] <- 2*sin(pi*x1[i]) +exp(4*x2[j])/(1+exp(4*x2[j]))}
   f1 <- as.vector(t(f1))
   f <- (f1-min(f1))/(max(f1)-min(f1))
   y <- f+rnorm(length(f))*0.1
   x11 <-  matrix(0,n,n)
   x11[,1:n] <- x1
   x11 <- as.vector(t(x11))
   x22 <- rep(x2,n)
   dat <- list(x1=x11,x2=x22,y=y)
## fit model ...
   b <- scam(y~s(x1,x2,k=c(10,10),bs="tesmi2",m=2),
            family=gaussian(link="identity"), data=dat)
## plot results ...
   par(mfrow=c(2,2),mar=c(4,4,2,2))
   plot(b,se=TRUE)
   plot(b,pers=TRUE, theta = 50, phi = 20)
   plot(y,b$fitted.values,xlab="Simulated data",ylab="Fitted data")
   x11()
   vis.scam(b,theta=50,phi=20)

## End(Not run)
```

---

summary.scam *Summary for a SCAM fit*

Description

Takes a fitted scam object produced by scam() and produces various useful summaries from it. The same code as in summary.gam of the mgcv package is used here with slight modifications to accept the exponentiated parameters of the shape constrained smooth terms and the corresponding covariance matrix.

Usage

```
## S3 method for class 'scam'
summary(object, dispersion=NULL, freq=FALSE, ...)

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

Arguments

`object`	a fitted `scam` object as produced by `scam()`.
`x`	a `summary.scam` object produced by `summary.scam()`.
`dispersion`	A known dispersion parameter. `NULL` to use estimate or default (e.g. 1 for Poisson).
`freq`	By default p-values for individual terms are calculated using the Bayesian estimated covariance matrix of the parameter estimators. If this is set to TRUE then the frequentist covariance matrix of the parameters is used instead.
`digits`	controls number of digits printed in output.
`signif.stars`	Should significance stars be printed alongside output.
`...`	other arguments.

Value

`summary.scam` produces the same list of summary information for a fitted `scam` object as in the unconstrained case `summary.gam` except for the last element `BFGS termination condition`.

`p.coeff`	is an array of estimates of the strictly parametric model coefficients.
`p.t`	is an array of the `p.coeff`'s divided by their standard errors.
`p.pv`	is an array of p-values for the null hypothesis that the corresponding parameter is zero. Calculated with reference to the t distribution with the estimated residual degrees of freedom for the model fit if the dispersion parameter has been estimated, and the standard normal if not.
`m`	The number of smooth terms in the model.
`chi.sq`	An array of test statistics for assessing the significance of model smooth terms. See details.
`s.pv`	An array of approximate p-values for the null hypotheses that each smooth term is zero. Be warned, these are only approximate.
`se`	array of standard error estimates for all parameter estimates.
`r.sq`	The adjusted r-squared for the model. Defined as the proportion of variance explained, where original variance and residual variance are both estimated using unbiased estimators. This quantity can be negative if your model is worse than a one parameter constant model, and can be higher for the smaller of two nested models! Note that proportion null deviance explained is probably more appropriate for non-normal errors.
`dev.expl`	The proportion of the null deviance explained by the model.
`edf`	array of estimated degrees of freedom for the model terms.
`residual.df`	estimated residual degrees of freedom.
`n`	number of data.
`gcv`	minimized GCV score for the model, if GCV used.
`ubre`	minimized UBRE score for the model, if UBRE used.
`scale`	estimated (or given) scale parameter.

`family` the family used.

`formula` the original scam formula.

`dispersion` the scale parameter.

`pTerms.df` the degrees of freedom associated with each parameteric term (excluding the constant).

`pTerms.chi.sq` a Wald statistic for testing the null hypothesis that the each parametric term is zero.

`pTerms.pv` p-values associated with the tests that each term is zero. For penalized fits these are approximate. The reference distribution is an appropriate chi-squared when the scale parameter is known, and is based on an F when it is not.

`cov.unscaled` The estimated covariance matrix of the parameters (or estimators if `freq=TRUE`), divided by scale parameter.

`cov.scaled` The estimated covariance matrix of the parameters (estimators if `freq=TRUE`).

`p.table` significance table for parameters

`s.table` significance table for smooths

`p.Terms` significance table for parametric model terms

`BFGS termination condition`
the value of the maximum component of the scaled GCV/UBRE gradient used as stopping condition. This value is printed if the termination code of the BFGS optimization process is not '1' (not full convergence) (see `bfgs_gcv.ubre`for details)

WARNING

The p-values are approximate.

Author(s)

Natalya Pya <nat.pya@gmail.com> based on `mgcv` by Simon Wood

References

Wood S.N. (2006) *Generalized Additive Models: An Introduction with R*. Chapman and Hall/CRC Press.

Pya, N. and Wood, S.N. (2015) Shape constrained additive models. Statistics and Computing, 25(3), 543-559

Pya, N. (2010) Additive models with shape constraints. PhD thesis. University of Bath. Department of Mathematical Sciences

See Also

`scam`

Examples

```
 ## Not run:
## simulating data...
n <- 200
set.seed(1)
x1 <- runif(n)*6-3
f1 <- 3*exp(-x1^2) # unconstrained smooth term
x2 <- runif(n)*4-1;
f2 <- exp(4*x2)/(1+exp(4*x2)) # monotone increasing smooth
x3 <- runif(n)*5;
f3 <- -log(x3)/5  # monotone decreasing smooth
f <- f1+f2+f3
y <- f + rnorm(n)*0.3
dat <- data.frame(x1=x1,x2=x2,x3=x3,y=y)
## fit model ...
b <- scam(y~s(x1,k=15,bs="cr",m=2)+s(x2,k=30,bs="mpi",m=2)+s(x3,k=30,bs="mpd",m=2),
     data=dat)

summary(b)
plot(b,pages=1)

## End(Not run)
```

`vis.scam` *Visualization of SCAM objects*

Description

Produces perspective or contour plot views of `scam` model predictions. The code is a clone of `vis.gam` of the `mgcv` package.

Usage

```
vis.scam(x,view=NULL,cond=list(),n.grid=30,too.far=0,col=NA,
        color="heat",contour.col=NULL,se=-1,type="link",
        plot.type="persp",zlim=NULL,nCol=50,...)
```

Arguments

	The documentation below is the same as in documentation object `vis.gam`.
`x`	a `scam` object, produced by `scam()`
`view`	an array containing the names of the two main effect terms to be displayed on the x and y dimensions of the plot. If omitted the first two suitable terms will be used.
`cond`	a named list of the values to use for the other predictor terms (not in `view`). Variables omitted from this list will have the closest observed value to the median for continuous variables, or the most commonly occuring level for factors. Parametric matrix variables have all the entries in each column set to the observed column entry closest to the column median.

n.grid	The number of grid nodes in each direction used for calculating the plotted surface.
too.far	plot grid nodes that are too far from the points defined by the variables given in view can be excluded from the plot. too.far determines what is too far. The grid is scaled into the unit square along with the view variables and then grid nodes more than too.far from the predictor variables are excluded.
col	The colours for the facets of the plot. If this is NA then if se>0 the facets are transparent, otherwise the colour scheme specified in color is used. If col is not NA then it is used as the facet colour.
color	the colour scheme to use for plots when se<=0. One of "topo", "heat", "cm", "terrain", "gray" or "bw". Schemes "gray" and "bw" also modify the colors used when se>0.
contour.col	sets the colour of contours when using plot.type="contour". Default scheme used if NULL.
se	if less than or equal to zero then only the predicted surface is plotted, but if greater than zero, then 3 surfaces are plotted, one at the predicted values minus se standard errors, one at the predicted values and one at the predicted values plus se standard errors.
type	"link" to plot on linear predictor scale and "response" to plot on the response scale.
plot.type	one of "contour" or "persp".
zlim	a two item array giving the lower and upper limits for the z-axis scale. NULL to choose automatically.
nCol	The number of colors to use in color schemes.
...	other options to pass on to persp, image or contour.

Value

Simply produces a plot.

Author(s)

Simon Wood <simon.wood@r-project.org>

See Also

persp, vis.gam, and scam.

Examples

```
library(scam)

# Example with factor variable
set.seed(0)
fac<-rep(1:4,20)
x <- runif(80)*5;
y <- fac+log(x)/5+rnorm(80)*0.1
```

```
fac <- factor(fac)
b <- scam(y~fac+s(x,bs="mpi"))

vis.scam(b,theta=-35,color="heat") # factor example

# Example with "by" variable

z<-rnorm(80)*0.4
y<-as.numeric(fac)+log(x)*z+rnorm(80)*0.1
b<-scam(y~fac+s(x,by=z))
g <- gam(y~fac+s(x,by=z))

vis.scam(b,theta=-35,color="terrain",cond=list(z=1)) # by variable example
vis.scam(b,view=c("z","x"),theta= 65) # plot against by variable
```

Index