# Package 'mgcv'

March 3, 2016

Version 1.8-12
Author Simon Wood <simon.wood@r-project.org></simon.wood@r-project.org>
Maintainer Simon Wood <simon.wood@r-project.org></simon.wood@r-project.org>
<b>Title</b> Mixed GAM Computation Vehicle with GCV/AIC/REML Smoothness Estimation
<b>Description</b> GAMs, GAMMs and other generalized ridge regression with multiple smoothing parameter estimation by GCV, REML or UBRE/AIC. Includes a gam() function, a wide variety of smoothers, JAGS support and distributions beyond the exponential family.
Priority recommended
<b>Depends</b> R (>= 2.14.0), nlme (>= 3.1-64)
Imports methods, stats, graphics, Matrix
Suggests splines, parallel, survival, MASS
LazyLoad yes
ByteCompile yes
License GPL (>= 2)
NeedsCompilation yes
Repository CRAN
<b>Date/Publication</b> 2016-03-03 07:57:37
R topics documented:
anova.gam
bam
bam.update
bandchol
betar
bug.reports.mgcv
choose.k
columb
concurvity

cox.ph		
cSplineDes		. 25
exclude.too.far		. 26
extract.lme.cov		. 27
family.mgcv		. 29
fix.family.link		. 30
fixDependence		. 31
formula.gam		. 33
formXtViX		. 35
fs.test		. 36
full.score		. 37
gam		. 38
gam.check		. 47
gam.control		. 50
gam.convergence		. 52
gam.fit		
gam.fit3		. 54
gam.models	•	. 57
gam.outer	• •	63
gam.scale	• •	. 65
gam.selection	• •	. 66
gam.side	٠.	. 69
gam.vcomp	• •	. 71
gam2objective		. 71
gamm		. 72
gamObject		. 74
gamSim		. 84
		. 85
gaulss		
get.var		
in.out		
influence.gam		
initial.sp		
inSide		
interpret.gam		
jagam		
ldTweedie		. 97
linear.functional.terms		
logLik.gam		
ls.size		
magic		
magic.post.proc		
mgcv.FAQ		
mgcv.package		
mgcv.parallel		
missing.data		
model.matrix.gam		
mono.con		. 119
mroot		. 120

multinom
mvn
negbin
new.name
notExp
notExp2
null.space.dimension
ocat
pcls
pdIdnot
pdTens
pen.edf
place.knots
plot.gam
polys.plot
predict.bam
predict.gam
Predict.matrix
Predict.matrix.cr.smooth
Predict.matrix.soap.film
print.gam
qq.gam
random.effects
residuals.gam
rig
rmvn
Rrank
rTweedie
s
scat
sdiag
single.index
slanczos
smooth.construct
smooth.construct.ad.smooth.spec
smooth.construct.bs.smooth.spec
smooth.construct.cr.smooth.spec
smooth.construct.ds.smooth.spec
smooth.construct.fs.smooth.spec
smooth.construct.gp.smooth.spec
smooth.construct.mrf.smooth.spec
smooth.construct.ps.smooth.spec
smooth.construct.re.smooth.spec
smooth.construct.so.smooth.spec
smooth.construct.sos.smooth.spec
smooth.construct.t2.smooth.spec
smooth.construct.tensor.smooth.spec
smooth.construct.tp.smooth.spec

anova.gam

smooth.terms
smoothCon
sp.vcov
spasm.construct
step.gam
summary.gam
t2
te
tensor.prod.model.matrix
trichol
Tweedie
uniquecombs
vcov.gam
vis.gam
ziP
ziplss

anova.gam

Index

Approximate hypothesis tests related to GAM fits

256

# **Description**

Performs hypothesis tests relating to one or more fitted gam objects. For a single fitted gam object, Wald tests of the significance of each parametric and smooth term are performed, so interpretation is analogous to drop1 rather than anova.lm (i.e. it's like type III ANOVA, rather than a sequential type I ANOVA). Otherwise the fitted models are compared using an analysis of deviance table: this latter approach should not be use to test the significance of terms which can be penalized to zero. See details.

## Usage

## **Arguments**

```
object,... fitted model objects of class gam as produced by gam().

x an anova.gam object produced by a single model call to anova.gam().

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".
```

anova.gam 5

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 $\operatorname{summary.gam}$ for details.
digits	number of digits to use when printing output.

#### **Details**

If more than one fitted model is provided than anova.glm is used, with the difference in model degrees of freedom being taken as the difference in effective degress of freedom. The p-values resulting from this are only approximate, and must be used with care. The approximation is most accurate when the comparison relates to unpenalized terms, or smoothers with a null space of dimension greater than zero. (Basically we require that the difference terms could be well approximated by unpenalized terms with degrees of freedom approximately the effective degrees of freedom). In simulations the p-values are usually slightly too low. For terms with a zero-dimensional null space (i.e. those which can be penalized to zero) the approximation is often very poor, and significance can be greatly overstated: i.e. p-values are often substantially too low. This case applies to random effect terms.

Note also that in the multi-model call to anova.gam, it is quite possible for a model with more terms to end up with lower effective degrees of freedom, but better fit, than the notionally null model with fewer terms. In such cases it is very rare that it makes sense to perform any sort of test, since there is then no basis on which to accept the notional null model.

If only one model is provided then the significance of each model term is assessed using Wald like tests, conditional on the smoothing parameter estimates: see summary.gam and Wood (2013a,b) for details. The p-values provided here are better justified than in the multi model case, and have close to the correct distribution under the null, unless smoothing parameters are poorly identified. ML or REML smoothing parameter selection leads to the best results in simulations as they tend to avoid occasional severe undersmoothing. In replication of the full simulation study of Scheipl et al. (2008) the tests give almost indistinguishable power to the method recommended there, but slightly too low p-values under the null in their section 3.1.8 test for a smooth interaction (the Scheipl et al. recommendation is not used directly, because it only applies in the Gaussian case, and requires model refits, but it is available in package RLRsim).

In the single model case print. anova. gam is used as the printing method.

By default the p-values for parametric model terms are also based on Wald tests using the Bayesian covariance matrix for the coefficients. This is appropriate when there are "re" terms present, and is otherwise rather similar to the results using the frequentist covariance matrix (freq=TRUE), since the parametric terms themselves are usually unpenalized. Default P-values for parameteric terms that are penalized using the paraPen argument will not be good.

## Value

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

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

print.anova.gam 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> with substantial improvements by Henric Nilsson.

#### 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

```
gam, predict.gam, gam.check, summary.gam
```

# **Examples**

```
library(mgcv)
set.seed(0)
dat <- gamSim(5,n=200,scale=2)

b<-gam(y ~ x0 + s(x1) + s(x2) + s(x3),data=dat)
anova(b)
b1<-gam(y ~ x0 + s(x1) + s(x2),data=dat)
anova(b,b1,test="F")</pre>
```

bam

Generalized additive models for very large datasets

# Description

Fits a generalized additive model (GAM) to a very large data set, the term 'GAM' being taken to include any quadratically penalized GLM. The degree of smoothness of model terms is estimated as part of fitting. In use the function is much like gam, except that the numerical methods are designed for datasets containing upwards of several tens of thousands of data (see Wood, Goude and Shaw, 2015). The advantage of bam is much lower memory footprint than gam, but it can also be much faster, for large datasets. bam can also compute on a cluster set up by the parallel package.

An alternative fitting approach is provided by the discrete==TRUE method. In this case a method based on discretization of covariate values and C code level parallelization (controlled by the nthreads argument instead of the cluster argument) is used. This extends both the data set and model size usable.

## Usage

```
bam(formula,family=gaussian(),data=list(),weights=NULL,subset=NULL,
    na.action=na.omit, offset=NULL,method="fREML",control=list(),
    select=FALSE,scale=0,gamma=1,knots=NULL,sp=NULL,min.sp=NULL,
    paraPen=NULL, chunk.size=10000, rho=0, AR.start=NULL, discrete=FALSE,
    sparse=FALSE, cluster=NULL, nthreads=NA, gc.level=1, use.chol=FALSE,
    samfrac=1,drop.unused.levels=TRUE,G=NULL,fit=TRUE,...)
```

#### Ar

rguments	
formula	A GAM formula (see formula.gam and also gam.models). This is exactly like the formula for a GLM except that smooth terms, s and te can be added to the right hand side to specify that the linear predictor depends on smooth functions of predictors (or linear functionals of these).
family	This is a family object specifying the distribution and link to use in fitting etc. See glm and family for more details. A negative binomial family is provided: see negbin, but only the known theta case is supported by bam.
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.
weights	prior weights on the contribution of the data to the log likelihood. Note that a weight of 2, for example, is equivalent to having made exactly the same observation twice. If you want to reweight the contributions of each datum without changing the overall magnitude of the log likelihood, then you should normalize the weights (e.g. weights <- weights/mean(weights)).
subset	an optional vector specifying a subset of observations to be used in the fitting process.
na.action	a function which indicates what should happen when the data contain 'NA's. The default is set by the 'na.action' setting of 'options', and is 'na.fail' if that is unset. The "factory-fresh" default is 'na.omit'.
offset	Can be 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 used to conform to the behaviour of lm and glm).
method	The smoothing parameter estimation method. "GCV.Cp" to use GCV for unknown scale parameter and Mallows' Cp/UBRE/AIC for known scale. "GACV.Cp" is equivalent, but using GACV in place of GCV. "REML" for REML estimation,

control

A list of fit control parameters to replace defaults returned by gam.control. Any control parameters not supplied stay at their default values.

including of unknown scale, "P-REML" for REML estimation, but using a Pearson estimate of the scale. "ML" and "P-ML" are similar, but using maximum likelihood in place of REML. Default "fREML" uses fast REML computation.

select Should selection penalties be added to the smooth effects, so that they can in principle be penalized out of the model? Has the side effect that smooths no longer have a fixed effect component (improper prior from a Bayesian perspective) allowing REML comparison of models with the same fixed effect structure. scale If this is positive then it is taken as the known scale parameter. Negative signals that the scale paraemter is unknown. 0 signals that the scale parameter is 1 for Poisson and binomial and unknown otherwise. Note that (RE)ML methods can only work with scale parameter 1 for the Poisson and binomial cases. gamma It is sometimes useful to inflate the model degrees of freedom in the GCV or UBRE/AIC score by a constant multiplier. This allows such a multiplier to be supplied. knots this is an optional list containing user specified knot values to be used for basis construction. For most bases the user simply supplies the knots to be used, which must match up with the k value supplied (note that the number of knots is not always just k). See tprs for what happens in the "tp"/"ts" case. Different terms can use different numbers of knots, unless they share a covariate. A vector of smoothing parameters can be provided here. Smoothing parameters sp must be supplied in the order that the smooth terms appear in the model formula. Negative elements indicate that the parameter should be estimated, and hence a mixture of fixed and estimated parameters is possible. If smooths share smoothing parameters then length(sp) must correspond to the number of underlying smoothing parameters. min.sp Lower bounds can be supplied for the smoothing parameters. Note that if this option is used then the smoothing parameters full. sp, in the returned object, will need to be added to what is supplied here to get the smoothing parameters actually multiplying the penalties. length(min.sp) should always be the same as the total number of penalties (so it may be longer than sp, if smooths share smoothing parameters). paraPen optional list specifying any penalties to be applied to parametric model terms. gam.models explains more. chunk.size The model matrix is created in chunks of this size, rather than ever being formed whole. Reset to 4\*p if chunk.size < 4\*p where p is the number of coefficients. rho An AR1 error model can be used for the residuals (based on dataframe order), of Gaussian-identity link models. This is the AR1 correlation parameter. Standardized residuals (approximately uncorrelated under correct model) returned in std.rsd if non zero. AR.start logical variable of same length as data, TRUE at first observation of an independent section of AR1 correlation. Very first observation in data frame does not need this. If NULL then there are no breaks in AR1 correlaion. discrete with method="fREML" it is possible to discretize covariates for storage and efficiency reasons. If discrete is TRUE, a number or a vector of numbers for each smoother term, then discretization happens. If numbers are supplied they give the number of discretization bins. Experimental at present. Deprecated. If all smooths are P-splines and all tensor products are of the form sparse

te(...,bs="ps",np=FALSE) then in principle computation could be made faster

using sparse matrix methods, and you could set this to TRUE. In practice the speed up is disappointing, and the computation is less well conditioned than the default. See details. cluster bam can compute the computationally dominant QR decomposition in parallel using parLapply from the parallel package, if it is supplied with a cluster on which to do this (a cluster here can be some cores of a single machine). See details and example code. nthreads Number of threads to use for non-cluster computation (e.g. combining results from cluster nodes). if NA set to max(1,length(cluster)). gc.level to keep the memory footprint down, it helps to call the garbage collector often, but this takes a substatial amount of time. Setting this to zero means that garbage collection only happens when R decides it should. Setting to 2 gives frequent garbage collection. 1 is in between. use.chol By default bam uses a very stable QR update approach to obtaining the QR decomposition of the model matrix. For well conditioned models an alternative accumulates the crossproduct of the model matrix and then finds its Choleski decomposition, at the end. This is somewhat more efficient, computationally. samfrac For very large sample size Generalized additive models the number of iterations needed for the model fit can be reduced by first fitting a model to a random sample of the data, and using the results to supply starting values. This initial fit is run with sloppy convergence tolerances, so is typically very low cost. samfrac is the sampling fraction to use. 0.1 is often reasonable. drop.unused.levels by default unused levels are dropped from factors before fitting. For some smooths involving factor variables you might want to turn this off. Only do so if you know what you are doing. G if not NULL then this should be the object returned by a previous call to bam with fit=FALSE. Causes all other arguments to be ignored except chunk. size, gamma, nthreads, cluster, rho, gc.level, samfrac, use.chol and method. fit if FALSE then the model is set up for fitting but not estimated, and an object is returned, suitable for passing as the G argument to bam. further arguments for passing on e.g. to gam. fit (such as mustart).

## Details

When discrete=FALSE, bam operates by first setting up the basis characteristics for the smooths, using a representative subsample of the data. Then the model matrix is constructed in blocks using predict.gam. For each block the factor R, from the QR decomposition of the whole model matrix is updated, along with Q'y. and the sum of squares of y. At the end of block processing, fitting takes place, without the need to ever form the whole model matrix.

In the generalized case, the same trick is used with the weighted model matrix and weighted pseudodata, at each step of the PIRLS. Smoothness selection is performed on the working model at each stage (performance oriented iteration), to maintain the small memory footprint. This is trivial to justify in the case of GCV or Cp/UBRE/AIC based model selection, and for REML/ML is justified via the asymptotic multivariate normality of Q'z where z is the IRLS pseudodata.

For full method details see Wood, Goude and Shaw (2015).

Note that POI is not as stable as the default nested iteration used with gam, but that for very large, information rich, datasets, this is unlikely to matter much.

Note also that it is possible to spend most of the computational time on basis evaluation, if an expensive basis is used. In practice this means that the default "tp" basis should be avoided: almost any other basis (e.g. "cr" or "ps") can be used in the 1D case, and tensor product smooths (te) are typically much less costly in the multi-dimensional case.

If cluster is provided as a cluster set up using makeCluster (or makeForkCluster) from the parallel package, then the rate limiting QR decomposition of the model matrix is performed in parallel using this cluster. Note that the speed ups are often not that great. On a multi-core machine it is usually best to set the cluster size to the number of physical cores, which is often less than what is reported by detectCores. Using more than the number of physical cores can result in no speed up at all (or even a slow down). Note that a highly parallel BLAS may negate all advantage from using a cluster of cores. Computing in parallel of course requires more memory than computing in series. See examples.

When discrete=TRUE the covariate data are first discretized. Discretization takes place on a smooth by smooth basis, or in the case of tensor product smooths (or any smooth that can be represented as such, such as random effects), separately for each marginal smooth. The required spline bases are then evaluated at the discrete values, and stored, along with index vectors indicating which original observation they relate to. Fitting is by a version of performance oriented iteration/PQL using REML smoothing parameter selection on each iterative working model (as for the default method). The iteration is based on the derivatives of the REML score, without computing the score itself, allowing the expensive computations to be reduced to one parallel block Cholesky decomposition per iteration (plus two basic operations of equal cost, but easily parallelized). Unlike standard POI/PQL, only one step of the smoothing parameter update for the working model is taken at each step (rather than iterating to the optimal set of smoothing parameters for each working model). At each step a weighted model matrix crossproduct of the model matrix is required - this is efficiently computed from the pre-computed basis functions evaluated at the discretized covariate values. Efficient computation with tensor product terms means that some terms within a tensor product may be re-ordered for maximum efficiency. Parallel computation is controlled using the nthreads argument. For this method no cluster computation is used, and the parallel package is not required.

If the deprecated argument sparse=TRUE then QR updating is replaced by an alternative scheme, in which the model matrix is stored whole as a sparse matrix. This only makes sense if all smooths are P-splines and all tensor products are of the form te(...,bs="ps",np=FALSE), but no check is made. The computations are then based on the Choleski decomposition of the crossproduct of the sparse model matrix. Although this crossproduct is nearly dense, sparsity should make its formation efficient, which is useful as it is the leading order term in the operations count. However there is no benefit in using sparse methods to form the Choleski decomposition, given that the crossproduct is dense. In practice the sparse matrix handling overheads mean that modest or no speed ups are produced by this approach, while the computation is less stable than the default, and the memory footprint often higher (but please let the author know if you find an example where the speedup is really worthwhile).

#### Value

An object of class "gam" as described in gamObject.

#### **WARNINGS**

The routine will be slow if the default "tp" basis is used.

You must have more unique combinations of covariates than the model has total parameters. (Total parameters is sum of basis dimensions plus sum of non-spline terms less the number of spline terms).

This routine is less stable than 'gam' for the same dataset.

The negbin family is only supported for the \*known theta\* case.

AIC computation does not currently take account of an AR1 model, if used.

# Author(s)

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

#### References

Wood, S.N., Goude, Y. & Shaw S. (2015) Generalized additive models for large datasets. Journal of the Royal Statistical Society, Series C 64(1): 139-155.

#### See Also

```
mgcv.parallel, mgcv-package, gamObject, gam.models, smooth.terms, linear.functional.terms,
s, te predict.gam, plot.gam, summary.gam, gam.side, gam.selection, gam.control gam.check,
linear.functional.terms negbin, magic, vis.gam
```

# **Examples**

```
library(mgcv)
## See help("mgcv-parallel") for using bam in parallel
## Some examples are marked 'Not run' purely to keep
## checking load on CRAN down. Sample sizes are small for
## the same reason.
set.seed(3)
dat <- gamSim(1,n=25000,dist="normal",scale=20)</pre>
bs <- "cr";k <- 12
b <- bam(y \sim s(x0,bs=bs)+s(x1,bs=bs)+s(x2,bs=bs,k=k)+
           s(x3,bs=bs),data=dat)
summary(b)
plot(b,pages=1,rug=FALSE) ## plot smooths, but not rug
plot(b,pages=1,rug=FALSE,seWithMean=TRUE) ## `with intercept' CIs
ba <- bam(y \sim s(x0,bs=bs,k=k)+s(x1,bs=bs,k=k)+s(x2,bs=bs,k=k)+
            s(x3,bs=bs,k=k),data=dat,method="GCV.Cp") ## use GCV
summary(ba)
## End(Not run)
## A Poisson example...
```

12 bam.update

bam.update

Update a strictly additive bam model for new data.

# **Description**

Gaussian with identity link models fitted by bam can be efficiently updated as new data becomes available, by simply updating the QR decomposition on which estimation is based, and re-optimizing the smoothing parameters, starting from the previous estimates. This routine implements this.

#### Usage

```
bam.update(b,data,chunk.size=10000)
```

# **Arguments**

data

b	A gam object fitted by bam and representing a strictly additive model (i.e. gaussian
	errors, identity link).

Extra data to augment the original data used to obtain b. Must include a weights

column if the original fit was weighted and a AR. start column if AR. start was

non NULL in original fit.

chunk.size size of subsets of data to process in one go when getting fitted values.

# **Details**

bam. update updates the QR decomposition of the (weighted) model matrix of the GAM represented by b to take account of the new data. The orthogonal factor multiplied by the response vector is also updated. Given these updates the model and smoothing parameters can be re-estimated, as if the whole dataset (original and the new data) had been fitted in one go. The function will use the bam.update 13

same AR1 model for the residuals as that employed in the original model fit (see rho parameter of bam).

Note that there may be small numerical differences in fit between fitting the data all at once, and fitting in stages by updating, if the smoothing bases used have any of their details set with reference to the data (e.g. default knot locations).

#### Value

An object of class "gam" as described in gamObject.

## Author(s)

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

#### References

```
http://www.maths.bris.ac.uk/~sw15190/
```

#### See Also

```
mgcv-package, bam
```

## **Examples**

```
library(mgcv)
## following is not *very* large, for obvious reasons...
set.seed(8)
n <- 5000
dat <- gamSim(1,n=n,dist="normal",scale=5)</pre>
dat[c(50,13,3000,3005,3100),]<- NA
dat1 <- dat[(n-999):n,]</pre>
dat0 <- dat[1:(n-1000),]</pre>
bs <- "ps";k <- 20
method <- "GCV.Cp"</pre>
b \leftarrow bam(y \sim s(x0,bs=bs,k=k)+s(x1,bs=bs,k=k)+s(x2,bs=bs,k=k)+s(x2,bs=bs,k=k)+s(x2,bs=bs,k=k)+s(x2,bs=bs,k=k)+s(x2,bs=bs,k=k)+s(x2,bs=bs,k=k)+s(x2,bs=bs,k=k)+s(x2,bs=bs,k=k)+s(x2,bs=bs,k=k)+s(x2,bs=bs,k=k)+s(x2,bs=bs,k=k)+s(x2,bs=bs,k=k)+s(x2,bs=bs,k=k)+s(x2,bs=bs,k=k)+s(x2,bs=bs,k=k)+s(x2,bs=bs,k=k)+s(x2,bs=bs,k=k)+s(x2,bs=bs,k=k)+s(x2,bs=bs,k=k)+s(x2,bs=bs,k=k)+s(x2,bs=bs,k=k)+s(x2,bs=bs,k=k)+s(x2,bs=bs,k=k)+s(x2,bs=bs,k=k)+s(x2,bs=bs,k=k)+s(x2,bs=bs,k=k)+s(x2,bs=bs,k=k)+s(x2,bs=bs,k=k)+s(x2,bs=bs,k=k)+s(x2,bs=bs,k=k)+s(x2,bs=bs,k=k)+s(x2,bs=bs,k=k)+s(x2,bs=bs,k=k)+s(x2,bs=bs,k=k)+s(x2,bs=bs,k=k)+s(x2,bs=bs,k=k)+s(x2,bs=bs,k=k)+s(x2,bs=bs,k=k)+s(x2,bs=bs,k=k)+s(x2,bs=bs,k=k)+s(x2,bs=bs,k=k)+s(x2,bs=bs,k=k)+s(x2,bs=bs,k=k)+s(x2,bs=bs,k=k)+s(x2,bs=bs,k=k)+s(x2,bs=bs,k=k)+s(x2,bs=bs,k=k)+s(x2,bs=bs,k=k)+s(x2,bs=bs,k=k)+s(x2,bs=bs,k=k)+s(x2,bs=bs,k=k)+s(x2,bs=bs,k=k)+s(x2,bs=bs,k=k)+s(x2,bs=bs,k=k)+s(x2,bs=bs,k=k)+s(x2,bs=bs,k=k)+s(x2,bs=bs,k=k)+s(x2,bs=bs,k=k)+s(x2,bs=bs,k=k)+s(x2,bs=bs,k=k)+s(x2,bs=bs,k=k)+s(x2,bs=bs,k=k)+s(x2,bs=bs,k=k)+s(x2,bs=bs,k=k)+s(x2,bs=bs,k=k)+s(x2,bs=bs,k=k)+s(x2,bs=bs,k=k)+s(x2,bs=bs,k=k)+s(x2,bs=bs,k=k)+s(x2,bs=bs,k=k)+s(x2,bs=bs,k=k)+s(x2,bs=bs,k=k)+s(x2,bs=bs,k=k)+s(x2,bs=bs,k=k)+s(x2,bs=bs,k=k)+s(x2,bs=bs,k=k)+s(x2,bs=bs,k=k)+s(x2,bs=bs,k=k)+s(x2,bs=bs,k=k)+s(x2,bs=bs,k=k)+s(x2,bs=bs,k=k)+s(x2,bs=bs,k=k)+s(x2,bs=bs,k=k)+s(x2,bs=bs,k=k)+s(x2,bs=bs,k=k)+s(x2,bs=bs,k=k)+s(x2,bs=bs,k=k)+s(x2,bs=bs,k=k)+s(x2,bs=bs,k=k)+s(x2,bs=bs,k=k)+s(x2,bs=bs,k=k)+s(x2,bs=bs,k=k)+s(x2,bs=bs,k=k)+s(x2,bs=bs,k=k)+s(x2,bs=bs,k=k)+s(x2,bs=bs,k=k)+s(x2,bs=bs,k=k)+s(x2,bs=bs,k=k)+s(x2,bs=bs,k=k)+s(x2,bs=bs,k=k)+s(x2,bs=bs,k=k)+s(x2,bs=bs,k=k)+s(x2,bs=bs,k=k)+s(x2,bs=bs,k=k)+s(x2,bs=bs,k=k)+s(x2,bs=bs,k=k)+s(x2,bs=bs,k=k)+s(x2,bs=bs,k=k)+s(x2,bs=bs,k=k)+s(x2,bs=bs,k=k)+s(x2,bs=bs,k=k)+s(x2,bs=bs,k=k)+s(x2,bs=bs,k=k)+s(x2,bs=bs,k=k)+s(x2,bs=bs,k=k)+s(x2,bs=bs,k=k)+s(x2,bs=bs,k=k)+s(x2,bs=bs,k=k)+s(x2,bs=bs,k=k)+s(x2,bs=bs,k=k)+s(x2,bs=bs,k=k)+s(x2,bs=bs,k=k)+s(x2,bs=bs,k=k)+s(x2,bs=bs,k=k)+s(x2,bs=bs,k=k)+
                                           s(x3,bs=bs,k=k),data=dat0,method=method)
b1 <- bam.update(b,dat1)</pre>
b2 <- bam.update(bam.update(b,dat1[1:500,]),dat1[501:1000,])</pre>
b3 \leftarrow bam(y \sim s(x0,bs=bs,k=k)+s(x1,bs=bs,k=k)+s(x2,bs=bs,k=k)+
                                           s(x3,bs=bs,k=k),data=dat,method=method)
b1;b2;b3
 ## example with AR1 errors...
e <- rnorm(n)
for (i in 2:n) e[i] \leftarrow e[i-1]*.7 + e[i]
dat$y <- dat$f + e*3
dat[c(50,13,3000,3005,3100),]<- NA
dat1 <- dat[(n-999):n,]</pre>
```

14 bandchol

bandchol

Choleski decomposition of a band diagonal matrix

# **Description**

Computes Choleski decomposition of a (symmetric positive definite) band-diagonal matrix, A.

# Usage

bandchol(B)

# **Arguments**

В

An n by k matrix containing the diagonals of the matrix A to be decomposed. First row is leading diagonal, next is first sub-diagonal, etc. sub-diagonals are zero padded at the end. Alternatively gives A directly, i.e. a square matrix with 2k-1 non zero diagonals (those from the lower triangle are not accessed).

# **Details**

Calls dpbtrf from LAPACK. The point of this is that it has  $O(k^2n)$  computational cost, rather than the  $O(n^3)$  required by dense matrix methods.

#### Value

Let R be the factor such that t(R)%\*R = A. R is upper triangular and if the rows of B contained the diagonals of A on entry, then what is returned is an n by k matrix containing the diagonals of R, packed as B was packed on entry. If B was square on entry, then R is returned directly. See examples.

# Author(s)

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

#### References

Anderson, E., Bai, Z., Bischof, C., Blackford, S., Dongarra, J., Du Croz, J., Greenbaum, A., Hammarling, S., McKenney, A. and Sorensen, D., 1999. LAPACK Users' guide (Vol. 9). Siam.

betar 15

## **Examples**

```
require(mgcv)
## simulate a banded diagonal matrix
n <- 7;set.seed(8)
A <- matrix(0,n,n)
sdiag(A) <- runif(n);sdiag(A,1) <- runif(n-1)
sdiag(A,2) <- runif(n-2)
A <- crossprod(A)

## full matrix form...
bandchol(A)
chol(A) ## for comparison

## compact storage form...
B <- matrix(0,3,n)
B[1,] <- sdiag(A);B[2,1:(n-1)] <- sdiag(A,1)
B[3,1:(n-2)] <- sdiag(A,2)
bandchol(B)</pre>
```

betar

GAM beta regression family

## **Description**

Family for use with gam, implementing regression for beta distributed data on (0,1). A linear predictor controls the mean,  $\mu$  of the beta distribution, while the variance is then  $\mu(1-\mu)/(1+\phi)$ , with parameter  $\phi$  being estimated during fitting, alongside the smoothing parameters.

## Usage

```
betar(theta = NULL, link = "logit",eps=.Machine$double.eps*100)
```

## **Arguments**

theta the extra parameter ( $\phi$  above).

link The link function: one of "logit", "probit", "cloglog" and "cauchit".

eps the response variable will be truncated to the interval [eps, 1-eps] if there are

values outside this range. This truncation is not entirely benign, but too small a value of eps will cause stability problems if there are zeroes or ones in the

response.

#### **Details**

These models are useful for proportions data which can not be modelled as binomial. Note the assumption that data are in (0,1), despite the fact that for some parameter values 0 and 1 are perfectly legitimate observations. The restriction is needed to keep the log likelihood bounded for all parameter values. Any data exactly at 0 or 1 are reset to be just above 0 or just below 1 using the

16 bug.reports.mgcv

eps argument (in fact any observation <eps is reset to eps and any observation >1-eps is reset to 1-eps). Note the effect of this resetting. If  $\mu\phi>1$  then impossible 0s are replaced with highly improbable eps values. If the inequality is reversed then 0s with infinite probability density are replaced with eps values having high finite probability density. The equivalent condition for 1s is  $(1-\mu)\phi>1$ . Clearly all types of resetting are somewhat unsatisfactory, and care is needed if data contain 0s or 1s (often it makes sense to manually reset the 0s and 1s in a manner that somehow reflects the sampling setup).

#### Value

An object of class extended. family.

#### **WARNINGS**

Do read the details section if your data contain 0s and or 1s.

# Author(s)

Natalya Pya (nyp20@bath.ac.uk) and Simon Wood (s.wood@r-project.org)

# **Examples**

```
library(mgcv)
## Simulate some beta data...
set.seed(3);n<-400
dat <- gamSim(1,n=n)
mu <- binomial()$linkinv(dat$f/4-2)
phi <- .5
a <- mu*phi;b <- phi - a;
dat$y <- rbeta(n,a,b)

bm <- gam(y~s(x0)+s(x1)+s(x2)+s(x3),family=betar(link="logit"),data=dat)
bm
plot(bm,pages=1)</pre>
```

bug.reports.mgcv

Reporting mgcv bugs.

#### Description

mgcv works largely because many people have reported bugs over the years. If you find something that looks like a bug, please report it, so that the package can be improved. mgcv does not have a large development budget, so it is a big help if bug reports follow the following guidlines.

The ideal report consists of an email to <simon.wood@r-project.org> with a subject line including mgcv somewhere, containing

1. The results of running sessionInfo in the R session where the problem occurs. This provides platform details, R and package version numbers, etc.

choose.k 17

- 2. A brief description of the problem.
- 3. Short cut and paste-able code that produces the problem, including the code for loading/generating the data (using standard R functions like load, read. table etc).
- 4. Any required data files. If you send real data it will only be used for the purposes of debugging.

Of course if you have dug deeper and have an idea of what is causing the problem, that is also helpful to know, as is any suggested code fix. (Don't send a fixed package .tar.gz file, however - I can't use this).

#### Author(s)

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

choose.k

Basis dimension choice for smooths

#### **Description**

Choosing the basis dimension, and checking the choice, when using penalized regression smoothers.

Penalized regression smoothers gain computational efficiency by virtue of being defined using a basis of relatively modest size, k. When setting up models in the mgcv package, using s or te terms in a model formula, k must be chosen: the defaults are essentially arbitrary.

In practice k-1 (or k) sets the upper limit on the degrees of freedom associated with an s smooth (1 degree of freedom is usually lost to the identifiability constraint on the smooth). For te smooths the upper limit of the degrees of freedom is given by the product of the k values provided for each marginal smooth less one, for the constraint. However the actual effective degrees of freedom are controlled by the degree of penalization selected during fitting, by GCV, AIC, REML or whatever is specified. The exception to this is if a smooth is specified using the fx=TRUE option, in which case it is unpenalized.

So, exact choice of k is not generally critical: it should be chosen to be large enough that you are reasonably sure of having enough degrees of freedom to represent the underlying 'truth' reasonably well, but small enough to maintain reasonable computational efficiency. Clearly 'large' and 'small' are dependent on the particular problem being addressed.

As with all model assumptions, it is useful to be able to check the choice of k informally. If the effective degrees of freedom for a model term are estimated to be much less than k-1 then this is unlikely to be very worthwhile, but as the EDF approach k-1, checking can be important. A useful general purpose approach goes as follows: (i) fit your model and extract the deviance residuals; (ii) for each smooth term in your model, fit an equivalent, single, smooth to the residuals, using a substantially increased k to see if there is pattern in the residuals that could potentially be explained by increasing k. Examples are provided below.

The obvious, but more costly, alternative is simply to increase the suspect k and refit the original model. If there are no statistically important changes as a result of doing this, then k was large enough. (Change in the smoothness selection criterion, and/or the effective degrees of freedom,

18 choose.k

when k is increased, provide the obvious numerical measures for whether the fit has changed substantially.)

gam. check runs a simple simulation based check on the basis dimensions, which can help to flag up terms for which k is too low. Grossly too small k will also be visible from partial residuals available with plot.gam.

One scenario that can cause confusion is this: a model is fitted with k=10 for a smooth term, and the EDF for the term is estimated as 7.6, some way below the maximum of 9. The model is then refitted with k=20 and the EDF increases to 8.7 - what is happening - how come the EDF was not 8.7 the first time around? The explanation is that the function space with k=20 contains a larger subspace of functions with EDF 8.7 than did the function space with k=10: one of the functions in this larger subspace fits the data a little better than did any function in the smaller subspace. These subtleties seldom have much impact on the statistical conclusions to be drawn from a model fit, however.

#### Author(s)

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

#### References

Wood, S.N. (2006) Generalized Additive Models: An Introduction with R. CRC.

```
http://www.maths.bris.ac.uk/~sw15190/
```

# **Examples**

```
## Simulate some data ....
library(mgcv)
set.seed(1)
dat <- gamSim(1,n=400,scale=2)</pre>
## fit a GAM with quite low `k'
b < -gam(y \sim s(x0, k=6) + s(x1, k=6) + s(x2, k=6) + s(x3, k=6), data=dat)
plot(b,pages=1,residuals=TRUE) ## hint of a problem in s(x2)
## the following suggests a problem with s(x2)
gam.check(b)
## Another approach (see below for more obvious method)....
## check for residual pattern, removeable by increasing `k'
## typically `k', below, chould be substantially larger than
## the original, `k' but certainly less than n/2.
## Note use of cheap "cs" shrinkage smoothers, and gamma=1.4
## to reduce chance of overfitting...
rsd <- residuals(b)</pre>
gam(rsd^s(x0,k=40,bs="cs"),gamma=1.4,data=dat) ## fine
gam(rsd^s(x1,k=40,bs="cs"),gamma=1.4,data=dat) ## fine
gam(rsd~s(x2,k=40,bs="cs"),gamma=1.4,data=dat) ## `k' too low
gam(rsd~s(x3,k=40,bs="cs"),gamma=1.4,data=dat) ## fine
## refit...
b \leftarrow gam(y^s(x0,k=6)+s(x1,k=6)+s(x2,k=20)+s(x3,k=6),data=dat)
```

choose.k

```
gam.check(b) ## better
## similar example with multi-dimensional smooth
b1 <- gam(y \sim s(x0) + s(x1, x2, k=15) + s(x3), data=dat)
rsd <- residuals(b1)
gam(rsd^s(x0, k=40, bs="cs"), gamma=1.4, data=dat) ## fine
gam(rsd~s(x1,x2,k=100,bs="ts"),gamma=1.4,data=dat) ## `k' too low
gam(rsd~s(x3,k=40,bs="cs"),gamma=1.4,data=dat) ## fine
gam.check(b1) ## shows same problem
## and a `te' example
b2 <- gam(y \sim s(x0) + te(x1, x2, k=4) + s(x3), data=dat)
rsd <- residuals(b2)</pre>
gam(rsd^s(x0,k=40,bs="cs"),gamma=1.4,data=dat) ## fine
gam(rsd^te(x1,x2,k=10,bs="cs"),gamma=1.4,data=dat) ## 'k' too low
gam(rsd\sim s(x3,k=40,bs="cs"),gamma=1.4,data=dat) ## fine
gam.check(b2) ## shows same problem
## same approach works with other families in the original model
dat <- gamSim(1,n=400,scale=.25,dist="poisson")</pre>
bp < -gam(y \sim s(x0, k=5) + s(x1, k=5) + s(x2, k=5) + s(x3, k=5),
        family=poisson,data=dat,method="ML")
gam.check(bp)
rsd <- residuals(bp)</pre>
gam(rsd^s(x0,k=40,bs="cs"),gamma=1.4,data=dat) ## fine
gam(rsd\sim s(x1,k=40,bs="cs"),gamma=1.4,data=dat) ## fine
gam(rsd^s(x2,k=40,bs="cs"),gamma=1.4,data=dat) ## k' too low
gam(rsd^s(x3,k=40,bs="cs"),gamma=1.4,data=dat) ## fine
rm(dat)
## More obvious, but more expensive tactic... Just increase
## suspicious k until fit is stable.
set.seed(0)
dat <- gamSim(1,n=400,scale=2)</pre>
## fit a GAM with quite low `k'
b \leftarrow gam(y^s(x0,k=6)+s(x1,k=6)+s(x2,k=6)+s(x3,k=6),
         data=dat,method="REML")
h
## edf for 3rd smooth is highest as proportion of k -- increase k
b \leftarrow gam(y \sim s(x0, k=6) + s(x1, k=6) + s(x2, k=12) + s(x3, k=6),
         data=dat,method="REML")
## edf substantially up, -ve REML substantially down
b \leftarrow gam(y \sim s(x0, k=6) + s(x1, k=6) + s(x2, k=24) + s(x3, k=6),
         data=dat,method="REML")
## slight edf increase and -ve REML change
```

20 columb

columb

Reduced version of Columbus OH crime data

# Description

By district crime data from Columbus OH, together with polygons describing district shape. Useful for illustrating use of simple Markov Random Field smoothers.

# Usage

```
data(columb)
data(columb.polys)
```

#### **Format**

columb is a 49 row data frame with the following columns

area land area of district

home.value housing value in 1000USD.

income household income in 1000USD.

crime residential burglaries and auto thefts per 1000 households.

open.space measure of open space in district.

district code identifying district, and matching names (columb.polys).

columb. polys contains the polygons defining the areas in the format described below.

# Details

The data frame columb relates to the districts whose boundaries are coded in columb.polys. columb.polys[[i]] is a 2 column matrix, containing the vertices of the polygons defining the boundary of the ith district. columb.polys[[2]] has an artificial hole inserted to illustrate how holes in districts can be spefified. Different polygons defining the boundary of a district are separated by NA rows in columb.polys[[1]], and a polygon enclosed within another is treated as a hole in that region (a hole should never come first). names(columb.polys) matches columb\$district (order unimportant).

#### Source

The data are adapted from the columbus example in the spdep package, where the original source is given as:

Anselin, Luc. 1988. Spatial econometrics: methods and models. Dordrecht: Kluwer Academic, Table 12.1 p. 189.

concurvity 21

## **Examples**

## see ?mrf help files

concurvity

GAM concurvity measures

## **Description**

Produces summary measures of concurvity between gam components.

#### **Usage**

concurvity(b, full=TRUE)

## **Arguments**

b An object inheriting from class "gam".

full If TRUE then concurvity of each term with the whole of the rest of the model is

considered. If FALSE then pairwise concurvity measures between each smooth

term (as well as the parametric component) are considered.

#### **Details**

Concurvity occurs when some smooth term in a model could be approximated by one or more of the other smooth terms in the model. This is often the case when a smooth of space is included in a model, along with smooths of other covariates that also vary more or less smoothly in space. Similarly it tends to be an issue in models including a smooth of time, along with smooths of other time varying covariates.

Concurvity can be viewed as a generalization of co-linearity, and causes similar problems of interpretation. It can also make estimates somewhat unstable (so that they become sensitive to apparently innocuous modelling details, for example).

This routine computes three related indices of concurvity, all bounded between 0 and 1, with 0 indicating no problem, and 1 indicating total lack of identifiability. The three indices are all based on the idea that a smooth term, f, in the model can be decomposed into a part, g, that lies entirely in the space of one or more other terms in the model, and a remainder part that is completely within the term's own space. If g makes up a large part of f then there is a concurvity problem. The indices used are all based on the square of ||g||/||f||, that is the ratio of the squared Euclidean norms of the vectors of f and g evaluated at the observed covariate values.

The three measures are as follows

worst This is the largest value that the square of ||g||/||f|| could take for any coefficient vector. This is a fairly pessimistic measure, as it looks at the worst case irrespective of data. This is the only measure that is symmetric.

**observed** This just returns the value of the square of ||g||/||f|| according to the estimated coefficients. This could be a bit over-optimistic about the potential for a problem in some cases.

cox.ph

**estimate** This is the squared F-norm of the basis for g divided by the F-norm of the basis for f. It is a measure of the extent to which the f basis can be explained by the g basis. It does not suffer from the pessimism or potential for over-optimism of the previous two measures, but is less easy to understand.

#### Value

If full=TRUE a matrix with one column for each term and one row for each of the 3 concurvity measures detailed below. If full=FALSE a list of 3 matrices, one for each of the three concurvity measures detailed below. Each row of the matrix relates to how the model terms depend on the model term supplying that rows name.

## Author(s)

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

#### References

```
http://www.maths.bris.ac.uk/~sw15190/
```

## **Examples**

```
library(mgcv)
## simulate data with concurvity...
set.seed(8);n<- 200
f2 \leftarrow function(x) 0.2 * x^11 * (10 * (1 - x))^6 + 10 *
            (10 * x)^3 * (1 - x)^10
t <- sort(runif(n)) ## first covariate
## make covariate x a smooth function of t + noise...
x \leftarrow f2(t) + rnorm(n)*3
## simulate response dependent on t and x...
y <- \sin(4*pi*t) + \exp(x/20) + rnorm(n)*.3
## fit model...
b \leftarrow gam(y \sim s(t,k=15) + s(x,k=15),method="REML")
## assess concurvity between each term and `rest of model'...
concurvity(b)
## ... and now look at pairwise concurvity between terms...
concurvity(b, full=FALSE)
```

cox.ph 23

# **Description**

The cox.ph family implements the Cox Proportional Hazards model with Peto's correction for ties, and estimation by penalized partial likelihood maximization, for use with gam. In the model formula, event time is the response. The weights vector provides the censoring information (0 for censoring, 1 for event).

# Usage

```
cox.ph(link="identity")
```

# **Arguments**

link

currently (and possibly for ever) only "identity" supported.

#### **Details**

Used with gam to fit Cox Proportional Hazards models to survival data. The model formula will have event/censoring times on the left hand side and the linear predictor specification on the right hand side. Censoring information is provided by the weights argument to gam, with 1 indicating an event and 0 indicating censoring.

Prediction from the fitted model object (using the predict method) with type="response" will predict on the survivor function scale. See example code below for extracting the baseline hazard/survival directly. Martingale or deviance residuals can be extracted. The fitted.values stored in the model object are survival function estimates for each subject at their event/censoring time.

Estimation of model coefficients is by maximising the log-partial likelihood penalized by the smoothing penalties. See e.g. Hastie and Tibshirani, 1990, section 8.3. for the partial likelihood used (with Peto's approximation for ties), but note that optimization of the partial likelihood does not follow Hastie and Tibshirani. See Klein amd Moeschberger (2003) for estimation of residuals, the baseline hazard, survival function and associated standard errors.

The percentage deviance explained reported for Cox PH models is based on the sum of squares of the deviance residuals, as the model deviance, and the sum of squares of the deviance residuals when the covariate effects are set to zero, as the null deviance. The same baseline hazard estimate is used for both.

#### Value

An object inheriting from class general.family.

#### References

Hastie and Tibshirani (1990) Generalized Additive Models, Chapman and Hall.

Klein, J.P and Moeschberger, M.L. (2003) Survival Analysis: Techniques for Censored and Truncated Data (2nd ed.) Springer.

Wood, S.N., N. Pya and B. Saefken (2015), Smoothing parameter and model selection for general smooth models. http://arxiv.org/abs/1511.03864

24 cox.ph

# **Examples**

```
library(mgcv)
library(survival) ## for data
col1 <- colon[colon$etype==1,] ## concentrate on single event</pre>
col1$differ <- as.factor(col1$differ)</pre>
col1$sex <- as.factor(col1$sex)</pre>
b <- gam(time~s(age,by=sex)+sex+s(nodes)+perfor+rx+obstruct+adhere,</pre>
          family=cox.ph(),data=col1,weights=status)
summary(b)
plot(b,pages=1,all.terms=TRUE) ## plot effects
plot(b$linear.predictors,residuals(b))
## plot survival function for patient j...
np < -300; j < -6
newd <- data.frame(time=seq(0,3000,length=np))</pre>
dname <- names(col1)</pre>
for (n in dname) newd[[n]] <- rep(col1[[n]][j],np)</pre>
newd$time <- seq(0,3000,length=np)</pre>
fv <- predict(b,newdata=newd,type="response",se=TRUE)</pre>
plot(newd\$time,fv\$fit,type="l",ylim=c(0,1),xlab="time",ylab="survival")
lines(newd$time,fv$fit+2*fv$se.fit,col=2)
lines(newd$time,fv$fit-2*fv$se.fit,col=2)
## crude plot of baseline survival...
plot(b$family$data$tr,exp(-b$family$data$h),type="1",ylim=c(0,1),
     xlab="time",ylab="survival")
lines(bfamily$data$tr,exp(-bfamily$data$h + 2*bfamily$data$q^.5),col=2)
lines(b$family$data$tr,exp(-b$family$data$h - 2*b$family$data$q^.5),col=2)
lines(b$family$data$tr,exp(-b$family$data$km),lty=2) ## Kaplan Meier
## Simple simulated known truth example...
ph.weibull.sim <- function(eta,gamma=1,h0=.01,t1=100) {</pre>
  lambda <- h0*exp(eta)</pre>
  n <- length(eta)</pre>
  U <- runif(n)
  t <- (-log(U)/lambda)^(1/gamma)
  d <- as.numeric(t <= t1)</pre>
  t[!d] <- t1
  list(t=t,d=d)
n \leftarrow 500; set.seed(2)
x0 \leftarrow runif(n, 0, 1); x1 \leftarrow runif(n, 0, 1)
x2 <- runif(n, 0, 1);x3 <- runif(n, 0, 1)
f0 \leftarrow function(x) 2 * sin(pi * x)
f1 \leftarrow function(x) \exp(2 * x)
f2 \leftarrow function(x) \ 0.2*x^11*(10*(1-x))^6+10*(10*x)^3*(1-x)^10
```

cSplineDes 25

```
f3 <- function(x) 0*x
f <- f0(x0) + f1(x1) + f2(x2)
g <- (f-mean(f))/5
surv <- ph.weibull.sim(g)
surv$x0 <- x0;surv$x1 <- x1;surv$x2 <- x2;surv$x3 <- x3
b <- gam(t~s(x0)+s(x1)+s(x2,k=15)+s(x3),family=cox.ph,weights=d,data=surv)
plot(b,pages=1)</pre>
```

cSplineDes

Evaluate cyclic B spline basis

# **Description**

Uses splineDesign to set up the model matrix for a cyclic B-spline basis.

## Usage

```
cSplineDes(x, knots, ord = 4)
```

## **Arguments**

x covariate values for smooth.

knots The knot locations: the range of these must include all the data.

ord order of the basis. 4 is a cubic spline basis. Must be >1.

#### **Details**

The routine is a wrapper that sets up a B-spline basis, where the basis functions wrap at the first and last knot locations.

# Value

A matrix with length(x) rows and length(knots)-1 columns.

# Author(s)

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

# See Also

```
cyclic.p.spline
```

26 exclude.too.far

## **Examples**

```
require(mgcv)
## create some x's and knots...
n <- 200
x <- 0:(n-1)/(n-1);k<- 0:5/5
X \leftarrow cSplineDes(x,k) ## cyclic spline design matrix
## plot evaluated basis functions...
plot(x,X[,1],type="l"); for (i in 2:5) lines(x,X[,i],col=i)
## check that the ends match up....
ee <- X[1,]-X[n,];ee
tol <- .Machine$double.eps^.75
if (all.equal(ee,ee*0,tolerance=tol)!=TRUE)
  stop("cyclic spline ends don't match!")
## similar with uneven data spacing...
x \leftarrow sort(runif(n)) + 1 ## sorting just makes end checking easy
k \le seq(min(x), max(x), length=8) ## create knots
X <- cSplineDes(x,k) ## get cyclic spline model matrix
plot(x,X[,1],type="l"); for (i in 2:ncol(X)) lines(x,X[,i],col=i)
ee <- X[1,]-X[n,];ee ## do ends match??
tol <- .Machine$double.eps^.75
if (all.equal(ee,ee*0,tolerance=tol)!=TRUE)
  stop("cyclic spline ends don't match!")
```

exclude.too.far

Exclude prediction grid points too far from data

## **Description**

Takes two arrays defining the nodes of a grid over a 2D covariate space and two arrays defining the location of data in that space, and returns a logical vector with elements TRUE if the corresponding node is too far from data and FALSE otherwise. Basically a service routine for vis.gam and plot.gam.

# Usage

```
exclude.too.far(g1,g2,d1,d2,dist)
```

## **Arguments**

g1		co-ordinates of grid relative to first axis.
g2		co-ordinates of grid relative to second axis.
d1		co-ordinates of data relative to first axis.
d2		co-ordinates of data relative to second axis.
di	st	how far away counts as too far. Grid and data are first scaled so that the grid lies exactly in the unit square, and dist is a distance within this unit square.

extract.lme.cov 27

#### **Details**

Linear scalings of the axes are first determined so that the grid defined by the nodes in g1 and g2 lies exactly in the unit square (i.e. on [0,1] by [0,1]). These scalings are applied to g1, g2, d1 and d2. The minimum Euclidean distance from each node to a datum is then determined and if it is greater than dist the corresponding entry in the returned array is set to TRUE (otherwise to FALSE). The distance calculations are performed in compiled code for speed without storage overheads.

#### Value

A logical array with TRUE indicating a node in the grid defined by g1, g2 that is 'too far' from any datum.

#### Author(s)

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

#### References

```
http://www.maths.bris.ac.uk/~sw15190/
```

#### See Also

```
vis.gam
```

# **Examples**

```
library(mgcv)
x<-rnorm(100);y<-rnorm(100) # some "data"
n<-40 # generate a grid....
mx<-seq(min(x),max(x),length=n)
my<-seq(min(y),max(y),length=n)
gx<-rep(mx,n);gy<-rep(my,rep(n,n))
tf<-exclude.too.far(gx,gy,x,y,0.1)
plot(gx[!tf],gy[!tf],pch=".");points(x,y,col=2)</pre>
```

extract.lme.cov

Extract the data covariance matrix from an lme object

## **Description**

This is a service routine for gamm. Extracts the estimated covariance matrix of the data from an 1me object, allowing the user control about which levels of random effects to include in this calculation. extract.1me.cov forms the full matrix explicitly: extract.1me.cov2 tries to be more economical than this.

# Usage

```
extract.lme.cov(b,data,start.level=1)
extract.lme.cov2(b,data,start.level=1)
```

28 extract.lme.cov

#### **Arguments**

b A fitted model object returned by a call to lme.

data The data frame/ model frame that was supplied to lme.

start.level The level of nesting at which to start including random effects in the calculation.

This is used to allow smooth terms to be estimated as random effects, but treated

like fixed effects for variance calculations.

#### **Details**

The random effects, correlation structure and variance structure used for a linear mixed model combine to imply a covariance matrix for the response data being modelled. These routines extracts that covariance matrix. The process is slightly complicated, because different components of the fitted model object are stored in different orders (see function code for details!).

The extract.lme.cov calculation is not optimally efficient, since it forms the full matrix, which may in fact be sparse. extract.lme.cov2 is more efficient. If the covariance matrix is diagonal, then only the leading diagonal is returned; if it can be written as a block diagonal matrix (under some permutation of the original data) then a list of matrices defining the non-zero blocks is returned along with an index indicating which row of the original data each row/column of the block diagonal matrix relates to. The block sizes are defined by the coarsest level of grouping in the random effect structure.

```
gamm uses extract.lme.cov2.
```

extract.lme.cov does not currently deal with the situation in which the grouping factors for a correlation structure are finer than those for the random effects. extract.lme.cov2 does deal with this situation.

#### Value

For extract. lme.cov an estimated covariance matrix.

For extract. 1me. cov2 a list containing the estimated covariance matrix and an indexing array. The covariance matrix is stored as the elements on the leading diagonal, a list of the matrices defining a block diagonal matrix, or a full matrix if the previous two options are not possible.

## Author(s)

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

# References

For 1me see:

Pinheiro J.C. and Bates, D.M. (2000) Mixed effects Models in S and S-PLUS. Springer

For details of how GAMMs are set up here for estimation using 1me see:

Wood, S.N. (2006) Low rank scale invariant tensor product smooths for Generalized Additive Mixed Models. Biometrics 62(4):1025-1036

or

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

family.mgcv 29

```
http://www.maths.bris.ac.uk/~sw15190/
```

#### See Also

```
gamm, formXtViX
```

# **Examples**

```
## see also ?formXtViX for use of extract.lme.cov2
require(mgcv)
library(nlme)
data(Rail)
b <- lme(travel~1,Rail,~1|Rail)
extract.lme.cov(b,Rail)</pre>
```

family.mgcv

Distribution families in mgcv

# Description

As well as the standard families documented in family (see also glm) which can be used with functions gam, bam and gamm, mgcv also supplies some extra families, most of which are currently only usable with gam. These are described here.

# **Details**

The following families are in the exponential family given the value of a single parameter. They are usable with all modelling functions.

- Tweedie An exponential family distribution for which the variance of the response is given by the mean response to the power p. p is in (1,2) and must be supplied. See tw to estimate p.
- negbin The negative binomial. See nb to estimate the theta parameter of the negative binomial.

The following families are for regression type models dependent on a single linear predictor, and with a log likelihood which is a sum of independent terms, each coprresponding to a single response observation. Usable only with gam, with smoothing parameter estimation by "REML" or "ML" (the latter does not integrate the unpenalized and parameteric effects out of the marginal likelihood optimized for the smoothing parameters).

- ocat for ordered categorical data.
- tw for Tweedie distributed data, when the power parameter relating the variance to the mean is to be estimated.
- nb for negative binomial data when the theta parameter is to be estimated.
- betar for proportions data on (0,1) when the binomial is not appropriate.
- scat scaled t for heavy tailed data that would otherwise be modelled as Gaussian.

30 fix.family.link

• ziP for zero inflated Poisson data, when the zero inflation rate depends simply on the Poisson mean.

The following families implement more general model classes. Usable only with gam and only with REML smoothing parameter estimation.

- cox.ph the Cox Proportional Hazards model for survival data.
- gaulss a Gaussian location-scale model where the mean and the standard deviation are both modelled using smooth linear predictors.
- ziplss a 'two-stage' zero inflated Poisson model, in which 'potential-presence' is modelled with one linear predictor, and Poisson mean abundance given potential presence is modelled with a second linear predictor.
- mvn: multivariate normal additive models.
- multinom: multinomial logistic regression, for unordered categorical responses.

## Author(s)

Simon N. Wood (s.wood@r-project.org) & Natalya Pya

#### References

Wood, S.N., N. Pya and B. Saefken (2015), Smoothing parameter and model selection for general smooth models. http://arxiv.org/abs/1511.03864

fix.family.link

Modify families for use in GAM fitting and checking

## Description

Generalized Additive Model fitting by 'outer' iteration, requires extra derivatives of the variance and link functions to be added to family objects. The first 3 functions add what is needed. Model checking can be aided by adding quantile and random deviate generating functions to the family. The final two functions do this.

# Usage

```
fix.family.link(fam)
fix.family.var(fam)
fix.family.ls(fam)
fix.family.qf(fam)
fix.family.rd(fam)
```

## **Arguments**

fam

A family.

fixDependence 31

#### **Details**

Consider the first 3 function first.

Outer iteration GAM estimation requires derivatives of the GCV, UBRE/gAIC, GACV, REML or ML score, which are obtained by finding the derivatives of the model coefficients w.r.t. the log smoothing parameters, using the implicit function theorem. The expressions for the derivatives require the second and third derivatives of the link w.r.t. the mean (and the 4th derivatives if Fisher scoring is not used). Also required are the first and second derivatives of the variance function w.r.t. the mean (plus the third derivative if Fisher scoring is not used). Finally REML or ML estimation of smoothing parameters requires the log saturated likelihood and its first two derivatives w.r.t. the scale parameter. These functions add functions evaluating these quantities to a family.

If the family already has functions dvar, d2var, d3var, d2link, d3link, d4link and for RE/ML ls, then these functions simply return the family unmodified: this allows non-standard links to be used with gam when using outer iteration (performance iteration operates with unmodified families). Note that if you only need Fisher scoring then d4link and d3var can be dummy, as they are ignored. similarly ls is only needed for RE/ML.

The dvar function is a function of a mean vector, mu, and returns a vector of corresponding first derivatives of the family variance function. The d2link function is also a function of a vector of mean values, mu: it returns a vector of second derivatives of the link, evaluated at mu. Higher derivatives are defined similarly.

If modifying your own family, note that you can often get away with supplying only a dvar and d2var, function if your family only requires links that occur in one of the standard families.

The second two functions are useful for investigating the distribution of residuals and are used by qq.gam. If possible the functions add quantile (qf) or random deviate (rd) generating functions to the family. If a family already has qf or rd functions then it is left unmodified. qf functions are only available for some families, and for quasi families neither type of function is available.

#### Value

A family object with extra component functions dvar, d2var, d2link, d3link, d4link, ls, and possibly qf and rd, depending on which functions are called.

#### Author(s)

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

# See Also

```
gam.fit3, qq.gam
```

fixDependence

Detect linear dependencies of one matrix on another

## Description

Identifies columns of a matrix X2 which are linearly dependent on columns of a matrix X1. Primarily of use in setting up identifiability constraints for nested GAMs.

32 fixDependence

## Usage

fixDependence(X1,X2,tol=.Machine\$double.eps^.5,rank.def=0,strict=FALSE)

# Arguments

X1	A matrix.
X2	A matrix, the columns of which may be partially linearly dependent on the columns of X1.
tol	The tolerance to use when assessing linear dependence.
rank.def	If the degree of rank deficiency in X2, given X1, is known, then it can be supplied here, and tol is then ignored. Unused unless positive and not greater than the number of columns in X2.
strict	if TRUE then only columns individually dependent on X1 are detected, if FALSE then enough columns to make the reduced X2 full rank and independent of X1 are detected.

## **Details**

The algorithm uses a simple approach based on QR decomposition: see Wood (2006, section 4.10.2) for details.

# Value

A vector of the columns of X2 which are linearly dependent on columns of X1 (or which need to be deleted to acheive independence and full rank if strict==FALSE). NULL if the two matrices are independent.

# Author(s)

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

# References

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

# **Examples**

```
library(mgcv)
n<-20;c1<-4;c2<-7
X1<-matrix(runif(n*c1),n,c1)
X2<-matrix(runif(n*c2),n,c2)
X2[,3]<-X1[,2]+X2[,4]*.1
X2[,5]<-X1[,1]*.2+X1[,2]*.04
fixDependence(X1,X2)
fixDependence(X1,X2,strict=TRUE)</pre>
```

formula.gam 33

formula.gam

GAM formula

## Description

Description of gam formula (see Details), and how to extract it from a fitted gam object.

## Usage

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

# Arguments

x fitted model objects of class gam (see gamObject) as produced by gam().

... un-used in this case

#### **Details**

gam will accept a formula or, with some families, a list of formulae. Other mgcv modelling functions will not accept a list. The list form provides a mechanism for specifying several linear predictors, and allows these to share terms: see below.

The formulae supplied to gam are exactly like those supplied to glm except that smooth terms, s, te, ti and t2 can be added to the right hand side (and . is not supported in gam formulae).

```
Smooth terms are specified by expressions of the form: s(x1, x2, ..., k=12, fx=FALSE, bs="tp", by=z, id=1)
```

where x1, x2, etc. are the covariates which the smooth is a function of, and k is the dimension of the basis used to represent the smooth term. If k is not specified then basis specific defaults are used. Note that these defaults are essentially arbitrary, and it is important to check that they are not so small that they cause oversmoothing (too large just slows down computation). Sometimes the modelling context suggests sensible values for k, but if not informal checking is easy: see choose.k and gam.check.

fx is used to indicate whether or not this term should be unpenalized, and therefore have a fixed number of degrees of freedom set by k (almost always k-1). bs indicates the basis to use for the smooth: the built in options are described in smooth. terms, and user defined smooths can be added (see user.defined.smooth). If bs is not supplied then the default "tp" (tprs) basis is used. by can be used to specify a variable by which the smooth should be multiplied. For example  $gam(y\sim s(x,by=z))$  would specify a model E(y)=f(x)z where  $f(\cdot)$  is a smooth function. The by option is particularly useful for models in which different functions of the same variable are required for each level of a factor and for 'varying coefficient models': see gam.models. id is used to give smooths identities: smooths with the same identity have the same basis, penalty and smoothing parameter (but different coefficients, so they are different functions).

```
An alternative for specifying smooths of more than one covariate is e.g.: te(x,z,bs=c("tp","tp"),m=c(2,3),k=c(5,10))
```

which would specify a tensor product smooth of the two covariates x and z constructed from

34 formula.gam

marginal t.p.r.s. bases of dimension 5 and 10 with marginal penalties of order 2 and 3. Any combination of basis types is possible, as is any number of covariates. te provides further information. ti terms are a variant designed to be used as interaction terms when the main effects (and any lower order interactions) are present. t2 produces tensor product smooths that are the natural low rank analogue of smoothing spline anova models.

s, te, ti and t2 terms accept an sp argument of supplied smoothing parameters: positive values are taken as fixed values to be used, negative to indicate that the parameter should be estimated. If sp is supplied then it over-rides whatever is in the sp argument to gam, if it is not supplied then it defaults to all negative, but does not over-ride the sp argument to gam.

```
Formulae can involve nested or "overlapping" terms such as y \sim s(x) + s(z) + s(x, z) or y \sim s(x, z) + s(z, v)
```

but nested models should really be set up using ti terms: see gam.side for further details and examples.

Smooth terms in a gam formula will accept matrix arguments as covariates (and corresponding by variable), in which case a 'summation convention' is invoked. Consider the example of s(X,Z,by=L) where X, Z and L are n by m matrices. Let F be the n by m matrix that results from evaluating the smooth at the values in X and Z. Then the contribution to the linear predictor from the term will be rowSums(F\*L) (note the element-wise multiplication). This convention allows the linear predictor of the GAM to depend on (a discrete approximation to) any linear functional of a smooth: see linear.functional.terms for more information and examples (including functional linear models/signal regression).

Note that gam allows any term in the model formula to be penalized (possibly by multiple penalties), via the paraPen argument. See gam.models for details and example code.

When several formulae are provided in a list, then they can be used to specify multiple linear predictors for families for which this makes sense (e.g. mvn). The first formula in the list must include a response variable, but later formulae need not (depending on the requirements of the family). Let the linear predictors be indexed, 1 to d where d is the number of linear predictors, and the indexing is in the order in which the formulae appear in the list. It is possible to supply extra formulae specifying that several linear predictors should share some terms. To do this a formula is supplied in which the response is replaced by numbers specifying the indices of the linear predictors which will shre the terms specified on the r.h.s. For example  $1+3\sim s(x)+z-1$  specifies that linear predictors 1 and 3 will share the terms s(x) and z (but we don't want an extra intercept, as this would usually be unidentifiable). Note that it is possible that a linear predictor only includes shared terms: it must still have its own formula, but the r.h.s. would simply be -1 (e.g.  $y\sim -1$  or  $\sim -1$ ).

#### Value

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

## Author(s)

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

#### See Also

gam

formXtViX 35

formXtViX

Form component of GAMM covariance matrix

# **Description**

This is a service routine for gamm. Given, V, an estimated covariance matrix obtained using extract.lme.cov2 this routine forms a matrix square root of  $X^TV^{-1}X$  as efficiently as possible, given the structure of V (usually sparse).

## Usage

```
formXtViX(V,X)
```

# Arguments

V A data covariance matrix list returned from extract.lme.cov2

X A model matrix.

#### **Details**

The covariance matrix returned by extract.lme.cov2 may be in a packed and re-ordered format, since it is usually sparse. Hence a special service routine is required to form the required products involving this matrix.

## Value

A matrix, R such that crossprod(R) gives  $X^TV^{-1}X$ .

## Author(s)

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

## References

For 1me see:

Pinheiro J.C. and Bates, D.M. (2000) Mixed effects Models in S and S-PLUS. Springer

For details of how GAMMs are set up for estimation using 1me see:

Wood, S.N. (2006) Low rank scale invariant tensor product smooths for Generalized Additive Mixed Models. Biometrics 62(4):1025-1036

```
http://www.maths.bris.ac.uk/~sw15190/
```

#### See Also

```
gamm, extract.lme.cov2
```

36 fs.test

## **Examples**

```
require(mgcv)
library(nlme)
data(ergoStool)
b <- lme(effort ~ Type, data=ergoStool, random=~1|Subject)
V1 <- extract.lme.cov(b, ergoStool)
V2 <- extract.lme.cov2(b, ergoStool)
X <- model.matrix(b, data=ergoStool)
crossprod(formXtViX(V2, X))
t(X)</pre>
```

fs.test

FELSPLINE test function

## **Description**

Implements a finite area test function based on one proposed by Tim Ramsay (2002).

# Usage

```
fs.test(x,y,r0=.1,r=.5,l=3,b=1,exclude=TRUE)
fs.boundary(r0=.1,r=.5,l=3,n.theta=20)
```

# Arguments

x,y	Points at which to evaluate the test function.
r0	The test domain is a sort of bent sausage. This is the radius of the inner bend
r	The radius of the curve at the centre of the sausage.
1	The length of an arm of the sausage.
b	The rate at which the function increases per unit increase in distance along the centre line of the sausage.
exclude	Should exterior points be set to NA?
n.theta	How many points to use in a piecewise linear representation of a quarter of a circle, when generating the boundary curve.

## **Details**

The function details are not given in the source article: but this is pretty close. The function is modified from Ramsay (2002), in that it bulges, rather than being flat: this makes a better test of the smoother.

#### Value

fs. test returns function evaluations, or NAs for points outside the boundary. fs. boundary returns a list of x, y points to be jointed up in order to define/draw the boundary.

full.score 37

## Author(s)

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

## References

Tim Ramsay (2002) "Spline smoothing over difficult regions" J.R.Statist. Soc. B 64(2):307-319

# **Examples**

```
require(mgcv)
## plot the function, and its boundary...
fsb <- fs.boundary()
m<-300;n<-150
xm <- seq(-1,4,length=m);yn<-seq(-1,1,length=n)
xx <- rep(xm,n);yy<-rep(yn,rep(m,n))
tru <- matrix(fs.test(xx,yy),m,n) ## truth
image(xm,yn,tru,col=heat.colors(100),xlab="x",ylab="y")
lines(fsb$x,fsb$y,lwd=3)
contour(xm,yn,tru,levels=seq(-5,5,by=.25),add=TRUE)</pre>
```

full.score

GCV/UBRE score for use within nlm

# **Description**

Evaluates GCV/UBRE score for a GAM, given smoothing parameters. The routine calls gam.fit to fit the model, and is usually called by nlm to optimize the smoothing parameters.

This is basically a service routine for gam, and is not usually called directly by users. It is only used in this context for GAMs fitted by outer iteration (see gam.outer) when the outer method is "nlm.fd" (see gam argument optimizer).

## Usage

```
full.score(sp,G,family,control,gamma,...)
```

# Arguments

sp	The logs of the smoothing parameters
G	a list returned by mgcv:::gam.setup
family	The family object for the GAM.
control	a list returned be gam.control
gamma	the degrees of freedom inflation factor (usually 1).
	other arguments, typically for passing on to gam.fit.

#### Value

The value of the GCV/UBRE score, with attribute "full.gam.object" which is the full object returned by gam.fit.

#### Author(s)

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

gam

Generalized additive models with integrated smoothness estimation

## **Description**

Fits a generalized additive model (GAM) to data, the term 'GAM' being taken to include any quadratically penalized GLM and a variety of other models estimated by a quadratically penalised likelihood type approach (see family.mgcv). The degree of smoothness of model terms is estimated as part of fitting. gam can also fit any GLM subject to multiple quadratic penalties (including estimation of degree of penalization). Confidence/credible intervals are readily available for any quantity predicted using a fitted model.

Smooth terms are represented using penalized regression splines (or similar smoothers) with smoothing parameters selected by GCV/UBRE/AIC/REML or by regression splines with fixed degrees of freedom (mixtures of the two are permitted). Multi-dimensional smooths are available using penalized thin plate regression splines (isotropic) or tensor product splines (when an isotropic smooth is inappropriate), and users can add smooths. Linear functionals of smooths can alos be included in models. For an overview of the smooths available see smooth.terms. For more on specifying models see gam.models, random.effects and linear.functional.terms. For more on model selection see gam.selection. Do read gam.check and choose.k.

See gam from package gam, for GAMs via the original Hastie and Tibshirani approach (see details for differences to this implementation).

For very large datasets see bam, for mixed GAM see gamm and random.effects.

## Usage

```
gam(formula,family=gaussian(),data=list(),weights=NULL,subset=NULL,
    na.action,offset=NULL,method="GCV.Cp",
    optimizer=c("outer","newton"),control=list(),scale=0,
    select=FALSE,knots=NULL,sp=NULL,min.sp=NULL,H=NULL,gamma=1,
    fit=TRUE,paraPen=NULL,G=NULL,in.out,drop.unused.levels=TRUE,...)
```

### **Arguments**

formula

A GAM formula, or a list of formulae (see formula.gam and also gam.models). These are exactly like the formula for a GLM except that smooth terms, s, te, ti and t2, can be added to the right hand side to specify that the linear predictor depends on smooth functions of predictors (or linear functionals of these).

family

This is a family object specifying the distribution and link to use in fitting etc (see glm and family). See family mgcv for a full list of what is available, which goes well beyond exponential family. Note that guasi families actually result

goes well beyond exponential family. Note that quasi families actually result in the use of extended quasi-likelihood if method is set to a RE/ML method

(McCullagh and Nelder, 1989, 9.6).

data A data frame or list containing the model response variable and covariates re-

 $quired \ by \ the \ formula. \ By \ default \ the \ variables \ are \ taken \ from \ environment \ (formula):$ 

typically the environment from which gam is called.

weights prior weights on the contribution of the data to the log likelihood. Note that a

weight of 2, for example, is equivalent to having made exactly the same observation twice. If you want to reweight the contributions of each datum without changing the overall magnitude of the log likelihood, then you should normalize

the weights (e.g. weights <- weights/mean(weights)).

subset an optional vector specifying a subset of observations to be used in the fitting

process.

na.action a function which indicates what should happen when the data contain 'NA's.

The default is set by the 'na.action' setting of 'options', and is 'na.fail' if that is

unset. The "factory-fresh" default is 'na.omit'.

offset Can be 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 used to conform to the behaviour of 1m and glm).

control A list of fit control parameters to replace defaults returned by gam.control.

Values not set assume default values.

method The smoothing parameter estimation method. "GCV.Cp" to use GCV for un-

known scale parameter and Mallows' Cp/UBRE/AIC for known scale. "GACV.Cp" is equivalent, but using GACV in place of GCV. "REML" for REML estimation, including of unknown scale, "P-REML" for REML estimation, but using a Pearson estimate of the scale. "ML" and "P-ML" are similar, but using maximum likelihood in place of REML. Beyond the exponential family "REML" is the de-

fault, and the only other option is "ML".

optimizer An array specifying the numerical optimization method to use to optimize the

smoothing parameter estimation criterion (given by method). "perf" for performance iteration. "outer" for the more stable direct approach. "outer" can use several alternative optimizers, specified in the second element of optimizer: "newton" (default), "bfgs", "optim", "nlm" and "nlm.fd" (the latter is based

entirely on finite differenced derivatives and is very slow).

scale If this is positive then it is taken as the known scale parameter. Negative signals

that the scale parameter is unknown. 0 signals that the scale parameter is 1 for Poisson and binomial and unknown otherwise. Note that (RE)ML methods can

only work with scale parameter 1 for the Poisson and binomial cases.

select If this is TRUE then gam can add an extra penalty to each term so that it can be

penalized to zero. This means that the smoothing parameter estimation that is part of fitting can completely remove terms from the model. If the corresponding smoothing parameter is estimated as zero then the extra penalty has no effect.

knots this is an optional list containing user specified knot values to be used for basis construction. For most bases the user simply supplies the knots to be used,

which must match up with the k value supplied (note that the number of knots is not always just k). See tprs for what happens in the "tp"/"ts" case. Different terms can use different numbers of knots, unless they share a covariate.

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. Negative elements indicate that the parameter should be estimated, and hence a mixture of fixed and estimated parameters is possible. If smooths share smoothing parameters then length(sp) must correspond to the number of underlying smoothing parameters.

min.sp

Lower bounds can be supplied for the smoothing parameters. Note that if this option is used then the smoothing parameters full.sp, in the returned object, will need to be added to what is supplied here to get the smoothing parameters actually multiplying the penalties. length(min.sp) should always be the same as the total number of penalties (so it may be longer than sp, if smooths share smoothing parameters).

Н

A user supplied fixed quadratic penalty on the parameters of the GAM can be supplied, with this as its coefficient matrix. A common use of this term is to add a ridge penalty to the parameters of the GAM in circumstances in which the model is close to un-identifiable on the scale of the linear predictor, but perfectly well defined on the response scale.

gamma

It is sometimes useful to inflate the model degrees of freedom in the GCV or UBRE/AIC score by a constant multiplier. This allows such a multiplier to be supplied.

fit

If this argument is TRUE then gam sets up the model and fits it, but if it is FALSE then the model is set up and an object G containing what would be required to fit is returned is returned. See argument G.

paraPen

optional list specifying any penalties to be applied to parametric model terms. gam.models explains more.

G

Usually NULL, but may contain the object returned by a previous call to gam with fit=FALSE, in which case all other arguments are ignored except for gamma, in.out, scale, control, method optimizer and fit.

in.out

optional list for initializing outer iteration. If supplied then this must contain two elements: sp should be an array of initialization values for all smoothing parameters (there must be a value for all smoothing parameters, whether fixed or to be estimated, but those for fixed s.p.s are not used); scale is the typical scale of the GCV/UBRE function, for passing to the outer optimizer, or the the initial value of the scale parameter, if this is to be estimated by RE/ML.

drop.unused.levels

by default unused levels are dropped from factors before fitting. For some smooths involving factor variables you might want to turn this off. Only do so if you know what you are doing.

so if you mie what you are doing

further arguments for passing on e.g. to gam. fit (such as mustart).

#### **Details**

A generalized additive model (GAM) is a generalized linear model (GLM) in which the linear predictor is given by a user specified sum of smooth functions of the covariates plus a conventional

parametric component of the linear predictor. A simple example is:

$$\log(E(y_i)) = f_1(x_{1i}) + f_2(x_{2i})$$

where the (independent) response variables  $y_i \sim \text{Poi}$ , and  $f_1$  and  $f_2$  are smooth functions of covariates  $x_1$  and  $x_2$ . The log is an example of a link function.

If absolutely any smooth functions were allowed in model fitting then maximum likelihood estimation of such models would invariably result in complex overfitting estimates of  $f_1$  and  $f_2$ . For this reason the models are usually fit by penalized likelihood maximization, in which the model (negative log) likelihood is modified by the addition of a penalty for each smooth function, penalizing its 'wiggliness'. To control the tradeoff between penalizing wiggliness and penalizing badness of fit each penalty is multiplied by an associated smoothing parameter: how to estimate these parameters, and how to practically represent the smooth functions are the main statistical questions introduced by moving from GLMs to GAMs.

The mgcv implementation of gam represents the smooth functions using penalized regression splines, and by default uses basis functions for these splines that are designed to be optimal, given the number basis functions used. The smooth terms can be functions of any number of covariates and the user has some control over how smoothness of the functions is measured.

gam in mgcv solves the smoothing parameter estimation problem by using the Generalized Cross Validation (GCV) criterion

$$nD/(n-DoF)^2$$

or an Un-Biased Risk Estimator (UBRE )criterion

$$D/n + 2sDoF/n - s$$

where D is the deviance, n the number of data, s the scale parameter and DoF the effective degrees of freedom of the model. Notice that UBRE is effectively just AIC rescaled, but is only used when s is known.

Alternatives are GACV, or a Laplace approximation to REML. There is some evidence that the latter may actually be the most effective choice. The main computational challenge solved by the mgcv package is to optimize the smoothness selection criteria efficiently and reliably. Various alternative numerical methods are provided which can be set by argument optimizer.

Broadly gam works by first constructing basis functions and one or more quadratic penalty coefficient matrices for each smooth term in the model formula, obtaining a model matrix for the strictly parametric part of the model formula, and combining these to obtain a complete model matrix (/design matrix) and a set of penalty matrices for the smooth terms. Some linear identifiability constraints are also obtained at this point. The model is fit using gam. fit, gam. fit3 or varaints, which are modifications of glm.fit. The GAM penalized likelihood maximization problem is solved by Penalized Iteratively Reweighted Least Squares (P-IRLS) (see e.g. Wood 2000). Smoothing parameter selection is integrated in one of two ways. (i) 'Performance iteration' uses the fact that at each P-IRLS iteration a penalized weighted least squares problem is solved, and the smoothing parameters of that problem can estimated by GCV or UBRE. Eventually, in most cases, both model parameter estimates and smoothing parameter estimates converge. (ii) Alternatively the P-IRLS scheme is iterated to convergence for each trial set of smoothing parameters, and GCV, UBRE or REML scores are only evaluated on convergence - optimization is then 'outer' to the P-IRLS loop: in this case the P-IRLS iteration has to be differentiated, to facilitate optimization, and gam.fit3 or one of its variants is used in place of gam.fit. The default is the second method, outer iteration.

Several alternative basis-penalty types are built in for representing model smooths, but alternatives can easily be added (see smooth.terms for an overview and smooth.construct for how to add smooth classes). The choice of the basis dimension (k in the s, te, ti and t2 terms) is something that should be considered carefully (the exact value is not critical, but it is important not to make it restrictively small, nor very large and computationally costly). The basis should be chosen to be larger than is believed to be necessary to approximate the smooth function concerned. The effective degrees of freedom for the smooth will then be controlled by the smoothing penalty on the term, and (usually) selected automatically (with an upper limit set by k-1 or occasionally k). Of course the k should not be made too large, or computation will be slow (or in extreme cases there will be more coefficients to estimate than there are data).

Note that gam assumes a very inclusive definition of what counts as a GAM: basically any penalized GLM can be used: to this end gam allows the non smooth model components to be penalized via argument paraPen and allows the linear predictor to depend on general linear functionals of smooths, via the summation convention mechanism described in linear.functional.terms. link{family.mgcv} details what is available beyond GLMs and the exponential family.

Details of the default underlying fitting methods are given in Wood (2011 and 2004). Some alternative methods are discussed in Wood (2000 and 2006).

gam() is not a clone of Trevor Hastie's original (as supplied in S-PLUS or package gam). The major differences are (i) that by default estimation of the degree of smoothness of model terms is part of model fitting, (ii) a Bayesian approach to variance estimation is employed that makes for easier confidence interval calculation (with good coverage probabilities), (iii) that the model can depend on any (bounded) linear functional of smooth terms, (iv) the parametric part of the model can be penalized, (v) simple random effects can be incorporated, and (vi) the facilities for incorporating smooths of more than one variable are different: specifically there are no 10 smooths, but instead (a) s terms can have more than one argument, implying an isotropic smooth and (b) te, ti or t2 smooths are provided as an effective means for modelling smooth interactions of any number of variables via scale invariant tensor product smooths. Splines on the sphere, Duchon splines and Gaussian Markov Random Fields are also available. (vii) Models beyond the exponential family are available. See gam from package gam, for GAMs via the original Hastie and Tibshirani approach.

#### Value

If fit=FALSE the function returns a list G of items needed to fit a GAM, but doesn't actually fit it. Otherwise the function returns an object of class "gam" as described in gamObject.

### **WARNINGS**

The default basis dimensions used for smooth terms are essentially arbitrary, and it should be checked that they are not too small. See choose.k and gam.check.

You must have more unique combinations of covariates than the model has total parameters. (Total parameters is sum of basis dimensions plus sum of non-spline terms less the number of spline terms).

Automatic smoothing parameter selection is not likely to work well when fitting models to very few response data.

For data with many zeroes clustered together in the covariate space it is quite easy to set up GAMs which suffer from identifiability problems, particularly when using Poisson or binomial families.

The problem is that with e.g. log or logit links, mean value zero corresponds to an infinite range on the linear predictor scale.

## Author(s)

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

Front end design inspired by the S function of the same name based on the work of Hastie and Tibshirani (1990). Underlying methods owe much to the work of Wahba (e.g. 1990) and Gu (e.g. 2002).

#### References

Key References on this implementation:

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. (2004) Stable and efficient multiple smoothing parameter estimation for generalized additive models. J. Amer. Statist. Ass. 99:673-686. [Default method for additive case by GCV (but no longer for generalized)]

Wood, S.N. (2003) Thin plate regression splines. J.R.Statist.Soc.B 65(1):95-114

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

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

Wood S.N., F. Scheipl and J.J. Faraway (2012) Straightforward intermediate rank tensor product smoothing in mixed models. Statistical Computing.

Marra, G and S.N. Wood (2012) Coverage Properties of Confidence Intervals for Generalized Additive Model Components. Scandinavian Journal of Statistics, 39(1), 53-74.

Key Reference on GAMs and related models:

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

Hastie and Tibshirani (1990) Generalized Additive Models. Chapman and Hall.

Wahba (1990) Spline Models of Observational Data. SIAM

Wood, S.N. (2000) Modelling and Smoothing Parameter Estimation with Multiple Quadratic Penalties. J.R.Statist.Soc.B 62(2):413-428 [The original mgcv paper, but no longer the default methods.]

Background References:

Green and Silverman (1994) Nonparametric Regression and Generalized Linear Models. Chapman and Hall.

Gu and Wahba (1991) Minimizing GCV/GML scores with multiple smoothing parameters via the Newton method. SIAM J. Sci. Statist. Comput. 12:383-398

Gu (2002) Smoothing Spline ANOVA Models, Springer.

McCullagh and Nelder (1989) Generalized Linear Models 2nd ed. Chapman & Hall.

O'Sullivan, Yandall and Raynor (1986) Automatic smoothing of regression functions in generalized linear models. J. Am. Statist.Ass. 81:96-103

Wood (2001) mgcv:GAMs and Generalized Ridge Regression for R. R News 1(2):20-25

Wood and Augustin (2002) GAMs with integrated model selection using penalized regression splines and applications to environmental modelling. Ecological Modelling 157:157-177

```
http://www.maths.bris.ac.uk/~sw15190/
```

#### See Also

mgcv-package, gamObject, gam.models, smooth.terms, linear.functional.terms, s, te predict.gam, plot.gam, summary.gam, gam.side, gam.selection, gam.controlgam.check, linear.functional.terms negbin, magic,vis.gam

## **Examples**

```
## see also examples in ?gam.models (e.g. 'by' variables,
## random effects and tricks for large binary datasets)
library(mgcv)
set.seed(2) ## simulate some data...
dat <- gamSim(1,n=400,dist="normal",scale=2)</pre>
b \leftarrow gam(y^s(x0)+s(x1)+s(x2)+s(x3), data=dat)
summary(b)
plot(b,pages=1,residuals=TRUE) ## show partial residuals
plot(b,pages=1,seWithMean=TRUE) ## 'with intercept' CIs
## run some basic model checks, including checking
## smoothing basis dimensions...
gam.check(b)
## same fit in two parts .....
G \leftarrow gam(y \sim s(x0) + s(x1) + s(x2) + s(x3), fit = FALSE, data = dat)
b <- gam(G=G)
print(b)
## change the smoothness selection method to REML
b0 <- gam(y \sim s(x0) + s(x1) + s(x2) + s(x3), data = dat, method = "REML")
## use alternative plotting scheme, and way intervals include
## smoothing parameter uncertainty...
plot(b0,pages=1,scheme=1,unconditional=TRUE)
## Would a smooth interaction of x0 and x1 be better?
## Use tensor product smooth of x0 and x1, basis
## dimension 49 (see ?te for details, also ?t2).
bt <- gam(y \sim te(x0, x1, k=7) + s(x2) + s(x3), data=dat,
          method="REML")
plot(bt,pages=1)
plot(bt,pages=1,scheme=2) ## alternative visualization
AIC(b0,bt) ## interaction worse than additive
## Alternative: test for interaction with a smooth ANOVA
## decomposition (this time between x2 and x1)
bt <- gam(y \sim s(x0) + s(x1) + s(x2) + s(x3) + ti(x1, x2, k=6),
            data=dat,method="REML")
summary(bt)
```

```
## If it is believed that x0 and x1 are naturally on
## the same scale, and should be treated isotropically
## then could try...
bs <- gam(y \sim s(x0, x1, k=40) + s(x2) + s(x3), data=dat,
          method="REML")
plot(bs,pages=1)
AIC(b0,bt,bs) ## additive still better.
## Now do automatic terms selection as well
b1 <- gam(y^s(x0)+s(x1)+s(x2)+s(x3), data=dat,
       method="REML",select=TRUE)
plot(b1,pages=1)
## set the smoothing parameter for the first term, estimate rest ...
bp <- gam(y^s(x0)+s(x1)+s(x2)+s(x3), sp=c(0.01,-1,-1,-1), data=dat)
plot(bp,pages=1,scheme=1)
## alternatively...
bp <- gam(y^s(x0, sp=.01)+s(x1)+s(x2)+s(x3), data=dat)
# set lower bounds on smoothing parameters ....
bp < -gam(y \sim s(x0) + s(x1) + s(x2) + s(x3),
        min.sp=c(0.001,0.01,0,10),data=dat)
print(b);print(bp)
# same with REML
bp < -gam(y \sim s(x0) + s(x1) + s(x2) + s(x3),
        min.sp=c(0.1,0.1,0,10), data=dat, method="REML")
print(b0);print(bp)
## now a GAM with 3df regression spline term & 2 penalized terms
b0 <- gam(y \sim s(x0, k=4, fx=TRUE, bs="tp")+s(x1, k=12)+s(x2, k=15), data=dat)
plot(b0,pages=1)
## now simulate poisson data...
dat <- gamSim(1,n=2000,dist="poisson",scale=.1)</pre>
## use "cr" basis to save time, with 2000 data...
b2 < -gam(y < x0, bs = "cr") + s(x1, bs = "cr") + s(x2, bs = "cr") +
        s(x3,bs="cr"),family=poisson,data=dat,method="REML")
plot(b2,pages=1)
## drop x3, but initialize sp's from previous fit, to
## save more time...
b2a < -gam(y \sim s(x0, bs = "cr") + s(x1, bs = "cr") + s(x2, bs = "cr"),
         family=poisson,data=dat,method="REML",
         in.out=list(sp=b2$sp[1:3],scale=1))
par(mfrow=c(2,2))
plot(b2a)
```

```
par(mfrow=c(1,1))
## similar example using performance iteration
dat <- gamSim(1,n=400,dist="poisson",scale=.25)</pre>
b3 < -gam(y \sim s(x0) + s(x1) + s(x2) + s(x3), family = poisson,
        data=dat,optimizer="perf")
plot(b3,pages=1)
## repeat using GACV as in Wood 2008...
b4 < -gam(y \sim s(x0) + s(x1) + s(x2) + s(x3), family = poisson,
         data=dat,method="GACV.Cp",scale=-1)
plot(b4,pages=1)
## repeat using REML as in Wood 2011...
b5 < -gam(y \sim s(x0) + s(x1) + s(x2) + s(x3), family = poisson,
        data=dat,method="REML")
plot(b5,pages=1)
## a binary example (see ?gam.models for large dataset version)...
dat <- gamSim(1,n=400,dist="binary",scale=.33)</pre>
lr.fit \leftarrow gam(y\sim s(x0)+s(x1)+s(x2)+s(x3),family=binomial,
               data=dat,method="REML")
## plot model components with truth overlaid in red
op <- par(mfrow=c(2,2))</pre>
fn <- c("f0", "f1", "f2", "f3"); xn <- c("x0", "x1", "x2", "x3")
for (k in 1:4) {
  plot(lr.fit,residuals=TRUE,select=k)
  ff <- dat[[fn[k]]];xx <- dat[[xn[k]]]</pre>
  ind <- sort.int(xx,index.return=TRUE)$ix</pre>
  lines(xx[ind],(ff-mean(ff))[ind]*.33,col=2)
}
par(op)
anova(lr.fit)
lr.fit1 \leftarrow gam(y^s(x0)+s(x1)+s(x2),family=binomial,
                data=dat,method="REML")
lr.fit2 \leftarrow gam(y\sim s(x1)+s(x2),family=binomial,
                data=dat,method="REML")
AIC(lr.fit,lr.fit1,lr.fit2)
## For a Gamma example, see ?summary.gam...
## For inverse Gaussian, see ?rig
## now 2D smoothing...
eg <- gamSim(2,n=500,scale=.1)</pre>
```

gam.check 47

```
attach(eg)
op <- par(mfrow=c(2,2), mar=c(4,4,1,1))
contour(truth$x,truth$z,truth$f) ## contour truth
b4 <- gam(y\sim s(x,z), data=data) ## fit model
fit1 <- matrix(predict.gam(b4,pr,se=FALSE),40,40)</pre>
contour(truth$x,truth$z,fit1) ## contour fit
persp(truth$x,truth$z,truth$f)
                                 ## persp truth
vis.gam(b4)
                               ## persp fit
detach(eg)
par(op)
## Not run:
## largish dataset example with user defined knots
par(mfrow=c(2,2))
n <- 5000
eg <- gamSim(2,n=n,scale=.5)</pre>
attach(eg)
ind<-sample(1:n,200,replace=FALSE)</pre>
b5 < -gam(y \sim s(x,z,k=40), data=data,
       knots=list(x=data$x[ind],z=data$z[ind]))
## various visualizations
vis.gam(b5,theta=30,phi=30)
plot(b5)
plot(b5, scheme=1, theta=50, phi=20)
plot(b5,scheme=2)
par(mfrow=c(1,1))
## and a pure "knot based" spline of the same data
b6 < -gam(y \sim s(x,z,k=64), data=data, knots=list(x=rep((1:8-0.5)/8,8),
       z=rep((1:8-0.5)/8,rep(8,8))))
vis.gam(b6,color="heat",theta=30,phi=30)
## varying the default large dataset behaviour via `xt'
b7 \leftarrow gam(y\sim s(x,z,k=40,xt=list(max.knots=500,seed=2)),data=data)
vis.gam(b7,theta=30,phi=30)
detach(eg)
## End(Not run)
```

48 gam.check

### **Description**

Takes a fitted gam object produced by gam() and produces some diagnostic information about the fitting procedure and results. The default is to produce 4 residual plots, some information about the convergence of the smoothness selection optimization, and to run diagnostic tests of whether the basis dimension choises are adequate.

## Usage

## **Arguments**

```
b a fitted gam object as produced by gam().
old.style If you want old fashioned plots, exactly as in Wood, 2006, set to TRUE.
type type of residuals, see residuals.gam, used in all plots.
k.sample Above this k testing uses a random sub-sample of data.
k.rep how many re-shuffles to do to get p-value for k testing.
rep, level, rl.col, rep.col
arguments passed to qq.gam() when old.style is false, see there.
extra graphics parameters to pass to plotting functions.
```

### **Details**

Checking a fitted gam is like checking a fitted glm, with two main differences. Firstly, the basis dimensions used for smooth terms need to be checked, to ensure that they are not so small that they force oversmoothing: the defaults are arbitrary. choose.k provides more detail, but the diagnostic tests described below and reported by this function may also help. Secondly, fitting may not always be as robust to violation of the distributional assumptions as would be the case for a regular GLM, so slightly more care may be needed here. In particular, the thoery of quasi-likelihood implies that if the mean variance relationship is OK for a GLM, then other departures from the assumed distribution are not problematic: GAMs can sometimes be more sensitive. For example, un-modelled overdispersion will typically lead to overfit, as the smoothness selection criterion tries to reduce the scale parameter to the one specified. Similarly, it is not clear how sensitive REML and ML smoothness selection will be to deviations from the assumed response disstribution. For these reasons this routine uses an enhanced residual QQ plot.

This function plots 4 standard diagnostic plots, some smoothing parameter estimation convergence information and the results of tests which may indicate if the smoothing basis dimension for a term is too low.

Usually the 4 plots are various residual plots. For the default optimization methods the convergence information is summarized in a readable way, but for other optimization methods, whatever is returned by way of convergence diagnostics is simply printed.

The test of whether the basis dimension for a smooth is adequate is based on computing an estimate of the residual variance based on differencing residuals that are near neighbours according to the

gam.check 49

(numeric) covariates of the smooth. This estimate divided by the residual variance is the k-index reported. The further below 1 this is, the more likely it is that there is missed pattern left in the residuals. The p-value is computed by simulation: the residuals are randomly re-shuffled k.rep times to obtain the null distribution of the differencing variance estimator, if there is no pattern in the residuals. For models fitted to more than k.sample data, the tests are based of k.sample randomly sampled data. Low p-values may indicate that the basis dimension, k, has been set too low, especially if the reported edf is close to k\', the maximum possible EDF for the term. Note the disconcerting fact that if the test statistic itself is based on random resampling and the null is true, then the associated p-values will of course vary widely from one replicate to the next. Currently smooths of factor variables are not supported and will give an NA p-value.

Doubling a suspect k and re-fitting is sensible: if the reported edf increases substantially then you may have been missing something in the first fit. Of course p-values can be low for reasons other than a too low k. See choose.k for fuller discussion.

The QQ plot produced is usually created by a call to qq.gam, and plots deviance residuals against approximate theoretical quantilies of the deviance residual distribution, according to the fitted model. If this looks odd then investigate further using qq.gam. Note that residuals for models fitted to binary data contain very little information useful for model checking (it is necessary to find some way of aggregating them first), so the QQ plot is unlikely to be useful in this case.

#### Value

A vector of reference quantiles for the residual distribution, if these can be computed.

#### Author(s)

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

## References

N.H. Augustin, E-A Sauleaub, S.N. Wood (2012) On quantile quantile plots for generalized linear models. Computational Statistics & Data Analysis. 56(8), 2404-3409.

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

```
http://www.maths.bris.ac.uk/~sw15190/
```

## See Also

```
choose.k, gam, magic
```

## **Examples**

```
library(mgcv)
set.seed(0)
dat <- gamSim(1,n=200)
b<-gam(y~s(x0)+s(x1)+s(x2)+s(x3),data=dat)
plot(b,pages=1)
gam.check(b,pch=19,cex=.3)</pre>
```

50 gam.control

gam.control

Setting GAM fitting defaults

## Description

This is an internal function of package mgcv which allows control of the numerical options for fitting a GAM. Typically users will want to modify the defaults if model fitting fails to converge, or if the warnings are generated which suggest a loss of numerical stability during fitting. To change the default choise of fitting method, see gam arguments method and optimizer.

#### Usage

#### **Arguments**

nthreads	Some parts of some smoothing parameter selection methods (e.g. REML) can
	use some parallelization in the C code if your R installation supports openMP.

and nthreads is set to more than 1. Note that it is usually better to use the number of physical cores here, rather than the number of hyper-threading cores.

irls.reg For most models this should be 0. The iteratively re-weighted least squares

method by which GAMs are fitted can fail to converge in some circumstances. For example, data with many zeroes can cause problems in a model with a log link, because a mean of zero corresponds to an infinite range of linear predictor values. Such convergence problems are caused by a fundamental lack of identifiability, but do not show up as lack of identifiability in the penalized linear model problems that have to be solved at each stage of iteration. In such circumstances it is possible to apply a ridge regression penalty to the model to

impose identifiability, and irls.reg is the size of the penalty.

epsilon This is used for judging conversion of the GLM IRLS loop in gam.fit or

gam.fit3.

maxit Maximum number of IRLS iterations to perform.

mgcv.tol The convergence tolerance parameter to use in GCV/UBRE optimization.

mgcv.half If a step of the GCV/UBRE optimization method leads to a worse GCV/UBRE

score, then the step length is halved. This is the number of halvings to try before

giving up.

trace Set this to TRUE to turn on diagnostic output.

rank. tol The tolerance used to estimate the rank of the fitting problem.

nlm list of control parameters to pass to nlm if this is used for outer estimation of

smoothing parameters (not default). See details.

gam.control 51

optim list of control parameters to pass to optim if this is used for outer estimation of smoothing parameters (not default). See details. list of control parameters to pass to default Newton optimizer used for outer newton estimation of log smoothing parameters. See details. outerPIsteps The number of performance interation steps used to initialize outer iteration. idLinksBases If smooth terms have their smoothing parameters linked via the id mechanism (see s), should they also have the same bases. Set this to FALSE only if you are sure you know what you are doing (you should almost surely set scalePenalty to FALSE as well in this case). scalePenalty gamm is somewhat sensitive to the absolute scaling of the penalty matrices of a smooth relative to its model matrix. This option rescales the penalty matrices to accomodate this problem. Probably should be set to FALSE if you are linking smoothing parameters but have set idLinkBases to FALSE. keepData Should a copy of the original data argument be kept in the gam object? Strict compatibility with class glm would keep it, but it wastes space to do so. How to estimate the scale parameter for exponential family models estimated by scale.est outer iteration. See gam. scale.

#### **Details**

Outer iteration using newton is controlled by the list newton with the following elements: conv.tol (default 1e-6) is the relative convergence tolerance; maxNstep is the maximum length allowed for an element of the Newton search direction (default 5); maxSstep is the maximum length allowed for an element of the steepest descent direction (only used if Newton fails - default 2); maxHalf is the maximum number of step halvings to permit before giving up (default 30).

If outer iteration using nlm is used for fitting, then the control list nlm stores control arguments for calls to routine nlm. The list has the following named elements: (i) ndigit is the number of significant digits in the GCV/UBRE score - by default this is worked out from epsilon; (ii) gradtol is the tolerance used to judge convergence of the gradient of the GCV/UBRE score to zero - by default set to 10\*epsilon; (iii) stepmax is the maximum allowable log smoothing parameter step - defaults to 2; (iv) steptol is the minimum allowable step length - defaults to 1e-4; (v) iterlim is the maximum number of optimization steps allowed - defaults to 200; (vi) check.analyticals indicates whether the built in exact derivative calculations should be checked numerically - defaults to 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)

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

### References

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. (2004) Stable and efficient multiple smoothing parameter estimation for generalized additive models. J. Amer. Statist. Ass. 99:673-686.

52 gam.convergence

http://www.maths.bris.ac.uk/~sw15190/

#### See Also

gam, gam.fit, glm.control

gam.convergence

GAM convergence and performance issues

## **Description**

When fitting GAMs there is a tradeoff between speed of fitting and probability of fit convergence. The default fitting options, specified by gam arguments method and optimizer, opt for certainty of convergence over speed of fit. In the Generalized Additive Model case it means using 'outer' iteration in preference to 'performance iteration': see gam.outer for details.

It is possible for the default 'outer' iteration to fail when finding intial smoothing parameters using a few steps of performance iteration (if you get a convergence failure message from magic when outer iterating, then this is what has happened): lower outerPIsteps in gam.control to fix this.

There are three things that you can try to speed up GAM fitting. (i) if you have large numbers of smoothing parameters in the generalized case, then try the "bfgs" method option in gam argument optimizer: this can be faster than the default. (ii) Change the optimizer argument to gam so that 'performance iteration' is used in place of the default outer iteration. Usually performance iteration converges well and it can sometimes be quicker than the default outer iteration. (iii) For large datasets it may be worth changing the smoothing basis to use bs="cr" (see s for details) for 1-d smooths, and to use te smooths in place of s smooths for smooths of more than one variable. This is because the default thin plate regression spline basis "tp" is costly to set up for large datasets (much over 1000 data, say). (iv) consider using bam.

If the GAM estimation process fails to converge when using performance iteration, then switch to outer iteration via the optimizer argument of gam. If it still fails, try increasing the number of IRLS iterations (see gam.control) or perhaps experiment with the convergence tolerance.

If you still have problems, it's worth noting that a GAM is just a (penalized) GLM and the IRLS scheme used to estimate GLMs is not guaranteed to converge. Hence non convergence of a GAM may relate to a lack of stability in the basic IRLS scheme. Therefore it is worth trying to establish whether the IRLS iterations are capable of converging. To do this fit the problematic GAM with all smooth terms specified with fx=TRUE so that the smoothing parameters are all fixed at zero. If this 'largest' model can converge then, then the maintainer would quite like to know about your problem! If it doesn't converge, then its likely that your model is just too flexible for the IRLS process itself. Having tried increasing maxit in gam.control, there are several other possibilities for stabilizing the iteration. It is possible to try (i) setting lower bounds on the smoothing parameters using the min.sp argument of gam: this may or may not change the model being fitted; (ii) reducing the flexibility of the model by reducing the basis dimensions k in the specification of s and te model terms: this obviously changes the model being fitted somewhat; (iii) introduce a small regularization term into the fitting via the irls.reg argument of gam.control: this option obviously changes the nature of the fit somewhat, since parameter estimates are pulled towards zero by doing this.

Usually, a major contributer to fitting difficulties is that the model is a very poor description of the data.

gam.fit 53

## Author(s)

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

gam.fit

GAM P-IRLS estimation with GCV/UBRE smoothness estimation

# Description

This is an internal function of package mgcv. It is a modification of the function glm.fit, designed to be called from gam when perfomance iteration is selected (not the default). The major modification is that rather than solving a weighted least squares problem at each IRLS step, a weighted, penalized least squares problem is solved at each IRLS step with smoothing parameters associated with each penalty chosen by GCV or UBRE, using routine magic. For further information on usage see code for gam. Some regularization of the IRLS weights is also permitted as a way of addressing identifiability related problems (see gam.control). Negative binomial parameter estimation is supported.

The basic idea of estimating smoothing parameters at each step of the P-IRLS is due to Gu (1992), and is termed 'performance iteration' or 'performance oriented iteration'.

## Usage

## **Arguments**

G	An object of the type returned by gam when fit=FALSE.
start	Initial values for the model coefficients.
etastart	Initial values for the linear predictor.
mustart	Initial values for the expected response.
family	The family object, specifying the distribution and link to use.
control	Control option list as returned by gam.control.
gamma	Parameter which can be increased to up the cost of each effective degree of freedom in the GCV or AIC/UBRE objective.
fixedSteps	How many steps to take: useful when only using this routine to get rough starting values for other methods.
	Other arguments: ignored.

54 gam.fit3

#### Value

A list of fit information.

## Author(s)

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

#### References

Gu (1992) Cross-validating non-Gaussian data. J. Comput. Graph. Statist. 1:169-179

Gu and Wahba (1991) Minimizing GCV/GML scores with multiple smoothing parameters via the Newton method. SIAM J. Sci. Statist. Comput. 12:383-398

Wood, S.N. (2000) Modelling and Smoothing Parameter Estimation with Multiple Quadratic Penalties. J.R.Statist.Soc.B 62(2):413-428

Wood, S.N. (2004) Stable and efficient multiple smoothing parameter estimation for generalized additive models. J. Amer. Statist. Ass. 99:637-686

#### See Also

```
gam.fit3, gam, magic
```

gam.fit3

P-IRLS GAM estimation with GCV \& UBRE/AIC or RE/ML derivative calculation

# Description

Estimation of GAM smoothing parameters is most stable if optimization of the UBRE/AIC, GCV, GACV, REML or ML score is outer to the penalized iteratively re-weighted least squares scheme used to estimate the model given smoothing parameters.

This routine estimates a GAM (any quadratically penalized GLM) given log smoothing paramaters, and evaluates derivatives of the smoothness selection scores of the model with respect to the log smoothing parameters. Calculation of exact derivatives is generally faster than approximating them by finite differencing, as well as generally improving the reliability of GCV/UBRE/AIC/REML score minimization.

The approach is to run the P-IRLS to convergence, and only then to iterate for first and second derivatives.

Not normally called directly, but rather service routines for gam.

gam.fit3 55

# Usage

```
gam.fit3(x, y, sp, Eb ,UrS=list(),
    weights = rep(1, nobs), start = NULL, etastart = NULL,
    mustart = NULL, offset = rep(0, nobs), U1 = diag(ncol(x)),
    Mp = -1, family = gaussian(), control = gam.control(),
    intercept = TRUE, deriv=2, gamma=1, scale=1,
    printWarn=TRUE, scoreType="REML", null.coef=rep(0, ncol(x)),
    pearson.extra=0, dev.extra=0, n.true=-1, Sl=NULL,...)
```

# **Arguments**

rguments		
x	The model matrix for the GAM (or any penalized GLM).	
У	The response variable.	
sp	The log smoothing parameters.	
Eb	A balanced version of the total penalty matrix: usd for numerical rank determination.	
UrS	List of square root penalties premultiplied by transpose of orthogonal basis for the total penalty.	
weights	prior weights for fitting.	
start	optional starting parameter guesses.	
etastart	optional starting values for the linear predictor.	
mustart	optional starting values for the mean.	
offset	the model offset	
U1	An orthogonal basis for the range space of the penalty — required for ML smoothness estimation only.	
Мр	The dimension of the total penalty null space — required for ML smoothness estimation only.	
family	the family - actually this routine would never be called with gaussian()	
control	control list as returned from glm.control	
intercept	does the model have and intercept, TRUE or FALSE	
deriv	Should derivatives of the GCV and UBRE/AIC scores be calculated? 0, 1 or 2, indicating the maximum order of differentiation to apply.	
gamma	The weight given to each degree of freedom in the GCV and UBRE scores can be varied (usually increased) using this parameter.	
scale	The scale parameter - needed for the UBRE/AIC score.	
printWarn	Set to FALSE to suppress some warnings. Useful in order to ensure that some warnings are only printed if they apply to the final fitted model, rather than an intermediate used in optimization.	
scoreType	specifies smoothing parameter selection criterion to use.	
null.coef	coefficients for a model which gives some sort of upper bound on deviance. This	

allows immediate divergence problems to be controlled.

56 gam.fit3

pearson.extra	Extra component to add to numerator of pearson statistic in P-REML/P-ML smoothness selection criteria.
dev.extra	Extra component to add to deviance for REML/ML type smoothness selection criteria.
n.true	Number of data to assume in smoothness selection criteria. <=0 indicates that it should be the number of rows of X.
Sl	A smooth list suitable for passing to gam.fit5.
	Other arguments: ignored.

#### **Details**

This routine is basically glm. fit with some modifications to allow (i) for quadratic penalties on the log likelihood; (ii) derivatives of the model coefficients with respect to log smoothing parameters to be obtained by use of the implicit function theorem and (iii) derivatives of the GAM GCV, UBRE/AIC, REML or ML scores to be evaluated at convergence.

In addition the routines apply step halving to any step that increases the penalized deviance substantially.

The most costly parts of the calculations are performed by calls to compiled C code (which in turn calls LAPACK routines) in place of the compiled code that would usually perform least squares estimation on the working model in the IRLS iteration.

Estimation of smoothing parameters by optimizing GCV scores obtained at convergence of the P-IRLS iteration was proposed by O'Sullivan et al. (1986), and is here termed 'outer' iteration.

Note that use of non-standard families with this routine requires modification of the families as described in fix.family.link.

## Author(s)

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

The routine has been modified from glm. fit in R 2.0.1, written by the R core (see glm. fit for further credits).

# References

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

O 'Sullivan, Yandall & Raynor (1986) Automatic smoothing of regression functions in generalized linear models. J. Amer. Statist. Assoc. 81:96-103.

```
http://www.maths.bris.ac.uk/~sw15190/
```

#### See Also

```
gam.fit, gam, magic
```

gam.models

Specifying generalized additive models

## Description

This page is intended to provide some more information on how to specify GAMs. A GAM is a GLM in which the linear predictor depends, in part, on a sum of smooth functions of predictors and (possibly) linear functionals of smooth functions of (possibly dummy) predictors.

Specifically let  $y_i$  denote an independent random variable with mean  $\mu_i$  and an exponential family distribution, or failing that a known mean variance relationship suitable for use of quasi-likelihood methods. Then the linear predictor of a GAM has a structure something like

$$g(\mu_i) = \mathbf{X}_i \beta + f_1(x_{1i}, x_{2i}) + f_2(x_{3i}) + L_i f_3(x_4) + \dots$$

where g is a known smooth monotonic 'link' function,  $\mathbf{X}_i\beta$  is the parametric part of the linear predictor, the  $x_j$  are predictor variables, the  $f_j$  are smooth functions and  $L_i$  is some linear functional of  $f_3$ . There may of course be multiple linear functional terms, or none.

The key idea here is that the dependence of the response on the predictors can be represented as a parametric sub-model plus the sum of some (functionals of) smooth functions of one or more of the predictor variables. Thus the model is quite flexible relative to strictly parametric linear or generalized linear models, but still has much more structure than the completely general model that says that the response is just some smooth function of all the covariates.

Note one important point. In order for the model to be identifiable the smooth functions usually have to be constrained to have zero mean (usually taken over the set of covariate values). The constraint is needed if the term involving the smooth includes a constant function in its span. gam always applies such constraints unless there is a by variable present, in which case an assessment is made of whether the constraint is needed or not (see below).

The following sections discuss specifying model structures for gam. Specification of the distribution and link function is done using the family argument to gam and works in the same way as for glm. This page therefore concentrates on the model formula for gam.

## Models with simple smooth terms

Consider the example model.

$$g(\mu_i) = \beta_0 + \beta_1 x_{1i} + \beta_2 x_{2i} + f_1(x_{3i}) + f_2(x_{4i}, x_{5i})$$

where the response variables  $y_i$  has expectation  $\mu_i$  and g is a link function.

The gam formula for this would be

$$y \sim x1 + x2 + s(x3) + s(x4,x5)$$
.

This would use the default basis for the smooths (a thin plate regression spline basis for each), with automatic selection of the effective degrees of freedom for both smooths. The dimension of the smoothing basis is given a default value as well (the dimension of the basis sets an upper limit on the maximum possible degrees of freedom for the basis - the limit is typically one less than basis dimension). Full details of how to control smooths are given in s and te, and further discussion of

basis dimension choice can be found in choose.k. For the moment suppose that we would like to change the basis of the first smooth to a cubic regression spline basis with a dimension of 20, while fixing the second term at 25 degrees of freedom. The appropriate formula would be:

```
y \sim x1 + x2 + s(x3,bs="cr",k=20) + s(x4,x5,k=26,fx=TRUE).
```

The above assumes that  $x_4$  and  $x_5$  are naturally on similar scales (e.g. they might be co-ordinates), so that isotropic smoothing is appropriate. If this assumption is false then tensor product smoothing might be better (see te).

```
y \sim x1 + x2 + s(x3) + te(x4, x5)
```

would generate a tensor product smooth of  $x_4$  and  $x_5$ . By default this smooth would have basis dimension 25 and use cubic regression spline marginals. Varying the defaults is easy. For example  $y \sim x_1 + x_2 + s(x_3) + te(x_4, x_5, bs=c("cr", "ps"), k=c(6,7))$ 

specifies that the tensor product should use a rank 6 cubic regression spline marginal and a rank 7 P-spline marginal to create a smooth with basis dimension 42.

#### Nested terms/functional ANOVA

Sometimes it is interesting to specify smooth models with a main effects + interaction structure such as

$$E(y_i) = f_1(x_i) + f_2(z_i) + f_3(x_i, z_i)$$

or

$$E(y_i) = f_1(x_i) + f_2(z_i) + f_3(v_i) + f_4(x_i, z_i) + f_5(z_i, v_i) + f_6(z_i, v_i) + f_7(x_i, z_i, v_i)$$

for example. Such models should be set up using ti terms in the model formula. For example:

$$y \sim ti(x) + ti(z) + ti(x,z), or$$

$$y \sim ti(x) + ti(z) + ti(v) + ti(x,z) + ti(x,v) + ti(z,v)+ti(x,z,v).$$

The ti terms produce interactions with the component main effects excluded appropriately. (There is in fact no need to use ti terms for the main effects here, s terms could also be used.)

gam allows nesting (or 'overlap') of te and s smooths, and automatically generates side conditions to make such models identifiable, but the resulting models are much less stable and interpretable than those constructed using ti terms.

# 'by' variables

by variables are the means for constructing 'varying-coefficient models' (geographic regression models) and for letting smooths 'interact' with factors or parametric terms. They are also the key to specifying general linear functionals of smooths.

The s and te terms used to specify smooths accept an argument by, which is a numeric or factor variable of the same dimension as the covariates of the smooth. If a by variable is numeric, then its  $i^{th}$  element multiples the  $i^{th}$  row of the model matrix corresponding to the smooth term concerned.

Factor smooth interactions (see also factor.smooth.interaction). If a by variable is a factor then it generates an indicator vector for each level of the factor, unless it is an ordered factor. In the non-ordered case, the model matrix for the smooth term is then replicated for each factor level, and each copy has its rows multiplied by the corresponding rows of its indicator variable. The smoothness penalties are also duplicated for each factor level. In short a different smooth is generated for each factor level (the id argument to s and te can be used to force all such smooths to have the same smoothing parameter). ordered by variables are handled in the same way, except that no smooth is generated for the first level of the ordered factor (see b3 example below). This is

useful for setting up identifiable models when the same smooth occurs more than once in a model, with different factor by variables.

As an example, consider the model

$$E(y_i) = \beta_0 + f(x_i)z_i$$

where f is a smooth function, and  $z_i$  is a numeric variable. The appropriate formula is:  $y \sim s(x,by=z)$ 

- the by argument ensures that the smooth function gets multiplied by covariate z. Note that when using factor by variables, centering constraints are applied to the smooths, which usually means that the by variable should be included as a parametric term, as well.

The example code below also illustrates the use of factor by variables.

by variables may be supplied as numeric matrices as part of specifying general linear functional terms

If a by variable is present and numeric (rather than a factor) then the corresponding smooth is only subjected to an identifiability constraint if (i) the by variable is a constant vector, or, (ii) for a matrix by variable, L, if L%\*%rep(1,ncol(L)) is constant or (iii) if a user defined smooth constructor supplies an identifiability constraint explicitly, and that constraint has an attibute "always.apply".

## Linking smooths with 'id'

It is sometimes desirable to insist that different smooth terms have the same degree of smoothness. This can be done by using the id argument to s or te terms. Smooths which share an id will have the same smoothing parameter. Really this only makes sense if the smooths use the same basis functions, and the default behaviour is to force this to happen: all smooths sharing an id have the same basis functions as the first smooth occurring with that id. Note that if you want exactly the same function for each smooth, then this is best achieved by making use of the summation convention covered under 'linear functional terms'.

As an example suppose that  $E(y_i) \equiv \mu_i$  and

$$g(\mu_i) = f_1(x_{1i}) + f_2(x_{2i}, x_{3i}) + f_3(x_{4i})$$

but that  $f_1$  and  $f_3$  should have the same smoothing parameters (and  $x_2$  and  $x_3$  are on different scales). Then the gam formula

$$y \sim s(x1,id=1) + te(x_2,x3) + s(x4,id=1)$$

would achieve the desired result. id can be numbers or character strings. Giving an id to a term with a factor by variable causes the smooths at each level of the factor to have the same smoothing parameter.

Smooth term ids are not supported by gamm.

#### Linear functional terms

General linear functional terms have a long history in the spline literature including in the penalized GLM context (see e.g. Wahba 1990). Such terms encompass varying coefficient models/ geographic regression, functional GLMs (i.e. GLMs with functional predictors), GLASS models, etc, and allow smoothing with respect to aggregated covariate values, for example.

Such terms are implemented in mgcv using a simple 'summation convention' for smooth terms: If the covariates of a smooth are supplied as matrices, then summation of the evaluated smooth over

the columns of the matrices is implied. Each covariate matrix and any by variable matrix must be of the same dimension. Consider, for example the term s(X,Z,by=L)

where X, Z and L are  $n \times p$  matrices. Let f denote the thin plate regression spline specified. The resulting contibution to the  $i^{\rm th}$  element of the linear predictor is

$$\sum_{j=1}^{p} L_{ij} f(X_{ij}, Z_{ij})$$

If no L is supplied then all its elements are taken as 1. In R code terms, let F denote the  $n \times p$  matrix obtained by evaluating the smooth at the values in X and Z. Then the contribution of the term to the linear predictor is rowSums(L\*F) (note that it's element by element multiplication here!).

The summation convention applies to te terms as well as s terms. More details and examples are provided in linear.functional.terms.

#### Random effects

Random effects can be added to gam models using s(...,bs="re") terms (see smooth.construct.re.smooth.spec), or the paraPen argument to gam covered below. See gam.vcomp, random.effects and smooth.construct.re.smooth.spec for further details. An alternative is to use the approach of gamm.

# Penalizing the parametric terms

In case the ability to add smooth classes, smooth identities, by variables and the summation convention are still not sufficient to implement exactly the penalized GLM that you require, gam also allows you to penalize the parametric terms in the model formula. This is mostly useful in allowing one or more matrix terms to be included in the formula, along with a sequence of quadratic penalty matrices for each.

Suppose that you have set up a model matrix X, and want to penalize the corresponding coefficients,  $\beta$  with two penalties  $\beta^T \mathbf{S}_1 \beta$  and  $\beta^T \mathbf{S}_2 \beta$ . Then something like the following would be appropriate: gam(y ~ X - 1,paraPen=list(X=list(S1,S2)))

The paraPen argument should be a list with elements having names corresponding to the terms being penalized. Each element of paraPen is itself a list, with optional elements L, rank and sp: all other elements must be penalty matrices. If present, rank is a vector giving the rank of each penalty matrix (if absent this is determined numerically). L is a matrix that maps underlying log smoothing parameters to the log smoothing parameters that actually multiply the individual quadratic penalties: taken as the identity if not supplied. sp is a vector of (underlying) smoothing parameter values: positive values are taken as fixed, negative to signal that the smoothing parameter should be estimated. Taken as all negative if not supplied.

An obvious application of paraPen is to incorporate random effects, and an example of this is provided below. In this case the supplied penalty matrices will be (generalized) inverse covariance matrices for the random effects — i.e. precision matrices. The final estimate of the covariance matrix corresponding to one of these penalties is given by the (generalized) inverse of the penalty matrix multiplied by the estimated scale parameter and divided by the estimated smoothing parameter for the penalty. For example, if you use an identity matrix to penalize some coefficients that are to be viewed as i.i.d. Gaussian random effects, then their estimated variance will be the estimated scale parameter divided by the estimate of the smoothing parameter, for this penalty. See the 'rail' example below.

P-values for penalized parametric terms should be treated with caution. If you must have them, then use the option freq=TRUE in anova.gam and summary.gam, which will tend to give reasonable results for random effects implemented this way, but not for terms with a rank defficient penalty (or penalties with a wide eigen-spectrum).

### Author(s)

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

#### References

Wahba (1990) Spline Models of Observational Data SIAM.

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

## **Examples**

```
require(mgcv)
set.seed(10)
## simulate date from y = f(x2)*x1 + error
dat <- gamSim(3, n=400)
b < -gam(y \sim s(x2,by=x1),data=dat)
plot(b,pages=1)
summary(b)
## Factor `by' variable example (with a spurious covariate x0)
## simulate data...
dat <- gamSim(4)
## fit model...
b \leftarrow gam(y \sim fac+s(x2,by=fac)+s(x0),data=dat)
plot(b,pages=1)
summary(b)
## note that the preceding fit is the same as....
b1 < -gam(y \sim s(x2, by=as.numeric(fac==1)) + s(x2, by=as.numeric(fac==2)) +
            s(x2,by=as.numeric(fac==3))+s(x0)-1,data=dat)
## ... the `-1' is because the intercept is confounded with the
## *uncentred* smooths here.
plot(b1,pages=1)
summary(b1)
## repeat forcing all s(x2) terms to have the same smoothing param
## (not a very good idea for these data!)
b2 \leftarrow gam(y \sim fac+s(x2,by=fac,id=1)+s(x0),data=dat)
plot(b2,pages=1)
summary(b2)
## now repeat with a single reference level smooth, and
## two `difference' smooths...
```

```
dat$fac <- ordered(dat$fac)</pre>
b3 <- gam(y ~ fac+s(x2)+s(x2,by=fac)+s(x0),data=dat,method="REML")
plot(b3,pages=1)
summary(b3)
rm(dat)
## An example of a simple random effects term implemented via
## penalization of the parametric part of the model...
dat <- gamSim(1,n=400,scale=2) ## simulate 4 term additive truth
## Now add some random effects to the simulation. Response is
## grouped into one of 20 groups by `fac' and each groups has a
## random effect added....
fac <- as.factor(sample(1:20,400,replace=TRUE))</pre>
dat$X <- model.matrix(~fac-1)</pre>
b <- rnorm(20)*.5
dat$y <- dat$y + dat$X%*%b
## now fit appropriate random effect model...
PP <- list(X=list(rank=20,diag(20)))
rm <- gam(y^{x} X+s(x0)+s(x1)+s(x2)+s(x3), data=dat, paraPen=PP)
plot(rm,pages=1)
## Get estimated random effects standard deviation...
sig.b <- sqrt(rm$sig2/rm$sp[1]);sig.b</pre>
## a much simpler approach uses "re" terms...
rm1 <- gam(y ~ s(fac,bs="re")+s(x0)+s(x1)+s(x2)+s(x3),data=dat,method="ML")
gam.vcomp(rm1)
## Simple comparison with lme, using Rail data.
## See ?random.effects for a simpler method
require(nlme)
b0 <- lme(travel~1,data=Rail,~1|Rail,method="ML")</pre>
Z <- model.matrix(~Rail-1,data=Rail,</pre>
     contrasts.arg=list(Rail="contr.treatment"))
b <- gam(travel~Z,data=Rail,paraPen=list(Z=list(diag(6))),method="ML")</pre>
(b$reml.scale/b$sp)^.5 ## `gam' ML estimate of Rail sd
b$reml.scale^.5
                       ## `gam' ML estimate of residual sd
b0 <- lme(travel~1,data=Rail,~1|Rail,method="REML")</pre>
Z <- model.matrix(~Rail-1,data=Rail,</pre>
     contrasts.arg=list(Rail="contr.treatment"))
b <- gam(travel~Z,data=Rail,paraPen=list(Z=list(diag(6))),method="REML")</pre>
h0
(b$reml.scale/b$sp)^.5 ## `gam' REML estimate of Rail sd
                       ## `gam' REML estimate of residual sd
b$reml.scale^.5
```

gam.outer 63

```
## Approximate large dataset logistic regression for rare events
## based on subsampling the zeroes, and adding an offset to
## approximately allow for this.
## Doing the same thing, but upweighting the sampled zeroes
## leads to problems with smoothness selection, and CIs.
n <- 50000 ## simulate n data
dat <- gamSim(1,n=n,dist="binary",scale=.33)</pre>
p <- binomial()$linkinv(dat$f-6) ## make 1's rare</pre>
dat$y <- rbinom(p,1,p)</pre>
                                                         ## re-simulate rare response
## Now sample all the 1's but only proportion S of the 0's
                                                            ## sampling fraction of zeroes
dat <- dat[dat$y==1 | runif(n) < S,] ## sampling</pre>
## Create offset based on total sampling fraction
dat$s <- rep(log(nrow(dat)/n),nrow(dat))</pre>
lr.fit \leftarrow gam(y^s(x0,bs="cr")+s(x1,bs="cr")+s(x2,bs="cr")+s(x3,bs="cr")+s(x3,bs="cr")+s(x3,bs="cr")+s(x3,bs="cr")+s(x3,bs="cr")+s(x3,bs="cr")+s(x3,bs="cr")+s(x3,bs="cr")+s(x3,bs="cr")+s(x3,bs="cr")+s(x3,bs="cr")+s(x3,bs="cr")+s(x3,bs="cr")+s(x3,bs="cr")+s(x3,bs="cr")+s(x3,bs="cr")+s(x3,bs="cr")+s(x3,bs="cr")+s(x3,bs="cr")+s(x3,bs="cr")+s(x3,bs="cr")+s(x3,bs="cr")+s(x3,bs="cr")+s(x3,bs="cr")+s(x3,bs="cr")+s(x3,bs="cr")+s(x3,bs="cr")+s(x3,bs="cr")+s(x3,bs="cr")+s(x3,bs="cr")+s(x3,bs="cr")+s(x3,bs="cr")+s(x3,bs="cr")+s(x3,bs="cr")+s(x3,bs="cr")+s(x3,bs="cr")+s(x3,bs="cr")+s(x3,bs="cr")+s(x3,bs="cr")+s(x3,bs="cr")+s(x3,bs="cr")+s(x3,bs="cr")+s(x3,bs="cr")+s(x3,bs="cr")+s(x3,bs="cr")+s(x3,bs="cr")+s(x3,bs="cr")+s(x3,bs="cr")+s(x3,bs="cr")+s(x3,bs="cr")+s(x3,bs="cr")+s(x3,bs="cr")+s(x3,bs="cr")+s(x3,bs="cr")+s(x3,bs="cr")+s(x3,bs="cr")+s(x3,bs="cr")+s(x3,bs="cr")+s(x3,bs="cr")+s(x3,bs="cr")+s(x3,bs="cr")+s(x3,bs="cr")+s(x3,bs="cr")+s(x3,bs="cr")+s(x3,bs="cr")+s(x3,bs="cr")+s(x3,bs="cr")+s(x3,bs="cr")+s(x3,bs="cr")+s(x3,bs="cr")+s(x3,bs="cr")+s(x3,bs="cr")+s(x3,bs="cr")+s(x3,bs="cr")+s(x3,bs="cr")+s(x3,bs="cr")+s(x3,bs="cr")+s(x3,bs="cr")+s(x3,bs="cr")+s(x3,bs="cr")+s(x3,bs="cr")+s(x3,bs="cr")+s(x3,bs="cr")+s(x3,bs="cr")+s(x3,bs="cr")+s(x3,bs="cr")+s(x3,bs="cr")+s(x3,bs="cr")+s(x3,bs="cr")+s(x3,bs="cr")+s(x3,bs="cr")+s(x3,bs="cr")+s(x3,bs="cr")+s(x3,bs="cr")+s(x3,bs="cr")+s(x3,bs="cr")+s(x3,bs="cr")+s(x3,bs="cr")+s(x3,bs="cr")+s(x3,bs="cr")+s(x3,bs="cr")+s(x3,bs="cr")+s(x3,bs="cr")+s(x3,bs="cr")+s(x3,bs="cr")+s(x3,bs="cr")+s(x3,bs="cr")+s(x3,bs="cr")+s(x3,bs="cr")+s(x3,bs="cr")+s(x3,bs="cr")+s(x3,bs="cr")+s(x3,bs="cr")+s(x3,bs="cr")+s(x3,bs="cr")+s(x3,bs="cr")+s(x3,bs="cr")+s(x3,bs="cr")+s(x3,bs="cr")+s(x3,bs="cr")+s(x3,bs="cr")+s(x3,bs="cr")+s(x3,bs="cr")+s(x3,bs="cr")+s(x3,bs="cr")+s(x3,bs="cr")+s(x3,bs="cr")+s(x3,bs="cr")+s(x3,bs="cr")+s(x3,bs="cr")+s(x3,bs="cr")+s(x3,bs="cr")+s(x3,bs="cr")+s(x3,bs="cr")+s(x3,bs="cr")+s(x3,bs="cr")+s(x3,bs="cr")+s(x3,bs="cr")+s(x3,bs="cr")+s(x3,bs="cr")+s(x3,bs="cr")+s(x3,bs="cr")
                             offset(s),family=binomial,data=dat,method="REML")
## plot model components with truth overlaid in red
op <- par(mfrow=c(2,2))</pre>
fn <- c("f0", "f1", "f2", "f3");xn <- c("x0", "x1", "x2", "x3")
for (k in 1:4) {
              plot(lr.fit,select=k,scale=0)
              ff <- dat[[fn[k]]];xx <- dat[[xn[k]]]</pre>
              ind <- sort.int(xx,index.return=TRUE)$ix</pre>
              lines(xx[ind],(ff-mean(ff))[ind]*.33,col=2)
}
par(op)
rm(dat)
## A Gamma example, by modify `gamSim' output...
dat <- gamSim(1,n=400,dist="normal",scale=1)</pre>
dat$f <- dat$f/4 ## true linear predictor</pre>
Ey <- exp(dat$f);scale <- .5 ## mean and GLM scale parameter
## Note that `shape' and `scale' in `rgamma' are almost
## opposite terminology to that used with GLM/GAM...
dat$y <- rgamma(Ey*0,shape=1/scale,scale=Ey*scale)</pre>
bg <- gam(y^{-} s(x0) + s(x1) + s(x2) + s(x3), family=Gamma(link=log),
                     data=dat,method="REML")
plot(bg,pages=1,scheme=1)
```

64 gam.outer

## **Description**

Estimation of GAM smoothing parameters is most stable if optimization of the smoothness selection score (GCV, GACV, UBRE/AIC, REML, ML etc) is outer to the penalized iteratively re-weighted least squares scheme used to estimate the model given smoothing parameters.

This routine optimizes a smoothness selection score in this way. Basically the score is evaluated for each trial set of smoothing parameters by estimating the GAM for those smoothing parameters. The score is minimized w.r.t. the parameters numerically, using newton (default), bfgs, optim or nlm. Exact (first and second) derivatives of the score can be used by fitting with gam. fit3. This improves efficiency and reliability relative to relying on finite difference derivatives.

Not normally called directly, but rather a service routine for gam.

## Usage

## **Arguments**

lsp	The log smoothing parameters.
fscale	Typical scale of the GCV or UBRE/AIC score.
family	the model family.
control	control argument to pass to gam. fit if pure finite differencing is being used.
method	method argument to gam defining the smoothness criterion to use (but depending on whether or not scale known).
optimizer	The argument to gam defining the numerical optimization method to use.
criterion	Which smoothness selction criterion to use. One of "UBRE", "GCV", "GACV", "REML" or "P-REML".
scale	Supplied scale parameter. Positive indicates known.
gamma	The degree of freedom inflation factor for the GCV/UBRE/AIC score.
G	List produced by mgcv:::gam.setup, containing most of what's needed to actually fit a GAM.
	other arguments, typically for passing on to gam. fit3 (ultimately).

## **Details**

See Wood (2008) for full details on 'outer iteration'.

## Author(s)

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

#### References

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

```
http://www.maths.bris.ac.uk/~sw15190/
```

gam.scale 65

## See Also

gam.fit3, gam, magic

gam.scale

Scale parameter estimation in GAMs

## **Description**

Scale parameter estimation in gam depends on the type of family. For extended families then the RE/ML estimate is used. For conventional exponential families, estimated by the default outer iteration, the scale estimator can be controlled using argument scale.est in gam.control. The options are "fletcher" (default), "pearson" or "deviance". The Pearson estimator is the (weighted) sum of squares of the pearson residuals, divided by the effective residual degrees of freedom. The Fletcher (2012) estimator is an improved version of the Pearson estimator. The deviance estimator simply substitutes deviance residuals for Pearson residuals.

Usually the Pearson estimator is recommended for GLMs, since it is asymptotically unbiased. However, it can also be unstable at finite sample sizes, if a few Pearson residuals are very large. For example, a very low Poisson mean with a non zero count can give a huge Pearson residual, even though the deviance residual is much more modest. The Fletcher (2012) estimator is designed to reduce these problems.

For performance iteration the Pearson estimator is always used.

gamm uses the estimate of the scale parameter from the underlying call to lme. bam uses the REML estimator if the method is "fREML". Otherwise the estimator is a Pearson estimator.

# Author(s)

Simon N. Wood <simon.wood@r-project.org> with help from Mark Bravington and David Peel

#### References

Fletcher, David J. (2012) Estimating overdispersion when fitting a generalized linear model to sparse data. Biometrika 99(1), 230-237.

### See Also

gam.control

66 gam.selection

gam.selection

Generalized Additive Model Selection

## **Description**

This page is intended to provide some more information on how to select GAMs. In particular, it gives a brief overview of smoothness selection, and then discusses how this can be extended to select inclusion/exclusion of terms. Hypothesis testing approaches to the latter problem are also discussed.

## Smoothness selection criteria

Given a model structure specified by a gam model formula, gam() attempts to find the appropriate smoothness for each applicable model term using prediction error criteria or likelihood based methods. The prediction error criteria used are Generalized (Approximate) Cross Validation (GCV or GACV) when the scale parameter is unknown or an Un-Biased Risk Estimator (UBRE) when it is known. UBRE is essentially scaled AIC (Generalized case) or Mallows' Cp (additive model case). GCV and UBRE are covered in Craven and Wahba (1979) and Wahba (1990). Alternatively REML of maximum likelihood (ML) may be used for smoothness selection, by viewing the smooth components as random effects (in this case the variance component for each smooth random effect will be given by the scale parameter divided by the smoothing parameter — for smooths with multiple penalties, there will be multiple variance components). The method argument to gam selects the smoothness selection criterion.

Automatic smoothness selection is unlikely to be successful with few data, particularly with multiple terms to be selected. In addition GCV and UBRE/AIC score can occasionally display local minima that can trap the minimisation algorithms. GCV/UBRE/AIC scores become constant with changing smoothing parameters at very low or very high smoothing parameters, and on occasion these 'flat' regions can be separated from regions of lower score by a small 'lip'. This seems to be the most common form of local minimum, but is usually avoidable by avoiding extreme smoothing parameters as starting values in optimization, and by avoiding big jumps in smoothing parameters while optimizing. Never the less, if you are suspicious of smoothing parameter estimates, try changing fit method (see gam arguments method and optimizer) and see if the estimates change, or try changing some or all of the smoothing parameters 'manually' (argument sp of gam, or sp arguments to s or te).

REML and ML are less prone to local minima than the other criteria, and may therefore be preferable.

## **Automatic term selection**

Unmodified smoothness selection by GCV, AIC, REML etc. will not usually remove a smooth from a model. This is because most smoothing penalties view some space of (non-zero) functions as 'completely smooth' and once a term is penalized heavily enough that it is in this space, further penalization does not change it.

However it is straightforward to modify smooths so that under heavy penalization they are penalized to the zero function and thereby 'selected out' of the model. There are two approaches.

gam.selection 67

The first approach is to modify the smoothing penalty with an additional shrinkage term. Smooth classescs.smooth and tprs.smooth (specified by "cs" and "ts" respectively) have smoothness penalties which include a small shrinkage component, so that for large enough smoothing parameters the smooth becomes identically zero. This allows automatic smoothing parameter selection methods to effectively remove the term from the model altogether. The shrinkage component of the penalty is set at a level that usually makes negligable contribution to the penalization of the model, only becoming effective when the term is effectively 'completely smooth' according to the conventional penalty.

The second approach leaves the original smoothing penalty unchanged, but constructs an additional penalty for each smooth, which penalizes only functions in the null space of the original penalty (the 'completely smooth' functions). Hence, if all the smoothing parameters for a term tend to infinity, the term will be selected out of the model. This latter approach is more expensive computationally, but has the advantage that it can be applied automatically to any smooth term. The select argument to gam turns on this method.

In fact, as implemented, both approaches operate by eigen-decomposiong the original penalty matrix. A new penalty is created on the null space: it is the matrix with the same eigenvectors as the original penalty, but with the originally postive egienvalues set to zero, and the originally zero eigenvalues set to something positive. The first approach just addes a multiple of this penalty to the original penalty, where the multiple is chosen so that the new penalty can not dominate the original. The second approach treats the new penalty as an extra penalty, with its own smoothing parameter.

Of course, as with all model selection methods, some care must be take to ensure that the automatic selection is sensible, and a decision about the effective degrees of freedom at which to declare a term 'negligible' has to be made.

## **Interactive term selection**

In general the most logically consistent method to use for deciding which terms to include in the model is to compare GCV/UBRE/ML scores for models with and without the term (REML scores should not be used to compare models with different fixed effects structures). When UBRE is the smoothness selection method this will give the same result as comparing by AIC (the AIC in this case uses the model EDF in place of the usual model DF). Similarly, comparison via GCV score and via AIC seldom yields different answers. Note that the negative binomial with estimated theta parameter is a special case: the GCV score is not informative, because of the theta estimation scheme used. More generally the score for the model with a smooth term can be compared to the score for the model with the smooth term replaced by appropriate parametric terms. Candidates for replacement by parametric terms are smooth terms with estimated degrees of freedom close to their minimum possible.

Candidates for removal can also be identified by reference to the approximate p-values provided by summary.gam, and by looking at the extent to which the confidence band for an estimated term includes the zero function. It is perfectly possible to perform backwards selection using p-values in the usual way: that is by sequentially dropping the single term with the highest non-significant p-value from the model and re-fitting, until all terms are significant. This suffers from the same problems as stepwise procedures for any GLM/LM, with the additional caveat that the p-values are only approximate. If adopting this approach, it is probably best to use ML smoothness selection.

Note that GCV and UBRE are not appropriate for comparing models using different families: in that case AIC should be used.

68 gam.selection

## Caveats/platitudes

Formal model selection methods are only appropriate for selecting between reasonable models. If formal model selection is attempted starting from a model that simply doesn't fit the data, then it is unlikely to provide meaningful results.

The more thought is given to appropriate model structure up front, the more successful model selection is likely to be. Simply starting with a hugely flexible model with 'everything in' and hoping that automatic selection will find the right structure is not often successful.

#### Author(s)

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

#### References

Marra, G. and S.N. Wood (2011) Practical variable selection for generalized additive models. Computational Statistics and Data Analysis 55,2372-2387.

Craven and Wahba (1979) Smoothing Noisy Data with Spline Functions. Numer. Math. 31:377-403

Venables and Ripley (1999) Modern Applied Statistics with S-PLUS

Wahba (1990) Spline Models of Observational Data. SIAM.

Wood, S.N. (2003) Thin plate regression splines. J.R.Statist.Soc.B 65(1):95-114

Wood, S.N. (2008) Fast stable direct fitting and smoothness selection for generalized additive models. J.R.Statist. Soc. 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

```
http://www.maths.bris.ac.uk/~sw15190/
```

# See Also

```
gam, step.gam
```

#### **Examples**

gam.side 69

gaii. Side Identifiability side conditions for a GAM	gam.side	Identifiability side conditions for a GAM	
--	----------	---	--

#### **Description**

GAM formulae with repeated variables may only correspond to identifiable models given some side conditions. This routine works out appropriate side conditions, based on zeroing redundant parameters. It is called from mgcv:::gam.setup and is not intended to be called by users.

The method identifies nested and repeated variables by their names, but numerically evaluates which constraints need to be imposed. Constraints are always applied to smooths of more variables in preference to smooths of fewer variables. The numerical approach allows appropriate constraints to be applied to models constructed using any smooths, including user defined smooths.

## Usage

```
gam.side(sm, Xp, tol=.Machine$double.eps^.5, with.pen=FALSE)
```

# **Arguments**

sm	A list of smooth objects as returned by smooth.construct.
Хр	The model matrix for the strictly parametric model components.
tol	The tolerance to use when assessing linear dependence of smooths.
with.pen	Should the computation of dependence consider the penalties or not. Doing so will lead to fewer constraints.

### **Details**

Models such as  $y^s(x)+s(z)+s(x,z)$  can be estimated by gam, but require identifiability constraints to be applied, to make them identifiable. This routine does this, effectively setting redundant parameters to zero. When the redundancy is between smooths of lower and higher numbers of variables, the constraint is always applied to the smooth of the higher number of variables.

Dependent smooths are identified symbolically, but which constraints are needed to ensure identifiability of these smooths is determined numerically, using fixDependence. This makes the routine rather general, and not dependent on any particular basis.

Xp is used to check whether there is a constant term in the model (or columns that can be linearly combined to give a constant). This is because centred smooths can appear independent, when they would be dependent if there is a constant in the model, so dependence testing needs to take account of this.

#### Value

A list of smooths, with model matrices and penalty matrices adjusted to automatically impose the required constraints. Any smooth that has been modified will have an attribute "del.index", listing the columns of its model matrix that were deleted. This index is used in the creation of prediction matrices for the term.

70 gam.side

## WARNINGS

Much better statistical stability will be obtained by using models like  $y^s(x)+s(z)+ti(x,z)$  or  $y^ti(x)+ti(z)+ti(x,z)$  rather than  $y^s(x)+s(z)+s(x,z)$ , since the former are designed not to require further constraint.

#### Author(s)

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

#### See Also

```
ti, gam. models
```

## **Examples**

```
## The first two examples here iluustrate models that cause
## gam.side to impose constraints, but both are a bad way
## of estimating such models. The 3rd example is the right
## way....
set.seed(7)
require(mgcv)
dat <- gamSim(n=400,scale=2) ## simulate data</pre>
## estimate model with redundant smooth interaction (bad idea).
b < -gam(y \sim s(x0) + s(x1) + s(x0,x1) + s(x2), data = dat)
plot(b,pages=1)
## Simulate data with real interation...
dat <- gamSim(2, n=500, scale=.1)
old.par<-par(mfrow=c(2,2))
## a fully nested tensor product example (bad idea)
b \leftarrow gam(y^s(x,bs="cr",k=6)+s(z,bs="cr",k=6)+te(x,z,k=6),
       data=dat$data)
plot(b)
old.par<-par(mfrow=c(2,2))
## A fully nested tensor product example, done properly,
## so that gam.side is not needed to ensure identifiability.
## ti terms are designed to produce interaction smooths
## suitable for adding to main effects (we could also have
## used s(x) and s(z) without a problem, but not s(z,x)
## or te(z,x)).
b \leftarrow gam(y \sim ti(x,k=6) + ti(z,k=6) + ti(x,z,k=6),
       data=dat$data)
plot(b)
par(old.par)
rm(dat)
```

gam.vcomp 71

gam.vcomp	Report gam smoothness estimates as variance components

## **Description**

GAMs can be viewed as mixed models, where the smoothing parameters are related to variance components. This routine extracts the estimated variance components associated with each smooth term, and if possible returns confidence intervals on the standard deviation scale.

### Usage

```
gam.vcomp(x,rescale=TRUE,conf.lev=.95)
```

## Arguments

a fitted model object of class gam as produced by gam().

the penalty matrices for smooths are rescaled before fitting, for numerical stability reasons, if TRUE this rescaling is reversed, so that the variance components are on the original scale.

conf.lev when the smoothing parameters are estimated by REML or ML, then confidence

when the smoothing parameters are estimated by REML or ML, then confidence intervals for the variance components can be obtained from large sample likeli-

hood results. This gives the confidence level to work at.

### **Details**

The (pseudo) inverse of the penalty matrix penalizing a term is proportional to the covariance matrix of the term's coefficients, when these are viewed as random. For single penalty smooths, it is possible to compute the variance component for the smooth (which multiplies the inverse penalty matrix to obtain the covariance matrix of the smooth's coefficients). This variance component is given by the scale parameter divided by the smoothing parameter.

This routine computes such variance components, for gam models, and associated confidence intervals, if smoothing parameter estimation was likelihood based. Note that variance components are also returned for tensor product smooths, but that their interpretation is not so straightforward.

The routine is particularly useful for model fitted by gam in which random effects have been incorporated.

#### Value

Either a vector of variance components for each smooth term (as standard deviations), or a matrix. The first column of the matrix gives standard deviations for each term, while the subsequent columns give lower and upper confidence bounds, on the same scale.

For models in which there are more smoothing parameters than actually estimated (e.g. if some were fixed, or smoothing parameters are linked) then a list is returned. The vc element is as above, the all element is a vector of variance components for all the smoothing parameters (estimated + fixed or replicated).

The routine prints a table of estimated standard deviations and confidence limits, if these can be computed, and reports the numerical rank of the covariance matrix.

72 gam2objective

### Author(s)

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

#### References

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

```
smooth.construct.re.smooth.spec
```

## **Examples**

```
set.seed(3)
require(mgcv)
## simulate some data, consisting of a smooth truth + random effects
dat <- gamSim(1,n=400,dist="normal",scale=2)</pre>
a <- factor(sample(1:10,400,replace=TRUE))</pre>
b <- factor(sample(1:7,400,replace=TRUE))</pre>
Xa <- model.matrix(~a-1)</pre>
                                                                                                                    ## random main effects
Xb <- model.matrix(~b-1)</pre>
Xab <- model.matrix(~a:b-1) ## random interaction</pre>
dat$y <- dat$y + Xa%*%rnorm(10)*.5 +
                                      Xb%*%rnorm(7)*.3 + Xab%*%rnorm(70)*.7
dat$a <- a;dat$b <- b
## Fit the model using "re" terms, and smoother linkage
mod \leftarrow gam(y \sim s(a,bs="re")+s(b,bs="re")+s(a,b,bs="re")+s(x0,id=1)+s(x1,id=1)+s(x1,id=1)+s(x1,id=1)+s(x1,id=1)+s(x1,id=1)+s(x1,id=1)+s(x1,id=1)+s(x1,id=1)+s(x1,id=1)+s(x1,id=1)+s(x1,id=1)+s(x1,id=1)+s(x1,id=1)+s(x1,id=1)+s(x1,id=1)+s(x1,id=1)+s(x1,id=1)+s(x1,id=1)+s(x1,id=1)+s(x1,id=1)+s(x1,id=1)+s(x1,id=1)+s(x1,id=1)+s(x1,id=1)+s(x1,id=1)+s(x1,id=1)+s(x1,id=1)+s(x1,id=1)+s(x1,id=1)+s(x1,id=1)+s(x1,id=1)+s(x1,id=1)+s(x1,id=1)+s(x1,id=1)+s(x1,id=1)+s(x1,id=1)+s(x1,id=1)+s(x1,id=1)+s(x1,id=1)+s(x1,id=1)+s(x1,id=1)+s(x1,id=1)+s(x1,id=1)+s(x1,id=1)+s(x1,id=1)+s(x1,id=1)+s(x1,id=1)+s(x1,id=1)+s(x1,id=1)+s(x1,id=1)+s(x1,id=1)+s(x1,id=1)+s(x1,id=1)+s(x1,id=1)+s(x1,id=1)+s(x1,id=1)+s(x1,id=1)+s(x1,id=1)+s(x1,id=1)+s(x1,id=1)+s(x1,id=1)+s(x1,id=1)+s(x1,id=1)+s(x1,id=1)+s(x1,id=1)+s(x1,id=1)+s(x1,id=1)+s(x1,id=1)+s(x1,id=1)+s(x1,id=1)+s(x1,id=1)+s(x1,id=1)+s(x1,id=1)+s(x1,id=1)+s(x1,id=1)+s(x1,id=1)+s(x1,id=1)+s(x1,id=1)+s(x1,id=1)+s(x1,id=1)+s(x1,id=1)+s(x1,id=1)+s(x1,id=1)+s(x1,id=1)+s(x1,id=1)+s(x1,id=1)+s(x1,id=1)+s(x1,id=1)+s(x1,id=1)+s(x1,id=1)+s(x1,id=1)+s(x1,id=1)+s(x1,id=1)+s(x1,id=1)+s(x1,id=1)+s(x1,id=1)+s(x1,id=1)+s(x1,id=1)+s(x1,id=1)+s(x1,id=1)+s(x1,id=1)+s(x1,id=1)+s(x1,id=1)+s(x1,id=1)+s(x1,id=1)+s(x1,id=1)+s(x1,id=1)+s(x1,id=1)+s(x1,id=1)+s(x1,id=1)+s(x1,id=1)+s(x1,id=1)+s(x1,id=1)+s(x1,id=1)+s(x1,id=1)+s(x1,id=1)+s(x1,id=1)+s(x1,id=1)+s(x1,id=1)+s(x1,id=1)+s(x1,id=1)+s(x1,id=1)+s(x1,id=1)+s(x1,id=1)+s(x1,id=1)+s(x1,id=1)+s(x1,id=1)+s(x1,id=1)+s(x1,id=1)+s(x1,id=1)+s(x1,id=1)+s(x1,id=1)+s(x1,id=1)+s(x1,id=1)+s(x1,id=1)+s(x1,id=1)+s(x1,id=1)+s(x1,id=1)+s(x1,id=1)+s(x1,id=1)+s(x1,id=1)+s(x1,id=1)+s(x1,id=1)+s(x1,id=1)+s(x1,id=1)+s(x1,id=1)+s(x1,id=1)+s(x1,id=1)+s(x1,id=1)+s(x1,id=1)+s(x1,id=1)+s(x1,id=1)+s(x1,id=1)+s(x1,id=1)+s(x1,id=1)+s(x1,id=1)+s(x1,id=1)+s(x1,id=1)+s(x1,id=1)+s(x1,id=1)+s(x1,id=1)+s(x1,id=1)+s(x1,id=1)+s(x1,id=1)+s(x1,id=1)+s(x1,id=1)+s(x1,id=1)+s(x1,id=1)+s(x1,id=1)+s(x1,id=1)+s(x1,id=1)+s(x1,id=1)+s(x1,id=1)+s(x1,id=1)+s(x1,id=1)+s(x1,id=1)+s(x1,id=1)+s(x1,id=1)+s(x1,id=1)+s(x1,id=1
                                                       s(x2,k=15)+s(x3),data=dat,method="ML")
gam.vcomp(mod)
```

gam2objective

Objective functions for GAM smoothing parameter estimation

# **Description**

Estimation of GAM smoothing parameters is most stable if optimization of the UBRE/AIC or GCV score is outer to the penalized iteratively re-weighted least squares scheme used to estimate the model given smoothing parameters. These functions evaluate the GCV/UBRE/AIC score of a GAM model, given smoothing parameters, in a manner suitable for use by optim or nlm. Not normally called directly, but rather service routines for gam.outer.

gam2objective 73

### Usage

```
gam2objective(lsp,args,...)
gam2derivative(lsp,args,...)
```

# **Arguments**

lsp	The log smoothing parameters.
-----	-------------------------------

args List of arguments required to call gam. fit3.

... Other arguments for passing to gam. fit3.

#### **Details**

gam2objective and gam2derivative are functions suitable for calling by optim, to evaluate the GCV/UBRE/AIC score and its derivatives w.r.t. log smoothing parameters.

gam4objective is an equivalent to gam2objective, suitable for optimization by nlm - derivatives of the GCV/UBRE/AIC function are calculated and returned as attributes.

The basic idea of optimizing smoothing parameters 'outer' to the P-IRLS loop was first proposed in O'Sullivan et al. (1986).

### Author(s)

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

# References

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

O 'Sullivan, Yandall & Raynor (1986) Automatic smoothing of regression functions in generalized linear models. J. Amer. Statist. Assoc. 81:96-103.

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

```
http://www.maths.bris.ac.uk/~sw15190/
```

## See Also

```
gam.fit3, gam, magic
```

gamm

Generalized Additive Mixed Models

## **Description**

Fits the specified generalized additive mixed model (GAMM) to data, by a call to 1me in the normal errors identity link case, or by a call to gammPQL (a modification of g1mmPQL from the MASS library) otherwise. In the latter case estimates are only approximately MLEs. The routine is typically slower than gam, and not quite as numerically robust.

To use lme4 in place of nlme as the underlying fitting engine, see gamm4 from package gamm4.

Smooths are specified as in a call to gam as part of the fixed effects model formula, but the wiggly components of the smooth are treated as random effects. The random effects structures and correlation structures availabale for lme are used to specify other random effects and correlations.

It is assumed that the random effects and correlation structures are employed primarily to model residual correlation in the data and that the prime interest is in inference about the terms in the fixed effects model formula including the smooths. For this reason the routine calculates a posterior covariance matrix for the coefficients of all the terms in the fixed effects formula, including the smooths.

To use this function effectively it helps to be quite familiar with the use of gam and lme.

#### Usage

```
gamm(formula,random=NULL,correlation=NULL,family=gaussian(),
data=list(),weights=NULL,subset=NULL,na.action,knots=NULL,
control=list(niterEM=0,optimMethod="L-BFGS-B"),
niterPQL=20,verbosePQL=TRUE,method="ML",drop.unused.levels=TRUE,...)
```

### **Arguments**

formula	A GAM formula (see also formula.gam and gam.models). This is like the formula for a glm except that smooth terms (s and te) can be added to the right hand side of the formula. Note that ids for smooths and fixed smoothing parameters are not supported.
random	The (optional) random effects structure as specified in a call to lme: only the list form is allowed, to facilitate manipulation of the random effects structure within gamm in order to deal with smooth terms. See example below.
correlation	An optional corStruct object (see corClasses) as used to define correlation structures in lme. Any grouping factors in the formula for this object are assumed to be nested within any random effect grouping factors, without the need to make this explicit in the formula (this is slightly different to the behaviour of lme). This is a GEE approach to correlation in the generalized case. See examples below.
family	A family as used in a call to glm or gam. The default gaussian with identity link causes gamm to fit by a direct call to lme procided there is no offset term,

otherwise gammPQL is used.

data A data frame or list containing the model response variable and covariates re-

quired by the formula. By default the variables are taken from environment (formula),

typically the environment from which gamm is called.

weights In the generalized case, weights with the same meaning as glm weights. An lme

type weights argument may only be used in the identity link gaussian case, with no offset (see documentation for 1me for details of how to use such an argument).

subset an optional vector specifying a subset of observations to be used in the fitting

process.

na.action a function which indicates what should happen when the data contain 'NA's.

The default is set by the 'na.action' setting of 'options', and is 'na.fail' if that is

unset. The "factory-fresh" default is 'na.omit'.

knots this is an optional list containing user specified knot values to be used for basis

construction. Different terms can use different numbers of knots, unless they

share a covariate.

control A list of fit control parameters for lme to replace the defaults returned by lmeControl.

Note the setting for the number of EM iterations used by lme: smooths are set up using custom pdMat classes, which are currently not supported by the EM iteration code. If you supply a list of control values, it is advisable to include niterEM=0, as well, and only increase from 0 if you want to perturb the starting values used in model fitting (usually to worse values!). The optimMethod option is only used if your version of R does not have the nlminb optimizer function.

niterPQL Maximum number of PQL iterations (if any).

verbosePQL Should PQL report its progress as it goes along?

method Which of "ML" or "REML" to use in the Gaussian additive mixed model case

when 1me is called directly. Ignored in the generalized case (or if the model has

an offset), in which case gammPQL is used.

drop.unused.levels

by default unused levels are dropped from factors before fitting. For some smooths involving factor variables you might want to turn this off. Only do

so if you know what you are doing.

... further arguments for passing on e.g. to lme

# Details

The Bayesian model of spline smoothing introduced by Wahba (1983) and Silverman (1985) opens up the possibility of estimating the degree of smoothness of terms in a generalized additive model as variances of the wiggly components of the smooth terms treated as random effects. Several authors have recognised this (see Wang 1998; Ruppert, Wand and Carroll, 2003) and in the normal errors, identity link case estimation can be performed using general linear mixed effects modelling software such as 1me. In the generalized case only approximate inference is so far available, for example using the Penalized Quasi-Likelihood approach of Breslow and Clayton (1993) as implemented in g1mmPQL by Venables and Ripley (2002). One advantage of this approach is that it allows correlated errors to be dealt with via random effects or the correlation structures available in the n1me library (using correlation structures beyond the strictly additive case amounts to using a GEE approach to fitting).

Some details of how GAMs are represented as mixed models and estimated using 1me or gammPQL in gamm can be found in Wood (2004,2006a,b). In addition gamm obtains a posterior covariance matrix for the parameters of all the fixed effects and the smooth terms. The approach is similar to that described in Lin & Zhang (1999) - the covariance matrix of the data (or pseudodata in the generalized case) implied by the weights, correlation and random effects structure is obtained, based on the estimates of the parameters of these terms and this is used to obtain the posterior covariance matrix of the fixed and smooth effects.

The bases used to represent smooth terms are the same as those used in gam, although adaptive smoothing bases are not available. Prediction from the returned gam object is straightforward using predict. gam, but this will set the random effects to zero. If you want to predict with random effects set to their predicted values then you can adapt the prediction code given in the examples below.

In the event of 1me convergence failures, consider modifying option(mgcv.vc.logrange): reducing it helps to remove indefiniteness in the likelihood, if that is the problem, but too large a reduction can force over or undersmoothing. See notExp2 for more information on this option. Failing that, you can try increasing the niterEM option in control: this will perturb the starting values used in fitting, but usually to values with lower likelihood! Note that this version of gamm works best with R 2.2.0 or above and nlme, 3.1-62 and above, since these use an improved optimizer.

#### Value

Returns a list with two items:

gam an object of class gam, less information relating to GCV/UBRE model selection.

At present this contains enough information to use predict, summary and print methods and vis.gam, but not to use e.g. the anova method function to compare

models.

1me the fitted model object returned by 1me or gammPQL. Note that the model for-

mulae and grouping structures may appear to be rather bizarre, because of the manner in which the GAMM is split up and the calls to 1me and gammPQL are

constructed.

#### WARNINGS

gamm was a somewhat different argument list to gam, gam arguments such as gamma supplied to gamm will just be ignored.

gamm performs poorly with binary data, since it uses PQL. It is better to use gam with s(...,bs="re") terms, or gamm4.

gamm assumes that you know what you are doing! For example, unlike glmmPQL from MASS it will return the complete lme object from the working model at convergence of the PQL iteration, including the 'log likelihood', even though this is not the likelihood of the fitted GAMM.

The routine will be very slow and memory intensive if correlation structures are used for the very large groups of data. e.g. attempting to run the spatial example in the examples section with many 1000's of data is definitely not recommended: often the correlations should only apply within clusters that can be defined by a grouping factor, and provided these clusters do not get too huge then fitting is usually possible.

Models must contain at least one random effect: either a smooth with non-zero smoothing parameter, or a random effect specified in argument random.

gamm is not as numerically stable as gam: an 1me call will occasionally fail. See details section for suggestions, or try the 'gamm4' package.

gamm is usually much slower than gam, and on some platforms you may need to increase the memory available to R in order to use it with large data sets (see mem.limits).

Note that the weights returned in the fitted GAM object are dummy, and not those used by the PQL iteration: this makes partial residual plots look odd.

Note that the gam object part of the returned object is not complete in the sense of having all the elements defined in gamObject and does not inherit from glm: hence e.g. multi-model anova calls will not work.

The parameterization used for the smoothing parameters in gamm, bounds them above and below by an effective infinity and effective zero. See notExp2 for details of how to change this.

Linked smoothing parameters and adaptive smoothing are not supported.

#### Author(s)

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

#### References

Breslow, N. E. and Clayton, D. G. (1993) Approximate inference in generalized linear mixed models. Journal of the American Statistical Association 88, 9-25.

Lin, X and Zhang, D. (1999) Inference in generalized additive mixed models by using smoothing splines. JRSSB. 55(2):381-400

Pinheiro J.C. and Bates, D.M. (2000) Mixed effects Models in S and S-PLUS. Springer

Ruppert, D., Wand, M.P. and Carroll, R.J. (2003) Semiparametric Regression. Cambridge

Silverman, B.W. (1985) Some aspects of the spline smoothing approach to nonparametric regression. JRSSB 47:1-52

Venables, W. N. and Ripley, B. D. (2002) Modern Applied Statistics with S. Fourth edition. Springer. Wahba, G. (1983) Bayesian confidence intervals for the cross validated smoothing spline. JRSSB 45:133-150

Wood, S.N. (2004) Stable and efficient multiple smoothing parameter estimation for generalized additive models. Journal of the American Statistical Association. 99:673-686

Wood, S.N. (2003) Thin plate regression splines. J.R.Statist.Soc.B 65(1):95-114

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

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

Wang, Y. (1998) Mixed effects smoothing spline analysis of variance. J.R. Statist. Soc. B 60, 159-174

http://www.maths.bris.ac.uk/~sw15190/

#### See Also

magic for an alternative for correlated data, te, s, predict.gam, plot.gam, summary.gam, negbin, vis.gam,pdTens, gamm4 (http://cran.r-project.org/package=gamm4)

### **Examples**

```
library(mgcv)
## simple examples using gamm as alternative to gam
set.seed(0)
dat <- gamSim(1,n=200,scale=2)</pre>
b <- gamm(y \sim s(x0) + s(x1) + s(x2) + s(x3), data = dat)
plot(b$gam,pages=1)
summary(b$lme) # details of underlying lme fit
summary(b$gam) # gam style summary of fitted model
anova(b$gam)
gam.check(b$gam) # simple checking plots
b <- gamm(y \sim te(x0,x1) + s(x2) + s(x3), data = dat)
op \leftarrow par(mfrow=c(2,2))
plot(b$gam)
par(op)
rm(dat)
## Add a factor to the linear predictor, to be modelled as random
dat <- gamSim(6,n=200,scale=.2,dist="poisson")</pre>
b2 < -gamm(y \sim s(x0) + s(x1) + s(x2), family = poisson,
         data=dat,random=list(fac=~1))
plot(b2$gam,pages=1)
fac <- dat$fac
rm(dat)
vis.gam(b2$gam)
## now an example with autocorrelated errors....
n <- 200; sig <- 2
x < 0:(n-1)/(n-1)
f <- 0.2*x^11*(10*(1-x))^6+10*(10*x)^3*(1-x)^10
e <- rnorm(n,0,sig)
for (i in 2:n) e[i] \leftarrow 0.6*e[i-1] + e[i]
y \leftarrow f + e
op <- par(mfrow=c(2,2))</pre>
## Fit model with AR1 residuals
b \leftarrow gamm(y\sim s(x,k=20), correlation=corAR1())
plot(b$gam);lines(x,f-mean(f),col=2)
## Raw residuals still show correlation, of course...
acf(residuals(b$gam),main="raw residual ACF")
## But standardized are now fine...
acf(residuals(b$lme,type="normalized"),main="standardized residual ACF")
## compare with model without AR component...
b <- gam(y \sim s(x, k=20))
plot(b);lines(x,f-mean(f),col=2)
## more complicated autocorrelation example - AR errors
## only within groups defined by `fac'
e <- rnorm(n,0,sig)
for (i in 2:n) e[i] \leftarrow 0.6*e[i-1]*(fac[i-1]==fac[i]) + e[i]
y \leftarrow f + e
```

```
b <- gamm(y~s(x,k=20),correlation=corAR1(form=~1|fac))</pre>
plot(b$gam);lines(x,f-mean(f),col=2)
par(op)
## more complex situation with nested random effects and within
## group correlation
set.seed(0)
n.g <- 10
n<-n.g*10*4
## simulate smooth part...
dat <- gamSim(1,n=n,scale=2)</pre>
f <- dat$f
## simulate nested random effects....
fa <- as.factor(rep(1:10,rep(4*n.g,10)))</pre>
ra <- rep(rnorm(10),rep(4*n.g,10))
fb <- as.factor(rep(rep(1:4,rep(n.g,4)),10))</pre>
rb <- rep(rnorm(4),rep(n.g,4))</pre>
for (i in 1:9) rb \leftarrow c(rb, rep(rnorm(4), rep(n.g, 4)))
## simulate auto-correlated errors within groups
e<-array(0,0)
for (i in 1:40) {
  eg <- rnorm(n.g, 0, sig)
  for (j in 2:n.g) eg[j] \leftarrow eg[j-1]*0.6+ eg[j]
  e < -c(e, eg)
dat$y <- f + ra + rb + e
dat$fa <- fa;dat$fb <- fb
## fit model ....
b \leftarrow gamm(y^s(x0,bs="cr")+s(x1,bs="cr")+s(x2,bs="cr")+
  s(x3,bs="cr"),data=dat,random=list(fa=~1,fb=~1),
  correlation=corAR1())
plot(b$gam,pages=1)
summary(b$gam)
vis.gam(b$gam)
## Prediction from gam object, optionally adding
## in random effects.
## Extract random effects and make names more convenient...
refa <- ranef(b$lme,level=5)</pre>
rownames(refa) <- substr(rownames(refa), start=9, stop=20)</pre>
refb <- ranef(b$lme,level=6)</pre>
rownames(refb) <- substr(rownames(refb), start=9, stop=20)</pre>
## make a prediction, with random effects zero...
p0 <- predict(b$gam,data.frame(x0=.3,x1=.6,x2=.98,x3=.77))</pre>
## add in effect for fa = "2" and fb="2/4"...
p <- p0 + refa["2",1] + refb["2/4",1]</pre>
## and a "spatial" example...
library(nlme);set.seed(1);n <- 100</pre>
```

```
dat <- gamSim(2,n=n,scale=0) ## standard example</pre>
attach(dat)
old.par<-par(mfrow=c(2,2))</pre>
contour(truth$x,truth$z,truth$f) ## true function
f <- data$f
                                    ## true expected response
## Now simulate correlated errors...
cstr <- corGaus(.1, form = ~x+z)
cstr <- Initialize(cstr,data.frame(x=data$x,z=data$z))</pre>
V <- corMatrix(cstr) ## correlation matrix for data</pre>
Cv <- chol(V)
e <- t(Cv) %*% rnorm(n)*0.05 # correlated errors
## next add correlated simulated errors to expected values
data$y <- f + e ## ... to produce response
b<- gamm(y*s(x,z,k=50),correlation=corGaus(.1,form=*x+z),
         data=data)
plot(b$gam) # gamm fit accounting for correlation
# overfits when correlation ignored.....
b1 <- gamm(y \sim s(x,z,k=50), data=data); plot(b1\$gam)
b2 \leftarrow gam(y^s(x,z,k=50),data=data);plot(b2)
par(old.par)
```

gamObject

Fitted gam object

# **Description**

A fitted GAM object returned by function gam and of class "gam" inheriting from classes "glm" and "lm". Method functions anova, logLik, influence, plot, predict, print, residuals and summary exist for this class.

All compulsory elements of "glm" and "lm" objects are present, but the fitting method for a GAM is different to a linear model or GLM, so that the elements relating to the QR decomposition of the model matrix are absent.

# Value

A gam object has the following elements:

aic	AIC of the fitted model: bear in mind that 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 in most cases, 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.
boundary	did parameters end up at boundary of parameter space?
call	the matched call (allows update to be used with gam objects, for example).
cmX	column means of the model matrix (with elements corresponding to smooths set to zero ) — useful for componentwise CI calculation.

coefficients the coefficients of the fitted model. Parametric coefficients are first, followed by

coefficients for each spline term in turn.

control the gam control list used in the fit.

converged indicates whether or not the iterative fitting method converged.

data the original supplied data argument (for class "glm" compatibility). Only in-

cluded if gam control argument element keepData is set to TRUE (default is

FALSE).

db. drho matrix of first derivatives of model coefficients w.r.t. log smoothing parameters.

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 similar, but using alternative estimate of EDF. Useful for testing.

edf2 if estimation is by ML or REML then an edf that accounts for smoothing param-

eter uncertainty can be computed, this is it. edf1 is a heuristic upper bound for

edf2.

family bject specifying distribution and link used.

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

formula the model formula.

full.sp full array of smoothing parameters multiplying penalties (excluding any contri-

bution from min. sp argument to gam). May be larger than sp if some terms share smoothing parameters, and/or some smoothing parameter values were supplied

in the sp argument of gam.

F Degrees of freedom matrix. This may be removed at some point, and should

probably not be used.

gcv.ubre The minimized smoothing parameter selection score: GCV, UBRE(AIC), GACV,

negative log marginal likelihood or negative log restricted likelihood.

hat array of elements from the leading diagonal of the 'hat' (or 'influence') matrix.

Same length as response data vector.

iter number of iterations of P-IRLS taken to get convergence.

linear.predictors

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

method One of "GCV" or "UBRE", "REML", "P-REML", "ML", "P-ML", "PQL", "lme.ML"

or "Ime.REML", depending on the fitting criterion used.

mgcv.conv A list of convergence diagnostics relating to the "magic" parts of smoothing pa-

rameter estimation - this will not be very meaningful for pure "outer" estimation of smoothing parameters. The items are: full.rank, The apparent rank of the problem given the model matrix and constraints; rank, The numerical rank of the problem; fully.converged, TRUE is multiple GCV/UBRE converged by meeting convergence criteria and FALSE if method stopped with a steepest descent step failure; hess.pos.defWas the hessian of the GCV/UBRE score positive definite at smoothing parameter estimation convergence?; iter How many

iterations were required to find the smoothing parameters? score.calls, and how many times did the GCV/UBRE score have to be evaluated?; rms.grad, root mean square of the gradient of the GCV/UBRE score at convergence.

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

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

na.action The na.action used in fitting.

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

null.deviance deviance for single parameter model.

offset model offset.

optimizer optimizer argument to gam, or "magic" if it's a pure additive model.

outer.info If 'outer' iteration has been used to fit the model (see gam argument optimizer)

then this is present and contains whatever was returned by the optimization rou-

tine used (currently nlm or optim).

paraPen If the paraPen argument to gam was used then this provides information on the

parametric penalties. NULL otherwise.

pred.formula one sided formula containing variables needed for prediction, used by predict.gam

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 (so need not be upper triangular).

rank apparent rank of fitted model.

reml.scale The scale (RE)ML scale parameter estimate, if (P-)(RE)ML used for smoothness

estimation.

residuals the working residuals for the fitted model.

rV If present, rV%\*%t(rV)\*sig2 gives the estimated Bayesian covariance matrix.

scale when present, the scale (as sig2)

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

what gets returned by the smooth. construct objects.

sp estimated smoothing parameters for the model. These are the underlying smooth-

ing parameters, subject to optimization. For the full set of smoothing parameters multiplying the penalties see full.sp. Divide the scale parameter by the smoothing parameters to get, variance components, but note that this is not valid

for smooths that have used rescaling to improve conditioning.

terms object of model model frame.

var.summary	A named list of summary information on the predictor variables. If a parametric variable is a matrix, then the summary is a one row matrix, containing the observed data value closest to the column median, for each matrix column. If the variable is a factor the then summary is the modal factor level, returned as a factor, with levels corresponding to those of the data. For numerics and matrix arguments of smooths, the summary is the mean, nearest observed value to median and maximum, as a numeric vector. Used by vis.gam, in particular.
Ve	frequentist estimated covariance matrix for the parameter estimators. Particularly useful for testing whether terms are zero. Not so useful for CI's as smooths are usually biased.
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. Paricularly useful for creating credible/confidence intervals.
Vc	Under ML or REML smoothing parameter estimation it is possible to correct the covariance matrix Vp for smoothing parameter uncertainty. This is the corrected version.
weights	final weights used in IRLS iteration.
у	response data.

# WARNINGS

This model object is different to that described in Chambers and Hastie (1993) in order to allow smoothing parameter estimation etc.

# Author(s)

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

## References

A Key Reference on this implementation:

Wood, S.N. (2006) Generalized Additive Models: An Introduction with R. Chapman & Hall/CRC, Boca Raton, Florida

Key Reference on GAMs generally:

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

Hastie and Tibshirani (1990) Generalized Additive Models. Chapman and Hall.

# See Also

gam

84 gamSim

gamSim Simulate example data for GAMs	
---------------------------------------	--

#### **Description**

Function used to simulate data sets to illustrate the use of gam and gamm. Mostly used in help files to keep down the length of the example code sections.

## Usage

```
gamSim(eg=1,n=400,dist="normal",scale=2,verbose=TRUE)
```

# Arguments

		1 ' '	.1	1 . 1
eg	numeric va	lue specifying	the exam	nle reamred
<b>С</b> Б	mumicine vu	iuc specii y iiig	, uic chain	pre required.

n number of data to simulate.

dist character string which may be used to speify the distribution of the response.

scale Used to set noise level.

verbose Should information about simulation type be printed?

#### **Details**

See the source code for exactly what is simulated in each case.

- 1. Gu and Wahba 4 univariate term example.
- 2. A smooth function of 2 variables.
- 3. Example with continuous by variable.
- 4. Example with factor by variable.
- 5. An additive example plus a factor variable.
- 6. Additive + random effect.
- 7. As 1 but with correlated covariates.

#### Value

Depends on eg, but usually a dataframe, which may also contain some information on the underlying truth. Sometimes a list with more items, including a data frame for model fitting. See source code or helpfile examples where the function is used for further information.

# Author(s)

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

#### See Also

gam, gamm

gaulss 85

### **Examples**

## see ?gam

gaulss

Gaussian location-scale model family

# **Description**

The gaulss family implements Gaussian location scale additive models in which the mean and the inverse of the standard deviation can depend on additive smooth predictors. Useable only with gam, the linear predictors are specified via a list of formulae.

## Usage

```
gaulss(link=list("identity","logb"),b=0.01)
```

### **Arguments**

link two item list specifying the link for the mean and the standard deviation. See

details.

b The minumum standard deviation, for the "logb" link.

#### **Details**

Used with gam to fit Gaussian location - scale models. gam is called with a list containing 2 formulae, the first specifies the response on the left hand side and the structure of the linear predictor for the mean on the right hand side. The second is one sided, specifying the linear predictor for the standard deviation on the right hand side.

Link functions "identity", "inverse", "log" and "sqrt" are available for the mean. For the standard deviation only the "logb" link is implemented:  $\eta = \log(\sigma - b)$  and  $\sigma = b + \exp(\eta)$ . This link is designed to avoid singularities in the likelihood caused by the standard deviation tending to zero.

The fitted values for this family will be a two column matrix. The first column is the mean, and the second column is the inverse of the standard deviation. Predictions using predict.gam will also produce 2 column matrices for type "link" and "response".

The null deviance reported for this family is the sum of squares of the difference between the response and the mean response divided by the standard deviation of the response according to the model. The deviance is the sum of squares of residuals divided by model standard deviations.

# Value

An object inheriting from class general.family.

#### References

Wood, S.N., N. Pya and B. Saefken (2015), Smoothing parameter and model selection for general smooth models. http://arxiv.org/abs/1511.03864

86 get.var

### **Examples**

get.var

Get named variable or evaluate expression from list or data.frame

# Description

This routine takes a text string and a data frame or list. It first sees if the string is the name of a variable in the data frame/ list. If it is then the value of this variable is returned. Otherwise the routine tries to evaluate the expression within the data.frame/list (but nowhere else) and if successful returns the result. If neither step works then NULL is returned. The routine is useful for processing gam formulae. If the variable is a matrix then it is coerced to a numeric vector, by default.

# Usage

```
get.var(txt,data,vecMat=TRUE)
```

### **Arguments**

txt a text string which is either the name of a variable in data or when parsed is an

expression that can be evaluated in data. It can also be neither in which case

the function returns NULL.

data A data frame or list.

vecMat Should matrices be coerced to numeric vectors?

# Value

The evaluated variable or NULL. May be coerced to a numeric vector if it's a matrix.

# Author(s)

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

### References

```
http://www.maths.bris.ac.uk/~sw15190/
```

### See Also

gam

in.out 87

### **Examples**

```
require(mgcv)
y <- 1:4;dat<-data.frame(x=5:10)
get.var("x",dat)
get.var("y",dat)
get.var("x==6",dat)
dat <- list(X=matrix(1:6,3,2))
get.var("X",dat)</pre>
```

in.out

Which of a set of points lie within a polygon defined region

### **Description**

Tests whether each of a set of points lie within a region defined by one or more (possibly nested) polygons. Points count as 'inside' if they are interior to an odd number of polygons.

## Usage

```
in.out(bnd,x)
```

# Arguments

bno	d A	<b>₹</b> t	wo co	lumn	matrix	the,	rows of	whi	ich defin	e th	e vertices of	рo	lygons	defining	
		1	1	1	c		D: CC		1	1	111		1.1		

the boundary of a region. Different polygons should be separated by an NA row,

and the polygons are assumed closed.

x A two column matrix. Each row is a point to test for inclusion in the region

defined by bnd.

#### **Details**

The algorithm works by counting boundary crossings (using compiled C code).

# Value

A logical vector of length nrow(x). TRUE if the corresponding row of x is inside the boundary and FALSE otherwise.

# Author(s)

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

# References

```
http://www.maths.bris.ac.uk/~sw15190/
```

88 influence.gam

### **Examples**

```
library(mgcv)
data(columb.polys)
bnd <- columb.polys[[2]]
plot(bnd,type="n")
polygon(bnd)
x <- seq(7.9,8.7,length=20)
y <- seq(13.7,14.3,length=20)
gr <- as.matrix(expand.grid(x,y))
inside <- in.out(bnd,gr)
points(gr,col=as.numeric(inside)+1)</pre>
```

influence.gam

Extract the diagonal of the influence/hat matrix for a GAM

# Description

Extracts the leading diagonal of the influence matrix (hat matrix) of a fitted gam object.

# Usage

```
## S3 method for class 'gam'
influence(model,...)
```

# **Arguments**

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

# **Details**

Simply extracts hat array from fitted model. (More may follow!)

## Value

An array (see above).

### Author(s)

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

## See Also

gam

initial.sp 89

initial.sp

Starting values for multiple smoothing parameter estimation

### **Description**

Finds initial smoothing parameter guesses for multiple smoothing parameter estimation. The idea is to find values such that the estimated degrees of freedom per penalized parameter should be well away from 0 and 1 for each penalized parameter, thus ensuring that the values are in a region of parameter space where the smoothing parameter estimation criterion is varying substantially with smoothing parameter value.

# Usage

```
initial.sp(X,S,off,expensive=FALSE,XX=FALSE)
```

# **Arguments**

_	
Χ	is the model matrix.
S	is a list of of penalty matrices. S[[i]] is the ith penalty matrix, but note that it is not stored as a full matrix, but rather as the smallest square matrix including all the non-zero elements of the penalty matrix. Element 1,1 of S[[i]] occupies element off[i], off[i] of the ith penalty matrix. Each S[[i]] must be positive semi-definite.
off	is an array indicating the first parameter in the parameter vector that is penalized by the penalty involving $S[[i]]$ .
expensive	if TRUE then the overall amount of smoothing is adjusted so that the average degrees of freedom per penalized parameter is exactly 0.5: this is numerically costly.
XX	if TRUE then X contains $X^TX$ , rather than $X$ .

## **Details**

Basically uses a crude approximation to the estimated degrees of freedom per model coefficient, to try and find smoothing parameters which bound these e.d.f.'s away from 0 and 1.

Usually only called by magic and gam.

# Value

An array of initial smoothing parameter estimates.

# Author(s)

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

### See Also

```
magic, gam.outer, gam,
```

90 inSide

inSide	Are points inside boundary?	

# **Description**

Assesses whether points are inside a boundary. The boundary must enclose the domain, but may include islands.

# Usage

```
inSide(bnd,x,y)
```

## **Arguments**

bnd	This should have two equal length columns with names matching whatever is supplied in x and y. This may contain several sections of boundary separated by NA. Alternatively bnd may be a list, each element of which contains 2 columns named as above. See below for details.
x	x co-ordinates of points to be tested.
У	y co-ordinates of points to be tested.

#### **Details**

Segments of boundary are separated by NAs, or are in separate list elements. The boundary coordinates are taken to define nodes which are joined by straight line segments in order to create the boundary. Each segment is assumed to define a closed loop, and the last point in a segment will be assumed to be joined to the first. Loops must not intersect (no test is made for this).

The method used is to count how many times a line, in the y-direction from a point, crosses a boundary segment. An odd number of crossings defines an interior point. Hence in geographic applications it would be usual to have an outer boundary loop, possibly with some inner 'islands' completely enclosed in the outer loop.

The routine calls compiled C code and operates by an exhaustive search for each point in x, y.

# Value

The function returns a logical array of the same dimension as x and y. TRUE indicates that the corresponding x, y point lies inside the boundary.

## Author(s)

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

# References

```
http://www.maths.bris.ac.uk/~sw15190/
```

interpret.gam 91

### **Examples**

```
require(mgcv)
m <- 300;n <- 150
xm <- seq(-1,4,length=m);yn<-seq(-1,1,length=n)
x <- rep(xm,n);y<-rep(yn,rep(m,n))
er <- matrix(fs.test(x,y),m,n)
bnd <- fs.boundary()
in.bnd <- inSide(bnd,x,y)
plot(x,y,col=as.numeric(in.bnd)+1,pch=".")
lines(bnd$x,bnd$y,col=3)
points(x,y,col=as.numeric(in.bnd)+1,pch=".")
## check boundary details ...
plot(x,y,col=as.numeric(in.bnd)+1,pch=".",ylim=c(-1,0),xlim=c(3,3.5))
lines(bnd$x,bnd$y,col=3)
points(x,y,col=as.numeric(in.bnd)+1,pch=".")</pre>
```

interpret.gam

Interpret a GAM formula

#### Description

This is an internal function of package mgcv. It is a service routine for gam which splits off the strictly parametric part of the model formula, returning it as a formula, and interprets the smooth parts of the model formula.

Not normally called directly.

#### Usage

```
interpret.gam(gf)
```

# **Arguments**

gf

A GAM formula as supplied to gam or gamm, or a list of such formulae, as supplied for some gam families.

# Value

An object of class split.gam.formula with the following items:

pf A model formula for the strictly parametric part of the model.

pfok TRUE if there is a pf formula.

smooth.spec A list of class xx.smooth.spec objects where xx depends on the basis specified

for the term. (These can be passed to smooth constructor method functions to

actually set up penalties and bases.)

full. formula An expanded version of the model formula in which the options are fully ex-

panded, and the options do not depend on variables which might not be available

later.

fake. formula A formula suitable for use in evaluating a model frame.

response Name of the response variable.

#### Author(s)

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

#### References

```
http://www.maths.bris.ac.uk/~sw15190/
```

### See Also

gam gamm

jagam

Just Another Gibbs Additive Modeller: JAGS support for mgcv.

## Description

Facilities to auto-generate model specification code and associated data to simulate with GAMs in JAGS (or BUGS). This is useful for inference about models with complex random effects structure best coded in JAGS. It is a very innefficient approach to making inferences about standard GAMs. The idea is that jagam generates template JAGS code, and associated data, for the smooth part of the model. This template is then directly edited to include other stochastic components. After simulation with the resulting model, facilities are provided for plotting and prediction with the model smooth components.

# Usage

```
jagam(formula,family=gaussian,data=list(),file,weights=NULL,na.action,
offset=NULL,knots=NULL,sp=NULL,drop.unused.levels=TRUE,
control=gam.control(),centred=TRUE,sp.prior = "gamma",diagonalize=FALSE)
sim2jam(sam,pregam,edf.type=2,burnin=0)
```

#### **Arguments**

formula A GAM formula (see formula.gam and also gam.models). This is exactly like

the formula for a GLM except that smooth terms, s, te, ti and t2 can be added to the right hand side to specify that the linear predictor depends on smooth

functions of predictors (or linear functionals of these).

family This is a family object specifying the distribution and link function to use. See

glm and family for more details. Currently only gaussian, poisson, binomial and Gamma families are supported, but the user can easily modify the assumed

distribution in the JAGS code.

data A data frame or list containing the model response variable and covariates re-

quired by the formula. By default the variables are taken from environment (formula):

typically the environment from which jagam is called.

file Name of the file to which JAGS model specification code should be written. See

setwd for setting and querying the current working directory.

weights prior weights on the data.

na.action a function which indicates what should happen when the data contain 'NA's.

The default is set by the 'na.action' setting of 'options', and is 'na.fail' if that is

unset. The "factory-fresh" default is 'na.omit'.

offset Can be 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 and glm.

control A list of fit control parameters to replace defaults returned by gam.control.

Any control parameters not supplied stay at their default values. little effect on

jagam.

knots this is an optional list containing user specified knot values to be used for basis

construction. For most bases the user simply supplies the knots to be used, which must match up with the k value supplied (note that the number of knots is not always just k). See tprs for what happens in the "tp"/"ts" case. Different

terms can use different numbers of knots, unless they share a covariate.

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 (without forgetting null space penalties). Negative elements indicate that the parameter should be estimated, and hence a mixture of fixed and estimated parameters is possible. If smooths share smoothing parameters then length(sp)

must correspond to the number of underlying smoothing parameters.

drop.unused.levels

by default unused levels are dropped from factors before fitting. For some

smooths involving factor variables you might want to turn this off. Only do

so if you know what you are doing.

centred Should centring constraints be applied to the smooths, as is usual with GAMS?

Only set this to FALSE if you know exactly what you are doing. If FALSE there

is a (usually global) intercept for each smooth.

sp.prior "gamma" or "log.uniform" prior for the smoothing parameters? Do check that

the default parameters are appropriate for your model in the JAGS code.

diagonalize Should smooths be re-parameterized to have i.i.d. Gaussian priors (where pos-

sible)? For Gaussian data this allows efficient conjugate samplers to be used, and it can also work well with GLMs if the JAGS "glm" module is loaded, but otherwise it is often better to undeterment headswise, and not do this

otherwise it is often better to update smoothers blockwise, and not do this.

sam jags sample object, containing at least fields b (coefficients) and rho (log smooth-

ing parameters). May also contain field mu containing monitored expected re-

sponse.

pregam standard mgcv GAM setup data, as returned in jagam return list.

edf. type Since EDF is not uniquely defined and may be affected by the stochastic struc-

ture added to the JAGS model file, 3 options are offered. See details.

burnin

the amount of burn in to discard from the simulation chains. Limited to .9 of the chain length.

#### **Details**

Smooths are easily incorportated into JAGS models using multivariate normal priors on the smooth coefficients. The smoothing parameters and smoothing penalty matrices directly specify the prior multivariate normal precision matrix. Normally a smoothing penalty does not correspond to a full rank precision matrix, implying an improper prior inappropriate for Gibbs sampling. To rectify this problem the null space penalties suggested in Marra and Wood (2011) are added to the usual penalties.

In an additive modelling context it is usual to centre the smooths, to avoid the identifiability issues associated with having an intercept for each smooth term (in addition to a global intercept). Under Gibbs sampling with JAGS it is technically possible to omit this centring, since we anyway force propriety on the priors, and this propiety implies formal model identifiability. However, in most situations this formal identifiability is rather artificial and does not imply statistically meaningfull identifiability. Rather it serves only to massively inflate confidence intervals, since the multiple intercept terms are not identifiable from the data, but only from the prior. By default then, jagam imposes standard GAM identifiability constraints on all smooths. The centred argument does allow you to turn this off, but it is not recommended. If you do set centred=FALSE then chain convergence and mixing checks should be particularly stringent.

The final technical issue for model setup is the setting of initial conditions for the coefficients and smoothing parameters. The approach taken is to take the default initial smoothing parameter values used elsewhere by mgcv, and to take a single PIRLS fitting step with these smoothing parameters in order to obtain starting values for the smooth coefficients. In the setting of fully conjugate updating the initial values of the coefficients are not critical, and good results are obtained without supplying them. But in the usual setting in which slice sampling is required for at least some of the updates then very poor results can sometimes be obtained without initial values, as the sampler simply fails to find the region of the posterior mode.

The sim2jam function takes the partial gam object (pregam) from jagam along with simulation output in standard rjags form and creates a reduced version of a gam object, suitable for plotting and prediction of the model's smooth components. sim2gam computes effective degrees of freedom for each smooth, but it should be noted that there are several possibilities for doing this in the context of a model with a complex random effects structure. The simplest approach (edf.type=0) is to compute the degrees of freedom that the smooth would have had if it had been part of an unweighted Gaussian additive model. One might choose to use this option if the model has been modified so that the response distribution and/or link are not those that were specified to jagam. The second option is (edf. type=1) uses the edf that would have been computed by gam had it produced these estimates - in the context in which the JAGS model modifications have all been about modifying the random effects structure, this is equivalent to simply setting all the random effects to zero for the effective degrees of freedom calculation. The default option (edf.type=2) is to base the EDF on the sample covariance matrix, Vp, of the model coefficients. If the simulation output (sim) includes a mu field, then this will be used to form the weight matrix W in XWX = t(X)%\*%W%\*X, where the EDF is computed from rowSums(Vp\*XWX)\*scale. If mu is not supplied then it is estimated from the the model matrix X and the mean of the simulated coefficients, but the resulting W may not be strictly comaptible with the Vp matrix in this case. In the situation in which the fitted model is very different in structure from the regression model of the template produced by jagam then the default option may make no sense, and indeed it may be best to use option 0.

#### Value

For jagam a three item list containing

pregam standard mgcv GAM setup data.

jags.data list of arguments to be supplied to JAGS containing information referenced in

model specification.

jags.ini initialization data for smooth coefficients and smoothing parameters.

For sim2jam an object of class "jam": a partial version of an mgcv gam0bject, suitable for plotting and predicting.

#### WARNINGS

Gibb's sampling is a very slow inferential method for standard GAMs. It is only likely to be worthwhile when complex random effects structures are required above what is possible with direct GAMM methods.

Check that the parameters of the priors on the parameters are fit for your purpose.

### Author(s)

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

#### References

Wood, S.N. (in press) Just Another Gibbs Additive Modeller: Interfacing JAGS and mgcv. Journal of Statistical Software

Marra, G. and S.N. Wood (2011) Practical variable selection for generalized additive models. Computational Statistics & Data Analysis 55(7): 2372-2387

Here is a key early reference to smoothing using BUGS (although the approach and smooths used are a bit different to jagam)

Crainiceanu, C. M. D Ruppert, & M.P. Wand (2005) Bayesian Analysis for Penalized Spline Regression Using WinBUGS Journal of Statistical Software 14.

### See Also

```
gam, gamm, bam
```

### **Examples**

```
## the following illustrates a typical workflow. To run the
## 'Not run' code you need rjags (and JAGS) to be installed.
require(mgcv)

set.seed(2) ## simulate some data...
n <- 400
dat <- gamSim(1,n=n,dist="normal",scale=2)
## regular gam fit for comparison...
b0 <- gam(y~s(x0)+s(x1) + s(x2)+s(x3),data=dat,method="REML")</pre>
```

```
## Set up JAGS code and data. In this one might want to diagonalize
## to use conjugate samplers. Usually call 'setwd' first, to set
## directory in which model file ("test.jags") will be written.
jd \leftarrow jagam(y\sim s(x0)+s(x1)+s(x2)+s(x3), data=dat, file="test.jags",
            sp.prior="gamma",diagonalize=TRUE)
## In normal use the model in "test.jags" would now be edited to add
## the non-standard stochastic elements that require use of JAGS....
## Not run:
require(rjags)
load.module("glm") ## improved samplers for GLMs often worth loading
jm <-jags.model("test.jags",data=jd$jags.data,inits=jd$jags.ini,n.chains=1)</pre>
list.samplers(jm)
sam <- jags.samples(jm,c("b","rho","scale"),n.iter=10000,thin=10)</pre>
jam <- sim2jam(sam,jd$pregam)</pre>
plot(jam,pages=1)
pd <- data.frame(x0=c(.5,.6),x1=c(.4,.2),x2=c(.8,.4),x3=c(.1,.1))
fv <- predict(jam,newdata=pd)</pre>
## and some minimal checking...
require(coda)
effectiveSize(as.mcmc.list(sam$b))
## End(Not run)
## a gamma example...
set.seed(1); n <- 400
dat <- gamSim(1,n=n,dist="normal",scale=2)</pre>
scale <- .5; Ey <- exp(datf/2)
dat$y <- rgamma(n,shape=1/scale,scale=Ey*scale)</pre>
jd \leftarrow jagam(y\sim s(x0)+te(x1,x2)+s(x3),data=dat,family=Gamma(link=log),
            file="test.jags",sp.prior="log.uniform")
## In normal use the model in "test.jags" would now be edited to add
## the non-standard stochastic elements that require use of JAGS....
## Not run:
require(rjags)
## following sets random seed, but note that under JAGS 3.4 many
## models are still not fully repeatable (JAGS 4 should fix this)
jd$jags.ini$.RNG.name <- "base::Mersenne-Twister" ## setting RNG</pre>
jd$jags.ini$.RNG.seed <- 6 ## how to set RNG seed
jm <-jags.model("test.jags",data=jd$jags.data,inits=jd$jags.ini,n.chains=1)</pre>
list.samplers(jm)
sam <- jags.samples(jm,c("b","rho","scale","mu"),n.iter=10000,thin=10)</pre>
jam <- sim2jam(sam,jd$pregam)</pre>
plot(jam,pages=1)
pd <- data.frame(x0=c(.5,.6),x1=c(.4,.2),x2=c(.8,.4),x3=c(.1,.1))
fv <- predict(jam,newdata=pd)</pre>
## End(Not run)
```

IdTweedie 97

ldTweedie	Log Tweedie density evaluation	
-----------	--------------------------------	--

# **Description**

A function to evaluate the log of the Tweedie density for variance powers between 1 and 2, inclusive. Also evaluates first and second derivatives of log density w.r.t. its scale parameter, phi, and p, or w.r.t. rho=log(phi) and theta where p = (a+b\*exp(theta))/(1+exp(theta)).

## Usage

ldTweedie(y,mu=y,p=1.5,phi=1,rho=NA,theta=NA,a=1.001,b=1.999)

## **Arguments**

У	values at which to evaluate density.
mu	corresponding means (either of same length as y or a single value).
p	the variance of y is proportional to its mean to the power p. p must be between 1 and 2. 1 is Poisson like (exactly Poisson if phi=1), 2 is gamma.
phi	The scale parameter. Variance of y is phi*mu^p.
rho	optional log scale parameter. Over-rides phi if theta also supplied.
theta	parameter such that $p = (a+b*exp(theta))/(1+exp(theta))$ . Over-rides p if rho also supplied.
а	lower limit parameter used in definition of p from theta.
b	upper limit parameter used in definition of p from theta.

## Details

A Tweedie random variable with 1<p<2 is a sum of N gamma random variables where N has a Poisson distribution. The p=1 case is a generalization of a Poisson distribution and is a discrete distribution supported on integer multiples of the scale parameter. For 1<p<2 the distribution is supported on the positive reals with a point mass at zero. p=2 is a gamma distribution. As p gets very close to 1 the continuous distribution begins to converge on the discretely supported limit at p=1.

ldTweedie is based on the series evaluation method of Dunn and Smyth (2005). Without the restriction on p the calculation of Tweedie densities is less straightforward. If you really need this case then the tweedie package is the place to start.

The rho, theta parameterization is useful for optimization of p and phi, in order to keep p bounded well away from 1 and 2, and phi positive. The derivatives near p=1 tend to infinity.

#### Value

A matrix with 6 columns. The first is the log density of y (log probability if p=1). The second and third are the first and second derivatives of the log density w.r.t. phi. 4th and 5th columns are first and second derivative w.r.t. p, final column is second derivative w.r.t. phi and p.

If rho and theta were supplied then derivatives are w.r.t. these.

### Author(s)

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

#### References

Dunn, P.K. and G.K. Smith (2005) Series evaluation of Tweedie exponential dispersion model densities. Statistics and Computing 15:267-280

Tweedie, M. C. K. (1984). An index which distinguishes between some important exponential families. Statistics: Applications and New Directions. Proceedings of the Indian Statistical Institute Golden Jubilee International Conference (Eds. J. K. Ghosh and J. Roy), pp. 579-604. Calcutta: Indian Statistical Institute.

# **Examples**

```
library(mgcv)
## convergence to Poisson illustrated
## notice how p>1.1 is OK
y <- seq(1e-10,10,length=1000)
p <- c(1.0001,1.001,1.01,1.1,1.2,1.5,1.8,2)
phi <- .5
fy <- exp(ldTweedie(y,mu=2,p=p[1],phi=phi)[,1])
plot(y,fy,type="1",ylim=c(0,3),main="Tweedie density as p changes")
for (i in 2:length(p)) {
   fy <- exp(ldTweedie(y,mu=2,p=p[i],phi=phi)[,1])
   lines(y,fy,col=i)
}</pre>
```

linear.functional.terms

Linear functionals of a smooth in GAMs

### **Description**

gam allows the response variable to depend on linear functionals of smooth terms. Specifically dependancies of the form

$$g(\mu_i) = \ldots + \sum_j L_{ij} f(x_{ij}) + \ldots$$

are allowed, where the  $x_{ij}$  are covariate values and the  $L_{ij}$  are fixed weights. i.e. the response can depend on the weighted sum of the same smooth evaluated at different covariate values. This allows, for example, for the response to depend on the derivatives or integrals of a smooth (approximated by finite differencing or quadrature, respectively). It also allows dependence on predictor functions (sometimes called 'signal regression').

The mechanism by which this is achieved is to supply matrices of covariate values to the model smooth terms specified by s or te terms in the model formula. Each column of the covariate matrix gives rise to a corresponding column of predictions from the smooth. Let the resulting matrix of evaluated smooth values be F (F will have the same dimension as the covariate matrices). In the absense of a by variable then these columns are simply summed and added to the linear predictor. i.e. the contribution of the term to the linear predictor is rowSums(F). If a by variable is present then it must be a matrix, L,say, of the same dimension as F (and the covariate matrices), and it contains the weights  $L_{ij}$  in the summation given above. So in this case the contribution to the linear predictor is rowSums(L\*F).

Note that if a L1 (i.e. rowSums(L)) is a constant vector, or there is no by variable then the smooth will automatically be centred in order to ensure identifiability. Otherwise it will not be. Note also that for centred smooths it can be worth replacing the constant term in the model with rowSums(L) in order to ensure that predictions are automatically on the right scale.

When predicting from the model it is not necessary to provide matrix covariate and by variable values. For example to simply examine the underlying smooth function one would use vectors of covariate values and vector by variables, with the by variable and equivalent of L1, above, set to vectors of ones.

#### Author(s)

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

### **Examples**

```
### matrix argument `linear operator' smoothing
library(mgcv)
set.seed(0)
####################################
## simple summation example...#
n<-400
sig<-2
x \leftarrow runif(n, 0, .9)
f2 \leftarrow function(x) \ 0.2*x^11*(10*(1-x))^6+10*(10*x)^3*(1-x)^10
x1 < -x + .1
f \leftarrow f2(x) + f2(x1) ## response is sum of f at two adjacent x values
y \leftarrow f + rnorm(n)*sig
X \leftarrow matrix(c(x,x1),n,2) ## matrix covariate contains both x values
b \leftarrow gam(y\sim s(X))
plot(b) ## reconstruction of f
```

```
plot(f,fitted(b))
## multivariate integral example. Function `test1' will be integrated#
## (by midpoint quadrature) over 100 equal area sub-squares covering #
## the unit square. Noise is added to the resulting simulated data. #
## `test1' is estimated from the resulting data using two alternative#
test1 <- function(x,z,sx=0.3,sz=0.4)
 { (pi**sx*sz)*(1.2*exp(-(x-0.2)^2/sx^2-(z-0.3)^2/sz^2)+
    0.8*exp(-(x-0.7)^2/sx^2-(z-0.8)^2/sz^2))
## create quadrature (integration) grid, in useful order
ig <- 5 ## integration grid within square
mx <- mz <- (1:ig-.5)/ig
ix <- rep(mx,ig);iz <- rep(mz,rep(ig,ig))</pre>
og <- 10 ## observarion grid
mx <- mz <- (1:og-1)/og
ox <- rep(mx,og);ox <- rep(ox,rep(ig^2,og^2))
oz <- rep(mz,rep(og,og));oz <- rep(oz,rep(ig^2,og^2))</pre>
x \leftarrow ox + ix/og; z \leftarrow oz + iz/og ## full grid, subsquare by subsquare
## create matrix covariates...
X <- matrix(x,og^2,ig^2,byrow=TRUE)</pre>
Z <- matrix(z,og^2,ig^2,byrow=TRUE)</pre>
## create simulated test data...
dA <- 1/(og*ig)^2 ## quadrature square area
F <- test1(X,Z) ## evaluate on grid
f <- rowSums(F)*dA ## integrate by midpoint quadrature</pre>
y \leftarrow f + rnorm(og^2)*5e-4 \# add noise
## ... so each y is a noisy observation of the integral of `test1'
## over a 0.1 by 0.1 sub-square from the unit square
## Now fit model to simulated data...
L <- X*0 + dA
\#\# ... let F be the matrix of the smooth evaluated at the x,z values
## in matrices X and Z. rowSums(L*F) gives the model predicted
## integrals of `test1' corresponding to the observed `y'
L1 <- rowSums(L) ## smooths are centred --- need to add in L%*%1
## fit models to reconstruct `test1'....
b \leftarrow gam(y\sim s(X,Z,by=L)+L1-1) ## (L1 and const are confounded here)
b1 <- gam(y\sim te(X,Z,by=L)+L1-1) ## tensor product alternative
```

```
## plot results...
old.par<-par(mfrow=c(2,2))
x<-runif(n);z<-runif(n);</pre>
xs < -seq(0,1,length=30); zs < -seq(0,1,length=30)
pr<-data.frame(x=rep(xs,30),z=rep(zs,rep(30,30)))</pre>
truth<-matrix(test1(pr$x,pr$z),30,30)</pre>
contour(xs,zs,truth)
plot(b)
vis.gam(b,view=c("X","Z"),cond=list(L1=1,L=1),plot.type="contour")
vis.gam(b1,view=c("X","Z"),cond=list(L1=1,L=1),plot.type="contour")
## A "signal" regression example...#
rf <- function(x=seq(0,1,length=100)) {</pre>
## generates random functions...
  m <- ceiling(runif(1)*5) ## number of components</pre>
  f <- x*0;
 mu \leftarrow runif(m, min(x), max(x)); sig \leftarrow (runif(m)+.5)*(max(x)-min(x))/10
  for (i in 1:m) f \leftarrow f + dnorm(x,mu[i],sig[i])
  f
}
x <- seq(0,1,length=100) ## evaluation points
## example functional predictors...
par(mfrow=c(3,3)); for (i in 1:9) plot(x,rf(x),type="l",xlab="x")
## 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
f2 <- function(x) { ## the coefficient function</pre>
  (0.2*x^11*(10*(1-x))^6+10*(10*x)^3*(1-x)^10)/10
}
f \leftarrow f2(x) ## the true coefficient function
y <- L%*%f + rnorm(200)*20 ## simulated response data
## Now fit the model E(y) = L%*%f(x) where f is a smooth function.
## The summation convention is used to evaluate smooth at each value
## in matrix X to get matrix F, say. Then rowSum(L*F) gives E(y).
## create matrix of eval points for each function. Note that
## `smoothCon' is smart and will recognize the duplication...
X <- matrix(x,200,100,byrow=TRUE)</pre>
b \leftarrow gam(y\sim s(X,by=L,k=20))
par(mfrow=c(1,1))
```

102 logLik.gam

```
plot(b, shade=TRUE); lines(x, f, col=2)
```

logLik.gam

Log likelihood for a fitted GAM, for AIC

# Description

Function to extract the log-likelihood for a fitted gam model (note that the models are usually fitted by penalized likelihood maximization). Used by AIC.

# Usage

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

# **Arguments**

object fitted model objects of class gam as produced by gam().
... un-used in this case

#### **Details**

Modification of logLik.glm which corrects the degrees of freedom for use with gam objects.

The function is provided so that AIC functions correctly with gam objects, and uses the appropriate degrees of freedom (accounting for penalization). Note, when using AIC for penalized models, that the degrees of freedom are the effective degrees of freedom and not the number of parameters, and the model maximizes the penalized likelihood, not the actual likelihood. (See e.g. Hastie and Tibshirani, 1990, section 6.8.3 and also Wood 2008),

By default this routine uses a definition of the effective degrees of freedom that includes smoothing parameter uncertainty, if this is available (i.e. if smoothing parameter selection is by some variety of marginal likelihood).

#### Value

```
Standard logLik object: see logLik.
```

### Author(s)

Simon N. Wood <simon.wood@r-project.org> based directly on logLik.glm

#### 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

ls.size 103

# See Also

AIC

ls.size

Size of list elements

# Description

Produces a named array giving the size, in bytes, of the elements of a list.

# Usage

```
ls.size(x)
```

# Arguments

Χ

A list.

## Value

A numeric vector giving the size in bytes of each element of the list x. The elements of the array have the same names as the elements of the list. If x is not a list then its size in bytes is returned, un-named.

# Author(s)

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

#### References

```
http://www.maths.bris.ac.uk/~sw15190/
```

# **Examples**

```
library(mgcv)
b <- list(M=matrix(runif(100),10,10),quote=
"The world is ruled by idiots because only an idiot would want to rule the world.",
fam=binomial())
ls.size(b)</pre>
```

magic

Stable Multiple Smoothing Parameter Estimation by GCV or UBRE

## Description

Function to efficiently estimate smoothing parameters in generalized ridge regression problems with multiple (quadratic) penalties, by GCV or UBRE. The function uses Newton's method in multidimensions, backed up by steepest descent to iteratively adjust the smoothing parameters for each penalty (one penalty may have a smoothing parameter fixed at 1).

For maximal numerical stability the method is based on orthogonal decomposition methods, and attempts to deal with numerical rank deficiency gracefully using a truncated singular value decomposition approach.

# Usage

### **Arguments**

guinents	
у	is the response data vector.
Χ	is the model matrix (more columns than rows are allowed).
sp	is the array of smoothing parameters. The vector L%*%log(sp) + 1sp0 contains the logs of the smoothing parameters that actually multiply the penalty matrices stored in S (L is taken as the identity if NULL). Any sp values that are negative are autoinitialized, otherwise they are taken as supplying starting values. A supplied starting value will be reset to a default starting value if the gradient of the GCV/UBRE score is too small at the supplied value.
S	is a list of of penalty matrices. S[[i]] is the ith penalty matrix, but note that it is not stored as a full matrix, but rather as the smallest square matrix including all the non-zero elements of the penalty matrix. Element 1,1 of S[[i]] occupies element off[i], off[i] of the ith penalty matrix. Each S[[i]] must be positive semi-definite. Set to list() if there are no smoothing parameters to be estimated.
off	is an array indicating the first parameter in the parameter vector that is penalized by the penalty involving S[[i]].
L	is a matrix mapping log(sp) to the log smoothing parameters that actually multiply the penalties defined by the elemts of S. Taken as the identity, if NULL. See above under sp.
lsp0	If L is not NULL this is a vector of constants in the linear transformation from log(sp) to the actual log smoothing parameters. So the logs of the smoothing parameters multiplying the S[[i]] are given by L%*%log(sp) + 1sp0. Taken

as 0 if NULL.

rank is an array specifying the ranks of the penalties. This is useful, but not essential, for forming square roots of the penalty matrices. Н is the optional offset penalty - i.e. a penalty with a smoothing parameter fixed at 1. This is useful for allowing regularization of the estimation process, fixed smoothing penalties etc. C is the optional matrix specifying any linear equality constraints on the fitting problem. If b is the parameter vector then the parameters are forced to satisfy Cb = 0. the regression weights. If this is a matrix then it is taken as being the square root w of the inverse of the covariance matrix of y, specifically  $V_y^{-1} = w'w$ . If w is an array then it is taken as the diagonal of this matrix, or simply the weight for each element of y. See below for an example using this. gamma is an inflation factor for the model degrees of freedom in the GCV or UBRE score. is the scale parameter for use with UBRE. scale should be set to TRUE if GCV is to be used, FALSE for UBRE. gcv ridge.parameter It is sometimes useful to apply a ridge penalty to the fitting problem, penalizing the parameters in the constrained space directly. Setting this parameter to a value greater than zero will cause such a penalty to be used, with the magnitude given by the parameter value. control is a list of iteration control constants with the following elements: tol The tolerance to use in judging convergence. **step.half** If a trial step fails then the method tries halving it up to a maximum of step.half times. **rank.tol** is a constant used to test for numerical rank deficiency of the problem. Basically any singular value less than rank\_tol multiplied by the largest singular value of the problem is set to zero. is a constant to be added to the residual sum of squares (squared norm) term in extra.rss the calculation of the GCV, UBRE and scale parameter estimate. In conjuction with n. score, this is useful for certain methods for dealing with very large data n.score number to use as the number of data in GCV/UBRE score calculation: usually the actual number of data, but there are methods for dealing with very large datasets that change this. magic can make use of multiple threads if this is set to >1. nthreads

# **Details**

The method is a computationally efficient means of applying GCV or UBRE (often approximately AIC) to the problem of smoothing parameter selection in generalized ridge regression problems of the form:

$$minimise^{\sim} \|\mathbf{W}(\mathbf{X}\mathbf{b} - \mathbf{y})\|^2 + \mathbf{b}'\mathbf{H}\mathbf{b} + \sum_{i=1}^{m} \theta_i \mathbf{b}' \mathbf{S}_i \mathbf{b}$$

possibly subject to constraints Cb = 0. X is a design matrix, b a parameter vector, y a data vector, W a weight matrix,  $S_i$  a positive semi-definite matrix of coefficients defining the ith penalty with associated smoothing parameter  $\theta_i$ , **H** is the positive semi-definite offset penalty matrix and C a matrix of coefficients defining any linear equality constraints on the problem. X need not be of full column rank.

The  $\theta_i$  are chosen to minimize either the GCV score:

$$V_g = \frac{n \|\mathbf{W}(\mathbf{y} - \mathbf{A}\mathbf{y})\|^2}{[tr(\mathbf{I} - \gamma \mathbf{A})]^2}$$

or the UBRE score:

$$V_u = \|\mathbf{W}(\mathbf{y} - \mathbf{A}\mathbf{y})\|^2 / n - 2\phi tr(\mathbf{I} - \gamma \mathbf{A}) / n + \phi$$

where  $\gamma$  is gamma the inflation factor for degrees of freedom (usually set to 1) and  $\phi$  is scale, the scale parameter. A is the hat matrix (influence matrix) for the fitting problem (i.e the matrix mapping data to fitted values). Dependence of the scores on the smoothing parameters is through Α.

The method operates by Newton or steepest descent updates of the logs of the  $\theta_i$ . A key aspect of the method is stable and economical calculation of the first and second derivatives of the scores w.r.t. the log smoothing parameters. Because the GCV/UBRE scores are flat w.r.t. very large or very small  $\theta_i$ , it's important to get good starting parameters, and to be careful not to step into a flat region of the smoothing parameter space. For this reason the algorithm rescales any Newton step that would result in a  $log(\theta_i)$  change of more than 5. Newton steps are only used if the Hessian of the GCV/UBRE is postive definite, otherwise steepest descent is used. Similarly steepest descent is used if the Newton step has to be contracted too far (indicating that the quadratic model underlying Newton is poor). All initial steepest descent steps are scaled so that their largest component is 1. However a step is calculated, it is never expanded if it is successful (to avoid flat portions of the objective), but steps are successively halved if they do not decrease the GCV/UBRE score, until they do, or the direction is deemed to have failed. (Given the smoothing parameters the optimal b parameters are easily found.)

The method is coded in C with matrix factorizations performed using LINPACK and LAPACK routines.

#### Value

The function returns a list with the following items:

b	The best fit parameters given the estimated smoothing parameters.
scale	the estimated (GCV) or supplied (UBRE) scale parameter.
score	the minimized GCV or UBRE score.
sp	an array of the estimated smoothing parameters.
sp.full	an array of the smoothing parameters that actually multiply the elements of S (same as sp if L was NULL). This is $\exp(L\%*\%\log(sp))$ .
rV	a factored form of the parameter covariance matrix. The (Bayesian) covariance matrix of the parametes b is given by rV%*%t(rV)*scale.

gcv.info

is a list of information about the performance of the method with the following elements:

**full.rank** The apparent rank of the problem: number of parameters less number of equality constraints.

**rank** The estimated actual rank of the problem (at the final iteration of the method).

**fully.converged** is TRUE if the method converged by satisfying the convergence criteria, and FALSE if it coverged by failing to decrease the score along the search direction.

**hess.pos.def** is TRUE if the hessian of the UBRE or GCV score was positive definite at convergence.

iter is the number of Newton/Steepest descent iterations taken.

**score.calls** is the number of times that the GCV/UBRE score had to be evaluated.

**rms.grad** is the root mean square of the gradient of the UBRE/GCV score w.r.t. the smoothing parameters.

**R** The factor R from the QR decomposition of the weighted model matrix. This is un-pivoted so that column order corresponds to X. So it may not be upper triangular.

Note that some further useful quantities can be obtained using magic.post.proc.

### Author(s)

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

#### References

Wood, S.N. (2004) Stable and efficient multiple smoothing parameter estimation for generalized additive models. J. Amer. Statist. Ass. 99:673-686

```
http://www.maths.bris.ac.uk/~sw15190/
```

#### See Also

```
magic.post.proc,gam
```

# **Examples**

108 magic.post.proc

```
b <- gam(G=G)
  mgfit$sp;b$sp # compare smoothing parameter estimates
  edf <- magic.post.proc(G$X,mgfit,G$w)$edf # get e.d.f. per param</pre>
  range(edf-b$edf) # compare
## p>n example... fit model to first 100 data only, so more
## params than data...
  mgfit \leftarrow magic(G\$y[1:100], G\$X[1:100,], G\$sp, G\$s, G\$off, rank=G\$rank)
  edf <- magic.post.proc(G$X[1:100,],mgfit,G$w[1:100])$edf</pre>
## constrain first two smooths to have identical smoothing parameters
  L <- diag(3);L <- rbind(L[1,],L)
  mgfit \leftarrow magic(G\$y,G\$X,rep(-1,3),G\$S,G\$off,L=L,rank=G\$rank,C=G\$C)
## Now a correlated data example ...
   library(nlme)
## simulate truth
   set.seed(1);n<-400;sig<-2
    x < 0:(n-1)/(n-1)
    f < 0.2*x^11*(10*(1-x))^6+10*(10*x)^3*(1-x)^10
## produce scaled covariance matrix for AR1 errors...
    V <- corMatrix(Initialize(corAR1(.6),data.frame(x=x)))</pre>
    Cv \leftarrow chol(V) \# t(Cv)\%*\%Cv=V
## Simulate AR1 errors ...
    e \leftarrow t(Cv)\%\%norm(n,0,sig) # so cov(e) = V * sig^2
## Observe truth + AR1 errors
   y <- f + e
## GAM ignoring correlation
    par(mfrow=c(1,2))
    b <- gam(y \sim s(x, k=20))
    plot(b);lines(x,f-mean(f),col=2);title("Ignoring correlation")
## Fit smooth, taking account of *known* correlation...
   w \leftarrow solve(t(Cv)) # V^{-1} = w'w
    ## Use `gam' to set up model for fitting...
    G \leftarrow gam(y\sim s(x,k=20),fit=FALSE)
    ## fit using magic, with weight *matrix*
    \label{eq:mgfit} \mbox{ $\mbox{$^{\circ}$}$ mgfit $\mbox{$^{\circ}$}$ magic(G$y,G$X,G$sp,G$S,G$off,rank=G$rank,C=G$C,w=w) $}
## Modify previous gam object using new fit, for plotting...
    mg.stuff <- magic.post.proc(G$X,mgfit,w)</pre>
    b$edf <- mg.stuff$edf;b$Vp <- mg.stuff$Vb
    b$coefficients <- mgfit$b
    plot(b);lines(x,f-mean(f),col=2);title("Known correlation")
```

magic.post.proc

Auxilliary information from magic fit

#### **Description**

Obtains Bayesian parameter covariance matrix, frequentist parameter estimator covariance matrix, estimated degrees of freedom for each parameter and leading diagonal of influence/hat matrix, for

magic.post.proc 109

a penalized regression estimated by magic.

## Usage

```
magic.post.proc(X,object,w=NULL)
```

## **Arguments**

X is the model matrix.

object is the list returned by magic after fitting the model with model matrix X.

w is the weight vector used in fitting, or the weight matrix used in fitting (i.e.

supplied to magic, if one was.). If w is a vector then its elements are typically proportional to reciprocal variances (but could even be negative). If w is a matrix then t(w)\*\*w should typically give the inverse of the covariance matrix of the

response data supplied to magic.

#### **Details**

object contains rV ( $\mathbf{V}$ , say), and scale ( $\phi$ , say) which can be used to obtain the require quantities as follows. The Bayesian covariance matrix of the parameters is  $\mathbf{V}\mathbf{V}'\phi$ . The vector of estimated degrees of freedom for each parameter is the leading diagonal of  $\mathbf{V}\mathbf{V}'\mathbf{X}'\mathbf{W}'\mathbf{W}\mathbf{X}$  where  $\mathbf{W}$  is either the weight matrix  $\mathbf{w}$  or the matrix  $\mathrm{diag}(\mathbf{w})$ . The hat/influence matrix is given by  $\mathbf{W}\mathbf{X}\mathbf{V}\mathbf{V}'\mathbf{X}'\mathbf{W}'$ .

The frequentist parameter estimator covariance matrix is  $VV'X'W'WXVV'\phi$ : it is sometimes useful for testing terms for equality to zero.

## Value

A list with three items:

Vb the Bayesian covariance matrix of the model parameters.

Ve the frequentist covariance matrix for the parameter estimators.

hat the leading diagonal of the hat (influence) matrix.

edf the array giving the estimated degrees of freedom associated with each parame-

ter.

## Author(s)

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

## See Also

magic

110 mgcv.FAQ

mgcv.FAQ

Frequently Asked Questions for package mgcv

### **Description**

This page provides answers to some of the questions that get asked most often about mgcv

## **FAQ list**

- 1. **How can I compare gamm models?** In the identity link normal errors case, then AIC and hypotheis testing based methods are fine. Otherwise it is best to work out a strategy based on the summary.gam Alternatively, simple random effects can be fitted with gam, which makes comparison straightforward. Package gamm4 is an alternative, which allows AIC type model selection for generalized models.
- 2. **How do I get the equation of an estimated smooth?** This slightly misses the point of semi-parametric modelling: the idea is that we estimate the form of the function from data without assuming that it has a particular simple functional form. Of course for practical computation the functions do have underlying mathematical representations, but they are not very helpful, when written down. If you do need the functional forms then see chapter 4 of Wood (2006). However for most purposes it is better to use predict.gam to evaluate the function for whatever argument values you need. If derivatives are required then the simplest approach is to use finite differencing (which also allows SEs etc to be calculated).
- 3. Some of my smooths are estimated to be straight lines and their confidence intervals vanish at some point in the middle. What is wrong? Nothing. Smooths are subject to sum-to-zero identifiability constraints. If a smooth is estimated to be a straight line then it consequently has one degree of freedom, and there is no choice about where it passes through zero so the CI must vanish at that point.
- 4. How do I test whether a smooth is significantly different from a straight line. See tprs and the example therein.
- 5. Some code from Wood (2006) causes an error: why? The book was written using mgcv version 1.3. To allow for REML estimation of smoothing parameters in versions 1.5, some changes had to be made to the syntax. In particular the function gam.method no longer exists. The smoothness selection method (GCV, REML etc) is now controlled by the method argument to gam while the optimizer is selected using the optimizer argument. See gam and http://www.maths.bris.ac.uk/~sw15190/igam/index.html for details.
- 6. Why is a model object saved under a previous mgcv version not usable with the current mgcv version? I'm sorry about this issue, I know it's really annoying. Here's my defence. Each mgcv version is run through an extensive test suite before release, to ensure that it gives the same results as before, unless there are good statistical reasons why not (e.g. improvements to p-value approximation, fixing of an error). However it is sometimes necessary to modify the internal structure of model objects in a way that makes an old style object unusable with a newer version. For example, bug fixes or new R features sometimes require changes in the way that things are computed which in turn require modification of the object structure. Similarly improvements, such as the ability to compute smoothing parameters by RE/ML require object level changes. The only fix to this problem is to access the old object using the original mgcv version (available on CRAN), or to recompute the fit using the current mgcv version.

mgcv.package 111

7. When using gamm or gamm4, the reported AIC is different for the gam object and the 1me or 1mer object. Why is this? There are several reasons for this. The most important is that the models being used are actually different in the two representations. When treating the GAM as a mixed model, you are implicitly assuming that if you gathered a replicate dataset, the smooths in your model would look completely different to the smooths from the original model, except for having the same degree of smoothness. Technically you would expect the smooths to be drawn afresh from their distribution under the random effects model. When viewing the gam from the usual penalized regression perspective, you would expect smooths to look broadly similar under replication of the data. i.e. you are really using Bayesian model for the smooths, rather than a random effects model (it's just that the frequentist random effects and Bayesian computations happen to coincide for computing the estimates). As a result of the different assumptions about the data generating process, AIC model comparisons can give rather different answers depending on the model adopted. Which you use should depend on which model you really think is appropriate. In addition the computations of the AICs are different. The mixed model AIC uses the marginal liklihood and the corresponding number of model parameters. The gam model uses the penalized likelihood and the effective degrees of freedom.

- 8. What does 'mgcv' stand for? 'Mixed GAM Computation Vehicle', is my current best effort (let me know if you can do better). Originally it stood for 'Multiple GCV', which has long since ceased to be usefully descriptive, (and I can't really change 'mgcv' now without causing disruption). On a bad inbox day 'Mad GAM Computing Vulture'.
- 9. My new method is failing to beat mgcv, what can I do? If speed is the problem, then make sure that you use the slowest basis possible ("tp") with a large sample size, and experiment with different optimizers to find one that is slow for your problem. For prediction error/MSE, then leaving the smoothing basis dimensions at their arbitrary defaults, when these are inappropriate for the problem setting, is a good way of reducing performance. Similarly, using p-splines in place of derivative penalty based splines will often shave a little more from the performance here. Unlike REML/ML, prediction error based smoothness selection criteria such as Mallows Cp and GCV often produce a small proportion of severe overfits, so careful choise of smoothness selection method can help further. In particular GCV etc. usually result in worse confidence interval and p-value performance than ML or REML. If all this fails, try using a really odd simulation setup for which mgcv is clearly not suited: for example poor performance is almost guaranteed for small noisy datasets with large numbers of predictors.

### Author(s)

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

#### References

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

mgcv.package

Mixed GAM Computation Vehicle with GCV/AIC/REML smoothness estimation and GAMMs by REML/PQL

112 mgcv.package

## **Description**

mgcv provides functions for generalized additive modelling (gam and bam) and generalized additive mixed modelling (gamm, and random.effects). The term GAM is taken to include any model dependent on unknown smooth functions of predictors and estimated by quadratically penalized (possibly quasi-) likelihood maximization. Available distributions are covered in family.mgcv and available smooths in smooth.terms.

Particular features of the package are facilities for automatic smoothness selection (Wood, 2004, 2011), and the provision of a variety of smooths of more than one variable. User defined smooths can be added. A Bayesian approach to confidence/credible interval calculation is provided. Linear functionals of smooths, penalization of parametric model terms and linkage of smoothing parameters are all supported. Lower level routines for generalized ridge regression and penalized linearly constrained least squares are also available.

#### **Details**

mgcv provides generalized additive modelling functions gam, predict.gam and plot.gam, which are very similar in use to the S functions of the same name designed by Trevor Hastie (with some extensions). However the underlying representation and estimation of the models is based on a penalized regression spline approach, with automatic smoothness selection. A number of other functions such as summary.gam and anova.gam are also provided, for extracting information from a fitted gamObject.

Use of gam is much like use of glm, except that within a gam model formula, isotropic smooths of any number of predictors can be specified using s terms, while scale invariant smooths of any number of predictors can be specified using te, ti or t2 terms. smooth.terms provides an overview of the built in smooth classes, and random.effects should be refered to for an overview of random effects terms (see also mrf for Markov random fields). Estimation is by penalized likelihood or quasi-likelihood maximization, with smoothness selection by GCV, GACV, gAIC/UBRE or (RE)ML. See gam, gam.models, linear.functional.terms and gam.selection for some discussion of model specification and selection. For detailed control of fitting see gam.convergence, gam arguments method and optimizer and gam.control. For checking and visualization see gam.check, choose.k, vis.gam and plot.gam. While a number of types of smoother are built into the package, it is also extendable with user defined smooths, see smooth.construct, for example.

A Bayesian approach to smooth modelling is used to derive standard errors on predictions, and hence credible intervals (see Marra and Wood, 2012). The Bayesian covariance matrix for the model coefficients is returned in Vp of the gamObject. See predict.gam for examples of how this can be used to obtain credible regions for any quantity derived from the fitted model, either directly, or by direct simulation from the posterior distribution of the model coefficients. Approximate p-values can also be obtained for testing individual smooth terms for equality to the zero function, using similar ideas (see Wood, 2013a,b). Frequentist approximations can be used for hypothesis testing based model comparison. See anova.gam and summary.gam for more on hypothesis testing.

For large datasets (that is large n) see bam which is a version of gam with a much reduced memory footprint.

The package also provides a generalized additive mixed modelling function, gamm, based on a PQL approach and lme from the nlme library (for an lme4 based version, see package gamm4). gamm is particularly useful for modelling correlated data (i.e. where a simple independence model for the residual variation is inappropriate). In addition, low level routine magic can fit models to data with a known correlation structure.

mgcv.package 113

Some underlying GAM fitting methods are available as low level fitting functions: see magic. But there is little functionality that can not be more conventiently accessed via gam. Penalized weighted least squares with linear equality and inequality constraints is provided by pcls.

For a complete list of functions type library(help=mgcv). See also mgcv.FAQ.

## Author(s)

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

with contributions and/or help from Natalya Pya, Thomas Kneib, Kurt Hornik, Mike Lonergan, Henric Nilsson, Fabian Scheipl and Brian Ripley.

Polish translation - Lukasz Daniel; German translation - Chris Leick, Detlef Steuer; French Translation - Philippe Grosjean

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

Part funded by EPSRC: EP/K005251/1

#### References

These provide details for the underlying mgcv methods, and fuller references to the large literature on which the methods are based.

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. (2004) Stable and efficient multiple smoothing parameter estimation for generalized additive models. J. Amer. Statist. Ass. 99:673-686.

Marra, G and S.N. Wood (2012) Coverage Properties of Confidence Intervals for Generalized Additive Model Components. Scandinavian Journal of Statistics, 39(1), 53-74.

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

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

Wood, S.N. (2006) Generalized Additive Models: an introduction with R, CRC

Development of mgcv version 1.8 was part funded by EPSRC grants EP/K005251/1 and EP/I000917/1.

## **Examples**

## see examples for gam and gamm

114 mgcv.parallel

mgcv.parallel

Parallel computation in mgcv.

## Description

mgcv can make some use of multiple cores or a cluster.

bam can use an openMP based parallelization approach alongside discretisation of covariates to achieve substantial speed ups. This is selected using the discrete=TRUE option to bam, withthe number of threads controlled via the nthreads argument. This is the approach that scales best. See example below.

Alternatively, function bam can use the facilities provided in the parallel package. See examples below. Note that most multi-core machines are memory bandwidth limited, so parallel speed up tends to be rather variable.

Function gam can use parallel threads on a (shared memory) multi-core machine via openMP (where this is supported). To do this, set the desired number of threads by setting nthreads to the number of cores to use, in the control argument of gam. Note that, for the most part, only the dominant  $O(np^2)$  steps are parallelized (n is number of data, p number of parameters). For additive Gaussian models estimated by GCV, the speed up can be disappointing as these employ an  $O(p^3)$  SVD step that can also have substantial cost in practice.

magic can also use multiple cores, but the same comments apply as for the GCV Gaussian additive model.

If control\$nthreads is set to more than the number of cores detected, then only the number of detected cores is used. Note that using virtual cores usually gives very little speed up, and can even slow computations slightly. For example, many Intel processors reporting 4 cores actually have 2 physical cores, each with 2 virtual cores, so using 2 threads gives a marked increase in speed, while using 4 threads makes little extra difference.

Note that on Intel and similar processors the maximum performance is usually achieved by disabling Hyper-Threading in BIOS, and then setting the number of threads to the number of physical cores used. This prevents the operating system scheduler from sending 2 floating point intensive threads to the same physical core, where they have to share a floating point unit (and cache) and therefore slow each other down. The scheduler tends to do this under the manager - worker multi-threading approach used in mgcv, since the manager thread looks very busy up to the point at which the workers are set to work, and at the point of scheduling the sceduler has no way of knowing that the manager thread actually has nothing more to do until the workers are finished. If you are working on a many cored platform where you can not disable hyper-threading then it may be worth setting the number of threads to one less than the number of physical cores, to reduce the frequency of such scheduling problems.

mgcv's work splitting always makes the simple assumption that all your cores are equal, and you are not sharing them with other floating point intensive threads.

In addition to hyper-threading several features may lead to apparently poor scaling. The first is that many CPUs have a Turbo mode, whereby a few cores can be run at higher frequency, provided the overall power used by the CPU does not exceed design limits, however it is not possible for all cores on the CPU to run at this frequency. So as you add threads eventually the CPU frequency has to be reduced below the Turbo frequency, with the result that you don't get the expected speed up

mgcv.parallel 115

from adding cores. Secondly, most modern CPUs have their frequency set dynamically according to load. You may need to set the system power management policy to favour high performance in order to maximize the chance that all threads run at the speed you were hoping for (you can turn off dynamic power control in BIOS, but then you turn off the possibility of Turbo also).

Because the computational burden in mgcv is all in the linear algebra, then parallel computation may provide reduced or no benefit with a tuned BLAS. This is particularly the case if you are using a multi threaded BLAS, but a BLAS that is tuned to make efficient use of a particular cache size may also experience loss of performance if threads have to share the cache.

## Author(s)

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

#### References

```
https://computing.llnl.gov/tutorials/openMP/
```

```
## illustration of multi-threading with gam...
require(mgcv);set.seed(9)
dat <- gamSim(1,n=2000,dist="poisson",scale=.1)</pre>
k <- 12;bs <- "cr";ctrl <- list(nthreads=2)</pre>
system.time(b1<-gam(y~s(x0,bs=bs)+s(x1,bs=bs)+s(x2,bs=bs,k=k)
            ,family=poisson,data=dat,method="REML"))[3]
system. time(b2 < -gam(y < x0, bs = bs) + s(x1, bs = bs) + s(x2, bs = bs, k = k),
            family=poisson,data=dat,method="REML",control=ctrl))[3]
## Poisson example on a cluster with 'bam'.
## Note that there is some overhead in initializing the
## computation on the cluster, associated with loading
## the Matrix package on each node. For this reason the
## sample sizes here are very small to keep CRAN happy, but at
## this low sample size you see little advantage of parallel computation.
k <- 13
dat <- gamSim(1,n=6000,dist="poisson",scale=.1)</pre>
require(parallel)
nc <- 2 ## cluster size, set for example portability
if (detectCores()>1) { ## no point otherwise
 cl <- makeCluster(nc)</pre>
 ## could also use makeForkCluster, but read warnings first!
} else cl <- NULL
system.time(b3 <- bam(y \sim s(x0,bs=bs,k=7)+s(x1,bs=bs,k=7)+s(x2,bs=bs,k=k)
            ,data=dat,family=poisson(),chunk.size=5000,cluster=cl))
fv <- predict(b3,cluster=cl) ## parallel prediction</pre>
```

116 missing.data

missing.data

Missing data in GAMs

## **Description**

If there are missing values in the reponse or covariates of a GAM then the default is simply to use only the 'complete cases'. If there are many missing covariates, this can get rather wasteful. One possibility is then to use imputation. Another is to substitute a simple random effects model in which the by variable mechanism is used to set s(x) to zero for any missing x, while a Gaussian random effect is then substituted for the 'missing' s(x). See the example for details of how this works, and gam.modelsfor the necessary background on by variables.

## Author(s)

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

## See Also

```
gam.vcomp, gam.models, s, smooth.construct.re.smooth.spec,gam
```

```
## The example takes a couple of minutes to run, so is marked
## don't run to save CRAN time.
## Not run:
require(mgcv)
par(mfrow=c(4,4), mar=c(4,4,1,1))
for (sim in c(1,7)) { ## cycle over uncorrelated and correlated covariates
 n \leftarrow 350; set.seed(2)
 ## simulate data but randomly drop 300 covariate measurements
 ## leaving only 50 complete cases...
 dat <- gamSim(sim,n=n,scale=3) ## 1 or 7</pre>
 drop <- sample(1:n,300) ## to
 for (i in 2:5) dat[drop[1:75+(i-2)*75],i] <- NA
 ## process data.frame producing binary indicators of missingness,
 ## mx0, mx1 etc. For each missing value create a level of a factor
 ## idx0, idx1, etc. So idx0 has as many levels as x0 has missing
 ## values. Replace the NA's in each variable by the mean of the
 ## non missing for that variable...
```

model.matrix.gam 117

```
dname <- names(dat)[2:5]</pre>
 dat1 <- dat
 for (i in 1:4) {
   by.name <- paste("m",dname[i],sep="")</pre>
   dat1[[by.name]] <- is.na(dat1[[dname[i]]])</pre>
   dat1[[dname[i]]][dat1[[by.name]]] <- mean(dat1[[dname[i]]],na.rm=TRUE)</pre>
    lev <- rep(1,n);lev[dat1[[by.name]]] <- 1:sum(dat1[[by.name]])</pre>
    id.name <- paste("id",dname[i],sep="")</pre>
   dat1[[id.name]] <- factor(lev)</pre>
   dat1[[by.name]] <- as.numeric(dat1[[by.name]])</pre>
 }
 ## Fit a gam, in which any missing value contributes zero
 ## to the linear predictor from its smooth, but each
 ## missing has its own random effect, with the random effect
 ## variances being specific to the variable. e.g.
 ## for s(x0,by=ordered(!mx0)), declaring the `by' as an ordered
 ## factor ensures that the smooth is centred, but multiplied
 ## by zero when mx0 is one (indicating a missing x0). This means
 ## that any value (within range) can be put in place of the
 ## NA for x0. s(idx0,bs="re",by=mx0) produces a separate Gaussian
 ## random effect for each missing value of x0 (in place of s(x0),
 ## effectively). The `by' variable simply sets the random effect to
 ## zero when x0 is non-missing, so that we can set idx0 to any
 ## existing level for these cases.
 b \leftarrow gam(y^s(x0,by=ordered(!mx0))+s(x1,by=ordered(!mx1))+
             s(x2,by=ordered(!mx2))+s(x3,by=ordered(!mx3))+
             s(idx0,bs="re",by=mx0)+s(idx1,bs="re",by=mx1)+
             s(idx2,bs="re",by=mx2)+s(idx3,bs="re",by=mx3)
             ,data=dat1,method="REML")
 for (i in 1:4) plot(b, select=i) ## plot the smooth effects from b
 ## fit the model to the `complete case' data...
 b2 <- gam(y^s(x0)+s(x1)+s(x2)+s(x3), data=dat, method="REML")
 plot(b2) ## plot the complete case results
## End(Not run)
```

model.matrix.gam

Extract model matrix from GAM fit

## Description

Obtains the model matrix from a fitted gam object.

118 model.matrix.gam

## Usage

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

## **Arguments**

```
object fitted model object of class gam as produced by gam().
... other arguments, passed to predict.gam.
```

## **Details**

Calls predict.gam with no newdata argument and type="lpmatrix" in order to obtain the model matrix of object.

#### Value

A model matrix.

## Author(s)

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

## References

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

#### See Also

gam

```
require(mgcv)
n <- 15
x <- runif(n)
y <- sin(x*2*pi) + rnorm(n)*.2
mod <- gam(y~s(x,bs="cc",k=6),knots=list(x=seq(0,1,length=6)))
model.matrix(mod)</pre>
```

mono.con 119

mono.con

Monotonicity constraints for a cubic regression spline

## **Description**

Finds linear constraints sufficient for monotonicity (and optionally upper and/or lower boundedness) of a cubic regression spline. The basis representation assumed is that given by the gam, "cr" basis: that is the spline has a set of knots, which have fixed x values, but the y values of which constitute the parameters of the spline.

## Usage

```
mono.con(x,up=TRUE,lower=NA,upper=NA)
```

## **Arguments**

x The array of knot locations.

up If TRUE then the constraints imply increase, if FALSE then decrease.

lower This specifies the lower bound on the spline unless it is NA in which case no

lower bound is imposed.

upper This specifies the upper bound on the spline unless it is NA in which case no

upper bound is imposed.

#### **Details**

Consider the natural cubic spline passing through the points  $\{x_i, p_i : i = 1...n\}$ . Then it is possible to find a relatively small set of linear constraints on  $\mathbf{p}$  sufficient to ensure monotonicity (and bounds if required):  $\mathbf{A}\mathbf{p} \geq \mathbf{b}$ . Details are given in Wood (1994).

## Value

a list containing constraint matrix A and constraint vector b.

## Author(s)

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

### References

Gill, P.E., Murray, W. and Wright, M.H. (1981) *Practical Optimization*. Academic Press, London. Wood, S.N. (1994) Monotonic smoothing splines fitted by cross validation. *SIAM Journal on Scientific Computing* **15**(5), 1126–1133.

```
http://www.maths.bris.ac.uk/~sw15190/
```

#### See Also

```
magic, pcls
```

120 mroot

## **Examples**

```
## see ?pcls
```

mroot

Smallest square root of matrix

## **Description**

Find a square root of a positive semi-definite matrix, having as few columns as possible. Uses either pivoted choleski decomposition or singular value decomposition to do this.

## Usage

```
mroot(A,rank=NULL,method="chol")
```

## **Arguments**

A The positive semi-definite matrix, a square root of which is to be found.

rank if the rank of the matrix A is known then it should be supplied. NULL or <1 imply

that it should be estimated.

method "chol" to use pivoted choloeski decompositon, which is fast but tends to over-

estimate rank. "svd" to use singular value decomposition, which is slow, but is

the most accurate way to estimate rank.

## **Details**

The function uses SVD, or a pivoted Choleski routine. It is primarily of use for turning penalized regression problems into ordinary regression problems.

## Value

A matrix, **B** with as many columns as the rank of **A**, and such that  $\mathbf{A} = \mathbf{B}\mathbf{B}'$ .

## Author(s)

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

```
require(mgcv)
set.seed(0)
a <- matrix(runif(24),6,4)
A <- a%*%t(a) ## A is +ve semi-definite, rank 4
B <- mroot(A) ## default pivoted choleski method
tol <- 100*.Machine$double.eps
chol.err <- max(abs(A-B%*%t(B)));chol.err
if (chol.err>tol) warning("mroot (chol) suspect")
B <- mroot(A,method="svd") ## svd method</pre>
```

multinom 121

```
svd.err <- max(abs(A-B%*%t(B)));svd.err
if (svd.err>tol) warning("mroot (svd) suspect")
```

multinom

GAM multinomial logistic regression

## **Description**

Family for use with gam, implementing regression for categorical response data. Categories must be coded 0 to K, where K is a positive integer. gam should be called with a list of K formulae, one for each category except category zero (extra formulae for shared terms may also be supplied). The first formula also specifies the response variable.

### Usage

```
multinom(K=1)
```

## **Arguments**

Κ

There are K+1 categories and K linear predictors.

## **Details**

The model has K linear predictors,  $\eta_j$ , each dependent on smooth functions of predictor variables, in the usual way. If response variable, y, contains the class labels 0,...,K then the likelihood for y>0 is  $\exp(\eta_y)/\{1+\sum_j\exp(\eta_j)\}$ . If y=0 the likelihood is  $1/\{1+\sum_j\exp(\eta_j)\}$ . In the two class case this is just a binary logistic regression model. The implementation uses the approach to GAMLSS models described in Wood, Pya and Saefken (2015).

The residuals returned for this model are simply the square root of -2 times the deviance for each observation, with a positive sign if the observed y is the most probable class for this observation, and a negative sign otherwise.

Use predict with type="response" to get the predicted probabilities in each category.

Note that the model is not completely invariant to category relabelling, even if all linear predictors have the same form. Realistically this model is unlikely to be suitable for problems with large numbers of categories. Missing categories are not supported.

## Value

An object of class general. family.

## Author(s)

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

### References

Wood, S.N., N. Pya and B. Saefken (2015), Smoothing parameter and model selection for general smooth models. http://arxiv.org/abs/1511.03864

122 mvn

## See Also

ocat

## **Examples**

```
library(mgcv)
set.seed(6)
## simulate some data from a three class model
n <- 1000
f1 \leftarrow function(x) \sin(3*pi*x)*exp(-x)
f2 \leftarrow function(x) x^3
f3 <- function(x) .5*exp(-x^2)-.2
f4 \leftarrow function(x) 1
x1 <- runif(n);x2 <- runif(n)</pre>
eta1 <- 2*(f1(x1) + f2(x2))-.5
eta2 <- 2*(f3(x1) + f4(x2))-1
p <- exp(cbind(0,eta1,eta2))</pre>
p <- p/rowSums(p) ## prob. of each category</pre>
cp <- t(apply(p,1,cumsum)) ## cumulative prob.</pre>
## simulate multinomial response with these probabilities
## see also ?rmultinom
y <- apply(cp,1,function(x) min(which(x>runif(1))))-1
## plot simulated data...
plot(x1,x2,col=y+3)
## now fit the model...
b \leftarrow gam(list(y\sim s(x1)+s(x2),\sim s(x1)+s(x2)),family=multinom(K=2))
plot(b,pages=1)
gam.check(b)
## now a simple classification plot...
expand.grid(x1=seq(0,1,length=40),x2=seq(0,1,length=40)) -> gr
pp <- predict(b,newdata=gr,type="response")</pre>
pc \leftarrow apply(pp, 1, function(x) which(max(x)==x)[1])-1
plot(gr,col=pc+3,pch=19)
```

mvn

Multivariate normal additive models

## **Description**

Family for use with gam implementing smooth multivariate Gaussian regression. The means for each dimension are given by a separate linear predictor, which may contain smooth components. The Choleski factor of the response precision matrix is estimated as part of fitting.

## Usage

```
mvn(d=2)
```

mvn 123

## **Arguments**

d

The dimension of the response (>1).

#### **Details**

The response is d dimensional multivariate normal, where the covariance matrix is estimated, and the means for each dimension have sperate linear predictors. Model sepcification is via a list of gam like formulae - one for each dimension. See example.

Currently the family ignores any prior weights, and is implemented using first derivative information sufficient for BFGS estimation of smoothing parameters. "response" residuals give raw residuals, while "deviance" residuals are standardized to be approximately independent standard normal if all is well.

#### Value

An object of class general.family.

#### Author(s)

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

## References

Wood, S.N., N. Pya and B. Saefken (2015), Smoothing parameter and model selection for general smooth models. http://arxiv.org/abs/1511.03864

### See Also

gaussian

```
library(mgcv)
 ## simulate some data...
V \leftarrow matrix(c(2,1,1,2),2,2)
 f0 \leftarrow function(x) 2 * sin(pi * x)
 f1 \leftarrow function(x) \exp(2 * x)
 f2 \leftarrow function(x) 0.2 \times x^{11} \times (10 \times (1 - x))^{6} + 10 \times (1 - x)^{11} + (10 \times (1 - x))^{11} + (10 \times (1 - x)
                                                                                (10 * x)^3 * (1 - x)^10
n <- 300
x0 <- runif(n);x1 <- runif(n);</pre>
x2 <- runif(n);x3 <- runif(n)</pre>
 y <- matrix(0,n,2)</pre>
 for (i in 1:n) {
             mu \leftarrow c(f0(x0[i])+f1(x1[i]),f2(x2[i]))
             y[i,] <- rmvn(1,mu,V)
dat <- data.frame(y0=y[,1],y1=y[,2],x0=x0,x1=x1,x2=x2,x3=x3)
  ## fit model...
```

124 negbin

```
b <- gam(list(y0~s(x0)+s(x1),y1~s(x2)+s(x3)),family=mvn(d=2),data=dat)\\ b\\ summary(b)\\ plot(b,pages=1)\\ solve(crossprod(b\$family\$data\$R))  ## estimated cov matrix
```

negbin

GAM negative binomial families

## Description

The gam modelling function is designed to be able to use the negbin family (a modification of MASS library negative.binomial family by Venables and Ripley), or the nb function designed for integrated estimation of parameter theta.  $\theta$  is the parameter such that  $var(y) = \mu + \mu^2/\theta$ , where  $\mu = E(y)$ .

Two approaches to estimating theta are available (with gam only):

- With negbin then if 'performance iteration' is used for smoothing parameter estimation (see gam), then smoothing parameters are chosen by GCV and theta is chosen in order to ensure that the Pearson estimate of the scale parameter is as close as possible to 1, the value that the scale parameter should have.
- If 'outer iteration' is used for smoothing parameter selection with the nb family then theta is estimated alongside the smoothing parameters by ML or REML.

To use the first option, set the optimizer argument of gam to "perf" (it can sometimes fail to converge).

### Usage

```
negbin(theta = stop("'theta' must be specified"), link = "log")
nb(theta = NULL, link = "log")
```

## **Arguments**

theta

Either i) a single value known value of theta or ii) two values of theta specifying the endpoints of an interval over which to search for theta (this is an option only for negbin). For nb then a positive supplied theta is treated as a fixed known parameter, otherwise it is estimated (the absolute value of a negative theta is taken as a starting value).

link

The link function: one of "log", "identity" or "sqrt"

negbin 125

#### **Details**

nb allows estimation of the theta parameter alongside the model smoothing parameters, but is only useable with gam (not bam or gamm).

For negbin, if a single value of theta is supplied then it is always taken as the known fixed value and this is useable with bam and gamm. If theta is two numbers (theta[2]>theta[1]) then they are taken as specifying the range of values over which to search for the optimal theta. This option should only be used with performance iteration estimation (see gam argument optimizer), in which case the method of estimation is to choose  $\theta$  so that the GCV (Pearson) estimate of the scale parameter is one (since the scale parameter is one for the negative binomial). In this case  $\theta$  estimation is nested within the IRLS loop used for GAM fitting. After each call to fit an iteratively weighted additive model to the IRLS pseudodata, the  $\theta$  estimate is updated. This is done by conditioning on all components of the current GCV/Pearson estimator of the scale parameter except  $\theta$  and then searching for the  $\hat{\theta}$  which equates this conditional estimator to one. The search is a simple bisection search after an initial crude line search to bracket one. The search will terminate at the upper boundary of the search region is a Poisson fit would have yielded an estimated scale parameter <1.

#### Value

For negbin an object inheriting from class family, with additional elements

dvar the function giving the first derivative of the variance function w.r.t. mu. d2var the function giving the second derivative of the variance function w.r.t. mu. getTheta

A function for retrieving the value(s) of theta. This also useful for retriving the

estimate of theta after fitting (see example).

For nb an object inheriting from class extended.family.

## WARNINGS

gamm and bam do not support theta estimation

The negative binomial functions from the MASS library are no longer supported.

#### Author(s)

Simon N. Wood < simon.wood@r-project.org > modified from Venables and Ripley's negative.binomial family.

### References

Wood, S.N., N. Pya and B. Saefken (2015), Smoothing parameter and model selection for general smooth models. http://arxiv.org/abs/1511.03864

Venables, B. and B.R. Ripley (2002) Modern Applied Statistics in S, Springer.

```
library(mgcv)
set.seed(3)
n<-400
```

126 new.name

```
dat <- gamSim(1,n=n)</pre>
g <- exp(dat$f/5)
## negative binomial data...
dat$y <- rnbinom(g,size=3,mu=g)</pre>
## known theta fit ...
b0 <- gam(y \sim s(x0) + s(x1) + s(x2) + s(x3), family = negbin(3), data = dat)
plot(b0,pages=1)
print(b0)
## same with theta estimation...
b \leftarrow gam(y^s(x0)+s(x1)+s(x2)+s(x3),family=nb(),data=dat)
plot(b,pages=1)
print(b)
b$family$getTheta(TRUE) ## extract final theta estimate
## unknown theta via performance iteration...
b1 <- gam(y \sim s(x0) + s(x1) + s(x2) + s(x3), family = negbin(c(1,10)),
           optimizer="perf",data=dat)
plot(b1,pages=1)
print(b1)
## another example...
set.seed(1)
f <- dat$f
f \leftarrow f - min(f) + 5; g \leftarrow f^2/10
dat$y <- rnbinom(g,size=3,mu=g)</pre>
b2 \leftarrow gam(y^s(x0)+s(x1)+s(x2)+s(x3),family=nb(link="sqrt"),
          data=dat,method="REML")
plot(b2,pages=1)
print(b2)
rm(dat)
```

new.name

Obtain a name for a new variable that is not already in use

## **Description**

gamm works by transforming a GAMM into something that can be estimated by 1me, but this involves creating new variables, the names of which should not clash with the names of other variables on which the model depends. This simple service routine checks a suggested name against a list of those in use, and if necessary modifies it so that there is no clash.

## Usage

```
new.name(proposed,old.names)
```

## Arguments

proposed a suggested name

old. names An array of names that must not be duplicated

notExp 127

#### Value

A name that is not in old.names.

### Author(s)

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

#### References

```
http://www.maths.bris.ac.uk/~sw15190/
```

## See Also

gamm

## **Examples**

```
require(mgcv)
old <- c("a","tuba","is","tubby")
new.name("tubby",old)</pre>
```

notExp

Functions for better-than-log positive parameterization

### **Description**

It is common practice in statistical optimization to use log-parameterizations when a parameter ought to be positive. i.e. if an optimization parameter a should be non-negative then we use a=exp(b) and optimize with respect to the unconstrained parameter b. This often works well, but it does imply a rather limited working range for b: using 8 byte doubles, for example, if b's magnitude gets much above 700 then a overflows or underflows. This can cause problems for numerical optimization methods.

notExp is a monotonic function for mapping the real line into the positive real line with much less extreme underflow and overflow behaviour than exp. It is a piece-wise function, but is continuous to second derivative: see the source code for the exact definition, and the example below to see what it looks like.

notLog is the inverse function of notExp.

The major use of these functions was originally to provide more robust pdMat classes for lme for use by gamm. Currently the notExp2 and notLog2 functions are used in their place, as a result of changes to the nlme optimization routines.

## Usage

```
notExp(x)
notLog(x)
```

128 notExp2

### **Arguments**

Х

Argument array of real numbers (notExp) or positive real numbers (notLog).

#### Value

An array of function values evaluated at the supplied argument values.

#### Author(s)

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

#### References

```
http://www.maths.bris.ac.uk/~sw15190/
```

#### See Also

```
pdTens, pdIdnot, gamm
```

## **Examples**

```
## Illustrate the notExp function:
## less steep than exp, but still monotonic.
require(mgcv)
x <- -100:100/10
op <- par(mfrow=c(2,2))
plot(x,notExp(x),type="1")
lines(x,exp(x),col=2)
plot(x,log(notExp(x)),type="1")
lines(x,log(exp(x)),col=2) # redundancy intended
x <- x/4
plot(x,notExp(x),type="1")
lines(x,exp(x),col=2)
plot(x,log(notExp(x)),type="1")
lines(x,log(exp(x)),col=2) # redundancy intended
par(op)
range(notLog(notExp(x))-x) # show that inverse works!</pre>
```

notExp2

Alternative to log parameterization for variance components

## Description

notLog2 and notExp2 are alternatives to log and exp or notLog and notExp for re-parameterization of variance parameters. They are used by the pdTens and pdIdnot classes which in turn implement smooths for gamm.

The functions are typically used to ensure that smoothing parameters are positive, but the notExp2 is not monotonic: rather it cycles between 'effective zero' and 'effective infinity' as its argument changes. The notLog2 is the inverse function of the notExp2 only over an interval centered on zero.

notExp2 129

Parameterizations using these functions ensure that estimated smoothing parameters remain positive, but also help to ensure that the likelihood is never indefinite: once a working parameter pushes a smoothing parameter below 'effective zero' or above 'effective infinity' the cyclic nature of the notExp2 causes the likelihood to decrease, where otherwise it might simply have flattened.

This parameterization is really just a numerical trick, in order to get 1me to fit gamm models, without failing due to indefiniteness. Note in particular that asymptotic results on the likelihood/REML criterion are not invalidated by the trick, unless parameter estimates end up close to the effective zero or effective infinity: but if this is the case then the asymptotics would also have been invalid for a conventional monotonic parameterization.

This reparameterization was made necessary by some modifications to the underlying optimization method in 1me introduced in nlme 3.1-62. It is possible that future releases will return to the notExp parameterization.

Note that you can reset 'effective zero' and 'effective infinity': see below.

## Usage

```
notExp2(x,d=.Options$mgcv.vc.logrange,b=1/d)
notLog2(x,d=.Options$mgcv.vc.logrange,b=1/d)
```

#### **Arguments**

- x Argument array of real numbers (notExp) or positive real numbers (notLog).
- d the range of notExp2 runs from exp(-d) to exp(d). To change the range used by gamm reset mgcv.vc.logrange using options.
- b determines the period of the cycle of notExp2.

#### Value

An array of function values evaluated at the supplied argument values.

## Author(s)

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

### References

```
http://www.maths.bris.ac.uk/~sw15190/
```

#### See Also

```
pdTens, pdIdnot, gamm
```

null.space.dimension

## **Examples**

```
## Illustrate the notExp2 function:
require(mgcv)
x <- seq(-50,50,length=1000)
op <- par(mfrow=c(2,2))
plot(x,notExp2(x),type="1")
lines(x,exp(x),col=2)
plot(x,log(notExp2(x)),type="1")
lines(x,log(exp(x)),col=2) # redundancy intended
x <- x/4
plot(x,notExp2(x),type="1")
lines(x,exp(x),col=2)
plot(x,log(notExp2(x)),type="1")
lines(x,log(exp(x)),col=2) # redundancy intended
par(op)</pre>
```

null.space.dimension The basis of the space of un-penalized functions for a TPRS

## Description

The thin plate spline penalties give zero penalty to some functions. The space of these functions is spanned by a set of polynomial terms. null.space.dimension finds the dimension of this space, M, given the number of covariates that the smoother is a function of, d, and the order of the smoothing penalty, m. If m does not satisfy 2m > d then the smallest possible dimension for the null space is found given d and the requirement that the smooth should be visually smooth.

## Usage

```
null.space.dimension(d,m)
```

## **Arguments**

d is a positive integer - the number of variables of which the t.p.s. is a function.

m a non-negative integer giving the order of the penalty functional, or signalling

that the default order should be used.

## **Details**

Thin plate splines are only visually smooth if the order of the wiggliness penalty, m, satisfies 2m > d+1. If 2m < d+1 then this routine finds the smallest m giving visual smoothness for the given d, otherwise the supplied m is used. The null space dimension is given by:

```
M = (m+d-1)!/(d!(m-1)!)
```

which is the value returned.

#### Value

An integer (array), the null space dimension M.

ocat 131

#### Author(s)

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

#### References

```
Wood, S.N. (2003) Thin plate regression splines. J.R.Statist.Soc.B 65(1):95-114 
http://www.maths.bris.ac.uk/~sw15190/
```

#### See Also

tprs

## **Examples**

```
require(mgcv)
null.space.dimension(2,0)
```

ocat

GAM ordered categorical family

## Description

Family for use with gam, implementing regression for ordered categorical data. A linear predictor provides the expected value of a latent variable following a logistic distribution. The probability of this latent variable lying between certain cut-points provides the probability of the ordered categorical variable being of the corresponding category. The cut-points are estimated along side the model smoothing parameters (using the same criterion). The observed categories are coded 1, 2, 3, ... up to the number of categories.

## Usage

```
ocat(theta=NULL,link="identity",R=NULL)
```

## **Arguments**

theta	cut point parameter vector (dimension R-2). If supplied and all positive, then taken to be the cut point increments (first cut point is fixed at -1). If any are negative then absolute values are taken as starting values for cutpoint increments.
link	The link function: only "identity" allowed at present (possibly for ever).
R	the number of catergories.

#### **Details**

Such cumulative threshold models are only identifiable up to an intercept, or one of the cut points. Rather than remove the intercept, ocat simply sets the first cut point to -1. Use predict.gam with type="response" to get the predicted probabilities in each category.

132 ocat

## Value

An object of class extended.family.

## Author(s)

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

## References

Wood, S.N., N. Pya and B. Saefken (2015), Smoothing parameter and model selection for general smooth models. http://arxiv.org/abs/1511.03864

```
library(mgcv)
## Simulate some ordered categorical data...
set.seed(3);n<-400
dat <- gamSim(1,n=n)</pre>
dat$f <- dat$f - mean(dat$f)</pre>
alpha <- c(-Inf,-1,0,5,Inf)
R <- length(alpha)-1
v <- dat$f
u <- runif(n)</pre>
u \leftarrow dat f + log(u/(1-u))
for (i in 1:R) {
  y[u > alpha[i]&u \le alpha[i+1]] <- i
dat$y <- y
## plot the data...
par(mfrow=c(2,2))
with(dat,plot(x0,y));with(dat,plot(x1,y))
with(dat,plot(x2,y));with(dat,plot(x3,y))
## fit ocat model to data...
b \leftarrow gam(y \sim s(x0) + s(x1) + s(x2) + s(x3), family = ocat(R=R), data = dat)
plot(b,pages=1)
gam.check(b)
summary(b)
b$family$getTheta(TRUE) ## the estimated cut points
## predict probabilities of being in each category
predict(b,dat[1:2,],type="response",se=TRUE)
```

pcls

pcls

Penalized Constrained Least Squares Fitting

## **Description**

Solves least squares problems with quadratic penalties subject to linear equality and inequality constraints using quadratic programming.

## Usage

pcls(M)

## **Arguments**

Μ

is the single list argument to pcls. It should have the following elements:

- y The response data vector.
- **w** A vector of weights for the data (often proportional to the reciprocal of the variance).
- X The design matrix for the problem, note that ncol(M\$X) must give the number of model parameters, while nrow(M\$X) should give the number of data.
- ${f C}$  Matrix containing any linear equality constraints on the problem (e.g.  ${f C}$  in  ${f Cp}={f c}$ ). If you have no equality constraints initialize this to a zero by zero matrix. Note that there is no need to supply the vector  ${f c}$ , it is defined implicitly by the initial parameter estimates  ${f p}$ .
- S A list of penalty matrices. S[[i]] is the smallest contiguous matrix including all the non-zero elements of the ith penalty matrix. The first parameter it penalizes is given by off[i]+1 (starting counting at 1).
- **off** Offset values locating the elements of M\$S in the correct location within each penalty coefficient matrix. (Zero offset implies starting in first location)
- **sp** An array of smoothing parameter estimates.
- p An array of feasible initial parameter estimates these must satisfy the constraints, but should avoid satisfying the inequality constraints as equality constraints.

**Ain** Matrix for the inequality constraints  $A_{in}p > b_{in}$ .

bin vector in the inequality constraints.

#### **Details**

This solves the problem:

$$minimise^{\sim} \|\mathbf{W}^{1/2}(\mathbf{X}\mathbf{p} - \mathbf{y})\|^2 + \sum_{i=1}^m \lambda_i \mathbf{p}' \mathbf{S}_i \mathbf{p}$$

subject to constraints  $C\mathbf{p} = \mathbf{c}$  and  $\mathbf{A}_{in}\mathbf{p} > \mathbf{b}_{in}$ , w.r.t.  $\mathbf{p}$  given the smoothing parameters  $\lambda_i$ .  $\mathbf{X}$  is a design matrix,  $\mathbf{p}$  a parameter vector,  $\mathbf{y}$  a data vector,  $\mathbf{W}$  a diagonal weight matrix,  $\mathbf{S}_i$  a positive

pcls

semi-definite matrix of coefficients defining the ith penalty and C a matrix of coefficients defining the linear equality constraints on the problem. The smoothing parameters are the  $\lambda_i$ . Note that X must be of full column rank, at least when projected into the null space of any equality constraints.  $A_{in}$  is a matrix of coefficients defining the inequality constraints, while  $b_{in}$  is a vector involved in defining the inequality constraints.

Quadratic programming is used to perform the solution. The method used is designed for maximum stability with least squares problems: i.e. X'X is not formed explicitly. See Gill et al. 1981.

#### Value

The function returns an array containing the estimated parameter vector.

### Author(s)

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

#### References

Gill, P.E., Murray, W. and Wright, M.H. (1981) Practical Optimization. Academic Press, London.

Wood, S.N. (1994) Monotonic smoothing splines fitted by cross validation SIAM Journal on Scientific Computing 15(5):1126-1133

```
http://www.maths.bris.ac.uk/~sw15190/
```

#### See Also

```
magic, mono.con
```

```
require(mgcv)
# first an un-penalized example - fit E(y)=a+bx subject to a>0
set.seed(0)
n <- 100
x \leftarrow runif(n); y \leftarrow x - 0.2 + rnorm(n)*0.1
M \leftarrow list(X=matrix(0,n,2),p=c(0.1,0.5),off=array(0,0),S=list(),
Ain=matrix(0,1,2), bin=0, C=matrix(0,0,0), sp=array(0,0), y=y, w=y*0+1)
M$X[,1] <- 1; M$X[,2] <- x; M$Ain[1,] <- c(1,0)
pcls(M) \rightarrow Mp
plot(x,y); abline(M$p,col=2); abline(coef(lm(y~x)),col=3)
# Penalized example: monotonic penalized regression spline .....
# Generate data from a monotonic truth.
x \leftarrow runif(100)*4-1; x \leftarrow sort(x);
f \leftarrow \exp(4*x)/(1+\exp(4*x)); y \leftarrow f+rnorm(100)*0.1; plot(x,y)
dat <- data.frame(x=x,y=y)</pre>
# Show regular spline fit (and save fitted object)
f.ug <- gam(y \sim (x,k=10,bs="cr")); lines(x,fitted(f.ug))
# Create Design matrix, constraints etc. for monotonic spline....
sm <- smoothCon(s(x,k=10,bs="cr"),dat,knots=NULL)[[1]]</pre>
```

pcls 135

```
F <- mono.con(sm$xp); # get constraints
G \leftarrow list(X=sm$X,C=matrix(0,0,0),sp=f.ug$sp,p=sm$xp,y=y,w=y*0+1)
G$Ain \leftarrow F$A;G$bin \leftarrow F$b;G$S \leftarrow sm$S;G$off \leftarrow 0
p <- pcls(G); # fit spline (using s.p. from unconstrained fit)</pre>
fv<-Predict.matrix(sm,data.frame(x=x))%*%p</pre>
lines(x,fv,col=2)
# now a tprs example of the same thing....
f.ug <- gam(y\sim s(x,k=10)); lines(x,fitted(f.ug))
# Create Design matrix, constriants etc. for monotonic spline....
sm <- smoothCon(s(x,k=10,bs="tp"),dat,knots=NULL)[[1]]</pre>
xc <- 0:39/39 # points on [0,1]
nc <- length(xc) # number of constraints</pre>
xc <- xc*4-1 # points at which to impose constraints
A0 <- Predict.matrix(sm,data.frame(x=xc))
# ... A0%*%p evaluates spline at xc points
A1 <- Predict.matrix(sm,data.frame(x=xc+1e-6))
A <- (A1-A0)/1e-6
## ... approx. constraint matrix (A%*%p is -ve
## spline gradient at points xc)
G \leftarrow list(X=sm$X,C=matrix(0,0,0),sp=f.ug$sp,y=y,w=y*0+1,S=sm$S,off=0)
G$Ain <- A; # constraint matrix
G$bin <- rep(0,nc); # constraint vector</pre>
G$p \leftarrow rep(0,10); G$p[10] \leftarrow 0.1
# ... monotonic start params, got by setting coefs of polynomial part
p <- pcls(G); # fit spline (using s.p. from unconstrained fit)</pre>
fv2 <- Predict.matrix(sm,data.frame(x=x))%*%p</pre>
lines(x,fv2,col=3)
## monotonic additive model example...
## First simulate data...
set.seed(10)
f1 \leftarrow function(x) 5*exp(4*x)/(1+exp(4*x));
f2 <- function(x) {</pre>
  ind <- x > .5
  f <- x*0
  f[ind] \leftarrow (x[ind] - .5)^2*10
  f
f3 \leftarrow function(x) 0.2 * x^11 * (10 * (1 - x))^6 +
      10 * (10 * x)^3 * (1 - x)^10
n <- 200
x <- runif(n); z <- runif(n); v <- runif(n)</pre>
mu \leftarrow f1(x) + f2(z) + f3(v)
y \leftarrow mu + rnorm(n)
```

pdIdnot

```
## Preliminary unconstrained gam fit...
G \leftarrow gam(y \sim s(x) + s(z) + s(v, k=20), fit=FALSE)
b \leftarrow gam(G=G)
## generate constraints, by finite differencing
## using predict.gam ....
eps <- 1e-7
pd0 <- data.frame(x=seq(0,1,length=100),z=rep(.5,100),</pre>
                   v=rep(.5,100)
pd1 <- data.frame(x=seq(0,1,length=100)+eps,z=rep(.5,100),</pre>
                   v=rep(.5,100))
X0 <- predict(b,newdata=pd0,type="lpmatrix")</pre>
X1 <- predict(b,newdata=pd1,type="lpmatrix")</pre>
Xx \leftarrow (X1 - X0)/eps \# Xx \% coef(b) must be positive
pd0 <- data.frame(z=seq(0,1,length=100),x=rep(.5,100),</pre>
                   v=rep(.5,100))
pd1 <- data.frame(z=seq(0,1,length=100)+eps,x=rep(.5,100),
                   v=rep(.5,100))
X0 <- predict(b,newdata=pd0,type="lpmatrix")</pre>
X1 <- predict(b,newdata=pd1,type="lpmatrix")</pre>
Xz <- (X1-X0)/eps
G$Ain <- rbind(Xx,Xz) ## inequality constraint matrix
G$bin <- rep(0,nrow(G$Ain))</pre>
GC = matrix(0,0,ncol(GX))
G$sp <- b$sp
G$p <- coef(b)
G$off <- G$off-1 ## to match what pcls is expecting
## force inital parameters to meet constraint
Gp[11:18] \leftarrow Gp[2:9] \leftarrow 0
p <- pcls(G) ## constrained fit
par(mfrow=c(2,3))
plot(b) ## original fit
b$coefficients <- p
plot(b) ## constrained fit
## note that standard errors in preceding plot are obtained from
## unconstrained fit
```

pdIdnot

Overflow proof pdMat class for multiples of the identity matrix

## **Description**

This set of functions is a modification of the pdMat class pdIdent from library nlme. The modification is to replace the log parameterization used in pdMat with a notLog2 parameterization, since the latter avoids indefiniteness in the likelihood and associated convergence problems: the parameters also relate to variances rather than standard deviations, for consistency with the pdTens class. The functions are particularly useful for working with Generalized Additive Mixed Models

pdIdnot 137

where variance parameters/smoothing parameters can be very large or very small, so that overflow or underflow can be a problem.

These functions would not normally be called directly, although unlike the pdTens class it is easy to do so.

## Usage

## **Arguments**

value Initialization values for parameters. Not normally used.

form A one sided formula specifying the random effects structure.

nam a names argument, not normally used with this class.

data frame in which to evaluate formula.

#### **Details**

The following functions are provided: Dim.pdIndot, coef.pdIdnot, corMatrix.pdIdnot, logDet.pdIdnot, pdConstruct.pdIdnot, pdFactor.pdIdnot, pdMatrix.pdIdnot, solve.pdIdnot, summary.pdIdnot. (e.g. mgcv:::coef.pdIdnot to access.)

Note that while the pdFactor and pdMatrix functions return the inverse of the scaled random effect covariance matrix or its factor, the pdConstruct function is initialised with estimates of the scaled covariance matrix itself.

#### Value

A class pdIdnot object, or related quantities. See the nlme documentation for further details.

## Author(s)

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

#### References

Pinheiro J.C. and Bates, D.M. (2000) Mixed effects Models in S and S-PLUS. Springer The nlme source code.

```
http://www.maths.bris.ac.uk/~sw15190/
```

### See Also

```
te, pdTens, notLog2, gamm
```

```
# see gamm
```

pdTens

pdTens	Functions implementing a pdMat class for tensor product smooths
	· · · · · · · · · · · · · · · ·

## **Description**

This set of functions implements an nlme library pdMat class to allow tensor product smooths to be estimated by lme as called by gamm. Tensor product smooths have a penalty matrix made up of a weighted sum of penalty matrices, where the weights are the smoothing parameters. In the mixed model formulation the penalty matrix is the inverse of the covariance matrix for the random effects of a term, and the smoothing parameters (times a half) are variance parameters to be estimated. It's not possible to transform the problem to make the required random effects covariance matrix look like one of the standard pdMat classes: hence the need for the pdTens class. A notLog2 parameterization ensures that the parameters are positive.

These functions (pdTens, pdConstruct.pdTens, pdFactor.pdTens, pdMatrix.pdTens, coef.pdTens and summary.pdTens) would not normally be called directly.

## Usage

### **Arguments**

ralue Initialization values for parameters. Not normally used.

A one sided formula specifying the random effects structure. The formula should have an attribute S which is a list of the penalty matrices the weighted sum of which gives the inverse of the covariance matrix for these random effects.

a names argument, not normally used with this class.

# data frame in which to evaluate formula.

## **Details**

If using this class directly note that it is worthwhile scaling the S matrices to be of 'moderate size', for example by dividing each matrix by its largest singular value: this avoids problems with lme defaults (smooth.construct.tensor.smooth.spec does this automatically).

This appears to be the minimum set of functions required to implement a new pdMat class.

Note that while the pdFactor and pdMatrix functions return the inverse of the scaled random effect covariance matrix or its factor, the pdConstruct function is sometimes initialised with estimates of the scaled covariance matrix, and sometimes intialized with its inverse.

### Value

A class pdTens object, or its coefficients or the matrix it represents or the factor of that matrix. pdFactor returns the factor as a vector (packed column-wise) (pdMatrix always returns a matrix).

pen.edf 139

## Author(s)

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

#### References

Pinheiro J.C. and Bates, D.M. (2000) Mixed effects Models in S and S-PLUS. Springer The nlme source code.

```
http://www.maths.bris.ac.uk/~sw15190/
```

## See Also

te gamm

## **Examples**

# see gamm

pen.edf

Extract the effective degrees of freedom associated with each penalty in a gam fit

## Description

Finds the coefficients penalized by each penalty and adds up their effective degrees of freedom. Very useful for t2 terms, but hard to interpret for terms where the penalties penalize overlapping sets of parameters (e.g. te terms).

## Usage

```
pen.edf(x)
```

#### **Arguments**

Х

an object inheriting from gam

### **Details**

Useful for models containing t2 terms, since it splits the EDF for the term up into parts due to different components of the smooth. This is useful for figuring out which interaction terms are actually needed in a model.

## Value

A vector of EDFs, named with labels identifying which penalty each EDF relates to.

#### Author(s)

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

140 place.knots

## See Also

t2

## **Examples**

```
require(mgcv)
set.seed(20)
dat <- gamSim(1,n=400,scale=2) ## simulate data</pre>
## following `t2' smooth basically separates smooth
## of x0,x1 into main effects + interaction....
b < -gam(y^2t^2(x^0, x^1, bs="tp", m=1, k=7) + s(x^2) + s(x^3),
         data=dat,method="ML")
pen.edf(b)
## label "rr" indicates interaction edf (range space times range space)
## label "nr" (null space for x0 times range space for x1) is main
              effect for x1.
## label "rn" is main effect for x0
## clearly interaction is negligible
## second example with higher order marginals.
b \leftarrow gam(y\sim t2(x0,x1,bs="tp",m=2,k=7,full=TRUE)
           +s(x2)+s(x3),data=dat,method="ML")
pen.edf(b)
## In this case the EDF is negligible for all terms in the t2 smooth
## apart from the `main effects' (r2 and 2r). To understand the labels
## consider the following 2 examples....
## "r1" relates to the interaction of the range space of the first
        marginal smooth and the first basis function of the null
        space of the second marginal smooth
##
## "2r" relates to the interaction of the second basis function of
##
        the null space of the first marginal smooth with the range
##
        space of the second marginal smooth.
```

place.knots

Automatically place a set of knots evenly through covariate values

## **Description**

Given a univariate array of covariate values, places a set of knots for a regression spline evenly through the covariate values.

## Usage

```
place.knots(x,nk)
```

## Arguments

x array of covariate values (need not be sorted).

nk integer indicating the required number of knots.

## **Details**

Places knots evenly throughout a set of covariates. For example, if you had 11 covariate values and wanted 6 knots then a knot would be placed at the first (sorted) covariate value and every second (sorted) value thereafter. With less convenient numbers of data and knots the knots are placed within intervals between data in order to achieve even coverage, where even means having approximately the same number of data between each pair of knots.

## Value

An array of knot locations.

### Author(s)

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

#### References

```
http://www.maths.bris.ac.uk/~sw15190/
```

### See Also

```
smooth.construct.cc.smooth.spec
```

## **Examples**

```
require(mgcv)
x<-runif(30)
place.knots(x,7)
rm(x)</pre>
```

plot.gam

Default GAM plotting

## **Description**

Takes a fitted gam object produced by gam() 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

## Arguments

n

x 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.

Setting to FALSE will speed up plotting for large datasets.

se when TRUE (default) upper and lower lines are added to the 1-d plots at 2 stan-

dard 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.

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 con-

touring.

pers Set to TRUE if you want perspective plots for 2-d terms.

theta One of the perspective plot angles.

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. Setting to zero can make plotting faster for large datasets, but care then needed with interpretation of plots.
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. Marra and Wood (2012) 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 or integer vector selecting a plotting scheme for each plot. See details.
	other graphics parameters to pass on to plotting commands. See details for smooth plot specific options.

## **Details**

Produces default plot showing the smooth components of a fitted GAM, and optionally parametric terms as well, when these can be handled by termplot.

For smooth terms plot.gam actually calls plot method functions depending on the class of the smooth. Currently random.effects, Markov random fields (mrf), Spherical.Spline and factor.smooth.interaction terms have special methods (documented in their help files), the rest use the defaults described below.

For plots of 1-d smooths, the x axis of each plot is labelled with the covariate name, while the y axis is labelled s(cov,edf) where cov is the covariate name, and edf the estimated (or user defined

for regression splines) degrees of freedom of the smooth. 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. The main title of the plot is something like s(var1, var2, edf), indicating the variables of which the term is a function, and the estimated degrees of freedom for the term. When se=TRUE, estimator variability is shown by overlaying contour plots at plus and minus 1 s.e. relative to the main estimate. If se is a positive number then contour plots are at plus or minus se multiplied by the s.e. Contour levels are chosen to try and ensure reasonable separation of the contours of the different plots, but this is not always easy to achieve. Note that these plots can not be modified to the same extent as the other plot.

For 2-d smooths scheme==1 produces a perspective plot, while scheme==2 produces a heatmap, with overlaid contours.

Smooths of more than 2 variables are not plotted, but see vis.gam.

Fine control of plots for parametric terms can be obtained by calling termplot directly, taking care to use its terms argument.

Note that, if seWithMean=TRUE, the confidence bands include the uncertainty about the overall mean. In other words although each smooth is shown centred, the confidence bands are obtained as if every other term in the model was constrained to have average 0, (average taken over the covariate values), except for the smooth concerned. This seems to correspond more closely to how most users interpret componentwise intervals in practice, and also results in intervals with close to nominal (frequentist) coverage probabilities by an extension of Nychka's (1988) results presented in Marra and Wood (2012).

Several smooth plots methods using image will accept a colors argument, which can be anything documented in heat.colors (in which case something like colors=rainbow(50) is appropriate), or the grey function (in which case somthing like colors=grey(0:50/50) is needed). Another option is contour.col which will set the contour colour for some plots. These options are useful for producing grey scale pictures instead of colour.

Sometimes you may want a small change to a default plot, and the arguments to plot.gam just won't let you do it. In this case, the quickest option is sometimes to clone the smooth.construct and Predict.matrix methods for the smooth concerned, modifying only the returned smoother class (e.g. to foo.smooth). Then copy the plot method function for the original class (e.g. mgcv:::plot.mgcv.smooth), modify the source code to plot exactly as you want and rename the plot method function (e.g. plot.foo.smooth). You can then use the cloned smooth in models (e.g. s(x,bs="foo")), and it will automatically plot using the modified plotting function.

## Value

The functions main purpose is its side effect of generating plots. It also silently returns a list of the data used to produce the plots, which can be used to generate customized plots.

### WARNING

Note that the behaviour of this function is not identical to plot.gam() in S-PLUS.

Plotting can be slow for models fitted to large datasets. Set rug=FALSE to improve matters. If it's still too slow set too. far=0, but then take care not to overinterpret smooths away from supporting data.

plot.gam 145

Plots of 2-D smooths with standard error contours shown can not easily be customized.

The function can not deal with smooths of more than 2 variables!

## Author(s)

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

Henric Nilsson <a href="henric.nilsson@statisticon.se">henric.nilsson@statisticon.se</a>> donated the code for the shade option.

The design is inspired by the S function of the same name described in Chambers and Hastie (1993) (but is not a clone).

#### References

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

Marra, G and S.N. Wood (2012) Coverage Properties of Confidence Intervals for Generalized Additive Model Components. Scandinavian Journal of Statistics.

Nychka (1988) Bayesian Confidence Intervals for Smoothing Splines. Journal of the American Statistical Association 83:1134-1143.

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

#### See Also

```
gam, predict.gam, vis.gam
```

```
library(mgcv)
set.seed(0)
## fake some data...
f1 \leftarrow function(x) \{exp(2 * x)\}
f2 <- function(x) {</pre>
  0.2*x^{11}*(10*(1-x))^6+10*(10*x)^3*(1-x)^10
f3 \leftarrow function(x) \{x*0\}
n<-200
sig2<-4
x0 < -rep(1:4,50)
x1 <- runif(n, 0, 1)
x2 <- runif(n, 0, 1)
x3 <- runif(n, 0, 1)
e <- rnorm(n, 0, sqrt(sig2))
y \leftarrow 2*x0 + f1(x1) + f2(x2) + f3(x3) + e
x0 <- factor(x0)
## fit and plot...
b < -gam(y \sim x0 + s(x1) + s(x2) + s(x3))
plot(b,pages=1,residuals=TRUE,all.terms=TRUE,shade=TRUE,shade.col=2)
plot(b,pages=1,seWithMean=TRUE) ## better coverage intervals
```

polys.plot

```
## just parametric term alone...
termplot(b,terms="x0",se=TRUE)
## more use of color...
op <- par(mfrow=c(2,2),bg="blue")</pre>
x <- 0:1000/1000
for (i in 1:3) {
  plot(b, select=i, rug=FALSE, col="green",
    col.axis="white",col.lab="white",all.terms=TRUE)
  for (j in 1:2) axis(j,col="white",labels=FALSE)
  box(col="white")
  eval(parse(text=paste("fx <- f",i,"(x)",sep="")))</pre>
  fx \leftarrow fx-mean(fx)
  lines(x,fx,col=2) ## overlay `truth' in red
}
par(op)
## example with 2-d plots, and use of schemes...
b1 <- gam(y \sim x0 + s(x1, x2) + s(x3))
op <- par(mfrow=c(2,2))
plot(b1,all.terms=TRUE)
par(op)
op <- par(mfrow=c(2,2))</pre>
plot(b1,all.terms=TRUE,scheme=1)
par(op)
op <- par(mfrow=c(2,2))
plot(b1,all.terms=TRUE,scheme=c(2,1))
par(op)
```

polys.plot

Plot geographic regions defined as polygons

## **Description**

Produces plots of geographic regions defined by polygons, optionally filling the polygons with a color or grey shade dependent on a covariate.

#### Usage

```
polys.plot(pc,z=NULL,scheme="heat",lab="",...)
```

## **Arguments**

рс

A named list of matrices. Each matrix has two columns. The matrix rows each define the vertex of a boundary polygon. If a boundary is defined by several polygons, then each of these must be separated by an NA row in the matrix. See mrf for an example.

Z	A vector of values associated with each area (item) of pc. If the vector elements have names then these are used to match elements of z to areas defined in pc. Otherwise pc and z are assumed to be in the same order. If z is NULL then polygons are not filled.
scheme	One of "heat" or "grey", indicating how to fill the polygons in accordance with the value of z.
lab	label for plot.
	other arguments to pass to plot (currently only if z is NULL).

## **Details**

Any polygon within another polygon counts as a hole in the area. Further nesting is dealt with by treating any point that is interior to an odd number of polygons as being within the area, and all other points as being exterior. The routine is provided to facilitate plotting with models containing mrf smooths.

#### Value

Simply produces a plot.

## Author(s)

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

#### See Also

```
mrf and columb.polys.
```

# Examples

```
## see also ?mrf for use of z
require(mgcv)
data(columb.polys)
polys.plot(columb.polys)
```

predict.bam

Prediction from fitted Big Additive Model model

## **Description**

Essentially a wrapper for predict. gam for prediction from a model fitted by bam. Can compute on a parallel cluster.

Takes a fitted bam object produced by bam 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. The routine can optionally return the matrix by which the model coefficients must be pre-multiplied in order to yield the values of the linear predictor at the supplied covariate values: this is useful for obtaining credible regions for quantities derived from the model (e.g. derivatives of smooths), and for lookup table prediction outside R (see example code below).

#### Usage

#### **Arguments**

object a fitted bam object as produced by bam.

newdata A data frame or list containing the values of the model covariates at which pre-

dictions 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 as-

sociated 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 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

below).

se.fit when this is TRUE (not default) standard error estimates are returned for each

prediction.

terms if type="terms" or type="iterms" then only results for the terms (smooth or

parametric) named in this array will be returned. Otherwise any smooth terms not named in this array will be set to zero. If NULL then all terms are included.

also be used to implement approximate prediction outside R (see example code,

exclude if type=="terms" or type="iterms" then terms (smooth or parametric) named

in this array will not be returned. Otherwise any smooth terms named in this

array will be set to zero. If NULL then no terms are excluded.

block.size maximum number of predictions to process per call to underlying code: larger

is quicker, but more memory intensive.

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.

cluster predict.bam can compute in parallel using parLapply from the parallel pack-

age, if it is supplied with a cluster on which to do this (a cluster here can be some

cores of a single machine). See details and example code for bam.

discrete if TRUE then discrete prediction methods used with model fitted by discrete meth-

ods. FALSE for regular prediction.

if se.fit=TRUE and discrete prediction is used then parallel computation can be n.threads

used to speed up se calcualtion. This specifies number of htreads to use.

other arguments.

#### **Details**

The standard errors produced by predict.gam are based on the Bayesian posterior covariance matrix of the parameters Vp in the fitted bam object.

To facilitate plotting with termplot, if object possesses an attribute "para.only" and type=="terms" then only parametric terms of order 1 are returned (i.e. those that termplot can handle).

Note that, in common with other prediction functions, any offset supplied to gam as an argument is always ignored when predicting, unlike offsets specified in the gam model formula.

See the examples in predict.gam for how to use the lpmatrix for obtaining credible regions for quantities derived from the model.

## 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)

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

The design is inspired by the S function of the same name described in Chambers and Hastie (1993) (but is not a clone).

#### References

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

Marra, G and S.N. Wood (2012) Coverage Properties of Confidence Intervals for Generalized Additive Model Components. Scandinavian Journal of Statistics.

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

#### See Also

```
bam, predict.gam
```

## **Examples**

```
## for parallel computing see examples for ?bam
## for general useage follow examples in ?predict.gam
```

predict.gam

Prediction from fitted GAM model

# **Description**

Takes a fitted gam object produced by gam() 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. The routine can optionally return the matrix by which the model coefficients must be pre-multiplied in order to yield the values of the linear predictor at the supplied covariate values: this is useful for obtaining credible regions for quantities derived from the model (e.g. derivatives of smooths), and for lookup table prediction outside R (see example code below).

## Usage

#### **Arguments**

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

newdata A data frame or list containing the values of the model covariates at which pre-

dictions 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 as-

sociated 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 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,

se.fit when this is TRUE (not default) standard error estimates are returned for each

prediction.

if type=="terms" or type="iterms" then only results for the terms (smooth or terms parametric) named in this array will be returned. Otherwise any smooth terms

not named in this array will be set to zero. If NULL then all terms are included.

if type=="terms" or type="iterms" then terms (smooth or parametric) named exclude in this array will not be returned. Otherwise any smooth terms named in this

array will be set to zero. If NULL then no terms are excluded.

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. If NULL then block size is 1000 if new data supplied, and the number of

rows in the model frame otherwise.

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.

what to do about NA values in newdata. With the default na.pass, any row na.action

> 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.

unconditional if TRUE then the smoothing parameter uncertainty corrected covariance matrix

is used, when available, otherwise the covariance matrix conditional on the esti-

mated smoothing parameters is used.

other arguments.

#### **Details**

The standard errors produced by predict.gam are based on the Bayesian posterior covariance matrix of the parameters Vp in the fitted gam object.

To facilitate plotting with termplot, if object possesses an attribute "para.only" and type=="terms" then only parametric terms of order 1 are returned (i.e. those that termplot can handle).

Note that, in common with other prediction functions, any offset supplied to gam as an argument is always ignored when predicting, unlike offsets specified in the gam model formula.

See the examples for how to use the lpmatrix for obtaining credible regions for quantities derived from the model.

#### Value

If type=="1pmatrix" 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.

## WARNING

Note that the behaviour of this function is not identical to predict.gam() in Splus.

type=="terms" does not exactly match what predict.lm does for parametric model components.

#### Author(s)

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

The design is inspired by the S function of the same name described in Chambers and Hastie (1993) (but is not a clone).

#### References

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

Marra, G and S.N. Wood (2012) Coverage Properties of Confidence Intervals for Generalized Additive Model Components. Scandinavian Journal of Statistics, 39(1), 53-74.

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

#### See Also

```
gam, gamm, plot.gam
```

```
## difference between "terms" and "iterms"
nd2 \leftarrow data.frame(x0=c(.25,.5),x1=c(.25,.5),x2=c(.25,.5),x3=c(.25,.5))
predict(b,nd2,type="terms",se=TRUE)
predict(b,nd2,type="iterms",se=TRUE)
## now get variance of sum of predictions using lpmatrix
Xp <- predict(b,newd,type="lpmatrix")</pre>
## Xp %*% coef(b) yields vector of predictions
a < - rep(1,31)
Xs <- t(a) %*% Xp ## Xs %*% coef(b) gives sum of predictions
var.sum <- Xs %*% b$Vp %*% t(Xs)</pre>
## Now get the variance of non-linear function of predictions
## by simulation from posterior distribution of the params
rmvn <- function(n,mu,sig) { ## MVN random deviates</pre>
 L <- mroot(sig);m <- ncol(L);
 t(mu + L%*%matrix(rnorm(m*n),m,n))
}
br <- rmvn(1000,coef(b),b$Vp) ## 1000 replicate param. vectors</pre>
res <- rep(0,1000)
for (i in 1:1000)
{ pr <- Xp %*% br[i,] ## replicate predictions
 res[i] <- sum(log(abs(pr))) ## example non-linear function</pre>
mean(res); var(res)
## loop is replace-able by following ....
res <- colSums(log(abs(Xp %*% t(br))))
## The following shows how to use use an "lpmatrix" as a lookup
## table for approximate prediction. The idea is to create
## approximate prediction matrix rows by appropriate linear
## interpolation of an existing prediction matrix. The additivity
## of a GAM makes this possible.
## There is no reason to ever do this in R, but the following
## code provides a useful template for predicting from a fitted
## gam *outside* R: all that is needed is the coefficient vector
## and the prediction matrix. Use larger `Xp'/ smaller `dx' and/or
## higher order interpolation for higher accuracy.
```

```
xn <- c(.341,.122,.476,.981) ## want prediction at these values
x0 <- 1
           ## intercept column
dx <- 1/30
              ## covariate spacing in `newd'
for (j in 0:2) { ## loop through smooth terms
 cols <- 1+j*9 +1:9
                     ## relevant cols of Xp
 i <- floor(xn[j+1]*30) ## find relevant rows of Xp</pre>
 w1 <- (xn[j+1]-i*dx)/dx ## interpolation weights
 ## find approx. predict matrix row portion, by interpolation
 x0 \leftarrow c(x0, Xp[i+2, cols]*w1 + Xp[i+1, cols]*(1-w1))
dim(x0) < -c(1,28)
fv <- x0%*%coef(b) + xn[4];fv
                                ## evaluate and add offset
se <- sqrt(x0%*%b$Vp%*%t(x0));se ## get standard error
## compare to normal prediction
predict(b, newdata=data.frame(x0=xn[1],x1=xn[2],
       x2=xn[3], x3=xn[4]), se=TRUE)
## Differentiating the smooths in a model (with CIs for derivatives)
## simulate data and fit model...
dat <- gamSim(1,n=300,scale=sig)</pre>
b < -gam(y \sim s(x0) + s(x1) + s(x2) + s(x3), data = dat)
plot(b,pages=1)
## now evaluate derivatives of smooths with associated standard
## errors, by finite differencing...
x.mesh <- seq(0,1,length=200) ## where to evaluate derivatives
newd <- data.frame(x0 = x.mesh, x1 = x.mesh, x2=x.mesh, x3=x.mesh)</pre>
X0 <- predict(b,newd,type="lpmatrix")</pre>
eps <- 1e-7 ## finite difference interval
x.mesh <- x.mesh + eps ## shift the evaluation mesh
newd <- data.frame(x0 = x.mesh, x1 = x.mesh, x2=x.mesh, x3=x.mesh)
X1 <- predict(b,newd,type="lpmatrix")</pre>
Xp <- (X1-X0)/eps ## maps coefficients to (fd approx.) derivatives
             ## can check which cols relate to which smooth
colnames(Xp)
par(mfrow=c(2,2))
for (i in 1:4) { ## plot derivatives and corresponding CIs
 Xi <- Xp*0
 Xi[,(i-1)*9+1:9+1] \leftarrow Xp[,(i-1)*9+1:9+1] \# Xi\%*coef(b) = smooth deriv i
 df <- Xi%*%coef(b)</pre>
                                ## ith smooth derivative
 df.sd <- rowSums(Xi%*%b$Vp*Xi)^.5 ## cheap diag(Xi%*%b$Vp%*%t(Xi))^.5
 plot(x.mesh,df,type="1",ylim=range(c(df+2*df.sd,df-2*df.sd)))
 lines(x.mesh, df+2*df.sd, lty=2); lines(x.mesh, df-2*df.sd, lty=2)
}
```

Predict.matrix 155

#### **Description**

Takes smooth objects produced by smooth.construct methods and obtains the matrix mapping the parameters associated with such a smooth to the predicted values of the smooth at a set of new covariate values.

In practice this method is often called via the wrapper function PredictMat.

#### Usage

```
Predict.matrix(object,data)
Predict.matrix2(object,data)
```

# **Arguments**

object is a smooth object produced by a smooth construct method function. The

object contains all the information required to specify the basis for a term of its class, and this information is used by the appropriate Predict.matrix function to produce a prediction matrix for new covariate values. Further details are given

in smooth.construct.

data A data frame containing the values of the (named) covariates at which the smooth

term is to be evaluated. Exact requirements are as for smooth.construct and

smooth.construct2.

## **Details**

Smooth terms in a GAM formula are turned into smooth specification objects of class xx.smooth.spec during processing of the formula. Each of these objects is converted to a smooth object using an appropriate smooth.construct function. The Predict.matrix functions are used to obtain the matrix that will map the parameters associated with a smooth term to the predicted values for the term at new covariate values.

Note that new smooth classes can be added by writing a new smooth.construct method function and a corresponding Predict.matrix method function: see the example code provided for smooth.construct for details.

#### Value

A matrix which will map the parameters associated with the smooth to the vector of values of the smooth evaluated at the covariate values given in object. If the smooth class is one which generates offsets the corresponding offset is returned as attribute "offset" of the matrix.

#### Author(s)

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

156 Predict.matrix.cr.smooth

#### References

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

#### See Also

```
gam,gamm, smooth.construct, PredictMat
```

# **Examples**

```
# See smooth.construct examples
```

```
Predict.matrix.cr.smooth
```

Predict matrix method functions

## **Description**

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

#### Usage

```
## S3 method for class 'cr.smooth'
Predict.matrix(object, data)
## S3 method for class 'cs.smooth'
Predict.matrix(object, data)
## S3 method for class 'cyclic.smooth'
Predict.matrix(object, data)
## S3 method for class 'pspline.smooth'
Predict.matrix(object, data)
## S3 method for class 'tensor.smooth'
Predict.matrix(object, data)
## S3 method for class 'tprs.smooth'
Predict.matrix(object, data)
## S3 method for class 'ts.smooth'
Predict.matrix(object, data)
## S3 method for class 't2.smooth'
Predict.matrix(object, data)
```

## Arguments

object a smooth object, usually generated by a smooth.construct method having pro-

cessed a smooth specification object generated by an s or te term in a gam for-

mula.

data A data frame containing the values of the (named) covariates at which the smooth

term is to be evaluated. Exact requirements are as for smooth.construct and

smooth.construct2.

#### **Details**

The Predict matrix function is not normally called directly, but is rather used internally by predict.gam etc. to predict from a fitted gam model. See Predict.matrix for more details, or the specific smooth.construct pages for details on a particular smooth class.

#### Value

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

#### Author(s)

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

#### References

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

## **Examples**

```
## see smooth.construct
```

```
Predict.matrix.soap.film
```

Prediction matrix for soap film smooth

## **Description**

Creates a prediction matrix for a soap film smooth object, mapping the coefficients of the smooth to the linear predictor component for the smooth. This is the Predict.matrix method function required by gam.

## Usage

```
## S3 method for class 'soap.film'
Predict.matrix(object,data)
## S3 method for class 'sw'
Predict.matrix(object,data)
## S3 method for class 'sf'
Predict.matrix(object,data)
```

## **Arguments**

```
object A class "soap.film", "sf" or "sw" object.
```

data A list list or data frame containing the arguments of the smooth at which predic-

tions are required.

#### **Details**

The smooth object will be largely what is returned from smooth.construct.so.smooth.spec, although elements X and S are not needed, and need not be present, of course.

#### Value

A matrix. This may have an "offset" attribute corresponding to the contribution from any known boundary conditions on the smooth.

## Author(s)

```
Simon N. Wood <s.wood@bath.ac.uk>
```

#### References

```
http://www.maths.bris.ac.uk/~sw15190/
```

#### See Also

```
smooth.construct.so.smooth.spec
```

```
## This is a lower level example. The basis and
## penalties are obtained explicitly
## and `magic' is used as the fitting routine...
require(mgcv)
set.seed(66)
## create a boundary...
fsb <- list(fs.boundary())</pre>
## create some internal knots...
knots <- data.frame(x=rep(seq(-.5,3,by=.5),4),
                      y=rep(c(-.6,-.3,.3,.6),rep(8,4)))
## Simulate some fitting data, inside boundary...
n<-1000
x \leftarrow runif(n)*5-1; y \leftarrow runif(n)*2-1
z \leftarrow fs.test(x,y,b=1)
ind <- inSide(fsb,x,y) ## remove outsiders</pre>
z \leftarrow z[ind]; x \leftarrow x[ind]; y \leftarrow y[ind]
n \leftarrow length(z)
z \leftarrow z + rnorm(n)*.3 \# add noise
## plot boundary with knot and data locations
plot(fsb[[1]]$x,fsb[[1]]$y,type="1");points(knots$x,knots$y,pch=20,col=2)
points(x,y,pch=".",col=3);
## set up the basis and penalties...
```

print.gam 159

```
sob <- smooth.construct2(s(x,y,bs="so",k=40,xt=list(bnd=fsb,nmax=100)),</pre>
               data=data.frame(x=x,y=y),knots=knots)
## ... model matrix is element `X' of sob, penalties matrices
## are in list element `S'.
## fit using `magic'
um <- magic(z, sob$X, sp=c(-1, -1), sob$S, off=c(1, 1))
beta <- um$b
## produce plots...
par(mfrow=c(2,2), mar=c(4,4,1,1))
m<-100; n<-50
xm \leftarrow seq(-1,3.5,length=m);yn \leftarrow seq(-1,1,length=n)
xx <- rep(xm,n);yy<-rep(yn,rep(m,n))</pre>
## plot truth...
tru <- matrix(fs.test(xx,yy),m,n) ## truth</pre>
image(xm,yn,tru,col=heat.colors(100),xlab="x",ylab="y")
lines(fsb[[1]]$x,fsb[[1]]$y,lwd=3)
contour(xm,yn,tru,levels=seq(-5,5,by=.25),add=TRUE)
## Plot soap, by first predicting on a fine grid...
## First get prediction matrix...
X <- Predict.matrix2(sob,data=list(x=xx,y=yy))</pre>
## Now the predictions...
fv <- X%*%beta
## Plot the estimated function...
image(xm,yn,matrix(fv,m,n),col=heat.colors(100),xlab="x",ylab="y")
lines(fsb[[1]]$x,fsb[[1]]$y,lwd=3)
points(x,y,pch=".")
contour(xm,yn,matrix(fv,m,n),levels=seq(-5,5,by=.25),add=TRUE)
## Plot TPRS...
b <- gam(z^s(x,y,k=100))
fv.gam <- predict(b,newdata=data.frame(x=xx,y=yy))</pre>
names(sob$sd$bnd[[1]]) <- c("xx","yy","d")</pre>
ind <- inSide(sob$sd$bnd,xx,yy)</pre>
fv.gam[!ind]<-NA</pre>
image(xm,yn,matrix(fv.gam,m,n),col=heat.colors(100),xlab="x",ylab="y")
lines(fsb[[1]]$x,fsb[[1]]$y,lwd=3)
points(x,y,pch=".")
contour(xm,yn,matrix(fv.gam,m,n),levels=seq(-5,5,by=.25),add=TRUE)
```

print.gam

## **Description**

The default print method for a gam object.

# Usage

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

## **Arguments**

x, ... fitted model objects of class gam as produced by gam().

#### **Details**

Prints out the family, model formula, effective degrees of freedom for each smooth term, and optimized value of the smoothness selection criterion used. See gamObject (or names(x)) for a listing of what the object contains. summary.gam provides more detail.

Note that the optimized smoothing parameter selection criterion reported is one of GCV, UBRE(AIC), GACV, negative log marginal likelihood (ML), or negative log restricted likelihood (REML).

If rank deficiency of the model was detected then the apparent rank is reported, along with the length of the cofficient vector (rank in absense of rank deficiency). Rank deficiency occurs when not all coefficients are identifiable given the data. Although the fitting routines (except gamm) deal gracefully with rank deficiency, interpretation of rank deficient models may be difficult.

## Author(s)

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

## References

Wood, S.N. (2006) Generalized Additive Models: An Introduction with R. CRC/ Chapmand and Hall, Boca Raton, Florida.

```
http://www.maths.bris.ac.uk/~sw15190/
```

# See Also

```
gam, summary.gam
```

qq.gam 161

qq.gam

QQ plots for gam model residuals

# **Description**

Takes a fitted gam object produced by gam() and produces QQ plots of its residuals (conditional on the fitted model coefficients and scale parameter). If the model distributional assumptions are met then usually these plots should be close to a straight line (although discrete data can yield marked random departures from this line).

## Usage

# **Arguments**

object	a fitted gam object as produced by gam() (or a glm object).
rep	How many replicate datasets to generate to simulate quantiles of the residual distribution. 0 results in an efficient simulation free method for direct calculation, if this is possible for the object family.
level	If simulation is used for the quantiles, then reference intervals can be provided for the QQ-plot, this specifies the level. 0 or less for no intervals, 1 or more to simply plot the QQ plot for each replicate generated.
s.rep	how many times to randomize uniform quantiles to data under direct computation.
type	what sort of residuals should be plotted? See residuals.gam.
pch	plot character to use. 19 is good.
rl.col	color for the reference line on the plot.
rep.col	color for reference bands or replicate reference plots.
	extra graphics parameters to pass to plotting functions.

## Details

QQ-plots of the the model residuals can be produced in one of two ways. The cheapest method generates reference quantiles by associating a quantile of the uniform distribution with each datum, and feeding these uniform quantiles into the quantile function associated with each datum. The resulting quantiles are then used in place of each datum to generate approximate quantiles of residuals. The residual quantiles are averaged over s.rep randomizations of the uniform quantiles to data.

The second method is to use direct simulatation. For each replicate, data are simulated from the fitted model, and the corresponding residuals computed. This is repeated rep times. Quantiles are readily obtained from the empirical distribution of residuals so obtained. From this method reference bands are also computable.

162 qq.gam

Even if rep is set to zero, the routine will attempt to simulate quantiles if no quantile function is available for the family. If no random deviate generating function family is available (e.g. for the quasi families), then a normal QQ-plot is produced. The routine conditions on the fitted model coefficients and the scale parameter estimate.

The plots are very similar to those proposed in Ben and Yohai (2004), but are substantially cheaper to produce (the interpretation of residuals for binary data in Ben and Yohai is not recommended).

Note that plots for raw residuals from fits to binary data contain almost no useful information about model fit. Whether the residual is negative or positive is decided by whether the response is zero or one. The magnitude of the residual, given its sign, is determined entirely by the fitted values. In consequence only the most gross violations of the model are detectable from QQ-plots of residuals for binary data. To really check distributional assumptions from residuals for binary data you have to be able to group the data somehow. Binomial models other than binary are ok.

#### Author(s)

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

#### References

N.H. Augustin, E-A Sauleaub, S.N. Wood (2012) On quantile quantile plots for generalized linear models Computational Statistics & Data Analysis. 56(8), 2404-2409.

M.G. Ben and V.J. Yohai (2004) JCGS 13(1), 36-47.

```
http://www.maths.bris.ac.uk/~sw15190/
```

#### See Also

```
choose.k, gam
```

```
library(mgcv)
## simulate binomial data...
set.seed(0)
n.samp < -400
dat <- gamSim(1,n=n.samp,dist="binary",scale=.33)</pre>
p <- binomial()$linkinv(dat$f) ## binomial p</pre>
n <- sample(c(1,3),n.samp,replace=TRUE) ## binomial n</pre>
dat$y <- rbinom(n,n,p)</pre>
dat$n <- n
lr.fit <- gam(y/n^s(x0)+s(x1)+s(x2)+s(x3))
              , family=binomial, data=dat, weights=n, method="REML")
par(mfrow=c(2,2))
## normal QQ-plot of deviance residuals
qqnorm(residuals(lr.fit),pch=19,cex=.3)
## Quick QQ-plot of deviance residuals
qq.gam(lr.fit,pch=19,cex=.3)
## Simulation based QQ-plot with reference bands
```

random.effects 163

```
qq.gam(lr.fit,rep=100,level=.9)
## Simulation based QQ-plot, Pearson resids, all
## simulated reference plots shown...
qq.gam(lr.fit,rep=100,level=1,type="pearson",pch=19,cex=.2)
## Now fit the wrong model and check....
pif <- gam(y \sim s(x0) + s(x1) + s(x2) + s(x3)
              ,family=poisson,data=dat,method="REML")
par(mfrow=c(2,2))
qqnorm(residuals(pif),pch=19,cex=.3)
qq.gam(pif,pch=19,cex=.3)
qq.gam(pif,rep=100,level=.9)
qq.gam(pif,rep=100,level=1,type="pearson",pch=19,cex=.2)
## Example of binary data model violation so gross that you see a problem
## on the QQ plot...
y \leftarrow c(rep(1,10), rep(0,20), rep(1,40), rep(0,10), rep(1,40), rep(0,40))
x <- 1:160
b <- glm(y~x,family=binomial)</pre>
par(mfrow=c(2,2))
## Note that the next two are not necessarily similar under gross
## model violation...
qq.gam(b)
qq.gam(b,rep=50,level=1)
## and a much better plot for detecting the problem
plot(x,residuals(b),pch=19,cex=.3)
plot(x,y);lines(x,fitted(b))
## alternative model
b \leftarrow gam(y \sim s(x,k=5),family=binomial,method="ML")
qq.gam(b)
qq.gam(b,rep=50,level=1)
plot(x,residuals(b),pch=19,cex=.3)
plot(b,residuals=TRUE,pch=19,cex=.3)
```

random.effects

Random effects in GAMs

## Description

The smooth components of GAMs can be viewed as random effects for estimation purposes. This means that more conventional random effects terms can be incorporated into GAMs in two ways. The first method converts all the smooths into fixed and random components suitable for estimation by standard mixed modelling software. Once the GAM is in this form then conventional random effects are easily added, and the whole model is estimated as a general mixed model. gamm and gamm4 from the gamm4 package operate in this way.

164 random.effects

The second method represents the conventional random effects in a GAM in the same way that the smooths are represented — as penalized regression terms. This method can be used with gam by making use of  $s(\dots,bs="re")$  terms in a model: see smooth.construct.re.smooth.spec, for full details. The basic idea is that, e.g., s(x,z,g,bs="re") generates an i.i.d. Gaussian random effect with model matrix given by  $model.matrix(\sim x:z:g-1)$  — in principle such terms can take any number of arguments. This simple approach is sufficient for implementing a wide range of commonly used random effect structures. For example if g is a factor then s(g,bs="re") produces a random coefficient for each level of g, with the radndom coefficients all modelled as i.i.d. normal. If g is a factor and x is numeric, then s(x,g,bs="re") produces an i.i.d. normal random slope relating the response to x for each level of g. If h is another factor then s(h,g,bs="re") produces the usual i.i.d. normal g - h interaction. Note that a rather useful approximate test for zero random effect is also implemented for tsuch terms based on Wood (2013). If the precision matrix is known to within a multiplicative constant, then this can be supplied via the xt argument of s. See smooth.construct.re.smooth.spec for details and example.

Alternatively, but less straightforwardly, the paraPen argument to gam can be used: see gam.models. If smoothing parameter estimation is by ML or REML (e.g. gam(...,method="REML")) then this approach is a completely conventional likelihood based treatment of random effects.

gam can be slow for fitting models with large numbers of random effects, because it does not exploit the sparcity that is often a feature of parametric random effects. It can not be used for models with more coefficients than data. However gam is often faster and more relaiable than gamm or gamm4, when the number of random effects is modest.

To facilitate the use of random effects with gam, gam.vcomp is a utility routine for converting smoothing parameters to variance components. It also provides confidence intervals, if smoothness estimation is by ML or REML.

Note that treating random effects as smooths does not remove the usual problems associated with testing variance components for equality to zero: see summary.gam and anova.gam.

## Author(s)

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

#### References

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

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

## See Also

gam.vcomp, gam.models, smooth.terms, smooth.construct.re.smooth.spec, gamm

residuals.gam 165

## **Examples**

```
## see also examples for gam.models, gam.vcomp, gamm and smooth.construct.re.smooth.spec
## simple comparison of lme and gam
require(mgcv)
require(nlme)
b0 <- lme(travel~1,data=Rail,~1|Rail,method="REML")

b <- gam(travel~s(Rail,bs="re"),data=Rail,method="REML")
intervals(b0)
gam.vcomp(b)
anova(b)</pre>
```

residuals.gam

Generalized Additive Model residuals

## **Description**

Returns residuals for a fitted gam model object. Pearson, deviance, working and response residuals are available.

#### Usage

```
## S3 method for class 'gam'
residuals(object, type = "deviance",...)
```

## **Arguments**

object a gam fitted model object.

type the type of residuals wanted. Usually one of "deviance", "pearson", "scaled.pearson",

"working", or "response".

... other arguments.

#### **Details**

Response residuals are the raw residuals (data minus fitted values). Scaled Pearson residuals are raw residuals divided by the standard deviation of the data according to the model mean variance relationship and estimated scale parameter. Pearson residuals are the same, but multiplied by the square root of the scale parameter (so they are independent of the scale parameter):  $((y-\mu)/\sqrt{V(\mu)})$ , where y is data  $\mu$  is model fitted value and V is model mean-variance relationship.). Both are provided since not all texts agree on the definition of Pearson residuals. Deviance residuals simply return the deviance residuals defined by the model family. Working residuals are the residuals returned from model fitting at convergence.

Families can supply their own residual function, which is used in place of the standard function if present, (e.g. cox.ph).

166 rig

# Value

A vector of residuals.

## Author(s)

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

## See Also

gam

rig

Generate inverse Gaussian random deviates

# **Description**

Generates inverse Gaussian random deviates.

## Usage

```
rig(n,mean,scale)
```

# **Arguments**

n the number of deviates required. If this has length > 1 then the length is taken

as the number of deviates required.

mean vector of mean values.

scale vector of scale parameter values (lambda, see below)

#### **Details**

If x if the returned vector, then E(x) = mean while var(x) = scale\*mean\*3. For density and distribution functions see the statmod package. The algorithm used is Algorithm 5.7 of Gentle (2003), based on Michael et al. (1976). Note that scale here is the scale parameter in the GLM sense, which is the reciprocal of the usual 'lambda' parameter.

# Value

A vector of inverse Gaussian random deviates.

# Author(s)

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

rmvn 167

## References

Gentle, J.E. (2003) Random Number Generation and Monte Carlo Methods (2nd ed.) Springer.

Michael, J.R., W.R. Schucany & R.W. Hass (1976) Generating random variates using transformations with multiple roots. The American Statistician 30, 88-90.

```
http://www.maths.bris.ac.uk/~sw15190/
```

## **Examples**

rmvn

Generate multivariate normal deviates

## **Description**

Generates multivariate normal random deviates.

## Usage

```
rmvn(n,mu,V)
```

# Arguments

n number of simulated vectors required.

mu the mean of the vectors: either a single vector of length p=ncol(V) or an n by p

matrix.

V A positive semi definite covariance matrix.

## **Details**

Uses a 'square root' of V to transform stadard normal deviates to multivariate normal with the correct covariance matrix.

168 Rrank

## Value

An n row matrix, with each row being a draw from a multivariate normal density with covariance matrix V and mean vector mu. Alternatively each row may have a different mean vector if mu is a vector.

#### Author(s)

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

#### See Also

```
ldTweedie, Tweedie
```

# **Examples**

```
library(mgcv) 
V <- matrix(c(2,1,1,2),2,2) 
mu <- c(1,3) 
n <- 1000 
z <- rmvn(n,mu,V) 
crossprod(sweep(z,2,colMeans(z)))/n ## observed covariance matrix 
colMeans(z) ## observed mu
```

Rrank

Find rank of upper triangular matrix

## **Description**

Finds rank of upper triangular matrix R, by estimating condition number of upper rank by rank block, and reducing rank until this is acceptably low. Assumes R has been computed by a method that uses pivoting, usually pivoted QR or Choleski.

# Usage

```
Rrank(R, tol=.Machine$double.eps^.9)
```

# **Arguments**

R An upper triangular matrix, obtained by pivoted QR or pivoted Choleski.

tol the tolerance to use for judging rank.

# **Details**

The method is based on Cline et al. (1979) as described in Golub and van Loan (1996).

#### Author(s)

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

rTweedie 169

## References

Cline, A.K., C.B. Moler, G.W. Stewart and J.H. Wilkinson (1979) An estimate for the condition number of a matrix. SIAM J. Num. Anal. 16, 368-375

Golub, G.H, and C.F. van Loan (1996) Matrix Computations 3rd ed. Johns Hopkins University Press, Baltimore.

# **Examples**

```
set.seed(0)
n <- 10;p <- 5
X <- matrix(runif(n*(p-1)),n,p)
qrx <- qr(X,LAPACK=TRUE)
Rrank(qr.R(qrx))</pre>
```

rTweedie

Generate Tweedie random deviates

# Description

Generates Tweedie random deviates, for powers between 1 and 2.

## Usage

```
rTweedie(mu,p=1.5,phi=1)
```

# Arguments

mu	vector of expected values for the deviates to be generated. One deviate generated for each element of mu.
p	the variance of a deviate is proportional to its mean, mu to the power p. p must be between 1 and 2. 1 is Poisson like (exactly Poisson if phi=1), 2 is gamma.
phi	The scale parameter. Variance of the deviates is given by is phi*mu^p.

## **Details**

A Tweedie random variable with  $1 is a sum of N gamma random variables where N has a Poisson distribution, with mean <math>mu^(2-p)/((2-p)*phi)$ . The Gamma random variables that are summed have shape parameter (2-p)/(p-1) and scale parameter  $phi*(p-1)*mu^(p-1)$  (note that this scale parameter is different from the scale parameter for a GLM with Gamma errors).

This is a restricted, but faster, version of rtweedie from the tweedie package.

#### Value

A vector of random deviates from a Tweedie distribution, expected value vector mu, variance vector phi\*mu^p.

170 s

#### Author(s)

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

#### References

Peter K Dunn (2009). tweedie: Tweedie exponential family models. R package version 2.0.2. http://CRAN.R-project.org/package=tweedie

## See Also

ldTweedie, Tweedie

# **Examples**

Defining smooths in GAM formulae

# Description

Function used in definition of smooth terms within gam model formulae. The function does not evaluate a (spline) smooth - it exists purely to help set up a model using spline based smooths.

#### Usage

s

```
s(..., k=-1,fx=FALSE,bs="tp",m=NA,by=NA,xt=NULL,id=NULL,sp=NULL)
```

# Arguments

k

fx

... a list of variables that are the covariates that this smooth is a function of.

the dimension of the basis used to represent the smooth term. The default depends on the number of variables that the smooth is a function of. k should not be less than the dimension of the null space of the penalty for the term (see null.space.dimension), but will be reset if it is. See choose.k for further information.

indicates whether the term is a fixed d.f. regression spline (TRUE) or a penalized regression spline (FALSE).

s 171

bs a two letter character string indicating the (penalized) smoothing basis to use. (eg "tp" for thin plate regression spline, "cr" for cubic regression spline). see smooth.terms for an over view of what is available.

The order of the penalty for this term (e.g. 2 for normal cubic spline penalty with 2nd derivatives when using default t.p.r.s basis). NA signals autoinitialization. Only some smooth classes use this. The "ps" class can use a 2 item array giving the basis and penalty order separately.

a numeric or factor variable of the same dimension as each covariate. In the numeric vector case the elements multiply the smooth, evaluated at the corresponding covariate values (a 'varying coefficient model' results). For the numeric by variable case the resulting smooth is not usually subject to a centering constraint (so the by variable should not be added as an additional main effect). In the factor by variable case a replicate of the smooth is produced for each factor level (these smooths will be centered, so the factor usually needs to be added as a main effect as well). See gam.models for further details. A by variable may also be a matrix if covariates are matrices: in this case implements linear functional of a smooth (see gam.models and linear.functional.terms for details).

Any extra information required to set up a particular basis. Used e.g. to set large data set handling behaviour for "tp" basis.

A label or integer identifying this term in order to link its smoothing parameters to others of the same type. If two or more terms have the same id then they will have the same smoothing parameters, and, by default, the same bases (first occurance defines basis type, but data from all terms used in basis construction). An id with a factor by variable causes the smooths at each factor level to have the same smoothing parameter.

any supplied smoothing parameters for this term. Must be an array of the same length as the number of penalties for this smooth. Positive or zero elements are taken as fixed smoothing parameters. Negative elements signal autoinitialization. Over-rides values supplied in sp argument to gam. Ignored by gamm.

## **Details**

m

by

хt

id

sp

The function does not evaluate the variable arguments. To use this function to specify use of your own smooths, note the relationships between the inputs and the output object and see the example in smooth.construct.

#### Value

term

A class xx.smooth.spec object, where xx is a basis identifying code given by the bs argument of s. These smooth.spec objects define smooths and are turned into bases and penalties by smooth.construct method functions.

The returned object contains the following items:

An array of text strings giving the names of the covariates that the term is a function of.

172 s

bs.dim	The dimension of the basis used to represent the smooth.
fixed	TRUE if the term is to be treated as a pure regression spline (with fixed degrees of freedom); FALSE if it is to be treated as a penalized regression spline
dim	The dimension of the smoother - i.e. the number of covariates that it is a function of.
p.order	The order of the t.p.r.s. penalty, or 0 for auto-selection of the penalty order.
by	is the name of any by variable as text ("NA" for none).
label	A suitable text label for this smooth term.
xt	The object passed in as argument xt.
id	An identifying label or number for the smooth, linking it to other smooths. Defaults to NULL for no linkage.
sp	array of smoothing parameters for the term (negative for auto-estimation). Defaults to $\ensuremath{NULL}$ .

## Author(s)

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

## References

Wood, S.N. (2003) Thin plate regression splines. J.R.Statist.Soc.B 65(1):95-114

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

```
http://www.maths.bris.ac.uk/~sw15190/
```

# See Also

```
te, gam, gamm
```

```
# example utilising `by' variables
library(mgcv)
set.seed(0)
n<-200;sig2<-4
x1 \leftarrow runif(n, 0, 1); x2 \leftarrow runif(n, 0, 1); x3 \leftarrow runif(n, 0, 1)
fac<-c(rep(1,n/2),rep(2,n/2)) # create factor
fac.1<-rep(0,n)+(fac==1);fac.2<-1-fac.1 # and dummy variables
fac<-as.factor(fac)</pre>
f1 \leftarrow exp(2 * x1) - 3.75887
f2 \leftarrow 0.2 \times x1^{11} \times (10 \times (1 - x1))^6 + 10 \times (10 \times x1)^3 \times (1 - x1)^{10}
f<-f1*fac.1+f2*fac.2+x2
e <- rnorm(n, 0, sqrt(abs(sig2)))
y \leftarrow f + e
# NOTE: smooths will be centered, so need to include fac in model....
b < -gam(y - fac + s(x1, by = fac) + x2)
plot(b,pages=1)
```

scat 173

scat

GAM scaled t family for heavy tailed data

## **Description**

Family for use with gam, implementing regression for the heavy tailed response variables, y, using a scaled t model. The idea is that  $(y-\mu)/\sigma \sim t_{\nu}$  where mu is determined by a linear predictor, while  $\sigma$  and  $\nu$  are parameters to be estimated alongside the smoothing parameters.

## Usage

```
scat(theta = NULL, link = "identity")
```

## **Arguments**

theta the parameters to be estimated  $\nu = 2 + \exp(\theta_1)$  and  $\sigma = \exp(\theta_2)$ . If supplied

and positive, then taken to be fixed values of  $\nu$  and  $\sigma$ . If any negative, then

absolute values taken as starting values.

link The link function: one of "identity", "log" or "inverse".

## **Details**

Useful in place of Gaussian, when data are heavy tailed.

#### Value

An object of class extended.family.

## Author(s)

Natalya Pya (nyp20@bath.ac.uk)

## References

Wood, S.N., N. Pya and B. Saefken (2015), Smoothing parameter and model selection for general smooth models. http://arxiv.org/abs/1511.03864

```
library(mgcv)
## Simulate some t data...
set.seed(3);n<-400
dat <- gamSim(1,n=n)
dat$y <- dat$f + rt(n,df=3)*2

b <- gam(y~s(x0)+s(x1)+s(x2)+s(x3),family=scat(link="identity"),data=dat)
b</pre>
```

174 sdiag

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

sdiag

Extract of modify diagonals of a matrix

# Description

Extracts of modifies sub- or super- diagonals of a matrix.

# Usage

```
sdiag(A,k=0)
sdiag(A,k=0) <- value</pre>
```

# Arguments

A a matrix

k sub- (negative) or super- (positive) diagonal of a matrix. 0 is the leading diago-

nal.

value single value, or vector of the same length as the diagonal.

## Value

A vector containing the requested diagonal, or a matris with the requested diagonal replaced by value.

## Author(s)

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

```
require(mgcv)
A <- matrix(1:35,7,5)
A
sdiag(A,1) ## first super diagonal
sdiag(A,-1) ## first sub diagonal
sdiag(A) <- 1 ## leading diagonal set to 1
sdiag(A,3) <- c(-1,-2) ## set 3rd super diagonal</pre>
```

single.index 175

single.index

Single index models with mgcv

## **Description**

Single index models contain smooth terms with arguments that are linear combinations of other covariates. e.g.  $s(X\alpha)$  where  $\alpha$  has to be estimated. For identifiability, assume  $\|\alpha\|=1$  with positive first element. One simple way to fit such models is to use gam to profile out the smooth model coefficients and smoothing parameters, leaving only the  $\alpha$  to be estimated by a general purpose optimizer.

Example code is provided below, which can be easily adapted to include multiple single index terms, parametric terms and further smooths. Note the initialization strategy. First estimate  $\alpha$  without penalization to get starting values and then do the full fit. Otherwise it is easy to get trapped in a local optimum in which the smooth is linear. An alternative is to initialize using fixed penalization (via the sp argument to gam).

## Author(s)

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

```
require(mgcv)
si <- function(theta, y, x, z, opt=TRUE, k=10, fx=FALSE) {
## Fit single index model using gam call, given theta (defines alpha).
## Return ML is opt==TRUE and fitted gam with theta added otherwise.
## Suitable for calling from 'optim' to find optimal theta/alpha.
 alpha <- c(1,theta) ## constrained alpha defined using free theta
 kk <- sqrt(sum(alpha^2))
 alpha <- alpha/kk ## so now ||alpha||=1
                      ## argument of smooth
 a <- x%*%alpha
 b \leftarrow gam(y \sim s(a, fx = fx, k = k) + s(z), family = poisson, method = "ML") ## fit model
 if (opt) return(b$gcv.ubre) else {
    b$alpha <- alpha ## add alpha
    J <- outer(alpha,-theta/kk^2) ## compute Jacobian</pre>
    for (j in 1:length(theta)) J[j+1,j] \leftarrow J[j+1,j] + 1/kk
    b$J <- J ## dalpha_i/dtheta_j
    return(b)
 }
} ## si
## simulate some data from a single index model...
set.seed(1)
f2 \leftarrow function(x) 0.2 * x^11 * (10 * (1 - x))^6 + 10 *
            (10 * x)^3 * (1 - x)^10
n <- 200;m <- 3
x \leftarrow matrix(runif(n*m),n,m) ## the covariates for the single index part
```

176 slanczos

```
z <- runif(n) ## another covariate</pre>
alpha \leftarrow c(1,-1,.5); alpha \leftarrow alpha/sqrt(sum(alpha^2))
eta <- as.numeric(f2((x%*%alpha+.41)/1.4)+1+z^2*2)/4
mu <- exp(eta)</pre>
y <- rpois(n,mu) ## Poi response
## now fit to the simulated data...
th0 <- c(-.8,.4) ## close to truth for speed
## get initial theta, using no penalization...
f0 \leftarrow nlm(si, th0, y=y, x=x, z=z, fx=TRUE, k=5)
## now get theta/alpha with smoothing parameter selection...
f1 <- nlm(si,f0$estimate,y=y,x=x,z=z,hessian=TRUE,k=10)</pre>
theta.est <-f1$estimate</pre>
## Alternative using 'optim' ('Not run' simply to keep
## CRAN check time down)...
## Not run:
th0 <- rep(0, m-1)
## get initial theta, using no penalization...
f0 <- optim(th0,si,y=y,x=x,z=z,fx=TRUE,k=5)</pre>
## now get theta/alpha with smoothing parameter selection...
f1 <- optim(f0$par,si,y=y,x=x,z=z,hessian=TRUE,k=10)</pre>
theta.est <-f1$par
## End(Not run)
## extract and examine fitted model...
b \leftarrow si(theta.est,y,x,z,opt=FALSE) ## extract best fit model
plot(b,pages=1)
b
b$alpha
## get sd for alpha...
Vt <- b$J%*%solve(f1$hessian,t(b$J))</pre>
diag(Vt)^.5
```

slanczos

Compute truncated eigen decomposition of a symmetric matrix

# **Description**

Uses Lanczos iteration to find the truncated eigen-decomposition of a symmetric matrix.

# Usage

```
slanczos(A,k=10,kl=-1,tol=.Machine$double.eps^.5,nt=1)
```

slanczos 177

## Arguments

Α	A symmetric matrix.
k	Must be non-negative. If k1 is negative, then the k largest magnitude eigenvalues are found, together with the corresponding eigenvectors. If k1 is non-negative then the k highest eigenvalues are found together with their eigenvectors and the k1 lowest eigenvalues with eigenvectors are also returned.
k1	If kl is non-negative then the kl lowest eigenvalues are returned together with their corresponding eigenvectors (in addition to the k highest eignevalues + vectors). negative kl signals that the k largest magnitude eigenvalues should be returned, with eigenvectors.
tol	tolerance to use for convergence testing of eigenvalues. Error in eigenvalues will be less than the magnitude of the dominant eigenvalue multiplied by tol (or the machine precision!).
nt	number of threads to use for leading order iterative multiplication of A by vector. May show no speed improvement on two processor machine.

## **Details**

If k1 is non-negative, returns the highest k and lowest k1 eigenvalues, with their corresponding eigenvectors. If k1 is negative, returns the largest magnitude k eigenvalues, with corresponding eigenvectors.

The routine implements Lanczos iteration with full re-orthogonalization as described in Demmel (1997). Lanczos iteraction iteratively constructs a tridiagonal matrix, the eigenvalues of which converge to the eigenvalues of A, as the iteration proceeds (most extreme first). Eigenvectors can also be computed. For small k and k1 the approach is faster than computing the full symmetric eigendecomposition. The tridiagonal eigenproblems are handled using LAPACK.

The implementation is not optimal: in particular the inner triadiagonal problems could be handled more efficiently, and there would be some savings to be made by not always returning eigenvectors.

#### Value

A list with elements values (array of eigenvalues); vectors (matrix with eigenvectors in its columns); iter (number of iterations required).

#### Author(s)

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

#### References

Demmel, J. (1997) Applied Numerical Linear Algebra. SIAM

## See Also

cyclic.p.spline

178 smooth.construct

## **Examples**

```
require(mgcv)
## create some x's and knots...
set.seed(1);
n <- 700;A <- matrix(runif(n*n),n,n);A <- A+t(A)

## compare timings of slanczos and eigen
system.time(er <- slanczos(A,10))
system.time(um <- eigen(A,symmetric=TRUE))

## confirm values are the same...
ind <- c(1:6,(n-3):n)
range(er$values-um$values[ind]);range(abs(er$vectors)-abs(um$vectors[,ind]))</pre>
```

smooth.construct

Constructor functions for smooth terms in a GAM

# Description

Smooth terms in a GAM formula are turned into smooth specification objects of class xx.smooth.spec during processing of the formula. Each of these objects is converted to a smooth object using an appropriate smooth.construct function. New smooth classes can be added by writing a new smooth.construct method function and a corresponding Predict.matrix method function (see example code below).

In practice, smooth.construct is usually called via smooth.construct2 and the wrapper function smoothCon, in order to handle by variables and centering constraints (see the smoothCon documentation if you need to handle these things directly, for a user defined smooth class).

## Usage

```
smooth.construct(object,data,knots)
smooth.construct2(object,data,knots)
```

## **Arguments**

object

is a smooth specification object, generated by an s or te term in a GAM formula. Objects generated by s terms have class xx.smooth.spec where xx is given by the bs argument of s (this convention allows the user to add their own smoothers). If object is not class tensor.smooth.spec it will have the following elements:

**term** The names of the covariates for this smooth, in an array.

**bs.dim** Argument k of the s term generating the object. This is the dimension of the basis used to represent the term (or, arguably, 1 greater than the basis dimension for cc terms). bs.dim<0 indicates that the constructor should set this to the default value.

**fixed** TRUE if the term is to be unpenalized, otherwise FALSE.

smooth.construct 179

dim the number covariates of which this smooth is a function.

**p.order** the order of the smoothness penalty or NA for autoselection of this. This is argument m of the s term that generated object.

**by** the name of any by variable to multiply this term as supplied as an argument to s. "NA" if there is no such term.

**label** A suitable label for use with this term.

- **xt** An object containing information that may be needed for basis setup (used, e.g. by "tp" smooths to pass optional information on big dataset handling).
- id Any identity associated with this term used for linking bases and smoothing parameters. NULL by default, indicating no linkage.
- **sp** Smoothing parameters for the term. Any negative are estimated, otherwise they are fixed at the supplied value. Unless NULL (default), over-rides sp argument to gam.

If object is of class tensor.smooth.spec then it was generated by a te term in the GAM formula, and specifies a smooth of several variables with a basis generated as a tensor product of lower dimensional bases. In this case the object will be different and will have the following elements:

**margin** is a list of smooth specification objects of the type listed above, defining the bases which have their tensor product formed in order to construct this term.

term is the array of names of the covariates that are arguments of the smooth.

by is the name of any by variable, or "NA".

**fx** is an array, the elements of which indicate whether (TRUE) any of the margins in the tensor product should be unpenalized.

label A suitable label for use with this term.

dim is the number of covariates of which this smooth is a function.

**mp** TRUE if multiple penalties are to be used.

- np TRUE if 1-D marginal smooths are to be re-parameterized in terms of function values.
- id Any identity associated with this term used for linking bases and smoothing parameters. NULL by default, indicating no linkage.
- **sp** Smoothing parameters for the term. Any negative are estimated, otherwise they are fixed at the supplied value. Unless NULL (default), over-rides sp argument to gam.

For smooth.construct a data frame or list containing the evaluation of the elements of object\$term, with names given by object\$term. The last entry will be the by variable, if object\$by is not "NA". For smooth.construct2 data need only be an object within which object\$term can be evaluated, the variables can be in any order, and there can be irrelevant variables present as well.

an optional data frame or list containing the knots relating to object\$term. If it is NULL then the knot locations are generated automatically. The structure of knots should be as for data, depending on whether smooth.construct or smooth.construct2 is used.

data

knots

180 smooth.construct

#### **Details**

There are built in methods for objects with the following classes: tp.smooth.spec (thin plate regression splines: see tprs); ts.smooth.spec (thin plate regression splines with shrinkage-to-zero); cr.smooth.spec (cubic regression splines: see cubic.regression.spline; cs.smooth.spec (cubic regression splines with shrinkage-to-zero); cc.smooth.spec (cyclic cubic regression splines); ps.smooth.spec (Eilers and Marx (1986) style P-splines: see p.spline); cp.smooth.spec (cyclic P-splines); ad.smooth.spec (adaptive smooths of 1 or 2 variables: see adaptive.smooth); re.smooth.spec (simple random effect terms); mrf.smooth.spec (Markov random field smoothers for smoothing over discrete districts); tensor.smooth.spec (tensor product smooths).

There is an implicit assumption that the basis only depends on the knots and/or the set of unique covariate combinations; i.e. that the basis is the same whether generated from the full set of covariates, or just the unique combinations of covariates.

Plotting of smooths is handled by plot methods for smooth objects. A default mgcv.smooth method is used if there is no more specific method available. Plot methods can be added for specific smooth classes, see source code for mgcv:::plot.sos.smooth, mgcv:::plot.random.effect, mgcv:::plot.mgcv.smooth for example code.

#### Value

Ср

The input argument object, assigned a new class to indicate what type of smooth it is and with at least the following items added:

X The model matrix from this term. This may have an "offset" attribute: a vector of length nrow(X) containing any contribution of the smooth to the model offset term. by variables do not need to be dealt with here, but if they are then an item

by . done must be added to the object.

S A list of positive semi-definite penalty matrices that apply to this term. The list

will be empty if the term is to be left un-penalized.

rank An array giving the ranks of the penalties.

null.space.dim The dimension of the penalty null space (before centering).

The following items may be added:

C The matrix defining any identifiability constraints on the term, for use when fitting. If this is NULL then smoothCon will add an identifiability constraint that each term should sum to zero over the covariate values. Set to a zero row matrix

if no constraints are required. If a supplied C has an attribute "always.apply" then it is never ignored, even if any by variables of a smooth imply that no constraint is actually needed. Code for creating C should check whether the specification object already contains a zero row matrix, and leave this unchanged

if it is (since this signifies no constraint should be produced).

An optional matrix supplying alternative identifiability constraints for use when predicting. By default the fitting constraints are used. This option is useful when some sort of simple sparse constraint is required for fitting, but the usual sumto-zero constraint is required for prediction so that, e.g. the CIs for model com-

ponents are as narrow as possible.

smooth.construct 181

with this term (when unpenalized and un- othCon will set it to the basis dimension.
mber of constraints.
tensor product marginal, 1 if it can be used at not plotted. Set to 1 if NULL.
ot be plotted by plot.gam. Set to TRUE if
th is never subject to side constraints as a
lying' smoothing parameters than there are atrix mapping the vector of underlying log clogs of the smoothing parameters actually ifies that there is one smoothing parameter
t u o f

Usually the returned object will also include extra information required to define the basis, and used by Predict.matrix methods to make predictions using the basis. See the Details section for links to the information included for the built in smooth classes.

tensor.smooth returned objects will additionally have each element of the margin list updated in the same way. tensor.smooths also have a list, XP, containing re-parameterization matrices for any 1-D marginal terms re-parameterized in terms of function values. This list will have NULL entries for marginal smooths that are not re-parameterized, and is only long enough to reach the last re-parameterized marginal in the list.

#### WARNING

User defined smooth objects should avoid having attributes names "qrc" or "nCons" as these are used internally to provide constraint free parameterizations.

## Author(s)

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

#### References

Wood, S.N. (2003) Thin plate regression splines. J.R.Statist.Soc.B 65(1):95-114

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

The code given in the example is based on the smooths advocated in:

Ruppert, D., M.P. Wand and R.J. Carroll (2003) Semiparametric Regression. Cambridge University Press.

182 smooth.construct

However if you want p-splines, rather than splines with derivative based penalties, then the built in "ps" class is probably a marginally better bet. It's based on

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

```
http://www.maths.bris.ac.uk/~sw15190/
```

#### See Also

```
s,get.var, gamm, gam, Predict.matrix, smoothCon, PredictMat
```

```
## Adding a penalized truncated power basis class and methods
## as favoured by Ruppert, Wand and Carroll (2003)
## Semiparametric regression CUP. (No advantage to actually
## using this, since mgcv can happily handle non-identity
## penalties.)
smooth.construct.tr.smooth.spec<-function(object,data,knots)</pre>
## a truncated power spline constructor method function
## object$p.order = null space dimension
{ m <- object$p.order[1]
 if (is.na(m)) m <- 2 ## default
 if (m<1) stop("silly m supplied")</pre>
 if (object$bs.dim<0) object$bs.dim <- 10 ## default
 nk<-object$bs.dim-m-1 ## number of knots
 if (nk<=0) stop("k too small for m")
 x <- data[[object$term]] ## the data</pre>
 x.shift <- mean(x) # shift used to enhance stability
 k <- knots[[object$term]] ## will be NULL if none supplied</pre>
 if (is.null(k)) \# space knots through data
 { n<-length(x)
    k < -quantile(x[2:(n-1)], seq(0,1,length=nk+2))[2:(nk+1)]
 if (length(k)!=nk) # right number of knots?
 stop(paste("there should be ",nk," supplied knots"))
 x \leftarrow x - x.shift # basis stabilizing shift
 k \leftarrow k - x.shift # knots treated the same!
 X<-matrix(0,length(x),object$bs.dim)</pre>
 for (i in 1:(m+1)) X[,i] <- x^{(i-1)}
 for (i in 1:nk) X[,i+m+1]<-(x-k[i])^m*as.numeric(x>k[i])
 object$X<-X # the finished model matrix
 if (!object$fixed) # create the penalty matrix
 { object$S[[1]]<-diag(c(rep(0,m+1),rep(1,nk)))
 object$rank<-nk # penalty rank
 object$null.space.dim <- m+1 # dim. of unpenalized space
 ## store "tr" specific stuff ...
 object$knots<-k;object$m<-m;object$x.shift <- x.shift
 object$df<-ncol(object$X) # maximum DoF (if unconstrained)</pre>
```

```
class(object)<-"tr.smooth" # Give object a class</pre>
  object
}
Predict.matrix.tr.smooth<-function(object,data)</pre>
## prediction method function for the `tr' smooth class
{ x <- data[[object$term]]
  x <- x - object$x.shift # stabilizing shift
  m <- object$m;</pre>
                      # spline order (3=cubic)
                    # knot locations
  k<-object$knots
  nk<-length(k)
                      # number of knots
  X<-matrix(0,length(x),object$bs.dim)</pre>
  for (i in 1:(m+1)) X[,i] <- x^{(i-1)}
  for (i in 1:nk) X[,i+m+1] \leftarrow (x-k[i])^m*as.numeric(x>k[i])
  X # return the prediction matrix
}
# an example, using the new class....
require(mgcv)
set.seed(100)
dat <- gamSim(1,n=400,scale=2)</pre>
b < -gam(y < x0, bs = "tr", m = 2) + s(x1, bs = "ps", m = c(1, 3)) +
         s(x2,bs="tr",m=3)+s(x3,bs="tr",m=2),data=dat)
plot(b,pages=1)
b < -gamm(y < x(0, bs = "tr", m = 2) + s(x1, bs = "ps", m = c(1, 3)) +
         s(x2,bs="tr",m=3)+s(x3,bs="tr",m=2),data=dat)
plot(b$gam,pages=1)
# another example using tensor products of the new class
dat <- gamSim(2,n=400,scale=.1)$data</pre>
b \leftarrow gam(y^te(x,z,bs=c("tr","tr"),m=c(2,2)),data=dat)
vis.gam(b)
```

smooth.construct.ad.smooth.spec

Adaptive smooths in GAMs

## **Description**

gam can use adaptive smooths of one or two variables, specified via terms like s(...,bs="ad",...). (gamm can not use such terms — check out package AdaptFit if this is a problem.) The basis for such a term is a (tensor product of) p-spline(s) or cubic regression spline(s). Discrete P-spline type penalties are applied directly to the coefficients of the basis, but the penalties themselves have a basis representation, allowing the strength of the penalty to vary with the covariates. The coefficients of the penalty basis are the smoothing parameters.

When invoking an adaptive smoother the k argument specifies the dimension of the smoothing basis (default 40 in 1D, 15 in 2D), while the m argument specifies the dimension of the penalty basis (default 5 in 1D, 3 in 2D). For an adaptive smooth of two variables k is taken as the dimension of both marginal bases: different marginal basis dimensions can be specified by making k a two element vector. Similarly, in the two dimensional case m is the dimension of both marginal bases for

the penalties, unless it is a two element vector, which specifies different basis dimensions for each marginal (If the penalty basis is based on a thin plate spline then m specifies its dimension directly).

By default, P-splines are used for the smoothing and penalty bases, but this can be modified by supplying a list as argument xt with a character vector xt\$bs specifying the smoothing basis type. Only "ps", "cp", "cc" and "cr" may be used for the smoothing basis. The penalty basis is always a B-spline, or a cyclic B-spline for cyclic bases.

The total number of smoothing parameters to be estimated for the term will be the dimension of the penalty basis. Bear in mind that adaptive smoothing places quite severe demands on the data. For example, setting m=10 for a univariate smooth of 200 data is rather like estimating 10 smoothing parameters, each from a data series of length 20. The problem is particularly serious for smooths of 2 variables, where the number of smoothing parameters required to get reasonable flexibility in the penalty can grow rather fast, but it often requires a very large smoothing basis dimension to make good use of this flexibility. In short, adaptive smooths should be used sparingly and with care.

In practice it is often as effective to simply transform the smoothing covariate as it is to use an adaptive smooth.

## Usage

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

## **Arguments**

object	a smooth specification object, usually generated by a term $s(\dots,bs="ad",\dots)$
data	a list containing just the data (including any by variable) required by this term, with names corresponding to object\$term (and object\$by). The by variable is the last element.
knots	a list containing any knots supplied for basis setup — in same order and with same names as data. Can be NULL

## **Details**

The constructor is not normally called directly, but is rather used internally by gam. To use for basis setup it is recommended to use smooth.construct2.

This class can not be used as a marginal basis in a tensor product smooth, nor by gamm.

## Value

An object of class "pspline.smooth" in the 1D case or "tensor.smooth" in the 2D case.

## Author(s)

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

```
## Comparison using an example taken from AdaptFit
## library(AdaptFit)
require(mgcv)
set.seed(0)
x <- 1:1000/1000
mu <- exp(-400*(x-.6)^2)+5*exp(-500*(x-.75)^2)/3+2*exp(-500*(x-.9)^2)
y < - mu + 0.5 * rnorm(1000)
##fit with default knots
## y.fit <- asp(y^f(x))
par(mfrow=c(2,2))
## plot(y.fit,main=round(cor(fitted(y.fit),mu),digits=4))
## lines(x,mu,col=2)
b <- gam(y^s(x,bs="ad",k=40,m=5)) ## adaptive
plot(b, shade=TRUE, main=round(cor(fitted(b), mu), digits=4))
lines(x,mu-mean(mu),col=2)
b < -gam(y^s(x,k=40))
                                    ## non-adaptive
plot(b, shade=TRUE, main=round(cor(fitted(b), mu), digits=4))
lines(x,mu-mean(mu),col=2)
b <- gam(y < s(x,bs = "ad",k = 40,m = 5,xt = list(bs = "cr")))
plot(b, shade=TRUE, main=round(cor(fitted(b), mu), digits=4))
lines(x,mu-mean(mu),col=2)
## A 2D example (marked, 'Not run' purely to reduce
## checking load on CRAN).
## Not run:
par(mfrow=c(2,2), mar=c(1,1,1,1))
x < - seq(-.5, 1.5, length = 60)
f3 <- function(x,z,k=15) { r<-sqrt(x^2+z^2);f<-exp(-r^2*k);f}
f \leftarrow outer(x, z, f3)
op <- par(bg = "white")
## Plot truth....
persp(x,z,f,theta=30,phi=30,col="lightblue",ticktype="detailed")
n <- 2000
x \leftarrow runif(n)*2-.5
z \leftarrow runif(n)*2-.5
f \leftarrow f3(x,z)
y \leftarrow f + rnorm(n)*.1
## Try tprs for comparison...
b0 < -gam(y^s(x,z,k=150))
vis.gam(b0,theta=30,phi=30,ticktype="detailed")
## Tensor product with non-adaptive version of adaptive penalty
```

```
b1 <- gam(y~s(x,z,bs="ad",k=15,m=1),gamma=1.4)
vis.gam(b1,theta=30,phi=30,ticktype="detailed")
## Now adaptive...
b <- gam(y~s(x,z,bs="ad",k=15,m=3),gamma=1.4)
vis.gam(b,theta=30,phi=30,ticktype="detailed")
cor(fitted(b0),f);cor(fitted(b),f)
## End(Not run)</pre>
```

```
smooth.construct.bs.smooth.spec

Penalized B-splines in GAMs
```

# Description

gam can use smoothing splines based on univariate B-spline bases with derivative based penalties, specified via terms like s(x,bs="bs",m=c(3,2)). m[1] controls the spline order, with m[1]=3 being a cubic spline, m[1]=2 being quadratic, and so on. The integrated square of the m[2]th derivative is used as the penalty. So m=c(3,2) is a convetional cubic spline. If m is supplied as a single number, then it is taken to be m[1] and m[2]=m[1]-1, which is only a conventional smoothing spline in the m=3, cubic spline case. Notice that the definition of the spline order in terms of m[1] is intuitive, but differs to that used with the tprs and p.spline bases.

## Usage

```
## $3 method for class 'bs.smooth.spec'
smooth.construct(object, data, knots)
## $3 method for class 'Bspline.smooth'
Predict.matrix(object, data)
```

#### **Arguments**

object	a smooth specification object, usually generated by a term $s(x,bs="bs",)$
data	a list containing just the data (including any by variable) required by this term, with names corresponding to object\$term (and object\$by). The by variable is the last element.
knots	a list containing any knots supplied for basis setup — in same order and with same names as data. Can be NULL. See details for further information.

## **Details**

The basis and penalty are sparse (although sparse matrices are not used to represent them). m[2]>m[1] will generate an error, since in that case the penalty would be based on an undefined derivative of the basis, which makes no sense.

The default basis dimension, k, is the larger of 10 and m. m[1] is the lower limit on basis dimension. If knots are supplied, then the number of supplied knots should be k + m[1] + 1, and the range of the middle k-m[1]+1 knots should include all the covariate values. Alternatively, 2 knots can be supplied, denoting the lower and upper limits between which the spline can be evaluated (Don't make this range too wide, however, or you can end up with no information about some basis coefficients, because the corresponding basis functions have a span that includes no data!). Unlike P-splines, splines with derivative based penalties can have uneven knot spacing, without a problem.

Linear extrapolation is used for prediction that requires extrapolation (i.e. prediction outside the range of the interior k-m[1]+1 knots). Such extrapolation is not allowed in basis construction, but is when predicting.

It is possible to set a deriv flag in a smooth specification or smooth object, so that a model or prediction matrix produces the requested derivative of the spline, rather than evaluating it.

#### Value

An object of class "Bspline.smooth". See smooth.construct, for the elements that this object will contain.

#### WARNING

m directly controls the spline order here, which is intuitively sensible, but different to other bases.

## Author(s)

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

#### See Also

```
p.spline
```

```
require(mgcv)
set.seed(5)
dat <- gamSim(1,n=400,dist="normal",scale=2)</pre>
bs <- "bs"
b < -gam(y^s(x0,bs=bs,m=c(4,2))+s(x1,bs=bs)+s(x2,k=15,bs=bs,m=c(4,3))+
          s(x3,bs=bs,m=c(1,0)),data=dat,method="REML")
plot(b,pages=1)
## construct smooth of x. Model matrix sm$X and penalty
## matrix sm$S[[1]] will have many zero entries...
x < - seq(0,1,length=100)
sm <- smoothCon(s(x,bs="bs"),data.frame(x))[[1]]</pre>
## another example checking penalty numerically...
m \leftarrow c(4,2); k \leftarrow 15; b \leftarrow runif(k)
sm <- smoothCon(s(x,bs="bs",m=m,k=k),data.frame(x),scale.penalty=FALSE)[[1]]</pre>
sm$deriv <- m[2]</pre>
h0 \leftarrow 1e-3; xk \leftarrow sm$knots[(m[1]+1):(k+1)]
```

```
\label{eq:continuous} $$Xp <- \operatorname{PredictMat}(sm, \operatorname{data.frame}(x=seq(xk[1]+h0/2, \max(xk)-h0/2, h0))) $$sum((Xp%*%b)^2*h0) $$\#$ numerical approximation to penalty $b%*%sm$S[[1]]%*%b $$\#$ `exact' version
```

```
smooth.construct.cr.smooth.spec
```

Penalized Cubic regression splines in GAMs

# Description

gam can use univariate penalized cubic regression spline smooths, specified via terms like s(x,bs="cr"). s(x,bs="cs") specifies a penalized cubic regression spline which has had its penalty modified to shrink towards zero at high enough smoothing parameters (as the smoothing parameter goes to infinity a normal cubic spline tends to a straight line.) s(x,bs="cc") specifies a cyclic penalized cubic regression spline smooth.

'Cardinal' spline bases are used: Wood (2006) sections 4.1.2 and 4.1.3 gives full details. These bases have very low setup costs. For a given basis dimension, k, they typically perform a little less well then thin plate regression splines, but a little better than p-splines. See te to use these bases in tensor product smooths of several variables.

Default k is 10.

## Usage

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

## **Arguments**

object	a smooth specification object, usually generated by a term s(,bs="cr",), s(,bs="cs",) or s(,bs="cc",)
data	a list containing just the data (including any by variable) required by this term, with names corresponding to object\$term (and object\$by). The by variable is the last element.
knots	a list containing any knots supplied for basis setup — in same order and with same names as data. Can be NULL. See details.

# Details

The constructor is not normally called directly, but is rather used internally by gam. To use for basis setup it is recommended to use smooth.construct2.

If they are not supplied then the knots of the spline are placed evenly throughout the covariate values to which the term refers: For example, if fitting 101 data with an 11 knot spline of x then there would be a knot at every 10th (ordered) x value. The parameterization used represents the spline in terms of its values at the knots. The values at neighbouring knots are connected by sections of cubic polynomial constrained to be continuous up to and including second derivative at the knots. The resulting curve is a natural cubic spline through the values at the knots (given two extra conditions specifying that the second derivative of the curve should be zero at the two end knots).

The shrinkage version of the smooth, eigen-decomposes the wiggliness penalty matrix, and sets its 2 zero eigenvalues to small multiples of the smallest strictly positive eigenvalue. The penalty is then set to the matrix with eigenvectors corresponding to those of the original penalty, but eigenvalues set to the peturbed versions. This penalty matrix has full rank and shrinks the curve to zero at high enough smoothing parameters.

Note that the cyclic smoother will wrap at the smallest and largest covariate values, unless knots are supplied. If only two knots are supplied then they are taken as the end points of the smoother (provided all the data lie between them), and the remaining knots are generated automatically.

The cyclic smooth is not subject to the condition that second derivatives go to zero at the first and last knots.

## Value

An object of class "cr.smooth" "cs.smooth" or "cyclic.smooth". In addition to the usual elements of a smooth class documented under smooth.construct, this object will contain:

xp giving the knot locations used to generate the basis.

BD class "cyclic.smooth" objects include matrix BD which transforms function values at the knots to second derivatives at the knots.

# Author(s)

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

# References

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

```
## cyclic spline example...
  require(mgcv)
  set.seed(6)
  x <- sort(runif(200)*10)
  z <- runif(200)
  f <- sin(x*2*pi/10)+.5
  y <- rpois(exp(f),exp(f))

## finished simulating data, now fit model...
  b <- gam(y ~ s(x,bs="cc",k=12) + s(z),family=poisson,knots=list(x=seq(0,10,length=12)))</pre>
```

```
smooth.construct.ds.smooth.spec

Low rank Duchon 1977 splines
```

# **Description**

Thin plate spline smoothers are a special case of the isotropic splines discussed in Duchon (1977). A subset of this more general class can be invoked by terms like s(x,z,bs="ds",m=c(1,.5)) in a gam model formula. In the notation of Duchon (1977) m is given by m[1] (default value 2), while s is given by m[2] (default value 0).

Duchon's (1977) construction generalizes the usual thin plate spline penalty as follows. The usual TPS penalty is given by the integral of the squared Euclidian norm of a vector of mixed partial mth order derivatives of the function w.r.t. its arguments. Duchon re-expresses this penalty in the Fourier domain, and then weights the squared norm in the integral by the Euclidean norm of the fourier frequencies, raised to the power 2s. s is a user selected constant taking integer values divided by 2. If d is the number of arguments of the smooth, then it is required that -d/2 < s < d/2. To obtain continuous functions we further require that m + s > d/2. If s=0 then the usual thin plate spline is recovered.

The construction is amenable to exactly the low rank approximation method given in Wood (2003) to thin plate splines, with similar optimality properties, so this approach to low rank smoothing is used here. For large datasets the same subsampling approach as is used in the tprs case is employed here to reduce computational costs.

These smoothers allow the use of lower orders of derivative in the penalty than conventional thin plate splines, while still yielding continuous functions. For example, we can set m = 1 and s = d/2. 5 in order to use first derivative penalization for any d (which has the advantage that the dimension of the null space of unpenalized functions is only d+1).

#### **Usage**

```
## S3 method for class 'ds.smooth.spec'
smooth.construct(object, data, knots)
## S3 method for class 'duchon.spline'
Predict.matrix(object, data)
```

#### **Arguments**

object a smooth specification object, usually generated by a term s(...,bs="ds",...).

data a list containing just the data (including any by variable) required by this term,

with names corresponding to object\$term (and object\$by). The by variable

is the last element.

knots a list containing any knots supplied for basis setup — in same order and with

same names as data. Can be NULL

#### Details

The default basis dimension for this class is k=M+k.def where M is the null space dimension (dimension of unpenalized function space) and k.def is 10 for dimension 1, 30 for dimension 2 and 100 for higher dimensions. This is essentially arbitrary, and should be checked, but as with all penalized regression smoothers, results are statistically insensitive to the exact choise, provided it is not so small that it forces oversmoothing (the smoother's degrees of freedom are controlled primarily by its smoothing parameter).

The constructor is not normally called directly, but is rather used internally by gam. To use for basis setup it is recommended to use smooth.construct2.

For these classes the specification object will contain information on how to handle large datasets in their xt field. The default is to randomly subsample 2000 'knots' from which to produce a reduced rank eigen approximation to the full basis, if the number of unique predictor variable combinations in excess of 2000. The default can be modified via the xt argument to s. This is supplied as a list with elements max.knots and seed containing a number to use in place of 2000, and the random number seed to use (either can be missing). Note that the random sampling will not effect the state of R's RNG.

For these bases knots has two uses. Firstly, as mentioned already, for large datasets the calculation of the tp basis can be time-consuming. The user can retain most of the advantages of the approach by supplying a reduced set of covariate values from which to obtain the basis - typically the number of covariate values used will be substantially smaller than the number of data, and substantially larger than the basis dimension, k. This approach is the one taken automatically if the number of unique covariate values (combinations) exceeds max.knots. The second possibility is to avoid the eigen-decomposition used to find the spline basis altogether and simply use the basis implied by the chosen knots: this will happen if the number of knots supplied matches the basis dimension, k. For a given basis dimension the second option is faster, but gives poorer results (and the user must be quite careful in choosing knot locations).

# Value

An object of class "duchon.spline". In addition to the usual elements of a smooth class documented under smooth.construct, this object will contain:

shift A record of the shift applied to each covariate in order to center it around zero

and avoid any co-linearity problems that might otehrwise occur in the penalty

null space basis of the term.

Xu A matrix of the unique covariate combinations for this smooth (the basis is con-

structed by first stripping out duplicate locations).

UZ The matrix mapping the smoother parameters back to the parameters of a full Duchon spline.

```
null.space.dimension
```

The dimension of the space of functions that have zero wiggliness according to the wiggliness penalty for this term.

#### Author(s)

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

#### References

Duchon, J. (1977) Splines minimizing rotation-invariant semi-norms in Solobev spaces. in W. Shemp and K. Zeller (eds) Construction theory of functions of several variables, 85-100, Springer, Berlin

Wood, S.N. (2003) Thin plate regression splines. J.R.Statist.Soc.B 65(1):95-114

# See Also

```
Spherical.Spline
```

# **Examples**

```
require(mgcv)
eg <- gamSim(2,n=200,scale=.05)
attach(eg)
op <- par(mfrow=c(2,2),mar=c(4,4,1,1))
b0 <- gam(y~s(x,z,bs="ds",m=c(2,0),k=50),data=data)  ## tps
b <- gam(y~s(x,z,bs="ds",m=c(1,.5),k=50),data=data)  ## first deriv penalty
b1 <- gam(y~s(x,z,bs="ds",m=c(2,.5),k=50),data=data)  ## modified 2nd deriv

persp(truth$x,truth$z,truth$f,theta=30) ## truth
vis.gam(b0,theta=30)
vis.gam(b1,theta=30)
vis.gam(b1,theta=30)</pre>
```

smooth.construct.fs.smooth.spec

Factor smooth interactions in GAMs

# Description

Simple factor smooth interactions, which are efficient when used with gamm. This smooth class allows a separate smooth for each level of a factor, with the same smoothing parameter for all smooths. It is an alternative to using factor by variables.

See the discussion of by variables in gam.models for more general alternatives for factor smooth interactions (including interactions of tensor product smooths with factors).

#### **Usage**

```
## $3 method for class 'fs.smooth.spec'
smooth.construct(object, data, knots)
## $3 method for class 'fs.interaction'
Predict.matrix(object, data)
```

# Arguments

data

object For the smooth construct method a smooth specification object, usually generated by a term s(x, ..., bs="fs",). For the predict Matrix method an object of the smooth construct method as smooth specification object, usually generated by a term s(x, ..., bs="fs",). For the predict Matrix method an object of the smooth construct method as smooth specification object, usually generated by a term s(x, ..., bs="fs",).

ject of class "fs.interaction" produced by the smooth.construct method.

a list containing just the data (including any by variable) required by this term,

with names corresponding to object\$term.

knots a list containing any knots supplied for smooth basis setup.

#### **Details**

This class produces a smooth for each level of a single factor variable. Within a gam formula this is done with something like s(x,fac,bs="fs"), which is almost equivalent to s(x,by=fac,id=1) (with the gam argument select=TRUE). The terms are fully penalized, with separate penalties on each null space component: for this reason they are not centred (no sum-to-zero constraint).

The class is particularly useful for use with gamm, where estimation efficiently exploits the nesting of the smooth within the factor. Note however that: i) gamm only allows one conditioning factor for smooths, so s(x)+s(z,fac,bs="fs")+s(v,fac,bs="fs") is OK, but s(x)+s(z,fac1,bs="fs")+s(v,fac2,bs="fs") is not; ii) all aditional random effects and correlation structures will be treated as nested within the factor of the smooth factor interaction.

Note that gamm4 from the gamm4 package suffers from none of the restrictions that apply to gamm, and "fs" terms can be used without side-effects.

Any singly penalized basis can be used to smooth at each factor level. The default is "tp", but alternatives can be supplied in the xt argument of s (e.g. s(x,fac,bs="fs",xt="cr")) or s(x,fac,bs="fs",xt=list(bs="cr")). The k argument to s(...,bs="fs") refers to the basis dimension to use for each level of the factor variable.

Note one computational bottleneck: currently gamm (or gamm4) will produce the full posterior covariance matrix for the smooths, including the smooths at each level of the factor. This matrix can get large and computationally costly if there are more than a few hundred levels of the factor. Even at one or two hundred levels, care should be taken to keep down k.

The plot method for this class has two schemes. scheme==0 is in colour, while scheme==1 is black and white.

## Value

An object of class "fs.interaction" or a matrix mapping the coefficients of the factor smooth interaction to the smooths themselves.

#### Author(s)

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

#### See Also

```
gam.models, gamm
```

## **Examples**

```
library(mgcv)
set.seed(0)
## simulate data...
f0 \leftarrow function(x) 2 * sin(pi * x)
f1 \leftarrow function(x,a=2,b=-1) \exp(a * x)+b
f2 \leftarrow function(x) 0.2 * x^11 * (10 * (1 - x))^6 + 10 *
             (10 * x)^3 * (1 - x)^10
n <- 500;nf <- 25
fac <- sample(1:nf,n,replace=TRUE)</pre>
x0 <- runif(n);x1 <- runif(n);x2 <- runif(n)</pre>
a <- rnorm(nf)*.2 + 2;b <- rnorm(nf)*.5
f \leftarrow f0(x0) + f1(x1,a[fac],b[fac]) + f2(x2)
fac <- factor(fac)</pre>
y \leftarrow f + rnorm(n)*2
## so response depends on global smooths of x0 and
## x2, and a smooth of x1 for each level of fac.
## fit model (note p-values not available when fit
## using gamm)...
bm <- gamm(y \sim s(x0) + s(x1, fac, bs = "fs", k = 5) + s(x2, k = 20))
plot(bm$gam,pages=1)
summary(bm$gam)
## Could also use...
## b <- gam(y\sim s(x0) + s(x1,fac,bs="fs",k=5)+s(x2,k=20),method="ML")
## ... but its slower (increasingly so with increasing nf)
## b <- gam(y\sim s(x0) + t2(x1,fac,bs=c("tp","re"),k=5,full=TRUE) +
##
          s(x2,k=20), method="ML"))
## ... is exactly equivalent.
```

smooth.construct.gp.smooth.spec

Low rank Gaussian process smooths

## **Description**

Gaussian process/kriging models based on simple covariance functions can be written in a very similar form to thin plate and Duchon spline models (e.g. Handcock, Meier, Nychka, 1994), and low rank versions produced by the eigen approximation method of Wood (2003). Kammann and Wand (2003) suggest a particularly simple form of the Matern covariance function with only a single smoothing parameter to estimate, and this class implements this and other similar models.

Usually invoked by an s(...,bs="gp") term in a gam formula. Argument m selects the covariance function, sets the range parameter and any power parameter. If m is not supplied then it defaults to NA

and the covariance function suggested by Kammann and Wand (2003) along with their suggested range parameter is used. Otherwise m[1] between 1 and 5 selects the correlation function from respectively, spherical, power exponential, and Matern with kappa = 1.5, 2.5 or 3.5. m[2] if present specifies the range parameter, with non-positive or absent indicating that the Kammann and Wand estimate should be used. m[3] can be used to specify the power for the power exponential which otherwise defaults to 1.

# Usage

```
## $3 method for class 'gp.smooth.spec'
smooth.construct(object, data, knots)
## $3 method for class 'gp.smooth'
Predict.matrix(object, data)
```

## **Arguments**

object	a smooth specification object, usually generated by a term $s(,bs="ms",)$ .
data	a list containing just the data (including any by variable) required by this term, with names corresponding to object\$term (and object\$by). The by variable is the last element.
knots	a list containing any knots supplied for basis setup — in same order and with same names as data. Can be NULL

## Details

Let  $\rho > 0$  be the range parameter,  $0 < \kappa \le 2$  and d denote the distance between two points. Then the correlation functions indexed by m[1] are:

```
1. 1-1.5d/\rho+0.5(d/\rho)^3 if d\leq \rho and 0 otherwise.

2. \exp(-(d/\rho)^\kappa).

3. \exp(-d/\rho)(1+d/\rho).

4. \exp(-d/\rho)(1+d/\rho+(d/\rho)^2/3).

5. \exp(-d/\rho)(1+d/\rho+2(d/\rho)^2/5+(d/\rho)^3/15).
```

See Fahrmeir et al. (2013) section 8.1.6, for example.

The default basis dimension for this class is k=M+k.def where M is the null space dimension (dimension of unpenalized function space) and k.def is 10 for dimension 1, 30 for dimension 2 and 100 for higher dimensions. This is essentially arbitrary, and should be checked, but as with all penalized regression smoothers, results are statistically insensitive to the exact choise, provided it is not so small that it forces oversmoothing (the smoother's degrees of freedom are controlled primarily by its smoothing parameter).

The constructor is not normally called directly, but is rather used internally by gam. To use for basis setup it is recommended to use smooth.construct2.

For these classes the specification object will contain information on how to handle large datasets in their xt field. The default is to randomly subsample 2000 'knots' from which to produce a reduced rank eigen approximation to the full basis, if the number of unique predictor variable combinations in excess of 2000. The default can be modified via the xt argument to s. This is supplied

as a list with elements max.knots and seed containing a number to use in place of 2000, and the random number seed to use (either can be missing). Note that the random sampling will not effect the state of R's RNG.

For these bases knots has two uses. Firstly, as mentioned already, for large datasets the calculation of the tp basis can be time-consuming. The user can retain most of the advantages of the approach by supplying a reduced set of covariate values from which to obtain the basis - typically the number of covariate values used will be substantially smaller than the number of data, and substantially larger than the basis dimension, k. This approach is the one taken automatically if the number of unique covariate values (combinations) exceeds max.knots. The second possibility is to avoid the eigen-decomposition used to find the spline basis altogether and simply use the basis implied by the chosen knots: this will happen if the number of knots supplied matches the basis dimension, k. For a given basis dimension the second option is faster, but gives poorer results (and the user must be quite careful in choosing knot locations).

## Value

An object of class "gp.smooth". In addition to the usual elements of a smooth class documented under smooth.construct, this object will contain:

shift	A record of the shift applied to each covariate in order to center it around zero
SIIIIC	Trecord of the shift applied to each covariate in order to center it around zero

and avoid any co-linearity problems that might otehrwise occur in the penalty

null space basis of the term.

Xu A matrix of the unique covariate combinations for this smooth (the basis is con-

structed by first stripping out duplicate locations).

UZ The matrix mapping the smoother parameters back to the parameters of a full

GP smooth.

null.space.dimension

The dimension of the space of functions that have zero wiggliness according to

the wiggliness penalty for this term.

gp. defn the type, range parameter and power parameter defining the correlation function.

#### Author(s)

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

### References

Fahrmeir, L., T. Kneib, S. Lang and B. Marx (2013) Regression, Springer.

Handcock, M. S., K. Meier and D. Nychka (1994) Journal of the American Statistical Association, 89: 401-403

Kammann, E. E. and M.P. Wand (2003) Geoadditive Models. Applied Statistics 52(1):1-18.

Wood, S.N. (2003) Thin plate regression splines. J.R.Statist.Soc.B 65(1):95-114

#### See Also

tprs

## **Examples**

```
require(mgcv)
eg <- gamSim(2,n=200,scale=.05)
attach(eg)
op <- par(mfrow=c(2,2),mar=c(4,4,1,1))
b0 <- gam(y~s(x,z,k=50),data=data)  ## tps
b <- gam(y~s(x,z,bs="gp",k=50),data=data)  ## Matern spline default range
b1 <- gam(y~s(x,z,bs="gp",k=50,m=c(1,.5)),data=data)  ## spherical

persp(truth$x,truth$z,truth$f,theta=30) ## truth
vis.gam(b0,theta=30)
vis.gam(b,theta=30)
vis.gam(b1,theta=30)
detach(eg)</pre>
```

# **Description**

For data observed over discrete spatial units, a simple Markov random field smoother is sometimes appropriate. These functions provide such a smoother class for mgcv. See details for how to deal with regions with missing data.

# Usage

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

# **Arguments**

object	For the smooth.construct method a smooth specification object, usually generated by a term $s(x,, bs="mrf", xt=list(polys=foo))$ . x is a factor variable giving labels for geographic districts, and the xt argument is obligatory: see details. For the Predict.Matrix method an object of class "mrf.smooth" produced by the smooth.construct method.
data	a list containing just the data (including any by variable) required by this term, with names corresponding to object\$term (and object\$by). The by variable is the last element.
knots	If there are more geographic areas than data were observed for, then this argument is used to provide the labels for all the areas (observed and unobserved).

#### **Details**

A Markov random field smooth over a set of discrete areas is defined using a set of area labels, and a neighbourhood structure for the areas. The covariate of the smooth is the vector of area labels corresponding to each obervation. This covariate should be a factor, or capable of being coerced to a factor.

The neighbourhood structure is supplied in the xt argument to s. This must contain at least one of the elements polys, nb or penalty.

polys contains the polygons defining the geographic areas. It is a list with as many elements as there are geographic areas. names(polys) must correspond to the levels of the argument of the smooth, in any order (i.e. it gives the area labels). polys[[i]] is a 2 column matrix the rows of which specify the vertices of the polygon(s) defining the boundary of the ith area. A boundary may be made up of several closed loops: these must be separated by NA rows. A polygon within another is treated as a hole. The first polygon in any polys[[i]] should not be a hole. An example of the structure is provided by columb.polys (which contains an artificial hole in its second element, for illustration). Any list elements with duplicate names are combined into a single NA separated matrix.

Plotting of the smooth is not possible without a polys object.

If polys is the only element of xt provided, then the neighbourhood structure is computed from it automatically. To count as neigbours, polygons must exactly share one of more vertices.

nb is a named list defining the neighbourhood structure. names(nb) must correspond to the levels of the covariate of the smooth (i.e. the area labels), but can be in any order. nb[[i]] is a vector indexing the neighbours of the ith area. All indices are relative to nb itself, but can be translated using names(nb).

If no penalty is provided then it is computed automatically from this list. The ith row of the penalty matrix will be zero everwhere, except in the ith column, which will contain the number of neighbours of the ith geographic area, and the columns corresponding to those geographic neighbours, which will each contain -1.

**penalty** if this is supplied, then it is used as the penalty matrix. It should be positive semi-definite. Its row and column names should correspond to the levels of the covariate.

If no basis dimension is supplied then the constructor produces a full rank MRF, with a coefficient for each geographic area. Otherwise a low rank approximation is obtained based on truncation of the parameterization given in Wood (2006) Section 4.10.4.

Note that smooths of this class have a built in plot method, and that the utility function in.out can be useful for working with discrete area data. The plot method has two schemes, scheme==0 is colour, scheme==1 is grey scale.

The situation in which there are areas with no data requires special handling. You should set drop.unused.levels=FALSE in the model fitting function, gam, bam or gamm, having first ensured that any fixed effect factors do not contain unobserved levels. Also make sure that the basis dimension is set to ensure that the total number of coefficients is less than the number of observations.

#### Value

An object of class "mrf.smooth" or a matrix mapping the coefficients of the MRF smooth to the predictions for the areas listed in data.

#### Author(s)

Simon N. Wood <simon.wood@r-project.org> and Thomas Kneib (Fabian Scheipl prototyped the low rank MRF idea)

#### References

Wood S.N. (2006) Generalized additive models: an introduction with R. CRC.

## See Also

```
in.out, polys.plot
```

## **Examples**

```
library(mgcv)
## Load Columbus Ohio crime data (see ?columbus for details and credits)
data(columb)
                   ## data frame
data(columb.polys) ## district shapes list
xt <- list(polys=columb.polys) ## neighbourhood structure info for MRF
par(mfrow=c(2,2))
## First a full rank MRF...
b <- gam(crime ~ s(district,bs="mrf",xt=xt),data=columb,method="REML")</pre>
plot(b, scheme=1)
## Compare to reduced rank version...
b <- gam(crime ~ s(district,bs="mrf",k=20,xt=xt),data=columb,method="REML")</pre>
plot(b, scheme=1)
## An important covariate added...
b <- gam(crime ~ s(district,bs="mrf",k=20,xt=xt)+s(income),</pre>
         data=columb,method="REML")
plot(b, scheme=c(0,1))
## plot fitted values by district
par(mfrow=c(1,1))
fv <- fitted(b)</pre>
names(fv) <- as.character(columb$district)</pre>
polys.plot(columb.polys,fv)
```

```
smooth.construct.ps.smooth.spec

P-splines in GAMs
```

## **Description**

gam can use univariate P-splines as proposed by Eilers and Marx (1996), specified via terms like s(x,bs="ps"). These terms use B-spline bases penalized by discrete penalties applied directly to the basis coefficients. Cyclic P-splines are specified by model terms like s(x,bs="cp",...). These bases can be used in tensor product smooths (see te).

The advantage of P-splines is the flexible way that penalty and basis order can be mixed. This often provides a useful way of 'taming' an otherwise poorly behave smooth. However, in regular use, splines with derivative based penalties (e.g. "tp" or "cr" bases) tend to result in slightly better MSE performance, presumably because the good approximation theoretic properties of splines are rather closely connected to the use of derivative penalties.

## Usage

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

## **Arguments**

object	a smooth specification object, usually generated by a term $s(x,bs="ps",)$
	on o(v, ho="on" )

or s(x,bs="cp",...)

data a list containing just the data (including any by variable) required by this term,

with names corresponding to object\$term (and object\$by). The by variable

is the last element.

knots a list containing any knots supplied for basis setup — in same order and with

same names as data. Can be NULL. See details for further information.

### **Details**

A smooth term of the form s(x,bs="ps",m=c(2,3)) specifies a 2nd order P-spline basis (cubic spline), with a third order difference penalty (0th order is a ridge penalty) on the coefficients. If m is a single number then it is taken as the basis order and penalty order. The default is the 'cubic spline like' m=c(2,2).

The default basis dimension, k, is the larger of 10 and m[1]+1 for a "ps" terms and the larger of 10 and m[1] for a "cp" term. m[1]+1 and m[1] are the lower limits on basis dimension for the two types.

If knots are supplied, then the number of knots should be one more than the basis dimension (i.e. k+1) for a "cp"smooth. For the "ps" basis the number of supplied knots should be k+m[1]+2, and the range of the middle k-m[1] knots should include all the covariate values. See example.

Alternatively, for both types of smooth, 2 knots can be supplied, denoting the lower and upper limits between which the spline can be evaluated (Don't make this range too wide, however, or you can end up with no information about some basis coefficients, because the corresponding basis functions have a span that includes no data!). Note that P-splines don't make much sense with uneven knot spacing.

Linear extrapolation is used for prediction that requires extrapolation (i.e. prediction outside the range of the interior k-m[1] knots). Such extrapolation is not allowed in basis construction, but is when predicting.

For the "ps" basis it is possible to set flags in the smooth specification object, requesting setup according to the SCOP-spline monotonic smoother construction of Pya and Wood (2015). As yet this is not supported by any modelling functions in mgcv (see package scam). Similarly it is possible

to set a deriv flag in a smooth specification or smooth object, so that a model or prediction matrix produces the requested derivative of the spline, rather than evaluating it. See examples below.

#### Value

An object of class "pspline.smooth" or "cp.smooth". See smooth.construct, for the elements that this object will contain.

## Author(s)

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

#### References

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

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

#### See Also

```
cSplineDes, adaptive.smooth
```

```
## see ?gam
## cyclic example ...
  require(mgcv)
  set.seed(6)
  x <- sort(runif(200)*10)</pre>
  z \leftarrow runif(200)
  f <- \sin(x*2*pi/10)+.5
  y <- rpois(exp(f),exp(f))</pre>
## finished simulating data, now fit model...
  b \leftarrow gam(y \sim s(x,bs="cp") + s(z,bs="ps"),family=poisson)
## example with supplied knot ranges for x and z (can do just one)
  b \leftarrow gam(y \sim s(x,bs="cp") + s(z,bs="ps"),family=poisson,
            knots=list(x=c(0,10),z=c(0,1)))
## example with supplied knots...
  bk \leftarrow gam(y \sim s(x,bs="cp",k=12) + s(z,bs="ps",k=13),family=poisson,
                        knots=list(x=seq(0,10,length=13),z=(-3):13/10))
## plot results...
  par(mfrow=c(2,2))
  plot(b, select=1, shade=TRUE); lines(x, f-mean(f), col=2)
  plot(b, select=2, shade=TRUE); lines(z,0*z,col=2)
  plot(bk, select=1, shade=TRUE); lines(x, f-mean(f), col=2)
  plot(bk, select=2, shade=TRUE); lines(z, 0*z, col=2)
```

```
## Example using montonic constraints via the SCOP-spline
## construction, and of computng derivatives...
 x \leftarrow seq(0,1,length=100); dat \leftarrow data.frame(x)
 sspec <- s(x,bs="ps")
 sspec$mono <- 1
 sm <- smoothCon(sspec,dat)[[1]]</pre>
 sm$deriv <- 1
 Xd <- PredictMat(sm,dat)</pre>
## generate random coeffients in the unconstrainted
## parameterization...
 b <- runif(10)*3-2.5
## exponentiate those parameters indicated by sm$g.index
## to obtain coefficients meeting the constraints...
 b[sm$g.index] <- exp(b[sm$g.index])</pre>
## plot monotonic spline and its derivative
 par(mfrow=c(2,2))
 plot(x,sm$X%*%b,type="l",ylab="f(x)")
 plot(x,Xd%*%b,type="l",ylab="f'(x)")
## repeat for decrease...
 sspec$mono <- -1
 sm1 <- smoothCon(sspec,dat)[[1]]</pre>
 sm1$deriv <- 1
 Xd1 <- PredictMat(sm1,dat)</pre>
 plot(x,sm1$X%*\%b,type="l",ylab="f(x)")
 plot(x,Xd1%*%b,type="l",ylab="f'(x)")
## Now with sum to zero constraints as well...
 sspec$mono <- 1
 sm <- smoothCon(sspec,dat,absorb.cons=TRUE)[[1]]</pre>
 sm$deriv <- 1
 Xd <- PredictMat(sm,dat)</pre>
 b <- b[-1] ## dropping first param
 plot(x,sm$X%*%b,type="l",ylab="f(x)")
 plot(x,Xd%*%b,type="l",ylab="f'(x)")
 sspec$mono <- -1
 sm1 <- smoothCon(sspec,dat,absorb.cons=TRUE)[[1]]</pre>
 sm1$deriv <- 1
 Xd1 <- PredictMat(sm1,dat)</pre>
 plot(x,sm1$X%*\%b,type="l",ylab="f(x)")
 plot(x,Xd1\%*b,type="l",ylab="f'(x)")
```

#### **Description**

gam can deal with simple independent random effects, by exploiting the link between smooths and random effects to treat random effects as smooths. s(x,bs="re") implements this. Such terms can can have any number of predictors, which can be any mixture of numeric or factor variables. The terms produce a parametric interaction of the predictors, and penalize the corresponding coefficients with a multiple of the identity matrix, corresponding to an assumption of i.i.d. normality. See details.

### Usage

```
## $3 method for class 're.smooth.spec'
smooth.construct(object, data, knots)
## $3 method for class 'random.effect'
Predict.matrix(object, data)
```

present.

### **Arguments**

object	For the smooth.construct method a smooth specification object, usually generated by a term $s(x,,bs="re",)$ . For the predict.Matrix method an object of class "random.effect" produced by the smooth.construct method.
data	a list containing just the data (including any by variable) required by this term, with names corresponding to object\$term (and object\$by). The by variable is the last element.
knots	generically a list containing any knots supplied for basis setup — unused at

#### **Details**

Exactly how the random effects are implemented is best seen by example. Consider the model term s(x,z,bs="re"). This will result in the model matrix component corresponding to  $\sim x:z-1$  being added to the model matrix for the whole model. The coefficients associated with the model matrix component are assumed i.i.d. normal, with unknown variance (to be estimated). This assumption is equivalent to an identity penalty matrix (i.e. a ridge penalty) on the coefficients. Because such a penalty is full rank, random effects terms do not require centering constraints.

If the nature of the random effect specification is not clear, consider a couple more examples: s(x,bs="re") results in model.matrix( $\sim x-1$ ) being appended to the overall model matrix, while s(x,v,w,bs="re") would result in model.matrix( $\sim x:v:w-1$ ) being appended to the model matrix. In both cases the corresponding model coefficients are assumed i.i.d. normal, and are hence subject to ridge penalties.

If the random effect precision matrix is of the form  $\sum_j \lambda_j S_j$  for known matrices  $S_j$  and unknown parameters  $\lambda_j$ , then a list containing the  $S_j$  can be supplied in the xt argument of s. In this case an array rank should also be supplied in xt giving the ranks of the  $S_j$  matrices. See simple example below.

Note that smooth ids are not supported for random effect terms. Unlike most smooth terms, side conditions are never applied to random effect terms in the event of nesting (since they are identifiable without side conditions).

Random effects implemented in this way do not exploit the sparse structure of many random effects, and may therefore be relatively inefficient for models with large numbers of random effects, when gamm4 or gamm may be better alternatives. Note also that gam will not support models with more coefficients than data.

The situation in which factor variable random effects intentionally have unobserved levels requires special handling. You should set drop.unused.levels=FALSE in the model fitting function, gam, bam or gamm, having first ensured that any fixed effect factors do not contain unobserved levels.

#### Value

An object of class "random.effect" or a matrix mapping the coefficients of the random effect to the random effects themselves.

#### Author(s)

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

#### References

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

#### See Also

```
gam.vcomp, gamm
```

```
## see ?gam.vcomp
require(mgcv)
## simulate simple random effect example
set.seed(4)
nb <- 50; n <- 400
b <- rnorm(nb)*2 ## random effect
r <- sample(1:nb,n,replace=TRUE) ## r.e. levels
y \leftarrow 2 + b[r] + rnorm(n)
r <- factor(r)
## fit model....
b <- gam(y ~ s(r,bs="re"),method="REML")</pre>
gam.vcomp(b)
## example with supplied precision matrices...
b <- c(rnorm(nb/2)*2,rnorm(nb/2)*.5) ## random effect now with 2 variances
r <- sample(1:nb,n,replace=TRUE) ## r.e. levels
y \leftarrow 2 + b[r] + rnorm(n)
r <- factor(r)
## known precision matrix components...
S \leftarrow list(diag(rep(c(1,0),each=nb/2)),diag(rep(c(0,1),each=nb/2)))
b <- gam(y ~ s(r,bs="re",xt=list(S=S,rank=c(nb/2,nb/2))),method="REML")</pre>
gam.vcomp(b)
```

summary(b)

```
smooth.construct.so.smooth.spec

Soap film smoother constructer
```

# **Description**

Sets up basis functions and wiggliness penalties for soap film smoothers (Wood, Bravington and Hedley, 2008). Soap film smoothers are based on the idea of constructing a 2-D smooth as a film of soap connecting a smoothly varying closed boundary. Unless smoothing very heavily, the film is distorted towards the data. The smooths are designed not to smooth across boundary features (peninsulas, for example).

The so version sets up the full smooth. The sf version sets up just the boundary interpolating soap film, while the sw version sets up the wiggly component of a soap film (zero on the boundary). The latter two are useful for forming tensor products with soap films, and can be used with gamm and gamm4. To use these to simply set up a basis, then call via the wrapper smooth.construct2 or smoothCon.

## Usage

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

## **Arguments**

object A smooth specification object as produced by a s(...,bs="so",xt=list(bnd=bnd,...))

term in a gam formula. Note that the xt argument to s \*must\* be supplied, and should be a list, containing at least a boundary specification list (see details). xt may also contain various options controlling the boundary smooth (see details), and PDE solution grid. The dimension of the bases for boundary loops is specified via the k argument of s, either as a single number to be used for each boundary loop, or as a vector of different basis dimensions for the various

boundary loops.

data A list or data frame containing the arguments of the smooth.

knots list or data frame with two named columns specifying the knot locations within the boundary. The column names should match the names of the arguments of

the smooth. The number of knots defines the \*interior\* basis dimension (i.e. it

is \*not\* supplied via argument k of s).

#### **Details**

For soap film smooths the following \*must\* be supplied:

- k the basis dimension for each boundary loop smooth.
- xt\$bnd the boundary specification for the smooth.
- knots the locations of the interior knots for the smooth.

When used in a GAM then k and xt are supplied via s while knots are supplied in the knots argument of gam.

The bnd element of the xt list is a list of lists (or data frames), specifying the loops that define the boundary. Each boundary loop list must contain 2 columns giving the co-ordinates of points defining a boundary loop (when joined sequentially by line segments). Loops should not intersect (not checked). A point is deemed to be in the region of interest if it is interior to an odd number of boundary loops. Each boundary loop list may also contain a column f giving known boundary conditions on a loop.

The bndSpec element of xt, if non-NULL, should contain

- bs the type of cyclic smoothing basis to use: one of "cc" and "cp". If not "cc" then a cyclic p-spline is used, and argument m must be supplied.
- knot.space set to "even" to get even knot spacing with the "cc" basis.
- m 1 or 2 element array specifying order of "cp" basis and penalty.

Currently the code will not deal with more than one level of nesting of loops, or with separate loops without an outer enclosing loop: if there are known boundary conditions (identifiability constraints get awkward).

Note that the function locator provides a simple means for defining boundaries graphically, using something like bnd <-as.data.frame(locator(type="l")), after producing a plot of the domain of interest (right click to stop). If the real boundary is very complicated, it is probably better to use a simpler smooth boundary enclosing the true boundary, which represents the major boundary features that you don't want to smooth across, but doesn't follow every tiny detail.

Model set up, and prediction, involves evaluating basis functions which are defined as the solution to PDEs. The PDEs are solved numerically on a grid using sparse matrix methods, with bilinear interpolation used to obtain values at any location within the smoothing domain. The dimension of the PDE solution grid can be controlled via element nmax (default 200) of the list supplied as argument xt of s in a gam formula: it gives the number of cells to use on the longest grid side.

A little theory: the soap film smooth f(x, y) is defined as the solution of

$$f_{xx} + f_{yy} = g$$

subject to the condition that f = s, on the boundary curve, where s is a smooth function (usually a cyclic penalized regression spline). The function g is defined as the solution of

$$g_{xx} + g_{yy} = 0$$

where g=0 on the boundary curve and  $g(x_k,y_k)=c_k$  at the 'knots' of the surface; the  $c_k$  are model coefficients.

In the simplest case, estimation of the coefficients of f (boundary coefficients plus  $c_k$ 's) is by minimization of

$$||z - f||^2 + \lambda_s J_s(s) + \lambda_f J_f(f)$$

where  $J_s$  is usually some cubic spline type wiggliness penalty on the boundary smooth and  $J_f$  is the integral of  $(f_x x + f_y y)^2$  over the interior of the boundary. Both penalties can be expressed as quadratic forms in the model coefficients. The  $\lambda$ 's are smoothing parameters, selectable by GCV, REML, AIC, etc. z represents noisy observations of f.

# Value

A list with all the elements of object plus

sd	A list defining the PDE solution grid and domain boundary, and including the sparse LU factorization of the PDE coefficient matrix.
X	The model matrix: this will have an "offset" attribute, if there are any known boundary conditions.
S	List of smoothing penalty matrices (in smallest non-zero submatrix form).
irng	A vector of scaling factors that have been applied to the model matrix, to ensure nice conditioning.

In addition there are all the elements usually added by smooth.construct methods.

# WARNINGS

Soap film smooths are quite specialized, and require more setup than most smoothers (e.g. you have to supply the boundary and the interior knots, plus the boundary smooth basis dimension(s)). It is worth looking at the reference.

## Author(s)

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

### References

Wood, S.N., M.V. Bravington and S.L. Hedley (2008) "Soap film smoothing", J.R.Statist.Soc.B 70(5), 931-955.

http://www.maths.bris.ac.uk/~sw15190/

## See Also

Predict.matrix.soap.film

```
require(mgcv)
###################################
## simple test function...
#####################################
fsb <- list(fs.boundary())</pre>
nmax <- 100
## create some internal knots...
knots \leftarrow data.frame(v=rep(seq(-.5,3,by=.5),4),
                     w=rep(c(-.6,-.3,.3,.6),rep(8,4)))
## Simulate some fitting data, inside boundary...
set.seed(0)
n<-600
v <- runif(n)*5-1;w<-runif(n)*2-1</pre>
y \leftarrow fs.test(v,w,b=1)
\mathsf{names}(\mathsf{fsb}[[1]]) \mathrel{<\!\!\!\!-} \mathsf{c}("\mathsf{v"},"\mathsf{w"})
ind <- inSide(fsb,x=v,y=w) ## remove outsiders</pre>
y \leftarrow y + rnorm(n)*.3 \# add noise
y <- y[ind]; v <- v[ind]; w <- w[ind]</pre>
n <- length(y)</pre>
par(mfrow=c(3,2))
## plot boundary with knot and data locations
plot(fsb[[1]]$v,fsb[[1]]$w,type="1");points(knots,pch=20,col=2)
points(v,w,pch=".");
## Now fit the soap film smoother. 'k' is dimension of boundary smooth.
## boundary supplied in 'xt', and knots in 'knots'...
nmax <- 100 ## reduced from default for speed.
b <- gam(y~s(v,w,k=30,bs="so",xt=list(bnd=fsb,nmax=nmax)),knots=knots)</pre>
plot(b) ## default plot
plot(b,scheme=1)
plot(b, scheme=2)
plot(b, scheme=3)
vis.gam(b,plot.type="contour")
# Fit same model in two parts...
par(mfrow=c(2,2))
vis.gam(b,plot.type="contour")
b1 <- gam(y \sim s(v, w, k=30, bs="sf", xt=list(bnd=fsb, nmax=nmax))+
             s(v,w,k=30,bs="sw",xt=list(bnd=fsb,nmax=nmax)) ,knots=knots)
vis.gam(b,plot.type="contour")
```

```
plot(b1)
 ## Now an example with known boundary condition...
## Evaluate known boundary condition at boundary nodes...
fsb[[1]]$f <- fs.test(fsb[[1]]$v,fsb[[1]]$w,b=1,exclude=FALSE)</pre>
 ## Now fit the smooth...
bk \leftarrow gam(y^s(v,w,bs="so",xt=list(bnd=fsb,nmax=nmax)),knots=knots)
plot(bk) ## default plot
 ## tensor product example (marked
## 'Not run' to reduce CRAN checking load)
## Not run:
n <- 10000
v \leftarrow runif(n)*5-1; w\leftarrow runif(n)*2-1
t <- runif(n)
y \leftarrow fs.test(v,w,b=1)
y < -y + 4.2
y <- y^{(.5+t)}
fsb <- list(fs.boundary())</pre>
names(fsb[[1]]) <- c("v","w")
ind <- inSide(fsb,x=v,y=w) ## remove outsiders</pre>
y <- y[ind]; v <- v[ind]; w <- w[ind]; t <- t[ind]</pre>
n <- length(y)</pre>
y \leftarrow y + rnorm(n)*.05 ## add noise
knots \leftarrow data.frame(v=rep(seq(-.5,3,by=.5),4),
                                                                w=rep(c(-.6,-.3,.3,.6),rep(8,4)))
 ## notice NULL element in 'xt' list - to indicate no xt object for "cr" basis...
bk <- gam(y^{\sim})
   \texttt{te}(\texttt{v}, \texttt{w}, \texttt{t}, \texttt{bs=c}(\texttt{"sf"}, \texttt{"cr"}), \texttt{k=c}(25, 4), \texttt{d=c}(2, 1), \texttt{xt=list}(\texttt{list}(\texttt{bnd=fsb}, \texttt{nmax=nmax}), \texttt{NULL})) + \texttt{v} = \texttt{
   te(v,w,t,bs=c("sw","cr"),k=c(25,4),d=c(2,1),xt=list(list(bnd=fsb,nmax=nmax),NULL))
                                ,knots=knots)
par(mfrow=c(3,2))
m<-100; n<-50
xm \leftarrow seq(-1,3.5,length=m); yn \leftarrow seq(-1,1,length=n)
xx <- rep(xm,n);yy<-rep(yn,rep(m,n))</pre>
tru <- matrix(fs.test(xx,yy),m,n)+4.2 ## truth</pre>
 image(xm,yn,tru^.5,col=heat.colors(100),xlab="v",ylab="w",
                   main="truth")
lines(fsb[[1]]$v,fsb[[1]]$w,lwd=3)
contour(xm,yn,tru^.5,add=TRUE)
vis.gam(bk,view=c("v","w"),cond=list(t=0),plot.type="contour")
 image(xm,yn,tru,col=heat.colors(100),xlab="v",ylab="w",
```

```
main="truth")
lines(fsb[[1]]$v,fsb[[1]]$w,lwd=3)
contour(xm,yn,tru,add=TRUE)
vis.gam(bk,view=c("v","w"),cond=list(t=.5),plot.type="contour")
image(xm,yn,tru^1.5,col=heat.colors(100),xlab="v",ylab="w",
      main="truth")
lines(fsb[[1]]$v,fsb[[1]]$w,lwd=3)
contour(xm,yn,tru^1.5,add=TRUE)
vis.gam(bk,view=c("v","w"),cond=list(t=1),plot.type="contour")
## End(Not run)
# nested boundary example...
bnd <- list(list(x=0,y=0),list(x=0,y=0))
seq(0,2*pi,length=100) \rightarrow theta
bnd[[1]]x <- sin(theta); bnd[[1]]$y <- cos(theta)
bnd[[2]]$x <- .3 + .3*sin(theta);
bnd[[2]]$y <- .3 + .3*cos(theta)
plot(bnd[[1]]$x,bnd[[1]]$y,type="1")
lines(bnd[[2]]$x,bnd[[2]]$y)
## setup knots
k <- 8
xm \leftarrow seq(-1,1,length=k); ym \leftarrow seq(-1,1,length=k)
x=rep(xm,k);y=rep(ym,rep(k,k))
ind <- inSide(bnd,x,y)</pre>
knots <- data.frame(x=x[ind],y=y[ind])</pre>
points(knots$x,knots$y)
## a test function
f1 <- function(x,y) {</pre>
  \exp(-(x-.3)^2-(y-.3)^2)
## plot the test function within the domain
par(mfrow=c(2,3))
m<-100; n<-100
xm \leftarrow seq(-1,1,length=m); yn \leftarrow seq(-1,1,length=n)
x <- rep(xm,n);y<-rep(yn,rep(m,n))</pre>
ff \leftarrow f1(x,y)
ind <- inSide(bnd,x,y)</pre>
ff[!ind] <- NA
image(xm,yn,matrix(ff,m,n),xlab="x",ylab="y")
contour(xm,yn,matrix(ff,m,n),add=TRUE)
lines(bnd[[1]]$x,bnd[[1]]$y,lwd=2);lines(bnd[[2]]$x,bnd[[2]]$y,lwd=2)
```

```
## Simulate data by noisy sampling from test function...
set.seed(1)
x \leftarrow runif(300)*2-1; y \leftarrow runif(300)*2-1
ind <- inSide(bnd,x,y)</pre>
x \leftarrow x[ind]; y \leftarrow y[ind]
n <- length(x)</pre>
z \leftarrow f1(x,y) + rnorm(n)*.1
## Fit a soap film smooth to the noisy data
nmax <- 60
b \leftarrow gam(z^s(x,y,k=c(30,15),b="so",xt=list(bnd=bnd,nmax=nmax)),knots=knots,method="REML")
plot(b) ## default plot
vis.gam(b,plot.type="contour") ## prettier version
## trying out separated fits....
ba <- gam(z^s(x,y,k=c(30,15),bs="sf",xt=list(bnd=bnd,nmax=nmax)) +\\
         s(x,y,k=c(30,15),b="sw",xt=list(bnd=bnd,nmax=nmax)),knots=knots,method="REML")\\
plot(ba)
vis.gam(ba,plot.type="contour")
```

## Description

gam can use isotropic smooths on the sphere, via terms like s(1a,lo,bs="sos",m=2,k=100). There must be exactly 2 arguments to such a smooth. The first is taken to be latitude (in degrees) and the second longitude (in degrees). m (default 0) is an integer in the range -1 to 4 determining the order of the penalty used. For m>0, (m+2)/2 is the penalty order, with m=2 equivalent to the usual second derivative penalty. m=0 signals to use the 2nd order spline on the sphere, computed by Wendelberger's (1981) method. m=-1 results in a Duchon. spline being used (with m=2 and s=1/2), following an unpublished suggestion of Jean Duchon.

k (default 50) is the basis dimension.

#### Usage

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

## Arguments

object a smooth specification object, usually generated by a term s(...,bs="sos",...).

data a list containing just the data (including any by variable) required by this term,

with names corresponding to object\$term (and object\$by). The by variable

is the last element.

knots a list containing any knots supplied for basis setup — in same order and with

same names as data. Can be NULL

#### **Details**

For m>0, the smooths implemented here are based on the pseudosplines on the sphere of Wahba (1981) (there is a correction of table 1 in 1982, but the correction has a misprint in the definition of A — the A given in the 1981 paper is correct). For m=0 (default) then a second order spline on the sphere is used which is the analogue of a second order thin plate spline in 2D: the computation is based on Chapter 4 of Wendelberger, 1981. Optimal low rank approximations are obtained using exactly the approach given in Wood (2003). For m = -1 a smooth of the general type discussed in Duchon (1977) is used: the sphere is embedded in a 3D Euclidean space, but smoothing employs a penalty based on second derivatives (so that locally as the smoothing parameter tends to zero we recover a "normal" thin plate spline on the tangent space). This is an unpublished suggestion of Jean Duchon.

Note that the null space of the penalty is always the space of constant functions on the sphere, whatever the order of penalty.

This class has a plot method, with 3 schemes. scheme==0 plots one hemisphere of the sphere, projected onto a circle. The plotting sphere has the north pole at the top, and the 0 meridian running down the middle of the plot, and towards the viewer. The smoothing sphere is rotated within the plotting sphere, by specifying the location of its pole in the co-ordinates of the viewing sphere. theta, phi give the longitude and latitude of the smoothing sphere pole within the plotting sphere (in plotting sphere co-ordinates). (You can visualize the smoothing sphere as a globe, free to rotate within the fixed transparent plotting sphere.) The value of the smooth is shown by a heat map overlaid with a contour plot. lat, lon gridlines are also plotted.

scheme==1 is as scheme==0, but in black and white, without the image plot. scheme>1 calls the default plotting method with scheme decremented by 2.

## Value

An object of class "sos.smooth". In addition to the usual elements of a smooth class documented under smooth.construct, this object will contain:

Xu A matrix of the unique covariate combinations for this smooth (the basis is con-

structed by first stripping out duplicate locations).

UZ The matrix mapping the parameters of the reduced rank spline back to the pa-

rameters of a full spline.

# Author(s)

Simon Wood <simon.wood@r-project.org>, with help from Grace Wahba (m=0 case) and Jean Duchon (m = -1 case).

## References

```
Wahba, G. (1981) Spline interpolation and smoothing on the sphere. SIAM J. Sci. Stat. Comput. 2(1):5-16
Wahba, G. (1982) Erratum. SIAM J. Sci. Stat. Comput. 3(3):385-386.
Wendelberger, J. (1981) PhD Thesis, University of Winsconsin.
Wood, S.N. (2003) Thin plate regression splines. J.R.Statist.Soc.B 65(1):95-114
```

#### See Also

```
Duchon.spline
```

```
require(mgcv)
set.seed(0)
n <- 400
f <- function(la,lo) { ## a test function...</pre>
  sin(lo)*cos(la-.3)
}
## generate with uniform density on sphere...
lo <- runif(n)*2*pi-pi ## longitude</pre>
la <- runif(3*n)*pi-pi/2</pre>
ind <- runif(3*n)<=cos(la)</pre>
la <- la[ind];</pre>
la <- la[1:n]
ff <- f(la,lo)
y \leftarrow ff + rnorm(n)*.2 ## test data
## generate data for plotting truth...
lam <- seq(-pi/2,pi/2,length=30)
lom <- seq(-pi,pi,length=60)</pre>
gr <- expand.grid(la=lam,lo=lom)</pre>
fz <- f(gr$la,gr$lo)</pre>
zm <- matrix(fz,30,60)</pre>
require(mgcv)
dat \leftarrow data.frame(la = la *180/pi, lo = lo *180/pi, y=y)
## fit spline on sphere model...
bp \leftarrow gam(y^s(la,lo,bs="sos",k=60),data=dat)
## pure knot based alternative...
ind <- sample(1:n,100)</pre>
bk <- gam(y^s(la, lo, bs="sos", k=60),
      knots=list(la=dat$la[ind],lo=dat$lo[ind]),data=dat)
b <- bp
```

```
cor(fitted(b),ff)
## plot results and truth...

pd <- data.frame(la=gr$la*180/pi,lo=gr$lo*180/pi)
fv <- matrix(predict(b,pd),30,60)

par(mfrow=c(2,2),mar=c(4,4,1,1))
contour(lom,lam,t(zm))
contour(lom,lam,t(fv))
plot(bp,rug=FALSE)
plot(bp,scheme=1,theta=-30,phi=20,pch=19,cex=.5)</pre>
```

smooth.construct.t2.smooth.spec

Tensor product smoothing constructor

# Description

A special smooth construct method function for creating tensor product smooths from any combination of single penalty marginal smooths, using the construction of Wood, Scheipl and Faraway (2013).

#### Usage

```
## S3 method for class 't2.smooth.spec'
smooth.construct(object, data, knots)
```

# **Arguments**

object a smooth specification object of class t2.smooth.spec, usually generated by a

term like t2(x,z) in a gam model formula

data a list containing just the data (including any by variable) required by this term,

with names corresponding to object\$term (and object\$by). The by variable

is the last element.

knots a list containing any knots supplied for basis setup — in same order and with

same names as data. Can be NULL. See details for further information.

### **Details**

Tensor product smooths are smooths of several variables which allow the degree of smoothing to be different with respect to different variables. They are useful as smooth interaction terms, as they are invariant to linear rescaling of the covariates, which means, for example, that they are insensitive to the measurement units of the different covariates. They are also useful whenever isotropic smoothing is inappropriate. See t2, te, smooth.construct and smooth.terms. The construction employed here produces tensor smooths for which the smoothing penalties are non-overlapping portions of the identity matrix. This makes their estimation by mixed modelling software rather easy.

# Value

```
An object of class "t2.smooth".
```

# Author(s)

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

# References

Wood, S.N., F. Scheipl and J.J. Faraway (2013) Straightforward intermediate rank tensor product smoothing in mixed models. Statistics and Computing 23: 341-360.

# See Also

t2

# **Examples**

```
## see ?t2
```

```
smooth.construct.tensor.smooth.spec
```

Tensor product smoothing constructor

# Description

A special smooth.construct method function for creating tensor product smooths from any combination of single penalty marginal smooths.

# Usage

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

# **Arguments**

object	a smooth specification object of class tensor.smooth.spec, usually generated by a term like $te(x,z)$ in a gam model formula
data	a list containing just the data (including any by variable) required by this term, with names corresponding to object\$term (and object\$by). The by variable is the last element.
knots	a list containing any knots supplied for basis setup — in same order and with same names as data. Can be NULL. See details for further information.

#### **Details**

Tensor product smooths are smooths of several variables which allow the degree of smoothing to be different with respect to different variables. They are useful as smooth interaction terms, as they are invariant to linear rescaling of the covariates, which means, for example, that they are insensitive to the measurement units of the different covariates. They are also useful whenever isotropic smoothing is inappropriate. See te, smooth construct and smooth terms.

#### Value

An object of class "tensor.smooth". See smooth.construct, for the elements that this object will contain.

#### Author(s)

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

#### References

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

#### See Also

cSplineDes

# **Examples**

## see ?gam

smooth.construct.tp.smooth.spec

Penalized thin plate regression splines in GAMs

## **Description**

gam can use isotropic smooths of any number of variables, specified via terms like s(x,z,bs="tp",m=3) (or just s(x,z) as this is the default basis). These terms are based on thin plate regression splines. m specifies the order of the derivatives in the thin plate spline penalty. If m is a vector of length 2 and the second element is zero, then the penalty null space of the smooth is not included in the smooth: this is useful if you need to test whether a smooth could be replaced by a linear term, for example.

Thin plate regression splines are constructed by starting with the basis and penalty for a full thin plate spline and then truncating this basis in an optimal manner, to obtain a low rank smoother. Details are given in Wood (2003). One key advantage of the approach is that it avoids the knot placement problems of conventional regression spline modelling, but it also has the advantage that smooths of lower rank are nested within smooths of higher rank, so that it is legitimate to use conventional hypothesis testing methods to compare models based on pure regression splines. Note

that the basis truncation does not change the meaning of the thin plate spline penalty (it penalizes exactly what it would have penalized for a full thin plate spline).

The t.p.r.s. basis and penalties can become expensive to calculate for large datasets. For this reason the default behaviour is to randomly subsample max.knots unique data locations if there are more than max.knots such, and to use the sub-sample for basis construction. The sampling is always done with the same random seed to ensure repeatability (does not reset R RNG). max.knots is 2000, by default. Both seed and max.knots can be modified using the xt argument to s. Alternatively the user can supply knots from which to construct a basis.

The "ts" smooths are t.p.r.s. with the penalty modified so that the term is shrunk to zero for high enough smoothing parameter, rather than being shrunk towards a function in the penalty null space (see details).

# Usage

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

## **Arguments**

object	a smooth specification object, usually generated by a term $s(,bs="tp",)$ or $s(,bs="ts",)$
data	a list containing just the data (including any by variable) required by this term, with names corresponding to object\$term (and object\$by). The by variable is the last element.
knots	a list containing any knots supplied for basis setup — in same order and with same names as data. Can be NULL

## Details

The default basis dimension for this class is k=M+k.def where M is the null space dimension (dimension of unpenalized function space) and k.def is 8 for dimension 1, 27 for dimension 2 and 100 for higher dimensions. This is essentially arbitrary, and should be checked, but as with all penalized regression smoothers, results are statistically insensitive to the exact choise, provided it is not so small that it forces oversmoothing (the smoother's degrees of freedom are controlled primarily by its smoothing parameter).

The default is to set m (the order of derivative in the thin plate spline penalty) to the smallest value satisfying 2m > d+1 where d if the number of covariates of the term: this yields 'visually smooth' functions. In any case 2m>d must be satisfied.

The constructor is not normally called directly, but is rather used internally by gam. To use for basis setup it is recommended to use smooth.construct2.

For these classes the specification object will contain information on how to handle large datasets in their xt field. The default is to randomly subsample 2000 'knots' from which to produce a tprs basis, if the number of unique predictor variable combinations in excess of 2000. The default can be modified via the xt argument to s. This is supplied as a list with elements max.knots and seed

containing a number to use in place of 2000, and the random number seed to use (either can be missing).

For these bases knots has two uses. Firstly, as mentioned already, for large datasets the calculation of the tp basis can be time-consuming. The user can retain most of the advantages of the t.p.r.s. approach by supplying a reduced set of covariate values from which to obtain the basis - typically the number of covariate values used will be substantially smaller than the number of data, and substantially larger than the basis dimension, k. This approach is the one taken automatically if the number of unique covariate values (combinations) exceeds max.knots. The second possibility is to avoid the eigen-decomposition used to find the t.p.r.s. basis altogether and simply use the basis implied by the chosen knots: this will happen if the number of knots supplied matches the basis dimension, k. For a given basis dimension the second option is faster, but gives poorer results (and the user must be quite careful in choosing knot locations).

The shrinkage version of the smooth, eigen-decomposes the wiggliness penalty matrix, and sets its zero eigenvalues to small multiples of the smallest strictly positive eigenvalue. The penalty is then set to the matrix with eigenvectors corresponding to those of the original penalty, but eigenvalues set to the peturbed versions. This penalty matrix has full rank and shrinks the curve to zero at high enough smoothing parameters.

### Value

An object of class "tprs.smooth" or "ts.smooth". In addition to the usual elements of a smooth class documented under smooth.construct, this object will contain:

shift A record of the shift applied to each covariate in order to center it around zero

and avoid any co-linearity problems that might otehrwise occur in the penalty

null space basis of the term.

Xu A matrix of the unique covariate combinations for this smooth (the basis is con-

structed by first stripping out duplicate locations).

UZ The matrix mapping the t.p.r.s. parameters back to the parameters of a full thin

plate spline.

null.space.dimension

The dimension of the space of functions that have zero wiggliness according to the wiggliness penalty for this term.

## Author(s)

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

#### References

Wood, S.N. (2003) Thin plate regression splines. J.R.Statist.Soc.B 65(1):95-114

```
require(mgcv); n <- 100; set.seed(2) x <- runif(n); y <- x + x^2*.2 + rnorm(n) *.1 ## is smooth significantly different from straight line? summary(gam(y~s(x,m=c(2,0))+x,method="REML")) ## not quite ## is smooth significatly different from zero?
```

smooth.terms 219

```
\label{eq:summary(gam(y^s(x),method="REML")) ## yes!} $$\# see ?gam
```

smooth.terms

Smooth terms in GAM

## **Description**

Smooth terms are specified in a gam formula using s, te, ti and t2 terms. Various smooth classes are available, for different modelling tasks, and users can add smooth classes (see user.defined.smooth). What defines a smooth class is the basis used to represent the smooth function and quadratic penalty (or multiple penalties) used to penalize the basis coefficients in order to control the degree of smoothness. Smooth classes are invoked directly by s terms, or as building blocks for tensor product smoothing via te, ti or t2 terms (only smooth classes with single penalties can be used in tensor products). The smooths built into the mgcv package are all based one way or another on low rank versions of splines. For the full rank versions see Wahba (1990).

Note that smooths can be used rather flexibly in gam models. In particular the linear predictor of the GAM can depend on (a discrete approximation to) any linear functional of a smooth term, using by variables and the 'summation convention' explained in linear.functional.terms.

The single penalty built in smooth classes are summarized as follows

Thin plate regression splines bs="tp". These are low rank isotropic smoothers of any number of covariates. By isotropic is meant that rotation of the covariate co-ordinate system will not change the result of smoothing. By low rank is meant that they have far fewer coefficients than there are data to smooth. They are reduced rank versions of the thin plate splines and use the thin plate spline penalty. They are the default smooth for s terms because there is a defined sense in which they are the optimal smoother of any given basis dimension/rank (Wood, 2003). Thin plate regression splines do not have 'knots' (at least not in any conventional sense): a truncated eigen-decomposition is used to achieve the rank reduction. See tprs for further details.

bs="ts" is as "tp" but with a modification to the smoothing penalty, so that the null space is also penalized slightly and the whole term can therefore be shrunk to zero.

**Duchon splines** bs="ds". These generalize thin plate splines. In particular, for any given number of covariates they allow lower orders of derivative in the penalty than thin plate splines (and hence a smaller null space). See Duchon.spline for further details.

**Cubic regression splines** bs="cr". These have a cubic spline basis defined by a modest sized set of knots spread evenly through the covariate values. They are penalized by the conventional intergrated square second derivative cubic spline penalty. For details see cubic.regression.spline and e.g. Wood (2006a).

bs="cs" specifies a shrinkage version of "cr".

bs="cc" specifies a cyclic cubic regression splines (see cyclic.cubic.spline). i.e. a penalized cubic regression splines whose ends match, up to second derivative.

**Splines on the sphere** bs="sos". These are two dimensional splines on a sphere. Arguments are latitude and longitude, and they are the analogue of thin plate splines for the sphere. Useful for data sampled over a large portion of the globe, when isotropy is appropriate. See Spherical.Spline for details.

220 smooth.terms

**P-splines** bs="ps". These are P-splines as proposed by Eilers and Marx (1996). They combine a B-spline basis, with a discrete penalty on the basis coefficients, and any sane combination of penalty and basis order is allowed. Although this penalty has no exact interpretation in terms of function shape, in the way that the derivative penalties do, P-splines perform almost as well as conventional splines in many standard applications, and can perform better in particular cases where it is advantageous to mix different orders of basis and penalty.

bs="cp" gives a cyclic version of a P-spline (see cyclic.p.spline).

- **Random effects** bs="re". These are parametric terms penalized by a ridge penalty (i.e. the identity matrix). When such a smooth has multiple arguments then it represents the parametric interaction of these arguments, with the coefficients penalized by a ridge penalty. The ridge penalty is equivalent to an assumption that the coefficients are i.i.d. normal random effects. See smooth.construct.re.smooth.spec.
- Markov Random Fields bs="mrf". These are popular when space is split up into discrete contiguous geographic units (districts of a town, for example). In this case a simple smoothing penalty is constructed based on the neighbourhood structure of the geographic units. See mrf for details and an example.
- **Gaussian process smooths** bs="gp". Gaussian process models with a variety of simple correlation functions can be represented as smooths. See gp. smooth for details.
- **Soap film smooths** bs="so" (actually not single penaltied, but bs="sw" and bs="sf" allows splitting into single penalty components for use in tensor product smoothing). These are finite area smoothers designed to smooth within complicated geographical boundaries, where the boundary matters (e.g. you do not want to smooth across boundary features). See soap for details.

Broadly speaking the default penalized thin plate regression splines tend to give the best MSE performance, but they are slower to set up than the other bases. The knot based penalized cubic regression splines (with derivative based penalties) usually come next in MSE performance, with the P-splines doing just a little worse. However the P-splines are useful in non-standard situations.

All the preceding classes (and any user defined smooths with single penalties) may be used as marginal bases for tensor product smooths specified via te, ti or t2 terms. Tensor product smooths are smooth functions of several variables where the basis is built up from tensor products of bases for smooths of fewer (usually one) variable(s) (marginal bases). The multiple penalties for these smooths are produced automatically from the penalties of the marginal smooths. Wood (2006b) and Wood, Scheipl and Faraway (2012), give the general recipe for these constructions.

- **te** te smooths have one penalty per marginal basis, each of which is interpretable in a similar way to the marginal penalty from which it is derived. See Wood (2006b).
- ti ti smooths exclude the basis functions associated with the 'main effects' of the marginal smooths, plus interactions other than the highest order specified. These provide a stable an interpretable way of specifying models with main effects and interactions. For example if we are interested in linear predicto  $f_1(x) + f_2(z) + f_3(x,z)$ , we might use model formula  $y\sim s(x)+s(z)+ti(x,z)$  or  $y\sim ti(x)+ti(z)+te(x,z)$ . A similar construction involving te terms instead will be much less statistically stable.
- t2 t2 uses an alternative tensor product construction that results in more penalties each having a simple non-overlapping structure allowing use with the gamm4 package. It is a natural generalization of the SS-ANOVA construction, but the penalties are a little harder to interpret. See Wood, Scheipl and Faraway (2012/13).

smooth.terms 221

Tensor product smooths often perform better than isotropic smooths when the covariates of a smooth are not naturally on the same scale, so that their relative scaling is arbitrary. For example, if smoothing with repect to time and distance, an isotropic smoother will give very different results if the units are cm and minutes compared to if the units are metres and seconds: a tensor product smooth will give the same answer in both cases (see te for an example of this). Note that te terms are knot based, and the thin plate splines seem to offer no advantage over cubic or P-splines as marginal bases.

Some further specialist smoothers that are not suitable for use in tensor products are also available.

Adaptive smoothers bs="ad" Univariate and bivariate adaptive smooths are available (see adaptive.smooth).

These are appropriate when the degree of smoothing should itself vary with the covariates to be smoothed, and the data contain sufficient information to be able to estimate the appropriate variation. Because this flexibility is achieved by splitting the penalty into several 'basis penalties' these terms are not suitable as components of tensor product smooths, and are not supported by gamm.

**Factor smooth interactions** bs="fs" Smooth factor interactions are often produced using by variables (see gam.models), but a special smoother class (see factor.smooth.interaction) is available for the case in which a smooth is required at each of a large number of factor levels (for example a smooth for each patient in a study), and each smooth should have the same smoothing parameter. The "fs" smoothers are set up to be efficient when used with gamm, and have penalties on each null sapce component (i.e. they are fully 'random effects').

## Author(s)

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

#### References

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

Wahba (1990) Spline Models of Observational Data. SIAM

Wood, S.N. (2003) Thin plate regression splines. J.R.Statist.Soc.B 65(1):95-114

Wood, S.N. (2006a) Generalized Additive Models: an introduction with R, CRC

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

Wood S.N., F. Scheipl and J.J. Faraway (2013) Straightforward intermediate rank tensor product smoothing in mixed models. Statistical Computing. 23(3), 341-360. [online 2012]

# See Also

```
s, te, t2 tprs,Duchon.spline, cubic.regression.spline,p.spline, mrf, soap, Spherical.Spline, adaptive.smooth, user.defined.smooth, smooth.construct.re.smooth.spec, smooth.construct.gp.smooth.spec,
```

# **Examples**

## see examples for gam and gamm

222 smoothCon

smoothCon	Prediction/Construction wrapper functions for GAM smooth terms

# Description

Wrapper functions for construction of and prediction from smooth terms in a GAM. The purpose of the wrappers is to allow user-transparant re-parameterization of smooth terms, in order to allow identifiability constraints to be absorbed into the parameterization of each term, if required. The routine also handles 'by' variables and construction of identifiability constraints automatically, although this behaviour can be over-ridden.

# Usage

# **Arguments**

	object	is a smooth specification object or a smooth object.
	data	A data frame, model frame or list containing the values of the (named) covariates at which the smooth term is to be evaluated. If it's a list then n must be supplied.
	knots	An optional data frame supplying any knot locations to be supplied for basis construction.
	absorb.cons	Set to TRUE in order to have identifiability constraints absorbed into the basis.
	scale.penalty	should the penalty coefficient matrix be scaled to have approximately the same 'size' as the inner product of the terms model matrix with itself? This can improve the performance of $gamm$ fitting.
	n	number of values for each covariate, or if a covariate is a matrix, the number of rows in that matrix: must be supplied explicitly if data is a list.
	dataX	Sometimes the basis should be set up using data in data, but the model matrix should be constructed with another set of data provided in dataX — n is assumed to be the same for both. Facilitates smooth id's.
null.space.penalty		
		Should an extra penalty be added to the smooth which will penalize the components of the smooth in the penalty null space: provides a way of penalizing terms out of the model altogether.
	apply.by	set to FALSE to have basis setup exactly as in default case, but to return add an additional matrix X0 to the return object, containing the model matrix without the by variable, if a by variable is present. Useful for bam discrete method setup.

smoothCon 223

sparse.cons

If 0 then default sum to zero constraints are used. If -1 then sweep and drop sum to zero constraints are used (default with bam). If 1 then one coefficient is set to zero as constraint for sparse smooths. If 2 then sparse coefficient sum to zero constraints are used for sparse smooths. None of these options has an effect if the smooth supplies its own constraint.

diagonal.penalty

If TRUE then the smooth is reparameterized to turn the penalty into an identity matrix, with the final diagonal elements zeroed (corresponding to the penalty nullspace). May result in a matrix diagRP in the returned object for use by PredictMat.

#### **Details**

These wrapper functions exist to allow smooths specified using smooth.construct and Predict.matrix method functions to be re-parameterized so that identifiability constraints are no longer required in fitting. This is done in a user transparent manner, but is typically of no importance in use of GAMs. The routine's also handle by variables and will create default identifiability constraints.

If a user defined smooth constructor handles by variables itself, then its returned smooth object should contain an object by done. If this does not exist then smoothCon will use the default code. Similarly if a user defined Predict.matrix method handles by variables internally then the returned matrix should have a "by.done" attribute.

Default centering constraints, that terms should sum to zero over the covariates, are produced unless the smooth constructor includes a matrix C of constraints. To have no constraints (in which case you had better have a full rank penalty!) the matrix C should have no rows. There is an option to use centering constraint that generate no, or limited infil, if the smoother has a sparse model matrix. smoothCon returns a list of smooths because factor by variables result in multiple copies of a smooth, each multiplied by the dummy variable associated with one factor level. smoothCon modifies the smooth object labels in the presence of by variables, to ensure that they are unique, it also stores the level of a by variable factor associated with a smooth, for later use by PredictMat.

The parameterization used by gam can be controlled via gam.control.

# Value

From smoothCon a list of smooth objects returned by the appropriate smooth.construct method function. If constraints are to be absorbed then the objects will have attributes "qrc" and "nCons". "nCons" is the number of constraints. "qrc" is usually the qr decomposition of the constraint matrix (returned by qr), but if it is a single positive integer it is the index of the coefficient to set to zero, and if it is a negative number then this indicates that the parameters are to sum to zero.

For predictMat a matrix which will map the parameters associated with the smooth to the vector of values of the smooth evaluated at the covariate values given in object.

## Author(s)

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

## References

http://www.maths.bris.ac.uk/~sw15190/

224 sp.vcov

## See Also

```
gam.control, smooth.construct, Predict.matrix
```

## **Examples**

```
## example of using smoothCon and PredictMat to set up a basis
## to use for regression and make predictions using the result
library(MASS) ## load for mcycle data.
## set up a smoother...
sm <- smoothCon(s(times,k=10),data=mcycle,knots=NULL)[[1]]</pre>
## use it to fit a regression spline model...
beta <- coef(lm(mcycle$accel~sm$X-1))</pre>
with(mcycle,plot(times,accel)) ## plot data
times <- seq(0,60,length=200) ## creat prediction times
## Get matrix mapping beta to spline prediction at 'times'
Xp <- PredictMat(sm,data.frame(times=times))</pre>
lines(times, Xp%*%beta) ## add smooth to plot
## Same again but using a penalized regression spline of
## rank 30....
sm <- smoothCon(s(times,k=30),data=mcycle,knots=NULL)[[1]]</pre>
E <- t(mroot(sm$S[[1]])) ## square root penalty</pre>
X <- rbind(sm$X,0.1*E) ## augmented model matrix
y <- c(mcycle$accel,rep(0,nrow(E))) ## augmented data
beta <- coef(lm(y^X-1)) ## fit penalized regression spline
Xp <- PredictMat(sm,data.frame(times=times)) ## prediction matrix</pre>
with(mcycle,plot(times,accel)) ## plot data
lines(times, Xp%*%beta) ## overlay smooth
```

sp.vcov

Extract smoothing parameter estimator covariance matrix from (RE)ML GAM fit

### **Description**

Extracts the estimated covariance matrix for the log smoothing parameter estimates from a (RE)ML estimated gam object, provided the fit was with a method that evaluated the required Hessian.

# Usage

```
sp.vcov(x)
```

## **Arguments**

Χ

a fitted model object of class gam as produced by gam().

### **Details**

Just extracts the inverse of the hessian matrix of the negative (restricted) log likelihood w.r.t the log smoothing parameters, if this has been obtained as part of fitting.

spasm.construct 225

# Value

A matrix corresponding to the estimated covariance matrix of the log smoothing parameter estimators, if this can be extracted, otherwise NULL. If the scale parameter has been (RE)ML estimated (i.e. if the method was "ML" or "REML" and the scale parameter was unknown) then the last row and column relate to the log scale parameter.

# Author(s)

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

## References

Wood, S.N. (2006) 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

```
gam, gam. vcomp
```

## **Examples**

spasm.construct

Experimental sparse smoothers

# **Description**

These are experimental sparse smoothing functions, and should be left well alone!

## Usage

```
spasm.construct(object,data)
spasm.sp(object,sp,w=rep(1,object$nobs),get.trH=TRUE,block=0,centre=FALSE)
spasm.smooth(object,X,residual=FALSE,block=0)
```

226 step.gam

### **Arguments**

object sparse smooth object

data data frame

sp smoothing parameter value

w optional weights

get.trH Should (estimated) trace of sparse smoother matrix be returned

block index of block, 0 for all blocks centre should sparse smooth be centred?

X what to smooth

residual apply residual operation?

#### WARNING

It is not recommended to use these yet

### Author(s)

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

step.gam Alternatives to step.gam

# Description

There is no step.gam in package mgcv. The mgcv default for model selection is to use either prediction error criteria such as GCV, GACV, Mallows' Cp/AIC/UBRE or the likelihood based methods of REML or ML. Since the smoothness estimation part of model selection is done in this way it is logically most consistent to perform the rest of model selection in the same way. i.e. to decide which terms to include or omit by looking at changes in GCV, AIC, REML etc.

To facilitate fully automatic model selection the package implements two smooth modification techniques which can be used to allow smooths to be shrunk to zero as part of smoothness selection.

Shrinkage smoothers are smoothers in which a small multiple of the identity matrix is added to the smoothing penalty, so that strong enough penalization will shrink all the coefficients of the smooth to zero. Such smoothers can effectively be penalized out of the model altogether, as part of smoothing parameter estimation. 2 classes of these shrinkage smoothers are implemented: "cs" and "ts", based on cubic regression spline and thin plate regression spline smoothers (see s)

**Null space penalization** An alternative is to construct an extra penalty for each smooth which penalizes the space of functions of zero wiggliness according to its existing penalties. If all the smoothing parameters for such a term tend to infinity then the term is penalized to zero, and is effectively dropped from the model. The advantage of this approach is that it can be implemented automatically for any smooth. The select argument to gam causes this latter approach to be used. Unpenalized terms (e.g. s(x,fx=TRUE)) remain unpenalized.

REML and ML smoothness selection are equivalent under this approach, and simulation evidence suggests that they tend to perform a little better than prediction error criteria, for model selection.

step.gam 227

## Author(s)

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

### References

Marra, G. and S.N. Wood (2011) Practical variable selection for generalized additive models Computational Statistics and Data Analysis 55,2372-2387

### See Also

```
gam.selection
```

```
## an example of GCV based model selection as
## an alternative to stepwise selection, using
## shrinkage smoothers...
library(mgcv)
set.seed(0);n \leftarrow 400
dat <- gamSim(1,n=n,scale=2)</pre>
dat$x4 <- runif(n, 0, 1)
dat$x5 <- runif(n, 0, 1)
attach(dat)
## Note the increased gamma parameter below to favour
## slightly smoother models...
b < -gam(y \sim s(x0, bs="ts") + s(x1, bs="ts") + s(x2, bs="ts") +
   s(x3,bs="ts")+s(x4,bs="ts")+s(x5,bs="ts"),gamma=1.4)
summary(b)
plot(b,pages=1)
## Same again using REML/ML
b < -gam(y < x0, bs = "ts") + s(x1, bs = "ts") + s(x2, bs = "ts") +
   s(x3,bs="ts")+s(x4,bs="ts")+s(x5,bs="ts"), method="REML")
summary(b)
plot(b,pages=1)
## And once more, but using the null space penalization
b < -gam(y \sim s(x0,bs="cr")+s(x1,bs="cr")+s(x2,bs="cr")+
   s(x3,bs="cr")+s(x4,bs="cr")+s(x5,bs="cr"),
   method="REML",select=TRUE)
summary(b)
plot(b,pages=1)
detach(dat);rm(dat)
```

summary.gam

Summary for a GAM fit

# Description

Takes a fitted gam object produced by gam() and produces various useful summaries from it. (See sink to divert output to a file.)

## Usage

# **Arguments**

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

x a summary.gam object produced by summary.gam().

dispersion A known dispersion parameter. NULL to use estimate or default (e.g. 1 for Pois-

son).

freq By default p-values for parametric terms are calculated using the Bayesian esti-

mated covariance matrix of the parameter estimators. If this is set to TRUE then

the frequentist covariance matrix of the parameters is used instead.

p. type determines how p-values are computed for smooth terms. 0 uses a test statistic

with distribution determined by the un-rounded edf of the term. 1 uses upwardly biased rounding of the edf and -1 uses a version of the test statistic with a null distribution that has to be simulated. 5 is the approximation in Wood (2006). Other options are poor, generate a warning, and are only of research interest.

See details.

digits controls number of digits printed in output.

signif.stars Should significance stars be printed alongside output.

... other arguments.

#### **Details**

Model degrees of freedom are taken as the trace of the influence (or hat) matrix  $\bf A$  for the model fit. Residual degrees of freedom are taken as number of data minus model degrees of freedom. Let  $\bf P_i$  be the matrix giving the parameters of the ith smooth when applied to the data (or pseudodata in the generalized case) and let  $\bf X$  be the design matrix of the model. Then  $tr(\bf XP_i)$  is the edf for the ith term. Clearly this definition causes the edf's to add up properly! An alternative version of EDF is more appropriate for p-value computation, and is based on the trace of  $2\bf A - A\bf A$ .

print.summary.gam tries to print various bits of summary information useful for term selection in a pretty way.

P-values for smooth terms are usually based on a test statistic motivated by an extension of Nychka's (1988) analysis of the frequentist properties of Bayesian confidence intervals for smooths (Marra and Wood, 2012). These have better frequentist performance (in terms of power and distribution under the null) than the alternative strictly frequentist approximation. When the Bayesian intervals have good across the function properties then the p-values have close to the correct null distribution and reasonable power (but there are no optimality results for the power). Full details are in Wood (2013b), although what is computed is actually a slight variant in which the components of the test statistic are weighted by the iterative fitting weights.

Note that for terms with no unpenalized terms (such as Gaussian random effects) the Nychka (1988) requirement for smoothing bias to be substantially less than variance breaks down (see e.g. appendix of Marra and Wood, 2012), and this results in incorrect null distribution for p-values computed using the above approach. In this case it is necessary to use an alternative approach designed for random effects variance components, and this is done. See Wood (2013a) for details: the test is based on a likelihood ratio statistic (with the reference distribution appropriate for the null hypothesis on the boundary of the parameter space).

All p-values are computed without considering uncertainty in the smoothing parameter estimates.

In simulations the p-values have best behaviour under ML smoothness selection, with REML coming second. In general the p-values behave well, but neglecting smoothing parameter uncertainty means that they may be somewhat too low when smoothing parameters are highly uncertain. High uncertainty happens in particular when smoothing parameters are poorly identified, which can occur with nested smooths or highly correlated covariates (high concurvity).

If p. type=5 then the frequentist approximation for p-values of smooth terms described in section 4.8.5 of Wood (2006) is used. The approximation is not as good as the default, and is no longer recommended.

By default the p-values for parametric model terms are also based on Wald tests using the Bayesian covariance matrix for the coefficients. This is appropriate when there are "re" terms present, and is otherwise rather similar to the results using the frequentist covariance matrix (freq=TRUE), since the parametric terms themselves are usually unpenalized. Default P-values for parameteric terms that are penalized using the paraPen argument will not be good. However if such terms represent conventional random effects with full rank penalties, then setting freq=TRUE is appropriate.

# Value

summary.gam produces a list of summary information for a fitted gam object.

See details.

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.

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 ex-

plained, 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! The proportion null deviance explained is probably more appropriate for non-normal errors. Note that r.sq does not include any offset in the one

parameter model.

dev.expl The proportion of the null deviance explained by the model. The null deviance is

computed taking account of any offset, so dev.expl can be substantially lower

than r. sq when an offset is present.

edf array of estimated degrees of freedom for the model terms.

residual.df estimated residual degrees of freedom.

n number of data.

np number of model coefficients (regression coefficients, not smoothing parameters

or other parameters of likelihood).

rank apparent model rank.

method The smoothing selection criterion used.

sp.criterion The minimized value of the smoothness selection criterion. Note that for ML

and REML methods, what is reported is the negative log marginal likelihood or

negative log restricted likelihood.

scale estimated (or given) scale parameter.

family the family used.

formula the original GAM formula.

dispersion the scale parameter.

pTerms.df the degrees of freedom associated with each parametric term (excluding the con-

stant).

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 parameterss. table significance table for smooths

p. Terms significance table for parametric model terms

### WARNING

The p-values are approximate and neglect smoothing parameter uncertainty. They are likely to be somewhat too low when smoothing parameter estimates are highly uncertain: do read the details section. If the exact values matter, read Wood (2013a or b).

P-values for terms penalized via 'paraPen' are unlikely to be correct.

## Author(s)

Simon N. Wood <simon.wood@r-project.org> with substantial improvements by Henric Nilsson.

### References

Marra, G and S.N. Wood (2012) Coverage Properties of Confidence Intervals for Generalized Additive Model Components. Scandinavian Journal of Statistics, 39(1), 53-74.

Nychka (1988) Bayesian Confidence Intervals for Smoothing Splines. Journal of the American Statistical Association 83:1134-1143.

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

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

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

## See Also

```
gam, predict.gam, gam.check, anova.gam, gam.vcomp, sp.vcov
```

```
library(mgcv)
set.seed(0)
dat <- gamSim(1,n=200,scale=2) ## simulate data
b \leftarrow gam(y \sim s(x0) + s(x1) + s(x2) + s(x3), data = dat)
plot(b,pages=1)
summary(b)
## now check the p-values by using a pure regression spline.....
b.d <- round(summary(b)$edf)+1 ## get edf per smooth</pre>
b.d <- pmax(b.d,3) # can't have basis dimension less than 3!</pre>
bc < -gam(y \sim s(x0, k=b.d[1], fx=TRUE) + s(x1, k=b.d[2], fx=TRUE) +
         s(x2,k=b.d[3],fx=TRUE)+s(x3,k=b.d[4],fx=TRUE),data=dat)
plot(bc,pages=1)
summary(bc)
## Example where some p-values are less reliable...
dat <- gamSim(6,n=200,scale=2)</pre>
b \leftarrow gam(y^s(x0,m=1)+s(x1)+s(x2)+s(x3)+s(fac,bs="re"),data=dat)
```

```
## Here s(x0,m=1) can be penalized to zero, so p-value approximation
## cruder than usual...
summary(b)
## p-value check - increase k to make this useful!
k<-20;n<-200;p<-rep(NA,k)
for (i in 1:k)
{ b<-gam(y~te(x,z),data=data.frame(y=rnorm(n),x=runif(n),z=runif(n)),</pre>
         method="ML")
 p[i] < -summary(b) $s.p[1]
plot(((1:k)-0.5)/k, sort(p))
abline(0,1,col=2)
ks.test(p, "punif") ## how close to uniform are the p-values?
## A Gamma example, by modify `gamSim' output...
dat <- gamSim(1,n=400,dist="normal",scale=1)</pre>
dat$f <- dat$f/4 ## true linear predictor
Ey <- exp(dat$f);scale <- .5 ## mean and GLM scale parameter
## Note that `shape' and `scale' in `rgamma' are almost
## opposite terminology to that used with GLM/GAM...
dat$y <- rgamma(Ey*0,shape=1/scale,scale=Ey*scale)</pre>
bg <- gam(y^{-} s(x0) + s(x1) + s(x2) + s(x3), family=Gamma(link=log),
          data=dat,method="REML")
summary(bg)
```

Define alternative tensor product smooths in GAM formulae

# Description

t2

Alternative to te for defining tensor product smooths in a gam formula. Results in a construction in which the penalties are non-overlapping multiples of identity matrices (with some rows and columns zeroed). The construction, which is due to Fabian Scheipl (mgcv implementation, 2010), is analogous to Smoothing Spline ANOVA (Gu, 2002), but using low rank penalized regression spline marginals. The main advantage of this construction is that it is useable with gamm4 from package gamm4.

### **Usage**

```
t2(..., k=NA,bs="cr",m=NA,d=NA,by=NA,xt=NULL,
id=NULL,sp=NULL,full=FALSE,ord=NULL)
```

# Arguments

a list of variables that are the covariates that this smooth is a function of.

k

the dimension(s) of the bases used to represent the smooth term. If not supplied then set to 5<sup>d</sup>. If supplied as a single number then this basis dimension is used for each basis. If supplied as an array then the elements are the dimensions of the component (marginal) bases of the tensor product. See <a href="https://creativecommons.org/representation-new-marginal">https://creativecommons.org/representation-new-marginal</a>) bases of the tensor product. See <a href="https://creativecommons.org/representation-new-marginal-new-

bs

array (or single character string) specifying the type for each marginal basis. "cr" for cubic regression spline; "cs" for cubic regression spline with shrinkage; "cc" for periodic/cyclic cubic regression spline; "tp" for thin plate regression spline; "ts" for t.p.r.s. with extra shrinkage. See smooth.terms for details and full list. User defined bases can also be used here (see smooth.construct for an example). If only one basis code is given then this is used for all bases.

m

The order of the spline and its penalty (for smooth classes that use this) for each term. If a single number is given then it is used for all terms. A vector can be used to supply a different m for each margin. For marginals that take vector m (e.g. p.spline and Duchon.spline), then a list can be supplied, with a vector element for each margin. NA autoinitializes. m is ignored by some bases (e.g. "cr").

d

array of marginal basis dimensions. For example if you want a smooth for 3 covariates made up of a tensor product of a 2 dimensional t.p.r.s. basis and a 1-dimensional basis, then set d=c(2,1). Incompatibilities between built in basis types and dimension will be resolved by resetting the basis type.

by

a numeric or factor variable of the same dimension as each covariate. In the numeric vector case the elements multiply the smooth evaluated at the corresponding covariate values (a 'varying coefficient model' results). In the factor case causes a replicate of the smooth to be produced for each factor level. See gam.models for further details. May also be a matrix if covariates are matrices: in this case implements linear functional of a smooth (see gam.models and linear.functional.terms for details).

хt

Either a single object, providing any extra information to be passed to each marginal basis constructor, or a list of such objects, one for each marginal basis.

id

A label or integer identifying this term in order to link its smoothing parameters to others of the same type. If two or more smooth terms have the same id then they will have the same smoothing parameters, and, by default, the same bases (first occurance defines basis type, but data from all terms used in basis construction).

sp

any supplied smoothing parameters for this term. Must be an array of the same length as the number of penalties for this smooth. Positive or zero elements are taken as fixed smoothing parameters. Negative elements signal autoinitialization. Over-rides values supplied in sp argument to gam. Ignored by

full

If TRUE then there is a separate penalty for each combination of null space column and range space. This gives strict invariance. If FALSE each combination of null space and range space generates one penalty, but the coulmns of each null space basis are treated as one group. The latter is more parsimonious, but does mean that invariance is only achieved by an arbitrary rescaling of null space basis vectors.

ord

an array giving the orders of terms to retain. Here order means number of marginal range spaces used in the construction of the component. NULL to retain everything.

### **Details**

Smooths of several covariates can be constructed from tensor products of the bases used to represent smooths of one (or sometimes more) of the covariates. To do this 'marginal' bases are produced with associated model matrices and penalty matrices. These are reparameterized so that the penalty is zero everywhere, except for some elements on the leading diagonal, which all have the same nonzero value. This reparameterization results in an unpenalized and a penalized subset of parameters, for each marginal basis (see e.g. appendix of Wood, 2004, for details).

The re-parameterized marginal bases are then combined to produce a basis for a single function of all the covariates (dimension given by the product of the dimensions of the marginal bases). In this set up there are multiple penalty matrices — all zero, but for a mixture of a constant and zeros on the leading diagonal. No two penalties have a non-zero entry in the same place.

Essentially the basis for the tensor product can be thought of as being constructed from a set of products of the penalized (range) or unpenalized (null) space bases of the marginal smooths (see Gu, 2002, section 2.4). To construct one of the set, choose either the null space or the range space from each marginal, and from these bases construct a product basis. The result is subject to a ridge penalty (unless it happens to be a product entirely of marginal null spaces). The whole basis for the smooth is constructed from all the different product bases that can be constructed in this way. The separately penalized components of the smooth basis each have an interpretation in terms of the ANOVA - decomposition of the term. See pen. edf for some further information.

Note that there are two ways to construct the product. When full=FALSE then the null space bases are treated as a whole in each product, but when full=TRUE each null space column is treated as a separate null space. The latter results in more penalties, but is the strict analog of the SS-ANOVA approach.

Tensor product smooths are especially useful for representing functions of covariates measured in different units, although they are typically not quite as nicely behaved as t.p.r.s. smooths for well scaled covariates.

Note also that GAMs constructed from lower rank tensor product smooths are nested within GAMs constructed from higher rank tensor product smooths if the same marginal bases are used in both cases (the marginal smooths themselves are just special cases of tensor product smooths.)

Note that tensor product smooths should not be centred (have identifiability constraints imposed) if any marginals would not need centering. The constructor for tensor product smooths ensures that this happens.

The function does not evaluate the variable arguments.

#### Value

A class t2.smooth.spec object defining a tensor product smooth to be turned into a basis and penalties by the smooth.construct.tensor.smooth.spec function.

The returned object contains the following items:

A list of smooth. spec objects of the type returned by s, defining the basis from margin which the tensor product smooth is constructed.

term	An array of text strings giving the names of the covariates that the term is a function of.
by	is the name of any by variable as text ("NA" for none).
fx	logical array with element for each penalty of the term (tensor product smooths have multiple penalties). TRUE if the penalty is to be ignored, FALSE, otherwise.
label	A suitable text label for this smooth term.
dim	The dimension of the smoother - i.e. the number of covariates that it is a function of.
mp	TRUE is multiple penalties are to be used (default).
np	TRUE to re-parameterize 1-D marginal smooths in terms of function values (defualt).
id	the id argument supplied to te.
sp	the sp argument supplied to te.

# Author(s)

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

## References

Wood S.N., F. Scheipl and J.J. Faraway (2013, online Feb 2012) Straightforward intermediate rank tensor product smoothing in mixed models. Statistical Computing. 23(3):341-360

Gu, C. (2002) Smoothing Spline ANOVA, Springer.

Alternative approaches to functional ANOVA decompositions, \*not\* implemented by t2 terms, are discussed in:

Belitz and Lang (2008) Simultaneous selection of variables and smoothing parameters in structured additive regression models. Computational Statistics & Data Analysis, 53(1):61-81

Lee, D-J and M. Durban (2011) P-spline ANOVA type interaction models for spatio-temporal smoothing. Statistical Modelling, 11:49-69

Wood, S.N. (2006) Low-Rank Scale-Invariant Tensor Product Smooths for Generalized Additive Mixed Models. Biometrics 62(4): 1025-1036.

# See Also

```
te s,gam,gamm,
```

```
# following shows how tensor product deals nicely with # badly scaled covariates (range of x 5% of range of z ) require(mgcv) test1<-function(x,z,sx=0.3,sz=0.4) { x<-x*20 (pi**sx*sz)*(1.2*exp(-(x-0.2)^2/sx^2-(z-0.3)^2/sz^2)+ 0.8*exp(-(x-0.7)^2/sx^2-(z-0.8)^2/sz^2))
```

```
n<-500
old.par<-par(mfrow=c(2,2))
x<-runif(n)/20;z<-runif(n);</pre>
xs < -seq(0,1,length=30)/20; zs < -seq(0,1,length=30)
pr<-data.frame(x=rep(xs,30),z=rep(zs,rep(30,30)))</pre>
truth<-matrix(test1(pr$x,pr$z),30,30)</pre>
f \leftarrow test1(x,z)
y < - f + rnorm(n) * 0.2
b1 < -gam(y \sim s(x,z))
persp(xs,zs,truth);title("truth")
vis.gam(b1);title("t.p.r.s")
b2 < -gam(y \sim t2(x,z))
vis.gam(b2);title("tensor product")
b3 < -gam(y \sim t2(x,z,bs=c("tp","tp")))
vis.gam(b3);title("tensor product")
par(old.par)
test2<-function(u,v,w,sv=0.3,sw=0.4)
{((pi**sv*sw)*(1.2*exp(-(v-0.2)^2/sv^2-(w-0.3)^2/sw^2)+
  0.8*exp(-(v-0.7)^2/sv^2-(w-0.8)^2/sw^2)))*(u-0.5)^2*20
}
n <- 500
v <- runif(n);w<-runif(n);u<-runif(n)</pre>
f \leftarrow test2(u,v,w)
y \leftarrow f + rnorm(n)*0.2
## tensor product of 2D Duchon spline and 1D cr spline
m <- list(c(1,.5),0)
b \leftarrow gam(y^t2(v,w,u,k=c(30,5),d=c(2,1),b=c("ds","cr"),m=m))
## look at the edf per penalty. "rr" denotes interaction term
## (range space range space). "rn" is interaction of null space
## for u with range space for v,w...
pen.edf(b)
## plot results...
op <- par(mfrow=c(2,2))</pre>
vis.gam(b,cond=list(u=0),color="heat",zlim=c(-0.2,3.5))
vis.gam(b,cond=list(u=.33),color="heat",zlim=c(-0.2,3.5))
\label{list} vis.gam(b,cond=list(u=.67),color="heat",zlim=c(-0.2,3.5))
vis.gam(b,cond=list(u=1),color="heat",zlim=c(-0.2,3.5))
par(op)
b \leftarrow gam(y^t2(v,w,u,k=c(25,5),d=c(2,1),b=c("tp","cr"),full=TRUE),
         method="ML")
## more penalties now. numbers in labels like "r1" indicate which
## basis function of a null space is involved in the term.
pen.edf(b)
```

te

Define tensor product smooths or tensor product interactions in GAM formulae

# Description

Functions used for the definition of tensor product smooths and interactions within gam model formulae. te produces a full tensor product smooth, while ti produces a tensor product interaction, appropriate when the main effects (and any lower interactions) are also present.

The functions do not evaluate the smooth - they exists purely to help set up a model using tensor product based smooths. Designed to construct tensor products from any marginal smooths with a basis-penalty representation (with the restriction that each marginal smooth must have only one penalty).

# Usage

### **Arguments**

... a list of variables that are the covariates that this smooth is a function of.

k the dimension(s) of the bases used to represent the smooth term. If not supplied then set to 5<sup>d</sup>. If supplied as a single number then this basis dimension is used for each basis. If supplied as an array then the elements are the dimensions of

the component (marginal) bases of the tensor product. See choose.k for further

information.

array (or single character string) specifying the type for each marginal basis. "cr" for cubic regression spline; "cs" for cubic regression spline with shrinkage; "cc" for periodic/cyclic cubic regression spline; "tp" for thin plate regression spline; "ts" for t.p.r.s. with extra shrinkage. See smooth.terms for details and full list. User defined bases can also be used here (see smooth.construct for an example). If only one basis code is given then this is used for all bases.

The order of the spline and its penalty (for smooth classes that use this) for each term. If a single number is given then it is used for all terms. A vector can be used to supply a different m for each margin. For marginals that take vector m (e.g. p.spline and Duchon.spline), then a list can be supplied, with a vector element for each margin. NA autoinitializes. m is ignored by some bases (e.g.

"cr").

array of marginal basis dimensions. For example if you want a smooth for 3 covariates made up of a tensor product of a 2 dimensional t.p.r.s. basis and a 1-dimensional basis, then set d=c(2,1). Incompatibilities between built in basis types and dimension will be resolved by resetting the basis type.

bs

m

d

by a numeric or factor variable of the same dimension as each covariate. In the numeric vector case the elements multiply the smooth evaluated at the corresponding covariate values (a 'varying coefficient model' results). In the factor case causes a replicate of the smooth to be produced for each factor level. See gam.models for further details. May also be a matrix if covariates are matrices: in this case implements linear functional of a smooth (see gam.models and linear.functional.terms for details). fx indicates whether the term is a fixed d.f. regression spline (TRUE) or a penalized regression spline (FALSE). TRUE to use multiple penalties for the smooth. FALSE to use only a single penalty: mp single penalties are not recommended and are deprecated - they tend to allow only rather wiggly models. TRUE to use the 'normal parameterization' for a tensor product smooth. This np represents any 1-d marginal smooths via parameters that are function values at 'knots', spread evenly through the data. The parameterization makes the penalties easily interpretable, however it can reduce numerical stability in some cases. Either a single object, providing any extra information to be passed to each хt marginal basis constructor, or a list of such objects, one for each marginal basis. id A label or integer identifying this term in order to link its smoothing parameters to others of the same type. If two or more smooth terms have the same id then they will have the same smoothing paramsters, and, by default, the same bases (first occurance defines basis type, but data from all terms used in basis construction). any supplied smoothing parameters for this term. Must be an array of the sp same length as the number of penalties for this smooth. Positive or zero elements are taken as fixed smoothing parameters. Negative elements signal autoinitialization. Over-rides values supplied in sp argument to gam. Ignored by gamm. For ti smooths you can specify which marginals should have centering conmc straints applied, by supplying 0/1 or FALSE/TRUE values for each marginal in this vector. By default all marginals are constrained, which is what is appropriate for, e.g., functional ANOVA models. Note that 'ti' only applies constraints to the marginals, so if you turn off all marginal constraints the term will have no identifiability constraints. Only use this if you really understand how marginal constraints work.

# Details

Smooths of several covariates can be constructed from tensor products of the bases used to represent smooths of one (or sometimes more) of the covariates. To do this 'marginal' bases are produced with associated model matrices and penalty matrices, and these are then combined in the manner described in tensor.prod.model.matrix and tensor.prod.penalties, to produce a single model matrix for the smooth, but multiple penalties (one for each marginal basis). The basis dimension of the whole smooth is the product of the basis dimensions of the marginal smooths.

An option for operating with a single penalty (The Kronecker product of the marginal penalties) is provided, but it is rarely of practical use, and is deprecated: the penalty is typically so rank deficient that even the smoothest resulting model will have rather high estimated degrees of freedom.

Tensor product smooths are especially useful for representing functions of covariates measured in different units, although they are typically not quite as nicely behaved as t.p.r.s. smooths for well scaled covariates.

It is sometimes useful to investigate smooth models with a main-effects + interactions structure, for example

$$f_1(x) + f_2(z) + f_3(x,z)$$

This functional ANOVA decomposition is supported by ti terms, which produce tensor product interactions from which the main effects have been excluded, under the assumption that they will be included separately. For example the  $\sim$  ti(x) + ti(z) + ti(x,z) would produce the above main effects + interaction structure. This is much better than attempting the same thing with sor te terms representing the interactions (although mgcv does not forbid it). Technically ti terms are very simple: they simply construct tensor product bases from marginal smooths to which identifiability constraints (usually sum-to-zero) have already been applied: correct nesting is then automatic (as with all interactions in a GLM framework).

The 'normal parameterization' (np=TRUE) re-parameterizes the marginal smooths of a tensor product smooth so that the parameters are function values at a set of points spread evenly through the range of values of the covariate of the smooth. This means that the penalty of the tensor product associated with any particular covariate direction can be interpreted as the penalty of the appropriate marginal smooth applied in that direction and averaged over the smooth. Currently this is only done for marginals of a single variable. This parameterization can reduce numerical stability when used with marginal smooths other than "cc", "cr" and "cs": if this causes problems, set np=FALSE.

Note that tensor product smooths should not be centred (have identifiability constraints imposed) if any marginals would not need centering. The constructor for tensor product smooths ensures that this happens.

The function does not evaluate the variable arguments.

## Value

A class tensor.smooth.spec object defining a tensor product smooth to be turned into a basis and penalties by the smooth.construct.tensor.smooth.spec function.

The returned object contains the following items:

margin	A list of smooth. spec objects of the type returned by s, defining the basis from which the tensor product smooth is constructed.
term	An array of text strings giving the names of the covariates that the term is a function of.
by	is the name of any by variable as text ("NA" for none).
fx	logical array with element for each penalty of the term (tensor product smooths have multiple penalties). TRUE if the penalty is to be ignored, FALSE, otherwise.
label	A suitable text label for this smooth term.
dim	The dimension of the smoother - i.e. the number of covariates that it is a function of.
mp	TRUE is multiple penalties are to be used (default).
np	TRUE to re-parameterize 1-D marginal smooths in terms of function values (defualt).

```
id the id argument supplied to te.

sp the sp argument supplied to te.

inter TRUE if the term was generated by ti, FALSE otherwise.

mc the argument mc supplied to ti.
```

### Author(s)

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

### References

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

```
http://www.maths.bris.ac.uk/~sw15190/
```

### See Also

```
s,gam,gamm, smooth.construct.tensor.smooth.spec
```

```
# following shows how tensor pruduct deals nicely with
\# badly scaled covariates (range of x 5% of range of z )
require(mgcv)
test1 <- function(x,z,sx=0.3,sz=0.4) {
  x <- x*20
  (pi**sx*sz)*(1.2*exp(-(x-0.2)^2/sx^2-(z-0.3)^2/sz^2)+
  0.8*exp(-(x-0.7)^2/sx^2-(z-0.8)^2/sz^2))
}
n <- 500
old.par <- par(mfrow=c(2,2))
x \leftarrow runif(n)/20; z \leftarrow runif(n);
xs <- seq(0,1,length=30)/20; zs <- seq(0,1,length=30)
pr \leftarrow data.frame(x=rep(xs,30),z=rep(zs,rep(30,30)))
truth <- matrix(test1(pr$x,pr$z),30,30)</pre>
f \leftarrow test1(x,z)
y \leftarrow f + rnorm(n)*0.2
b1 <- gam(y \sim s(x,z))
persp(xs,zs,truth);title("truth")
vis.gam(b1);title("t.p.r.s")
b2 <- gam(y\sim te(x,z))
vis.gam(b2);title("tensor product")
b3 <- gam(y \sim ti(x) + ti(z) + ti(x,z))
vis.gam(b3);title("tensor anova")
## now illustrate partial ANOVA decomp...
vis.gam(b3);title("full anova")
b4 <- gam(y^- ti(x) + ti(x,z,mc=c(0,1))) ## note z constrained!
vis.gam(b4);title("partial anova")
plot(b4)
```

```
par(old.par)
## now with a multivariate marginal....
test2<-function(u,v,w,sv=0.3,sw=0.4)
{((pi**sv*sw)*(1.2*exp(-(v-0.2)^2/sv^2-(w-0.3)^2/sw^2)+
 0.8*exp(-(v-0.7)^2/sv^2-(w-0.8)^2/sw^2)))*(u-0.5)^2*20
}
n <- 500
v <- runif(n);w<-runif(n);u<-runif(n)</pre>
f <- test2(u,v,w)
y < -f + rnorm(n)*0.2
# tensor product of 2D Duchon spline and 1D cr spline
m \leftarrow list(c(1,.5), rep(0,0)) ## example of list form of m
b \leftarrow gam(y^te(v,w,u,k=c(30,5),d=c(2,1),b=c("ds","cr"),m=m))
op <- par(mfrow=c(2,2))
vis.gam(b,cond=list(u=0),color="heat",zlim=c(-0.2,3.5))
vis.gam(b,cond=list(u=.33),color="heat",zlim=c(-0.2,3.5))
vis.gam(b,cond=list(u=.67),color="heat",zlim=c(-0.2,3.5))
vis.gam(b,cond=list(u=1),color="heat",zlim=c(-0.2,3.5))
par(op)
```

tensor.prod.model.matrix

Utility functions for constructing tensor product smooths

# Description

Produce model matrices or penalty matrices for a tensor product smooth from the model matrices or penalty matrices for the marginal bases of the smooth.

## **Usage**

```
tensor.prod.model.matrix(X)
tensor.prod.penalties(S)
```

# **Arguments**

- X a list of model matrices for the marginal bases of a smooth
- S a list of penalties for the marginal bases of a smooth.

### **Details**

If X[[1]], X[[2]] ... X[[m]] are the model matrices of the marginal bases of a tensor product smooth then the ith row of the model matrix for the whole tensor product smooth is given by X[[1]][i,]%x%X[[2]][i,]%x% ... X[[m]][i,], where %x% is the Kronecker product. Of course the routine operates column-wise, not row-wise!

242 trichol

If S[[1]], S[[2]] ... S[[m]] are the penalty matrices for the marginal bases, and I[[1]], I[[2]] ... I[[m]] are corresponding identity matrices, each of the same dimension as its corresponding penalty, then the tensor product smooth has m associate penalties of the form:

```
S[[1]]%x%I[[2]]%x% ... I[[m]],
I[[1]]%x%S[[2]]%x% ... I[[m]]
...
I[[1]]%x%I[[2]]%x% ... S[[m]].
```

Of course it's important that the model matrices and penalty matrices are presented in the same order when constructing tensor product smooths.

### Value

Either a single model matrix for a tensor product smooth, or a list of penalty terms for a tensor product smooth.

# Author(s)

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

## References

Wood, S.N. (2006) Low rank scale invariant tensor product smooths for Generalized Additive Mixed Models. Biometrics 62(4):1025-1036

# See Also

```
te, smooth.construct.tensor.smooth.spec
```

## **Examples**

```
require(mgcv)
X <- list(matrix(1:4,2,2),matrix(5:10,2,3))
tensor.prod.model.matrix(X)

S<-list(matrix(c(2,1,1,2),2,2),matrix(c(2,1,0,1,2,1,0,1,2),3,3))
tensor.prod.penalties(S)</pre>
```

trichol

Choleski decomposition of a tri-diagonal matrix

# **Description**

Computes Choleski decomposition of a (symmetric positive definite) tri-diagonal matrix stored as a leading diagonal and sub/super diagonal.

trichol 243

## Usage

```
trichol(ld,sd)
```

# **Arguments**

ld	leading diagonal of matrix
sd	sub-super diagonal of matrix

### **Details**

Calls dpttrf from LAPACK. The point of this is that it has O(n) computational cost, rather than the  $O(n^3)$  required by dense matrix methods.

## Value

A list with elements 1d and sd. 1d is the leading diagonal and sd is the super diagonal of bidiagonal matrix  $\mathbf{B}$  where  $\mathbf{B}^T\mathbf{B} = \mathbf{T}$  and  $\mathbf{T}$  is the original tridiagonal matrix.

# Author(s)

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

#### References

Anderson, E., Bai, Z., Bischof, C., Blackford, S., Dongarra, J., Du Croz, J., Greenbaum, A., Hammarling, S., McKenney, A. and Sorensen, D., 1999. LAPACK Users' guide (Vol. 9). Siam.

# See Also

bandchol

```
require(mgcv)
## simulate some diagonals...
set.seed(19); k <- 7
Id <- runif(k)+1
sd <- runif(k-1) -.5
## get diagonals of chol factor...
trichol(ld,sd)
## compare to dense matrix result...
A <- diag(ld); for (i in 1:(k-1)) A[i,i+1] <- A[i+1,i] <- sd[i]
R <- chol(A)
diag(R); diag(R[,-1])</pre>
```

244 Tweedie

Tweedie	GAM Tweedie families	

## **Description**

Tweedie families, designed for use with gam from the mgcv library. Restricted to variance function powers between 1 and 2. A useful alternative to quasi when a full likelihood is desirable. Tweedie is for use with fixed p. tw is for use when p is to be estimated during fitting. For fixed p between 1 and 2 the Tweedie is an exponential family distribution with variance given by the mean to the power p.

tw is only useable with gam, not bam or gamm. Tweedie works with all three.

## Usage

```
Tweedie(p=1, link = power(0))
tw(theta = NULL, link = "log",a=1.01,b=1.99)
```

## **Arguments**

р	the variance of an observation is proportional to its mean to the power p. p must be greater than 1 and less than or equal to 2. 1 would be Poisson, 2 is gamma.
link	The link function: one of "log", "identity", "inverse", "sqrt", or a power link (Tweedie only).
theta	Related to the Tweedie power parameter by $p=\exp(a+b\exp(\theta))/(1+\exp(\theta))$ . If this is supplied as a positive value then it is taken as the fixed value for p. If it is a negative values then its absolute value is taken as the initial value for p.
а	lower limit on p for optimization.
b	upper limit on p for optimization.

# Details

A Tweedie random variable with 1<p<2 is a sum of N gamma random variables where N has a Poisson distribution. The p=1 case is a generalization of a Poisson distribution and is a discrete distribution supported on integer multiples of the scale parameter. For 1<p<2 the distribution is supported on the positive reals with a point mass at zero. p=2 is a gamma distribution. As p gets very close to 1 the continuous distribution begins to converge on the discretely supported limit at p=1, and is therefore highly multimodal. See 1dTweedie for more on this behaviour.

Tweedie is based partly on the poisson family, and partly on tweedie from the statmod package. It includes extra components to work with all mgcv GAM fitting methods as well as an aic function.

The Tweedie density involves a normalizing constant with no closed form, so this is evaluated using the series evaluation method of Dunn and Smyth (2005), with extensions to also compute the derivatives w.r.t. p and the scale parameter. Without restricting p to (1,2) the calculation of Tweedie densities is more difficult, and there does not currently seem to be an implementation which offers any benefit over quasi. If you need this case then the tweedie package is the place to start.

Tweedie 245

### Value

For Tweedie, an object inheriting from class family, with additional elements

dvar the function giving the first derivative of the variance function w.r.t. mu.

d2var the function giving the second derivative of the variance function w.r.t. mu.

A function returning a 3 element array: the saturated log likelihood followed by

its first 2 derivatives w.r.t. the scale parameter.

For tw, an object of class extended. family.

## Author(s)

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

### References

Dunn, P.K. and G.K. Smyth (2005) Series evaluation of Tweedie exponential dispersion model densities. Statistics and Computing 15:267-280

Tweedie, M. C. K. (1984). An index which distinguishes between some important exponential families. Statistics: Applications and New Directions. Proceedings of the Indian Statistical Institute Golden Jubilee International Conference (Eds. J. K. Ghosh and J. Roy), pp. 579-604. Calcutta: Indian Statistical Institute.

Wood, S.N., N. Pya and B. Saefken (2015), Smoothing parameter and model selection for general smooth models. http://arxiv.org/abs/1511.03864

# See Also

ldTweedie, rTweedie

```
library(mgcv)
set.seed(3)
n<-400
## Simulate data...
dat <- gamSim(1,n=n,dist="poisson",scale=.2)</pre>
dat$y <- rTweedie(exp(dat$f),p=1.3,phi=.5) ## Tweedie response</pre>
## Fit a fixed p Tweedie, with wrong link ...
b \leftarrow gam(y^s(x0)+s(x1)+s(x2)+s(x3),family=Tweedie(1.25,power(.1)),
         data=dat)
plot(b,pages=1)
print(b)
## Same by approximate REML...
b1 <- gam(y \sim s(x0) + s(x1) + s(x2) + s(x3), family = Tweedie(1.25, power(.1)),
          data=dat, method="REML")
plot(b1,pages=1)
print(b1)
```

246 uniquecombs

uniquecombs

find the unique rows in a matrix

# **Description**

This routine returns a matrix or data frame containing all the unique rows of the matrix or data frame supplied as its argument. That is, all the duplicate rows are stripped out. Note that the ordering of the rows on exit is not the same as on entry. It also returns an index attribute for relating the result back to the original matrix.

## Usage

```
uniquecombs(x)
```

# **Arguments**

Х

is an R matrix (numeric), or data frame.

## **Details**

Models with more parameters than unique combinations of covariates are not identifiable. This routine provides a means of evaluating the number of unique combinations of coavariates in a model. The routine calls compiled C code, and is based on sorting, with consequent  $O(n\log(n))$  cost. In principle a hash table based solution should be only O(n).

unique and duplicated, can sometimes be used in place of this, if the full index is not needed. Relative performance is variable.

If x is not a matrix or data frame on entry then an attmept is made to coerce it to a data frame.

# Value

A matrix or data frame consisting of the unique rows of x (in arbitrary order).

The matrix or data frame has an "index" attribute. index[i] gives the row of the returned matrix that contains row i of the original matrix.

## Author(s)

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

vcov.gam 247

## See Also

```
unique, duplicated.
```

## **Examples**

vcov.gam

Extract parameter (estimator) covariance matrix from GAM fit

# **Description**

Extracts the Bayesian posterior covariance matrix of the parameters or frequentist covariance matrix of the parameter estimators from a fitted gam object.

# Usage

```
## S3 method for class 'gam'
vcov(object, freq = FALSE, dispersion = NULL,unconditional=FALSE, ...)
```

# **Arguments**

object fitted model object of class gam as produced by gam().

freq TRUE to return the frequentist covariance matrix of the parameter estimators,

FALSE to return the Bayesian posterior covariance matrix of the parameters.

dispersion a value for the dispersion parameter: not normally used.

unconditional if TRUE (and freq==FALSE) then the Bayesian smoothing parameter uncertainty

corrected covariance matrix is returned, if available.

... other arguments, currently ignored.

# **Details**

Basically, just extracts object\$Ve or object\$Vp from a gamObject.

248 vis.gam

# Value

A matrix corresponding to the estimated frequentist covariance matrix of the model parameter estimators/coefficients, or the estimated posterior covariance matrix of the parameters, depending on the argument freq.

## Author(s)

Henric Nilsson. Maintained by Simon N. Wood <simon.wood@r-project.org>

## References

Wood, S.N. (2006) 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

gam

# **Examples**

```
require(mgcv)
n <- 100
x <- runif(n)
y <- sin(x*2*pi) + rnorm(n)*.2
mod <- gam(y~s(x,bs="cc",k=10),knots=list(x=seq(0,1,length=10)))
diag(vcov(mod))</pre>
```

vis.gam

Visualization of GAM objects

## **Description**

Produces perspective or contour plot views of gam model predictions, fixing all but the values in view to the values supplied in cond.

# Usage

vis.gam 249

# Arguments

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. Note that variables coerced to factors in the model formula won't work as view variables, and vis.gam can not detect that this has happened when setting defaults.  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.
ables omitted from this list will have the closest observed value to the median for continuous variables, or the most commonly occurring 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.
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. In particular ticktype="detailed" will add proper axes labelling to the plots.

# **Details**

The x and y limits are determined by the ranges of the terms named in view. If se<=0 then a single (height colour coded, by default) surface is produced, otherwise three (by default see-through) meshes are produced at mean and +/- se standard errors. Parts of the x-y plane too far from data can be excluded by setting too.far

250 vis.gam

All options to the underlying graphics functions can be reset by passing them as extra arguments . . . : such supplied values will always over-ride the default values used by vis.gam.

### Value

Simply produces a plot.

## **WARNINGS**

The routine can not detect that a variable has been coerced to factor within a model formula, and will therefore fail if such a variable is used as a view variable. When setting default view variables it can not detect this situation either, which can cause failures if the coerced variables are the first, otherwise suitable, variables encountered.

# Author(s)

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

Based on an original idea and design by Mike Lonergan.

#### See Also

persp and gam.

```
library(mgcv)
set.seed(0)
n<-200;sig2<-4
x0 \leftarrow runif(n, 0, 1); x1 \leftarrow runif(n, 0, 1)
x2 <- runif(n, 0, 1)
y<-x0^2+x1*x2 +runif(n,-0.3,0.3)
g < -gam(y \sim s(x0, x1, x2))
old.par<-par(mfrow=c(2,2))
# display the prediction surface in x0, x1 ....
vis.gam(g,ticktype="detailed",color="heat",theta=-35)
vis.gam(g,se=2,theta=-35) # with twice standard error surfaces
vis.gam(g, view=c("x1","x2"),cond=list(x0=0.75)) # different view
vis.gam(g, view=c("x1","x2"),cond=list(x0=.75),theta=210,phi=40,
        too.far=.07)
# ..... areas where there is no data are not plotted
# contour examples....
vis.gam(g, view=c("x1","x2"),plot.type="contour",color="heat")
vis.gam(g, view=c("x1","x2"),plot.type="contour",color="terrain")
vis.gam(g, view=c("x1","x2"),plot.type="contour",color="topo")
vis.gam(g, view=c("x1","x2"),plot.type="contour",color="cm")
par(old.par)
# Examples with factor and "by" variables
```

ziP 251

```
fac<-rep(1:4,20)
x<-runif(80)
y<-fac+2*x^2+rnorm(80)*0.1
fac<-factor(fac)
b<-gam(y^fac+s(x))

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

z<-rnorm(80)*0.4
y<-as.numeric(fac)+3*x^2*z+rnorm(80)*0.1
b<-gam(y^fac+s(x,by=z))

vis.gam(b,theta=-35,color="heat",cond=list(z=1)) # by variable example

vis.gam(b,view=c("z","x"),theta= -135) # plot against by variable</pre>
```

ziP

GAM zero-inflated Poisson regression family

# **Description**

Family for use with gam, implementing regression for zero inflated Poisson data when the complimentary log log of the zero probability is linearly dependent on the log of the Poisson parameter. Use with great care, noting that simply having many zero response observations is not an indication of zero inflation: the question is whether you have too many zeroes given the specified model.

This sort of model is really only appropriate when none of your covariates help to explain the zeroes in your data. If your covariates predict which observations are likely to have zero mean then adding a zero inflated model on top of this is likely to lead to identifiability problems. Identifiability problems may lead to fit failures, or absurd values for the linear predictor or predicted values.

# Usage

```
ziP(theta = NULL, link = "identity",b=0)
```

# Arguments

theta	the 2 parameters controlling the slope and intercept of the linear transform of
	the mean controlling the zero inflation rate. If supplied then treated as fixed parameters ( $\theta_1$ and $\theta_2$ ), otherwise estimated.
link	The link function: only the "identity" is currently supported.
b	a non-negative constant, specifying the minimum dependence of the zero inflation rate on the linear predictor.

252 ziP

### **Details**

The probability of a zero count is given by 1-p, whereas the probability of count y>0 is given by the truncated Poisson probability function  $p\mu^y/((\exp(\mu)-1)y!)$ . The linear predictor gives  $\log \mu$ , while  $\eta = \log(-\log(1-p))$  and  $\eta = \theta_1 + \{b + \exp(\theta_2)\} \log \mu$ . The theta parameters are estimated alongside the smoothing parameters. Increasing the b parameter from zero can greatly reduce identifiability problems, particularly when there are very few non-zero data.

The fitted values for this model are the log of the Poisson parameter. Use the predict function with type=="response" to get the predicted expected response. Note that the theta parameters reported in model summaries are  $\theta_1$  and  $b + \exp(\theta_2)$ .

These models should be subject to very careful checking, especially if fitting has not converged. It is quite easy to set up models with identifiability problems, particularly if the data are not really zero inflated, but simply have many zeroes because the mean is very low in some parts of the covariate space. See example for some obvious checks. Take convergence warnings seriously.

#### Value

An object of class extended. family.

### **WARNINGS**

Zero inflated models are often over-used. Having lots of zeroes in the data does not in itself imply zero inflation. Having too many zeroes \*given the model mean\* may imply zero inflation.

## Author(s)

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

## References

Wood, S.N., N. Pya and B. Saefken (2015), Smoothing parameter and model selection for general smooth models. http://arxiv.org/abs/1511.03864

## See Also

```
ziplss
```

```
rzip <- function(gamma,theta= c(-2,.3)) {
## generate zero inflated Poisson random variables, where
## lambda = exp(gamma), eta = theta[1] + exp(theta[2])*gamma
## and 1-p = exp(-exp(eta)).
    y <- gamma; n <- length(y)
    lambda <- exp(gamma)
    eta <- theta[1] + exp(theta[2])*gamma
    p <- 1- exp(-exp(eta))
    ind <- p > runif(n)
    y[!ind] <- 0
    np <- sum(ind)</pre>
```

ziplss 253

```
## generate from zero truncated Poisson, given presence...
   y[ind] <- qpois(runif(np,dpois(0,lambda[ind]),1),lambda[ind])</pre>
   У
}
library(mgcv)
## Simulate some ziP data...
set.seed(1);n<-400
dat <- gamSim(1,n=n)</pre>
dat$y <- rzip(dat$f/4-1)
b \leftarrow gam(y\sim s(x0)+s(x1)+s(x2)+s(x3),family=ziP(),data=dat)
b$outer.info ## check convergence!!
b
plot(b,pages=1)
plot(b,pages=1,unconditional=TRUE) ## add s.p. uncertainty
gam.check(b)
## more checking...
## 1. If the zero inflation rate becomes decoupled from the linear predictor,
## it is possible for the linear predictor to be almost unbounded in regions
## containing many zeroes. So examine if the range of predicted values
## is sane for the zero cases?
range(predict(b,type="response")[b$y==0])
## 2. Further plots...
par(mfrow=c(2,2))
plot(predict(b,type="response"),residuals(b))
plot(predict(b,type="response"),b$y);abline(0,1,col=2)
plot(b$linear.predictors,b$y)
qq.gam(b,rep=20,level=1)
## 3. Refit fixing the theta parameters at their estimated values, to check we
## get essentially the same fit...
thb <- b$family$getTheta()</pre>
b0 <- gam(y\sim s(x0)+s(x1)+s(x2)+s(x3),family=ziP(theta=thb),data=dat)
b;b0
## Example fit forcing minimum linkage of prob present and
## linear predictor. Can fix some identifiability problems.
b2 \leftarrow gam(y^s(x0)+s(x1)+s(x2)+s(x3),family=ziP(b=.3),data=dat)
```

ziplss

Zero inflated Poisson location-scale model family

## **Description**

The ziplss family implements a zero inflated Poisson model in which one linear predictor controls the probability of presence and the other controls the mean given presence. Useable only with gam,

254 ziplss

the linear predictors are specified via a list of formulae. Should be used with care: simply having a large number of zeroes is not an indication of zero inflation.

Requires integer count data.

# Usage

```
ziplss(link=list("identity", "identity"))
```

## **Arguments**

link

two item list specifying the link - currently only identity links are possible, as parameterization is directly in terms of log of Poisson response and logit of probability of presence.

## **Details**

Used with gam to fit 2 stage zero inflated Poisson models. gam is called with a list containing 2 formulae, the first specifies the response on the left hand side and the structure of the linear predictor for the Poisson parameter on the right hand side. The second is one sided, specifying the linear predictor for the probability of presence on the right hand side.

The fitted values for this family will be a two column matrix. The first column is the log of the Poisson parameter, and the second column is the complimentary log log of probability of presnece.. Predictions using predict.gam will also produce 2 column matrices for type "link" and "response".

The null deviance computed for this model assumes that a single probability of presence and a single Poisson parameter are estimated.

For data with large areas of covariate space over which the response is zero it may be advisable to use low order penalties to avoid problems. For 1D smooths uses e.g. s(x,m=1) and for isotropic smooths use Duchon.splines in place of thin plaste terms with order 1 penalties, e.g s(x,z,m=c(1,.5)) — such smooths penaltize towards constants, thereby avoiding extreme estimates when the data are uninformative.

## Value

An object inheriting from class general. family.

### **WARNINGS**

Zero inflated models are often over-used. Having lots of zeroes in the data does not in itself imply zero inflation. Having too many zeroes \*given the model mean\* may imply zero inflation.

## Author(s)

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

## References

Wood, S.N., N. Pya and B. Saefken (2015), Smoothing parameter and model selection for general smooth models. http://arxiv.org/abs/1511.03864

ziplss 255

```
library(mgcv)
## simulate some data...
f0 \leftarrow function(x) 2 * sin(pi * x); f1 \leftarrow function(x) exp(2 * x)
f2 \leftarrow function(x) 0.2 * x^11 * (10 * (1 - x))^6 + 10 *
             (10 * x)^3 * (1 - x)^10
n <- 500; set.seed(5)</pre>
x0 <- runif(n); x1 <- runif(n)</pre>
x2 <- runif(n); x3 <- runif(n)</pre>
## Simulate probability of potential presence...
eta1 <- f0(x0) + f1(x1) - 3
p <- binomial()$linkinv(eta1)</pre>
y <- as.numeric(runif(n)<p) ## 1 for presence, 0 for absence
## Simulate y given potentially present (not exactly model fitted!)...
ind <- y>0
eta2 <- f2(x2[ind])/3
y[ind] <- rpois(exp(eta2),exp(eta2))</pre>
## Fit ZIP model...
b \leftarrow gam(list(y\sim s(x2)+s(x3),\sim s(x0)+s(x1)),family=ziplss())
b$outer.info ## check convergence
summary(b)
plot(b,pages=1)
```

# **Index**

*Topic <b>hplot</b>	get.var,86
exclude.too.far, 26	influence.gam, 88
plot.gam, 141	initial.sp,89
polys.plot, 146	inSide, 90
vis.gam, 248	interpret.gam, 91
*Topic <b>models</b>	jagam, 92
anova.gam,4	ldTweedie, 97
bam, 6	linear.functional.terms,98
bam.update, 12	logLik.gam, 102
bandchol, 14	magic, 104
betar, 15	magic.post.proc, 108
choose.k, 17	mgcv.FAQ, 110
cox.ph, 22	mgcv.package, 111
cSplineDes, 25	mgcv.parallel, 114
extract.lme.cov, 27	model.matrix.gam, 117
family.mgcv, 29	mono.con, 119
fix.family.link, 30	mroot, 120
fixDependence, 31	multinom, 121
formula.gam, 33	mvn, 122
formXtViX, 35	negbin, 124
fs.test, 36	new.name, 126
full.score, 37	notExp, 127
gam, 38	notExp2, 128
gam.check, 47	null.space.dimension, 130
gam.control, 50	ocat, 131
gam.convergence, 52	pcls, 133
gam.fit,53	pdIdnot, 136
gam.fit3, 54	pdTens, 138
gam.models, 57	pen.edf, 139
gam.outer, 63	place.knots, 140
gam.scale, 65	plot.gam, 141
gam.selection, 66	polys.plot, 146
gam.side,69	predict.bam, 147
gam.vcomp, 71	predict.gam, 150
gam2objective, 72	Predict.matrix, 155
gamm, 74	Predict.matrix.cr.smooth, 156
gamObject, 80	Predict.matrix.soap.film, 157
gamSim, 84	print.gam, 159
gaulss, 85	qq.gam, 161

residuals.gam, 165	uniquecombs, 246
rmvn, 167	vcov.gam, 247
Rrank, 168	vis.gam, 248
rTweedie, 169	ziP, 251
s, 170	ziplss, 253
scat, 173	*Topic <b>package</b>
sdiag, 174	mgcv.package, 111
single.index, 175	mgcv.parallel, 114
slanczos, 176	*Topic <b>regression</b>
smooth.construct, 178	anova.gam,4
smooth.construct.ad.smooth.spec,	bam, 6
183	bam.update, 12
<pre>smooth.construct.bs.smooth.spec,</pre>	bandchol, 14
186	betar, 15
<pre>smooth.construct.cr.smooth.spec,</pre>	choose.k, 17
188	cox.ph, 22
<pre>smooth.construct.ds.smooth.spec,</pre>	cSplineDes, 25
190	extract.lme.cov, 27
<pre>smooth.construct.fs.smooth.spec,</pre>	family.mgcv, 29
192	fix.family.link, 30
<pre>smooth.construct.gp.smooth.spec,</pre>	fixDependence, 31
194	formula.gam,33
<pre>smooth.construct.mrf.smooth.spec,</pre>	formXtViX, 35
197	fs.test, 36
<pre>smooth.construct.ps.smooth.spec,</pre>	full.score, 37
199	gam,38
<pre>smooth.construct.re.smooth.spec,</pre>	gam.check, 47
202	gam.control, $50$
<pre>smooth.construct.so.smooth.spec,</pre>	gam.convergence, 52
205	gam.fit,53
<pre>smooth.construct.sos.smooth.spec,</pre>	gam.fit3, <u>54</u>
211	gam.models, 57
smooth.construct.t2.smooth.spec,	gam.outer, 63
214	gam.scale,65
<pre>smooth.construct.tensor.smooth.spec,</pre>	gam.selection, 66
215	gam.side,69
smooth.construct.tp.smooth.spec,	gam.vcomp, 71
216	gam2objective, 72
smoothCon, 222	gamm, <b>74</b>
sp.vcov, 224	gamObject, $80$
spasm.construct, 225	gamSim, 84
step.gam, 226	gaulss, 85
summary.gam, 228	get.var, <mark>86</mark>
t2, 232	influence.gam, 88
te, 237	initial.sp, 89
tensor.prod.model.matrix, 241	inSide, 90
trichol, 242	interpret.gam, 91
Tweedie, 244	jagam, 92

ldTweedie, 97	186
linear.functional.terms, 98	smooth.construct.cr.smooth.spec,
logLik.gam, 102	188
magic, 104	<pre>smooth.construct.ds.smooth.spec,</pre>
magic.post.proc, 108	190
mgcv.FAQ, 110	<pre>smooth.construct.fs.smooth.spec,</pre>
mgcv.package, 111	192
mgcv.parallel, 114	<pre>smooth.construct.gp.smooth.spec,</pre>
missing.data, 116	194
model.matrix.gam, 117	smooth.construct.mrf.smooth.spec,
mono.con, 119	197
mroot, 120	smooth.construct.ps.smooth.spec,
multinom, 121	199
mvn, 122	<pre>smooth.construct.re.smooth.spec,</pre>
negbin, 124	202
new.name, 126	smooth.construct.so.smooth.spec,
notExp, 127	205
notExp2, 128	<pre>smooth.construct.sos.smooth.spec,</pre>
null.space.dimension, 130	211
ocat, 131	<pre>smooth.construct.t2.smooth.spec,</pre>
pcls, 133	214
pdIdnot, 136	<pre>smooth.construct.tensor.smooth.spec,</pre>
pdTens, 138	215
pen.edf, 139	<pre>smooth.construct.tp.smooth.spec,</pre>
place.knots, 140	216
plot.gam, 141	smooth.terms, 219
polys.plot, 146	smoothCon, 222
predict.bam, 147	sp.vcov, 224
predict.gam, 150	spasm.construct, 225
Predict.matrix, 155	step.gam, 226
Predict.matrix.cr.smooth, 156	summary.gam, 228
Predict.matrix.soap.film, 157	t2, 232
print.gam, 159	te, 237
qq.gam, 161	tensor.prod.model.matrix, 241
random.effects, 163	trichol, 242
residuals.gam, 165	Tweedie, 244
rmvn, 167	uniquecombs, 246
Rrank, 168	vcov.gam, 247
rTweedie, 169	vis.gam, 248
s, 170	ziP, 251
scat, 173	ziplss, 253
sdiag, 174	*Topic <b>smooth</b>
single.index, 175	anova.gam,4
slanczos, 176	bam, 6
smooth.construct, 178	bam.update, 12
smooth.construct.ad.smooth.spec,	bandchol, 14
183	cSplineDes, 25
<pre>smooth.construct.bs.smooth.spec,</pre>	extract.lme.cov,27

formula.gam, 33	Rrank, 168
formXtViX, 35	s, 170
fs. test, 36	sdiag, 174
full.score, 37	slanczos, 176
gam, 38	smooth.construct, 178
gam.check, 47	<pre>smooth.construct.so.smooth.spec,</pre>
gam.control, 50	205
gam.convergence, 52	smoothCon, 222
gam.fit,53	sp.vcov, 224
gam.fit3, 54	spasm.construct, 225
gam.outer, 63	summary.gam, 228
gam.scale, 65	t2, 232
gam.vcomp, 71	te, 237
gam2objective, 72	tensor.prod.model.matrix, 241
gamm, 74	trichol, 242
gamObject, 80	vcov.gam, 247
gamSim, 84	vis.gam, 248
get.var, 86	
influence.gam, 88	adaptive.smooth, 180, 201, 221
initial.sp, 89	adaptive.smooth
inSide, 90	(smooth.construct.ad.smooth.spec)
interpret.gam, 91	183
jagam, 92	AIC, 67, 102, 103
logLik.gam, 102	AIC.gam(logLik.gam), 102
magic, 104	anova.gam, 4, 61, 112, 164, 231
magic.post.proc, 108	anova.glm, $5$
mgcv.package, 111	
mgcv.parallel, 114	b.spline
model.matrix.gam, 117	(smooth.construct.bs.smooth.spec)
mono.con, 119	186
mroot, 120	bam, 6, 12, 13, 29, 38, 52, 65, 95, 112, 114,
new.name, 126	125, 147–150, 198, 204, 223
notExp, 127	bam.update, 12
notExp2, 128	bandchol, 14, 243
pcls, 133	betar, 15, 29
pdIdnot, 136	bug.reports.mgcv, 16
pdTens, 138	choose.k, 17, 33, 38, 42, 48, 49, 58, 112, 162,
pen.edf, 139	170, 233, 237
place.knots, 140	coef.pdIdnot(pdIdnot), 136
plot.gam, 141	coef.pdTens (pdTens), 138
polys.plot, 146	columb, 20
predict.bam, 147	columb.polys, <i>147</i> , <i>198</i>
predict.gam, 150	concurvity, 21
Predict.matrix, 155	contour, 249
Predict.matrix.soap.film, 157	corClasses, 74
print.gam, 159	corMatrix.pdIdnot(pdIdnot), 136
qq.gam, 161	cox.ph, 22, 30, 165
residuals.gam, 165	cSplineDes, 25, 201, 216

cubic.regression.spline, <i>180</i> , <i>219</i> , <i>221</i>	gam, 6, 10, 15, 21, 23, 29–31, 33, 34, 37, 38,
cubic.regression.spline	38, 42, 48–50, 52–54, 56, 57, 60,
<pre>(smooth.construct.cr.smooth.spec),</pre>	64–69, 71, 73, 74, 76, 81–86, 88, 89,
188	91, 92, 94, 95, 98, 107, 110,
cyclic.cubic.spline, 219	112–114, 116, 118, 121, 122, 124,
cyclic.cubic.spline	125, 131, 145, 149, 151, 152, 156,
<pre>(smooth.construct.cr.smooth.spec),</pre>	157, 160, 162, 164, 166, 171–173,
188	175, 179, 182–184, 186, 188, 190,
cyclic.p.spline, <i>25</i> , <i>177</i> , <i>220</i>	191, 193, 195, 198, 199, 203, 204,
cyclic.p.spline	206, 211, 214–217, 219, 223, 225,
<pre>(smooth.construct.ps.smooth.spec),</pre>	226, 231–233, 235, 238, 240, 244,
199	248, 250, 251, 253, 254
	gam.check, 6, 11, 18, 33, 38, 42, 44, 47, 112,
detectCores, 10	231
Dim.pdIdnot (pdIdnot), 136	gam.control, 7, 11, 37, 39, 44, 50, 52, 53, 65,
drop1, 4	93, 112, 223, 224
Duchon.spline, 211, 213, 219, 221, 233, 237,	gam.convergence, 52, 112
254	gam.fit, 37, 38, 41, 50, 52, 53, 56, 64
Duchon.spline	gam.fit3, <i>31</i> , <i>41</i> , <i>50</i> , <i>54</i> , 54, <i>64</i> , <i>65</i> , <i>73</i>
(smooth.construct.ds.smooth.spec),	gam.models, 7, 8, 11, 33, 34, 38, 40, 44, 57,
190	70, 74, 92, 112, 116, 164, 171, 192,
duplicated, 246, 247	194, 221, 233, 238
	gam.outer, 37, 52, 63, 72, 89
exclude.too.far,26	gam.performance(gam.convergence), 52
extract.lme.cov, 27	gam.scale, <i>51</i> , <i>65</i>
extract.lme.cov2, 35	gam.selection, 11, 38, 44, 66, 112, 227
extract.lme.cov2(extract.lme.cov), 27	gam.side, 11, 34, 44, 69
	gam.vcomp, 60, 71, 116, 164, 204, 225, 231
factor, 58	gam2derivative (gam2objective), 72
factor.smooth.interaction, 58, 143, 221	gam2objective, 72
factor.smooth.interaction	gamm, 27–29, 35, 38, 51, 60, 65, 74, 84, 91, 92,
(smooth.construct.fs.smooth.spec),	95, 112, 125–129, 137, 139, 152,
192	156, 163, 164, 172, 181–183,
family, 7, 29, 39, 57, 92	192–194, 198, 204, 205, 221, 222,
family.mgcv, 29, 38, 39, 112	235, 240
fix.family.link, 30, 56	gamObject, 10, 11, 13, 33, 42, 44, 77, 80, 95,
fix.family.ls(fix.family.link),30	112, 160, 247
fix.family.qf(fix.family.link), 30	gamSim, 84
fix.family.rd(fix.family.link),30	gaulss, <i>30</i> , 85
fix.family.var(fix.family.link), 30	gaussian, 123
fixDependence, 31, 69	get.var, 86, 182
formula.gam, 7, 33, 38, 74, 92	glm, 7, 29, 33, 39, 57, 74, 75, 92, 112
formXtViX, 29, 35	glm.control, <i>52</i> , <i>55</i>
fs.boundary (fs.test), 36	glm.fit, 41,56
fs.test, 36	gp. smooth, 220
full.score, 37	gp.smooth
function.predictors	(smooth.construct.gp.smooth.spec),
(linear.functional.terms).98	194

grey, <i>144</i>	nlm, <i>37</i> , <i>50</i> , <i>51</i> , <i>72</i> , <i>73</i> , <i>82</i>
	notExp, 127, <i>128</i> , <i>129</i>
heat.colors, 144	notExp2, <i>76</i> , <i>77</i> , <i>127</i> , 128
imaga 144 240	notLog, <i>128</i>
image, 144, 249	notLog (notExp), 127
in.out, 87, 198, 199	notLog2, <i>127</i> , <i>136–138</i>
influence.gam, 88	notLog2 (notExp2), 128
initial.sp, 89	null.space.dimension, 130, 170
inSide, 90	
interpret.gam,91	ocat, 29, 122, 131
jagam, 92	optim, 51, 72, 73, 82
, a.g.a, , _	options, 129
ldTweedie, 97, 168, 170, 244, 245	ordered, 58
linear.functional.terms, <i>11</i> , <i>34</i> , <i>38</i> , <i>42</i> ,	ordered.categorical (ocat), 131
44, 60, 98, 112, 171, 219, 233, 238	<b>5</b> , , , , ,
lme, 28, 74, 75, 126	p.spline, 180, 186, 187, 221, 233, 237
lmeControl, 75	p.spline
locator, 206	(smooth.construct.ps.smooth.spec),
logDet.pdIdnot(pdIdnot), 136	199
logLik, <i>102</i>	parallel, <i>6</i> , <i>114</i>
logLik.gam, 102	par Lapply, 9, 149
ls.size, 103	pcls, 113, 119, 133
	pdConstruct.pdIdnot(pdIdnot), 136
magic, 11, 44, 49, 53, 54, 56, 65, 73, 77, 89,	pdConstruct.pdTens (pdTens), 138
104, 109, 112–114, 119, 134	
magic.post.proc, <i>107</i> , 108	pdFactor.pdIdnot (pdIdnot), 136
makeCluster, <i>10</i>	pdFactor.pdTens (pdTens), 138
makeForkCluster, <i>10</i>	pdIdnot, 128, 129, 136
nem.limits,77	pdMatrix.pdIdnot(pdIdnot), 136
mgcv (mgcv.package), 111	pdMatrix.pdTens(pdTens), 138
mgcv-package (mgcv.package), 111	pdTens, 77, 128, 129, 136, 137, 138
ngcv.FAQ, 110, <i>113</i>	pen.edf, 139, 234
ngcv.package, 111	persp, 249, 250
ngcv.parallel, <i>11</i> ,114	persp.gam (vis.gam), 248
missing.data,116	place.knots, 140
model.matrix.gam, 117	plot.gam, 11, 18, 44, 77, 112, 141, 152, 181
nono.con, 119, <i>134</i>	poisson, 244
nrf, 112, 143, 146, 147, 220, 221	polys.plot, 146, <i>199</i>
<pre>mrf(smooth.construct.mrf.smooth.spec),</pre>	power, 244
197	predict.bam, 147
mroot, 120	predict.gam, 6, 9, 11, 44, 76, 77, 85, 110,
multinom, <i>30</i> , 121	112, 118, 131, 145, 147, 149, 150,
nvn, <i>30</i> , <i>34</i> , 122	150, <i>157</i> , <i>231</i> , <i>254</i>
	Predict.matrix, 155, 155, 156, 157, 178,
na.action, 82	181, 182, 223, 224
nb, 29, 124	Predict.matrix.Bspline.smooth
nb (negbin), 124	<pre>(smooth.construct.bs.smooth.spec),</pre>
negbin, 7, 11, 29, 44, 77, 124, 124	186
new.name, 126	Predict.matrix.cr.smooth, 156

Predict.matrix.cs.smooth	residuals.gam, <i>48</i> , <i>161</i> ,165
(Predict.matrix.cr.smooth), 156	rig, 166
Predict.matrix.cyclic.smooth	rmvn, 167
(Predict.matrix.cr.smooth), 156	Rrank, 168
Predict.matrix.duchon.spline	rTweedie, 169, 245
(smooth.construct.ds.smooth.spec),	
190	s, 11, 17, 33, 38, 42, 44, 51, 52, 57–59, 66, 74,
Predict.matrix.fs.interaction	77, 92, 99, 112, 116, 156, 170, 178,
(smooth.construct.fs.smooth.spec),	182, 191, 195, 203, 217, 219, 221,
192	226, 234, 235, 239, 240
Predict.matrix.gp.smooth	scat, 29, 173
<pre>(smooth.construct.gp.smooth.spec),</pre>	sdiag, 174
194	sdiag<- (sdiag), 174
Predict.matrix.mrf.smooth	sessionInfo, <i>16</i>
<pre>(smooth.construct.mrf.smooth.spec),</pre>	setwd, <i>93</i>
197	signal.regression
Predict.matrix.pspline.smooth	(linear.functional.terms), 98
(Predict.matrix.cr.smooth), 156	sim2jam(jagam), 92
Predict.matrix.random.effect	single.index, 175
(smooth.construct.re.smooth.spec),	sink, 228
202	slanczos, 176
Predict.matrix.sf	smooth.construct, 42, 69, 82, 112, 155, 156,
(Predict.matrix.soap.film), 157	171, 178, 187, 189, 191, 196, 201,
Predict.matrix.soap.film, 157, 207	212, 214, 216, 218, 223, 224, 233,
Predict.matrix.sos.smooth	237
<pre>(smooth.construct.sos.smooth.spec),</pre>	smooth.construct.ad.smooth.spec, 183
211	smooth.construct.bs.smooth.spec, 186
Predict.matrix.sw	<pre>smooth.construct.cc.smooth.spec, 141</pre>
(Predict.matrix.soap.film), 157	<pre>smooth.construct.cc.smooth.spec</pre>
Predict.matrix.t2.smooth	(smooth.construct.cr.smooth.spec)
(Predict.matrix.cr.smooth), 156	188
Predict.matrix.tensor.smooth	<pre>smooth.construct.cp.smooth.spec</pre>
(Predict.matrix.cr.smooth), 156	(smooth.construct.ps.smooth.spec)
Predict.matrix.tprs.smooth	199
(Predict.matrix.cr.smooth), 156	smooth.construct.cr.smooth.spec, 188
Predict.matrix.ts.smooth	<pre>smooth.construct.cs.smooth.spec</pre>
(Predict.matrix.cr.smooth), 156	(smooth.construct.cr.smooth.spec)
Predict.matrix2 (Predict.matrix), 155	188
PredictMat, <i>155</i> , <i>156</i> , <i>18</i> 2	smooth.construct.ds.smooth.spec, 190
PredictMat (smoothCon), 222	smooth.construct.fs.smooth.spec, 192
print.anova.gam(anova.gam),4	smooth.construct.gp.smooth.spec, 194,
print.gam, 159	221
print.summary.gam(summary.gam), 228	smooth.construct.mrf.smooth.spec, 197
	smooth.construct.ps.smooth.spec, 199
qq.gam, <i>31</i> , <i>48</i> , <i>49</i> , 161	<pre>smooth.construct.re.smooth.spec, 60, 72,</pre>
qr, <i>223</i>	116, 164, 202, 220, 221
quasi, <i>244</i>	<pre>smooth.construct.sf.smooth.spec</pre>
	(smooth.construct.so.smooth.spec)
random.effects, 38, 60, 112, 143, 163	205

smooth.construct.so.smooth.spec, <i>158</i> ,	termplot, 143, 144, 149, 151
205	ti, 33, 34, 38, 42, 58, 70, 92, 112, 219, 220
smooth.construct.sos.smooth.spec, 211	ti (te), 237
smooth.construct.sw.smooth.spec	tprs, 8, 33, 40, 93, 110, 131, 180, 186, 190,
(smooth.construct.so.smooth.spec),	196, 219, 221
205	<pre>tprs (smooth.construct.tp.smooth.spec),</pre>
smooth.construct.t2.smooth.spec, 214	216
smooth.construct.tensor.smooth.spec,	trichol, 242
138, 215, 240, 242	tw, 29
smooth.construct.tp.smooth.spec, 216	tw (Tweedie), 244
smooth.construct.ts.smooth.spec	Tweedie, 29, 168, 170, 244
(smooth.construct.tp.smooth.spec),	
216	unique, 246, 247
smooth.construct2, 184, 188, 191, 195, 205,	uniquecombs, 246
217	user.defined.smooth, <i>33</i> , <i>219</i> , <i>221</i>
<pre>smooth.construct2 (smooth.construct),</pre>	<pre>user.defined.smooth(smooth.construct),</pre>
178	178
smooth.terms, 11, 33, 38, 42, 44, 112, 164,	
<i>171</i> , <i>214</i> , <i>216</i> , 219, <i>233</i> , <i>237</i>	vcov.gam, 247
smoothCon, 178, 182, 205, 222	vis.gam, 11, 27, 44, 77, 83, 112, 144, 145, 248
soap, 220, 221	-: D 20 251
<pre>soap (smooth.construct.so.smooth.spec),</pre>	ziP, 30, 251
205	ziplss, <i>30</i> , <i>252</i> , <i>253</i>
solve.pdIdnot(pdIdnot), 136	
sp. vcov, 224, 231	
spasm.construct, 225	
spasm.smooth(spasm.construct), 225	
spasm.sp(spasm.construct), 225	
Spherical. Spline, <i>143</i> , <i>192</i> , <i>219</i> , <i>221</i>	
Spherical.Spline	
(smooth.construct.sos.smooth.spec),	
211	
step.gam, 68, 226	
summary.gam, 5, 6, 11, 44, 61, 77, 110, 112,	
160, 164, 228	
summary.pdIdnot(pdIdnot), 136	
summary.pdTens(pdTens), 138	
Summary . parens (parens), 130	
t.scaled(scat), 173	
t2, 33, 34, 38, 42, 92, 112, 139, 140, 214, 215,	
219–221, 232	
te, 11, 17, 33, 34, 38, 42, 44, 52, 57–59, 66,	
74, 77, 92, 99, 112, 137, 139, 156,	
172, 178, 188, 199, 214, 216,	
219–221, 232, 235, 237, 242	
tensor.prod.model.matrix, 238, 241	
tensor.prod.penalties, 238	
tensor.prod.penalties	
(tensor.prod.model.matrix), 241	