Package: mgcv (via r-universe)

October 18, 2026

Version 1.9-4

Title Mixed GAM Computation Vehicle with Automatic Smoothness Estimation

Description Generalized additive (mixed) models, some of their extensions and other generalized ridge regression with multiple smoothing parameter estimation by (Restricted) Marginal Likelihood, Cross Validation and similar, or using iterated nested Laplace approximation for fully Bayesian inference. See Wood (2025) <doi:10.1146/annurev-statistics-112723-034249> for an overview. Includes a gam() function, a wide variety of smoothers, 'JAGS' support and distributions beyond the exponential family.

Priority recommended

Depends R (>= 4.4.0), nlme (>= 3.1-64)

Imports methods, stats, graphics, Matrix, splines, utils

Suggests parallel, survival, MASS

LazyLoad yes

ByteCompile yes

License GPL (>= 2)

NeedsCompilation yes

Author Simon Wood [aut, cre]

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

Repository https://noamross.r-universe.dev

Date/Publication 2025-11-07 17:39:30 UTC

RemoteUrl https://github.com/noamross/mgcv

RemoteRef HEAD

RemoteSha 1b6a4c8374612da27e36420b4459e93acb183f2d

Contents

anova.gam . . . 5
bam . . . 8
bam.update . . . 14
bandchol . . . 16
bcg . . . 17
betar . . . 19
blas.thread.test . . . 21
bug.reports.mgcv . . . 22
choldrop . . . 22
choose.k . . . 24
clog . . . 26
cnorm . . . 28
columb . . . 30
concurvity . . . 32
cox.ph . . . 33
cox.pht . . . 38
cpois . . . 40
cSplineDes . . . 42
dDeta . . . 43
dpnorm . . . 44
dppois . . . 45
exclude.too.far . . . 46
extract.lme.cov . . . 47
factor.smooth . . . 49
family.mgcv . . . 50
FFdes . . . 52
fix.family.link . . . 53
fixDependence . . . 54
formula.gam . . . 56
formXtViX . . . 58
fs.test . . . 59
gam . . . 60
gam.check . . . 71
gam.control . . . 73
gam.convergence . . . 75
gam.fit . . . 77
gam.fit3 . . . 78
gam.fit5.post.proc . . . 81
gam.mh . . . 82
gam.models . . . 84
gam.outer . . . 91
gam.reparam . . . 92
gam.scale . . . 93
gam.selection . . . 94
gam.side . . . 97
gam.vcomp . . . 99

gam2objective . . . 101
gamlss.etamu . . . 102
gamlss.gH . . . 103
gamm . . . 104
gammals . . . 111
gamObject . . . 112
gamSim . . . 116
gaulss . . . 117
get.var . . . 118
gevlss . . . 119
gfam . . . 121
ginla . . . 124
gumbls . . . 127
identifiability . . . 129
in.out . . . 130
influence.gam . . . 131
initial.sp . . . 132
inSide . . . 133
interpret.gam . . . 134
jagam . . . 135
k.check . . . 140
ldetS . . . 141
ldTweedie . . . 143
linear.functional.terms . . . 144
logLik.gam . . . 149
lp . . . 150
ls.size . . . 152
magic . . . 153
magic.post.proc . . . 158
mchol . . . 159
mgcv.FAQ . . . 160
mgcv.package . . . 162
mgcv.parallel . . . 164
mini.roots . . . 166
missing.data . . . 167
model.matrix.gam . . . 169
mono.con . . . 170
mroot . . . 171
multinom . . . 172
mvn . . . 174
NCV . . . 175
negbin . . . 178
new.name . . . 181
notExp . . . 182
notExp2 . . . 183
null.space.dimension . . . 184
ocat . . . 185
one.se.rule . . . 187

pcls . . . 188
pdIdnot . . . 192
pdTens . . . 193
pen.edf . . . 195
place.knots . . . 196
plot.gam . . . 197
polys.plot . . . 202
predict.bam . . . 204
predict.gam . . . 207
Predict.matrix . . . 213
Predict.matrix.cr.smooth . . . 214
Predict.matrix.soap.film . . . 216
print.gam . . . 218
psum.chisq . . . 219
qq.gam . . . 221
random.effects . . . 223
residuals.gam . . . 226
rig . . . 227
rmvn . . . 228
Rrank . . . 229
rTweedie . . . 230
s . . . 231
scasm . . . 234
scat . . . 239
sdiag . . . 240
shash . . . 241
single.index . . . 244
Sl.inirep . . . 245
Sl.repara . . . 246
Sl.setup . . . 247
slanczos . . . 248
smooth.construct . . . 250
smooth.construct.ad.smooth.spec . . . 255
smooth.construct.bs.smooth.spec . . . 258
smooth.construct.cr.smooth.spec . . . 261
smooth.construct.ds.smooth.spec . . . 263
smooth.construct.fs.smooth.spec . . . 266
smooth.construct.gp.smooth.spec . . . 268
smooth.construct.mrf.smooth.spec . . . 271
smooth.construct.ps.smooth.spec . . . 274
smooth.construct.re.smooth.spec . . . 277
smooth.construct.so.smooth.spec . . . 279
smooth.construct.sos.smooth.spec . . . 285
smooth.construct.sz.smooth.spec . . . 288
smooth.construct.t2.smooth.spec . . . 290
smooth.construct.tensor.smooth.spec . . . 291
smooth.construct.tp.smooth.spec . . . 293
smooth.info . . . 296

smooth.terms . . . 297
smooth2random . . . 300
smoothCon . . . 302
sp.vcov . . . 305
spasm.construct . . . 306
step.gam . . . 307
summary.gam . . . 308
t2 . . . 313
te . . . 317
tensor.prod.model.matrix . . . 321
totalPenaltySpace . . . 323
trichol . . . 324
trind.generator . . . 325
Tweedie . . . 327
twlss . . . 329
uniquecombs . . . 331
vcov.gam . . . 332
vis.gam . . . 334
XWXd . . . 336
ziP . . . 340
ziplss . . . 342

Index **345**

`anova.gam` *Approximate hypothesis tests related to GAM fits*

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 or GLRT test: this latter approach should not be use to test the significance of terms which can be penalized to zero. Models to be compared should be fitted to the same data using the same smoothing parameter selection method.

Usage

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

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"`. Reset to `"Chisq"` for extended and general families unless `NULL`.
`freq`	whether to use frequentist or Bayesian approximations for parametric term p-values. See `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 (when possible this is a smoothing parameter uncertainty corrected version). For extended and general families this is set so that a GLRT test is used. The p-values resulting from the multi-model case 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 (such as random effects) 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 doi:10.1093/biomet/ass048

Wood, S.N. (2013b) A simple test for random effects in regression models. Biometrika 100:1005-1010 doi:10.1093/biomet/ast038

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")
```

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 extended families listed in family.mgcv can also be used). 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 (Wood et al. 2017, Li and Wood, 2019) 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 that are practical. Number of response data can not exceed .Machine$integer.max.

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,
    cluster=NULL,nthreads=1,gc.level=0,use.chol=FALSE,samfrac=1,
    coef=NULL,drop.unused.levels=TRUE,G=NULL,fit=TRUE,drop.intercept=NULL,
    in.out=NULL,nei=NULL,...)
```

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 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. The extended families listed in family.mgcv can also be used.
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, 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. When `discrete=TRUE` then `"NCV"` is also possible.

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

`select` Should selection penalties be added to the smooth effects, so that they can in principle be penalized out of the model? See `gamma` to increase penalization. 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` Increase above 1 to force smoother fits. `gamma` is used to multiply the effective degrees of freedom in the GCV/UBRE/AIC score (so `log(n)/2` is BIC like). `n/gamma` can be viewed as an effective sample size, which allows it to play a similar role for RE/ML smoothing parameter estimation.

`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. Note that `discrete=TRUE`may result in re-ordering of variables in tensor product smooths for improved efficiency, and `sp` must be supplied in re-ordered order.

`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. Also usable with other models when `discrete=TRUE`, in which case the AR model is applied to the working residuals and corresponds to a GEE approximation.

`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. Parametric terms use the maximum number specified.

`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))`. See details.

`gc.level` to keep the memory footprint down, it can help 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. Not as much of a problem as it used to be, but can really matter for very large datasets.

`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.

`coef` initial values for model coefficients

`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 sp, chunk.size, gamma,nthreads, cluster, rho, gc.level, samfrac, use.chol, method and scale (if >0).
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.
drop.intercept	Set to TRUE to force the model to really not have the a constant in the parametric model part, even with factor variables present.
in.out	If supplied then this is a two item list of intial values. sp is initial smoothing parameter estiamtes and scale the initial scale parameter estimate (set to 1 if famiy does not have one).
nei	list describing neighbourhood structure for NCV model smoothing parameter selection. See NCV for details.
...	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. See Wood et al (2017) and Li and Wood (2019) for full details.

With `discrete=TRUE` NCV can be used for smoothing parameter estimation. See `NCV` for details of how to set up neighbourhoods using the `nei` argument. NCV is applied to the working penalized linear model using in iterative fitting. The computation is more efficient than that used by `gam` provided neighbourhoods are small, but is still costly. It is not efficient for k-fold cross validation, for example. The small neighbourhood restriction is because a matrix of side length given by the neibourhood side needs to be inverted for each neighbourhood. Because computational cost is typically much higher than for REML, for large data/models it is usually worth basing the NCV criterion on a sub-sample of the full data set. Supplying a `sample` element of the `nei` list is the way to do this. See `NCV`.

When `discrete=TRUE` parallel computation is controlled using the `nthreads` argument. For this method no cluster computation is used, and the `parallel` package is not required. Note that actual speed up from parallelization depends on the BLAS installed and your hardware. With the (R default) reference BLAS using several threads can make a substantial difference, but with a single threaded tuned BLAS, such as openblas, the effect is less marked (since cache use is typically optimized for one thread, and is then sub optimal for several). However the tuned BLAS is usually much faster than using the reference BLAS, however many threads you use. If you have a multi-threaded BLAS installed then you should leave `nthreads` at 1, since calling a multi-threaded BLAS from multiple threads usually slows things down: the only exception to this is that you might choose to form discrete matrix cross products (the main cost in the fitting routine) in a multi-threaded way, but use single threaded code for other computations: this can be achieved by e.g. `nthreads=c(2,1)`, which would use 2 threads for discrete inner products, and 1 for most code calling BLAS. Not that the basic reason that multi-threaded performance is often disappointing is that most computers are heavily memory bandwidth limited, not flop rate limited. It is hard to get data to one core fast enough, let alone trying to get data simultaneously to several cores.

`discrete=TRUE` will often produce identical results to the methods without discretization, since covariates often only take a modest number of discrete values anyway, so no approximation at all is involved in the discretization process. Even when some approximation is involved, the differences are often very small as the algorithms discretize marginally whenever possible. For example each margin of a tensor product smooth is discretized separately, rather than discretizing onto a grid of covariate values (for an equivalent isotropic smooth we would have to discretize onto a grid). The marginal approach allows quite fine scale discretization and hence very low approximation error. Note that when using the smooth `id` mechanism to link smoothing parameters, the discrete method cannot force the linked bases to be identical, so some differences to the none discrete methods will be noticable.

The extended families given in family.mgcv can also be used. The extra parameters of these are estimated by maximizing the penalized likelihood, rather than the restricted marginal likelihood as in gam. So estimates may differ slightly from those returned by gam. Estimation is accomplished by a Newton iteration to find the extra parameters (e.g. the theta parameter of the negative binomial or the degrees of freedom and scale of the scaled t) maximizing the log likelihood given the model coefficients at each iteration of the fitting procedure.

Value

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

WARNINGS

For method set to "fREML", "REML" or "ML" the returned smoothness criterion value is for the model itself, otherwise for the working model. The former is comparable to gam (but typically a little larger, since it was the working model version that was actually optimized), the latter not.

The routine may be slower than optimal if the default "tp" basis is used.

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

With discrete=TRUE, te terms are efficiently computed, but t2 are not.

Anything close to the maximum n of .Machine$integer.max will need a very large amount of RAM and probably gc.level=1.

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. doi:10.1111/rssc.12068

Wood, S.N., Li, Z., Shaddick, G. & Augustin N.H. (2017) Generalized additive models for gigadata: modelling the UK black smoke network daily data. Journal of the American Statistical Association. 112(519):1199-1210 doi:10.1080/01621459.2016.1195744

Li, Z & S.N. Wood (2020) Faster model matrix crossproducts for large generalized linear models with discretized covariates. Statistics and Computing. 30:19-25 doi:10.1007/s11222019098642

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

## Sample sizes are small for fast run times.
```

```
set.seed(3)
dat <- gamSim(1,n=25000,dist="normal",scale=20)
bs <- "cr";k <- 12
b <- bam(y ~ 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 ~ 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)

## A Poisson example...

k <- 15
dat <- gamSim(1,n=21000,dist="poisson",scale=.1)

system.time(b1 <- bam(y ~ s(x0,bs=bs)+s(x1,bs=bs)+s(x2,bs=bs,k=k),
            data=dat,family=poisson()))
b1

## Similar using faster discrete method...


system.time(b2 <- bam(y ~ 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,family=poisson(),discrete=TRUE))
b2
```

`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

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

WARNINGS

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

Author(s)

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

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)
dat[c(50,13,3000,3005,3100),]<- NA
dat1 <- dat[(n-999):n,]
dat0 <- dat[1:(n-1000),]
bs <- "ps";k <- 20
method <- "GCV.Cp"
b <- bam(y ~ 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=dat0,method=method)

b1 <- bam.update(b,dat1)

b2 <- bam.update(bam.update(b,dat1[1:500,]),dat1[501:1000,])

b3 <- bam(y ~ 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] <- 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,]
dat0 <- dat[1:(n-1000),]

b <- bam(y ~ 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=dat0,rho=0.7)

b1 <- bam.update(b,dat1)


summary(b1);summary(b2);summary(b3)
```

---

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

B	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.

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)
```

---

bcg *Censored Box-Cox Gaussian family*

Description

Family for use with gam or bam, implementing regression for censored data that can be modelled as Gaussian after Box-Cox transformation. If $y > 0$ is the response and

$$z = \begin{cases} (y^\lambda - 1)/\lambda & \lambda \neq 0 \\ \log(y) & \lambda = 0 \end{cases}$$

with mean μ and standard deviation $w^{-1/2}\exp(\theta)$, then $w^{1/2}(z-\mu)\exp(-\theta)$ follows an $N(0,1)$ distribution. That is

$$z \sim N(\mu, e^{2\theta}w^{-1}).$$

θ is a single scalar for all observations, and similarly Box-Cox parameter λ is a single scalar. Observations may be left, interval or right censored or uncensored.

Note that the regression model here specifies the mean of the Box-Cox transformed response, not the mean of the response itself: this is rather different to the usual GLM approach.

Usage

```
bcg(theta=NULL,link="identity")
```

Arguments

`theta`	a 2-vector containing the Box-Cox parameter λ and the log standard deviation parameter θ. If supplied and positive then taken as a fixed value of λ and $\exp(\theta)$. If supplied and second element negative taken as initial value and negative of initial value respectively.
`link`	The link function: `"identity"`, `"log"` or `"sqrt"`.

Details

If the family is used with a vector response, then it is assumed that there is no censoring. If there is censoring then the response should be supplied as a two column matrix. The first column is always numeric. Entries in the second column are as follows.

- If an entry is identical to the corresponding first column entry, then it is an uncensored observation.
- If an entry is numeric and different to the first column entry then there is interval censoring. The first column entry is the lower interval limit and the second column entry is the upper interval limit. y is only known to be between these limits.
- If the second column entry is `0` then the observation is left censored at the value of the entry in the first column. It is only known that y is less than or equal to the first column value.
- If the second column entry is `Inf` then the observation is right censored at the value of the entry in the first column. It is only known that y is greater than or equal to the first column value.

Any mixture of censored and uncensored data is allowed, but be aware that data consisting only of right and/or left censored data contain very little information.

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 (2016), Smoothing parameter and model selection for general smooth models. Journal of the American Statistical Association 111, 1548-1575 doi:10.1080/01621459.2016.1180986

See Also

cnorm, cpois, clog

Examples

```
library(mgcv)

set.seed(3) ## Simulate some gamma data?
dat <- gamSim(1,n=400,dist="normal",scale=1)
dat$f <- dat$f/4 ## true linear predictor
Ey <- exp(dat$f);scale <- .5 ## mean and GLM scale parameter
dat$y <- rgamma(Ey*0,shape=1/scale,scale=Ey*scale)

## Just Box-Cox no censoring...
b <- gam(y~s(x0)+s(x1)+s(x2)+s(x3),family=bcg,data=dat)
summary(b)
plot(b,pages=1,scheme=1)

## try various censoring...
yb <- cbind(dat$y,dat$y)
ind <- 1:100
yb[ind,2] <- yb[ind,1] + runif(100)*3
yb[51:100,2] <- 0 ## left censored
yb[101:140,2] <- Inf ## right censored

b <- gam(yb~s(x0)+s(x1)+s(x2)+s(x3),family=bcg,data=dat)
summary(b)
plot(b,pages=1,scheme=1)
```

---

betar *GAM beta regression family*

Description

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

Usage

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

Arguments

theta	the extra parameter (ϕ 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 `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 (nat.pya@gmail.com) 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)
```

blas.thread.test *BLAS thread safety*

Description

Most BLAS implementations are thread safe, but some versions of OpenBLAS, for example, are not. This routine is a diagnostic helper function, which you will never need if you don't set `nthreads>1`, and even then are unlikely to need.

Usage

```
blas.thread.test(n=1000,nt=4)
```

Arguments

`n`	Number of iterations to run of parallel BLAS calling code.
`nt`	Number of parallel threads to use

Details

While single threaded OpenBLAS 0.2.20 was thread safe, versions 0.3.0-0.3.6 are not, and from version 0.3.7 thread safety of the single threaded OpenBLAS requires making it with the option `USE_LOCKING=1`. The reference BLAS is thread safe, as are MKL and ATLAS. This routine repeatedly calls the BLAS from multi-threaded code and is sufficient to detect the problem in single threaded OpenBLAS 0.3.x.

A multi-threaded BLAS is often no faster than a single-threaded BLAS, while judicious use of threading in the code calling the BLAS can still deliver a modest speed improvement. For this reason it is often better to use a single threaded BLAS and the `nthreads` options to `bam` or `gam`. For `bam(...,discrete=TRUE)` using several threads can be a substantial benefit, especially with the reference BLAS.

The MKL BLAS is mutlithreaded by default. Under linux setting environment variable `MKL_NUM_THREADS=1` before starting R gives single threaded operation.

Author(s)

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

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 guidelines.

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.
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>

choldrop *Deletion and rank one Cholesky factor update*

Description

Given a Cholesky factor, R, of a matrix, A, choldrop finds the Cholesky factor of A[-k,-k], where k is an integer. cholup finds the factor of $A + uu^T$ (update) or $A - uu^T$ (downdate).

Usage

```
choldrop(R,k)
cholup(R,u,up)
```

Arguments

R	Cholesky factor of a matrix, A.
k	row and column of A to drop.
u	vector defining rank one update.
up	if TRUE compute update, otherwise downdate.

Details

First consider `choldrop`. If `R` is upper triangular then `t(R[,-k])%*%R[,-k] == A[-k,-k]`, but `R[,-k]` has elements on the first sub-diagonal, from its kth column onwards. To get from this to a triangular Cholesky factor of `A[-k,-k]` we can apply a sequence of Givens rotations from the left to eliminate the sub-diagonal elements. The routine does this. If `R` is a lower triangular factor then Givens rotations from the right are needed to remove the extra elements. If `n` is the dimension of `R` then the update has $O(n^2)$ computational cost.

`cholup` (which assumes `R` is upper triangular) updates based on the observation that $R^TR + uu^T = [u, R^T][u, R^T]^T = [u, R^T]Q^TQ[u, R^T]^T$, and therefore we can construct Q so that $Q[u, R^T]^T = [0, R_1^T]^T$, where R_1 is the modified factor. Q is constructed from a sequence of Givens rotations in order to zero the elements of u. Downdating is similar except that hyperbolic rotations have to be used in place of Givens rotations — see Golub and van Loan (2013, section 6.5.4) for details. Downdating only works if $A - uu^T$ is positive definite. Again the computational cost is $O(n^2)$.

Note that the updates are vector oriented, and are hence not susceptible to speed up by use of an optimized BLAS. The updates are set up to be relatively Cache friendly, in that in the upper triangular case successive Givens rotations are stored for sequential application column-wise, rather than being applied row-wise as soon as they are computed. Even so, the upper triangular update is slightly slower than the lower triangular update.

Author(s)

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

References

Golub GH and CF Van Loan (2013) Matrix Computations (4th edition) Johns Hopkins

Examples

```
require(mgcv)
set.seed(0)
n <- 6
A <- crossprod(matrix(runif(n*n),n,n))
R0 <- chol(A)
k <- 3
Rd <- choldrop(R0,k)
range(Rd-chol(A[-k,-k]))
Rd;chol(A[-k,-k])

## same but using lower triangular factor A = LL'
L <- t(R0)
Ld <- choldrop(L,k)
range(Ld-t(chol(A[-k,-k])))
Ld;t(chol(A[-k,-k]))

## Rank one update example
u <- runif(n)
R <- cholup(R0,u,TRUE)
Ru <- chol(A+u %*% t(u)) ## direct for comparison
R;Ru
```

```
range(R-Ru)

## Downdate - just going back from R to R0
Rd <-  cholup(R,u,FALSE)
R0;Rd
range(R0-Rd)
```

---

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, 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. (2017) Generalized Additive Models: An Introduction with R (2nd edition). CRC/Taylor & Francis.

Examples

```
## Simulate some data ....
library(mgcv)
set.seed(1)
dat <- gamSim(1,n=400,scale=2)

## fit a GAM with quite low `k'
b<-gam(y~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)
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 <- gam(y~s(x0,k=6)+s(x1,k=6)+s(x2,k=20)+s(x3,k=6),data=dat)
gam.check(b) ## better

## similar example with multi-dimensional smooth
b1 <- gam(y~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~s(x0)+te(x1,x2,k=4)+s(x3),data=dat)
rsd <- residuals(b2)
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~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")
bp<-gam(y~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)
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

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)
## fit a GAM with quite low `k'
b <- gam(y~s(x0,k=6)+s(x1,k=6)+s(x2,k=6)+s(x3,k=6),
         data=dat,method="REML")
b
## edf for 3rd smooth is highest as proportion of k -- increase k
b <- gam(y~s(x0,k=6)+s(x1,k=6)+s(x2,k=12)+s(x3,k=6),
         data=dat,method="REML")
b
## edf substantially up, -ve REML substantially down
b <- gam(y~s(x0,k=6)+s(x1,k=6)+s(x2,k=24)+s(x3,k=6),
         data=dat,method="REML")
b
## slight edf increase and -ve REML change
b <- gam(y~s(x0,k=6)+s(x1,k=6)+s(x2,k=40)+s(x3,k=6),
         data=dat,method="REML")
b
## defintely stabilized (but really k around 20 would have been fine)
```

---

clog *GAM censored logistic distribution family for log-logistic AFT models*

Description

Family for use with gam or bam, implementing regression for censored logistic data. If y is the response with mean μ and scale parameter $s = w^{-1/2}\exp(\theta)$, then y has p.d.f.

$$\frac{\exp\{-(y-\mu)/s\}}{s[1+\exp\{-(y-\mu)/s\}]^2}.$$

θ is a single scalar for all observations. Observations may be left, interval or right censored or uncensored.

Useful for log-logistic accelerated failure time (AFT) models, for example.

Usage

```
clog(theta=NULL,link="identity")
```

Arguments

theta	log scale parameter. If supplied and positive then taken as a fixed value of standard deviation (not its log). If supplied and negative taken as negative of initial value for standard deviation (not its log).
link	The link function: "identity", "log" or "sqrt".

Details

If the family is used with a vector response, then it is assumed that there is no censoring, and a regular Gaussian regression results. If there is censoring then the response should be supplied as a two column matrix. The first column is always numeric. Entries in the second column are as follows.

- If an entry is identical to the corresponding first column entry, then it is an uncensored observation.
- If an entry is numeric and different to the first column entry then there is interval censoring. The first column entry is the lower interval limit and the second column entry is the upper interval limit. y is only known to be between these limits.
- If the second column entry is -Inf then the observation is left censored at the value of the entry in the first column. It is only known that y is less than or equal to the first column value.
- If the second column entry is Inf then the observation is right censored at the value of the entry in the first column. It is only known that y is greater than or equal to the first column value.

Any mixture of censored and uncensored data is allowed, but be aware that data consisting only of right and/or left censored data contain very little information.

Value

An object of class extended.family.

Author(s)

Chris Shen

References

Wood, S.N., N. Pya and B. Saefken (2016), Smoothing parameter and model selection for general smooth models. Journal of the American Statistical Association 111, 1548-1575 doi:10.1080/01621459.2016.1180986

Examples

```
library(mgcv)

#######################################################
## AFT model example for colon cancer survivial data...
#######################################################

library(survival) ## for data
col1 <- colon[colon$etype==1,] ## concentrate on single event
col1$differ <- as.factor(col1$differ)
col1$sex <- as.factor(col1$sex)

## set up the AFT response...
logt <- cbind(log(col1$time),log(col1$time))
logt[col1$status==0,2] <- Inf ## right censoring
col1$logt <- -logt ## -ve conventional for AFT versus Cox PH comparison

## fit the model...
b <- gam(logt~s(age,by=sex)+sex+s(nodes)+perfor+rx+obstruct+adhere,
         family=clog(),data=col1)
plot(b,pages=1)
## ... compare this to ?cox.ph
```

---

cnorm *GAM censored normal family for log-normal AFT and Tobit models*

Description

Family for use with gam or bam, implementing regression for censored normal data. If y is the response with mean μ and standard deviation $w^{-1/2}\exp(\theta)$, then $w^{1/2}(y-\mu)\exp(-\theta)$ follows an $N(0,1)$ distribution. That is

$$y \sim N(\mu, e^{2\theta}w^{-1}).$$

θ is a single scalar for all observations. Observations may be left, interval or right censored or uncensored.

Useful for log-normal accelerated failure time (AFT) models, Tobit regression, and crudely rounded data, for example.

Usage

```
cnorm(theta=NULL,link="identity")
```

Arguments

`theta`	log standard deviation parameter. If supplied and positive then taken as a fixed value of standard deviation (not its log). If supplied and negative taken as negative of initial value for standard deviation (not its log).
`link`	The link function: `"identity"`, `"log"` or `"sqrt"`.

Details

If the family is used with a vector response, then it is assumed that there is no censoring, and a regular Gaussian regression results. If there is censoring then the response should be supplied as a two column matrix. The first column is always numeric. Entries in the second column are as follows.

- If an entry is identical to the corresponding first column entry, then it is an uncensored observation.
- If an entry is numeric and different to the first column entry then there is interval censoring. The first column entry is the lower interval limit and the second column entry is the upper interval limit. y is only known to be between these limits.
- If the second column entry is `-Inf` then the observation is left censored at the value of the entry in the first column. It is only known that y is less than or equal to the first column value.
- If the second column entry is `Inf` then the observation is right censored at the value of the entry in the first column. It is only known that y is greater than or equal to the first column value.

Any mixture of censored and uncensored data is allowed, but be aware that data consisting only of right and/or left censored data contain very little information.

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 (2016), Smoothing parameter and model selection for general smooth models. Journal of the American Statistical Association 111, 1548-1575 doi:10.1080/01621459.2016.1180986

Examples

```
library(mgcv)

#######################################################
## AFT model example for colon cancer survivial data...
#######################################################
```

```
library(survival) ## for data
col1 <- colon[colon$etype==1,] ## concentrate on single event
col1$differ <- as.factor(col1$differ)
col1$sex <- as.factor(col1$sex)

## set up the AFT response...
logt <- cbind(log(col1$time),log(col1$time))
logt[col1$status==0,2] <- Inf ## right censoring
col1$logt <- -logt ## -ve conventional for AFT versus Cox PH comparison

## fit the model...
b <- gam(logt~s(age,by=sex)+sex+s(nodes)+perfor+rx+obstruct+adhere,
         family=cnorm(),data=col1)
plot(b,pages=1)
## ... compare this to ?cox.ph

################################
## A Tobit regression example...
################################

set.seed(3);n<-400
dat <- gamSim(1,n=n)
ys <- dat$y - 5 ## shift data down

## truncate at zero, and set up response indicating this has happened...
y <- cbind(ys,ys)
y[ys<0,2] <- -Inf
y[ys<0,1] <- 0
dat$yt <- y
b <- gam(yt~s(x0)+s(x1)+s(x2)+s(x3),family=cnorm,data=dat)
plot(b,pages=1)

#############################
## A model for rounded data...
#############################

dat <- gamSim(1,n=n)
y <- round(dat$y)
y <- cbind(y-.5,y+.5) ## set up to indicate interval censoring
dat$yi <- y
b <- gam(yi~s(x0)+s(x1)+s(x2)+s(x3),family=cnorm,data=dat)
plot(b,pages=1)
```

---

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.

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.

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>

Examples

```
library(mgcv)
## simulate data with concurvity...
set.seed(8);n<- 200
f2 <- 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 <- 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 <- gam(y ~ 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 — *Additive Cox Proportional Hazard Model*

Description

The cox.ph family implements the Cox Proportional Hazards model with Peto's correction for ties, optional stratification, and estimation by penalized partial likelihood maximization, for use with gam. In the model formula, event time is the response. Under stratification the response has two columns: time and a numeric index for stratum. The weights vector provides the censoring information (0 for censoring, 1 for event). cox.ph deals with the case in which each subject has one event/censoring time and one row of covariate values. When each subject has several time dependent covariates see cox.pht.

See example below for conditional logistic regression.

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.

Stratification is possible, allowing for different baseline hazards in different strata. In that case the response has two columns: the first is event/censoring time and the second is a numeric stratum index. See below for an example.

Prediction from the fitted model object (using the predict method) with type="response" will predict on the survivor function scale. This requires evaluation times to be provided as well as covariates (see example). Also see example code below for extracting the cumulative baseline hazard/survival directly. The fitted.values stored in the model object are survival function estimates for each subject at their event/censoring time.

deviance,martingale,score, or schoenfeld residuals can be extracted. See Klein amd Moeschberger (2003) for descriptions. The score residuals are returned as a matrix of the same dimension as the model matrix, with a "terms" attribute, which is a list indicating which model matrix columns belong to which model terms. The score residuals are scaled. For parameteric terms this is by the standard deviation of associated model coefficient. For smooth terms the sub matrix of score residuals for the term is postmultiplied by the transposed Cholesky factor of the covariance matrix for the term's coefficients. This is a transformation that makes the coefficients approximately independent, as required to make plots of the score residuals against event time interpretable for checking the proportional hazards assumption (see Klein amd Moeschberger, 2003, p376). Penalization causes drift in the score residuals, which is also removed, to allow the residuals to be approximately interpreted as unpenalized score residuals. Schoenfeld and score residuals are computed by strata. See the examples for simple PH assuption checks by plotting score residuals, and Klein amd Moeschberger (2003, section 11.4) for details. Note that high correlation between terms can undermine these checks.

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 cumulative baseline hazard, survival function and associated standard errors (the survival standard error expression has a typo).

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.

This family deals efficiently with the case in which each subject has one event/censoring time and one row of covariate values. For studies in which there are multiple time varying covariate measures

for each subject then the equivalent Poisson model should be fitted to suitable pseudodata using `bam(...,discrete=TRUE)`. See `cox.pht`.

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 (2016), Smoothing parameter and model selection for general smooth models. Journal of the American Statistical Association 111, 1548-1575 doi:10.1080/01621459.2016.1180986

See Also

`cox.pht`, `cnorm`

Examples

```
library(mgcv)
library(survival) ## for data
col1 <- colon[colon$etype==1,] ## concentrate on single event
col1$differ <- as.factor(col1$differ)
col1$sex <- as.factor(col1$sex)

b <- gam(time~s(age,by=sex)+sex+s(nodes)+perfor+rx+obstruct+adhere,
         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))
dname <- names(col1)
for (n in dname) newd[[n]] <- rep(col1[[n]][j],np)
newd$time <- seq(0,3000,length=np)
fv <- predict(b,newdata=newd,type="response",se=TRUE)
plot(newd$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="l",ylim=c(0,1),
     xlab="time",ylab="survival")
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$h - 2*b$family$data$q^.5),col=2)
lines(b$family$data$tr,exp(-b$family$data$km),lty=2) ## Kaplan Meier

## Checking the proportional hazards assumption via scaled score plots as
## in Klein and Moeschberger Section 11.4 p374-376...

ph.resid <- function(b,stratum=1) {
## convenience function to plot scaled score residuals against time,
## by term. Reference lines at 5% exceedance prob for Brownian bridge
## (see KS test statistic distribution).
  rs <- residuals(b,"score");term <- attr(rs,"term")
  if (is.matrix(b$y)) {
    ii <- b$y[,2] == stratum;b$y <- b$y[ii,1];rs <- rs[ii,]
  }
  oy <- order(b$y)
  for (i in 1:length(term)) {
    ii <- term[[i]]; m <- length(ii)
    plot(b$y[oy],rs[oy,ii[1]],ylim=c(-3,3),type="l",ylab="score residuals",
         xlab="time",main=names(term)[i])
    if (m>1) for (k in 2:m) lines(b$y[oy],rs[oy,ii[k]],col=k);
    abline(-1.3581,0,lty=2);abline(1.3581,0,lty=2)
  }
}
par(mfrow=c(2,2))
ph.resid(b)

## stratification example, with 2 randomly allocated strata
## so that results should be similar to previous....
col1$strata <- sample(1:2,nrow(col1),replace=TRUE)
bs <- gam(cbind(time,strata)~s(age,by=sex)+sex+s(nodes)+perfor+rx+obstruct
          +adhere,family=cox.ph(),data=col1,weights=status)
plot(bs,pages=1,all.terms=TRUE) ## plot effects

## baseline survival plots by strata...

for (i in 1:2) { ## loop over strata
## create index selecting elements of stored hazard info for stratum i...
ind <- which(bs$family$data$tr.strat == i)
if (i==1) plot(bs$family$data$tr[ind],exp(-bs$family$data$h[ind]),type="l",
      ylim=c(0,1),xlab="time",ylab="survival",lwd=2,col=i) else
      lines(bs$family$data$tr[ind],exp(-bs$family$data$h[ind]),lwd=2,col=i)
lines(bs$family$data$tr[ind],exp(-bs$family$data$h[ind] +
      2*bs$family$data$q[ind]^.5),lty=2,col=i) ## upper ci
lines(bs$family$data$tr[ind],exp(-bs$family$data$h[ind] -
      2*bs$family$data$q[ind]^.5),lty=2,col=i) ## lower ci
lines(bs$family$data$tr[ind],exp(-bs$family$data$km[ind]),col=i) ## KM
}


## Simple simulated known truth example...
```

```
ph.weibull.sim <- function(eta,gamma=1,h0=.01,t1=100) {
  lambda <- h0*exp(eta)
  n <- length(eta)
  U <- runif(n)
  t <- (-log(U)/lambda)^(1/gamma)
  d <- as.numeric(t <= t1)
  t[!d] <- t1
  list(t=t,d=d)
}
n <- 500;set.seed(2)
x0 <- runif(n, 0, 1);x1 <- runif(n, 0, 1)
x2 <- runif(n, 0, 1);x3 <- runif(n, 0, 1)
f0 <- function(x) 2 * sin(pi * x)
f1 <- function(x) exp(2 * x)
f2 <- function(x) 0.2*x^11*(10*(1-x))^6+10*(10*x)^3*(1-x)^10
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)

## Another one, including a violation of proportional hazards for
## effect of x2...

set.seed(2)
h <- exp((f0(x0)+f1(x1)+f2(x2)-10)/5)
t <- rexp(n,h);d <- as.numeric(t<20)

## first with no violation of PH in the simulation...
b <- gam(t~s(x0)+s(x1)+s(x2)+s(x3),family=cox.ph,weights=d)
plot(b,pages=1)
ph.resid(b) ## fine

## Now violate PH for x2 in the simulation...
ii <- t>1.5
h1 <- exp((f0(x0)+f1(x1)+3*f2(x2)-10)/5)
t[ii] <- 1.5 + rexp(sum(ii),h1[ii]);d <- as.numeric(t<20)

b <- gam(t~s(x0)+s(x1)+s(x2)+s(x3),family=cox.ph,weights=d)
plot(b,pages=1)
ph.resid(b) ## The checking plot picks up the problem in s(x2)


## conditional logistic regression models are often estimated using the
## cox proportional hazards partial likelihood with a strata for each
## case-control group. A dummy vector of times is created (all equal).
## The following compares to 'clogit' for a simple case. Note that
## the gam log likelihood is not exact if there is more than one case
## per stratum, corresponding to clogit's approximate method.
```

```
library(survival);library(mgcv)
infert$dumt <- rep(1,nrow(infert))
mg <- gam(cbind(dumt,stratum) ~ spontaneous + induced, data=infert,
          family=cox.ph,weights=case)
ms <- clogit(case ~ spontaneous + induced + strata(stratum), data=infert,
             method="approximate")
summary(mg)$p.table[1:2,]; ms
```

`cox.pht` *Additive Cox proportional hazard models with time varying covariates*

Description

The `cox.ph` family only allows one set of covariate values per subject. If each subject has several time varying covariate measurements then it is still possible to fit a proportional hazards regression model, via an equivalent Poisson model. The recipe is provided by Whitehead (1980) and is equally valid in the smooth additive case. Its drawback is that the equivalent Poisson dataset can be quite large.

The trick is to generate an artificial Poisson observation for each subject in the risk set at each non-censored event time. The corresponding covariate values for each subject are whatever they are at the event time, while the Poisson response is zero for all subjects except those experiencing the event at that time (this corresponds to Peto's correction for ties). The linear predictor for the model must include an intercept for each event time (the cumulative sum of the exponential of these is the Breslow estimate of the baseline hazard).

Below is some example code employing this trick for the `pbcseq` data from the `survival` package. It uses `bam` for fitting with the `discrete=TRUE` option for efficiency: there is some approximation involved in doing this, and the exact equivalent to what is done in `cox.ph` is rather obtained by using `gam` with `method="REML"` (taking many times the computational time for the example below). An alternative fits the model as a conditional logistic model using stratified Cox PH with event times as strata (see example). This would be identical in the unpenalized case, but smoothing parameter estimates can differ.

The function `tdpois` in the example code uses crude piecewise constant interpolation for the covariates, in which the covariate value at an event time is taken to be whatever it was the previous time that it was measured. Obviously more sophisticated interpolation schemes might be preferable.

References

Whitehead (1980) Fitting Cox's regression model to survival data using GLIM. Applied Statistics 29(3):268-275

Examples

```
require(mgcv);require(survival)

## First define functions for producing Poisson model data frame

app <- function(x,t,to) {
```

```
## wrapper to approx for calling from apply...
  y <- if (sum(!is.na(x))<1) rep(NA,length(to)) else
       approx(t,x,to,method="constant",rule=2)$y
  if (is.factor(x)) factor(levels(x)[y],levels=levels(x)) else y
} ## app

tdpois <- function(dat,event="z",et="futime",t="day",status="status1",
                                                              id="id") {
## dat is data frame. id is patient id; et is event time; t is
## observation time; status is 1 for death 0 otherwise;
## event is name for Poisson response.
  if (event %in% names(dat)) warning("event name in use")
  require(utils) ## for progress bar
  te <- sort(unique(dat[[et]][dat[[status]]==1])) ## event times
  sid <- unique(dat[[id]])
  inter <- interactive()
  if (inter) prg <- txtProgressBar(min = 0, max = length(sid), initial = 0,
         char = "=",width = NA, title="Progress", style = 3)
  ## create dataframe for poisson model data
  dat[[event]] <- 0; start <- 1
  dap <- dat[rep(1:length(sid),length(te)),]
  for (i in 1:length(sid)) { ## work through patients
    di <- dat[dat[[id]]==sid[i],] ## ith patient's data
    tr <- te[te <= di[[et]][1]] ## times required for this patient
    ## Now do the interpolation of covariates to event times...
    um <- data.frame(lapply(X=di,FUN=app,t=di[[t]],to=tr))
    ## Mark the actual event...
    if (um[[et]][1]==max(tr)&&um[[status]][1]==1) um[[event]][nrow(um)] <- 1
    um[[et]] <- tr ## reset time to relevant event times
    dap[start:(start-1+nrow(um)),] <- um ## copy to dap
    start <- start + nrow(um)
    if (inter) setTxtProgressBar(prg, i)
  }
  if (inter) close(prg)
  dap[1:(start-1),]
} ## tdpois

## The following typically takes a minute or less...


## Convert pbcseq to equivalent Poisson form...
pbcseq$status1 <- as.numeric(pbcseq$status==2) ## death indicator
pb <- tdpois(pbcseq) ## conversion
pb$tf <- factor(pb$futime) ## add factor for event time

## Fit Poisson model...
b <- bam(z ~ tf - 1 + sex + trt + s(sqrt(protime)) + s(platelet)+ s(age)+
s(bili)+s(albumin), family=poisson,data=pb,discrete=TRUE,nthreads=2)

pb$dumt <- rep(1,nrow(pb)) ## dummy time
## Fit as conditional logistic...
b1 <- gam(cbind(dumt,tf) ~ sex + trt + s(sqrt(protime)) + s(platelet)
+ s(age) + s(bili) + s(albumin),family=cox.ph,data=pb,weights=z)
```

```
par(mfrow=c(2,3))
plot(b,scale=0)

## compute residuals...
chaz <- tapply(fitted(b),pb$id,sum) ## cum haz by subject
d <- tapply(pb$z,pb$id,sum) ## censoring indicator
mrsd <- d - chaz ## Martingale
drsd <- sign(mrsd)*sqrt(-2*(mrsd + d*log(chaz))) ## deviance

## plot survivor function and s.e. band for subject 25
te <- sort(unique(pb$futime)) ## event times
di <- pbcseq[pbcseq$id==25,] ## data for subject 25
pd <- data.frame(lapply(X=di,FUN=app,t=di$day,to=te)) ## interpolate to te
pd$tf <- factor(te)
X <- predict(b,newdata=pd,type="lpmatrix")
eta <- drop(X%*%coef(b)); H <- cumsum(exp(eta))
J <- apply(exp(eta)*X,2,cumsum)
se <- diag(J%*%vcov(b)%*%t(J))^.5
plot(stepfun(te,c(1,exp(-H))),do.points=FALSE,ylim=c(0.7,1),
     ylab="S(t)",xlab="t (days)",main="",lwd=2)
lines(stepfun(te,c(1,exp(-H+se))),do.points=FALSE)
lines(stepfun(te,c(1,exp(-H-se))),do.points=FALSE)
rug(pbcseq$day[pbcseq$id==25]) ## measurement times
```

`cpois` *GAM censored Poisson family*

Description

Family for use with `gam` or `bam`, implementing regression for censored Poisson data. Observations may be left, interval or right censored or uncensored.

Usage

```
cpois(link="log")
```

Arguments

`link`	The link function: `"identity"`, `"log"` or `"sqrt"`.

Details

If the family is used with a vector response, then it is assumed that there is no censoring, and a regular Poisson regression results. If there is censoring then the response should be supplied as a two column matrix. The first column is always numeric. Entries in the second column are as follows.

- If an entry is identical to the corresponding first column entry, then it is an uncensored observation.
- If an entry is numeric and different to the first column entry then there is interval censoring. The first column entry is the lower interval limit and the second column entry is the upper interval limit (both should be non-integer). y is only known to be between these limits.
- If the second column entry is `-Inf` then the observation is left censored at the value of the entry (non-integer) in the first column. It is only known that y is below the first column value.
- If the second column entry is `Inf` then the observation is right censored at the value (non-integer) of the entry in the first column. It is only known that y is above the first column value.

Any mixture of censored and uncensored data is allowed, but be aware that data consisting only of right and/or left censored data contain very little information. It is strongly recommended to use non-integer values for censoring limits, to avoid any possibility of ambiguity. For example if y is known to be 3 or above, set the lower censoring limit to 2.5, or any other value in $(2, 3)$.

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 (2016), Smoothing parameter and model selection for general smooth models. Journal of the American Statistical Association 111, 1548-1575 doi:10.1080/01621459.2016.1180986

Examples

```
library(mgcv)
set.seed(6); n <- 2000
dat <- gamSim(1,n=n,dist="poisson",scale=.1) ## simulate Poi data

## illustration that cpois an poisson give same results if there
## is no censoring...

b0 <- gam(y~s(x0,bs="cr")+s(x1,bs="cr")+s(x2,bs="cr")+
             s(x3,bs="cr"),family=poisson,data=dat,method="REML")
plot(b0,pages=1,scheme=2)

b1 <- gam(y~s(x0,bs="cr")+s(x1,bs="cr")+s(x2,bs="cr")+
            s(x3,bs="cr"),family=cpois,data=dat)
plot(b1,pages=1,scheme=2)

b0;b1

## Now censor some observations...
```

```
dat1 <- dat
m <- 300
y <- matrix(dat$y,n,ncol=2) ## response matrix
ii <- sample(n,3*m) ## censor these
for (i in 1:m) { ## right, left, interval...
  j <- ii[i]; if (y[j,1] > 4.5) y[j,] <- c(4.5,Inf)
  j <- ii[m+i]; if (y[j,1] < 2.5) y[j,] <- c(2.5,-Inf)
  j <- ii[2*m+i];if (y[j,1] > 1.5 & y[j,1]< 5.5) y[j,] <- c(1.5,5.5)
}
n - sum(y[,1]==y[,2]) ## number of censored obs
dat1$y <- y

## now fit model with censoring...
b2 <- gam(y~s(x0,bs="cr")+s(x1,bs="cr")+s(x2,bs="cr")+
          s(x3,bs="cr"),family=cpois,data=dat1)
plot(b2,pages=1,scheme=2);b2
```

`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, derivs=0)
```

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.
`derivs`	order of derivative of the spline to evaluate, between 0 and `ord`-1. Recycled to length of `x`.

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

Examples

```
require(mgcv)
## create some x's and knots...
n <- 200
x <- 0:(n-1)/(n-1);k<- 0:5/5
X <- 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 <- sort(runif(n)) + 1 ## sorting just makes end checking easy
k <- 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!")
```

---

dDeta *Obtaining derivative w.r.t. linear predictor*

Description

INTERNAL function. Distribution families provide derivatives of the deviance and link w.r.t. `mu = inv_link(eta)`. This routine converts these to the required derivatives of the deviance w.r.t. eta, the linear predictor.

Usage

```
dDeta(y, mu, wt, theta, fam, deriv = 0)
```

Arguments

`y`	vector of observations.
`mu`	if `eta` is the linear predictor, `mu = inv_link(eta)`. In a traditional GAM `mu=E(y)`.
`wt`	vector of weights.
`theta`	vector of family parameters that are not regression coefficients (e.g. scale parameters).

`fam`	the family object.
`deriv`	the order of derivative of the smoothing parameter score required.

Value

A list of derivatives.

Author(s)

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

`dpnorm` *Stable evaluation of difference between normal c.d.f.s*

Description

Evaluates the difference between two $N(0,1)$ cumulative distribution functions avoiding cancellation error.

Usage

```
dpnorm(x0,x1,log.p=FALSE)
```

Arguments

`x0`	vector of lower values at which to evaluate standard normal distribution function.
`x1`	vector of upper values at which to evaluate standard normal distribution function.
`log.p`	set to `TRUE` to compute on log scale, avoiding underflow.

Details

Equivalent to `pnorm(x1)-pnorm(x0)`, but stable when `x0` and `x1` values are very close, or in the upper tail of the standard normal.

Author(s)

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

Examples

```
require(mgcv)
x <- seq(-10,10,length=10000)
eps <- 1e-10
y0 <- pnorm(x+eps)-pnorm(x) ## cancellation prone
y1 <- dpnorm(x,x+eps)       ## stable
## illustrate stable computation in black, and
## cancellation prone in red...
par(mfrow=c(1,2),mar=c(4,4,1,1))
plot(log(y1),log(y0),type="l")
lines(log(y1[x>0]),log(y0[x>0]),col=2)
plot(x,log(y1),type="l")
lines(x,log(y0),col=2)
```

`dppois` *Stable evaluation of difference between Poisson c.d.f.s*

Description

Evaluates the difference between two Poisson cumulative distribution functions avoiding cancellation error.

Usage

```
dppois(y0,y1,mu,log.p=TRUE)
```

Arguments

`y0`	vector of lower values at which to evaluate Poisson distribution function.
`y1`	vector of upper values at which to evaluate Poisson distribution function.
`mu`	vector giving Poisson means
`log.p`	set to `FALSE` to compute on probability scale.

Details

Equivalent to `log(ppois(y1,mu)-ppois(y0,mu))`, but stable when `y0` and `y1` are highly improbable and in the same tail of the distribution (avoids cancellation to `log(0)`).

Author(s)

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

Examples

```
require(mgcv)
n <- 10
mu <- rep(2,n)
y0 <- c(0:5,10,23,0,2)
y1 <- c(1:6,11,1000,100,40)
dppois(y0,y1,mu)
log(ppois(y1,mu)-ppois(y0,mu))
```

`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.
`dist`	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.

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>

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)
```

---

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 lme object, allowing the user control about which levels of random effects to include in this calculation. extract.lme.cov forms the full matrix explicitly: extract.lme.cov2 tries to be more economical than this.

Usage

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

Arguments

b	A fitted model object returned by a call to lme
.	
data	The data frame/ model frame that was supplied to lme, but with any rows removed by the na action dropped. Uses the data stored in the model object if not supplied.
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.lme.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 `lme` 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 `lme` 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. (2017) Generalized Additive Models: An Introduction with R (2nd edition). Chapman and Hall/CRC Press.

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)
extract.lme.cov2(b)
```

`factor.smooth` *Factor smooth interactions in GAMs*

Description

The interaction of one or more factors with a smooth effect, produces a separate smooth for each factor level. These smooths can have different smoothing parameters, or all have the same smoothing parameter. There are several vays to set them up.

Factor `by` **variables.** If the `by` variables for a smooth (specified using `s`, `te`, `ti` or `t2`) is a factor, then a separate smooth is produced for each factor level. If the factor is ordered, then no smooth is produced for its first level: this is useful for setting up models which have a reference level smooth and then difference to reference smooths for each factor level except the first (which is the reference). Giving the smooth an `id` forces the same smoothing parameter to be used for all levels of the factor. For example `s(x,by=fac,id=1)` would produce a separate smooth of `x` for each level of `fac`, with each smooth having the same smoothing parameter. See gam.models for more.

Sum to zero smooth interactions `bs="sz"` These factor smooth interactions are specified using `s(...,bs="sz")`. There may be several factors supplied, and a smooth is produced for each combination of factor levels. The smooths are constructed to exclude the 'main effect' smooth, or the effects of individual smooths produced for lower order combinations of factor levels. For example, with a single factor, the smooths for the different factor levels are so constrained that the sum over all factor levels of equivalent spline coefficients are all zero. This allows the meaningful and identifiable construction of models with a main effect smooth plus smooths for the difference between each factor level and the main effect. Such a construction is often more natural than the `by` variable with ordered factors construction. See `smooth.construct.sz.smooth.spec`.

Random wiggly curves `bs="fs"` This approach produces a smooth curve for each level of a single factor, treating the curves as entirely random. This means that in principle a model can be constructed with a main effect plus factor level smooth deviations from that effect. However the model is not forced to make the main effect do as much of the work as possible, in the way that the `"sz"` approach does. This approach can be very efficient with `gamm` as it exploits the nested estimation available in `lme`. See `smooth.construct.fs.smooth.spec`.

Author(s)

Simon N. Wood <`simon.wood@r-project.org`> with input from Matteo Fasiolo.

See Also

smooth.construct.fs.smooth.spec, smooth.construct.sz.smooth.spec

Examples

```
library(mgcv)
set.seed(0)
## simulate data...
f0 <- function(x) 2 * sin(pi * x)
f1 <- function(x,a=2,b=-1) exp(a * x)+b
f2 <- 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)
x0 <- runif(n);x1 <- runif(n);x2 <- runif(n)
a <- rnorm(nf)*.2 + 2;b <- rnorm(nf)*.5
f <- f0(x0) + f1(x1,a[fac],b[fac]) + f2(x2)
fac <- factor(fac)
y <- 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...
bm <- gamm(y~s(x0)+ s(x1,fac,bs="fs",k=5)+s(x2,k=20))
plot(bm$gam,pages=1)
summary(bm$gam)

bd <- bam(y~s(x0)+ s(x1) + s(x1,fac,bs="sz",k=5)+s(x2,k=20),discrete=TRUE)
plot(bd,pages=1)
summary(bd)



## Could also use...
## b <- gam(y~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~s(x0)+ t2(x1,fac,bs=c("tp","re"),k=5,full=TRUE)+
##        s(x2,k=20),method="ML"))
## ... is exactly equivalent.
```

---

family.mgcv *Distribution families in mgcv*

Description

As well as the standard families (of class family) 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, although some can also be used with bam. These are described here.

Details

The following families (class family) 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. Alternatively, see tw to estimate p (gam/bam only).
- negbin The negative binomial. Alternatively see nb to estimate the theta parameter of the negative binomial (gam/bam only).

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

- betar for proportions data on (0,1) when the binomial is not appropriate.
- cnorm censored normal distribution, for log normal accelerated failure time models, Tobit regression and rounded data, for example.
- bcg Box-Cox transformed censored Gaussian.
- clog censored logistic distribution, for accelerated failure time models.
- cpois censored Poisson distribution.
- nb for negative binomial data when the theta parameter is to be estimated.
- ocat for ordered categorical data.
- scat scaled t for heavy tailed data that would otherwise be modelled as Gaussian.
- tw for Tweedie distributed data, when the power parameter relating the variance to the mean is to be estimated.
- ziP for zero inflated Poisson data, when the zero inflation rate depends simply on the Poisson mean.

The above families of class family and extended.family can be combined to model data where different response observations come from different distributions. For example, when modelling the combination of presence-absence and abundance data, binomial and nb families might be used.

- gfam creates a 'grouped family' (or 'family group') from a list of families. The response is supplied as a two column matrix, the first containing the response observations, and the second an index of the family to which each observation relates.

The following families (class general.family) implement more general model classes. Usable only with gam and only with REML or NCV smoothing parameter estimation.

- cox.ph the Cox Proportional Hazards model for survival data (no NCV).
- gammals a gamma location-scale model, where the mean and standard deviation are modelled with separate linear predictors.
- gaulss a Gaussian location-scale model where the mean and the standard deviation are both modelled using smooth linear predictors.

- gevlss a generalized extreme value (GEV) model where the location, scale and shape parameters are each modelled using a linear predictor.
- gumbls a Gumbel location-scale model (2 linear predictors).
- multinom: multinomial logistic regression, for unordered categorical responses.
- mvn: multivariate normal additive models (no NCV).
- shash Sinh-arcsinh location scale and shape model family (4 linear predicors).
- twlss Tweedie location scale and variance power model family (3 linear predicors). Can only be fitted using EFS method.
- 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.

Author(s)

Simon N. Wood (s.wood@r-project.org), Natalya Pya, Matteo Fasiolo and Chris Shen.

References

Wood, S.N., N. Pya and B. Saefken (2016), Smoothing parameter and model selection for general smooth models. Journal of the American Statistical Association 111, 1548-1575 doi:10.1080/01621459.2016.1180986

FFdes *Level 5 fractional factorial designs*

Description

Computes level 5 fractional factorial designs for up to 120 factors using the agorithm of Sanchez and Sanchez (2005), and optionally central composite designs.

Usage

```
FFdes(size=5,ccd=FALSE)
```

Arguments

size	number of factors up to 120.
ccd	if TRUE, adds points along each axis at the same distance from the origin as the points in the fractional factorial design, to create the outer points of a central composite design. Add central points to complete.

Details

Basically a translation of the code provided in the appendix of Sanchez and Sanchez (2005).

Author(s)

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

References

Sanchez, S. M. & Sanchez, P. J. (2005) Very large fractional factorial and central composite designs. ACM Transactions on Modeling and Computer Simulation. 15: 362-377

Examples

```
require(mgcv)
plot(rbind(0,FFdes(2,TRUE)),xlab="x",ylab="y",
     col=c(2,1,1,1,1,4,4,4,4),pch=19,main="CCD")
FFdes(5)
FFdes(5,TRUE)
```

---

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.

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. Similalry 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. fix.family.var also adds a variable scale set to negative to indicate that family has a free scale parameter.

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.

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 (2017, section 5.6.3) 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. (2017) Generalized Additive Models: An Introduction with R (2nd edition). 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)
```

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~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

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~s(x)+s(z)+s(x,z) or y~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~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 ~ -1 or ~ -1). See multinom for an example.

Value

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

WARNING

A gam formula should not refer to variables using e.g. dat[["x"]].

Author(s)

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

See Also

gam

---

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 $X^T V^{-1} X$ 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 $X^T V^{-1} X$.

Author(s)

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

References

For lme 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 lme see:

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

See Also

gamm, extract.lme.cov2

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)
```

`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.
`l`	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 `NA`s 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.

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)
```

---

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/NCV 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 also 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 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,
    drop.intercept=NULL,nei=NULL,discrete=FALSE,...)
```

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 `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 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 re-weight 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`).

`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 unknown scale parameter and Mallows' Cp/UBRE/AIC for known scale. `"GACV.Cp"` is equivalent, but using GACV in place of GCV. `"NCV"` for neighbourhood cross-validation using the neighbourhood structure speficied by `nei` (`"QNCV"` for numerically more ribust version). `"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 default, and the only other options are `"ML"`, `"NCV"` or `"QNCV"`.

`optimizer` An array specifying the numerical optimization method to use to optimize the smoothing parameter estimation criterion (given by `method`). `"outer"` for the direct nested optimization approach. `"outer"` can use several alternative optimizers, specified in the second element of `optimizer`: `"newton"` (default), `"bfgs"`, `"optim"` or `"nlm"`. `"efs"` for the extended Fellner Schall method of Wood and Fasiolo (2017).

`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. Use `gamma` to increase level of penalization.

`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).

`H` 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` Increase this beyond 1 to produce smoother models. `gamma` multiplies the effective degrees of freedom in the GCV or UBRE/AIC. `n/gamma` can be viewed as an effective sample size in the GCV score, and this also enables it to be used with REML/ML. Ignored with P-RE/ML or the `efs` optimizer.

`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 sp, 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.

drop.intercept — Set to TRUE to force the model to really not have a constant in the parametric model part, even with factor variables present. Can be vector when formula is a list.

nei — A list specifying the neighbourhood structure for NCV. a is the vector of indices to be dropped for each neighbourhood and ma gives the end of each neighbourhood. So nei$a[(nei$ma[j-1]+1):nei$ma[j]] gives the points dropped for the neighbourhood j. d is the vector of indices of points to predict, with corresponding endpoints md. So nei$d[(nei$md[j-1]+1):nei$md[j]] indexes the points to predict for neighbourhood j. If nei==NULL (or a or ma are missing) then leave-one-out cross validation is obtained. If jackknife is supplied then TRUE indicates to use raw jackknife covariances estimator and FALSE to use the conventional Bayes estimate. If not supplied then the estimator accounting for neighbourhood structure is used. jackknife ignored when method is not NCV.

discrete — experimental option for setting up models for use with discrete methods employed in bam. Do not modify.

... — further arguments for passing on e.g. to gam.fit3 (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:

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

where the (independent) response variables $y_i \sim \mathrm{Poi}$, 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. Note that to be identifiable the model requires constraints on the smooth functions. By default these are imposed automatically and require that the function sums to zero over the observed covariate values (the presence of a metric by variable is the only case which usually suppresses this).

If absolutely any smooth functions were allowed in model fitting then maximum likelihood estimation of such models would invariably result in complex over-fitting 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 trade-off 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, NCV 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.

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. The linear identifiability constraints are also obtained at this point. The model is fit using gam.fit3 or variants, which are modifications of glm.fit. The GAM penalized likelihood maximization problem is solved by Penalized Iteratively Re-weighted Least Squares (P-IRLS) (see e.g. Wood 2000). Smoothing parameter selection is possible in one of three ways. (i) 'Performance iteration' uses the fact that at each P-IRLS step a working penalized linear model is estimated, and the smoothing parameter estimation can be performed for each such working model. Eventually, in most cases, both model parameter estimates and smoothing parameter estimates converge. This option is available in bam and gamm. (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. (iii) The extended Fellner-Schall algorithm of Wood and Fasiolo (2017) alternates estimation of model coefficients with simple updates of smoothing parameters, eventually approximately maximizing the marginal likelihood of the model (REML). gam uses the second method, outer iteration, by default.

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, 2004) and Wood, Pya and Saefken (2016). Some alternative methods are discussed in Wood (2000, 2017).

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 lo 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 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.

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. (2025) Generalized Additive Models. Annual Review of Statistics and Its Applications 12:497-526 doi:10.1146/annurevstatistics112723034249

Wood, S.N., N. Pya and B. Saefken (2016), Smoothing parameter and model selection for general smooth models (with discussion). Journal of the American Statistical Association 111, 1548-1575 doi:10.1080/01621459.2016.1180986

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 doi:10.1111/j.14679868.2010.00749.x

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 doi:10.1111/1467-9868.00374

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

Wood S.N. (2017) Generalized Additive Models: An Introduction with R (2nd edition). Chapman and Hall/CRC Press. doi:10.1201/9781315370279

Wood, S.N. and M. Fasiolo (2017) A generalized Fellner-Schall method for smoothing parameter optimization with application to Tweedie location, scale and shape models. Biometrics 73 (4), 1071-1081 doi:10.1111/biom.12666

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. doi:10.1007/s112220129314z

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. doi:10.1111/j.1467-9469.2011.00760.x

Key Reference on GAMs and related models:

Wood, S. N. (2020) Inference and computation with generalized additive models and their extensions. Test 29(2): 307-339. doi:10.1007/s11749020007115

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

https://webhomes.maths.ed.ac.uk/~swood34/

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.control gam.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)
b <- 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

## plot derivaives of smooths w.r.t. covariates - the
## direct analogue of linear model regression coefficients
## - also selecting a more colourful plotting scheme...
plot(b,pages=1,scale=0,deriv=TRUE,scheme=2)

## run some basic model checks, including checking
## smoothing basis dimensions...
gam.check(b)

## same fit in two parts .....
G <- gam(y~s(x0)+s(x1)+s(x2)+s(x3),fit=FALSE,data=dat)
b <- gam(G=G)
print(b)

## 2 part fit enabling manipulation of smoothing parameters...
G <- gam(y~s(x0)+s(x1)+s(x2)+s(x3),fit=FALSE,data=dat,sp=b$sp)
G$lsp0 <- log(b$sp*10) ## provide log of required sp vec
gam(G=G) ## it's smoother

## change the smoothness selection method to REML
b0 <- gam(y~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~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~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~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~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~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~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...
set.seed(6)
dat <- gamSim(1,n=2000,dist="poisson",scale=.1)

## use "cr" basis to save time, with 2000 data...
b2<-gam(y~s(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~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 GACV...

dat <- gamSim(1,n=400,dist="poisson",scale=.25)

b4<-gam(y~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~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)

lr.fit <- gam(y~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))
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]]]
  ind <- sort.int(xx,index.return=TRUE)$ix
  lines(xx[ind],(ff-mean(ff))[ind]*.33,col=2)
}
par(op)
anova(lr.fit)
```

```
lr.fit1 <- gam(y~s(x0)+s(x1)+s(x2),family=binomial,
               data=dat,method="REML")
lr.fit2 <- gam(y~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)
attach(eg)

op <- par(mfrow=c(2,2),mar=c(4,4,1,1))

contour(truth$x,truth$z,truth$f) ## contour truth
b4 <- gam(y~s(x,z),data=data) ## fit model
fit1 <- matrix(predict.gam(b4,pr,se=FALSE),40,40)
contour(truth$x,truth$z,fit1)   ## contour fit
persp(truth$x,truth$z,truth$f)    ## persp truth
vis.gam(b4)                     ## persp fit
detach(eg)
par(op)

##################################################
## largish dataset example with user defined knots
##################################################

par(mfrow=c(2,2))
n <- 5000
eg <- gamSim(2,n=n,scale=.5)
attach(eg)

ind<-sample(1:n,200,replace=FALSE)
b5<-gam(y~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~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 <- gam(y~s(x,z,k=40,xt=list(max.knots=500,seed=2)),data=data)
vis.gam(b7,theta=30,phi=30)
```

```
detach(eg)
```

---

gam.check *Some diagnostics for a fitted gam model*

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. Care should be taken in interpreting the results when applied to gam objects returned by gamm.

Usage

```
gam.check(b, old.style=FALSE,
          type=c("deviance","pearson","response"),
          k.sample=5000,k.rep=200,
          rep=0, level=.9, rl.col=2, rep.col="gray80", ...)
```

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 dsistribution. 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 (Wood, 2017, section 5.9) is based on computing an estimate of the residual variance based on differencing residuals that are near neighbours according to the (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.

Take care when interpreting results from applying this function to a model fitted using `gamm`. In this case the returned `gam` object is based on the working model used for estimation, and will treat all the random effects as part of the error. This means that the residuals extracted from the `gam` object are not standardized for the family used or for the random effects or correlation structure. Usually it is necessary to produce your own residual checks based on consideration of the model structure you have used.

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. (2017) Generalized Additive Models: An Introduction with R (2nd edition). Chapman and Hall/CRC Press.

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)
```

---

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

```
gam.control(nthreads=1,ncv.threads=1,irls.reg=0.0,epsilon = 1e-07,
            maxit = 200,mgcv.tol=1e-7,mgcv.half=15, trace = FALSE,
            rank.tol=.Machine$double.eps^0.5,nlm=list(),
     optim=list(),newton=list(),
     idLinksBases=TRUE,scalePenalty=TRUE,efs.lspmax=15,
     efs.tol=.1,keepData=FALSE,scale.est="fletcher",
     edge.correct=FALSE)
```

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.
ncv.threads	The computations for neighbourhood cross-validation (NCV) typically scale better than the rest of the GAM computations and are worth parallelizing. ncv.threads allows you to set the number of theads to use separately.
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.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.

`optim` list of control parameters to pass to `optim` if this is used for outer estimation of smoothing parameters (not default). See details.

`newton` list of control parameters to pass to default Newton optimizer used for outer estimation of log smoothing parameters. See details.

`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`.

`efs.lspmax` maximum log smoothing parameters to allow under extended Fellner Schall smoothing parameter optimization.

`efs.tol` change in REML to count as negligible when testing for EFS convergence. If the step is small and the last 3 steps led to a REML change smaller than this, then stop.

`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.

`scale.est` How to estimate the scale parameter for exponential family models estimated by outer iteration. See `gam.scale`.

`edge.correct` With RE/ML smoothing parameter selection in `gam` using the default Newton RE/ML optimizer, it is possible to improve inference at the 'completely smooth' edge of the smoothing parameter space, by decreasing smoothing parameters until there is a small increase in the negative RE/ML (e.g. 0.02). Set to `TRUE` or to a number representing the target increase to use. Only changes the corrected smoothing parameter matrix, `Vc`.

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.

See Also

`gam`, `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 fitting methods used by `gam` opt for certainty of convergence over speed of fit. `bam` opts for speed.

`gam` uses a nested iteration method (see `gam.outer`), in which each trial set of smoothing parameters proposed by an outer Newton algorithm require an inner Newton algorithm (penalized iteratively re-weighted least squares, PIRLS) to find the corresponding best fit model coefficients. Implicit differentiation is used to find the derivatives of the coefficients with respect to log smoothing parameters, so that the derivatives of the smoothness selection criterion can be obtained, as required

by the outer iteration. This approach is less expensive than it at first appears, since excellent starting values for the inner iteration are available as soon as the smoothing parameters start to converge. See Wood (2011) and Wood, Pya and Saefken (2016).

bam uses an alternative approach similar to 'performance iteration' or 'PQL'. A single PIRLS iteration is run to find the model coefficients. At each step this requires the estimation of a working penalized linear model. Smoothing parameter selection is applied directly to this working model at each step (as if it were a Gaussian additive model). This approach is more straightforward to code and in principle less costly than the nested approach. However it is not guaranteed to converge, since the smoothness selection criterion is changing at each iteration. It is sometimes possible for the algorithm to cycle around a small set of smoothing parameter, coefficient combinations without ever converging. bam includes some checks to limit this behaviour, and the further checks in the algorithm used by `bam(...,discrete=TRUE)` actually guarantee convergence in some cases, but in general guarantees are not possible. See Wood, Goude and Shaw (2015) and Wood et al. (2017).

gam when used with 'general' families (such as multinom or `cox.ph`) can also use a potentially faster scheme based on the extended Fellner-Schall method (Wood and Fasiolo, 2017). This also operates with a single iteration and is not guaranteed to converge, theoretically.

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) Try using bam (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.

If you have convergence 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.

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

Please report convergence problems, especially if you there is no obvious pathology in the data/model that suggests convergence should fail.

Author(s)

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

References

Key References on this implementation:

Wood, S.N., N. Pya and B. Saefken (2016), Smoothing parameter and model selection for general smooth models (with discussion). Journal of the American Statistical Association 111, 1548-1575 doi:10.1080/01621459.2016.1180986

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

Wood, S.N., Li, Z., Shaddick, G. & Augustin N.H. (2017) Generalized additive models for gigadata: modelling the UK black smoke network daily data. Journal of the American Statistical Association.

Wood, S.N. and M. Fasiolo (2017) A generalized Fellner-Schall method for smoothing parameter optimization with application to Tweedie location, scale and shape models, Biometrics.

Wood S.N. (2017) Generalized Additive Models: An Introduction with R (2nd edition). Chapman and Hall/CRC Press.

`gam.fit` *GAM P-IRLS estimation with GCV/UBRE smoothness estimation (deprecated)*

Description

This is a deprecated 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

```
gam.fit(G, start = NULL, etastart = NULL,
        mustart = NULL, family = gaussian(),
        control = gam.control(),gamma=1,
        fixedSteps=(control$maxit+1),...)
```

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.

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.

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,nei=NULL,...)
```

Arguments

x	The model matrix for the GAM (or any penalized GLM).
y	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.
Mp	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.
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.
nei	List specifying neighbourhood structure if NCV used. See gam.
...	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 and Raynor (1986) Automatic smoothing of regression functions in generalized linear models. J. Amer. Statist. Assoc. 81:96-103.

See Also

gam, magic

gam.fit5.post.proc *Post-processing output of gam.fit5*

Description

INTERNAL function for post-processing the output of `gam.fit5`.

Usage

```
gam.fit5.post.proc(object, Sl, L, lsp0, S, off, gamma)
```

Arguments

`object`	output of `gam.fit5`.
`Sl`	penalty object, output of `Sl.setup`.
`L`	matrix mapping the working smoothing parameters.
`lsp0`	log smoothing parameters.
`S`	penalty matrix.
`off`	vector of offsets.
`gamma`	parameter for increasing model smoothness in fitting.

Value

A list containing:

- `R`: unpivoted Choleski of estimated expected hessian of log-likelihood.
- `Vb`: the Bayesian covariance matrix of the model parameters.
- `Ve`: "frequentist" alternative to `Vb`.
- `Vc`: corrected covariance matrix.
- `F`: matrix of effective degrees of freedom (EDF).
- `edf`: `diag(F)`.
- `edf2`: `diag(2F-FF)`.

Author(s)

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

gam.mh *Simple posterior simulation with gam fits*

Description

GAM coefficients can be simulated directly from the Gaussian approximation to the posterior for the coefficients, or using a simple Metropolis Hastings sampler. See also ginla.

Usage

```
gam.mh(b,ns=10000,burn=1000,t.df=40,rw.scale=.25,thin=1)
```

Arguments

b	a fitted model object from gam. bam fits are not supported.
ns	the number of samples to generate.
burn	the length of any initial burn in period to discard (in addition to ns).
t.df	degrees of freedom for static multivariate t proposal. Lower for heavier tailed proposals.
rw.scale	Factor by which to scale posterior covariance matrix when generating random walk proposals. Negative or non finite to skip the random walk step.
thin	retain only every thin samples.

Details

Posterior simulation is particularly useful for making inferences about non-linear functions of the model coefficients. Simulate random draws from the posterior, compute the function for each draw, and you have a draw from the posterior for the function. In many cases the Gaussian approximation to the posterior of the model coefficients is accurate, and samples generated from it can be treated as samples from the posterior for the coefficients. See example code below. This approach is computationally very efficient.

In other cases the Gaussian approximation can become poor. A typical example is in a spatial model with a log or logit link when there is a large area of observations containing only zeroes. In this case the linear predictor is poorly identified and the Gaussian approximation can become useless (an example is provided below). In that case it can sometimes be useful to simulate from the posterior using a Metropolis Hastings sampler. A simple approach alternates fixed proposals, based on the Gaussian approximation to the posterior, with random walk proposals, based on a shrunken version of the approximate posterior covariane matrix. gam.mh implements this. The fixed proposal often promotes rapid mixing, while the random walk component ensures that the chain does not become stuck in regions for which the fixed Gaussian proposal density is much lower than the posterior density.

The function reports the acceptance rate of the two types of step. If the random walk acceptance probability is higher than a quarter then rw.step should probably be increased. Similarly if the acceptance rate is too low, it should be decreased. The random walk steps can be turned off altogether (see above), but it is important to check the chains for stuck sections if this is done.

Value

A list containing the retained simulated coefficients in matrix bs and two entries for the acceptance probabilities.

Author(s)

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

References

Wood, S.N. (2015) Core Statistics, Cambridge

Examples

```
library(mgcv)
set.seed(3);n <- 400

###########################################
## First example: simulated Tweedie model...
###########################################

dat <- gamSim(1,n=n,dist="poisson",scale=.2)
dat$y <- rTweedie(exp(dat$f),p=1.3,phi=.5) ## Tweedie response
b <- gam(y~s(x0)+s(x1)+s(x2)+s(x3),family=tw(),
          data=dat,method="REML")

## simulate directly from Gaussian approximate posterior...
br <- rmvn(1000,coef(b),vcov(b))

## Alternatively use MH sampling...
br <- gam.mh(b,thin=2,ns=2000,rw.scale=.15)$bs
## If 'coda' installed, can check effective sample size
## require(coda);effectiveSize(as.mcmc(br))

## Now compare simulation results and Gaussian approximation for
## smooth term confidence intervals...
x <- seq(0,1,length=100)
pd <- data.frame(x0=x,x1=x,x2=x,x3=x)
X <- predict(b,newdata=pd,type="lpmatrix")
par(mfrow=c(2,2))
for(i in 1:4) {
  plot(b,select=i,scale=0,scheme=1)
  ii <- b$smooth[[i]]$first.para:b$smooth[[i]]$last.para
  ff <- X[,ii]%*%t(br[,ii]) ## posterior curve sample
  fq <- apply(ff,1,quantile,probs=c(.025,.16,.84,.975))
  lines(x,fq[1,],col=2,lty=2);lines(x,fq[4,],col=2,lty=2)
  lines(x,fq[2,],col=2);lines(x,fq[3,],col=2)
}

##############################################################
## Second example, where Gaussian approximation is a failure...
##############################################################
```

```
y <- c(rep(0, 89), 1, 0, 1, 0, 0, 1, rep(0, 13), 1, 0, 0, 1,
       rep(0, 10), 1, 0, 0, 1, 1, 0, 1, rep(0,4), 1, rep(0,3),
       1, rep(0, 3), 1, rep(0, 10), 1, rep(0, 4), 1, 0, 1, 0, 0,
       rep(1, 4), 0, rep(1, 5), rep(0, 4), 1, 1, rep(0, 46))
set.seed(3);x <- sort(c(0:10*5,rnorm(length(y)-11)*20+100))
b <- gam(y ~ s(x, k = 15),method = 'REML', family = binomial)
br <- gam.mh(b,thin=2,ns=2000,rw.scale=.4)$bs
X <- model.matrix(b)
par(mfrow=c(1,1))
plot(x, y, col = rgb(0,0,0,0.25), ylim = c(0,1))
ff <- X%*%t(br) ## posterior curve sample
linv <- b$family$linkinv
## Get intervals for the curve on the response scale...
fq <- linv(apply(ff,1,quantile,probs=c(.025,.16,.5,.84,.975)))
lines(x,fq[1,],col=2,lty=2);lines(x,fq[5,],col=2,lty=2)
lines(x,fq[2,],col=2);lines(x,fq[4,],col=2)
lines(x,fq[3,],col=4)
## Compare to the Gaussian posterior approximation
fv <- predict(b,se=TRUE)
lines(x,linv(fv$fit))
lines(x,linv(fv$fit-2*fv$se.fit),lty=3)
lines(x,linv(fv$fit+2*fv$se.fit),lty=3)
## ... Notice the useless 95% CI (black dotted) based on the
## Gaussian approximation!
```

---

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 μ_i and an exponential family distribution, or failing that a known mean variance relationship suitable for use of quasi-likelihood methods. Then the the linear predictor of a GAM has a structure something like

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

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 μ_i and g is a link function.

The gam formula for this would be
`y ~ 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 ~ 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 ~ 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 ~ x1 + x2 + s(x3) + te(x4,x5,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 ~ ti(x) + ti(z) + ti(x,z)`, or

y ~ 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 ith element multiples the ith 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 ~ 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 ~ 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 ith 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 $\mathbf{X}$, and want to penalize the corresponding coefficients, β 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. (2017) Generalized Additive Models: An Introduction with R (2nd edition). 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 ~ 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 <- gam(y ~ 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 ~ 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 <- gam(y ~ 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)
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))
dat$X <- model.matrix(~fac-1)
b <- rnorm(20)*.5
dat$y <- dat$y + dat$X%*%b

## now fit appropriate random effect model...
PP <- list(X=list(rank=20,diag(20)))
rm <- gam(y~ X+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
```

```
## 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")
Z <- model.matrix(~Rail-1,data=Rail,
     contrasts.arg=list(Rail="contr.treatment"))
b <- gam(travel~Z,data=Rail,paraPen=list(Z=list(diag(6))),method="ML")

b0
(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")
Z <- model.matrix(~Rail-1,data=Rail,
     contrasts.arg=list(Rail="contr.treatment"))
b <- gam(travel~Z,data=Rail,paraPen=list(Z=list(diag(6))),method="REML")

b0
(b$reml.scale/b$sp)^.5 ## `gam' REML estimate of Rail sd
b$reml.scale^.5         ## `gam' REML estimate of residual sd

#################################################################
## 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)
p <- binomial()$linkinv(dat$f-6) ## make 1's rare
dat$y <- rbinom(p,1,p)      ## re-simulate rare response

## Now sample all the 1's but only proportion S of the 0's
S <- 0.02                   ## sampling fraction of zeroes
dat <- dat[dat$y==1 | runif(n) < S,] ## sampling

## Create offset based on total sampling fraction
dat$s <- rep(log(nrow(dat)/n),nrow(dat))

lr.fit <- gam(y~s(x0,bs="cr")+s(x1,bs="cr")+s(x2,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))
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]]]
        ind <- sort.int(xx,index.return=TRUE)$ix
        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)
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)
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)
```

`gam.outer` *Minimize GCV or UBRE score of a GAM using 'outer' iteration*

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

```
gam.outer(lsp,fscale,family,control,method,optimizer,
          criterion,scale,gamma,G,start=NULL,nei=NULL,...)
```

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.fit3` etc.

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.
start	starting parameter values.
nei	List specifying neighbourhood structure if NCV used. See 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

See Also

gam.fit3, gam, magic

---

gam.reparam *Finding stable orthogonal re-parameterization of the square root penalty.*

Description

INTERNAL function for finding an orthogonal re-parameterization which avoids "dominant machine zero leakage" between components of the square root penalty.

Usage

```
gam.reparam(rS, lsp, deriv)
```

Arguments

rS	list of the square root penalties: last entry is root of fixed penalty, if fixed.penalty==TRUE (i.e. length(rS)>length(sp)). The assumption here is that rS[[i]] are in a null space of total penalty already; see e.g. totalPenaltySpace and mini.roots.
lsp	vector of log smoothing parameters.
deriv	if deriv==1 also the first derivative of the log-determinant of the penalty matrix is returned, if deriv>1 also the second derivative is returned.

Value

A list containing

- S: the total penalty matrix similarity transformed for stability.
- rS: the component square roots, transformed in the same way.
- Qs: the orthogonal transformation matrix S = t(Qs)%*%S0%*%Qs, where S0 is the untransformed total penalty implied by sp and rS on input.
- det: log|S|.
- det1: dlog|S|/dlog(sp) if deriv >0.
- det2: hessian of log|S| wrt log(sp) if deriv>1.

Author(s)

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

---

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

---

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.

The first approach is to modify the smoothing penalty with an additional shrinkage term. Smooth classes`cs.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.

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

See Also

gam, step.gam

Examples

```
## an example of automatic model selection via null space penalization
library(mgcv)
set.seed(3);n<-200
dat <- gamSim(1,n=n,scale=.15,dist="poisson") ## simulate data
dat$x4 <- runif(n, 0, 1);dat$x5 <- runif(n, 0, 1) ## spurious

b<-gam(y~s(x0)+s(x1)+s(x2)+s(x3)+s(x4)+s(x5),data=dat,
         family=poisson,select=TRUE,method="REML")
```

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

---

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=TRUE)
```

Arguments

`sm`	A list of smooth objects as returned by `smooth.construct`.
`Xp`	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.

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
## estimate model with redundant smooth interaction (bad idea).
b<-gam(y~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 <- 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 <- gam(y ~ 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 *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)

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

Arguments

x	a fitted model object of class gam as produced by gam(), or an onject of class gam.vcomp for the print method.
rescale	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 intervals for the variance components can be obtained from large sample likelihood results. This gives the confidence level to work at.
...	other arguments.

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 list. A list will be returned if the smoothness selection method used was REML or ML, or where a model has more smoothing parameters than actually estimated.

If a list, it will contain some or all of the elements described depending on the method of smoothness selection used, and whether some smoothing parameters were not estimated.

Depending on the smoothness selection method, element `vc` may contain a matrix or a vector. If REML or ML smoothness selection was used, `vc` will be a matrix whose first column gives standard deviations for each term, while the subsequent columns give lower and upper confidence bounds, on the same scale. For models fitted with other smoothness selection methods, component `vc` will be a vector of standard deviations.

The `all` element is a vector of variance components for all the smoothing parameters (estimated + fixed or replicated).

Additionally, for REML or ML smoothness selection, the numerical rank of the covariance matrix and the Hessian matrix are returns as elements `rank` and `rank.hess` respectively.

Author(s)

Simon N. Wood <`simon.wood@r-project.org`> modified by Gavin Simpson

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)
a <- factor(sample(1:10,400,replace=TRUE))
b <- factor(sample(1:7,400,replace=TRUE))
Xa <- model.matrix(~a-1)    ## random main effects
Xb <-  model.matrix(~b-1)
Xab <- model.matrix(~a:b-1) ## random interaction
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 <- gam(y~s(a,bs="re")+s(b,bs="re")+s(a,b,bs="re")+s(x0,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`.

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

See Also

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

gamlss.etamu *Transform derivatives wrt mu to derivatives wrt linear predictor*

Description

Mainly intended for internal use in specifying location scale models. Let `g(mu) = lp`, where `lp` is the linear predictor, and `g` is the link function. Assume that we have calculated the derivatives of the log-likelihood wrt `mu`. This function uses the chain rule to calculate the derivatives of the log-likelihood wrt `lp`. See `trind.generator` for array packing conventions.

Usage

```
gamlss.etamu(l1, l2, l3 = NULL, l4 = NULL, ig1, g2, g3 = NULL,
  g4 = NULL, i2, i3 = NULL, i4 = NULL, deriv = 0)
```

Arguments

`l1`	array of 1st order derivatives of log-likelihood wrt mu.
`l2`	array of 2nd order derivatives of log-likelihood wrt mu.
`l3`	array of 3rd order derivatives of log-likelihood wrt mu.
`l4`	array of 4th order derivatives of log-likelihood wrt mu.
`ig1`	reciprocal of the first derivative of the link function wrt the linear predictor.
`g2`	array containing the 2nd order derivative of the link function wrt the linear predictor.
`g3`	array containing the 3rd order derivative of the link function wrt the linear predictor.
`g4`	array containing the 4th order derivative of the link function wrt the linear predictor.
`i2`	two-dimensional index array, such that `l2[,i2[i,j]]` contains the partial w.r.t. params indexed by i,j with no restriction on the index values (except that they are in 1,...,ncol(l1)).
`i3`	third-dimensional index array, such that `l3[,i3[i,j,k]]` contains the partial w.r.t. params indexed by i,j,k.
`i4`	third-dimensional index array, such that `l4[,i4[i,j,k,l]]` contains the partial w.r.t. params indexed by i,j,k,l.
`deriv`	if `deriv==0` only first and second order derivatives will be calculated. If `deriv==1` the function goes up to 3rd order, and if `deriv==2` it provides also 4th order derivatives.

Value

A list where the arrays `l1`, `l2`, `l3`, `l4` contain the derivatives (up to order four) of the log-likelihood wrt the linear predictor.

Author(s)

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

See Also

trind.generator

---

gamlss.gH *Calculating derivatives of log-likelihood wrt regression coefficients*

Description

Mainly intended for internal use with location scale model families. Given the derivatives of the log-likelihood wrt the linear predictor, this function obtains the derivatives and Hessian wrt the regression coefficients and derivatives of the Hessian w.r.t. the smoothing parameters. For input derivative array packing conventions see trind.generator.

Usage

```
gamlss.gH(X, jj, l1, l2, i2, l3 = 0, i3 = 0, l4 = 0, i4 = 0, d1b = 0,
  d2b = 0, deriv = 0, fh = NULL, D = NULL,sandwich=FALSE)
```

Arguments

X	matrix containing the model matrices of all the linear predictors.
jj	list of index vectors such that X[,jj[[i]]] is the model matrix of the i-th linear predictor.
l1	array of 1st order derivatives of each element of the log-likelihood wrt each parameter.
l2	array of 2nd order derivatives of each element of the log-likelihood wrt each parameter.
i2	two-dimensional index array, such that l2[,i2[i,j]] contains the partial w.r.t. params indexed by i,j with no restriction on the index values (except that they are in 1,...,ncol(l1)).
l3	array of 3rd order derivatives of each element of the log-likelihood wrt each parameter.
i3	third-dimensional index array, such that l3[,i3[i,j,k]] contains the partial w.r.t. params indexed by i,j,k.
l4	array of 4th order derivatives of each element of the log-likelihood wrt each parameter.
i4	third-dimensional index array, such that l4[,i4[i,j,k,l]] contains the partial w.r.t. params indexed by i,j,k,l.
d1b	first derivatives of the regression coefficients wrt the smoothing parameters.
d2b	second derivatives of the regression coefficients wrt the smoothing parameters.

deriv	if deriv==0 only first and second order derivatives will be calculated. If deriv==1 the function return also the diagonal of the first derivative of the Hessian, if deriv==2 it return the full 3rd order derivative and if deriv==3 it provides also 4th order derivatives.
fh	eigen-decomposition or Cholesky factor of the penalized Hessian.
D	diagonal matrix, used to provide some scaling.
sandwich	set to TRUE to return sandwich estimator 'filling', as opposed to the Hessian, in l2.

Value

A list containing lb - the grad vector w.r.t. coefs; lbb - the Hessian matrix w.r.t. coefs; d1H - either a list of the derivatives of the Hessian w.r.t. the smoothing parameters, or a single matrix whose columns are the leading diagonals of these dervative matrices; trHid2H - the trace of the inverse Hessian multiplied by the second derivative of the Hessian w.r.t. all combinations of smoothing parameters.

Author(s)

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

See Also

trind.generator

---

gamm *Generalized Additive Mixed Models*

Description

Fits the specified generalized additive mixed model (GAMM) to data, by a call to lme in the normal errors identity link case, or by a call to gammPQL (a modification of glmmPQL 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 available 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",returnObject=TRUE),
niterPQL=20,verbosePQL=TRUE,method="ML",drop.unused.levels=TRUE,
mustart=NULL, etastart=NULL,...)
```

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`, `te` etc.) can be added to the right hand side of the formula. Note that `ids` for smooths and fixed smoothing parameters are not supported. Any offset should be specified in the formula.

`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` provided there is no offset term, otherwise `gammPQL` is used.

`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 `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 `lme` 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 `lme` 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.

`mustart` starting values for mean if PQL used.

`etastart` starting values for linear predictor if PQL used (over-rides `mustart` if supplied).

`...` 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 `lme`. 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 `glmmPQL` 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 `nlme` 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 `lme` 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 `lme` convergence failures, consider modifying `options(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. This is based on the working model when using `gammPQL`.
- `lme` — the fitted model object returned by `lme` or `gammPQL`. Note that the model formulae and grouping structures may appear to be rather bizarre, because of the manner in which the GAMM is split up and the calls to `lme` and `gammPQL` are constructed.

WARNINGS

`gamm` has 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 `lme` 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 `memory.limit`).

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. It is also based on the working model when PQL is used.

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

See Also

`magic` for an alternative for correlated data, `te`, `s`, `predict.gam`, `plot.gam`, `summary.gam`, `negbin`, `vis.gam`,`pdTens`, gamm4 (https://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)
b <- gamm(y~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~te(x0,x1)+s(x2)+s(x3),data=dat)
op <- 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")
b2 <- gamm(y~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)

## In the generalized case the 'gam' object is based on the working
## model used in the PQL fitting. Residuals for this are not
## that useful on their own as the following illustrates...

gam.check(b2$gam)

## But more useful residuals are easy to produce on a model
## by model basis. For example...

fv <- exp(fitted(b2$lme)) ## predicted values (including re)
rsd <- (b2$gam$y - fv)/sqrt(fv) ## Pearson residuals (Poisson case)
op <- par(mfrow=c(1,2))
qqnorm(rsd);plot(fv^.5,rsd)
par(op)

## 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] <- 0.6*e[i-1] + e[i]
y <- f + e
op <- par(mfrow=c(2,2))
## Fit model with AR1 residuals
b <- gamm(y~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~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] <- 0.6*e[i-1]*(fac[i-1]==fac[i]) + e[i]
y <- f + e
b <- gamm(y~s(x,k=20),correlation=corAR1(form=~1|fac))
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)
f <- dat$f
## simulate nested random effects....
fa <- as.factor(rep(1:10,rep(4*n.g,10)))
ra <- rep(rnorm(10),rep(4*n.g,10))
fb <- as.factor(rep(rep(1:4,rep(n.g,4)),10))
rb <- rep(rnorm(4),rep(n.g,4))
for (i in 1:9) rb <- c(rb,rep(rnorm(4),rep(n.g,4)))
## simulate auto-correlated errors within groups
e<-array(0,0)
for (i in 1:40) {
  eg <- rnorm(n.g, 0, sig)
  for (j in 2:n.g) eg[j] <- eg[j-1]*0.6+ eg[j]
  e<-c(e,eg)
}
dat$y <- f + ra + rb + e
dat$fa <- fa;dat$fb <- fb
## fit model ....
b <- 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$lme,level=5)
rownames(refa) <- substr(rownames(refa),start=9,stop=20)
refb <- ranef(b$lme,level=6)
rownames(refb) <- substr(rownames(refb),start=9,stop=20)

## make a prediction, with random effects zero...
p0 <- predict(b$gam,data.frame(x0=.3,x1=.6,x2=.98,x3=.77))

## add in effect for fa = "2" and fb="2/4"...
p <- p0 + refa["2",1] + refb["2/4",1]

## and a "spatial" example...
library(nlme);set.seed(1);n <- 100
dat <- gamSim(2,n=n,scale=0) ## standard example
attach(dat)
old.par<-par(mfrow=c(2,2))
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$z))
V <- corMatrix(cstr) ## correlation matrix for data
Cv <- chol(V)
e <- t(Cv) %*% rnorm(n)*0.05 # correlated errors
## next add correlated simulated errors to expected values
data$y <- f + e ## ... to produce response
b<- gamm(y~s(x,z,k=50),correlation=corGaus(.1,form=~x+z),
         data=data)
plot(b$gam) # gamm fit accounting for correlation
# overfits when correlation ignored.....
b1 <- gamm(y~s(x,z,k=50),data=data);plot(b1$gam)
b2 <- gam(y~s(x,z,k=50),data=data);plot(b2)
par(old.par)
```

---

gammals *Gamma location-scale model family*

Description

The gammals family implements gamma location scale additive models in which the log of the mean, μ, and the log of the scale parameter, ϕ (see details) can depend on additive smooth predictors. The parameterization is of the usual GLM type where the variance of the response is given by $\phi\mu^2$. Useable only with gam, the linear predictors are specified via a list of formulae.

Usage

```
gammals(link=list("identity","log"),b=-7)
```

Arguments

link	two item list specifying the link for the mean and the standard deviation. See details for meaning which may not be intuitive.
b	The minumum log scale parameter.

Details

Used with gam to fit gamma location - scale models parameterized in terms of the log mean and the log scale parameter (the response variance is the squared mean multiplied by the scale parameter). Note that identity links mean that the linear predictors give the log mean and log scale directly. By default the log link for the scale parameter simply forces the log scale parameter to have a lower limit given by argument b: if η is the linear predictor for the log scale parameter, ϕ, then $\log\phi = b + \log(1 + e^{\eta - b})$.

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 log mean on the right hand side. The second is one sided, specifying the linear predictor for the log scale on the right hand side.

The fitted values for this family will be a two column matrix. The first column is the mean (on original, not log, scale), and the second column is the log scale. Predictions using `predict.gam` will also produce 2 column matrices for type `"link"` and `"response"`. The first column is on the original data scale when `type="response"` and on the log mean scale of the linear predictor when `type="link"`. The second column when `type="response"` is again the log scale parameter, but is on the linear predictor when `type="link"`.

The null deviance reported for this family computed by setting the fitted values to the mean response, but using the model estimated scale.

Value

An object inheriting from class `general.family`.

References

Wood, S.N., N. Pya and B. Saefken (2016), Smoothing parameter and model selection for general smooth models. Journal of the American Statistical Association 111, 1548-1575 doi:10.1080/01621459.2016.1180986

Examples

```
library(mgcv)
## simulate some data
f0 <- function(x) 2 * sin(pi * x)
f1 <- function(x) exp(2 * x)
f2 <- function(x) 0.2 * x^11 * (10 * (1 - x))^6 + 10 *
            (10 * x)^3 * (1 - x)^10
f3 <- function(x) 0 * x
n <- 400;set.seed(9)
x0 <- runif(n);x1 <- runif(n);
x2 <- runif(n);x3 <- runif(n);
mu <- exp((f0(x0)+f2(x2))/5)
th <- exp(f1(x1)/2-2)
y <- rgamma(n,shape=1/th,scale=mu*th)

b1 <- gam(list(y~s(x0)+s(x2),~s(x1)+s(x3)),family=gammals)
plot(b1,pages=1)
summary(b1)
gam.check(b1)
plot(mu,fitted(b1)[,1]);abline(0,1,col=2)
plot(log(th),fitted(b1)[,2]);abline(0,1,col=2)
```

`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 included 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 parameter uncertainty can be computed, this is it. edf1 is a heuristic upper bound for edf2.
family	family object 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 contribution 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 "lme.REML", depending on the fitting criterion used.

mgcv.conv — A list of convergence diagnostics relating to the "magic" parts of smoothing parameter 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 routine 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 smoothing 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	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.
y	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. (2017) Generalized Additive Models: An Introduction with R (2nd edition). 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

---

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

eg	numeric value specifying the example required.
n	number of data to simulate.
dist	character string which may be used to specify 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

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 logb of the standard deviation (see details) 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. Note that internally the family is parameterized in terms of the $\tau = \sigma^{-1}$, i.e. the standard deviation of the precision, so the link and inverse link are coded to reflect this, however the relationships between the linear predictor and the standard deviation are as given above.

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 second column when `type="response"` is again on the reciprocal standard deviation scale (i.e. the square root precision scale). The second column when `type="link"` is $\log(\sigma - b)$. Also `plot.gam` will plot smooths relating to σ on the $\log(\sigma - b)$ scale (so high values correspond to high standard deviation and low values to low standard deviation). Similarly the smoothing penalties are applied on the (log) standard deviation scale, not the log precision scale.

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 (2016), Smoothing parameter and model selection for general smooth models. Journal of the American Statistical Association 111, 1548-1575 doi:10.1080/01621459.2016.1180986

Examples

```
library(mgcv);library(MASS)
b <- gam(list(accel~s(times,k=20,bs="ad"),~s(times)),
            data=mcycle,family=gaulss())
summary(b)
plot(b,pages=1,scale=0)
```

`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>

See Also

gam

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)
```

---

gevlss — *Generalized Extreme Value location-scale model family*

Description

The `gevlss` family implements Generalized Extreme Value location scale additive models in which the location, scale and shape parameters depend on additive smooth predictors. Usable only with gam, the linear predictors are specified via a list of formulae.

Usage

```
gevlss(link=list("identity","identity","logit"))
```

Arguments

`link`	three item list specifying the link for the location scale and shape parameters. See details.

Details

Used with gam to fit Generalized Extreme Value distribution location scale and shape models. The p.d.f. is

$$t(y)^{\xi+1}e^{-t(y)}/\sigma$$

where $t(y) = \{1+\xi(y-\mu)/\sigma\}^{-1/\xi}$ if $\xi \neq 0$ and $t(y) = \exp\{-(y-\mu)/\sigma\}$ if $\xi = 0$.

gam is called with a list containing 3 formulae: the first specifies the response on the left hand side and the structure of the linear predictor for the location parameter, μ, on the right hand side. The second is one sided, specifying the linear predictor for the log scale parameter, $\rho = \log(\sigma)$, on the right hand side. The third is one sided specifying the linear predictor for the shape parameter, ξ.

Link functions "identity" and "log" are available for the location (μ) parameter. There is no choice of link for the log scale parameter ($\rho = \log(\sigma)$). The shape parameter (ξ) defaults to a modified logit link restricting its range to (-1,.5), the upper limit is required to ensure finite variance, while the lower limit ensures consistency of the MLE (Smith, 1985).

The fitted values for this family will be a three column matrix. The first column is the location parameter, the second column is the log scale parameter, the third column is the shape parameter.

This family does not produce a null deviance. Note that the distribution for $\xi = 0$ is approximated by setting ξ to a small number.

The derivative system code for this family is mostly auto-generated, and the family is still somewhat experimental.

The GEV distribution is rather challenging numerically, and for small datasets or poorly fitting models improved numerical robustness may be obtained by using the extended Fellner-Schall method of Wood and Fasiolo (2017) for smoothing parameter estimation. See examples.

Value

An object inheriting from class general.family.

References

Smith, R.L. (1985) Maximum likelihood estimation in a class of nonregular cases. Biometrika 72(1):67-90

Wood, S.N., N. Pya and B. Saefken (2016), Smoothing parameter and model selection for general smooth models. Journal of the American Statistical Association 111, 1548-1575 doi:10.1080/01621459.2016.1180986

Wood, S.N. and M. Fasiolo (2017) A generalized Fellner-Schall method for smoothing parameter optimization with application to Tweedie location, scale and shape models. Biometrics 73(4): 1071-1081. doi:10.1111/biom.12666

Examples

```
library(mgcv)
Fi.gev <- function(z,mu,sigma,xi) {
## GEV inverse cdf.
  xi[abs(xi)<1e-8] <- 1e-8 ## approximate xi=0, by small xi
  x <- mu + ((-log(z))^-xi-1)*sigma/xi
}
```

```
## simulate test data...
f0 <- function(x) 2 * sin(pi * x)
f1 <- function(x) exp(2 * x)
f2 <- function(x) 0.2 * x^11 * (10 * (1 - x))^6 + 10 *
            (10 * x)^3 * (1 - x)^10
set.seed(1)
n <- 500
x0 <- runif(n);x1 <- runif(n);x2 <- runif(n)
mu <- f2(x2)
rho <- f0(x0)
xi <- (f1(x1)-4)/9
y <- Fi.gev(runif(n),mu,exp(rho),xi)
dat <- data.frame(y,x0,x1,x2);pairs(dat)

## fit model....
b <- gam(list(y~s(x2),~s(x0),~s(x1)),family=gevlss,data=dat)

## same fit using the extended Fellner-Schall method which
## can provide improved numerical robustness...
b <- gam(list(y~s(x2),~s(x0),~s(x1)),family=gevlss,data=dat,
         optimizer="efs")

## plot and look at residuals...
plot(b,pages=1,scale=0)
summary(b)

par(mfrow=c(2,2))
mu <- fitted(b)[,1];rho <- fitted(b)[,2]
xi <- fitted(b)[,3]
## Get the predicted expected response...
fv <- mu + exp(rho)*(gamma(1-xi)-1)/xi
rsd <- residuals(b)
plot(fv,rsd);qqnorm(rsd)
plot(fv,residuals(b,"pearson"))
plot(fv,residuals(b,"response"))
```

---

gfam *Grouped families*

Description

Family for use with gam or bam allowing a univariate response vector to be made up of variables from several different distributions. The response variable is supplied as a 2 column matrix, where the first column contains the response observations and the second column indexes the distribution (family) from which it comes. gfam takes a list of families as its single argument.

Useful for modelling data from different sources that are linked by a model sharing some components. Smooth model components that are not shared are usually handled with by variables (see gam.models).

Usage

```
gfam(fl)
```

Arguments

`fl`	A list of families. These can be any families inheriting from `family` or `extended.family` usable with `gam`, provided that they do not usually require a matrix response variable.

Details

Each component function of `gfam` uses the families supplied in the list `fl` to obtain the required quantities for that family's subset of data, and combines the results appropriately. For example it provides the total deviance (twice negative log-likelihood) of the model, along with its derivatives, by computing the family specific deviance and derivatives from each family applied to its subset of data, and summing them. Other quantities are computed in the same way.

Regular exponential families do not compute the same quantities as extended families, so `gfam` converts what these families produce to `extended.family` form internally.

Scale parameters obviously have to be handled separately for each family, and treated as parameters to be estimated, just like other `extended.family` non-location distribution parameters. Again this is handled internally. This requirement is part of the reason that an `extended.family` is always produced, even if all elements of `fl` are standard exponential families. In consequence smoothing parameter estimation is always by REML or NCV.

Note that the null deviance is currently computed by assuming a single parameter model for each family, rather than just one parameter, which may slightly lower explained deviances. Note also that residual checking should probably be done by disaggregating the residuals by family. For this reason functions are not provided to facilitate residual checking with `qq.gam`.

Prediction on the response scale requires that a family index vector is supplied, with the name of the response, as part of the new prediction data. However, families such as `ocat` which usually produce matrix predictions for prediction type `"response"`, will not be able to do so when part of `gfam`.

`gfam` relies on the methods in Wood, Pya and Saefken (2016).

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 (2016), Smoothing parameter and model selection for general smooth models. Journal of the American Statistical Association 111, 1548-1575 doi:10.1080/01621459.2016.1180986

Examples

```
library(mgcv)
## a mixed family simulator function to play with...
sim.gfam <- function(dist,n=400) {
## dist can be norm, pois, gamma, binom, nbinom, tw, ocat (R assumed 4)
## links used are identitiy, log or logit.
  dat <- gamSim(1,n=n,verbose=FALSE)
  nf <- length(dist) ## how many families
  fin <- c(1:nf,sample(1:nf,n-nf,replace=TRUE)) ## family index
  dat[,6:10] <- dat[,6:10]/5 ## a scale that works for all links used
  y <- dat$y;
  for (i in 1:nf) {
    ii <- which(fin==i) ## index of current family
    ni <- length(ii);fi <- dat$f[ii]
    if (dist[i]=="norm") {
      y[ii] <- fi + rnorm(ni)*.5
    } else if (dist[i]=="pois") {
      y[ii] <- rpois(ni,exp(fi))
    } else if (dist[i]=="gamma") {
      scale <- .5
      y[ii] <- rgamma(ni,shape=1/scale,scale=exp(fi)*scale)
    } else if (dist[i]=="binom") {
      y[ii] <- rbinom(ni,1,binomial()$linkinv(fi))
    } else if (dist[i]=="nbinom") {
      y[ii] <- rnbinom(ni,size=3,mu=exp(fi))
    } else if (dist[i]=="tw") {
      y[ii] <- rTweedie(exp(fi),p=1.5,phi=1.5)
    } else if (dist[i]=="ocat") {
      alpha <- c(-Inf,1,2,2.5,Inf)
      R <- length(alpha)-1
      yi <- fi
      u <- runif(ni)
      u <- yi + log(u/(1-u))
      for (j in 1:R) {
        yi[u > alpha[j]&u <= alpha[j+1]] <- j
      }
      y[ii] <- yi
    }
  }
  dat$y <- cbind(y,fin)
  dat
} ## sim.gfam

## some examples

dat <- sim.gfam(c("binom","tw","norm"))
b <- gam(y~s(x0)+s(x1)+s(x2)+s(x3),
         family=gfam(list(binomial,tw,gaussian)),data=dat)
predict(b,data.frame(y=1:3,x0=c(.5,.5,.5),x1=c(.3,.2,.3),
        x2=c(.2,.5,.8),x3=c(.1,.5,.9)),type="response",se=TRUE)
summary(b)
plot(b,pages=1)
```

```
## set up model so that only the binomial observations depend
## on x0...

dat$id1 <- as.numeric(dat$y[,2]==1)
b1 <- gam(y~s(x0,by=id1)+s(x1)+s(x2)+s(x3),
         family=gfam(list(binomial,tw,gaussian)),data=dat)
plot(b1,pages=1) ## note the CI width increase
```

---

ginla *GAM Integrated Nested Laplace Approximation Newton Enhanced*

Description

Apply Integrated Nested Laplace Approximation (INLA, Rue et al. 2009) to models estimable by gam or bam, using the INLA variant described in Wood (2019). Produces marginal posterior densities for each coefficient, selected coefficients or linear transformations of the coefficient vector.

Usage

```
ginla(G,A=NULL,nk=16,nb=100,J=1,interactive=FALSE,integ=0,approx=0)
```

Arguments

G	A pre-fit gam object, as produced by gam(...,fit=FALSE) or bam(...,discrete=TRUE,fit=FALSE).
A	Either a matrix whose rows are transforms of the coefficients that are of interest (no more rows than columns, full row rank), or an array of indices of the parameters of interest. If NULL then distributions are produced for all coefficients.
nk	Number of values of each coefficient at which to evaluate its log marginal posterior density. These points are then spline interpolated.
nb	Number of points at which to evaluate posterior density of coefficients for returning as a gridded function.
J	How many determinant updating steps to take in the log determinant approximation step. Not recommended to increase this.
interactive	If this is >0 or TRUE then every approximate posterior is plotted in red, overlaid on the simple Gaussian approximate posterior. If 2 then waits for user to press return between each plot. Useful for judging whether anything is gained by using INLA approach.
integ	0 to skip integration and just use the posterior modal smoothing parameter. >0 for integration using the CCD approach proposed in Rue et al. (2009).
approx	0 for full approximation; 1 to update Hessian, but use approximate modes; 2 as 1 and assume constant Hessian. See details.

Details

Let β, θ and y denote the model coefficients, hyperparameters/smoothing parameters and response data, respectively. In principle, INLA employs Laplace approximations for $\pi(\beta_i|\theta, y)$ and $\pi(\theta|y)$ and then obtains the marginal posterior distribution $\pi(\beta_i|y)$ by intergrating the approximations to $\pi(\beta_i|\theta, y)\pi(\theta|y)$ w.r.t θ (marginals for the hyperparameters can also be produced). In practice the Laplace approximation for $\pi(\beta_i|\theta, y)$ is too expensive to compute for each β_i and must itself be approximated. To this end, there are two quantities that have to be computed: the posterior mode $\beta^*|\beta_i$ and the determinant of the Hessian of the joint log density $\log \pi(\beta, \theta, y)$ w.r.t. β at the mode. Rue et al. (2009) originally approximated the posterior conditional mode by the conditional mode implied by a simple Gaussian approximation to the posterior $\pi(\beta|y)$. They then approximated the log determinant of the Hessian as a function of β_i using a first order Taylor expansion, which is cheap to compute for the sparse model representaiton that they use, but not when using the dense low rank basis expansions used by `gam`. They also offer a more expensive alternative approximation based on computing the log determiannt with respect only to those elements of β with sufficiently high correlation with β_i according to the simple Gaussian posterior approximation: efficiency again seems to rest on sparsity. Wood (2020) suggests computing the required posterior modes exactly, and basing the log determinant approximation on a BFGS update of the Hessian at the unconditional model. The latter is efficient with or without sparsity, whereas the former is a 'for free' improvement. Both steps are efficient because it is cheap to obtain the Cholesky factor of $H[-i, -i]$ from that of H - see `choldrop`. This is the approach taken by this routine.

The `approx` argument allows two further approximations to speed up computations. For `approx==1` the exact posterior conditional modes are not used, but instead the conditional modes implied by the simple Gaussian posterior approximation. For `approx==2` the same approximation is used for the modes and the Hessian is assumed constant. The latter is quite fast as no log joint density gradient evaluations are required.

Note that for many models the INLA estimates are very close to the usual Gaussian approximation to the posterior, the `interactive` argument is useful for investigating this issue.

`bam` models are only supported with the `disrete=TRUE` option. The `discrete=FALSE` approach would be too inefficient. AR1 models are not supported (related arguments are simply ignored).

Value

A list with elements `beta` and `density`, both of which are matrices. Each row relates to one coefficient (or linear coefficient combination) of interest. Both matrices have `nb` columns. If `int!=0` then a further element `reml` gives the integration weights used in the CCD integration, with the central point weight given first.

WARNINGS

This routine is still somewhat experimental, so details are liable to change. Also currently not all steps are optimally efficient.

The routine is written for relatively expert users.

`ginla` is not designed to deal with rank deficient models.

Author(s)

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

References

Rue, H, Martino, S. & Chopin, N. (2009) Approximate Bayesian inference for latent Gaussian models by using integrated nested Laplace approximations (with discussion). Journal of the Royal Statistical Society, Series B. 71: 319-392.

Wood (2020) Simplified Integrated Laplace Approximation. Biometrika 107(1): 223-230. [Note: There is an error in the theorem proof - theoretical properties are weaker than claimed - under investigation]

Examples

```
require(mgcv); require(MASS)

## example using a scale location model for the motorcycle data. A simple
## plotting routine is produced first...

plot.inla <- function(x,inla,k=1,levels=c(.025,.1,.5,.9,.975),
             lcol = c(2,4,4,4,2),lwd = c(1,1,2,1,1),lty=c(1,1,1,1,1),
      xlab="x",ylab="y",cex.lab=1.5) {
  ## a simple effect plotter, when distributions of function values of
  ## 1D smooths have been computed
  require(splines)
  p <- length(x)
  betaq <- matrix(0,length(levels),p) ## storage for beta quantiles
  for (i in 1:p) { ## work through x and betas
    j <- i + k - 1
    p <- cumsum(inla$density[j,])*(inla$beta[j,2]-inla$beta[j,1])
    ## getting quantiles of function values...
    betaq[,i] <- approx(p,y=inla$beta[j,],levels)$y
  }
  xg <- seq(min(x),max(x),length=200)
  ylim <- range(betaq)
  ylim <- 1.1*(ylim-mean(ylim))+mean(ylim)
  for (j in 1:length(levels)) { ## plot the quantiles
    din <- interpSpline(x,betaq[j,])
    if (j==1) {
      plot(xg,predict(din,xg)$y,ylim=ylim,type="l",col=lcol[j],
           xlab=xlab,ylab=ylab,lwd=lwd[j],cex.lab=1.5,lty=lty[j])
    } else lines(xg,predict(din,xg)$y,col=lcol[j],lwd=lwd[j],lty=lty[j])
  }
} ## plot.inla

## set up the model with a `gam' call...

G <- gam(list(accel~s(times,k=20,bs="ad"),~s(times)),
          data=mcycle,family=gaulss(),fit=FALSE)
b <- gam(G=G,method="REML") ## regular GAM fit for comparison

## Now use ginla to get posteriors of estimated effect values
## at evenly spaced times. Create A matrix for this...

rat <- range(mcycle$times)
```

```
pd0 <- data.frame(times=seq(rat[1],rat[2],length=20))
X0 <- predict(b,newdata=pd0,type="lpmatrix")
X0[,21:30] <- 0
pd1 <- data.frame(times=seq(rat[1],rat[2],length=10))
X1 <- predict(b,newdata=pd1,type="lpmatrix")
X1[,1:20] <- 0
A <- rbind(X0,X1) ## A maps coefs to required function values

## call ginla. Set integ to 1 for integrated version.
## Set interactive = 1 or 2 to plot marginal posterior distributions
## (red) and simple Gaussian approximation (black).

inla <- ginla(G,A,integ=0)

par(mfrow=c(1,2),mar=c(5,5,1,1))
fv <- predict(b,se=TRUE) ## usual Gaussian approximation, for comparison

## plot inla mean smooth effect...
plot.inla(pd0$times,inla,k=1,xlab="time",ylab=expression(f[1](time)))

## overlay simple Gaussian equivalent (in grey) ...
points(mcycle$times,mcycle$accel,col="grey")
lines(mcycle$times,fv$fit[,1],col="grey",lwd=2)
lines(mcycle$times,fv$fit[,1]+2*fv$se.fit[,1],lty=2,col="grey",lwd=2)
lines(mcycle$times,fv$fit[,1]-2*fv$se.fit[,1],lty=2,col="grey",lwd=2)

## same for log sd smooth...
plot.inla(pd1$times,inla,k=21,xlab="time",ylab=expression(f[2](time)))
lines(mcycle$times,fv$fit[,2],col="grey",lwd=2)
lines(mcycle$times,fv$fit[,2]+2*fv$se.fit[,2],col="grey",lty=2,lwd=2)
lines(mcycle$times,fv$fit[,2]-2*fv$se.fit[,2],col="grey",lty=2,lwd=2)

## ... notice some real differences for the log sd smooth, especially
## at the lower and upper ends of the time interval.
```

`gumbls` *Gumbel location-scale model family*

Description

The `gumbls` family implements Gumbel location scale additive models in which the location and scale parameters (see details) can depend on additive smooth predictors. Useable only with `gam`, the linear predictors are specified via a list of formulae.

Usage

```
gumbls(link=list("identity","log"),b=-7)
```

Arguments

link	two item list specifying the link for the location μ and log scale parameter β. See details for meaning, which may not be intuitive.
b	The minumum log scale parameter.

Details

Let $z = (y - \mu)e^{-\beta}$, then the log Gumbel density is $l = -\beta - z - e^{-z}$. The expected value of a Gumbel r.v. is $\mu + \gamma e^{\beta}$ where γ is Eulers constant (about 0.57721566). The corresponding variance is $\pi^2 e^{2\beta}/6$.

gumbls is used with gam to fit Gumbel location - scale models parameterized in terms of location parameter μ and the log scale parameter β. Note that identity link for the scale parameter means that the corresponding linear predictor gives β directly. By default the log link for the scale parameter simply forces the log scale parameter to have a lower limit given by argument b: if η is the linear predictor for the log scale parameter, β, then $\beta = b + \log(1 + e^{\eta - b})$.

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 location parameter, μ, on the right hand side. The second is one sided, specifying the linear predictor for the lg scale, β, on the right hand side.

The fitted values for this family will be a two column matrix. The first column is the mean, and the second column is the log scale parameter, β. Predictions using predict.gam will also produce 2 column matrices for type "link" and "response". The first column is on the original data scale when type="response" and on the log mean scale of the linear predictor when type="link". The second column when type="response" is again the log scale parameter, but is on the linear predictor when type="link".

Value

An object inheriting from class general.family.

References

Wood, S.N., N. Pya and B. Saefken (2016), Smoothing parameter and model selection for general smooth models. Journal of the American Statistical Association 111, 1548-1575 doi:10.1080/01621459.2016.1180986

Examples

```
library(mgcv)
## simulate some data
f0 <- function(x) 2 * sin(pi * x)
f1 <- function(x) exp(2 * x)
f2 <- function(x) 0.2 * x^11 * (10 * (1 - x))^6 + 10 *
            (10 * x)^3 * (1 - x)^10
n <- 400;set.seed(9)
x0 <- runif(n);x1 <- runif(n);
x2 <- runif(n);x3 <- runif(n);
mu <- f0(x0)+f1(x1)
beta <- exp(f2(x2)/5)
y <- mu - beta*log(-log(runif(n))) ## Gumbel quantile function
```

```
b <- gam(list(y~s(x0)+s(x1),~s(x2)+s(x3)),family=gumbls)
plot(b,pages=1,scale=0)
summary(b)
gam.check(b)
```

---

identifiability *Identifiability constraints*

Description

Smooth terms are generally only identifiable up to an additive constant. In consequence sum-to-zero identifiability constraints are imposed on most smooth terms. The exceptions are terms with by variables which cause the smooth to be identifiable without constraint (that doesn't include factor by variables), and random effect terms. Alternatively smooths can be set up to pass through zero at a user specified point.

Details

By default each smooth term is subject to the sum-to-zero constraint

$$\sum_i f(x_i) = 0.$$

The constraint is imposed by reparameterization. The sum-to-zero constraint causes the term to be orthogonal to the intercept: alternative constraints lead to wider confidence bands for the constrained smooth terms.

No constraint is used for random effect terms, since the penalty (random effect covariance matrix) anyway ensures identifiability in this case. Also if a by variable means that the smooth is anyway identifiable, then no extra constraint is imposed. Constraints are imposed for factor by variables, so that the main effect of the factor must usually be explicitly added to the model (the example below is an exception).

Occasionally it is desirable to substitute the constraint that a particular smooth curve should pass through zero at a particular point: the pc argument to s, te, ti and t2 allows this: if specified then such constraints are always applied.

Author(s)

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

Examples

```
## Example of three groups, each with a different smooth dependence on x
## but each starting at the same value...
require(mgcv)
set.seed(53)
n <- 100;x <- runif(3*n);z <- runif(3*n)
```

```
fac <- factor(rep(c("a","b","c"),each=100))
y <- c(sin(x[1:100]*4),exp(3*x[101:200])/10-.1,exp(-10*(x[201:300]-.5))/
       (1+exp(-10*(x[201:300]-.5)))-0.9933071) + z*(1-z)*5 + rnorm(100)*.4

## 'pc' used to constrain smooths to 0 at x=0...
b <- gam(y~s(x,by=fac,pc=0)+s(z))
plot(b,pages=1)
```

`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

`bnd`	A two column matrix, the rows of which define the vertices of polygons defining the boundary of a region. Different polygons should be separated by an `NA` row, and the polygons are assumed closed. Alternatively can be a lists where `bnd[[i]][[1]]`, `bnd[[i]][[2]]` defines the ith boundary loop.
`x`	A two column matrix. Each row is a point to test for inclusion in the region defined by `bnd`. Can also be a 2-vector, defining a single point.

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>

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)
```

`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 *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

X	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 $X^T X$, 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,

`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,xname=NULL,yname=NULL)
```

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.
`y`	y co-ordinates of points to be tested.
`xname`	name of variable in `bnd` that `x` relates to. Defaults to name of variable supplied as `x`.
`yname`	name of variable in `bnd` that `y` relates to. Defaults to name of variable supplied as `y`.

Details

Segments of boundary are separated by `NA`s, or are in separate list elements. The boundary co-ordinates 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>

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=".")

## alternatively, give the names of x and y
d <- data.frame(x, y); rm(x, y)
in.bnd2 <- inSide(bnd, d$x, d$y, xname="x", yname="y")
all.equal(in.bnd, in.bnd2)
```

`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, extra.special = NULL)
```

Arguments

`gf`	A GAM formula as supplied to `gam` or `gamm`, or a list of such formulae, as supplied for some `gam` families.
`extra.special`	Name of any extra special in formula in addition to `s`, `te`, `ti` and `t2`.

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 expanded, 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>

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 required 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 possible)? 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 update smoothers blockwise, and not do this.

sam
: jags sample object, containing at least fields b (coefficients) and rho (log smoothing parameters). May also contain field mu containing monitored expected response.

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 structure 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 specifiy 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 possibilites 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 gamObject, 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. (2016) Just Another Gibbs Additive Modeller: Interfacing JAGS and mgcv. Journal of Statistical Software 75(7):1-15 doi:10.18637/jss.v075.i07)

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")

## Set directory and file name for file containing jags code.
## In real use you would *never* use tempdir() for this. It is
## only done here to keep CRAN happy, and avoid any chance of
## an accidental overwrite. Instead you would use
## setwd() to set an appropriate working directory in which
## to write the file, and just set the file name to what you
## want to call it (e.g. "test.jags" here).

jags.file <- paste(tempdir(),"/test.jags",sep="")

## 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 <- jagam(y~s(x0)+s(x1)+s(x2)+s(x3),data=dat,file=jags.file,
            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(jags.file,data=jd$jags.data,inits=jd$jags.ini,n.chains=1)
list.samplers(jm)
sam <- jags.samples(jm,c("b","rho","scale"),n.iter=10000,thin=10)
jam <- sim2jam(sam,jd$pregam)
plot(jam,pages=1)
jam
pd <- data.frame(x0=c(.5,.6),x1=c(.4,.2),x2=c(.8,.4),x3=c(.1,.1))
fv <- predict(jam,newdata=pd)
## and some minimal checking...
require(coda)
effectiveSize(as.mcmc.list(sam$b))

## End(Not run)

## a gamma example...
set.seed(1); n <- 400
dat <- gamSim(1,n=n,dist="normal",scale=2)
scale <- .5; Ey <- exp(dat$f/2)
```

```
dat$y <- rgamma(n,shape=1/scale,scale=Ey*scale)
jd <- jagam(y~s(x0)+te(x1,x2)+s(x3),data=dat,family=Gamma(link=log),
            file=jags.file,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
jd$jags.ini$.RNG.seed <- 6 ## how to set RNG seed
jm <-jags.model(jags.file,data=jd$jags.data,inits=jd$jags.ini,n.chains=1)
list.samplers(jm)
sam <- jags.samples(jm,c("b","rho","scale","mu"),n.iter=10000,thin=10)
jam <- sim2jam(sam,jd$pregam)
plot(jam,pages=1)
jam
pd <- data.frame(x0=c(.5,.6),x1=c(.4,.2),x2=c(.8,.4),x3=c(.1,.1))
fv <- predict(jam,newdata=pd)

## End(Not run)
```

`k.check` *Checking smooth basis dimension*

Description

Takes a fitted `gam` object produced by `gam()` and runs diagnostic tests of whether the basis dimension choises are adequate.

Usage

```
k.check(b, subsample=5000, n.rep=400)
```

Arguments

`b`	a fitted `gam` object as produced by `gam()`.
`subsample`	above this number of data, testing uses a random sub-sample of data of this size.
`n.rep`	how many re-shuffles to do to get p-value for k testing.

Details

The test of whether the basis dimension for a smooth is adequate (Wood, 2017, section 5.9) is based on computing an estimate of the residual variance based on differencing residuals that are near neighbours according to the (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 n.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 subsample data, the tests are based of subsample 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.

Value

A matrix contaning the output of the tests described above.

Author(s)

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

References

Wood S.N. (2017) Generalized Additive Models: An Introduction with R (2nd edition). Chapman and Hall/CRC Press.

See Also

choose.k, gam, gam.check

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)
k.check(b)
```

---

ldetS — *Getting log generalized determinant of penalty matrices*

Description

INTERNAL function calculating the log generalized determinant of penalty matrix S stored block-wise in an Sl list (which is the output of Sl.setup).

Usage

```
ldetS(Sl, rho, fixed, np, root = FALSE,Stot=FALSE,repara = TRUE,
      nt = 1,deriv=2,sparse=FALSE)
```

Arguments

`Sl`	the output of `Sl.setup`.
`rho`	the log smoothing parameters.
`fixed`	an array indicating whether the smoothing parameters are fixed (or free).
`np`	number of coefficients.
`root`	indicates whether or not to return the matrix square root, `E`, of the total penalty S_tot.
`Stot`	indicates whether or not to return the total penalty S_tot.
`repara`	if TRUE multi-term blocks will be re-parameterized using `gam.reparam`, and a re-parameterization object supplied in the returned object.
`nt`	number of parallel threads to use.
`deriv`	order of derivative to use
`sparse`	should `E` and/or `S` be sparse?

Value

A list containing:

- `ldetS`: the log-determinant of S.
- `ldetS1`: the gradient of the log-determinant of S.
- `ldetS2`: the Hessian of the log-determinant of S.
- `Sl`: with modified rS terms, if needed and rho added to each block
- `rp`: a re-parameterization list.
- `E`: a total penalty square root such that `t(E)%*%E = S_tot` (if `root==TRUE`).
- `S`: the total penalty matrix (if `Stot==TRUE`).

Author(s)

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

`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,all.derivs=FALSE)
```

Arguments

`y`	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.
`a`	lower limit parameter (>1) used in definition of `p` from `theta`.
`b`	upper limit parameter (<2) used in definition of `p` from `theta`.
`all.derivs`	if `TRUE` then derivatives w.r.t. `mu` are also returned. Only available with `rho` and `phi` parameterization.

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.

Note that if `p` and `phi` (or `theta` and `rho`) both contain only a single unique value, then the underlying code is able to use buffering to avoid repeated calls to expensive log gamma, di-gamma and tri-gamma functions (`mu` can still be a vector of different values). This is much faster than is possible when these parameters are vectors with different values.

Value

A matrix with 6 columns, or 10 if `all.derivs=TRUE`. 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. In this case, and if `all.derivs=TRUE` then the 7th colmn is the derivative w.r.t. `mu`, the 8th is the 2nd derivative w.r.t. `mu`, the 9th is the mixed derivative w.r.t. `theta` and`mu` and the 10th is the mixed derivative w.r.t. `rho` and `mu`.

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="l",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)
}
```

`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 $\mathbf{L1}$ (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.

predict.gam can accept matrix predictors for prediction with such terms, in which case its `newdata` argument will need to be a list. However 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.

The mechanism is usable with random effect smooths which take factor arguments, by using a trick to create a 2D array of factors. Simply create a factor vector containing the columns of the factor matrix stacked end to end (column major order). Then reset the dimensions of this vector to create the appropriate 2D array: the first dimension should be the number of response data and the second the number of columns of the required factor matrix. You can not use `matrix` or `data.matrix` to set up the required matrix of factor levels. See example below.

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 <- runif(n, 0, .9)
f2 <- function(x) 0.2*x^11*(10*(1-x))^6+10*(10*x)^3*(1-x)^10
x1 <- x + .1

f <- f2(x) + f2(x1)  ## response is sum of f at two adjacent x values
y <- f + rnorm(n)*sig

X <- matrix(c(x,x1),n,2) ## matrix covariate contains both x values
b <- gam(y~s(X))

plot(b)  ## reconstruction of f
plot(f,fitted(b))

## example of prediction with summation convention...
predict(b,list(X=X[1:3,]))

## example of prediction that simply evaluates smooth (no summation)...
predict(b,data.frame(X=c(.2,.3,.7)))

#######################################################################
## Simple random effect model example.
## model: y[i] = f(x[i]) + b[k[i]] - b[j[i]] + e[i]
## k[i] and j[i] index levels of i.i.d. random effects, b.
#######################################################################

set.seed(7)
n <- 200
x <- runif(n) ## a continuous covariate

## set up a `factor matrix'...
fac <- factor(sample(letters,n*2,replace=TRUE))
dim(fac) <- c(n,2)

## simulate data from such a model...
nb <- length(levels(fac))
b <- rnorm(nb)
y <- 20*(x-.3)^4 + b[fac[,1]] - b[fac[,2]] + rnorm(n)*.5

L <- matrix(-1,n,2);L[,1] <- 1 ## the differencing 'by' variable

mod <- gam(y ~ s(x) + s(fac,by=L,bs="re"),method="REML")
gam.vcomp(mod)
plot(mod,page=1)

## example of prediction using matrices...
dat <- list(L=L[1:20,],fac=fac[1:20,],x=x[1:20],y=y[1:20])
predict(mod,newdata=dat)
```

```
#####################################################################
## 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#
## smooths.                                                          #
#####################################################################

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))

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))

x <- ox + ix/og;z <- oz + iz/og ## full grid, subsquare by subsquare

## create matrix covariates...
X <- matrix(x,og^2,ig^2,byrow=TRUE)
Z <- matrix(z,og^2,ig^2,byrow=TRUE)

## 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
y <- 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 <- gam(y~s(X,Z,by=L)+L1-1)   ## (L1 and const are confounded here)
b1 <- gam(y~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);
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)))
truth<-matrix(test1(pr$x,pr$z),30,30)
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)) {
## generates random functions...
  m <- ceiling(runif(1)*5) ## number of components
  f <- x*0;
  mu <- runif(m,min(x),max(x));sig <- (runif(m)+.5)*(max(x)-min(x))/10
  for (i in 1:m) f <- f+ dnorm(x,mu[i],sig[i])
  f
}

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
  (0.2*x^11*(10*(1-x))^6+10*(10*x)^3*(1-x)^10)/10
}

f <- 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)

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

logLik.gam *AIC and Log likelihood for a fitted GAM*

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. See details for more information on AIC computation.

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). See e.g. Wood, Pya and Saefken (2016) for a derivation of an appropriate AIC.

For gaussian family models the MLE of the scale parameter is used. For other families with a scale parameter the estimated scale parameter is used. This is usually not exactly the MLE, and is not the simple deviance based estimator used with glm models. This is because the simple deviance based estimator can be badly biased in some cases, for example when a Tweedie distribution is employed with low count data.

There are two possibile AIC's that might be considered for use with GAMs. Marginal AIC is based on the marginal likelihood of the GAM, that is the likelihood based on treating penalized (e.g. spline) coefficients as random and integrating them out. The degrees of freedom is then the number of smoothing/variance parameters + the number of fixed effects. The problem with Marginal AIC is that marginal likelihood underestimates variance components/oversmooths, so that the approach favours simpler models excessively (substituting REML does not work, because REML is not comparable between models with different unpenalized/fixed components). Conditional AIC uses the likelihood of all the model coefficients, evaluated at the penalized MLE. The degrees of freedom to use then is the effective degrees of freedom for the model. However, Greven and Kneib (2010) show that the neglect of smoothing parameter uncertainty can lead to this conditional AIC being excessively likely to select larger models. Wood, Pya and Saefken (2016) propose a simple correction to the effective degrees of freedom to fix this problem. mgcv applies this correction whenever possible: that is when using ML or REML smoothing parameter selection with gam or bam. The correction is not computable when using the Extended Fellner Schall or BFGS optimizer (since the correction requires an estimate of the covariance matrix of the log smoothing parameters).

Value

Standard `logLik` object: see `logLik`.

Author(s)

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

References

Greven, S., and Kneib, T. (2010), On the Behaviour of Marginal and Conditional AIC in Linear Mixed Models, Biometrika, 97, 773-789.

Wood, S.N., N. Pya and B. Saefken (2016), Smoothing parameter and model selection for general smooth models (with discussion). Journal of the American Statistical Association 111, 1548-1575 doi:10.1080/01621459.2016.1180986

Wood S.N. (2017) Generalized Additive Models: An Introduction with R (2nd edition). Chapman and Hall/CRC Press. doi:10.1201/9781315370279

See Also

`AIC`

---

lp *Basic linear programming*

Description

Solves, for $\mathbf{x}$, linear programming problems in the standard form

$$\min \mathbf{c}^T\mathbf{x} \text{ s.t. } \mathbf{A}\mathbf{x} = \mathbf{b}, \quad \mathbf{b} \geq \mathbf{0}$$

or in the form

$$\min \mathbf{c}^T\mathbf{x} \text{ s.t. } \mathbf{A}\mathbf{x} \geq \mathbf{b}, \quad \mathbf{C}\mathbf{x} = \mathbf{d}$$

A lightweight implementation of the simplex method, designed for problems of modest size, not large scale problems requiring sparse methods.

`feasible` finds starting values meeting the constraints, which is also useful for finding feasible initial values for quadratic programming.

Usage

```
lp(c,A,b,C=NULL,d=NULL,Bi=NULL,maxit=max(1000, nrow(A) * 10), phase1 = FALSE)
feasible(A,b,C=NULL,d=NULL,maxit = max(1000, nrow(A) * 10))
```

Arguments

`c`	The vector defining the linear program objective function $\mathbf{c}^T\mathbf{x}$.
`A`	The constraint matrix.
`b`	The vector defining the r.h.s. of the constraint involving `A`.
`C`	`NULL` to signal a problem in standard form. Otherwise the matrix defining the equality constraints.
`d`	`NULL` to signal a problem in standard form. Otherwise the vector defining the r.h.s. of the equality constraint.
`Bi`	For a problem in standard form, index of the elements of $\mathbf{x}$ initially non-zero. $\mathbf{A}[,Bi]\mathbf{x}^* = \mathbf{b}$ must have a unique solution with $\mathbf{x}^* \geq \mathbf{0}$. `NULL` to have this set found automatically by solution of a phase 1 linear program.
`maxit`	Maximum number of iterations of the simplex method to allow.
`phase1`	signals that function is being called to solve a phase 1 problem.

Details

This code was written to provide a lightweight method for finding feasible initial coefficients for shape constrained splines, but is suitable for solving general linear programming problems of modest size. Given that it uses dense matrix computations, it is not suitable for large scale problems where it is important to exploit sparcity. Neither is it suitable for the sort of linear programming problems arising from integer programs, where degeneracy may be a substantial problem.

The function uses the simplex method (see e.g. Chapter 13 of Nocedal and Wright, 2006). Computational efficiency is ensured by QR decomposing $\mathbf{A}[,Bi]$ and then efficiently updating the decomposition, every time the active set `Bi` changes, using Givens based updating schemes broadly similar to those given in Golub and van Loan (2013) 5.1.3 p240 and 6.5.3 p337. Given the QR factorization solution of systems involving $\mathbf{A}[,Bi]$ is efficient.

Value

A vector. Either the solution of the problem (`lp`), or a feaible initial vector (`feasible`). Produces an error if there is no solution or `maxit` is exceeded.

WARNINGS

Not designed for large scale problems requiring sparse methods, nor for problems where significant degeneracy is expected.

Author(s)

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

References

Golub G.H. and C.F. van Loan (2013) Matrix Computations (4th edition) Johns Hopkins

Nocedal J. and S. Wright (2006) Numerical Optimization (2nd edition) Springer

See Also

pcls

Examples

```
library(mgcv)
## very simple linear program...
c <- c(-4,-2,0,0)
A <- matrix(c(1,2,1,.5,1,0,0,1),2,4)
b <- c(5,8)
x <- lp(c,A,b,c(3,4));sum(c*x);x ## Bi given

x <- lp(c,A,b);sum(c*x);x ## Bi found automatically

## equivalent formulation Ax>b
A <- -matrix(c(1,2,1,.5),2,2)
b <- -c(5,8)
C <- matrix(0,0,2); d <- numeric(0)
c <- c(-4,-2)
x <- lp(c,A,b,C=C,d=d);sum(c*x);x

## equivalent formulation Ax>b, Cx=d
c <- c(-4,-2,0)
A <- matrix(c(0,-2,0,-.5,1,0),2,3)
C <- matrix(c(1,1,1),1,3); d <- 5
b <- c(0,-8)
x <- lp(c,A,b,C=C,d=d);sum(c*x);x
```

---

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

x	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>

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)
```

---

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 multi-dimensions, 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

```
magic(y,X,sp,S,off,L=NULL,lsp0=NULL,rank=NULL,H=NULL,C=NULL,
      w=NULL,gamma=1,scale=1,gcv=TRUE,ridge.parameter=NULL,
      control=list(tol=1e-6,step.half=25,rank.tol=
      .Machine$double.eps^0.5),extra.rss=0,n.score=length(y),nthreads=1)
```

Arguments

`y`	is the response data vector.
`X`	is the model matrix (more columns than rows are allowed).
`sp`	is the array of smoothing parameters. The vector `L%*%log(sp) + lsp0` 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) + lsp0`. 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.

`H` 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 $\mathbf{b}$ is the parameter vector then the parameters are forced to satisfy $\mathbf{Cb} = \mathbf{0}$.

`w` the regression weights. If this is a matrix then it is taken as being the square root of the inverse of the covariance matrix of `y`, specifically $\mathbf{V}_y^{-1} = \mathbf{w}'\mathbf{w}$. 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.

`scale` is the scale parameter for use with UBRE.

`gcv` should be set to `TRUE` if GCV is to be used, `FALSE` for UBRE.

`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.

`extra.rss` is a constant to be added to the residual sum of squares (squared norm) term in 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 sets.

`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.

`nthreads` `magic` can make use of multiple threads if this is set to >1.

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:

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

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

The θ_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 γ is `gamma` the inflation factor for degrees of freedom (usually set to 1) and ϕ is `scale`, the scale parameter. $\mathbf{A}$ 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 $\mathbf{A}$.

The method operates by Newton or steepest descent updates of the logs of the θ_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 θ_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 $\mathbf{b}$ 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

See Also

magic.post.proc,gam

Examples

```
## Use `magic' for a standard additive model fit ...
   library(mgcv)
   set.seed(1);n <- 200;sig <- 1
   dat <- gamSim(1,n=n,scale=sig)
```

```
   k <- 30
## set up additive model
   G <- gam(y~s(x0,k=k)+s(x1,k=k)+s(x2,k=k)+s(x3,k=k),fit=FALSE,data=dat)
## fit using magic (and gam default tolerance)
   mgfit <- magic(G$y,G$X,G$sp,G$S,G$off,rank=G$rank,
                  control=list(tol=1e-7,step.half=15))
## and fit using gam as consistency check
   b <- gam(G=G)
   mgfit$sp;b$sp  # compare smoothing parameter estimates
   edf <- magic.post.proc(G$X,mgfit,G$w)$edf # get e.d.f. per param
   range(edf-b$edf)  # compare

## p>n example... fit model to first 100 data only, so more
## params than data...

   mgfit <- 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

## constrain first two smooths to have identical smoothing parameters
   L <- diag(3);L <- rbind(L[1,],L)
   mgfit <- 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)))
    Cv <- chol(V)  # t(Cv)%*%Cv=V
## Simulate AR1 errors ...
    e <- t(Cv)%*%rnorm(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~s(x,k=20))
    plot(b);lines(x,f-mean(f),col=2);title("Ignoring correlation")
## Fit smooth, taking account of *known* correlation...
    w <- solve(t(Cv)) # V^{-1} = w'w
    ## Use `gam' to set up model for fitting...
    G <- gam(y~s(x,k=20),fit=FALSE)
    ## fit using magic, with weight *matrix*
    mgfit <- 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)
    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 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` (ϕ, say) which can be used to obtain the require quantities as follows. The Bayesian covariance matrix of the parameters is $\mathbf{VV}'\phi$. The vector of estimated degrees of freedom for each parameter is the leading diagonal of $\mathbf{VV}'\mathbf{X}'\mathbf{W}'\mathbf{WX}$ where $\mathbf{W}$ is either the weight matrix `w` or the matrix `diag(w)`. The hat/influence matrix is given by $\mathbf{WXVV}'\mathbf{X}'\mathbf{W}'$.

The frequentist parameter estimator covariance matrix is $\mathbf{VV}'\mathbf{X}'\mathbf{W}'\mathbf{WXVV}'\phi$: 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 parameter.

Author(s)

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

See Also

magic

---

mchol *Sparse* chol *function*

Description

A wrapper for the Cholesky function from the Matrix package that produces the sparse pivoted Cholesky factor of a positive definite matrix, and returns the pivot vector for this as an attribute. The Matrix package chol function no longer retuns the pivots.

Usage

```
mchol(A)
```

Arguments

A	A sparse matrix suitable for passing to Cholesky function from the Matrix package.

Details

The Matrix version of chol performs sparse Cholesky decomposition with sparsity maintaining pivoting, but no longer returns the pivot vector, rendering the returned factor useless for many purposes. This wrapper function simply fixes this. It also ensures that there are no numerical zeroes below the leading diagonal (not the default behaviour of expand1, which can put some numerical zeroes there, in place of structural ones, at least at version 1.6-5). Calls chol(A,pivot=TRUE) if A inherits from class matrix.

Value

If A is positive definite then the upper triangular Cholesky factor matrix, with "pivot" attribute and "rank" attribute which is the diminsion of A. Otherwise -1 with "rank" attribute -1.

Author(s)

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

Examples

```
library(mgcv)
## A sparse +ve def matrix
u <- sample(1:100,10)*.001
x <- i <- j <- 1:10
ii <- sample(1:10,10,replace=TRUE);
jj <- sample(1:10,10,replace=TRUE)
x <- c(x,u,u);
```

```
i <- c(i,ii,jj)
j <- c(j,jj,ii)
A <- Matrix::sparseMatrix(i=i,j=j,x=x)
R <- mchol(A)
piv <- attr(R,"pivot")
range(crossprod(R)-A[piv,piv])

i <- sample(1:5,10,replace=TRUE)
j <- sample(1:10,10,replace=TRUE)
u <- sample(1:100,10)*.001
A <- crossprod(Matrix::sparseMatrix(i=i,j=j,x=u))
mchol(A)
```

---

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 5 of Wood (2017). 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. **An example from an mgcv helpfile gives an error - is this a bug?** It might be, but first please check that the version of mgcv you have loaded into R corresponds to the version from which the helpfile came. Many such problems are caused by trying to run code only supported in a later mgcv version in an earlier version. Another possibility is that you have an object

loaded whose name clashes with an mgcv function (for example you are trying to use the mgcv `multinom` function, but have another object called `multinom` loaded.)

6. **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` for details.

7. **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.

8. **When using `gamm` or `gamm4`, the reported AIC is different for the `gam` object and the `lme` or `lmer` 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.

9. **What does 'mgcv' stand for?** '**M**ixed **G**AM **C**omputation **V**ehicle', 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 '**M**ad **G**AM **C**omputing **V**ulture'.

10. **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.

Wood S.N. (2017) Generalized Additive Models: An Introduction with R (2nd edition). Chapman and Hall/CRC Press.

`mgcv.package` *Mixed GAM Computation Vehicle with GCV/AIC/REML/NCV smoothness estimation and GAMMs by REML/PQL*

Description

mgcv provides functions for generalized additive modelling (gam and bam) and generalized additive mixed modelling (gamm, and random.effects), including location scale and shape extensions. Shape constrained models can be estimated using scasm. 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. In addition to the main modelling functions, jagam provided facilities to ease the set up of models for use with JAGS, while ginla provides marginal inference via a version of Integrated Nested Laplace Approximation.

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, NCV 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. bam(...,discrete=TRUE) offers the very efficient methods of Wood et al. (2017) and Li and Wood (2020).

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). See random.effects for including random effects in models estimated by gam, bam or scasm.

In addition, low level routine magic can fit models to data with a known correlation structure.

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

https://webhomes.maths.ed.ac.uk/~swood34/

These provide details for the underlying mgcv methods, and fuller references to the large literature on which the methods are based.

Wood, S.N. (2025) Generalized Additive Models. Annual Review of Statistics and Its Applications 12:497-526 doi:10.1146/annurevstatistics112723034249

Wood, S. N. (2020) Inference and computation with generalized additive models and their extensions. Test 29(2): 307-339. doi:10.1007/s11749020007115

Wood, S.N., N. Pya and B. Saefken (2016), Smoothing parameter and model selection for general smooth models (with discussion). Journal of the American Statistical Association 111, 1548-1575 doi:10.1080/01621459.2016.1180986

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 doi:10.1093/biomet/ast038

Wood, S.N. (2013b) On p-values for smooth components of an extended generalized additive model. Biometrika 100:221-228 doi:10.1093/biomet/ass048

Wood, S.N. (2017) *Generalized Additive Models: an introduction with R (2nd edition)*, CRC doi:10.1201/9781315370279

Wood, S.N., Li, Z., Shaddick, G. & Augustin N.H. (2017) Generalized additive models for gigadata: modelling the UK black smoke network daily data. Journal of the American Statistical Association. 112(519):1199-1210 doi:10.1080/01621459.2016.1195744

Li, Z & S.N. Wood (2020) Faster model matrix crossproducts for large generalized linear models with discretized covariates. Statistics and Computing. 30:19-25 doi:10.1007/s11222019098642

Development of mgcv version 1.8 was part funded by EPSRC grants EP/K005251/1 and EP/I000917/1.

Examples

```
## see examples for gam, bam and gamm
```

`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`, with the 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.

When using NCV with gam worthwhile performance improvements are available by setting ncv.threads in gam.control.

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 scheduler 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 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://hpc-tutorials.llnl.gov/openmp/

Examples

```
## illustration of multi-threading with gam...

require(mgcv);set.seed(9)
dat <- gamSim(1,n=2000,dist="poisson",scale=.1)
k <- 12;bs <- "cr";ctrl <- list(nthreads=2)

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~s(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. Sample sizes are low
## here to keep example quick -- for such a small model
## little or no advantage is likely to be seen.
k <- 13;set.seed(9)
dat <- gamSim(1,n=6000,dist="poisson",scale=.1)

require(parallel)
nc <- 2   ## cluster size, set for example portability
if (detectCores()>1) { ## no point otherwise
  cl <- makeCluster(nc)
  ## could also use makeForkCluster, but read warnings first!
} else cl <- NULL

system.time(b3 <- bam(y ~ 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

if (!is.null(cl)) stopCluster(cl)
b3

## Alternative, better scaling example, using the discrete option with bam...

system.time(b4 <- bam(y ~ s(x0,bs=bs,k=7)+s(x1,bs=bs,k=7)+s(x2,bs=bs,k=k)
            ,data=dat,family=poisson(),discrete=TRUE,nthreads=2))
```

`mini.roots` *Obtain square roots of penalty matrices*

Description

INTERNAL function to obtain square roots, B[[i]], of the penalty matrices S[[i]]'s having as few columns as possible.

Usage

```
mini.roots(S, off, np, rank = NULL)
```

Arguments

S	a list of penalty matrices, in packed form.
off	a vector where the i-th element is the offset for the i-th matrix. The elements in columns 1:off[i] of B[[i]] will be equal to zero.
np	total number of parameters.
rank	here rank[i] is optional supplied rank of S[[i]]. Set rank[i] < 1, or rank=NULL to estimate.

Value

A list of matrix square roots such that S[[i]]=B[[i]]%*%t(B[[i]]).

Author(s)

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

---

missing.data *Missing data in GAMs*

Description

If there are missing values in the response 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.models for 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

Examples

```
## The example takes a couple of minutes to 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 <- 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
  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...

  dname <- names(dat)[2:5]
  dat1 <- dat
  for (i in 1:4) {
    by.name <- paste("m",dname[i],sep="")
    dat1[[by.name]] <- is.na(dat1[[dname[i]]])
    dat1[[dname[i]]][dat1[[by.name]]] <- mean(dat1[[dname[i]]],na.rm=TRUE)
    lev <- rep(1,n);lev[dat1[[by.name]]] <- 1:sum(dat1[[by.name]])
    id.name <- paste("id",dname[i],sep="")
    dat1[[id.name]] <- factor(lev)
    dat1[[by.name]] <- as.numeric(dat1[[by.name]])
  }

  ## 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 <- bam(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,discrete=TRUE)

  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
}
```

`model.matrix.gam` *Extract model matrix from GAM fit*

Description

Obtains the model matrix from a fitted `gam` object.

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`

Examples

```
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)
```

`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 $\{x_i, p_i : i = 1 \ldots n\}$. Then it is possible to find a relatively small set of linear constraints on $\mathbf{p}$ sufficient to ensure monotonicity (and bounds if required): $\mathbf{Ap} \geq \mathbf{b}$. 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.

See Also

magic, pcls

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 Cholesky 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 Cholesky decompositon, which is fast but tends to overestimate rank. "svd" to use singular value decomposition, which is slower, but is the most accurate way to estimate rank.

Details

The function is primarily of use for turning penalized regression problems into ordinary regression problems. Given that A is positive semi-definite the SVD option actually uses a symmetric eigen routine, which gives the same result more efficiently.

Value

A matrix, $\mathbf{B}$ with as many columns as the rank of $\mathbf{A}$, and such that $\mathbf{A} = \mathbf{B}\mathbf{B}'$.

Author(s)

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

Examples

```
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
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: see `formula.gam`). The first formula also specifies the response variable.

Usage

```
multinom(K=1)
```

Arguments

K	There are K+1 categories and K linear predictors.

Details

The model has K linear predictors, η_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 (2016).

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>, with a variance bug fix from Max Goplerud.

References

Wood, S.N., N. Pya and B. Saefken (2016), Smoothing parameter and model selection for general smooth models. Journal of the American Statistical Association 111, 1548-1575 doi:10.1080/01621459.2016.1180986

See Also

`ocat`

Examples

```
library(mgcv)
set.seed(6)
## simulate some data from a three class model
n <- 1000
f1 <- function(x) sin(3*pi*x)*exp(-x)
f2 <- function(x) x^3
f3 <- function(x) .5*exp(-x^2)-.2
f4 <- function(x) 1
x1 <- runif(n);x2 <- runif(n)
eta1 <- 2*(f1(x1) + f2(x2))-.5
eta2 <- 2*(f3(x1) + f4(x2))-1
p <- exp(cbind(0,eta1,eta2))
p <- p/rowSums(p) ## prob. of each category
cp <- t(apply(p,1,cumsum)) ## cumulative prob.
## 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 <- gam(list(y~s(x1)+s(x2),~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")
pc <- apply(pp,1,function(x) which(max(x)==x)[1])-1
plot(gr,col=pc+3,pch=19)

## example sharing a smoother between linear predictors
## ?formula.gam gives more details.
```

```
b <- gam(list(y~s(x1),~s(x1),1+2~s(x2)-1),family=multinom(K=2))
plot(b,pages=1)
```

---

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. Extra linear predictors may also be specified giving terms which are shared between components (see formula.gam). The Choleski factor of the response precision matrix is estimated as part of fitting.

Usage

```
mvn(d=2)
```

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 (2016), Smoothing parameter and model selection for general smooth models. Journal of the American Statistical Association 111, 1548-1575 doi:10.1080/01621459.2016.1180986

See Also

gaussian

Examples

```
library(mgcv)
## simulate some data...
V <- matrix(c(2,1,1,2),2,2)
f0 <- function(x) 2 * sin(pi * x)
f1 <- function(x) exp(2 * x)
f2 <- function(x) 0.2 * x^11 * (10 * (1 - x))^6 + 10 *
            (10 * x)^3 * (1 - x)^10
n <- 300
x0 <- runif(n);x1 <- runif(n);
x2 <- runif(n);x3 <- runif(n)
y <- matrix(0,n,2)
for (i in 1:n) {
  mu <- 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...

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
```

NCV *Neighbourhood Cross Validation*

Description

NCV estimates smoothing parameters by optimizing the average ability of a model to predict subsets of data when subsets of data are omitted from fitting. Usually the predicted subset is a subset of the omitted subset. If both subsets are the same single datapoint, and the average is over all datapoints, then NCV is leave-one-out cross validation. QNCV is a quadratic approximation to NCV, guaranteed finite for any family link combination.

In detail, suppose that a model is estimated by minimizing a penalized loss

$$\sum_i D(y_i, \theta_i) + \sum_j \lambda_j \beta^\mathsf{T} S_j \beta$$

where D is a loss (such as a negative log likelihood), dependent on response y_i and parameter vector θ_i, which in turn depends on covariates via one or more smooth linear predictors with coefficients

β. The quadratic penalty terms penalize model complexity: S_j is a known matrix and λ_j an unknown smoothing parameter. Given smoothing parameters the penalized loss is readily minimized to estimate β.

The smoothing parameters also have to be estimated. To this end, choose $k = 1, \ldots, m$ subsets $\alpha(k) \subset \{1, \ldots, n\}$ and $\delta(k) \subset \{1, \ldots, n\}$. Usually $\delta(k)$ is a subset of (or equal to) $\alpha(k)$. Let $\theta_i^{\alpha(k)}$ denote the estimate of θ_i when the points indexed by $\alpha(k)$ are omitted from fitting. Then the NCV criterion

$$V = \sum_{k=1}^{m} \sum_{i \in \delta(k)} D(y_i, \theta_i^{\alpha(k)})$$

is minimized w.r.t. the smoothing parameters, λ_j. If $m = n$ and $\alpha(k) = \delta(k) = k$ then ordinary leave-one-out cross validation is recovered. This formulation covers many of the variants of cross validation reviewed in Arlot and Celisse (2010), for example.

Except for a quadratic loss, V can not be computed exactly, but it can be computed to $O(n^{-2})$ accuracy (fixed basis size), by taking single Newton optimization steps from the full data β estimates to the equivalent when each $\alpha(k)$ is dropped. This is what `mgcv` does. The Newton steps require update of the full model Hessian to the equivalent when each datum is dropped. This can be achieved at $O(p^2)$ cost, where p is the dimension of β. Hence, for example, the ordinary cross validation criterion is computable at the $O(np^2)$ cost of estimating the model given smoothing parameters.

The NCV score computed in this way is optimized using a BFGS quasi-Newton method, adapted to the case in which smoothing parameters tending to infinity may cause indefiniteness.

For `bam(...,discrete=TRUE)` NCV can be used to estimate the smoothing parameters of the working penalized weighted linear model. This is typically very expensive relative to REML estimation, but the cost can be mitigated by basing the NCV criterion on a sample (usually random of the neighbourhoods). Note that in the case in which leave-out-neighbourhood CV is used to reduce the effects of autocorrelation, it is important to supply a neighbourhood for each observation, and to supply a `sample` element in `nei`. This ensures that the parameter covariance matrix is estimated correctly.

Spatial and temporal short range autocorrelation

A routine applied problem is that smoothing parameters tend to be underestimated in the presence of un-modelled short range autocorrelation, as the smooths try to fit the local excursions in the data caused by the local autocorrelation. Cross validation will tend to 'fit the noise' when there is autocorellation, since a model that fits the noise in the data correlated with an omitted datum, will also tend to closely fit the noise in the omitted datum, because of the correlation. That is autocorrelation works against the avoidance of overfit that cross validation seeks to achieve.

For short range autocorrelation the problems can be avoided, or at least mitigated, by predicting each datum when all the data in its 'local' neighbourhood are omitted. The neighbourhoods being constructed in order that un-modelled correlation is minimized between the point of interest and points outside its neighbourhood. That is we set $m = n$, $\delta(k) = k$ and $\alpha(k) = \texttt{nei}(k)$, where `nei(k)` are the indices of the neighbours of point k. This approach has been known for a long time (e.g. Chu and Marron, 1991; Robert et al. 2017), but was previously rather too expensive for regular use for smoothing parameter estimation.

Specifying the neighbourhoods

The neighbourhood subsets $\alpha(k)$ and $\delta(k)$ have to be supplied to gam, and the nei argument does this. It is a list with the following arguments.

- a is the vector of indices to be dropped for each neighbourhood.
- ma gives the end of each neighbourhood. So nei$a[(nei$ma[j-1]+1):nei$ma[j]] gives the points dropped for the neighbourhood j: that is $\alpha(j)$.
- d is the vector of indices of points to predict.
- md gives the corresponding endpoints md. So nei$d[(nei$md[j-1]+1):nei$md[j]] indexes the points to predict for neighbourhood j: that is $\delta(j)$.
- sample is an optional element used by bam. If it is a single number it gives the number of neighbourhoods to randomly sample to construct the NCV criterion. If it is a vector then it should contain the indices of the neighbourhoods to use.
- jackknife is an optional element used by gam. If supplied and TRUE then variance estimates are based on the raw Jackkife estimate, if FALSE then on the standard Bayesian results. If not supplied (usual) then an estimator accounting for the neighbourhood structure is used, that largely accounts for any correlation present within neighbourhoods. jackknife is ignored if NCV is being calculated for a model where another method is used for smoothing parameter selection.

If nei==NULL (or a or ma are missing) then leave-one-out cross validation is used. If nei is supplied but NCV is not selected as the smoothing parameter estimation method, then it is simply computed (but not optimized).

Numerical issues

If a model is specified in which some coefficient values, β, have non-finite likelihood then the NCV criterion computed with single Newton steps could also be non-finite. A simple fix replaces the NCV criterion with a quadratic approximation to the criterion around the full data fit. The quadratic approximation is always finite. This 'QNCV' is essential for some families, such as gevlss. QNCV is not needed for bam estimation.

Although the leading order cost of NCV is the same as REML or GCV, the actual cost is higher because the dominant operations costs are in matrix-vector, rather than matrix-matrix, operations, so BLAS speed ups are small. However multi-core computing is worthwhile for NCV. See the option ncv.threads in gam.control.

Author(s)

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

References

Chu and Marron (1991) Comparison of two bandwidth selectors with dependent errors. The Annals of Statistics. 19, 1906-1918

Arlot, S. and A. Celisse (2010). A survey of cross-validation procedures for model selection. Statistics Surveys 4, 40-79

Roberts et al. (2017) Cross-validation strategies for data with temporal, spatial, hierarchical, or phylogenetic structure. Ecography 40(8), 913-929.

Wood S.N. (2026) On Neighbourhood Cross Validation. Statistical Science (in press). https://arxiv.org/abs/2404.16490

Examples

```
require(mgcv)
nei.cor <- function(h,n) { ## construct nei structure
  nei <- list(md=1:n,d=1:n)
  nei$ma <- cumsum(c((h+1):(2*h+1),rep(2*h+1,n-2*h-2),(2*h+1):(h+1)))
  a0 <- rep(0,0); if (h>0) for (i in 1:h) a0 <- c(a0,1:(h+i))
  a1 <- n-a0[length(a0):1]+1
  nei$a <- c(a0,1:(2*h+1)+rep(0:(n-2*h-1),each=2*h+1),a1)
  nei
}
set.seed(1)
n <- 500;sig <- .6
x <- 0:(n-1)/(n-1)
f <- sin(4*pi*x)*exp(-x*2)*5/2
e <- rnorm(n,0,sig)
for (i in 2:n) e[i] <- 0.6*e[i-1] + e[i]
y <- f + e ## autocorrelated data
nei <- nei.cor(4,n) ## construct neighbourhoods to mitigate
b0 <- gam(y~s(x,k=40)) ## GCV based fit
gc <- gam.control(ncv.threads=2)
b1 <- gam(y~s(x,k=40),method="NCV",nei=nei,control=gc)
## use "QNCV", which is identical here...
b2 <- gam(y~s(x,k=40),method="QNCV",nei=nei,control=gc)
## plot GCV and NCV based fits...
f <- f - mean(f)
par(mfrow=c(1,2))
plot(b0,rug=FALSE,scheme=1);lines(x,f,col=2)
plot(b1,rug=FALSE,scheme=1);lines(x,f,col=2)
```

---

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. θ 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, and is deprecated). 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"

Details

nb allows estimation of the theta parameter alongside the model smoothing parameters, but is only usable with gam or bam (not 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 is deprecated and should only be used with performance iteration estimation (see gam argument optimizer), in which case the method of estimation is to choose $\hat{\theta}$ 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 θ 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 θ estimate is updated. This is done by conditioning on all components of the current GCV/Pearson estimator of the scale parameter except θ 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 does 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

Venables, B. and B.R. Ripley (2002) Modern Applied Statistics in S, Springer.

Wood, S.N., N. Pya and B. Saefken (2016), Smoothing parameter and model selection for general smooth models. Journal of the American Statistical Association 111, 1548-1575 doi:10.1080/01621459.2016.1180986

Examples

```
library(mgcv)
set.seed(3)
n<-400
dat <- gamSim(1,n=n)
g <- exp(dat$f/5)

## negative binomial data...
dat$y <- rnbinom(g,size=3,mu=g)
## known theta fit ...
b0 <- gam(y~s(x0)+s(x1)+s(x2)+s(x3),family=negbin(3),data=dat)
plot(b0,pages=1)
print(b0)

## same with theta estimation...
b <- 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


## another example...
set.seed(1)
f <- dat$f
f <- f - min(f)+5;g <- f^2/10
dat$y <- rnbinom(g,size=3,mu=g)
b2 <- 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 lme, 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 neccesary 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

Value

A name that is not in old.names.

Author(s)

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

See Also

gamm

Examples

```
require(mgcv)
old <- c("a","tuba","is","tubby")
new.name("tubby",old)
```

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)
```

Arguments

x	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>

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="l")
lines(x,exp(x),col=2)
plot(x,log(notExp(x)),type="l")
lines(x,log(exp(x)),col=2) # redundancy intended
x <- x/4
plot(x,notExp(x),type="l")
lines(x,exp(x),col=2)
plot(x,log(notExp(x)),type="l")
lines(x,log(exp(x)),col=2) # redundancy intended
par(op)
range(notLog(notExp(x))-x) # show that inverse works!
```

---

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.

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 'effetive 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 lme 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 lme 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>

See Also

pdTens, pdIdnot, gamm

Examples

```
## Illustrate the notExp2 function:
require(mgcv)
x <- seq(-50,50,length=1000)
op <- par(mfrow=c(2,2))
plot(x,notExp2(x),type="l")
lines(x,exp(x),col=2)
plot(x,log(notExp2(x)),type="l")
lines(x,log(exp(x)),col=2) # redundancy intended
x <- x/4
plot(x,notExp2(x),type="l")
lines(x,exp(x),col=2)
plot(x,log(notExp2(x)),type="l")
lines(x,log(exp(x)),col=2) # redundancy intended
par(op)
```

---

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.

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

See Also

tprs

Examples

```
require(mgcv)
null.space.dimension(2,0)
```

---

ocat *GAM ordered categorical family*

Description

Family for use with gam or bam, 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.

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 (2016), Smoothing parameter and model selection for general smooth models. Journal of the American Statistical Association 111, 1548-1575 doi:10.1080/01621459.2016.1180986

Examples

```
library(mgcv)
## Simulate some ordered categorical data...
set.seed(3);n<-400
dat <- gamSim(1,n=n)
dat$f <- dat$f - mean(dat$f)

alpha <- c(-Inf,-1,0,5,Inf)
R <- length(alpha)-1
y <- dat$f
u <- runif(n)
u <- dat$f + log(u/(1-u))
for (i in 1:R) {
  y[u > alpha[i]&u <= 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 <- gam(y~s(x0)+s(x1)+s(x2)+s(x3),family=ocat(R=R),data=dat)
b
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)
```

`one.se.rule` *The one standard error rule for smoother models*

Description

The 'one standard error rule' (see e.g. Hastie, Tibshirani and Friedman, 2009) is a way of producing smoother models than those directly estimated by automatic smoothing parameter selection methods. In the single smoothing parameter case, we select the largest smoothing parameter within one standard error of the optimum of the smoothing parameter selection criterion. This approach can be generalized to multiple smoothing parameters estimated by REML or ML.

Details

Under REML or ML smoothing parameter selection an asyptotic distributional approximation is available for the log smoothing parameters. Let ρ denote the log smoothing parameters that we want to increase to obtain a smoother model. The large sample distribution of the estimator of ρ is $N(\rho, V)$ where V is the matrix returned by `sp.vcov`. Drop any elements of ρ that are already at 'effective infinity', along with the corresponding rows and columns of V. The standard errors of the log smoothing parameters can be obtained from the leading diagonal of V. Let the vector of these be d. Now suppose that we want to increase the estimated log smoothing parameters by an amount αd. We choose α so that $\alpha d^T V^{-1} d = \sqrt{2p}$, where p is the dimension of d and 2p the variance of a chi-squared r.v. with p degrees of freedom.

The idea is that we increase the log smoothing parameters in proportion to their standard deviation, until the RE/ML is increased by 1 standard deviation according to its asypmtotic distribution.

Author(s)

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

References

Hastie, T, R. Tibshirani and J. Friedman (2009) The Elements of Statistical Learning 2nd ed. Springer.

See Also

gam

Examples

```
require(mgcv)
set.seed(2) ## simulate some data...
dat <- gamSim(1,n=400,dist="normal",scale=2)
b <- gam(y~s(x0)+s(x1)+s(x2)+s(x3),data=dat,method="REML")
b
## only the first 3 smoothing parameters are candidates for
## increasing here...
V <- sp.vcov(b)[1:3,1:3] ## the approx cov matrix of sps
d <- diag(V)^.5          ## sp se.
## compute the log smoothing parameter step...
d <- sqrt(2*length(d))/d
sp <- b$sp ## extract original sp estimates
sp[1:3] <- sp[1:3]*exp(d) ## apply the step
## refit with the increased smoothing parameters...
b1 <- gam(y~s(x0)+s(x1)+s(x2)+s(x3),data=dat,method="REML",sp=sp)
b;b1 ## compare fits
```

---

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

M 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.
- **C** Matrix containing any linear equality constraints on the problem (e.g. $\mathbf{C}$ in $\mathbf{Cp} = \mathbf{c}$). If you have no equality constraints initialize this to a zero by zero matrix. Note that there is no need to supply the vector $\mathbf{c}$, it is defined implicitly by the initial parameter estimates $\mathbf{p}$.

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 (unless these are constraints in `active`).

Ain Matrix for the inequality constraints $\mathbf{A}_{in}\mathbf{p} > \mathbf{b}_{in}$.

bin vector in the inequality constraints.

active if present this indexes any inequality constraints currently active. Typically only used in applications calling `pcls` sequentially. Must never exceed dimension of `p` minus the number of fixed constraints.

Details

This solves the problem:

$$minimise\ \|\mathbf{W}^{1/2}(\mathbf{X}\mathbf{p} - \mathbf{y})\|^2 + \sum_{i=1}^{m} \lambda_i \mathbf{p}'\mathbf{S}_i\mathbf{p}$$

subject to constraints $\mathbf{C}\mathbf{p} = \mathbf{c}$ and $\mathbf{A}_{in}\mathbf{p} > \mathbf{b}_{in}$, w.r.t. $\mathbf{p}$ given the smoothing parameters λ_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 semi-definite matrix of coefficients defining the ith penalty and $\mathbf{C}$ a matrix of coefficients defining the linear equality constraints on the problem. The smoothing parameters are the λ_i. Note that $\mathbf{X}$ must be of full column rank, at least when projected into the null space of any equality constraints. $\mathbf{A}_{in}$ is a matrix of coefficients defining the inequality constraints, while $\mathbf{b}_{in}$ 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. $\mathbf{X}'\mathbf{X}$ is not formed explicitly. See Gill et al. 1981.

Value

The function returns a vector of the estimated parameter values. This has an attribute `active` giving the indices of the active constraints. If none are active this attribute will be of length 0.

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

See Also

magic, mono.con

Examples

```
require(mgcv)
# first an un-penalized example - fit E(y)=a+bx subject to a>0
set.seed(0)
n <- 100
x <- runif(n); y <- x - 0.2 + rnorm(n)*0.1
M <- 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$Ain[1,] <- c(1,0)
pcls(M) -> M$p
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 <- runif(100)*4-1;x <- sort(x);
f <- exp(4*x)/(1+exp(4*x)); y <- f+rnorm(100)*0.1; plot(x,y)
dat <- data.frame(x=x,y=y)
# Show regular spline fit (and save fitted object)
f.ug <- gam(y~s(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]]
F <- mono.con(sm$xp);   # get constraints
G <- list(X=sm$X,C=matrix(0,0,0),sp=f.ug$sp,p=sm$xp,y=y,w=y*0+1)
G$Ain <- F$A;G$bin <- F$b;G$S <- sm$S;G$off <- 0

p <- pcls(G);  # fit spline (using s.p. from unconstrained fit)

fv<-Predict.matrix(sm,data.frame(x=x))%*%p
lines(x,fv,col=2)

# now a tprs example of the same thing....

f.ug <- gam(y~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]]
xc <- 0:39/39 # points on [0,1]
nc <- length(xc)  # number of constraints
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 <- 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
```

```
G$p <- rep(0,10); G$p[10] <- 0.1
# ... monotonic start params, got by setting coefs of polynomial part
p <- pcls(G);  # fit spline (using s.p. from unconstrained fit)

fv2 <- Predict.matrix(sm,data.frame(x=x))%*%p
lines(x,fv2,col=3)

#####################################
## monotonic additive model example...
#####################################

## First simulate data...

set.seed(10)
f1 <- function(x) 5*exp(4*x)/(1+exp(4*x));
f2 <- function(x) {
  ind <- x > .5
  f <- x*0
  f[ind] <- (x[ind] - .5)^2*10
  f
}
f3 <- 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)
mu <- f1(x) + f2(z) + f3(v)
y <- mu + rnorm(n)

## Preliminary unconstrained gam fit...
G <- gam(y~s(x)+s(z)+s(v,k=20),fit=FALSE)
b <- 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),
                  v=rep(.5,100))
pd1 <- data.frame(x=seq(0,1,length=100)+eps,z=rep(.5,100),
                  v=rep(.5,100))
X0 <- predict(b,newdata=pd0,type="lpmatrix")
X1 <- predict(b,newdata=pd1,type="lpmatrix")
Xx <- (X1 - X0)/eps ## Xx %*% coef(b) must be positive
pd0 <- data.frame(z=seq(0,1,length=100),x=rep(.5,100),
                  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")
X1 <- predict(b,newdata=pd1,type="lpmatrix")
Xz <- (X1-X0)/eps
G$Ain <- rbind(Xx,Xz) ## inequality constraint matrix
G$bin <- rep(0,nrow(G$Ain))
G$C = matrix(0,0,ncol(G$X))
G$sp <- b$sp
```

```
G$p <- coef(b)
G$off <- G$off-1 ## to match what pcls is expecting
## force inital parameters to meet constraint
G$p[11:18] <- G$p[2:9]<- 0
p <- pcls(G) ## constrained fit
par(mfrow=c(2,3))
plot(b) ## original fit
b$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 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

```
pdIdnot(value = numeric(0), form = NULL,
       nam = NULL, data = sys.frame(sys.parent()))
```

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	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.

See Also

te, pdTens, notLog2, gamm

Examples

```
# see gamm
```

---

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

```
pdTens(value = numeric(0), form = NULL,
       nam = NULL, data = sys.frame(sys.parent()))
```

Arguments

value	Initialization values for parameters. Not normally used.
form	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.
nam	a names argument, not normally used with this class.
data	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).

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.

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

x	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>

See Also

t2

Examples

```
require(mgcv)
set.seed(20)
dat <- gamSim(1,n=400,scale=2) ## simulate data
## following `t2' smooth basically separates smooth
## of x0,x1 into main effects + interaction....

b <- gam(y~t2(x0,x1,bs="tp",m=1,k=7)+s(x2)+s(x3),
         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 <- gam(y~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>

See Also

smooth.construct.cc.smooth.spec

Examples

```
require(mgcv)
x<-runif(30)
place.knots(x,7)
rm(x)
```

---

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 derivatives of 1-D plots may be plotted. Optionally produces term plots for parametric model components as well.

Usage

```
## S3 method for class 'gam'
plot(x,residuals=FALSE,rug=NULL,se=TRUE,pages=0,select=NULL,scale=-1,
         n=100,n2=40,n3=3,theta=30,phi=30,jit=FALSE,xlab=NULL,
         ylab=NULL,main=NULL,ylim=NULL,xlim=NULL,too.far=0.1,
         all.terms=FALSE,shade.col="gray80",shift=0,
         trans=I,seWithMean=FALSE,unconditional=FALSE,by.resids=FALSE,
         scheme=0,deriv=FALSE,...)
```

Arguments

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 (square root) IRLS weights, in order that they have constant variance if the model is correct. 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 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. The default of NULL sets rug to TRUE when the dataset size is <= 10000 and FALSE otherwise.

se	when TRUE (default) upper and lower lines are added to the 1-d plots corresponding to 95% confidence limits. For 2-d plots, surfaces corresponding to lower and upper limits of 68% confidence intervals are contoured and overlayed on the contour plot for the estimate. If one or more numbers (between 0 and 1) are supplied then these are the confidence level used whatever the dimension. Currently scheme 2 uses 2 levels. Any value 1 or above resets to TRUE, failing which any value 0 or below rests to FALSE.
pages	(default 0) the number of pages over which to spread the output. For example, if pages=1 then all terms will be plotted on one page with the layout performed automatically. Set to 0 to have the routine leave all graphics settings as they are.
select	Allows the plot for a single model term to be selected for printing. e.g. if you just want the plot for the second smooth term set select=2.
scale	set to -1 (default) to have the same y-axis scale for each plot, and to 0 for a different y axis for each plot. Ignored if ylim supplied.
n	number of points used for each 1-d plot - for a nice smooth plot this needs to be several times the estimated degrees of freedom for the smooth. Default value 100.
n2	Square root of number of points used to grid estimates of 2-d functions for contouring.
n3	Square root of number of panels to use when displaying 3 or 4 dimensional functions.
theta	One of the perspective plot angles.
phi	The other perspective plot angle.
jit	Set to TRUE if you want rug plots for 1-d terms to be jittered.
xlab	If supplied then this will be used as the x label for all plots.
ylab	If supplied then this will be used as the y label for all plots.
main	Used as title (or z axis label) for plots if supplied.
ylim	If supplied then this pair of numbers are used as the y limits for each plot.
xlim	If supplied then this pair of numbers are used as the x limits for each plot.
too.far	If greater than 0 then this is used to determine when a location is too far from data to be plotted when plotting 2-D smooths. This is useful since smooths tend to go wild away from data. The data are scaled into the unit square before deciding what to exclude, and too.far is a distance within the unit square. 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. Also see warnings.
shade.col	define the color used for shading confidence bands, for schemes that use this. For schemes using 2 colours this should be a 2 element vector of colours to over-ride defaults (rgb can be useful).
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	monotonic function to apply to each smooth (after any shift), before plotting. Monotonicity is not checked, but default plot limits assume it. 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. If seWithMean=2 then the intervals include the uncertainty in the mean of the fixed effects (but not in the mean of any uncentred smooths or random effects). 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.
deriv	set to TRUE to plot derivative of smooth w.r.t covariate for 1-D terms supporting this.
...	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 1-D smooths with supporting plot methods deriv=TRUE specifies plotting of derivative of the smooth w.r.t. the covariate and its confidence band. Such derivatives are the direct analogues of the coefficients in a linear regression, giving the rate of change of the linear predictor w.r.t. the covariate. The plot label is modified to indicate that the derivative is being plotted.

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. scheme == 2 shows 68% and 95% confidence regions (levels can be reset by se).

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 for 68% confidence limits. If se is a positive number in (0,1) then contour plots are at the estimate and confidence limits at the giuven level. 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 and scheme==3 a greyscale heatmap (contour.col controls the contour colour).

Smooths of 3 and 4 variables are displayed as tiled heatmaps with overlaid contours. In the 3 variable case the third variable is discretized and a contour plot of the first 2 variables is produced for each discrete value. The panels in the lower and upper rows are labelled with the corresponding third variable value. The lowest value is bottom left, and highest at top right. For 4 variables, two of the variables are coarsely discretized and a square array of image plots is produced for each combination of the discrete values. The first two arguments of the smooth are the ones used for the image/contour plots, unless a tensor product term has 2D marginals, in which case the first 2D marginal is image/contour plotted. n3 controls the number of panels. See also 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). There are two possible variants of this approach. In the default variant the extra uncertainty is in the mean of all other terms in the model (fixed and random, including uncentred smooths). Alternatively, if seWithMean=2 then only the uncertainty in parametric fixed effects is included in the extra uncertainty (this latter option actually tends to lead to wider intervals when the model contains random effects).

Several smooth plots methods using image will accept an hcolors argument, which can be anything documented in heat.colors (in which case something like hcolors=rainbow(50) is appropriate), or the grey function (in which case somthing like hcolors=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.

Plots of 2-D smooths with standard error contours shown can not easily be customized.

`all.terms` uses `termplot` which looks for the original data in the environment of the fitted model object formula. Since `gam` resets this environment to avoid large saved model objects containing data in hidden environments, this can fail.

Author(s)

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

Henric Nilsson <henric.nilsson@statisticon.se> 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. (2017) Generalized Additive Models: An Introduction with R (2nd edition). Chapman and Hall/CRC Press.

See Also

`gam`, `predict.gam`, `vis.gam`

Examples

```
library(mgcv)
set.seed(0)
## fake some data...
f1 <- function(x) {exp(2 * x)}
f2 <- function(x) {
  0.2*x^11*(10*(1-x))^6+10*(10*x)^3*(1-x)^10
}
f3 <- 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 <- 2*x0 + f1(x1) + f2(x2) + f3(x3) + e
x0 <- factor(x0)

## fit and plot...
```

```
b<-gam(y~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
plot(b,pages=1,scale=0,scheme=2,deriv=TRUE) ## plot derivs of smooths

## just parametric term alone...
termplot(b,terms="x0",se=TRUE)

## more use of color...
op <- par(mfrow=c(2,2),bg="blue")
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="")))
  fx <- 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~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))
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)

## 3 and 4 D smooths can also be plotted
dat <- gamSim(1,n=400)
b1 <- gam(y~te(x0,x1,x2,d=c(1,2),k=c(5,15))+s(x3),data=dat)

## Now plot. Use cex.lab and cex.axis to control axis label size,
## n3 to control number of panels, n2 to control panel grid size,
## scheme=1 to get greyscale...

plot(b1,pages=1)
```

`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

pc	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

In most cases essentially a wrapper for predict.gam for prediction from a model fitted by bam. Can compute on a parallel cluster. For models fitted using discrete methods with discrete=TRUE then discrete prediction methods are used instead.

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.

Usage

```
## S3 method for class 'bam'
predict(object,newdata,type="link",se.fit=FALSE,terms=NULL,
        exclude=NULL,block.size=50000,newdata.guaranteed=FALSE,
       na.action=na.pass,cluster=NULL,discrete=TRUE,n.threads=1,gc.level=0,...)
```

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 predictions are required. If this is not provided then predictions corresponding to the original data are returned. If newdata is provided then it should contain all the variables needed for prediction: a warning is generated if not.

type — When this has the value "link" (default) the linear predictor (possibly with associated standard errors) is returned. When type="terms" each component of the linear predictor is returned seperately (possibly with standard errors): this includes parametric model components, followed by each smooth component, but excludes any offset and any intercept. type="iterms" is the same, except that any standard errors returned for smooth components will include the uncertainty about the intercept/overall mean. When type="response" predictions on the scale of the response are returned (possibly with approximate standard errors). When type="lpmatrix" then a matrix is returned which yields the values of the linear predictor (minus any offset) when postmultiplied by the parameter vector (in this case se.fit is ignored). The latter option is most useful for getting variance estimates for quantities derived from the model: for example integrated quantities, or derivatives of smooths. A linear predictor matrix can also be used to implement approximate prediction outside R (see example code, below).

`se.fit` when this is TRUE (not default) standard error estimates are returned for each prediction.

`terms` if `type=="terms"` or `type="iterms"` then only results for the terms (smooth or parametric) named in this array will be returned. Otherwise any terms not named in this array will be set to zero. If `NULL` then all terms are included. `"(Intercept)"` is the intercept term.

`exclude` if `type=="terms"` or `type="iterms"` then terms (smooth or parametric) named in this array will not be returned. Otherwise any terms named in this array will be set to zero. If `NULL` then no terms are excluded. To avoid supplying covariate values for excluded smooth terms, set `newdata.guaranteed=TRUE`, but note that this skips all checks of `newdata`.

`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` 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 for `bam`.

`discrete` if `TRUE` then discrete prediction methods used with model fitted by discrete methods. `FALSE` for regular prediction. See details.

`n.threads` if `se.fit=TRUE` and discrete prediction is used then parallel computation can be used to speed up se calcualtion. This specifies number of htreads to use.

`gc.level` increase from 0 to up the level of garbage collection if default does not give enough.

`...` 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 `bam` as an argument is always ignored when predicting, unlike offsets specified in the bam model formula.

See the examples in `predict.gam` for how to use the `lpmatrix` for obtaining credible regions for quantities derived from the model.

When `discrete=TRUE` the prediction data in `newdata` is discretized in the same way as is done when using discrete fitting methods with `bam`. However the discretization grids are not currently identical to those used during fitting. Instead, discretization is done afresh for the prediction data. This means that if you are predicting for a relatively small set of prediction data, or on a regular grid, then the results may in fact be identical to those obtained without discretization. The disadvantage to this approach is that if you make predictions with a large data frame, and then split it into smaller data frames to make the predictions again, the results may differ slightly, because of slightly different discretization errors.

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.

WARNING

Predictions are likely to be incorrect if data dependent transformations of the covariates are used within calls to smooths. See examples in `predict.gam`.

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

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

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 predictions are required. If this is not provided then predictions corresponding to the original data are returned. If newdata is provided then it should contain all the variables needed for prediction: a warning is generated if not. See details for use with link{linear.functional.terms}.

`type` When this has the value "link" (default) the linear predictor (possibly with associated standard errors) is returned. When type="terms" each component of the linear predictor is returned seperately (possibly with standard errors): this includes parametric model components, followed by each smooth component, but excludes any offset and any intercept. type="iterms" is the same, except that any standard errors returned for smooth components will include the uncertainty about the intercept/overall mean. When type="response" predictions on the scale of the response are returned (possibly with approximate standard errors). When type="lpmatrix" then a matrix is returned which yields the values of the linear predictor (minus any offset) when postmultiplied by the parameter vector (in this case se.fit is ignored). The latter option is most useful for getting variance estimates for quantities derived from the model: for example

integrated quantities, or derivatives of smooths. A linear predictor matrix can also be used to implement approximate prediction outside R (see example code, below).

`se.fit` when this is TRUE (not default) standard error estimates are returned for each prediction. If set to a number between 0 and 1 then this is taken as the confidence level for intervals that are returned instead.

`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 terms not named in this array will be set to zero. If `NULL` then all terms are included.

`exclude` if `type=="terms"` or `type="iterms"` then terms (smooth or parametric) named in this array will not be returned. Otherwise any terms named in this array will be set to zero. If `NULL` then no terms are excluded. Note that this is the term names as it appears in the model summary, see example. You can avoid providing the covariates for excluded smooth terms by setting `newdata.guaranteed=TRUE`, which will avoid all checks on `newdata` (covariates for parametric terms can not be skipped).

`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.

`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`.

`unconditional` if `TRUE` then the smoothing parameter uncertainty corrected covariance matrix is used, when available, otherwise the covariance matrix conditional on the estimated smoothing parameters is used.

`iterms.type` if `type="iterms"` then standard errors can either include the uncertainty in the overall mean (default, withfixed and random effects included) or the uncertainty in the mean of the non-smooth fixed effects only (`iterms.type=2`).

`...` 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.

When predicting from models with `linear.functional.terms` then there are two possibilities. If the summation convention is to be used in prediction, as it was in fitting, then `newdata` should be a list, with named matrix arguments corresponding to any variables that were matrices in fitting. Alternatively one might choose to simply evaluate the constitutent smooths at particular values

in which case arguments that were matrices can be replaced by vectors (and newdata can be a dataframe). See linear.functional.terms for example code.

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=="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. If se.fit is a number between 0 and 1 then in place of se.fit two arrays are returned ll and ul giving the confidence interval limits.

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

WARNING

Predictions are likely to be incorrect if data dependent transformations of the covariates are used within calls to smooths. See examples.

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. doi:10.1111/j.1467-9469.2011.00760.x

Wood S.N. (2017, 2nd ed) Generalized Additive Models: An Introduction with R. Chapman and Hall/CRC Press. doi:10.1201/9781315370279

See Also

gam, gamm, plot.gam

Examples

```
library(mgcv)
n <- 200
sig <- 2
dat <- gamSim(1,n=n,scale=sig)

b <- gam(y~s(x0)+s(I(x1^2))+s(x2)+offset(x3),data=dat)

newd <- data.frame(x0=(0:30)/30,x1=(0:30)/30,x2=(0:30)/30,x3=(0:30)/30)
pred <- predict.gam(b,newd)
pred0 <- predict(b,newd,exclude="s(x0)") ## prediction excluding a term
## ...and the same, but without needing to provide x0 prediction data...
newd1 <- newd;newd1$x0 <- NULL ## remove x0 from `newd1'
pred1 <- predict(b,newd1,exclude="s(x0)",newdata.guaranteed=TRUE)

## custom perspective plot...

m1 <- 20;m2 <- 30; n <- m1*m2
x1 <- seq(.2,.8,length=m1);x2 <- seq(.2,.8,length=m2) ## marginal grid points
df <- data.frame(x0=rep(.5,n),x1=rep(x1,m2),x2=rep(x2,each=m1),x3=rep(0,n))
pf <- predict(b,newdata=df,type="terms")
persp(x1,x2,matrix(pf[,2]+pf[,3],m1,m2),theta=-130,col="blue",zlab="")

#############################################
## difference between "terms" and "iterms"
#############################################
nd2 <- 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")

## 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)


#############################################################
## 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
  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
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
}
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
  w1 <- (xn[j+1]-i*dx)/dx ## interpolation weights
  ## find approx. predict matrix row portion, by interpolation
  x0 <- 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)

#############################################################
## Example of producing a prediction interval for non Gaussian
## data...
#############################################################
```

```
f <- function(x) 0.2 * x^11 * (10 * (1 - x))^6 + 10 *
            (10 * x)^3 * (1 - x)^10
set.seed(6);n <- 100;x <- sort(runif(n))
Ey <- exp(f(x)/4);scale <- .5
y <- rgamma(n,shape=1/scale,scale=Ey*scale) ## sim gamma dataexit
b <- gam(y~s(x,k=20),family=Gamma(link=log),method="REML")
Xp <- predict(b,type="lpmatrix")
br <- rmvn(10000,coef(b),vcov(b)) ## 1000 replicate param. vectors
fr <- Xp
yr <- apply(fr,2,function(x) rgamma(length(x),shape=1/b$scale,
            scale=exp(x)*b$scale)) ## replicate data
pi <- apply(yr,1,quantile,probs=c(.1,.9),type=9) ## 80% PI
plot(x,y);lines(x,fitted(b));lines(x,pi[1,]);lines(x,pi[2,])
mean(y>pi[1,]&y<pi[2,]) ## check it


#################################################################
# illustration of unsafe scale dependent transforms in smooths....
#################################################################

b0 <- gam(y~s(x0)+s(x1)+s(x2)+x3,data=dat) ## safe
b1 <- gam(y~s(x0)+s(I(x1/2))+s(x2)+scale(x3),data=dat) ## safe
b2 <- gam(y~s(x0)+s(scale(x1))+s(x2)+scale(x3),data=dat) ## unsafe
pd <- dat; pd$x1 <- pd$x1/2; pd$x3 <- pd$x3/2
par(mfrow=c(1,2))
plot(predict(b0,pd),predict(b1,pd),main="b0 and b1 predictions match")
abline(0,1,col=2)
plot(predict(b0,pd),predict(b2,pd),main="b2 unsafe, doesn't match")
abline(0,1,col=2)


##################################################################
## Differentiating the smooths in a model (with CIs for derivatives)
##################################################################

## simulate data and fit model...
dat <- gamSim(1,n=300,scale=sig)
b<-gam(y~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)
X0 <- predict(b,newd,type="lpmatrix")

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")

Xp <- (X1-X0)/eps ## maps coefficients to (fd approx.) derivatives
colnames(Xp)      ## can check which cols relate to which smooth
```

```
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] <- Xp[,(i-1)*9+1:9+1] ## Xi%*%coef(b) = smooth deriv i
  df <- Xi%*%coef(b)              ## ith smooth derivative
  df.sd <- rowSums(Xi%*%b$Vp*Xi)^.5 ## cheap diag(Xi%*%b$Vp%*%t(Xi))^.5
  plot(x.mesh,df,type="l",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` *Prediction methods for smooth terms in a GAM*

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>

References

Wood S.N. (2017) Generalized Additive Models: An Introduction with R (2nd edition). 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 processed a smooth specification object generated by an s or te term in a gam formula.
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 coeffients 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. (2017) Generalized Additive Models: An Introduction with R (2nd edition). 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 'sf.wiggly'
Predict.matrix(object,data)
## S3 method for class 'sf.film'
Predict.matrix(object,data)
```

Arguments

`object`	A class `"soap.film"`, `"sf.wiggly"` or `"sf.film"` object.
`data`	A list list or data frame containing the arguments of the smooth at which predictions 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.

See Also

`smooth.construct.so.smooth.spec`

Examples

```
## 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())

## 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 <- runif(n)*5-1;y<-runif(n)*2-1
z <- fs.test(x,y,b=1)
ind <- inSide(fsb,x,y) ## remove outsiders
z <- z[ind];x <- x[ind]; y <- y[ind]
n <- length(z)
z <- z + rnorm(n)*.3 ## add noise

## plot boundary with knot and data locations
plot(fsb[[1]]$x,fsb[[1]]$y,type="l");points(knots$x,knots$y,pch=20,col=2)
points(x,y,pch=".",col=3);

## set up the basis and penalties...
sob <- smooth.construct2(s(x,y,bs="so",k=40,xt=list(bnd=fsb,nmax=100)),
              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 <- seq(-1,3.5,length=m);yn<-seq(-1,1,length=n)
xx <- rep(xm,n);yy<-rep(yn,rep(m,n))

## plot truth...
tru <- matrix(fs.test(xx,yy),m,n) ## truth
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))

## 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))
names(sob$sd$bnd[[1]]) <- c("xx","yy","d")
ind <- inSide(sob$sd$bnd,xx,yy)
fv.gam[!ind]<-NA
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` *Print a Generalized Additive Model object.*

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 deficieny). 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. (2017) Generalized Additive Models: An Introduction with R (2nd edition). CRC/ Chapman and Hall, Boca Raton, Florida.

See Also

gam, summary.gam

---

psum.chisq *Evaluate the c.d.f. of a weighted sum of chi-squared deviates*

Description

Evaluates the c.d.f. of a weighted sum of chi-squared random variables by the method of Davies (1973, 1980). That is it computes

$$P(q < \sum_{i=1}^{r} \lambda_i X_i + \sigma_z Z)$$

where X_j is a chi-squared random variable with df[j] (integer) degrees of freedom and non-centrality parameter nc[j], while Z is a standard normal deviate.

Usage

```
psum.chisq(q,lb,df=rep(1,length(lb)),nc=rep(0,length(lb)),sigz=0,
           lower.tail=FALSE,tol=2e-5,nlim=100000,trace=FALSE)
```

Arguments

q	is the vector of quantile values at which to evaluate.
lb	contains λ_i, the weight for deviate i. Weights can be positive and/or negative.
df	is the integer vector of chi-squared degrees of freedom.
nc	is the vector of non-centrality parameters for the chi-squared deviates.
sigz	is the multiplier for the standard normal deviate. Non- positive to exclude this term.
lower.tail	indicates whether lower of upper tail probabilities are required.
tol	is the numerical tolerance to work to.
nlim	is the maximum number of integration steps to allow
trace	can be set to TRUE to return some trace information and a fault code as attributes.

Details

This calls a C translation of the original Algol60 code from Davies (1980), which numerically inverts the characteristic function of the distribution (see Davies, 1973). Some modifications have been made to remove `goto` statements and global variables, to use a slightly more efficient sorting of `lb` and to use R functions for `log(1+x)`. In addition the integral and associated error are accumulated in single terms, rather than each being split into 2, since only their sums are ever used. If `q` is a vector then `psum.chisq` calls the algorithm separately for each `q[i]`.

If the Davies algorithm returns an error then an attempt will be made to use the approximation of Liu et al (2009) and a warning will be issued. If that is not possible then an `NA` is returned. A warning will also be issued if the algorithm detects that round off errors may be significant.

If `trace` is set to `TRUE` then the result will have two attributes. `"ifault"` is 0 for no problem, 1 if the desired accuracy can not be obtained, 2 if round-off error may be significant, 3 is invalid parameters have been supplied or 4 if integration parameters can not be located. `"trace"` is a 7 element vector: 1. absolute value sum; 2. total number of integration terms; 3. number of integrations; 4. integration interval in main integration; 5. truncation point in initial integration; 6. sd of convergence factor term; 7. number of cycles to locate integration parameters. See Davies (1980) for more details. Note that for vector `q` these attributes relate to the final element of `q`.

Author(s)

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

References

Davies, R. B. (1973). Numerical inversion of a characteristic function. Biometrika, 60(2), 415-417.

Davies, R. B. (1980) Algorithm AS 155: The Distribution of a Linear Combination of Chi-squared Random Variables. J. R. Statist. Soc. C 29, 323-333

Liu, H.; Tang, Y. & Zhang, H. H (2009) A new chi-square approximation to the distribution of non-negative definite quadratic forms in non-central normal variables. Computational Statistics & Data Analysis 53,853-856

Examples

```
require(mgcv)
lb <- c(4.1,1.2,1e-3,-1) ## weights
df <- c(2,1,1,1) ## degrees of freedom
nc <- c(1,1.5,4,1) ## non-centrality parameter
q <- c(1,6,20) ## quantiles to evaluate

psum.chisq(q,lb,df,nc)

## same by simulation...

psc.sim <- function(q,lb,df=lb*0+1,nc=df*0,ns=10000) {
  r <- length(lb);p <- q
  X <- rowSums(rep(lb,each=ns) *
       matrix(rchisq(r*ns,rep(df,each=ns),rep(nc,each=ns)),ns,r))
  apply(matrix(q),1,function(q) mean(X>q))
} ## psc.sim
```

```
psum.chisq(q,lb,df,nc)
psc.sim(q,lb,df,nc,100000)
```

---

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

```
qq.gam(object, rep=0, level=.9,s.rep=10,
       type=c("deviance","pearson","response"),
       pch=".", rl.col=2, rep.col="gray80", ...)
```

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.

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 coefficents 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.

See Also

`choose.k`, `gam`

Examples

```
library(mgcv)
## simulate binomial data...
set.seed(0)
n.samp <- 400
dat <- gamSim(1,n=n.samp,dist="binary",scale=.33)
p <- binomial()$linkinv(dat$f) ## binomial p
n <- sample(c(1,3),n.samp,replace=TRUE) ## binomial n
dat$y <- rbinom(n,n,p)
dat$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
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~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 <- 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)
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 <- gam(y~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.

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(...,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(~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 random 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 such 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. Some models require differences between different levels of the same random effect: these can be implemented as described in linear.functional.terms.

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, using a full Laplace approximation to the (restricted) marginal likelihood. However note that MAP estimates of the model coefficients are returned. For the fixed effects in non-Gaussian cases, these are only approximately the MLEs (with the random effects integrated out), and the difference can be large in some circumstances.

gam can be slow for fitting models with large numbers of random effects, because it does not exploit the sparsity 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 reliable 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.

Note also that

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

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)
plot(b)

## simulate example...
dat <- gamSim(1,n=400,scale=2) ## simulate 4 term additive truth

fac <- sample(1:20,400,replace=TRUE)
b <- rnorm(20)*.5
dat$y <- dat$y + b[fac]
dat$fac <- as.factor(fac)

rm1 <- gam(y ~ s(fac,bs="re")+s(x0)+s(x1)+s(x2)+s(x3),data=dat,method="ML")
gam.vcomp(rm1)

fv0 <- predict(rm1,exclude="s(fac)") ## predictions setting r.e. to 0
fv1 <- predict(rm1) ## predictions setting r.e. to predicted values
## prediction setting r.e. to 0 and not having to provide 'fac'...
pd <- dat; pd$fac <- NULL
fv0 <- predict(rm1,pd,exclude="s(fac)",newdata.guaranteed=TRUE)

## Prediction with levels of fac not in fit data.
## The effect of the new factor levels (or any interaction involving them)
## is set to zero.
xx <- seq(0,1,length=10)
pd <- data.frame(x0=xx,x1=xx,x2=xx,x3=xx,fac=c(1:10,21:30))
fv <- predict(rm1,pd)
pd$fac <- NULL
fv0 <- predict(rm1,pd,exclude="s(fac)",newdata.guaranteed=TRUE)
```

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 μ 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).

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>

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.

Examples

```
require(mgcv)
set.seed(7)
## An inverse.gaussian GAM example, by modify `gamSim' output...
dat <- gamSim(1,n=400,dist="normal",scale=1)
dat$f <- dat$f/4 ## true linear predictor
Ey <- exp(dat$f);scale <- .5 ## mean and GLM scale parameter
## simulate inverse Gaussian response...
```

```
dat$y <- rig(Ey,mean=Ey,scale=.2)
big <- gam(y~ s(x0)+ s(x1)+s(x2)+s(x3),family=inverse.gaussian(link=log),
          data=dat,method="REML")
plot(big,pages=1)
gam.check(big)
summary(big)
```

`rmvn` *Generate from or evaluate multivariate normal or t densities.*

Description

Generates multivariate normal or t random deviates, and evaluates the corresponding log densities.

Usage

```
rmvn(n,mu,V)
r.mvt(n,mu,V,df)
dmvn(x,mu,V,R=NULL)
d.mvt(x,mu,V,df,R=NULL)
```

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.
`df`	The degrees of freedom for a t distribution.
`x`	A vector or matrix to evaluate the log density of.
`R`	An optional Cholesky factor of V (not pivoted).

Details

Uses a 'square root' of `V` to transform standard normal deviates to multivariate normal with the correct covariance matrix.

Value

An `n` row matrix, with each row being a draw from a multivariate normal or t density with covariance matrix `V` and mean vector `mu`. Alternatively each row may have a different mean vector if `mu` is a vector.

For density functions, a vector of log densities.

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
dmvn(z,mu,V)
```

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>

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 <- runif(n*(p-1))
X <- matrix(c(x,x[1:n]),n,p)
qrx <- qr(X,LAPACK=TRUE)
Rrank(qr.R(qrx))
```

---

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<p<2 is a sum of N gamma random variables where N has a Poisson distribution, with mean 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.

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. https://cran.r-project.org/package=tweedie

See Also

ldTweedie, Tweedie

Examples

```
library(mgcv)
f2 <- function(x) 0.2 * x^11 * (10 * (1 - x))^6 + 10 *
          (10 * x)^3 * (1 - x)^10
n <- 300
x <- runif(n)
mu <- exp(f2(x)/3+.1);x <- x*10 - 4
y <- rTweedie(mu,p=1.5,phi=1.3)
b <- gam(y~s(x,k=20),family=Tweedie(p=1.5))
b
plot(b)
```

---

s *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(..., k=-1,fx=FALSE,bs="tp",m=NA,by=NA,xt=NULL,id=NULL,sp=NULL,pc=NULL)
```

Arguments

...	a list of variables that are the covariates that this smooth is a function of. Transformations whose form depends on the values of the data are best avoided here: e.g. s(log(x)) is fine, but s(I(x/sd(x))) is not (see predict.gam).
k	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.
fx	indicates whether the term is a fixed d.f. regression spline (TRUE) or a penalized regression spline (FALSE).
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.

m — 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.

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). 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).

xt — Any extra information required to set up a particular basis. Used e.g. to set large data set handling behaviour for "tp" basis. If xt$sumConv exists and is FALSE then the summation convention for matrix arguments is turned off.

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 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). An id with a factor by variable causes the smooths at each factor level to have the same smoothing parameter.

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 gamm.

pc — If not NULL, signals a point constraint: the smooth should pass through zero at the point given here (as a vector or list with names corresponding to the smooth names). Never ignored if supplied as a number. See identifiability. Can also be supplied as a list of lists to specify general linear constraints as described in scasm - ignored by other modelling routines.

Details

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

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:

term	An array of text strings giving the names of the covariates that the term is a function of.
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 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. (2017) Generalized Additive Models: An Introduction with R (2nd edition). Chapman and Hall/CRC Press.

See Also

te, gam, gamm

Examples

```
# example utilising `by' variables
library(mgcv)
set.seed(0)
n<-200;sig2<-4
x1 <- runif(n, 0, 1);x2 <- runif(n, 0, 1);x3 <- 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)
f1 <-  exp(2 * x1) - 3.75887
f2 <-  0.2 * x1^11 * (10 * (1 - x1))^6 + 10 * (10 * x1)^3 * (1 - x1)^10
f<-f1*fac.1+f2*fac.2+x2
e <- rnorm(n, 0, sqrt(abs(sig2)))
y <- 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)
```

scasm *Shape constrained additive smooth models*

Description

Fits a generalized additive model (GAM, GAMM, GAMLSS) subject to shape constriants on the components, using quadratic and linear programming and the extended Fellner-Schall method for smoothness estimation. Usable with all the families listed in family.mgcv.

Usage

```
scasm(formula,family=gaussian(),data=list(),weights=NULL,subset=NULL,
      na.action, offset=NULL,control=list(),scale=0,select=FALSE,
      knots=NULL,sp=NULL,gamma=1,fit=TRUE,G=NULL,drop.unused.levels=TRUE,
      drop.intercept=NULL,bs=0,...)
```

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 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 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).

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

`select` Should selection penalties be added to the smooth effects, so that they can in principle be penalized out of the model? See `gamma` to increase penalization. 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` Increase above 1 to force smoother fits. `gamma` is used to multiply the effective degrees of freedom in the GCV/UBRE/AIC score (so `log(n)/2` is BIC like). `n/gamma` can be viewed as an effective sample size, which allows it to play a similar role for RE/ML smoothing parameter estimation.

`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. Note that `discrete=TRUE`may result in re-ordering of variables in tensor product smooths for improved efficiency, and `sp` must be supplied in re-ordered order.

`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 `sp`, `chunk.size`, `gamma`,`nthreads`, `cluster`, `rho`, `gc.level`, `samfrac`, `use.chol`, `method` and `scale` (if >0).

`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`.

`drop.intercept` Set to `TRUE` to force the model to really not have the a constant in the parametric model part, even with factor variables present.

`bs` number of bootstrap replicates to use to approximate sampling distribution of model coefficients. If not zero or `FALSE` then results in a matrix `bs` being returned in the fit object, whose columns are the bootstrap replicate coefficients.

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

Details

Operates much as gam, but allows linear constraints on smooths to facilitate shape constraints. Smoothness selection uses the extended Fellner-Schall method of Wood and Fasiolo (2017). Model coefficient estimation uses linear programming to find initial feasible coefficients, and then sequential quadratic programming to maximize the penalized likelihood given smooting parameters.

For simple shape constraints on univariate smooths (and hence tensor product smooths) see smooth.construct.sc.smooth.s General linear constraints can be imposed on any smooth term via the pc argument to s etc. To do this pc is provided as a list of two lists. The first list component specifies the linear inequality constraints, and the second, if present specifies linear equality constraints.

The first list has a matrix element for each covariate of the smooth, named as the covariates. Two further elements are un-named: a matrix of weights and a vector specifying the right hand side of the condition to be met. For example, suppose that a constraint is to be applied to a smooth term term $f(X, Z)$. The list would contain named matrices $\mathbf{X}$ and $\mathbf{Z}$ and a third unnamed matrix, $\mathbf{W}$, say. All are of the same dimension. The list also contains an unnaed vector with the same number of rows as the matrices, $\mathbf{b}$, say. The constraints encoded are

$$\sum_j f(X_{ij}, Z_{ij}) W_{ij} > b_i$$

where the summation is over all columns of the matrices. There is one constraint for each row of the $\mathbf{b}$. The second list, if present, has the same structure, but encodes equality constraints. If the first list is empty then there are no inequality constraints.

Notice that this structure is quite general - derivative and integral constraints are easily implemented numerically. Also note that the simple constraints of smooth.construct.sc.smooth.spec can be combined with general pc constraints.

Value

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

WARNINGS

This is generally slower than gam, and can only use EFS soothness estimation

Author(s)

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

References

Wood, S.N. and M. Fasiolo (2017) A generalized Fellner-Schall method for smoothing parameter optimization with application to Tweedie location, scale and shape models. Biometrics 73 (4), 1071-1081 doi:10.1111/biom.12666

See Also

smooth.construct.sc.smooth.spec, mgcv-package, gamObject, gam.models, smooth.terms, predict.gam, plot.gam, summary.gam, gam.check

Examples

```
library(mgcv)
## a density estimation example, illustrating general constraint use...
n <- 400; set.seed(4)
x <- c(rnorm(n*.8,.4,.13),runif(n*.2))
y <- rep(1e-10,n)
xp=(1:n-.5)/n
options(warn=-1)
## Use an exponential distribution, inverse link and zero pseudoresponse
## data to get correct likelihood. Use a positive smooth and impose
## integration to 1 via a general linear 'pc' constraint...
b <- scasm(y ~ s(x,bs="sc",xt="+",k=15,pc=list(list(x=matrix(xp,1,n),
      matrix(1/n,1,n),1)))-1, family=Gamma,scale=1,knots=list(x=c(0,1)))
options(warn=0)
plot(b,scheme=2)
lines(xp,.2+dnorm(xp,.4,.13)*.8,col=2)

## some more standard examples

fs <- function(x,k) { ## some simulation functions
  if (k==2) 0.2*x^11*(10*(1-x))^6 + 10*(10*x)^3* (1-x)^10 else
  if (k==0) 2 * sin(pi * x) else
  if (k==1) exp(2 * x) else
  if (k==3) exp((x-.4)*20)/(1+exp((x-.4)*20)) else
  if (k==4) (x-.55)^4 else x
} ## fs

## Simple Poisson example...
n <- 400; set.seed(4)
x0 <- runif(n);x1 <- runif(n);x2 <- runif(n);x3 <- runif(n);
fac <- factor(sample(1:40,n,replace=TRUE))
f <- fs(x0,k=0) + 2*fs(x1,k=3) + fs(x2,k=2) + 10*fs(x3,k=4)
mu <- exp((f-1)/4)
y <- rpois(n,mu)
k <- 10
b0 <- gam(y~s(x0,bs="bs",k=k)+s(x1,bs="bs",k=k)+s(x2,bs="bs",k=k)+
          s(x3,bs="bs",k=k),method="REML",family=poisson)

b1 <- scasm(y~s(x0,bs="bs",k=k)+s(x1,bs="bs",k=k)+s(x2,bs="bs",k=k)+
            s(x3,bs="bs",k=k),family=poisson)
b0;b1 ## note similarity when no constraints

plot(b1,pages=1,scheme=2) ## second term not monotonic, so impose this...

b2 <- scasm(y~s(x0,bs="bs",k=k)+s(x1,bs="sc",xt="m+",k=k)+
            s(x2,bs="bs",k=k)+s(x3,bs="bs",k=k),family=poisson)
plot(b2,pages=1,scheme=2)

## constraint respecting bootstrap CIs...

b3 <- scasm(y~s(x0,bs="bs",k=k)+s(x1,bs="sc",xt="m+",k=k)+
            s(x2,bs="bs",k=k)+s(x3,bs="bs",k=k),family=poisson,bs=200)
```

```
plot(b3,pages=1,scheme=2)
fv <- predict(b3,se=TRUE)
fv1 <- predict(b3,se=.95)
plot(fv$se*3.92,fv1$ul-fv1$ll) ## compare CI widths
abline(0,1,col=2)

## gamma distribution location-scale example

## simulate data...
x0 <- runif(n);x1 <- runif(n);x2 <- runif(n);x3 <- runif(n);
fac <- factor(sample(1:40,n,replace=TRUE))
f <- fs(x0,k=0) + 2*fs(x1,k=3) + fs(x2,k=2) #+ 10*fs(x3,k=4)
mu <- exp((fs(x0,k=0)+ fs(x2,k=2))/5)
th <- exp((fs(x1,k=1)+ 10*fs(x3,k=4))/2-2)
y <- rgamma(n,shape=1/th,scale=mu*th)

## fit model
b <- scasm(list(y~s(x0,bs="sc",xt="c-")+s(x2),~s(x1,bs="sc",xt="m+")
       +s(x3)),family=gammals,bs=200)
plot(b,scheme=2,pages=1,scale=0) ## bootstrap CIs
b$bs <- NULL; plot(b,scheme=2,pages=1,scale=0) ## constraint free CI

## A survival example...

library(survival) ## for data
col1 <- colon[colon$etype==1,] ## focus on single event
col1$differ <- as.factor(col1$differ)
col1$sex <- as.factor(col1$sex)

## set up the AFT response...
logt <- cbind(log(col1$time),log(col1$time))
logt[col1$status==0,2] <- Inf ## right censoring
col1$logt <- -logt ## -ve conventional for AFT versus Cox PH comparison

## fit the model...
b0 <- gam(logt~s(age,by=sex)+sex+s(nodes)+perfor+rx+obstruct+adhere,
         family=cnorm(),data=col1)
plot(b0,pages=1)

## nodes effect should be increasing...
b <- scasm(logt~s(age,by=sex)+sex+s(nodes,bs="sc",xt="m+")+perfor+rx+
           obstruct+adhere,family=cnorm(),data=col1)
plot(b,pages=1)
b

## same with logistic distribution...
b <- scasm(logt~s(age,by=sex)+sex+s(nodes,bs="sc",xt="m+")+perfor+rx+
           obstruct+adhere,family=clog(),data=col1)
plot(b,pages=1)

## and with Cox PH model...
b <- scasm(time~s(age,by=sex)+sex+s(nodes,bs="sc",xt="m+")+perfor+rx+
           obstruct+adhere,family=cox.ph(),data=col1,weights=status)
```

```
summary(b)
plot(b,pages=1) ## plot effects
```

---

scat *GAM scaled t family for heavy tailed data*

Description

Family for use with gam or bam, 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 σ and ν are parameters to be estimated alongside the smoothing parameters.

Usage

```
scat(theta = NULL, link = "identity",min.df=3)
```

Arguments

theta	the parameters to be estimated $\nu = b + \exp(\theta_1)$ (where 'b' is min.df) and $\sigma = \exp(\theta_2)$. If supplied and both positive, then taken to be fixed values of ν and σ. If any negative, then absolute values taken as starting values.
link	The link function: one of "identity", "log" or "inverse".
min.df	minimum degrees of freedom. Should not be set to 2 or less as this implies infinite response variance.

Details

Useful in place of Gaussian, when data are heavy tailed. min.df can be modified, but lower values can occasionally lead to convergence problems in smoothing parameter estimation. In any case min.df should be >2, since only then does a t random variable have finite variance.

Value

An object of class extended.family.

Author(s)

Natalya Pya (nat.pya@gmail.com)

References

Wood, S.N., N. Pya and B. Saefken (2016), Smoothing parameter and model selection for general smooth models. Journal of the American Statistical Association 111, 1548-1575 doi:10.1080/01621459.2016.1180986

Examples

```
library(mgcv)
## Simulate some t data...
set.seed(3);n<-400
dat <- gamSim(1,n=n)
dat$y <- dat$f + rt(n,df=4)*2

b <- gam(y~s(x0)+s(x1)+s(x2)+s(x3),family=scat(link="identity"),data=dat)

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

---

sdiag *Extract or modify diagonals of a matrix*

Description

Extracts or modifies sub- or super- diagonals of a matrix.

Usage

```
sdiag(A,k=0)
sdiag(A,k=0) <- value
```

Arguments

A	a matrix
k	sub- (negative) or super- (positive) diagonal of a matrix. 0 is the leading diagonal.
value	single value, or vector of the same length as the diagonal.

Value

A vector containing the requested diagonal, or a matrix with the requested diagonal replaced by value.

Author(s)

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

Examples

```
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
```

---

shash *Sinh-arcsinh location scale and shape model family*

Description

The shash family implements the four-parameter sinh-arcsinh (shash) distribution of Jones and Pewsey (2009). The location, scale, skewness and kurtosis of the density can depend on additive smooth predictors. Useable only with gam, the linear predictors are specified via a list of formulae. It is worth carefully considering whether the data are sufficient to support estimation of such a flexible model before using it.

Usage

```
shash(link = list("identity", "logeb", "identity", "identity"),
      b = 1e-2, phiPen = 1e-3)
```

Arguments

link	vector of four characters indicating the link function for location, scale, skewness and kurtosis parameters.
b	positive parameter of the logeb link function, see Details.
phiPen	positive multiplier of a ridge penalty on kurtosis parameter. Do not touch it unless you know what you are doing, see Details.

Details

The density function of the shash family is

$$p(y|\mu,\sigma,\epsilon,\delta) = C(z)\exp\{-S(z)^2/2\}\{2\pi(1+z^2)\}^{-1/2}/\sigma,$$

where $C(z) = \{1+S(z)^2\}^{1/2}$, $S(z) = \sinh\{\delta\sinh^{-1}(z)-\epsilon\}$ and $z = (y-\mu)/(\sigma\delta)$. Here μ and $\sigma > 0$ control, respectively, location and scale, ϵ determines skewness, while $\delta > 0$ controls tailweight. shash can model skewness to either side, depending on the sign of ϵ. Also, shash can have tails that are lighter ($\delta > 1$) or heavier ($0 < \delta < 1$) that a normal. For fitting purposes, here we are using $\tau = \log(\sigma)$ and $\phi = \log(\delta)$.

The density is based on the expression given on the second line of section 4.1 and equation (2) of Jones and Pewsey (2009), and uses the simple reparameterization given in section 4.3.

The link function used for τ is logeb with is $\eta = \log\{\exp(\tau) - b\}$ so that the inverse link is $\tau = \log(\sigma) = \log\{\exp(\eta) + b\}$. The point is that we are don't allow σ to become smaller than a small constant b. The likelihood includes a ridge penalty $-phiPen * \phi^2$, which shrinks ϕ toward zero. When sufficient data is available the ridge penalty does not change the fit much, but it is useful to include it when fitting the model to small data sets, to avoid ϕ diverging to +infinity (a problem already identified by Jones and Pewsey (2009)).

Value

An object inheriting from class `general.family`.

Author(s)

Matteo Fasiolo <matteo.fasiolo@gmail.com> and Simon N. Wood.

References

Jones, M. and A. Pewsey (2009). Sinh-arcsinh distributions. Biometrika 96 (4), 761-780. doi:10.1093/biomet/asp053

Wood, S.N., N. Pya and B. Saefken (2016), Smoothing parameter and model selection for general smooth models. Journal of the American Statistical Association 111, 1548-1575 doi:10.1080/01621459.2016.1180986

Examples

```
###############
# Shash dataset
###############
##  Simulate some data from shash
set.seed(847)
n <- 1000
x <- seq(-4, 4, length.out = n)

X <- cbind(1, x, x^2)
beta <- c(4, 1, 1)
mu <- X %*% beta

sigma =  .5+0.4*(x+4)*.5            # Scale
eps = 2*sin(x)                      # Skewness
del = 1 + 0.2*cos(3*x)              # Kurtosis

dat <-  mu + (del*sigma)*sinh((1/del)*asinh(qnorm(runif(n))) + (eps/del))
dataf <- data.frame(cbind(dat, x))
names(dataf) <- c("y", "x")
plot(x, dat, xlab = "x", ylab = "y")

## Fit model
fit <- gam(list(y ~ s(x), # <- model for location
                  ~ s(x),   # <- model for log-scale
```

```
                  ~ s(x),   # <- model for skewness
                  ~ s(x, k = 20)), # <- model for log-kurtosis
            data = dataf,
            family = shash, # <- new family
            optimizer = "efs")

## Plotting truth and estimates for each parameters of the density
muE <- fit$fitted[ , 1]
sigE <- exp(fit$fitted[ , 2])
epsE <- fit$fitted[ , 3]
delE <- exp(fit$fitted[ , 4])

par(mfrow = c(2, 2))
plot(x, muE, type = 'l', ylab = expression(mu(x)), lwd = 2)
lines(x, mu, col = 2, lty = 2, lwd = 2)
legend("top", c("estimated", "truth"), col = 1:2, lty = 1:2, lwd = 2)

plot(x, sigE, type = 'l', ylab = expression(sigma(x)), lwd = 2)
lines(x, sigma, col = 2, lty = 2, lwd = 2)

plot(x, epsE, type = 'l', ylab = expression(epsilon(x)), lwd = 2)
lines(x, eps, col = 2, lty = 2, lwd = 2)

plot(x, delE, type = 'l', ylab = expression(delta(x)), lwd = 2)
lines(x, del, col = 2, lty = 2, lwd = 2)

## Plotting true and estimated conditional density
par(mfrow = c(1, 1))
plot(x, dat, pch = '.', col = "grey", ylab = "y", ylim = c(-35, 70))
for(qq in c(0.001, 0.01, 0.1, 0.5, 0.9, 0.99, 0.999)){
  est <- fit$family$qf(p=qq, mu = fit$fitted)
  true <- mu + (del * sigma) * sinh((1/del) * asinh(qnorm(qq)) + (eps/del))
  lines(x, est, type = 'l', col = 1, lwd = 2)
  lines(x, true, type = 'l', col = 2, lwd = 2, lty = 2)
}
legend("topleft", c("estimated", "truth"), col = 1:2, lty = 1:2, lwd = 2)

#####################
## Motorcycle example
#####################

# Here shash is overkill, in fact the fit is not good, relative
# to what we would get with mgcv::gaulss
library(MASS)

b <- gam(list(accel~s(times, k=20, bs = "ad"), ~s(times, k = 10), ~1, ~1),
         data=mcycle, family=shash)

par(mfrow = c(1, 1))
xSeq <- data.frame(cbind("accel" = rep(0, 1e3),
                   "times" = seq(2, 58, length.out = 1e3)))
pred <- predict(b, newdata = xSeq)
plot(mcycle$times, mcycle$accel, ylim = c(-180, 100))
```

```
for(qq in c(0.1, 0.3, 0.5, 0.7, 0.9)){
  est <- b$family$qf(p=qq, mu = pred)
  lines(xSeq$times, est, type = 'l', col = 2)
}

plot(b, pages = 1, scale = FALSE)
```

`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 α 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 α 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 α 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>

Examples

```
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 if 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
  a <- x%*%alpha     ## argument of smooth
  b <- gam(y~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
    for (j in 1:length(theta)) J[j+1,j] <- 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 <- function(x) 0.2 * x^11 * (10 * (1 - x))^6 + 10 *
            (10 * x)^3 * (1 - x)^10
n <- 200;m <- 3
x <- matrix(runif(n*m),n,m) ## the covariates for the single index part
z <- runif(n) ## another covariate
alpha <- c(1,-1,.5); alpha <- alpha/sqrt(sum(alpha^2))
eta <- as.numeric(f2((x%*%alpha+.41)/1.4)+1+z^2*2)/4
mu <- exp(eta)
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 <- 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$estimate,y=y,x=x,z=z,hessian=TRUE,k=10)
theta.est <-f1$estimate

## Alternative using 'optim'...

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)
## now get theta/alpha with smoothing parameter selection...
f1 <- optim(f0$par,si,y=y,x=x,z=z,hessian=TRUE,k=10)
theta.est <-f1$par

## extract and examine fitted model...

b <- 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))
diag(Vt)^.5
```

`Sl.inirep` *Re-parametrizing model matrix X*

Description

INTERNAL routine to apply initial Sl re-parameterization to model matrix X, or, if `inverse==TRUE`, to apply inverse re-parametrization to parameter vector or covariance matrix.

Usage

```
Sl.inirep(Sl,X,l,r,nt=1)

Sl.initial.repara(Sl, X, inverse = FALSE, both.sides = TRUE, cov = TRUE,
  nt = 1)
```

Arguments

`Sl`	the output of `Sl.setup`.
`X`	the model matrix.
`l`	if non-zero apply transform (positive) or inverse transform from left. 1 or -1 of transform, 2 or -2 for transpose.
`r`	if non-zero apply transform (positive) or inverse transform from right. 1 or -1 of transform, 2 or -2 for transpose.
`inverse`	if `TRUE` an inverse re-parametrization is performed.
`both.sides`	if `inverse==TRUE` and `both.sides==FALSE` then the re-parametrization only applied to rhs, as appropriate for a choleski factor. If `both.sides==FALSE`, `X` is a vector and `inverse==FALSE` then `X` is taken as a coefficient vector (so re-parametrization is inverse of that for the model matrix).
`cov`	boolean indicating whether `X` is a covariance matrix.
`nt`	number of parallel threads to be used.

Value

A re-parametrized version of `X`.

Author(s)

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

`Sl.repara` *Applying re-parameterization from log-determinant of penalty matrix to model matrix.*

Description

INTERNAL routine to apply re-parameterization from log-determinant of penalty matrix, `ldetS` to model matrix, `X`, blockwise.

Usage

```
Sl.repara(rp, X, inverse = FALSE, both.sides = TRUE)
```

Arguments

rp	reparametrization.
X	if X is a matrix it is assumed to be a model matrix whereas if X is a vector it is assumed to be a parameter vector.
inverse	if TRUE an inverse re-parametrization is performed.
both.sides	if inverse==TRUE and both.sides==FALSE then the re-parametrization only applied to rhs, as appropriate for a choleski factor. If both.sides==FALSE, X is a vector and inverse==FALSE then X is taken as a coefficient vector (so re-parametrization is inverse of that for the model matrix).

Value

A re-parametrized version of X.

Author(s)

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

Sl.setup — *Setting up a list representing a block diagonal penalty matrix*

Description

INTERNAL function for setting up a list representing a block diagonal penalty matrix from the object produced by gam.setup.

Usage

```
Sl.setup(G,cholesky=FALSE,no.repara=FALSE,sparse=FALSE,keepS=FALSE)
```

Arguments

G	the output of gam.setup.
cholesky	re-parameterize using Cholesky only.
no.repara	set to TRUE to turn off all initial reparameterization.
sparse	sparse setup?
keepS	store original penalties in S0? (after any splitting of multiple penalties into singletons, but before reparameterization)

Value

A list with an element for each block. For block, b, `Sl[[b]]` is a list with the following elements

- `repara`: should re-parameterization be applied to model matrix, etc? Usually `FALSE` if non-linear in coefficients.
- `start, stop`: such that `start:stop` are the indexes of the parameters of this block.
- `S`: a list of penalty matrices for the block (`dim = stop-start+1`) If `length(S)==1` then this will be an identity penalty. Otherwise it is a multiple penalty, and an `rS` list of square root penalty matrices will be added. `S` (if `repara==TRUE`) and `rS` (always) will be projected into range space of total penalty matrix.
- `rS`: square root of penalty matrices if multiple penalties are used.
- `D`: a reparameterization matrix for the block. Applies to cols/params in `start:stop`. If numeric then `X[,start:stop]%*%diag(D)` is re-parametrization of `X[,start:stop]`, and `b.orig = D*b.repara` (where `b.orig` is the original parameter vector). If matrix then `X[,start:stop]%*%D` is re-parametrization of `X[,start:stop]`, and `b.orig = D%*%b.repara` (where `b.orig` is the original parameter vector).
- `S0`: original penalties if requested.

Author(s)

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

---

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)
```

Arguments

`A`	A symmetric matrix.
`k`	Must be non-negative. If `kl` is negative, then the `k` largest magnitude eigenvalues are found, together with the corresponding eigenvectors. If `kl` is non-negative then the `k` highest eigenvalues are found together with their eigenvectors and the `kl` lowest eigenvalues with eigenvectors are also returned.
`kl`	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 `kl` is non-negative, returns the highest `k` and lowest `kl` eigenvalues, with their corresponding eigenvectors. If `kl` 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 `kl` the approach is faster than computing the full symmetric eigendecompostion. 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

`tprs`

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]))
```

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.
- **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.

data — 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.

knots — 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.

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

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).
- `Cp` — An optional matrix supplying alternative identifiability constraints for use when predicting. By default the fitting constrants are used. This option is useful when some sort of simple sparse constraint is required for fitting, but the usual sum-to-zero constraint is required for prediction so that, e.g. the CIs for model components are as narrow as possible.
- `no.rescale` — if this is non-NULL then the penalty coefficient matrix of the smooth will not be rescaled for enhaced numerical stability (rescaling is the default, because `gamm` requires it). Turning off rescaling is useful if the values of the smoothing parameters should be interpretable in a model, for example because they are inverse variance components.
- `df` — the degrees of freedom associated with this term (when unpenalized and unconstrained). If this is null then `smoothCon` will set it to the basis dimension. `smoothCon` will reduce this by the number of constraints.
- `te.ok` — `0` if this term should not be used as a tensor product marginal, `1` if it can be used and plotted, and `2` is it can be used but not plotted. Set to `1` if `NULL`.
- `plot.me` — Set to `FALSE` if this smooth should not be plotted by `plot.gam`. Set to `TRUE` if `NULL`.

side.constrain Set to FALSE to ensure that the smooth is never subject to side constraints as a result of nesting.

L smooths may depend on fewer 'underlying' smoothing parameters than there are elements of S. In this case L is the matrix mapping the vector of underlying log smoothing parameters to the vector of logs of the smoothing parameters actually multiplying the S[[i]]. L=NULL signifies that there is one smoothing parameter per S[[i]].

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.

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

See Also

s,get.var, gamm, gam, Predict.matrix, smoothCon, PredictMat

Examples

```
## 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) {
## 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")
  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
  x.shift <- mean(x) # shift used to enhance stability
  k <- knots[[object$term]] ## will be NULL if none supplied
  if (is.null(k)) # space knots through data
  { n<-length(x)
    k<-quantile(x[2:(n-1)],seq(0,1,length=nk+2))[2:(nk+1)]
  }
  if (length(k)!=nk) # right number of knots?
  stop(paste("there should be ",nk," supplied knots"))
  x <- x - x.shift # basis stabilizing shift
  k <- k - x.shift # knots treated the same!
  X<-matrix(0,length(x),object$bs.dim)
  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)

  class(object)<-"tr.smooth"  # Give object a class
  object
}

Predict.matrix.tr.smooth<-function(object,data) {
## prediction method function for the `tr' smooth class
  x <- data[[object$term]]
  x <- x - object$x.shift # stabilizing shift
  m <- object$m;     # spline order (3=cubic)
  k<-object$knots    # knot locations
  nk<-length(k)      # number of knots
```

```
  X<-matrix(0,length(x),object$bs.dim)
  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])
  X # return the prediction matrix
}

# an example, using the new class....
require(mgcv)
set.seed(100)
dat <- gamSim(1,n=400,scale=2)
b<-gam(y~s(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~s(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$gam,pages=1)
# another example using tensor products of the new class
dat <- gamSim(2,n=400,scale=.1)$data
b <- 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.

Shape constrained versions are available for use with `scasm`, with constraints speficied as in `smooth.construct.sc.smooth.`

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

Arguments

`object`	a smooth specification object, usually generated by a term `s(...,bs="ad",...)`
`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`>

Examples

```
## 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).

par(mfrow=c(2,2),mar=c(1,1,1,1))
x <- seq(-.5, 1.5, length= 60)
z <- x
f3 <- function(x,z,k=15) { r<-sqrt(x^2+z^2);f<-exp(-r^2*k);f}
f <- 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 <- runif(n)*2-.5
z <- runif(n)*2-.5
f <- f3(x,z)
y <- 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)
```

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 conventional cubic spline. Any further elements of m, after the first 2, define the order of derivative in further penalties. 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. See details for options for controlling the interval over which the penalty is evaluated (which can matter if it is necessary to extrapolate).

Shape constrained "sc" versions are available for use with scasm.

Usage

```
## S3 method for class 'bs.smooth.spec'
smooth.construct(object, data, knots)
## S3 method for class 'sc.smooth.spec'
smooth.construct(object, data, knots)
## S3 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 terms can have multiple penalties of different orders, for example s(x,bs="bs",m=c(3,2,1,0)) specifies a cubic basis with 3 penalties: a conventional cubic spline penalty, an integrated square of first derivative penalty, and an integrated square of function value penalty.

The default basis dimension, k, is the larger of 10 and m[1]. 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 (making this range too wide mean that there is 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.

Another option is to supply 4 knots. Then the outer 2 define the interval over which the penalty is to be evaluated, while the inner 2 define an interval within which all but the outermost 2 knots should lie. Normally the outer 2 knots would be the interval over which predictions might be required, while the inner 2 knots define the interval within which the data lie. This option allows the penalty to apply over a wider interval than the data, while still placing most of the basis functions where the data are. This is useful in situations in which it is necessary to extrapolate slightly with a smooth. Only applying the penalty over the interval containing the data amounts to a model in which the function could be less smooth outside the interval than within it, and leads to very wide extrapolation confidence intervals. However the alternative of evaluating the penalty over the whole real line amounts to asserting certainty that the function has some derivative zeroed away from the data, which is equally unreasonable. It is prefereable to build a model in which the same smoothness assumtions apply over both data and extrapolation intervals, but not over the whole real line. See example code for practical illustration.

Linear extrapolation is used for prediction that requires extrapolation (i.e. prediction outside the range of the interior k-m[1]+1 knots — the interval over which the penalty is evaluated). 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.

Shape constrained versions for use with scasm are specified by model terms like s(x,bs="sc",xt="m+") for a monotonic increasing function, or s(x,bs="sc",xt=c("m+","c-")) for increasing and concave (negative second derivative). The vector supplied for xt can contain any meaningful combination of "+" (positive), "m+" (increasing), "m-" (decreasing), "c+" (convex) and "c-" (concave). The constraints will only be processed when used with scasm. Constraints can be used in tensor product terms.

Value

An object of class "Bspline.smooth". See smooth.construct, for the elements that this object will contain.

WARNING

m[1] 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>. Extrapolation ideas joint with David Miller.

References

Wood, S.N. (2017) P-splines with derivative based penalties and tensor product smoothing of unevenly distributed data. Statistics and Computing. 27(4) 985-989 https://arxiv.org/abs/1605.02446 doi:10.1007/s112220169666x

See Also

p.spline

Examples

```
require(mgcv)
set.seed(5)
dat <- gamSim(1,n=400,dist="normal",scale=2)
bs <- "bs"
## note the double penalty on the s(x2) term...
b <- gam(y~s(x0,bs=bs,m=c(4,2))+s(x1,bs=bs)+s(x2,k=15,bs=bs,m=c(4,3,0))+
         s(x3,bs=bs,m=c(1,0)),data=dat,method="REML")
plot(b,pages=1)

## Extrapolation example, illustrating the importance of considering
## the penalty carefully if extrapolating...
f3 <- function(x) 0.2 * x^11 * (10 * (1 - x))^6 + 10 * (10 * x)^3 *
            (1 - x)^10 ## test function
n <- 100;x <- runif(n)
y <- f3(x) + rnorm(n)*2
## first a model with first order penalty over whole real line (red)
b0 <- gam(y~s(x,m=1,k=20),method="ML")
## now a model with first order penalty evaluated over (-.5,1.5) (black)
op <- options(warn=-1)
b <- gam(y~s(x,bs="bs",m=c(3,1),k=20),knots=list(x=c(-.5,0,1,1.5)),
         method="ML")
options(op)
## and the equivalent with same penalty over data range only (blue)
b1 <- gam(y~s(x,bs="bs",m=c(3,1),k=20),method="ML")
pd <- data.frame(x=seq(-.7,1.7,length=200))
fv <- predict(b,pd,se=TRUE)
ul <- fv$fit + fv$se.fit*2; ll <- fv$fit - fv$se.fit*2
plot(x,y,xlim=c(-.7,1.7),ylim=range(c(y,ll,ul)),main=
"Order 1 penalties: red tps; blue bs on (0,1); black bs on (-.5,1.5)")
## penalty defined on (-.5,1.5) gives plausible predictions and intervals
## over this range...
lines(pd$x,fv$fit);lines(pd$x,ul,lty=2);lines(pd$x,ll,lty=2)
fv <- predict(b0,pd,se=TRUE)
ul <- fv$fit + fv$se.fit*2; ll <- fv$fit - fv$se.fit*2
## penalty defined on whole real line gives constant width intervals away
## from data, as slope there must be zero, to avoid infinite penalty:
lines(pd$x,fv$fit,col=2)
lines(pd$x,ul,lty=2,col=2);lines(pd$x,ll,lty=2,col=2)
fv <- predict(b1,pd,se=TRUE)
ul <- fv$fit + fv$se.fit*2; ll <- fv$fit - fv$se.fit*2
## penalty defined only over the data interval (0,1) gives wild and wide
## extrapolation since penalty has been `turned off' outside data range:
lines(pd$x,fv$fit,col=4)
lines(pd$x,ul,lty=2,col=4);lines(pd$x,ll,lty=2,col=4)

## 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]]

## another example checking penalty numerically...
m <- c(4,2); k <- 15; b <- runif(k)
sm <- smoothCon(s(x,bs="bs",m=m,k=k),data.frame(x),
                scale.penalty=FALSE)[[1]]
sm$deriv <- m[2]
h0 <- 1e-3; xk <- sm$knots[(m[1]+1):(k+1)]
Xp <- PredictMat(sm,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

## ...repeated with uneven knot spacing...
m <- c(4,2); k <- 15; b <- runif(k)
## produce the required 20 unevenly spaced knots...
knots <- data.frame(x=c(-.4,-.3,-.2,-.1,-.001,.05,.15,
      .21,.3,.32,.4,.6,.65,.75,.9,1.001,1.1,1.2,1.3,1.4))
sm <- smoothCon(s(x,bs="bs",m=m,k=k),data.frame(x),
      knots=knots,scale.penalty=FALSE)[[1]]
sm$deriv <- m[2]
h0 <- 1e-3; xk <- sm$knots[(m[1]+1):(k+1)]
Xp <- PredictMat(sm,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 (2017) sections 5.3.1 and 5.3.2 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.
F	For class `"cr.smooth"` and `"cs.smooth"` objects `t(F)` transforms function values at the knots to second derivatives at the knots.
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. (2017) Generalized Additive Models: An Introduction with R (2nd edition). Chapman and Hall/CRC Press.

Examples

```
## 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)))
## or more simply
   b <- gam(y ~ s(x,bs="cc",k=12) + s(z),family=poisson,
                      knots=list(x=c(0,10)))

## plot results...
  par(mfrow=c(2,2))
  plot(x,y);plot(b,select=1,shade=TRUE);lines(x,f-mean(f),col=2)
  plot(b,select=2,shade=TRUE);plot(fitted(b),residuals(b))
```

---

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 constructed 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(b,theta=30)
vis.gam(b1,theta=30)

detach(eg)
```

---

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 factor.smooth for more genral alternatives for factor smooth interactions (including interactions of tensor product smooths with factors).

Usage

```
## S3 method for class 'fs.smooth.spec'
smooth.construct(object, data, knots)
## S3 method for class 'fs.interaction'
Predict.matrix(object, data)
```

Arguments

object	For the smooth.construct method a smooth specification object, usually generated by a term s(x,...,bs="fs",). May have a gamm attribute: see details. For the predict.Matrix method an object of class "fs.interaction" 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.
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. To facilitate this the constructor is called from gamm with an

attribute "gamm" attached to the smooth specification object. The result differs from that resulting from the case where this is not done.

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. Constructor is still called with a smooth specification object having a "gamm" attribute.

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. The contents of an "fs.interaction" object will depend on whether or not smooth.construct was called with an object with attribute gamm: see below.

Author(s)

Simon N. Wood <simon.wood@r-project.org> with input from Matteo Fasiolo.

See Also

factor.smooth, gamm, smooth.construct.sz.smooth.spec

Examples

```
library(mgcv)
set.seed(0)
## simulate data...
f0 <- function(x) 2 * sin(pi * x)
f1 <- function(x,a=2,b=-1) exp(a * x)+b
f2 <- 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)
x0 <- runif(n);x1 <- runif(n);x2 <- runif(n)
a <- rnorm(nf)*.2 + 2;b <- rnorm(nf)*.5
f <- f0(x0) + f1(x1,a[fac],b[fac]) + f2(x2)
fac <- factor(fac)
y <- 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...
```

```
bm <- gamm(y~s(x0)+ s(x1,fac,bs="fs",k=5)+s(x2,k=20))
plot(bm$gam,pages=1)
summary(bm$gam)

## Also efficient using bam(..., discrete=TRUE)
bd <- bam(y~s(x0)+ s(x1,fac,bs="fs",k=5)+s(x2,k=20),discrete=TRUE)
plot(bd,pages=1)
summary(bd)

## Could also use...
## b <- gam(y~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~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 `abs(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. The sign of `m[1]` determines whether a linear trend in the covariates is added to the Guassian process (positive), or not (negative). The latter ensures stationarity. `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

```
## S3 method for class 'gp.smooth.spec'
smooth.construct(object, data, knots)
## S3 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, $0 < \kappa \leq 2$ 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. Note that setting `r` to too small a value will lead to unpleasant results, as most points become all but independent (especially for the spherical model. Note: Wood 2017, Figure 5.20 right is based on a buggy implementation).

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 and avoid any co-linearity problems that might otherwise occur in the penalty null space basis of the term.
- Xu: A matrix of the unique covariate combinations for this smooth (the basis is constructed 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

Wood, S.N. (2017) Generalized Additive Models: an introduction with R (2nd ed). CRC/Taylor and Francis

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)
```

```
## compare non-stationary (b1) and stationary (b2)
b2 <- gam(y~s(x,z,bs="gp",k=50,m=c(-1,.5)),data=data)  ## sph stationary
vis.gam(b1,theta=30);vis.gam(b2,theta=30)
x <- seq(-1,2,length=200); z <- rep(.5,200)
pd <- data.frame(x=x,z=z)
plot(x,predict(b1,pd),type="l");lines(x,predict(b2,pd),col=2)
abline(v=c(0,1))
plot(predict(b1),predict(b2))

detach(eg)
```

---

smooth.construct.mrf.smooth.spec

Markov Random Field Smooths

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 numeric vector indexing the neighbours of the ith area (and should not include `i`). All indices are relative to `nb` itself, but can be translated using `names(nb)`. See example code. As an alternative each `nb[[i]]` can be an array of the names of the neighbours, but these will be converted to the arrays of numeric indices internally.

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 (2017) Section 5.4.2. See Wood (2017, section 5.8.1).

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. (2017) Generalized additive models: an introduction with R (2nd edition). 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")
plot(b,scheme=1)
## Compare to reduced rank version...
b <- gam(crime ~ s(district,bs="mrf",k=20,xt=xt),data=columb,method="REML")
plot(b,scheme=1)
## An important covariate added...
b <- gam(crime ~ s(district,bs="mrf",k=20,xt=xt)+s(income),
         data=columb,method="REML")
plot(b,scheme=c(0,1))

## plot fitted values by district
par(mfrow=c(1,1))
fv <- fitted(b)
names(fv) <- as.character(columb$district)
polys.plot(columb.polys,fv)

## Examine an example neighbourhood list - this one auto-generated from
## 'polys' above.

nb <- b$smooth[[1]]$xt$nb
head(nb)
names(nb) ## these have to match the factor levels of the smooth
## look at the indices of the neighbours of the first entry,
## named '0'...
nb[['0']] ## by name
nb[[1]]   ## same by index
## ... and get the names of these neighbours from their indices...
names(nb)[nb[['0']]]
b1 <- gam(crime ~ s(district,bs="mrf",k=20,xt=list(nb=nb))+s(income),
         data=columb,method="REML")
b1 ## fit unchanged
```

```
plot(b1) ## but now there is no information with which to plot the mrf
```

---

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 (but see also d.spline). 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",...) 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, d.spline

Examples

```
## see ?gam
## cyclic 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="cp") + s(z,bs="ps"),family=poisson)

## example with supplied knot ranges for x and z (can do just one)
  b <- gam(y ~ 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 <- gam(y ~ 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 <- seq(0,1,length=100); dat <- data.frame(x)
  sspec <- s(x,bs="ps")
  sspec$mono <- 1
  sm <- smoothCon(sspec,dat)[[1]]
  sm$deriv <- 1
  Xd <- PredictMat(sm,dat)
## 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$g.index])
## plot monotonic spline and its derivative
  par(mfrow=c(2,2))
  plot(x,sm$X%*%b,type="l",ylab="f(x)")
  plot(x,Xd%*%b,type="l",ylab="f'(x)")
## repeat for decrease...
  sspec$mono <- -1
  sm1 <- smoothCon(sspec,dat)[[1]]
  sm1$deriv <- 1
  Xd1 <- PredictMat(sm1,dat)
  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$mono <- 1
  sm <- smoothCon(sspec,dat,absorb.cons=TRUE)[[1]]
  sm$deriv <- 1
  Xd <- PredictMat(sm,dat)
  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]]
sm1$deriv <- 1
Xd1 <- PredictMat(sm1,dat)
plot(x,sm1$X%*%b,type="l",ylab="f(x)")
plot(x,Xd1%*%b,type="l",ylab="f'(x)")
```

---

smooth.construct.re.smooth.spec

Simple random effects in GAMs

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

```
## S3 method for class 're.smooth.spec'
smooth.construct(object, data, knots)
## S3 method for class 'random.effect'
Predict.matrix(object, data)
```

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 present.

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 `~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(~x-1)` being appended to the overall model matrix, while `s(x,v,w,bs="re")` would result in `model.matrix(~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.

Some models require differences between the coefficients corresponding to different levels of the same random effect. See `linear.functional.terms` for how to implement this.

If the random effect precision matrix is of the form $\sum_j \lambda_j S_j$ for known matrices S_j and unknown parameters λ_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.

The implementation is designed so that supplying random effect factor levels to `predict.gam` that were not levels of the factor when fitting, will result in the corresponding random effect (or interactions involving it) being set to zero (with zero standard error) for prediction. See `random.effects` for an example. This is achieved by the `Predict.matrix` method zeroing any rows of the prediction matrix involving factors that are `NA`. `predict.gam` will set any factor observation to `NA` if it is a level not present in the fit data.

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`

Examples

```
## 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 <- 2 + b[r] + rnorm(n)
r <- factor(r)
## fit model....
b <- gam(y ~ s(r,bs="re"),method="REML")
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 <- 2 + b[r] + rnorm(n)
r <- factor(r)
## known precision matrix components...
S <- 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")
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 λ'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. doi:10.1111/j.14679868.2008.00665.x

See Also

Predict.matrix.soap.film

Examples

```
require(mgcv)

##########################
## simple test function...
##########################

fsb <- list(fs.boundary())
nmax <- 100
## create some internal knots...
knots <- 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
y <- fs.test(v,w,b=1)
names(fsb[[1]]) <- c("v","w")
ind <- inSide(fsb,x=v,y=w) ## remove outsiders
y <- y + rnorm(n)*.3 ## add noise
y <- y[ind];v <- v[ind]; w <- w[ind]
n <- length(y)

par(mfrow=c(3,2))
## plot boundary with knot and data locations
plot(fsb[[1]]$v,fsb[[1]]$w,type="l");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)

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~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)

## Now fit the smooth...
bk <- gam(y~s(v,w,bs="so",xt=list(bnd=fsb,nmax=nmax)),knots=knots)
plot(bk) ## default plot

##########################################
## tensor product example...
##########################################

set.seed(9)
n <- 10000
v <- runif(n)*5-1;w<-runif(n)*2-1
t <- runif(n)
y <- fs.test(v,w,b=1)
y <- y + 4.2
y <- y^(.5+t)
fsb <- list(fs.boundary())
names(fsb[[1]]) <- c("v","w")
ind <- inSide(fsb,x=v,y=w) ## remove outsiders
y <- y[ind];v <- v[ind]; w <- w[ind]; t <- t[ind]
n <- length(y)
y <- y + rnorm(n)*.05 ## add noise
knots <- 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~ te(v,w,t,bs=c("sf","cr"),k=c(25,4),d=c(2,1),
          xt=list(list(bnd=fsb,nmax=nmax),NULL))+
          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 <- seq(-1,3.5,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)+4.2 ## truth

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")


#############################
# nested boundary example...
#############################

bnd <- list(list(x=0,y=0),list(x=0,y=0))
seq(0,2*pi,length=100) -> 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="l")
lines(bnd[[2]]$x,bnd[[2]]$y)

## setup knots
k <- 8
xm <- seq(-1,1,length=k);ym <- seq(-1,1,length=k)
x=rep(xm,k);y=rep(ym,rep(k,k))
ind <- inSide(bnd,x,y)
knots <- data.frame(x=x[ind],y=y[ind])
points(knots$x,knots$y)

## a test function

f1 <- function(x,y) {
  exp(-(x-.3)^2-(y-.3)^2)
}

## plot the test function within the domain
```

```
par(mfrow=c(2,3))
m<-100;n<-100
xm <- seq(-1,1,length=m);yn<-seq(-1,1,length=n)
x <- rep(xm,n);y<-rep(yn,rep(m,n))
ff <- f1(x,y)
ind <- inSide(bnd,x,y)
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 <- runif(300)*2-1;y <- runif(300)*2-1
ind <- inSide(bnd,x,y)
x <- x[ind];y <- y[ind]
n <- length(x)
z <- f1(x,y) + rnorm(n)*.1

## Fit a soap film smooth to the noisy data
nmax <- 60
b <- gam(z~s(x,y,k=c(30,15),bs="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),bs="sw",xt=list(bnd=bnd,nmax=nmax)),
   knots=knots,method="REML")
plot(ba)
vis.gam(ba,plot.type="contour")
```

`smooth.construct.sos.smooth.spec`

Splines on the sphere

Description

`gam` can use isotropic smooths on the sphere, via terms like `s(la,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. `m = -2` is the same but with first derivative penalization.

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 constructed by first stripping out duplicate locations).

UZ The matrix mapping the parameters of the reduced rank spline back to the parameters 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

Examples

```
require(mgcv)
set.seed(0)
n <- 400

f <- function(la,lo) { ## a test function...
  sin(lo)*cos(la-.3)
}

## generate with uniform density on sphere...
lo <- runif(n)*2*pi-pi ## longitude
la <- runif(3*n)*pi-pi/2
ind <- runif(3*n)<=cos(la)
la <- la[ind];
la <- la[1:n]

ff <- f(la,lo)
y <- 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)
gr <- expand.grid(la=lam,lo=lom)
fz <- f(gr$la,gr$lo)
zm <- matrix(fz,30,60)

require(mgcv)
dat <- data.frame(la = la *180/pi,lo = lo *180/pi,y=y)

## fit spline on sphere model...
```

```
bp <- gam(y~s(la,lo,bs="sos",k=60),data=dat)

## pure knot based alternative...
ind <- sample(1:n,100)
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)
```

`smooth.construct.sz.smooth.spec`
Constrained factor smooth interactions in GAMs

Description

Factor smooth interactions constructed to exclude main effects (and lower order factor smooth interactions). A smooth is constucted for each combination of the supplied factor levels. By appropriate application of sum to zero contrasts to equivalent smooth coefficients across factor levels, the required exclusion of lower order effects is achieved.

See factor.smooth for alternative factor smooth interactions.

Usage

```
## S3 method for class 'sz.smooth.spec'
smooth.construct(object, data, knots)
## S3 method for class 'sz.interaction'
Predict.matrix(object, data)
```

Arguments

`object`	For the `smooth.construct` method a smooth specification object, usually generated by a term `s(x,...,bs="sz",)`. For the `predict.Matrix` method an object of class `"sz.interaction"` 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`.
`knots`	a list containing any knots supplied for smooth basis setup.

Details

This class produces a smooth for each combination of the levels of the supplied factor variables. `s(fac,x,bs="sz")` produces a smooth of `x` for each level of `fac`, for example. The smooths are constrained to represent deviations from the main effect smooth, so that models such as

$$g(\mu_i) = f(x_i) + f_{k(i)}(x_i)$$

can be estimated in an identifiable manner, where $k(i)$ indicates the level of some factor that applies for the ith observation. Identifiability in this case is ensured by constraining the coefficients of the splines representing the f_k. In particular if β_{ki} is the ith coefficient of f_k then the constraints are $\sum_k \beta_{ki} = 0$.

Such sum to zero constraints are implemented using sum to zero contrasts: identity matrices with an extra row of -1s appended. Consider the case of a single factor first. The model matrix corresponding to a smooth per factor level is the row tensor product (see `tensor.prod.model.matrix`) of the model matrix for the factor, and the model matrix for the smooth. The contrast matrix is then the Kronecker product of the sum to zero contrast for the factor, and an identity matrix of dimension determined by the number of coefficients of the smooth.

If there are multiple factors then the overall model matrix is the row Kronecker product of all the factor model matrices and the smooth, while the contrast is the Kronecker product of all the sum-to-zero contrasts for the factors and a final identity matrix. Notice that this construction means that the main effects (and any interactions) of the factors are included in the factor level dependent smooths. In other words the individual smooths are not each centered. This means that adding main effects or interactions of the factors will lead to a rank deficient model.

The terms can have a smoothing parameter per smooth, or a single smoothing parameter for all the smooths. The latter is specified by giving the smooth term an `id`. e.g. `s(fac,x,bs="sz",id=1)`.

The basis for the smooths can be selected by supplying a list as the `xt` argument to `s`, with a `bs` item. e.g. `s(fac,x,xt=list(bs="cr"))` selectes the `"cr"` basis. The default is `"tp"`

The plot method for this class has two schemes. `scheme==0` is in colour, while `scheme==1` is black and white. Currently it only works for 1D smooths.

Value

An object of class `"sz.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`> with input from Matteo Fasiolo.

See Also

`gam.models`, `gamm`, factor.smooth

Examples

```
library(mgcv)
set.seed(0)
dat <- gamSim(4)
```

```
b <- gam(y ~ s(x2)+s(fac,x2,bs="sz")+s(x0),data=dat,method="REML")
plot(b,pages=1)
summary(b)

## Example involving 2 factors

f1 <- function(x2) 2 * sin(pi * x2)
f2 <- function(x2) exp(2 * x2) - 3.75887
f3 <- function(x2) 0.2 * x2^11 * (10 * (1 - x2))^6 + 10 * (10 * x2)^3 *
            (1 - x2)^10

n <- 600
x <- runif(n)
f1 <- factor(sample(c("a","b","c"),n,replace=TRUE))
f2 <- factor(sample(c("foo","bar"),n,replace=TRUE))

mu <- f3(x)
for (i in 1:3) mu <- mu + exp(2*(2-i)*x)*(f1==levels(f1)[i])
for (i in 1:2) mu <- mu + 10*i*x*(1-x)*(f2==levels(f2)[i])
y <- mu + rnorm(n)
dat <- data.frame(y=y,x=x,f1=f1,f2=f2)
b <- gam(y ~ s(x)+s(f1,x,bs="sz")+s(f2,x,bs="sz")+s(f1,f2,x,bs="sz",id=1),data=dat,method="REML")
plot(b,pages=1,scale=0)
```

`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, or construct models with odd nesting structures.

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 constructed 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

Examples

```
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?
summary(gam(y~s(x),method="REML")) ## yes!

## Fool bam(...,discrete=TRUE) into (strange) nested
## model fit...
set.seed(2) ## simulate some data...
dat <- gamSim(1,n=400,dist="normal",scale=2)
dat$x1a <- dat$x1 ## copy x1 so bam allows 2 copies of x1
## Following removes identifiability problem, by removing
## linear terms from second smooth, and then re-inserting
## the one that was not a duplicate (x2)...
b <- bam(y~s(x0,x1)+s(x1a,x2,m=c(2,0))+x2,data=dat,discrete=TRUE)

## example of knot based tprs...
k <- 10; m <- 2
y <- y[order(x)];x <- x[order(x)]
b <- gam(y~s(x,k=k,m=m),method="REML",
         knots=list(x=seq(0,1,length=k)))
X <- model.matrix(b)
par(mfrow=c(1,2))
plot(x,X[,1],ylim=range(X),type="l")
for (i in 2:ncol(X)) lines(x,X[,i],col=i)

## compare with eigen based (default)
b1 <- gam(y~s(x,k=k,m=m),method="REML")
X1 <- model.matrix(b1)
plot(x,X1[,1],ylim=range(X1),type="l")
for (i in 2:ncol(X1)) lines(x,X1[,i],col=i)
## see ?gam
```

smooth.info *Generic function to provide extra information about smooth specification*

Description

Takes a smooth specification object and adds extra basis specific information to it before smooth constructor called. Default method returns supplied object unmodified.

Usage

```
smooth.info(object)
```

Arguments

object	is a smooth specification object

Details

Sometimes it is necessary to know something about a smoother before it is constructed, beyond what is in the initial smooth specification object. For example, some smooth terms could be set up as tensor product smooths and it is useful for bam to take advantage of this when discrete covariate methods are used. However, bam needs to know whether a smoother falls into this category before it is constructed in order to discretize its covariates marginally instead of jointly. Rather than bam having a hard coded list of such smooth classes it is preferable for the smooth specification object to report this themselves. smooth.info method functions are the means for achieving this. When interpreting a gam formula the smooth.info function is applied to each smooth specification object as soon as it is produced (in interpret.gam0).

Value

A smooth specification object, which may be modified in some way.

Author(s)

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

References

Wood S.N. (2017) Generalized Additive Models: An Introduction with R (2nd edition). Chapman and Hall/CRC Press.

See Also

bam, smooth.construct, PredictMat

Examples

```
# See smooth.construct examples
spec <- s(a,bs="re")
class(spec)
spec$tensor.possible
spec <- smooth.info(spec)
spec$tensor.possible
```

---

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 (2017).

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.

B-splines `bs="bs"`. B-spline basis with integrated squared derivative penalties. The order of basis and penalty can be chosen separately, and several penalties of different orders can be applied. Somewhat like a derivative penalty version of P-splines. See b.spline for details.

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 (2006) 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 (2006).

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 predictor $f_1(x) + f_2(z) + f_3(x, z)$, we might use model formula `y~s(x)+s(z)+ti(x,z)` or `y~ti(x)+ti(z)+ti(x,z)`. A similar construction involving `te` terms instead will be much less statsitically 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).

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="sz"` Smooth factor interactions (see factor.smooth) are often produced using `by` variables (see `gam.models`), but it is often desirable to include smooths which represent the deviations from some main effect smooth that apply for each level of a factor (or combination of factors). See `smooth.construct.sz.smooth.spec` for details.

Random factor smooth interactions `bs="fs"` A special smoother class (see `smooth.construct.fs.smooth.spec`) 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 doi:10.1111/1467-9868.00374

Wood, S.N. (2017, 2nd ed) *Generalized Additive Models: an introduction with R*, CRC doi:10.1201/9781315370279

Wood, S.N. (2006) Low rank scale invariant tensor product smooths for generalized additive mixed models. Biometrics 62(4):1025-1036 doi:10.1111/j.15410420.2006.00574.x

Wood, S.N., M.V. Bravington and S.L. Hedley (2008) "Soap film smoothing", J.R.Statist.Soc.B 70(5), 931-955. doi:10.1111/j.14679868.2008.00665.x

Wood S.N., F. Scheipl and J.J. Faraway (2013) [online 2012] Straightforward intermediate rank tensor product smoothing in mixed models. Statistics and Computing. 23(3):341-360 doi:10.1007/s112220129314z

Wood, S.N. (2017) P-splines with derivative based penalties and tensor product smoothing of unevenly distributed data. Statistics and Computing. 27(4) 985-989 https://arxiv.org/abs/1605.02446 doi:10.1007/s112220169666x

See Also

s, te, t2, tprs, Duchon.spline, cubic.regression.spline, p.spline, d.spline, mrf, soap, Spherical.Spline, adaptive.smooth, user.defined.smooth, smooth.construct.re.smooth.spec, smooth.construct.gp.smooth.spec, factor.smooth.interaction

Examples

```
## see examples for gam and gamm
```

---

smooth2random *Convert a smooth to a form suitable for estimating as random effect*

Description

A generic function for converting mgcv smooth objects to forms suitable for estimation as random effects by e.g. lme. Exported mostly for use by other package developers.

Usage

```
smooth2random(object,vnames,type=1)
```

Arguments

object	an mgcv smooth object.
vnames	a vector of names to avoid as dummy variable names in the random effects form.
type	1 for lme, otherwise lmer.

Details

There is a duality between smooths and random effects which means that smooths can be estimated using mixed modelling software. This function converts standard mgcv smooth objects to forms suitable for estimation by lme, for example. A service routine for gamm exported for use by package developers. See examples for creating prediction matrices for new data, corresponding to the random and fixed effect matrices returned when type=2.

Value

A list.

rand	a list of random effects, including grouping factors, and a fixed effects matrix. Grouping factors, model matrix and model matrix name attached as attributes, to each element. Alternatively, for type=2 a list of random effect model matrices, each corresponding to an i.i.d. Gaussian random effect with a single variance component.
trans.D	A vector, trans.D, that transforms coefs, in order [rand1, rand2,... fix] back to original parameterization. If null, then taken as vector of ones. b.original = trans.U %*% (trans.D*b.fit).
trans.U	A matrix, trans.U, that transforms coefs, in order [rand1, rand2,... fix] back to original parameterization. If null, then not needed. If null then taken as identity.
Xf	A matrix for the fixed effects, if any.
fixed	TRUE/FALSE, indicating if term was unpenalized or not. If unpenalized then other stuff may not be returned (it's not a random effect).
rind	an index vector such that if br is the vector of random coefficients for the term, br[rind] is the coefs in order for this term.
pen.ind	index of which penalty penalizes each coefficient: 0 for unpenalized.

Author(s)

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

References

Wood S.N. (2017) Generalized Additive Models: An Introduction with R (2nd edition). Chapman and Hall/CRC Press.

See Also

gamm

Examples

```
## Simple type 1 'lme' style...
library(mgcv)
x <- runif(30)
sm <- smoothCon(s(x),data.frame(x=x))[[1]]
smooth2random(sm,"")

## Now type 2 'lme4' style...
z <- runif(30)
dat <- data.frame(x=x,z=z)
sm <- smoothCon(t2(x,z),dat)[[1]]
re <- smooth2random(sm,"",2)
str(re)
```

```
## For prediction after fitting we might transform parameters back to
## original parameterization using 'rind', 'trans.D' and 'trans.U',
## and call PredictMat(sm,newdata) to get the prediction matrix to
## multiply these transformed parameters by.
## Alternatively we could obtain fixed and random effect Prediction
## matrices corresponding to the results from smooth2random, which
## can be used with the fit parameters without transforming them.
## The following shows how...

s2rPred <- function(sm,re,data) {
## Function to aid prediction from smooths represented as type==2
## random effects. re must be the result of smooth2random(sm,...,type=2).
  X <- PredictMat(sm,data)   ## get prediction matrix for new data
  ## transform to r.e. parameterization
  if (!is.null(re$trans.U)) X <- X%*%re$trans.U
  X <- t(t(X)*re$trans.D)
  ## re-order columns according to random effect re-ordering...
  X[,re$rind] <- X[,re$pen.ind!=0]
  ## re-order penalization index in same way
  pen.ind <- re$pen.ind; pen.ind[re$rind] <- pen.ind[pen.ind>0]
  ## start return object...
  r <- list(rand=list(),Xf=X[,which(re$pen.ind==0),drop=FALSE])
  for (i in 1:length(re$rand)) { ## loop over random effect matrices
    r$rand[[i]] <- X[,which(pen.ind==i),drop=FALSE]
    attr(r$rand[[i]],"s.label") <- attr(re$rand[[i]],"s.label")
  }
  names(r$rand) <- names(re$rand)
  r
} ## s2rPred

## use function to obtain prediction random and fixed effect matrices
## for first 10 elements of 'dat'. Then confirm that these match the
## first 10 rows of the original model matrices, as they should...

r <- s2rPred(sm,re,dat[1:10,])
range(r$Xf-re$Xf[1:10,])
range(r$rand[[1]]-re$rand[[1]][1:10,])
```

---

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

```
smoothCon(object,data,knots=NULL,absorb.cons=FALSE,
          scale.penalty=TRUE,n=nrow(data),dataX=NULL,
          null.space.penalty=FALSE,sparse.cons=0,
          diagonal.penalty=FALSE,apply.by=TRUE,modCon=0)
PredictMat(object,data,n=nrow(data))
```

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.

`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`.

`modCon` force modification of any smooth supplied constraints. 0 - do nothing. 1 - delete supplied constraints, replacing with automatically generated ones. 2 - set fit and predict constraint to predict constraint. 3 - set fit and predict constraint to fit constraint.

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>

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]]
## use it to fit a regression spline model...
beta <- coef(lm(mcycle$accel~sm$X-1))
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))
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]]
E <- t(mroot(sm$S[[1]])) ## square root penalty
X <- rbind(sm$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
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,edge.correct=TRUE,reg=1e-3)
```

Arguments

`x`	a fitted model object of class `gam` as produced by `gam()`.
`edge.correct`	if the model was fitted with `edge.correct=TRUE` (see `gam.control`), then thereturned covariance matrix will be for the edge corrected log smoothing parameters.
`reg`	regularizer for Hessian - default is equivalent to prior variance of 1000 on log smoothing parameters.

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.

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. If `edge.correct=TRUE` and this was used in fitting then the edge corrected smoothing parameters are in attribute `lsp` of the returned matrix.

Author(s)

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

References

Wood, S.N., N. Pya and B. Saefken (2016), Smoothing parameter and model selection for general smooth models (with discussion). Journal of the American Statistical Association 111, 1548-1575 doi:10.1080/01621459.2016.1180986

See Also

gam, gam.vcomp

Examples

```
require(mgcv)
n <- 100
x <- runif(n);z <- runif(n)
y <- sin(x*2*pi) + rnorm(n)*.2
mod <- gam(y~s(x,bs="cc",k=10)+s(z),knots=list(x=seq(0,1,length=10)),
           method="REML")
sp.vcov(mod)
```

---

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)
```

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.

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

Examples

```
## an example of GCV based model selection as
## an alternative to stepwise selection, using
## shrinkage smoothers...
library(mgcv)
set.seed(0);n <- 400
dat <- gamSim(1,n=n,scale=2)
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~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~s(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~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

```
## S3 method for class 'gam'
summary(object, dispersion=NULL, freq=FALSE, re.test=TRUE, ...)

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

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 Poisson).
`freq`	By default p-values for parametric terms are calculated using the Bayesian estimated covariance matrix of the parameter estimators. If this is set to `TRUE` then the frequentist covariance matrix of the parameters is used instead.
`re.test`	Should tests be performed for random effect terms (including any term with a zero dimensional null space)? For large models these tests can be computationally expensive.
`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 $\mathbf{A}$ for the model fit. Residual degrees of freedom are taken as number of data minus model degrees of freedom. Let $\mathbf{P}_i$ be the matrix giving the parameters of the ith smooth when applied to the data (or pseudodata in the generalized case) and let $\mathbf{X}$ be the design matrix of the model. Then $tr(\mathbf{XP}_i)$ 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 $2\mathbf{A} - \mathbf{AA}$.

`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).

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.

`p.coeff`	is an array of estimates of the strictly parametric model coefficients.
`p.t`	is an array of the `p.coeff`'s divided by their standard errors.
`p.pv`	is an array of p-values for the null hypothesis that the corresponding parameter is zero. Calculated with reference to the t distribution with the estimated residual degrees of freedom for the model fit if the dispersion parameter has been estimated, and the standard normal if not.
`m`	The number of smooth terms in the model.
`chi.sq`	An array of test statistics for assessing the significance of model smooth terms. See details.
`s.pv`	An array of approximate p-values for the null hypotheses that each smooth term is zero. Be warned, these are only approximate.
`se`	array of standard error estimates for all parameter estimates.
`r.sq`	The adjusted r-squared for the model. Defined as the proportion of variance explained, where original variance and residual variance are both estimated using unbiased estimators. This quantity can be negative if your model is worse than a one parameter constant model, and can be higher for the smaller of two nested models! 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 constant).
`pTerms.chi.sq`	a Wald statistic for testing the null hypothesis that the each parametric term is zero.
`pTerms.pv`	p-values associated with the tests that each term is zero. For penalized fits these are approximate. The reference distribution is an appropriate chi-squared when the scale parameter is known, and is based on an F when it is not.
`cov.unscaled`	The estimated covariance matrix of the parameters (or estimators if `freq=TRUE`), divided by scale parameter.
`cov.scaled`	The estimated covariance matrix of the parameters (estimators if `freq=TRUE`).
`p.table`	significance table for parameters
`s.table`	significance table for smooths
`p.Terms`	significance table for parametric model terms

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. doi:10.1111/j.1467-9469.2011.00760.x

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 doi:10.1093/biomet/ast038

Wood, S.N. (2013b) On p-values for smooth components of an extended generalized additive model. Biometrika 100:221-228 doi:10.1093/biomet/ass048

Wood S.N. (2017) Generalized Additive Models: An Introduction with R (2nd edition). Chapman and Hall/CRC Press. doi:10.1201/9781315370279

See Also

gam, predict.gam, gam.check, anova.gam, gam.vcomp, sp.vcov

Examples

```
library(mgcv)
set.seed(0)

dat <- gamSim(1,n=200,scale=2) ## simulate data

b <- gam(y~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
b.d <- pmax(b.d,3) # can't have basis dimension less than 3!
bc<-gam(y~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)
b <- 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)),
         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)
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)
bg <- gam(y~ s(x0)+ s(x1)+s(x2)+s(x3),family=Gamma(link=log),
          data=dat,method="REML")
summary(bg)
```

t2 *Define alternative tensor product smooths in GAM formulae*

Description

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,pc=NULL)
```

Arguments

...	a list of variables that are the covariates that this smooth is a function of. Transformations whose form depends on the values of the data are best avoided here: e.g. t2(log(x),z) is fine, but t2(I(x/sd(x)),z) is not (see predict.gam).
k	the dimension(s) of the bases used to represent the smooth term. If not supplied then set to 5^d. 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.
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).

xt 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 paramsters, 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 auto-initialization. Over-rides values supplied in sp argument to gam. Ignored by gamm.

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.

pc If not NULL, signals a point constraint: the smooth should pass through zero at the point given here (as a vector or list with names corresponding to the smooth names). Never ignored if supplied. See identifiability.

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 non-zero 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:

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.

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. Statistics and Computing. 23(3):341-360 doi:10.1007/s112220129314z

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,

Examples

```
# 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);
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)))
truth<-matrix(test1(pr$x,pr$z),30,30)
f <- test1(x,z)
y <- f + rnorm(n)*0.2
b1<-gam(y~s(x,z))
persp(xs,zs,truth);title("truth")
vis.gam(b1);title("t.p.r.s")
b2<-gam(y~t2(x,z))
vis.gam(b2);title("tensor product")
b3<-gam(y~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)
f <- test2(u,v,w)
y <- f + rnorm(n)*0.2

## tensor product of 2D Duchon spline and 1D cr spline
m <- list(c(1,.5),0)
b <- gam(y~t2(v,w,u,k=c(30,5),d=c(2,1),bs=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))
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)

b <- gam(y~t2(v,w,u,k=c(25,5),d=c(2,1),bs=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

```
te(..., k=NA,bs="cr",m=NA,d=NA,by=NA,fx=FALSE,
               np=TRUE,xt=NULL,id=NULL,sp=NULL,pc=NULL)
ti(..., k=NA,bs="cr",m=NA,d=NA,by=NA,fx=FALSE,
               np=TRUE,xt=NULL,id=NULL,sp=NULL,mc=NULL,pc=NULL)
```

Arguments

`...` a list of variables that are the covariates that this smooth is a function of. Transformations whose form depends on the values of the data are best avoided here: e.g. `te(log(x),z)` is fine, but `te(I(x/sd(x)),z)` is not (see `predict.gam`).

`k` the dimension(s) of the bases used to represent the smooth term. If not supplied then set to `5^d`. 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.

`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).

`fx` indicates whether the term is a fixed d.f. regression spline (`TRUE`) or a penalized regression spline (`FALSE`).

`np` `TRUE` to use the 'normal parameterization' for a tensor product smooth. This 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.

`xt` 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 paramsters, 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 auto-initialization. Over-rides values supplied in sp argument to gam. Ignored by gamm.

mc — For ti smooths you can specify which marginals should have centering constraints 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.

pc — If not NULL, signals a point constraint: the smooth should pass through zero at the point given here (as a vector or list with names corresponding to the smooth names). Never ignored if supplied as a number. See identifiability. Can also be supplied as a list of lists to specify general linear constraints as described in scasm - ignored by other modelling routines.

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.

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 ~ 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). See Wood (2017, section 5.6.3).

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. (2006) Low rank scale invariant tensor product smooths for generalized additive mixed models. Biometrics 62(4):1025-1036 doi:10.1111/j.15410420.2006.00574.x

Wood S.N. (2017) Generalized Additive Models: An Introduction with R (2nd edition). Chapman and Hall/CRC Press. doi:10.1201/9781315370279

See Also

`s`,`gam`,`gamm`, `smooth.construct.tensor.smooth.spec`

Examples

```
# 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 <- runif(n)/20;z <- runif(n);
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)))
truth <- matrix(test1(pr$x,pr$z),30,30)
f <- test1(x,z)
y <- f + rnorm(n)*0.2
b1 <- gam(y~s(x,z))
persp(xs,zs,truth);title("truth")
vis.gam(b1);title("t.p.r.s")
b2 <- gam(y~te(x,z))
vis.gam(b2);title("tensor product")
b3 <- gam(y~ 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)
f <- test2(u,v,w)
y <- f + rnorm(n)*0.2
# tensor product of 2D Duchon spline and 1D cr spline
m <- list(c(1,.5),rep(0,0)) ## example of list form of m
b <- gam(y~te(v,w,u,k=c(30,5),d=c(2,1),bs=c("ds","cr"),m=m))
plot(b)
```

`tensor.prod.model.matrix`
Row Kronecker product/ tensor product smooth construction

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 (marginals and results can be sparse). The model matrix construction uses row Kronecker products.

Usage

```
tensor.prod.model.matrix(X)
tensor.prod.penalties(S)
a%.%b
```

Arguments

`X`	a list of model matrices for the marginal bases of a smooth. Items can be class `"matrix"` or `"dgCMatrix"`, but not a mixture of the two.
`S`	a list of penalties for the marginal bases of a smooth.
`a`	a matrix with the same number of rows as `A`.
`b`	a matrix with the same number of rows as `B`.

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!

`A%.%B` is the operator form of this 'row Kronecker product'.

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 (of the same class as the marginals), 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)
## Dense row Kronecker product example...
X <- list(matrix(0:3,2,2),matrix(c(5:8,0,0),2,3))
tensor.prod.model.matrix(X)
X[[1]]%.%X[[2]]

## sparse equivalent...
Xs <- lapply(X,as,"dgCMatrix")
tensor.prod.model.matrix(Xs)
Xs[[1]]%.%Xs[[2]]

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)
## Sparse equivalent...
Ss <- lapply(S,as,"dgCMatrix")
tensor.prod.penalties(Ss)
```

---

totalPenaltySpace — *Obtaining (orthogonal) basis for null space and range of the penalty matrix*

Description

INTERNAL function to obtain (orthogonal) basis for the null space and range space of the penalty, and obtain actual null space dimension components are roughly rescaled to avoid any dominating.

Usage

```
totalPenaltySpace(S, H, off, p)
```

Arguments

S	a list of penalty matrices, in packed form.
H	the coefficient matrix of an user supplied fixed quadratic penalty on the parameters of the GAM.
off	a vector where the i-th element is the offset for the i-th matrix.
p	total number of parameters.

Value

A list of matrix square roots such that `S[[i]]=B[[i]]%*%t(B[[i]])`.

Author(s)

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

---

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.

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 `ld` and `sd`. `ld` 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

Examples

```
require(mgcv)
## simulate some diagonals...
set.seed(19); k <- 7
ld <- 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])
```

---

trind.generator *Generates index arrays for upper triangular storage*

Description

Generates index arrays for upper triangular storage up to order four. Useful when working with higher order derivatives, which generate symmetric arrays. Mainly intended for internal use.

Usage

```
trind.generator(K = 2, ifunc=FALSE, reverse= !ifunc)
```

Arguments

K	positive integer determining the size of the array.
ifunc	if TRUE index functions are returned in place of index arrays.
reverse	should the reverse indices be computed? Probably not if ifunc==TRUE.

Details

Suppose that `m=1` and you fill an array using code like `for(i in 1:K) for(j in i:K) for(k in j:K) for(l in k:K) {a[,m] <- something; m <- m+1 }` and do this because actually the same "something" would be stored for any permutation of the indices i,j,k,l. Clearly in storage we have the restriction l>=k>=j>=i, but for access we want no restriction on the indices. `i4[i,j,k,l]` produces the appropriate `m` for unrestricted indices. `i3` and `i2` do the same for 3d and 2d arrays. If `ifunc==TRUE` then `i2`, `i3` and `i4` are functions, so `i4(i,j,k,l)` returns appropriate `m`. For high `K` the function versions save storage, but are slower.

If computed, the reverse indices pick out the unique elements of a symmetric array stored redundantly. The indices refer to the location of the elements when the redundant array is accessed as its underlying vector. For example the reverse indices for a 3 by 3 symmetric matrix are 1,2,3,5,6,9.

Value

A list where the entries `i1` to `i4` are arrays in up to four dimensions, containing K indexes along each dimension. If `ifunc==TRUE` index functions are returned in place of index arrays. If `reverse==TRUE` reverse indices `i1r` to `i4r` are returned (always as arrays).

Author(s)

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

Examples

```
library(mgcv)
A <- trind.generator(3,reverse=TRUE)

# All permutations of c(1, 2, 3) point to the same index (5)
A$i3[1, 2, 3]
A$i3[2, 1, 3]
A$i3[2, 3, 1]
A$i3[3, 1, 2]
A$i3[1, 3, 2]

## use reverse indices to pick out unique elements
## just for illustration...
A$i2;A$i2[A$i2r]
A$i3[A$i3r]


## same again using function indices...
A <- trind.generator(3,ifunc=TRUE)
A$i3(1, 2, 3)
A$i3(2, 1, 3)
A$i3(2, 3, 1)
A$i3(3, 1, 2)
A$i3(1, 3, 2)
```

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 and bam but not 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

p	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 $p = (a + b\exp(\theta))/(1 + \exp(\theta))$. 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.
a	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 ldTweedie 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.

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`.
`ls`	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`.

WARNINGS

As `p` and the scale parameter tend to 1 the Tweedie distribution tends towards a Poisson, however it remains continuously supported, meaning that the corresponding log likelihood does not converge on the Poisson log likelihood. For example, AIC can be radically different for Poisson and Tweedie models for which the fit is almost identical.

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 (2016), Smoothing parameter and model selection for general smooth models. Journal of the American Statistical Association 111, 1548-1575 doi:10.1080/01621459.2016.1180986

See Also

`ldTweedie`, `rTweedie`

Examples

```
library(mgcv)
set.seed(3)
n<-400
## Simulate data...
dat <- gamSim(1,n=n,dist="poisson",scale=.2)
dat$y <- rTweedie(exp(dat$f),p=1.3,phi=.5) ## Tweedie response

## Fit a fixed p Tweedie, with wrong link ...
b <- 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~s(x0)+s(x1)+s(x2)+s(x3),family=Tweedie(1.25,power(.1)),
          data=dat,method="REML")
plot(b1,pages=1)
print(b1)

## estimate p as part of fitting

b2 <- gam(y~s(x0)+s(x1)+s(x2)+s(x3),family=tw(),
          data=dat,method="REML")
plot(b2,pages=1)
print(b2)

rm(dat)
```

---

twlss *Tweedie location scale family*

Description

Tweedie family in which the mean, scale and power parameters can all depend on smooth linear predictors. Restricted to estimation via the extended Fellner Schall method of Wood and Fasiolo (2017). Only usable with gam. Tweedie distributions are exponential family with variance given by $\phi\mu^p$ where ϕ is a scale parameter, p a parameter (here between 1 and 2) and μ is the mean.

Usage

```
twlss(link=list("log","identity","identity"),a=1.01,b=1.99)
```

Arguments

link	The link function list: currently no choise.
a	lower limit on the power parameter relating variance to mean.
b	upper limit on power parameter.

Details

The first linear predictor defines the mean parameter μ (default link log). The Tweedie variance is given by $\phi\mu^p$. The second linear predictor defines the log scale parameter $\rho = \log(\phi)$ (no link choice). The third linear predictor, θ, defines $p = \{a + b\exp(\theta)\}/\{1 + \exp(\theta)\}$ (again with no choice of link).

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 ldTweedie for more on this behaviour.

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.

Value

An object inheriting from class general.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. and Fasiolo, M., (2017). A generalized Fellner-Schall method for smoothing parameter optimization with application to Tweedie location, scale and shape models. Biometrics, 73(4), pp.1071-1081. doi:10.1111/biom.12666

Wood, S.N., N. Pya and B. Saefken (2016). Smoothing parameter and model selection for general smooth models. Journal of the American Statistical Association 111, 1548-1575 doi:10.1080/01621459.2016.1180986

See Also

Tweedie, ldTweedie, rTweedie

Examples

```
library(mgcv)
set.seed(3)
n<-400
## Simulate data...
dat <- gamSim(1,n=n,dist="poisson",scale=.2)
dat$y <- rTweedie(exp(dat$f),p=1.3,phi=.5) ## Tweedie response

## Fit a fixed p Tweedie ...
b <- gam(list(y~s(x0)+s(x1)+s(x2)+s(x3),~1,~1),family=twlss(),
         data=dat)
plot(b,pages=1)
print(b)
```

```
eb <- exp(coef(b));nb <- length(eb)
eb[nb-1] ## scale
(1+2*eb[nb])/(1+eb[nb]) ## p

rm(dat)
```

---

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 need not be the same as on entry. It also returns an index attribute for relating the result back to the original matrix.

Usage

```
uniquecombs(x,ordered=FALSE)
```

Arguments

x	is an R matrix (numeric), or data frame.
ordered	set to TRUE to have the rows of the returned object in the same order regardless of input ordering.

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 covariates in a model.

When x has only one column then the routine uses unique and match to get the index. When there are multiple columns then it uses paste0 to produce labels for each row, which should be unique if the row is unique. Then unique and match can be used as in the single column case. Obviously the pasting is inefficient, but still quicker for large n than the C based code that used to be called by this routine, which had O(nlog(n)) cost. In principle a hash table based solution in C would be only O(n) and much quicker in the multicolumn case.

unique and duplicated, can 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 attempt 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.

WARNINGS

If a dataframe contains variables of a type other than numeric, logical, factor or character, which either have no `as.character` method, or whose `as.character` method is a many to one mapping, then the routine is likely to fail.

If the character representation of a dataframe variable (other than of class factor of character) contains `*` then in principle the method could fail (but with a warning).

Author(s)

Simon N. Wood <simon.wood@r-project.org> with thanks to Jonathan Rougier

See Also

`unique`, `duplicated`, `match`.

Examples

```
require(mgcv)

## matrix example...
X <- matrix(c(1,2,3,1,2,3,4,5,6,1,3,2,4,5,6,1,1,1),6,3,byrow=TRUE)
print(X)
Xu <- uniquecombs(X);Xu
ind <- attr(Xu,"index")
## find the value for row 3 of the original from Xu
Xu[ind[3],];X[3,]

## same with fixed output ordering
Xu <- uniquecombs(X,TRUE);Xu
ind <- attr(Xu,"index")
## find the value for row 3 of the original from Xu
Xu[ind[3],];X[3,]


## data frame example...
df <- data.frame(f=factor(c("er",3,"b","er",3,3,1,2,"b")),
      x=c(.5,1,1.4,.5,1,.6,4,3,1.7),
      bb = c(rep(TRUE,5),rep(FALSE,4)),
      fred = c("foo","a","b","foo","a","vf","er","r","g"),
      stringsAsFactors=FALSE)
uniquecombs(df)
```

`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, sandwich=FALSE, freq = FALSE, dispersion = NULL,unconditional=FALSE, ...)
```

Arguments

object	fitted model object of class gam as produced by gam().
sandwich	compute sandwich estimate of covariance matrix. Currently expensive for discrete bam fits.
freq	TRUE to return the frequentist covariance matrix of the parameter estimators, FALSE to return the Bayesian posterior covariance matrix of the parameters. The latter option includes the expected squared bias according to the Bayesian smoothing prior.
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$Ve, object$Vp or object$Vc (if available) from a gamObject, unless sandwich==TRUE in which case the sandwich estimate is computed (with or without the squared bias component).

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. (2017) Generalized Additive Models: An Introductio with R (2nd ed) CRC Press

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))
```

---

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

Arguments

x	a gam object, produced by gam()
view	an array containing the names of the two main effect terms to be displayed on the x and y dimensions of the plot. If omitted the first two suitable terms will be used. 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.
n.grid	The number of grid nodes in each direction used for calculating the plotted surface.
too.far	plot grid nodes that are too far from the points defined by the variables given in view can be excluded from the plot. too.far determines what is too far. The grid is scaled into the unit square along with the view variables and then grid nodes more than too.far from the predictor variables are excluded.
col	The colours for the facets of the plot. If this is NA then if se>0 the facets are transparent, otherwise the colour scheme specified in color is used. If col is not NA then it is used as the facet colour.
color	the colour scheme to use for plots when se<=0. One of "topo", "heat", "cm", "terrain", "gray" or "bw". Schemes "gray" and "bw" also modify the colors used when se>0.
contour.col	sets the colour of contours when using plot.type="contour". Default scheme used if NULL.

se	if less than or equal to zero then only the predicted surface is plotted, but if greater than zero, then 3 surfaces are plotted, one at the predicted values minus se standard errors, one at the predicted values and one at the predicted values plus se standard errors.
type	"link" to plot on linear predictor scale and "response" to plot on the response scale.
plot.type	one of "contour" or "persp".
zlim	a two item array giving the lower and upper limits for the z-axis scale. NULL to choose automatically.
nCol	The number of colors to use in color schemes.
lp	selects the linear predictor for models with more than one.
...	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

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.

Examples

```
library(mgcv)
set.seed(0)
n<-200;sig2<-4
x0 <- runif(n, 0, 1);x1 <- runif(n, 0, 1)
x2 <- runif(n, 0, 1)
y<-x0^2+x1*x2 +runif(n,-0.3,0.3)
g<-gam(y~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

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
```

XWXd *Internal functions for discretized model matrix handling*

Description

Routines for computing with discretized model matrices as described in Wood et al. (2017) and Li and Wood (2019).

Usage

```
XWXd(X,w,k,ks,ts,dt,v,qc,nthreads=1,drop=NULL,ar.stop=-1,ar.row=-1,ar.w=-1,
     lt=NULL,rt=NULL)
XWyd(X,w,y,k,ks,ts,dt,v,qc,drop=NULL,ar.stop=-1,ar.row=-1,ar.w=-1,lt=NULL)
Xbd(X,beta,k,ks,ts,dt,v,qc,drop=NULL,lt=NULL)
diagXVXd(X,V,k,ks,ts,dt,v,qc,drop=NULL,nthreads=1,lt=NULL,rt=NULL)
ijXVXd(i,j,X,V,k,ks,ts,dt,v,qc,drop=NULL,nthreads=1,lt=NULL,rt=NULL)
```

Arguments

`X`	A list of the matrices containing the unique rows of model matrices for terms of a full model matrix, or the model matrices of the terms margins. if term subsetting arguments `lt` and `rt` are non-NULL then this requires an `"lpip"` attribute: see details. The elements of `X` may be sparse matrices of class `"dgCMatrix"`, in which case the list requires attributes `"r"` and `"off"` defining reverse indices (see details).
`w`	An n-vector of weights
`y`	n-vector of data.
`beta`	coefficient vector.
`k`	A matrix whose columns are index n-vectors each selecting the rows of an X[[i]] required to create the full matrix.
`ks`	The ith term has index vectors `ks[i,1]:(ks[i,2]-1)`. The corresponing full model matrices are summed over.
`ts`	The element of `X` at which each model term starts.
`dt`	How many elements of `X` contribute to each term.
`v`	`v[[i]]` is Householder vector for ith term, if `qc[i]>0`.
`qc`	if `qc[i]>0` then term has a constraint.
`nthreads`	number of threads to use
`drop`	list of columns of model matrix/parameters to drop
`ar.stop`	Negative to ignore. Otherwise sum rows `(ar.stop[i-1]+1):ar.stop[i]` of the rows selected by `ar.row` and weighted by `ar.w` to get ith row of model matrix to use.
`ar.row`	extract these rows...
`ar.w`	weight by these weights, and sum up according to `ar.stop`. Used to implement AR models.
`lt`	use only columns of X corresponding to these model matrix terms (for left hand `X` in `XWXd`). If `NULL` set to `rt`.
`rt`	as `lt` for right hand `X`. If `NULL` set to `lt`. If `lt` and `rt` are `NULL` use all columns.

V	Coefficient covariance matrix.
i	vector of rows of XVX^T at which to evaluate.
j	vector of corresponding columns.

Details

These functions are really intended to be internal, but are exported so that they can be used in the initialization code of families without problem. They are primarily used by bam to implement the methods given in the references. XWXd produces X^TWX, XWy produces X^TWy, Xbd produces $X\beta$ and diagXVXd produces the diagonal of XVX^T, while ijXVXd evaluates the scattered i, j elements indexed in i and j.

The "lpip" attribute of X is a list of the coefficient indices for each term. Required if subsetting via lt and rt.

X can be a list of sparse matrices of class "dgCMatrix", in which case reverse indices are needed, mapping stored matrix rows to rows in the full matrix (that is the reverse of k which maps full matrix rows to the stored unique matrix rows). r is the same dimension as k while off is a list with as many elements as k has columns. r and off are supplied as attributes to X . For simplicity let r and off denote a single column and element corresponding to each other: then r[off[j]:(off[j+1]-1)] contains the rows of the full matrix corresponding to row j of the stored matrix. The reverse indices are essential for efficient computation with sparse matrices. See the example code for how to create them efficiently from the forward index matrix, k.

Author(s)

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

References

Wood, S.N., Li, Z., Shaddick, G. & Augustin N.H. (2017) Generalized additive models for gigadata: modelling the UK black smoke network daily data. Journal of the American Statistical Association. 112(519):1199-1210 doi:10.1080/01621459.2016.1195744

Li, Z & S.N. Wood (2019) Faster model matrix crossproducts for large generalized linear models with discretized covariates. Statistics and Computing. doi:10.1007/s11222019098642

Examples

```
library(mgcv);library(Matrix)
## simulate some data creating a marginal matrix sequence...
set.seed(0);n <- 4000
dat <- gamSim(1,n=n,dist="normal",scale=2)
dat$x4 <- runif(n)
dat$y <- dat$y + 3*exp(dat$x4*15-5)/(1+exp(dat$x4*15-5))
dat$fac <- factor(sample(1:20,n,replace=TRUE))
G <- gam(y ~ te(x0,x2,k=5,bs="bs",m=1)+s(x1)+s(x4)+s(x3,fac,bs="fs"),
         fit=FALSE,data=dat,discrete=TRUE)
p <- ncol(G$X)
## create a sparse version...
Xs <- list(); r <- G$kd*0; off <- list()
for (i in 1:length(G$Xd)) Xs[[i]] <- as(G$Xd[[i]],"dgCMatrix")
```

```
for (j in 1:nrow(G$ks)) { ## create the reverse indices...
  nr <- nrow(Xs[[j]]) ## make sure we always tab to final stored row
  for (i in G$ks[j,1]:(G$ks[j,2]-1)) {
    r[,i] <- (1:length(G$kd[,i]))[order(G$kd[,i])]
    off[[i]] <- cumsum(c(1,tabulate(G$kd[,i],nbins=nr)))-1
  }
}
attr(Xs,"off") <- off;attr(Xs,"r") <- r

par(mfrow=c(2,3))

beta <- runif(p)
Xb0 <- Xbd(G$Xd,beta,G$kd,G$ks,G$ts,G$dt,G$v,G$qc)
Xb1 <- Xbd(Xs,beta,G$kd,G$ks,G$ts,G$dt,G$v,G$qc)
range(Xb0-Xb1);plot(Xb0,Xb1,pch=".")

bb <- cbind(beta,beta+runif(p)*.3)
Xb0 <- Xbd(G$Xd,bb,G$kd,G$ks,G$ts,G$dt,G$v,G$qc)
Xb1 <- Xbd(Xs,bb,G$kd,G$ks,G$ts,G$dt,G$v,G$qc)
range(Xb0-Xb1);plot(Xb0,Xb1,pch=".")

p <- length(beta) ## extract full model matrix...
X <- matrix(Xbd(G$Xd,diag(p),G$kd,G$ks,G$ts,G$dt,G$v,G$qc),ncol=p)

w <- runif(n)
XWy0 <- XWyd(G$Xd,w,y=dat$y,G$kd,G$ks,G$ts,G$dt,G$v,G$qc)
XWy1 <- XWyd(Xs,w,y=dat$y,G$kd,G$ks,G$ts,G$dt,G$v,G$qc)
range(XWy1-XWy0);plot(XWy1,XWy0,pch=".")

yy <- cbind(dat$y,dat$y+runif(n)-.5)
XWy0 <- XWyd(G$Xd,w,y=yy,G$kd,G$ks,G$ts,G$dt,G$v,G$qc)
XWy1 <- XWyd(Xs,w,y=yy,G$kd,G$ks,G$ts,G$dt,G$v,G$qc)
range(XWy1-XWy0);plot(XWy1,XWy0,pch=".")

A <- XWXd(G$Xd,w,G$kd,G$ks,G$ts,G$dt,G$v,G$qc)
B <- XWXd(Xs,w,G$kd,G$ks,G$ts,G$dt,G$v,G$qc)
D <- crossprod(X,w*X) ## direct computation
range(A-D)
range(A-B);plot(A,B,pch=".")
## compute some cross product terms only...
A <- XWXd(G$Xd,w,G$kd,G$ks,G$ts,G$dt,G$v,G$qc,lt=1:3,rt=4:5)
range(A-D[1:nrow(A),(nrow(A)+1):ncol(D)])

V <- crossprod(matrix(runif(p*p),p,p))
ii <- c(20:30,100:200)
jj <- c(50:90,150:160)
V[ii,jj] <- 0;V[jj,ii] <- 0
d1 <- diagXVXd(G$Xd,V,G$kd,G$ks,G$ts,G$dt,G$v,G$qc)
Vs <- as(V,"dgCMatrix")
d2 <- diagXVXd(Xs,Vs,G$kd,G$ks,G$ts,G$dt,G$v,G$qc)
range(d1-d2);plot(d1,d2,pch=".")
```

ziP *GAM zero-inflated (hurdle) Poisson regression family*

Description

Family for use with gam or bam, 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 (θ_1 and θ_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.

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 θ_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 (2016), Smoothing parameter and model selection for general smooth models. Journal of the American Statistical Association 111, 1548-1575 doi:10.1080/01621459.2016.1180986

See Also

ziplss

Examples

```
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)
   ## generate from zero truncated Poisson, given presence...
   y[ind] <- qpois(runif(np,dpois(0,lambda[ind]),1),lambda[ind])
   y
}

library(mgcv)
## Simulate some ziP data...
set.seed(1);n<-400
dat <- gamSim(1,n=n)
dat$y <- rzip(dat$f/4-1)

b <- gam(y~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()
b0 <- gam(y~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 <- gam(y~s(x0)+s(x1)+s(x2)+s(x3),family=ziP(b=.3),data=dat)
```

`ziplss` *Zero inflated (hurdle) Poisson location-scale model family*

Description

The `ziplss` family implements a zero inflated (hurdle) Poisson model in which one linear predictor controls the probability of presence and the other controls the mean given presence. Useable only with `gam`, 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"))
zipll(y,g,eta,deriv=0)
```

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 complementary log log of probability of presence.
`y`	response
`g`	gamma vector
`eta`	eta vector
`deriv`	number of derivatives to compute

Details

ziplss is 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 complementary log log of probability of presence.. 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 penalize towards constants, thereby avoiding extreme estimates when the data are uninformative.

zipll is a function used by ziplss, exported only to allow external use of the ziplss family. It is not usually called directly.

Value

For ziplss An object inheriting from class general.family.

WARNINGS

Zero inflated models are often over-used. Having many 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 (2016), Smoothing parameter and model selection for general smooth models. Journal of the American Statistical Association 111, 1548-1575 doi:10.1080/01621459.2016.1180986

Examples

```
library(mgcv)
## simulate some data...
f0 <- function(x) 2 * sin(pi * x); f1 <- function(x) exp(2 * x)
f2 <- function(x) 0.2 * x^11 * (10 * (1 - x))^6 + 10 *
            (10 * x)^3 * (1 - x)^10
n <- 500;set.seed(5)
x0 <- runif(n); x1 <- runif(n)
x2 <- runif(n); x3 <- runif(n)
```

```
## Simulate probability of potential presence...
eta1 <- f0(x0) + f1(x1) - 3
p <- binomial()$linkinv(eta1)
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))

## Fit ZIP model...
b <- gam(list(y~s(x2)+s(x3),~s(x0)+s(x1)),family=ziplss())
b$outer.info ## check convergence

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

Index

∗ **Functional linear model**
bam, 8
bam.update, 14
gam, 60
jagam, 135
scasm, 234
∗ **Generalized Additive Model**
bam, 8
bam.update, 14
gam, 60
jagam, 135
scasm, 234
∗ **Generalized Cross Validation**
bam, 8
bam.update, 14
gam, 60
jagam, 135
∗ **Generalized ridge regression**
bam, 8
bam.update, 14
gam, 60
jagam, 135
∗ **P-spline**
bam, 8
bam.update, 14
gam, 60
jagam, 135
∗ **Penalized GLM**
bam, 8
bam.update, 14
gam, 60
jagam, 135
scasm, 234
∗ **Penalized regression spline**
bam, 8
bam.update, 14
gam, 60
jagam, 135
scasm, 234
∗ **Penalized regression**
bam, 8
bam.update, 14
gam, 60
jagam, 135
scasm, 234
∗ **Shape constraint**
scasm, 234
∗ **Smoothing parameter selection**
bam, 8
bam.update, 14
gam, 60
jagam, 135
scasm, 234
∗ **Spline smoothing**
bam, 8
bam.update, 14
gam, 60
jagam, 135
scasm, 234
∗ **Varying coefficient model**
bam, 8
bam.update, 14
gam, 60
jagam, 135
scasm, 234
∗ **hplot**
exclude.too.far, 46
plot.gam, 197
polys.plot, 202
vis.gam, 334
∗ **linear program**
lp, 150
∗ **models**
anova.gam, 5
bam, 8
bam.update, 14
bandchol, 16
bcg, 17

betar, 19
blas.thread.test, 21
choldrop, 22
choose.k, 24
clog, 27
cnorm, 28
cox.ph, 33
cox.pht, 38
cpois, 40
cSplineDes, 42
dpnorm, 44
dppois, 45
extract.lme.cov, 47
factor.smooth, 49
family.mgcv, 50
FFdes, 52
fix.family.link, 53
fixDependence, 54
formula.gam, 56
formXtViX, 58
fs.test, 59
gam, 60
gam.check, 71
gam.control, 73
gam.convergence, 75
gam.fit, 77
gam.fit3, 78
gam.mh, 82
gam.models, 84
gam.outer, 91
gam.scale, 93
gam.selection, 94
gam.side, 97
gam.vcomp, 99
gam2objective, 101
gamm, 104
gammals, 111
gamObject, 112
gamSim, 116
gaulss, 117
get.var, 118
gevlss, 119
gfam, 121
ginla, 124
gumbls, 127
identifiability, 129
influence.gam, 131
initial.sp, 132
inSide, 133
interpret.gam, 134
jagam, 135
k.check, 140
ldTweedie, 143
linear.functional.terms, 144
logLik.gam, 149
magic, 153
magic.post.proc, 158
mchol, 159
mgcv.FAQ, 160
mgcv.package, 162
mgcv.parallel, 164
model.matrix.gam, 169
mono.con, 170
mroot, 171
multinom, 172
mvn, 174
NCV, 175
negbin, 178
new.name, 181
notExp, 182
notExp2, 183
null.space.dimension, 184
ocat, 185
one.se.rule, 187
pcls, 188
pdIdnot, 192
pdTens, 193
pen.edf, 195
place.knots, 196
plot.gam, 197
polys.plot, 202
predict.bam, 204
predict.gam, 207
Predict.matrix, 213
Predict.matrix.cr.smooth, 214
Predict.matrix.soap.film, 216
print.gam, 218
psum.chisq, 219
qq.gam, 221
residuals.gam, 226
rmvn, 228
Rrank, 229
rTweedie, 230
s, 231
scasm, 234
scat, 239

sdiag, 240
single.index, 244
slanczos, 248
smooth.construct, 250
smooth.construct.ad.smooth.spec, 255
smooth.construct.bs.smooth.spec, 258
smooth.construct.cr.smooth.spec, 261
smooth.construct.ds.smooth.spec, 263
smooth.construct.fs.smooth.spec, 266
smooth.construct.gp.smooth.spec, 268
smooth.construct.mrf.smooth.spec, 271
smooth.construct.ps.smooth.spec, 274
smooth.construct.re.smooth.spec, 277
smooth.construct.so.smooth.spec, 279
smooth.construct.sos.smooth.spec, 285
smooth.construct.sz.smooth.spec, 288
smooth.construct.t2.smooth.spec, 290
smooth.construct.tensor.smooth.spec, 291
smooth.construct.tp.smooth.spec, 293
smooth.info, 296
smooth2random, 300
smoothCon, 302
sp.vcov, 305
spasm.construct, 306
step.gam, 307
summary.gam, 308
t2, 313
te, 317
tensor.prod.model.matrix, 322
trichol, 324
Tweedie, 327
twlss, 329
uniquecombs, 331
vcov.gam, 332
vis.gam, 334
XWXd, 336
ziP, 340
ziplss, 342

∗ **package**
mgcv.package, 162
mgcv.parallel, 164

∗ **regression**
anova.gam, 5
bam, 8
bam.update, 14
bandchol, 16
bcg, 17
betar, 19
blas.thread.test, 21
choldrop, 22
choose.k, 24
clog, 27
cnorm, 28
cox.ph, 33
cox.pht, 38
cpois, 40
cSplineDes, 42
dpnorm, 44
dppois, 45
extract.lme.cov, 47
factor.smooth, 49
family.mgcv, 50
FFdes, 52
fix.family.link, 53
fixDependence, 54
formula.gam, 56
formXtViX, 58
fs.test, 59
gam, 60
gam.check, 71
gam.control, 73
gam.convergence, 75
gam.fit, 77
gam.fit3, 78
gam.mh, 82
gam.models, 84
gam.outer, 91
gam.scale, 93
gam.selection, 94
gam.side, 97
gam.vcomp, 99

gam2objective, 101
gamm, 104
gammals, 111
gamObject, 112
gamSim, 116
gaulss, 117
get.var, 118
gevlss, 119
gfam, 121
ginla, 124
gumbls, 127
identifiability, 129
influence.gam, 131
initial.sp, 132
inSide, 133
interpret.gam, 134
jagam, 135
k.check, 140
ldTweedie, 143
linear.functional.terms, 144
logLik.gam, 149
magic, 153
magic.post.proc, 158
mchol, 159
mgcv.FAQ, 160
mgcv.package, 162
mgcv.parallel, 164
missing.data, 167
model.matrix.gam, 169
mono.con, 170
mroot, 171
multinom, 172
mvn, 174
NCV, 175
negbin, 178
new.name, 181
notExp, 182
notExp2, 183
null.space.dimension, 184
ocat, 185
one.se.rule, 187
pcls, 188
pdIdnot, 192
pdTens, 193
pen.edf, 195
place.knots, 196
plot.gam, 197
polys.plot, 202
predict.bam, 204
predict.gam, 207
Predict.matrix, 213
Predict.matrix.cr.smooth, 214
Predict.matrix.soap.film, 216
print.gam, 218
psum.chisq, 219
qq.gam, 221
random.effects, 223
residuals.gam, 226
rmvn, 228
Rrank, 229
rTweedie, 230
s, 231
scasm, 234
scat, 239
sdiag, 240
single.index, 244
slanczos, 248
smooth.construct, 250
smooth.construct.ad.smooth.spec, 255
smooth.construct.bs.smooth.spec, 258
smooth.construct.cr.smooth.spec, 261
smooth.construct.ds.smooth.spec, 263
smooth.construct.fs.smooth.spec, 266
smooth.construct.gp.smooth.spec, 268
smooth.construct.mrf.smooth.spec, 271
smooth.construct.ps.smooth.spec, 274
smooth.construct.re.smooth.spec, 277
smooth.construct.so.smooth.spec, 279
smooth.construct.sos.smooth.spec, 285
smooth.construct.sz.smooth.spec, 288
smooth.construct.t2.smooth.spec, 290
smooth.construct.tensor.smooth.spec, 291

smooth.construct.tp.smooth.spec, 293
smooth.info, 296
smooth.terms, 297
smooth2random, 300
smoothCon, 302
sp.vcov, 305
spasm.construct, 306
step.gam, 307
summary.gam, 308
t2, 313
te, 317
tensor.prod.model.matrix, 322
trichol, 324
Tweedie, 327
twlss, 329
uniquecombs, 331
vcov.gam, 332
vis.gam, 334
XWXd, 336
ziP, 340
ziplss, 342

∗ **shape constraint**
lp, 150

∗ **smooth**
anova.gam, 5
bam, 8
bam.update, 14
bandchol, 16
blas.thread.test, 21
choldrop, 22
cSplineDes, 42
dpnorm, 44
dppois, 45
extract.lme.cov, 47
FFdes, 52
formula.gam, 56
formXtViX, 58
fs.test, 59
gam, 60
gam.check, 71
gam.control, 73
gam.convergence, 75
gam.fit, 77
gam.fit3, 78
gam.mh, 82
gam.outer, 91
gam.scale, 93
gam.vcomp, 99
gam2objective, 101
gamm, 104
gamObject, 112
gamSim, 116
get.var, 118
ginla, 124
influence.gam, 131
initial.sp, 132
inSide, 133
interpret.gam, 134
jagam, 135
k.check, 140
logLik.gam, 149
magic, 153
magic.post.proc, 158
mgcv.package, 162
mgcv.parallel, 164
model.matrix.gam, 169
mono.con, 170
mroot, 171
new.name, 181
notExp, 182
notExp2, 183
one.se.rule, 187
pcls, 188
pdIdnot, 192
pdTens, 193
pen.edf, 195
place.knots, 196
plot.gam, 197
polys.plot, 202
predict.bam, 204
predict.gam, 207
Predict.matrix, 213
Predict.matrix.soap.film, 216
print.gam, 218
psum.chisq, 219
qq.gam, 221
residuals.gam, 226
Rrank, 229
s, 231
scasm, 234
sdiag, 240
slanczos, 248
smooth.construct, 250
smooth.construct.so.smooth.spec, 279

smooth.info, 296
smooth2random, 300
smoothCon, 302
sp.vcov, 305
spasm.construct, 306
summary.gam, 308
t2, 313
te, 317
tensor.prod.model.matrix, 322
trichol, 324
vcov.gam, 332
vis.gam, 334
XWXd, 336

∗ **tensor product smoothing**
bam, 8
bam.update, 14
gam, 60
jagam, 135

∗ **thin plate spline**
bam, 8
bam.update, 14
gam, 60
jagam, 135

%.% *(*tensor.prod.model.matrix*)*, 322

adaptive.smooth, *251*, *275*, *299, 300*
adaptive.smooth *(*smooth.construct.ad.smooth.spec*)*, 255
AIC, *95*, *149, 150*
AIC.gam *(*logLik.gam*)*, 149
anova.gam, 5, *88*, *162, 163*, *224*, *312*
anova.glm, *7*

b.spline, *298*
b.spline *(*smooth.construct.bs.smooth.spec*)*, 258
bam, 8, *14, 15*, *17*, *19*, *21*, *27, 28*, *38*, *40*, *50, 51*, *60*, *63, 64*, *75, 76*, *82*, *93*, *121*, *124, 125*, *138*, *149*, *162–165*, *177*, *179*, *185*, *204–206*, *239*, *272*, *278*, *296*, *303*, *327*, *338*, *340*
bam.update, 14
bandchol, 16, *325*
bcg, 17, *51*
betar, 19, *51*
blas.thread.test, 21
bug.reports.mgcv, 22

choldrop, 22, *125*
cholup *(*choldrop*)*, 22
choose.k, 24, *56*, *60*, *65*, *71–73*, *85*, *141*, *163*, *222*, *231*, *313*, *318*
clog, *19*, 26, *51*
cnorm, *19*, 28, *35*, *51*
coef.pdIdnot *(*pdIdnot*)*, 192
coef.pdTens *(*pdTens*)*, 193
columb, 30
columb.polys, *203*, *272*
concurvity, 32
contour, *335*
corClasses, *105*
corMatrix.pdIdnot *(*pdIdnot*)*, 192
cox.ph, 33, *38*, *51*, *226*
cox.pht, *33*, *35*, 38
cpois, *19*, 40, *51*
cSplineDes, 42, *275*, *292*
cubic.regression.spline, *251*, *297*, *300*
cubic.regression.spline *(*smooth.construct.cr.smooth.spec*)*, 261
cyclic.cubic.spline, *297*
cyclic.cubic.spline *(*smooth.construct.cr.smooth.spec*)*, 261
cyclic.p.spline, *43*, *298*
cyclic.p.spline *(*smooth.construct.ps.smooth.spec*)*, 274

d.mvt *(*rmvn*)*, 228
d.spline, *274, 275*, *300*
d.spline *(*smooth.construct.bs.smooth.spec*)*, 258
dDeta, 43
detectCores, *11*
diagXVXd *(*XWXd*)*, 336
Dim.pdIdnot *(*pdIdnot*)*, 192
dmvn *(*rmvn*)*, 228
dpnorm, 44
dppois, 45
drop1, *5*
Duchon.spline, *285*, *287*, *297*, *300*, *313*, *318*, *343*
Duchon.spline *(*smooth.construct.ds.smooth.spec*)*, 263

duplicated, *331, 332*

exclude.too.far, 46
extract.lme.cov, 47
extract.lme.cov2, *58*
extract.lme.cov2 *(*extract.lme.cov*)*, 47

factor, *86*
factor.smooth, 49, *266, 267*, *288, 289*, *299*
factor.smooth.interaction, *86*, *199*, *300*
family, *8*, *50*, *61*, *85*, *136*, *234*
family.mgcv, *8*, *13*, 50, *60, 61*, *162*, *234*
feasible *(*lp*)*, 150
FFdes, 52
fix.family.link, 53, *80*
fix.family.ls *(*fix.family.link*)*, 53
fix.family.qf *(*fix.family.link*)*, 53
fix.family.rd *(*fix.family.link*)*, 53
fix.family.var *(*fix.family.link*)*, 53
fixDependence, 54, *97*
formula.gam, *8*, 56, *61*, *105*, *135*, *172*, *174*, *234*
formXtViX, *48*, 58
fs.boundary *(*fs.test*)*, 59
fs.test, 59
function.predictors *(*linear.functional.terms*)*, 144

gam, *7, 8*, *11–13*, *17*, *19*, *21*, *27, 28*, *32–34*, *38*, *40*, *50, 51*, *54*, *56*, *58*, 60, *71*, *73*, *75–80*, *82*, *85*, *87*, *91–97*, *99*, *101*, *104–107*, *111*, *113, 114*, *116, 117*, *119–121*, *124, 125*, *127, 128*, *131, 132*, *134, 135*, *137, 138*, *140, 141*, *145*, *149*, *156*, *160–163*, *165*, *167*, *169*, *172*, *174*, *177–179*, *185*, *188*, *201*, *209, 210*, *214–216*, *219*, *222*, *224*, *226*, *232, 233*, *236*, *239*, *244*, *250, 251*, *253*, *255, 256*, *258*, *261–264*, *266*, *269*, *272*, *274*, *277, 278*, *280*, *285*, *290*, *292–294*, *297*, *304*, *306, 307*, *312–314*, *316*, *319, 320*, *327*, *329*, *333*, *335*, *340*, *342, 343*
gam.check, *7*, *13*, *24*, *56*, *60*, *65*, *67*, 71, *141*, *163*, *236*, *312*
gam.control, *9*, *13*, *61*, *67*, 73, *77, 78*, *93, 94*, *136*, *163*, *165*, *177*, *235*, *304, 305*
gam.convergence, 75, *163*
gam.fit, 77
gam.fit3, *54*, *64*, *74*, 78, *78*, *91, 92*, *101*
gam.fit5.post.proc, 81
gam.mh, 82
gam.models, *8*, *10*, *13*, *49*, *56, 57*, *60–62*, *67*, 84, *98*, *105*, *121*, *135*, *163*, *167*, *224, 225*, *232*, *234*, *236*, *289*, *299*, *314*, *318*
gam.outer, *75*, 91, *101*, *132*
gam.performance *(*gam.convergence*)*, 75
gam.reparam, 92
gam.scale, *74*, 93
gam.selection, *13*, *60*, *67*, 94, *163*, *307*
gam.side, *13*, *57*, *67*, 97
gam.vcomp, *87*, 99, *167*, *224, 225*, *278*, *306*, *312*
gam2derivative *(*gam2objective*)*, 101
gam2objective, 101
gamlss.etamu, 102
gamlss.gH, 103
gamm, *47–50*, *58*, *60*, *64*, *71, 72*, *74*, *87*, *93*, 104, *116, 117*, *134, 135*, *138*, *162, 163*, *179–184*, *193, 194*, *210*, *214*, *224, 225*, *233*, *252, 253*, *255*, *266, 267*, *272*, *278, 279*, *289*, *299–301*, *303*, *316*, *320*
gammals, *51*, 111
gamObject, *13*, *15*, *56*, *65*, *67*, *107*, 112, *138*, *162, 163*, *218*, *236*, *333*
gamSim, 116
gaulss, *51*, 117
gaussian, *149*, *175*
get.var, 118, *253*
gevlss, *52*, 119, *177*
gfam, *51*, 121
ginla, *82*, 124, *162*
glm, *8*, *50*, *56*, *61*, *85*, *105*, *136*, *149*, *163*, *234*
glm.control, *75*, *79*
glm.fit, *64*, *80*
gp.smooth, *298*
gp.smooth *(*smooth.construct.gp.smooth.spec*)*, 268
grey, *200*
grouped families *(*gfam*)*, 121
gumbls, *52*, 127

heat.colors, *200*

identifiability, 129, *232*, *314*, *319*
ijXVXd *(XWXd)*, 336
image, *200*, *335*
in.out, 130, *272, 273*
influence.gam, 131
initial.sp, 132
inSide, 133
interpret.gam, 134

jagam, 135, *162*

k.check, 140

ldetS, 141
ldTweedie, 143, *229*, *231*, *327, 328*, *330*
linear.functional.terms, *13*, *57*, *60*, *65*, *67*, *87*, 144, *163*, *208, 209*, *224*, *232*, *278*, *297*, *314*, *318*
lme, *47*, *104, 105*, *181*
lmeControl, *105*
locator, *280*
logDet.pdIdnot *(pdIdnot)*, 192
logLik, *150*
logLik.gam, 149
lp, 150
ls.size, 152

magic, *13*, *67*, *73*, *77, 78*, *80*, *92*, *101*, *108*, *132*, 153, *159*, *163*, *165*, *171*, *190*
magic.post.proc, *156*, 158
makeCluster, *11*
makeForkCluster, *11*
match, *331, 332*
mchol, 159
memory.limit, *107*
mgcv *(mgcv.package)*, 162
mgcv-package *(mgcv.package)*, 162
mgcv.FAQ, 160, *163*
mgcv.package, 162
mgcv.parallel, *13*, 164
mini.roots, 166
missing.data, 167
model.matrix.gam, 169
mono.con, 170, *190*
mrf, *163*, *199*, *203*, *298*, *300*
mrf *(smooth.construct.mrf.smooth.spec)*, 271
mroot, 171
multinom, *52*, *57*, *76*, 172
mvn, *52*, *57*, 174

na.action, *114*
nb, *51*, *178*
nb *(negbin)*, 178
NCV, *11, 12*, *63, 64*, *163*, *165*, 175
negbin, *13*, *51*, *67*, *108*, 178, *178*
new.name, 181
nlm, *74, 75*, *101*, *114*
notExp, 182, *183*
notExp2, *106, 107*, *182*, 183
notLog, *183*
notLog *(notExp)*, 182
notLog2, *182*, *192, 193*
notLog2 *(notExp2)*, 183
null.space.dimension, 184, *231*

ocat, *51*, *122*, *173*, 185
one.se.rule, 187
optim, *74, 75*, *101*, *114*
options, *184*
ordered, *86*
ordered.categorical *(ocat)*, 185

p.spline, *251*, *258*, *260*, *300*, *313*, *318*
p.spline *(smooth.construct.ps.smooth.spec)*, 274
parallel, *8*, *165*
parLapply, *10*, *205*
paste0, *331*
pbcseq, *38*
pcls, *152*, *163*, *171*, 188
pdConstruct.pdIdnot *(pdIdnot)*, 192
pdConstruct.pdTens *(pdTens)*, 193
pdFactor.pdIdnot *(pdIdnot)*, 192
pdFactor.pdTens *(pdTens)*, 193
pdIdnot, *182–184*, 192
pdMatrix.pdIdnot *(pdIdnot)*, 192
pdMatrix.pdTens *(pdTens)*, 193
pdTens, *108*, *182–184*, *192*, 193, *193*
pen.edf, 195, *315*
persp, *335*
persp.gam *(vis.gam)*, 334
place.knots, 196
plot.gam, *13*, *24*, *67*, *108*, *118*, *162, 163*, 197, *210*, *236*, *252*
poisson, *327*
polys.plot, 202, *273*

posterior.simulation (gam.mh), 82
power, *327*
predict.bam, 204
predict.gam, *7*, *11*, *13*, *67*, *106*, *108*, *112*, *118*, *128*, *145*, *160*, *162*, *163*, *169*, *186*, *201*, *204–206*, 207, *215*, *231*, *236*, *278*, *312*, *313*, *318*, *343*
Predict.matrix, 213, *214–216*, *250*, *253*, *304*
Predict.matrix.Bspline.smooth (smooth.construct.bs.smooth.spec), 258
Predict.matrix.cr.smooth, 214
Predict.matrix.cs.smooth (Predict.matrix.cr.smooth), 214
Predict.matrix.cyclic.smooth (Predict.matrix.cr.smooth), 214
Predict.matrix.duchon.spline (smooth.construct.ds.smooth.spec), 263
Predict.matrix.fs.interaction (smooth.construct.fs.smooth.spec), 266
Predict.matrix.gp.smooth (smooth.construct.gp.smooth.spec), 268
Predict.matrix.mrf.smooth (smooth.construct.mrf.smooth.spec), 271
Predict.matrix.pspline.smooth (Predict.matrix.cr.smooth), 214
Predict.matrix.random.effect (smooth.construct.re.smooth.spec), 277
Predict.matrix.sf.film (Predict.matrix.soap.film), 216
Predict.matrix.sf.wiggly (Predict.matrix.soap.film), 216
Predict.matrix.soap.film, 216, *282*
Predict.matrix.sos.smooth (smooth.construct.sos.smooth.spec), 285
Predict.matrix.sz.interaction (smooth.construct.sz.smooth.spec), 288
Predict.matrix.t2.smooth (Predict.matrix.cr.smooth), 214
Predict.matrix.tensor.smooth (Predict.matrix.cr.smooth), 214
Predict.matrix.tprs.smooth (Predict.matrix.cr.smooth), 214
Predict.matrix.ts.smooth (Predict.matrix.cr.smooth), 214
Predict.matrix2 (Predict.matrix), 213
PredictMat, *213*, *214*, *253*, *296*
PredictMat (smoothCon), 302
print.anova.gam (anova.gam), 5
print.gam, 218
print.gam.vcomp (gam.vcomp), 99
print.summary.gam (summary.gam), 308
psum.chisq, 219

qq.gam, *54*, *71*, *72*, *122*, 221
qr, *304*
quasi, *327*, *330*

r.mvt (rmvn), 228
random.effects, *60*, *87*, *162*, *163*, *199*, 223, *278*
residuals.gam, *71*, *221*, 226
rgb, *198*
rig, 227
rmvn, 228
Rrank, 229
rTweedie, 230, *328*, *330*

s, *13*, *24*, *49*, *56*, *61*, *65*, *67*, *74*, *76*, *85*, *86*, *94*, *105*, *108*, *129*, *135*, *145*, *163*, *167*, *215*, 231, *234*, *236*, *250*, *253*, *264*, *269*, *278*, *289*, *294*, *297*, *300*, *307*, *315*, *316*, *320*
scasm, *162*, *232*, 234, *256*, *258*, *259*, *319*
scat, *51*, 239
sdiag, 240
sdiag<- (sdiag), 240
sessionInfo, *22*
setwd, *136*
shash, *52*, 241
signal.regression (linear.functional.terms), 144
sim2jam (jagam), 135
single.index, 244
sink, *308*
Sl.inirep, 245
Sl.initial.repara (Sl.inirep), 245
Sl.repara, 246
Sl.setup, 247

slanczos, 248
smooth.construct, *64*, *97*, *115*, *163*, *213–215*, *232*, 250, *259*, *262*, *265*, *270*, *275*, *286*, *291*, *292*, *294*, *296*, *304*, *313*, *318*
smooth.construct.ad.smooth.spec, 255
smooth.construct.bs.smooth.spec, 258
smooth.construct.cc.smooth.spec, *197*
smooth.construct.cc.smooth.spec (smooth.construct.cr.smooth.spec), 261
smooth.construct.cp.smooth.spec (smooth.construct.ps.smooth.spec), 274
smooth.construct.cr.smooth.spec, 261
smooth.construct.cs.smooth.spec (smooth.construct.cr.smooth.spec), 261
smooth.construct.ds.smooth.spec, 263
smooth.construct.fs.smooth.spec, *49*, *50*, 266, *299*
smooth.construct.gp.smooth.spec, 268, *300*
smooth.construct.mrf.smooth.spec, 271
smooth.construct.ps.smooth.spec, 274
smooth.construct.re.smooth.spec, *87*, *100*, *167*, *224*, *225*, 277, *298*, *300*
smooth.construct.sc.smooth.spec, *236*, *256*
smooth.construct.sc.smooth.spec (smooth.construct.bs.smooth.spec), 258
smooth.construct.scad.smooth.spec (smooth.construct.ad.smooth.spec), 255
smooth.construct.sf.smooth.spec (smooth.construct.so.smooth.spec), 279
smooth.construct.so.smooth.spec, *216*, 279
smooth.construct.sos.smooth.spec, 285
smooth.construct.sw.smooth.spec (smooth.construct.so.smooth.spec), 279
smooth.construct.sz.smooth.spec, *49*, *50*, *267*, 288, *299*
smooth.construct.t2.smooth.spec, 290
smooth.construct.tensor.smooth.spec, *194*, 291, *320*, *323*
smooth.construct.tp.smooth.spec, 293
smooth.construct.ts.smooth.spec (smooth.construct.tp.smooth.spec), 293
smooth.construct2, *256*, *262*, *264*, *269*, *279*, *294*
smooth.construct2 (smooth.construct), 250
smooth.info, 296
smooth.terms, *13*, *56*, *60*, *64*, *67*, *162*, *163*, *225*, *231*, *236*, *291*, *292*, 297, *313*, *318*
smooth2random, 300
smoothCon, *250*, *253*, *279*, 302
smooths (smooth.terms), 297
soap, *298*, *300*
soap (smooth.construct.so.smooth.spec), 279
solve.pdIdnot (pdIdnot), 192
sp.vcov, *187*, 305, *312*
spasm.construct, 306
spasm.smooth (spasm.construct), 306
spasm.sp (spasm.construct), 306
Spherical.Spline, *199*, *265*, *298*, *300*
Spherical.Spline (smooth.construct.sos.smooth.spec), 285
step.gam, *96*, 307
summary.gam, *6*, *7*, *13*, *67*, *88*, *108*, *160*, *162*, *163*, *218*, *219*, *224*, *236*, 308
summary.pdIdnot (pdIdnot), 192
summary.pdTens (pdTens), 193

t.scaled (scat), 239
t2, *49*, *56*, *57*, *61*, *65*, *129*, *135*, *163*, *195*, *234*, *291*, *297*, *298*, *300*, 313
te, *13*, *24*, *49*, *56*, *57*, *61*, *65*, *67*, *76*, *85*, *86*, *94*, *105*, *108*, *129*, *135*, *145*, *163*, *193–195*, *215*, *233*, *234*, *250*, *261*, *274*, *291*, *292*, *297–300*, *313*, *316*, 317, *323*
tensor.prod.model.matrix, *289*, *319*, 321
tensor.prod.penalties, *319*
tensor.prod.penalties (tensor.prod.model.matrix), 322
termplot, *198–201*, *205*, *209*
ti, *49*, *56*, *57*, *61*, *65*, *85*, *98*, *129*, *135*, *163*, *234*, *297*, *298*

ti (te), 317
Tobit (cnorm), 28
totalPenaltySpace, 323
tprs, *9*, *56*, *62*, *136*, *160*, *185*, *235*, *249*, *251*, *258*, *264*, *270*, *297*, *300*
tprs (smooth.construct.tp.smooth.spec), 293
trichol, 324
trind.generator, *102–104*, 325
tw, *51*
tw (Tweedie), 327
Tweedie, *51*, *229*, *231*, 327, *330*
twlss, *52*, 329

unique, *331, 332*
uniquecombs, 331
user.defined.smooth, *56*, *297*, *300*
user.defined.smooth (smooth.construct), 250

vcov.gam, 332
vis.gam, *13*, *47*, *67*, *108*, *115*, *163*, *200, 201*, 334

Xbd (XWXd), 336
XWXd, 336
XWyd (XWXd), 336

ziP, *51*, 340
zipll (ziplss), 342
ziplss, *52*, *341*, 342