Package: gam (via r-universe)

October 28, 2026

Type Package

Title Generalized Additive Models

Date 2025-12-08

Version 1.22-7

Description Functions for fitting and working with generalized additive models, as described in chapter 7 of ``Statistical Models in S" (Chambers and Hastie (eds), 1991), and ``Generalized Additive Models" (Hastie and Tibshirani, 1990).

Depends R (>= 4.0), stats, splines, foreach

Suggests interp, testthat

License GPL-2

RoxygenNote 7.3.3

Encoding UTF-8

Imports methods

NeedsCompilation yes

Author Trevor Hastie [aut, cre], Balasubramanian Narasimhan [ctb]

Maintainer Trevor Hastie <hastie@stanford.edu>

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

Date/Publication 2025-12-10 09:40:02 UTC

RemoteUrl https://github.com/cran/gam

RemoteRef HEAD

RemoteSha 0d8107323a9abaf0baf27fda4a6927566fa538eb

Contents

gam-package 2
anova.Gam 2
gam 4
gam.control 8
gam.data 9

gam.exact 10
gam.lo 11
gam.random 13
gam.s 15
gam.scope 16
gam.smoothers 17
kyphosis 18
na.gam.replace 19
plot.Gam 20
predict.Gam 22
step.Gam 24

Index **27**

`gam-package` *Generalized Additive Models*

Description

This package provides functions for fitting and working with generalized additive models as described in chapter 7 of "Statistical Models in S" (Chambers and Hastie (eds), 1991) and "Generalized Additive Models" (Hastie and Tibshirani, 1990).

Author(s)

Trevor Hastie

`anova.Gam` *Analysis of Deviance for a Generalized Additive Model*

Description

Produces an ANODEV table for a set of GAM models, or else a summary for a single GAM model

Usage

```
## S3 method for class 'Gam'
anova(object, ..., test = c("Chisq", "F", "Cp"))

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

Arguments

object	a fitted Gam
...	other fitted Gams for anova
test	a character string specifying the test statistic to be used. Can be one of '"F"', '"Chisq"' or '"Cp"', with partial matching allowed, or 'NULL' for no test.
dispersion	a dispersion parameter to be used in computing standard errors

Details

These are methods for the functions anova or summary for objects inheriting from class Gam. See anova for the general behavior of this function and for the interpretation of test.

When called with a single Gam object, a special pair of anova tables for Gam models is returned. This gives a breakdown of the degrees of freedom for all the terms in the model, separating the projection part and nonparametric part of each, and returned as a list of two anova objects. For example, a term specified by s() is broken down into a single degree of freedom for its linear component, and the remainder for the nonparametric component. In addition, a type of score test is performed for each of the nonparametric terms. The nonparametric component is set to zero, and the linear part is updated, holding the other nonparametric terms fixed. This is done efficiently and simulataneously for all terms.

Author(s)

Written by Trevor Hastie, following closely the design in the "Generalized Additive Models" chapter (Hastie, 1992) in Chambers and Hastie (1992).

References

Hastie, T. J. (1992) *Generalized additive models.* Chapter 7 of *Statistical Models in S* eds J. M. Chambers and T. J. Hastie, Wadsworth & Brooks/Cole.

Hastie, T. and Tibshirani, R. (1990) *Generalized Additive Models.* London: Chapman and Hall.

Venables, W. N. and Ripley, B. D. (2002) *Modern Applied Statistics with S.* New York: Springer.

Examples

```
data(gam.data)
Gam.object <- gam(y~s(x,6)+z,data=gam.data)
anova(Gam.object)
Gam.object2 <- update(Gam.object, ~.-z)
anova(Gam.object, Gam.object2, test="Chisq")
```

gam *Fitting Generalized Additive Models*

Description

gam is used to fit generalized additive models, specified by giving a symbolic description of the additive predictor and a description of the error distribution. gam uses the *backfitting algorithm* to combine different smoothing or fitting methods. The methods currently supported are local regression and smoothing splines.

Usage

```
gam(
  formula,
  family = gaussian,
  data,
  weights,
  subset,
  na.action,
  start = NULL,
  etastart,
  mustart,
  control = gam.control(...),
  model = TRUE,
  method = "glm.fit",
  x = FALSE,
  y = TRUE,
  ...
)

gam.fit(
  x,
  y,
  smooth.frame,
  weights = rep(1, nobs),
  start = NULL,
  etastart = NULL,
  mustart = NULL,
  offset = rep(0, nobs),
  family = gaussian(),
  control = gam.control()
)
```

Arguments

formula	a formula expression as for other regression models, of the form response ~ predictors. See the documentation of lm and formula for details. Built-in

	nonparametric smoothing terms are indicated by `s` for smoothing splines or `lo` for `loess` smooth terms. See the documentation for `s` and `lo` for their arguments. Additional smoothers can be added by creating the appropriate interface functions. Interactions with nonparametric smooth terms are not fully supported, but will not produce errors; they will simply produce the usual parametric interaction.
`family`	a description of the error distribution and link function to be used in the model. This can be a character string naming a family function, a family function or the result of a call to a family function. (See `family` for details of family functions.)
`data`	an optional data frame containing the variables in the model. If not found in `data`, the variables are taken from `environment(formula)`, typically the environment from which `gam` is called.
`weights`	an optional vector of weights to be used in the fitting process.
`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`. A special method `na.gam.replace` allows for mean-imputation of missing values (assumes missing at random), and works gracefully with `gam`
`start`	starting values for the parameters in the additive predictor.
`etastart`	starting values for the additive predictor.
`mustart`	starting values for the vector of means.
`control`	a list of parameters for controlling the fitting process. See the documentation for `gam.control` for details. These can also be set as arguments to `gam()` itself.
`model`	a logical value indicating whether *model frame* should be included as a component of the returned value. Needed if `gam` is called and predicted from inside a user function. Default is `TRUE`.
`method`	the method to be used in fitting the parametric part of the model. The default method `"glm.fit"` uses iteratively reweighted least squares (IWLS). The only current alternative is `"model.frame"` which returns the model frame and does no fitting.
`x, y`	For `gam`: logical values indicating whether the response vector and model matrix used in the fitting process should be returned as components of the returned value. For `gam.fit`: `x` is a model matrix of dimension `n * p`, and `y` is a vector of observations of length `n`.
`...`	further arguments passed to or from other methods.
`smooth.frame`	for `gam.fit` only. This is essentially a subset of the model frame corresponding to the smooth terms, and has the ingredients needed for smoothing each variable in the backfitting algorithm. The elements of this frame are produced by the formula functions `lo` and `s`.
`offset`	this can be used to specify an *a priori* known component to be included in the additive predictor during fitting.

Details

The gam model is fit using the local scoring algorithm, which iteratively fits weighted additive models by backfitting. The backfitting algorithm is a Gauss-Seidel method for fitting additive models, by iteratively smoothing partial residuals. The algorithm separates the parametric from the nonparametric part of the fit, and fits the parametric part using weighted linear least squares within the backfitting algorithm. This version of gam remains faithful to the philosophy of GAM models as outlined in the references below.

An object `gam.slist` (currently set to `c("lo","s","random")`) lists the smoothers supported by gam. Corresponding to each of these is a smoothing function `gam.lo`, `gam.s` etc that take particular arguments and produce particular output, custom built to serve as building blocks in the backfitting algorithm. This allows users to add their own smoothing methods. See the documentation for these methods for further information. In addition, the object `gam.wlist` (currently set to `c("s","lo")`) lists the smoothers for which efficient backfitters are provided. These are invoked if all the smoothing methods are of one kind (either all `"lo"` or all `"s"`).

Value

gam returns an object of class `Gam`, which inherits from both `glm` and `lm`.

Gam objects can be examined by `print`, `summary`, `plot`, and `anova`. Components can be extracted using extractor functions `predict`, `fitted`, `residuals`, `deviance`, `formula`, and `family`. Can be modified using `update`. It has all the components of a `glm` object, with a few more. This also means it can be queried, summarized etc by methods for `glm` and `lm` objects. Other generic functions that have methods for `Gam` objects are `step` and `preplot`.

The following components must be included in a legitimate 'Gam' object. The residuals, fitted values, coefficients and effects should be extracted by the generic functions of the same name, rather than by the `"$"` operator. The `family` function returns the entire family object used in the fitting, and `deviance` can be used to extract the deviance of the fit.

`coefficients` the coefficients of the parametric part of the `additive.predictors`, which multiply the columns of the model matrix. The names of the coefficients are the names of the single-degree-of-freedom effects (the columns of the model matrix). If the model is overdetermined there will be missing values in the coefficients corresponding to inestimable coefficients.

`additive.predictors`
the additive fit, given by the product of the model matrix and the coefficients, plus the columns of the `$smooth` component.

`fitted.values` the fitted mean values, obtained by transforming the component `additive.predictors` using the inverse link function.

`smooth`, `nl.df`, `nl.chisq`, `var`
these four characterize the nonparametric aspect of the fit. `smooth` is a matrix of smooth terms, with a column corresponding to each smooth term in the model; if no smooth terms are in the `Gam` model, all these components will be missing. Each column corresponds to the strictly nonparametric part of the term, while the parametric part is obtained from the model matrix. `nl.df` is a vector giving the approximate degrees of freedom for each column of `smooth`. For smoothing splines specified by `s(x)`, the approximate `df` will be the trace of the implicit smoother matrix minus 2. `nl.chisq` is a vector containing a type of score test

	for the removal of each of the columns of `smooth`. `var` is a matrix like `smooth`, containing the approximate pointwise variances for the columns of `smooth`.
`smooth.frame`	This is essentially a subset of the model frame corresponding to the smooth terms, and has the ingredients needed for making predictions from a `Gam` object
`residuals`	the residuals from the final weighted additive fit; also known as residuals, these are typically not interpretable without rescaling by the weights.
`deviance`	up to a constant, minus twice the maximized log-likelihood. Similar to the residual sum of squares. Where sensible, the constant is chosen so that a saturated model has deviance zero.
`null.deviance`	The deviance for the null model, comparable with `deviance`. The null model will include the offset, and an intercept if there is one in the model
`iter`	the number of local scoring iterations used to compute the estimates.
`bf.iter`	a vector of length `iter` giving number of backfitting iterations used at each inner loop.
`family`	a three-element character vector giving the name of the family, the link, and the variance function; mainly for printing purposes.
`weights`	the *working* weights, that is the weights in the final iteration of the local scoring fit.
`prior.weights`	the case weights initially supplied.
`df.residual`	the residual degrees of freedom.
`df.null`	the residual degrees of freedom for the null model.

The object will also have the components of a `lm` object: `coefficients`, `residuals`, `fitted.values`, `call`, `terms`, and some others involving the numerical fit. See `lm.object`.

Author(s)

Written by Trevor Hastie, following closely the design in the "Generalized Additive Models" chapter (Hastie, 1992) in Chambers and Hastie (1992), and the philosophy in Hastie and Tibshirani (1991). This version of `gam` is adapted from the S version to match the `glm` and `lm` functions in R.

Note that this version of `gam` is different from the function with the same name in the R library `mgcv`, which uses only smoothing splines with a focus on automatic smoothing parameter selection via GCV. To avoid issues with S3 method handling when both packages are loaded, the object class in package "gam" is now "Gam".

References

Hastie, T. J. (1991) *Generalized additive models.* Chapter 7 of *Statistical Models in S* eds J. M. Chambers and T. J. Hastie, Wadsworth & Brooks/Cole.

Hastie, T. and Tibshirani, R. (1990) *Generalized Additive Models.* London: Chapman and Hall.

Venables, W. N. and Ripley, B. D. (2002) *Modern Applied Statistics with S.* New York: Springer.

See Also

`glm`, `family`, `lm`.

Examples

```
data(kyphosis)
gam(Kyphosis ~ s(Age,4) + Number, family = binomial, data=kyphosis,
trace=TRUE)
data(airquality)
gam(Ozone^(1/3) ~ lo(Solar.R) + lo(Wind, Temp), data=airquality, na=na.gam.replace)
gam(Kyphosis ~ poly(Age,2) + s(Start), data=kyphosis, family=binomial, subset=Number>2)
data(gam.data)
Gam.object <- gam(y ~ s(x,6) + z,data=gam.data)
summary(Gam.object)
plot(Gam.object,se=TRUE)
data(gam.newdata)
predict(Gam.object,type="terms",newdata=gam.newdata)
```

---

gam.control *Auxilliary for controlling GAM fitting*

Description

Auxiliary function as user interface for 'gam' fitting. Typically only used when calling 'gam' or 'gam.fit'.

Usage

```
gam.control(
  epsilon = 1e-07,
  bf.epsilon = 1e-07,
  maxit = 30,
  bf.maxit = 30,
  trace = FALSE,
  ...
)
```

Arguments

epsilon	convergence threshold for local scoring iterations
bf.epsilon	convergence threshold for backfitting iterations
maxit	maximum number of local scoring iterations
bf.maxit	maximum number of backfitting iterations
trace	should iteration details be printed while gam is fitting the model.
...	placemark for additional arguments

Value

a list is returned, consisting of the five parameters, conveniently packaged up to supply the control argument to gam. The values for gam.control can be supplied directly in a call to gam; these are then filtered through gam.control inside gam.

References

Hastie, T. J. (1992) *Generalized additive models.* Chapter 7 of *Statistical Models in S* eds J. M. Chambers and T. J. Hastie, Wadsworth & Brooks/Cole.

Examples

```
## Not run: gam(formula, family, control = gam.control(bf.maxit=15))
## Not run: gam(formula, family, bf.maxit = 15) # these are equivalent
```

---

gam.data *Simulated dataset for gam*

Description

A simple simulated dataset, used to test out the gam functions

Format

A data frame with 100 observations on the following 6 variables:

x a numeric vector - predictor

y a numeric vector - the response

z a numeric vector - noise predictor

f a numeric vector - true function

probf a numeric vector - probability function

ybin a numeric vector - binary response

Details

This dataset is artificial, and is used to test out some of the features of gam.

Examples

```
data(gam.data)
gam(y ~ s(x) + z, data=gam.data)
```

`gam.exact` *A method for gam producing asymptotically exact standard errors for linear estimates*

Description

This function is a "wrapper" for a Gam object, and produces exact standard errors for each linear term in the gam call (except for the intercept).

Usage

```
gam.exact(Gam.obj)
```

Arguments

`Gam.obj`	a Gam object

Details

Only standard errors for the linear terms are produced. There is a print method for the Gamex class.

Value

A list (of class Gamex) containing a table of coefficients and a variance covariance matrix for the linear terms in the formula of the gam call.

Author(s)

Aidan McDermott, Department of Biostatistics, Johns Hopkins University. Modified by Trevor Hastie for R

References

Issues in Semiparametric Regression: A Case Study of Time Series Models in Air Pollution and Mortality, Dominici F., McDermott A., Hastie T.J., *JASA*, December 2004, 99(468), 938-948. See https://hastie.su.domains/Papers/dominiciR2.pdf

Examples

```
set.seed(31)
n     <- 200
x     <- rnorm(n)
y     <- rnorm(n)
a     <- rep(1:10,length=n)
b     <- rnorm(n)
z    <- 1.4 + 2.1*a + 1.2*b + 0.2*sin(x/(3*max(x))) + 0.3*cos(y/(5*max(y))) + 0.5 * rnorm(n)
dat   <- data.frame(x,y,a,b,z,testit=b*2)
### Model 1: Basic
Gam.o <- gam(z ~ a + b + s(x,3) + s(y,5), data=dat)
```

```
coefficients(summary.glm(Gam.o))
gam.exact(Gam.o)
### Model 2: Poisson
Gam.o <- gam(round(abs(z)) ~ a + b + s(x,3) + s(y,5), data=dat,family=poisson)
coefficients(summary.glm(Gam.o))
gam.exact(Gam.o)
```

`gam.lo` *Specify a loess fit in a GAM formula*

Description

A symbolic wrapper to indicate a smooth term in a formala argument to gam

Usage

```
gam.lo(
  x,
  y,
  w = rep(1, length(y)),
  span = 0.5,
  degree = 1,
  ncols = p,
  xeval = x
)

lo(..., span = 0.5, degree = 1)
```

Arguments

`x`	for `gam.lo`, the appropriate basis of polynomials generated from the arguments to `lo`. These are also the variables that receive linear coefficients in the GAM fit.
`y`	a response variable passed to `gam.lo` during backfitting
`w`	weights
`span`	the number of observations in a neighborhood. This is the smoothing parameter for a `loess` fit. If specified, the full argument name `span` must be written.
`degree`	the degree of local polynomial to be fit; currently restricted to be `1` or `2`. If specified, the full argument name `degree` must be written.
`ncols`	for `gam.lo` the number of columns in `x` used as the smoothing inputs to local regression. For example, if `degree=2`, then `x` has two columns defining a degree-2 polynomial basis. Both are needed for the parameteric part of the fit, but `ncol=1` telling the local regression routine that the first column is the actually smoothing variable.
`xeval`	If this argument is present, then `gam.lo` produces a prediction at `xeval`.

`...` the unspecified `...{}` can be a comma-separated list of numeric vectors, numeric matrix, or expressions that evaluate to either of these. If it is a list of vectors, they must all have the same length.

Details

A smoother in gam separates out the parametric part of the fit from the non-parametric part. For local regression, the parametric part of the fit is specified by the particular polynomial being fit locally. The workhorse function `gam.lo` fits the local polynomial, then strips off this parametric part. All the parametric pieces from all the terms in the additive model are fit simultaneously in one operation for each loop of the backfitting algorithm.

Value

`lo` returns a numeric matrix. The simplest case is when there is a single argument to `lo` and `degree=1`; a one-column matrix is returned, consisting of a normalized version of the vector. If `degree=2` in this case, a two-column matrix is returned, consisting of a degree-2 polynomial basis. Similarly, if there are two arguments, or the single argument is a two-column matrix, either a two-column matrix is returned if `degree=1`, or a five-column matrix consisting of powers and products up to degree 2. Any dimensional argument is allowed, but typically one or two vectors are used in practice.

The matrix is endowed with a number of attributes; the matrix itself is used in the construction of the model matrix, while the attributes are needed for the backfitting algorithms `general.wam` (weighted additive model) or `lo.wam` (currently not implemented). Local-linear curve or surface fits reproduce linear responses, while local-quadratic fits reproduce quadratic curves or surfaces. These parts of the `loess` fit are computed exactly together with the other parametric linear parts

When two or more smoothing variables are given, the user should make sure they are in a commensurable scale; `lo()` does no normalization. This can make a difference, since `lo()` uses a spherical (isotropic) neighborhood when establishing the nearest neighbors.

Note that `lo` itself does no smoothing; it simply sets things up for `gam`; `gam.lo` does the actual smoothing. of the model.

One important attribute is named `call`. For example, `lo(x)` has a call component `gam.lo(data[["lo(x)"]], z, w, span = 0.5, degree = 1,ncols = 1)`. This is an expression that gets evaluated repeatedly in `general.wam` (the backfitting algorithm).

`gam.lo` returns an object with components

`residuals`	The residuals from the smooth fit. Note that the smoother removes the parametric part of the fit (using a linear fit with the columns in `x`), so these residual represent the nonlinear part of the fit.
`nl.df`	the nonlinear degrees of freedom
`var`	the pointwise variance for the nonlinear fit

When `gam.lo` is evaluated with an `xeval` argument, it returns a matrix of predictions.

Author(s)

Written by Trevor Hastie, following closely the design in the "Generalized Additive Models" chapter (Hastie, 1992) in Chambers and Hastie (1992).

References

Hastie, T. J. (1992) *Generalized additive models.* Chapter 7 of *Statistical Models in S* eds J. M. Chambers and T. J. Hastie, Wadsworth & Brooks/Cole.

Hastie, T. and Tibshirani, R. (1990) *Generalized Additive Models.* London: Chapman and Hall.

See Also

s, bs, ns, poly, loess

Examples

```
y ~ Age + lo(Start)
     # fit Start using a loess smooth with a (default) span of 0.5.
y ~ lo(Age) + lo(Start, Number)
y ~ lo(Age, span=0.3) # the argument name span cannot be abbreviated.
```

---

gam.random *Specify a Random Effects Fit in a GAM Formula*

Description

A symbolic wrapper for a factor term, to specify a random effect term in a formula argument to gam

Usage

```
gam.random(f, y, w, df = sum(non.zero), lambda = 0, intercept = TRUE, xeval)

random(f, df = NULL, lambda = 0, intercept = TRUE)
```

Arguments

f	factor variable, or expression that evaluates to a factor.
y	a response variable passed to gam.random during backfitting
w	weights
df	the target equivalent degrees of freedom, used as a smoothing parameter. The real smoothing parameter (lambda below) is found such that df=tr(S), where S is the implicit smoother matrix. Values for df should be greater than 0 and less than the number of levels of f. If both df and lambda are supplied, the latter takes precedence. Note that df is not necessarily an integer.
lambda	the non-negative penalty parameter. This is interpreted as a variance ratio in a mixed effects model - namely the ratio of the noise variance to the random-effect variance.
intercept	if intercept=TRUE (the default) then the estimated level effects are centered to average zero, otherwise they are left alone.
xeval	If this argument is present, then gam.random produces a prediction at xeval.

Details

This "smoother" takes a factor as input and returns a shrunken-mean fit. If `lambda=0`, it simply computes the mean of the response at each level of `f`. With `lambda>0`, it returns a shrunken mean, where the j'th level is shrunk by `nj/(nj+lambda)`, with `nj` being the number of observations (or sum of their weights) at level `j`. Using such smoother(s) in gam is formally equivalent to fitting a mixed-effect model by generalized least squares.

Value

`random` returns the vector `f`, endowed with a number of attributes. The vector itself is used in computing the means in backfitting, while the attributes are needed for the backfitting algorithms `general.wam`. Note that `random` itself does no smoothing; it simply sets things up for `gam`.

One important attribute is named `call`. For example, `random(f,lambda=2)` has a call component `gam.random(data[["random(f, lambda = 2)"]], z, w, df = NULL, lambda = 2, intercept = TRUE)`. This is an expression that gets evaluated repeatedly in `general.wam` (the backfitting algorithm).

`gam.random` returns an object with components

`residuals`	The residuals from the smooth fit.
`nl.df`	the degrees of freedom
`var`	the pointwise variance for the fit
`lambda`	the value of `lambda` used in the fit

When `gam.random` is evaluated with an `xeval` argument, it returns a vector of predictions.

Author(s)

Written by Trevor Hastie, following closely the design in the "Generalized Additive Models" chapter (Hastie, 1992) in Chambers and Hastie (1992).

References

Hastie, T. J. (1992) *Generalized additive models.* Chapter 7 of *Statistical Models in S* eds J. M. Chambers and T. J. Hastie, Wadsworth & Brooks/Cole.

Hastie, T. and Tibshirani, R. (1990) *Generalized Additive Models.* London: Chapman and Hall.

Cantoni, E. and hastie, T. (2002) Degrees-of-freedom tests for smoothing splines, *Biometrika* 89(2), 251-263

See Also

`lo`, `s`, `bs`, `ns`, `poly`

Examples

```
# fit a model with a linear term in Age and a random effect in the factor Level
y ~ Age + random(Level, lambda=1)
```

gam.s *Specify a Smoothing Spline Fit in a GAM Formula*

Description

A symbolic wrapper to indicate a smooth term in a formala argument to gam

Usage

```
gam.s(x, y, w = rep(1, length(x)), df = 4, spar = 1, xeval)

s(x, df = 4, spar = 1)
```

Arguments

x	the univariate predictor, or expression, that evaluates to a numeric vector.
y	a response variable passed to gam.s during backfitting
w	weights
df	the target equivalent degrees of freedom, used as a smoothing parameter. The real smoothing parameter (spar below) is found such that df=tr(S)-1, where S is the implicit smoother matrix. Values for df should be greater than 1, with df=1 implying a linear fit. If both df and spar are supplied, the former takes precedence. Note that df is not necessarily an integer.
spar	can be used as smoothing parameter, with values typically in (0,1]. See smooth.spline for more details.
xeval	If this argument is present, then gam.s produces a prediction at xeval.

Value

s returns the vector x, endowed with a number of attributes. The vector itself is used in the construction of the model matrix, while the attributes are needed for the backfitting algorithms general.wam (weighted additive model) or s.wam. Since smoothing splines reproduces linear fits, the linear part will be efficiently computed with the other parametric linear parts of the model.

Note that s itself does no smoothing; it simply sets things up for gam.

One important attribute is named call. For example, s(x) has a call component gam.s(data[["s(x)"]], z, w, spar = 1, df = 4). This is an expression that gets evaluated repeatedly in general.wam (the backfitting algorithm).

gam.s returns an object with components

residuals	The residuals from the smooth fit. Note that the smoother removes the parametric part of the fit (using a linear fit in x), so these residual represent the nonlinear part of the fit.
nl.df	the nonlinear degrees of freedom
var	the pointwise variance for the nonlinear fit

When gam.s is evaluated with an xeval argument, it returns a vector of predictions.

Author(s)

Written by Trevor Hastie, following closely the design in the "Generalized Additive Models" chapter (Hastie, 1992) in Chambers and Hastie (1992).

References

Hastie, T. J. (1992) *Generalized additive models.* Chapter 7 of *Statistical Models in S* eds J. M. Chambers and T. J. Hastie, Wadsworth & Brooks/Cole.

Hastie, T. and Tibshirani, R. (1990) *Generalized Additive Models.* London: Chapman and Hall.

Cantoni, E. and hastie, T. (2002) Degrees-of-freedom tests for smoothing splines, *Biometrika* 89(2), 251-263

See Also

lo, smooth.spline, bs, ns, poly

Examples

```
# fit Start using a smoothing spline with 4 df.
y ~ Age + s(Start, 4)
# fit log(Start) using a smoothing spline with 5 df.
y ~ Age + s(log(Start), df=5)
```

---

gam.scope *Generate a scope for step.Gam*

Description

Given a data.frame as an argument, generate a scope list for use in step.Gam, each element of which gives the candidates for that term.

Usage

```
gam.scope(frame, response = 1, smoother = "s", arg = NULL, form = TRUE)
```

Arguments

frame	a data.frame to be used in step.Gam. Apart from the response column, all other columns will be used.
response	The column in frame used as the response. Default is 1.
smoother	which smoother to use for the nonlinear terms; i.e. "s" or "lo", or any other supplied smoother. Default is "s".
arg	a character (vector), which is the argument to smoother. For example, arg="df=6" would result in the expression s(x,df=6) for a column named "x". This can be a vector, for example arg=c("df=4","df=6"), which would result two smooth terms.
form	if TRUE, each term is a formula, else a character vector.

Details

This function creates a similar scope formula for each variable in the frame. A column named "x" by default will generate a scope term `~1+x+s(x)`. With `arg=c("df=4","df=6")` we get `~1+x+s(x,df=4)+s(x,df=6)`. With form=FALSE, we would get the character vector `c("1","x","s(x,df=4)","s(x,df=6")`.

Value

a scope list is returned, with either a formula or a character vector for each term, which describes the candidates for that term in the Gam.

Author(s)

Written by Trevor Hastie, following closely the design in the "Generalized Additive Models" chapter (Hastie, 1992) in Chambers and Hastie (1992). This version of `gam.scope` is adapted from the S version.

References

Hastie, T. J. (1991) *Generalized additive models.* Chapter 7 of *Statistical Models in S* eds J. M. Chambers and T. J. Hastie, Wadsworth & Brooks/Cole.

See Also

`step.Gam`

Examples

```
data(gam.data)
gdata=gam.data[,1:3]
gam.scope(gdata,2)
gam.scope(gdata,2,arg="df=5")
gam.scope(gdata,2,arg="df=5",form=FALSE)
gam.scope(gdata,2,arg=c("df=4","df=6"))
```

---

gam.smoothers *Smoothers available for backfitting*

Description

Auxiliary function as user interface for 'gam' fitting. Lists what smoothers are implemented, and allows users to include new smoothers.

Usage

```
gam.smoothers(slist = c("s", "lo", "random"), wlist = c("s", "lo"))
```

Arguments

slist	character vector giving names of smoothers available for general backfitting. For every entry, eg "lo", there must exist a formula function "lo()" that prepares the data, and a fitting function with the name "gam.lo" which actually does the fitting. Look at "lo" and "s" as examples.
wlist	character vector (subset of slist) giving names of smoothers for which a special backfitting algorithm is available, when only that smoother appears (multiple times) in the formula, along with other non smooth terms.

Value

a list is returned, consisting of the two named vectors. If the function is called with no arguments, it gets the version of "gam.smooth.list"' in the search path, by default from the package name space. Once it is called with either of the arguments, it places a local copy in the users namespace.

References

Hastie, T. J. (1992) *Generalized additive models.* Chapter 7 of *Statistical Models in S* eds J. M. Chambers and T. J. Hastie, Wadsworth & Brooks/Cole.

Examples

```
## Not run: gam.smoothers()$slist # get the gam.smooth.list, and extract component slist
## Not run: gam.smoothers(slist=c("s","lo","random","tps") # add a new smoother "tps" to the list
```

---

kyphosis *A classic example dataset for GAMs*

Description

Data on the results of a spinal operation "laminectomy" on children, to correct for a condition called "kyphosis"; see Hastie and Tibshirani (1990) for details

Usage

```
data(kyphosis)
```

Format

A data frame with 81 observations on the following 4 variables.

Kyphosis a response factor with levels `absent` `present`.

Age of child in months, a numeric vector

Number of vertebra involved in the operation,a numeric vector

Start level of the operation, a numeric vector

Source

Hastie, T. and Tibshirani, R. (1990) *Generalized Additive Models.* London: Chapman and Hall.

`na.gam.replace` *Missing Data Filter for GAMs*

Description

A method for dealing with missing values, friendly to GAM models.

Usage

```
na.gam.replace(frame)
```

Arguments

`frame`	a model or data frame

Value

a model or data frame is returned, with the missing observations (NAs) replaced. The following rules are used. A factor with missing data is replaced by a new factor with one more level, labelled `"NA"`, which records the missing data. Ordered factors are treated similarly, except the result is an unordered factor. A missing numeric vector has its missing entires replaced by the mean of the non-missing entries. Similarly, a matrix with missing entries has each missing entry replace by the mean of its column. If `frame` is a model frame, the response variable can be identified, as can the weights (if present). Any rows for which the response or weight is missing are removed entirely from the model frame.

The word `"gam"` in the name is relevant, because `gam()` makes special use of this filter. All columns of a model frame that were created by a call to `lo()` or `s()` have an attribute names `"NAs"` if NAs are present in their columns. Despite the replacement by means, these attributes remain on the object, and `gam()` takes appropriate action when smoothing against these columns. See section 7.3.2 in Hastie (1992) for more details.

Author(s)

Trevor Hastie

References

Hastie, T. J. (1992) *Generalized additive models.* Chapter 7 of *Statistical Models in S* eds J. M. Chambers and T. J. Hastie, Wadsworth & Brooks/Cole.

See Also

`na.fail`, `na.omit`, `gam`

Examples

```
data(airquality)
gam(Ozone^(1/3) ~ lo(Solar.R) + lo(Wind, Temp), data=airquality, na=na.gam.replace)
```

`plot.Gam` *Plot Components of a GAM Object*

Description

A plot method for GAM objects, which can be used on GLM and LM objects as well. It focuses on terms (main-effects), and produces a suitable plot for terms of different types

Usage

```
## S3 method for class 'Gam'
plot(
  x,
  residuals = NULL,
  rugplot = TRUE,
  se = FALSE,
  scale = 0,
  ask = FALSE,
  terms = labels.Gam(x),
  ...
)

## S3 method for class 'Gam'
preplot(object, newdata, terms = labels.Gam(object), ...)
```

Arguments

`x`	a `Gam` object, or a `preplot.Gam` object. The first thing `plot.Gam()` does is check if `x` has a component called `preplot`; if not, it computes one using `preplot.Gam()`. Either way, it is this `preplot.Gam` object that is required for plotting a `Gam` object.
`residuals`	if `TRUE`, partial deviance residuals are plotted along with the fitted terms—default is `FALSE`. If `residuals` is a vector with the same length as each fitted term in `x`, then these are taken to be the overall residuals to be used for constructing the partial residuals.
`rugplot`	if `TRUE` (the default), a univariate histogram or `rugplot` is displayed along the base of each plot, showing the occurrence of each `x`; ties are broken by jittering.
`se`	if `TRUE`, upper and lower pointwise twice-standard-error curves are included for each plot. The default is `FALSE`.

scale	a lower limit for the number of units covered by the limits on the y for each plot. The default is scale=0, in which case each plot uses the range of the functions being plotted to create their ylim. By setting scale to be the maximum value of diff(ylim) for all the plots, then all subsequent plots will produced in the same vertical units. This is essential for comparing the importance of fitted terms in additive models.
ask	if TRUE, plot.Gam() operates in interactive mode.
terms	subsets of the terms can be selected
...	Additonal plotting arguments, not all of which will work (like xlim)
object	same as x
newdata	if supplied to preplot.Gam, the preplot object is based on them rather than the original.

Value

a plot is produced for each of the terms in the object x. The function currently knows how to plot all main-effect functions of one or two predictors. So in particular, interactions are not plotted. An appropriate x-y is produced to display each of the terms, adorned with residuals, standard-error curves, and a rugplot, depending on the choice of options. The form of the plot is different, depending on whether the x-value for each plot is numeric, a factor, or a matrix.

When ask=TRUE, rather than produce each plot sequentially, plot.Gam() displays a menu listing all the terms that can be plotted, as well as switches for all the options.

A preplot.Gam object is a list of precomputed terms. Each such term (also a preplot.Gam object) is a list with components x, y and others—the basic ingredients needed for each term plot. These are in turn handed to the specialized plotting function gplot(), which has methods for different classes of the leading x argument. In particular, a different plot is produced if x is numeric, a category or factor, a matrix, or a list. Experienced users can extend this range by creating more gplot() methods for other classes. Graphical parameters (see par) may also be supplied as arguments to this function. This function is a method for the generic function plot() for class "Gam".

It can be invoked by calling plot(x) for an object x of the appropriate class, or directly by calling plot.Gam(x) regardless of the class of the object.

Author(s)

Written by Trevor Hastie, following closely the design in the "Generalized Additive Models" chapter (Hastie, 1992) in Chambers and Hastie (1992).

References

Hastie, T. J. (1992) *Generalized additive models.* Chapter 7 of *Statistical Models in S* eds J. M. Chambers and T. J. Hastie, Wadsworth & Brooks/Cole.

Hastie, T. and Tibshirani, R. (1990) *Generalized Additive Models.* London: Chapman and Hall.

See Also

preplot, predict.Gam

Examples

```
data(gam.data)
Gam.object <- gam(y ~ s(x,6) + z,data=gam.data)
plot(Gam.object,se=TRUE)
data(gam.newdata)
preplot(Gam.object,newdata=gam.newdata)
```

`predict.Gam` *Predict method for GAM fits*

Description

Obtains predictions and optionally estimates standard errors of those predictions from a fitted generalized additive model object.

Usage

```
## S3 method for class 'Gam'
predict(
  object,
  newdata,
  type = c("link", "response", "terms"),
  dispersion = NULL,
  se.fit = FALSE,
  na.action = na.pass,
  terms = labels(object),
  ...
)
```

Arguments

`object`	a fitted `Gam` object, or one of its inheritants, such as a `glm` or `lm` object.
`newdata`	a data frame containing the values at which predictions are required. This argument can be missing, in which case predictions are made at the same values used to compute the object. Only those predictors, referred to in the right side of the formula in object need be present by name in `newdata`.
`type`	type of predictions, with choices `"link"` (the default), `"response"`, or `"terms"`. The default produces predictions on the scale of the additive predictors, and with `newdata` missing, `predict` is simply an extractor function for this component of a `Gam` object. If `"response"` is selected, the predictions are on the scale of the response, and are monotone transformations of the additive predictors, using the inverse link function. If `type="terms"` is selected, a matrix of predictions is produced, one column for each term in the model.
`dispersion`	the dispersion of the GLM fit to be assumed in computing the standard errors. If omitted, that returned by 'summary' applied to the object is used

se.fit	if TRUE, pointwise standard errors are computed along with the predictions.
na.action	function determining what should be done with missing values in 'newdata'. The default is to predict 'NA'.
terms	if type="terms", the terms= argument can be used to specify which terms should be included; the default is labels(object).
...	Placemark for additional arguments to predict

Value

a vector or matrix of predictions, or a list consisting of the predictions and their standard errors if se.fit = TRUE. If type="terms", a matrix of fitted terms is produced, with one column for each term in the model (or subset of these if the terms= argument is used). There is no column for the intercept, if present in the model, and each of the terms is centered so that their average over the original data is zero. The matrix of fitted terms has a "constant" attribute which, when added to the sum of these centered terms, gives the additive predictor. See the documentation of predict for more details on the components returned.

When newdata are supplied, predict.Gam simply invokes inheritance and gets predict.glm to produce the parametric part of the predictions. For each nonparametric term, predict.Gam reconstructs the partial residuals and weights from the final iteration of the local scoring algorithm. The appropriate smoother is called for each term, with the appropriate xeval argument (see s or lo), and the prediction for that term is produced.

The standard errors are based on an approximation given in Hastie (1992). Currently predict.Gam does not produce standard errors for predictions at newdata.

Warning: naive use of the generic predict can produce incorrect predictions when the newdata argument is used, if the formula in object involves transformations such as sqrt(Age - min(Age)).

Author(s)

Written by Trevor Hastie, following closely the design in the "Generalized Additive Models" chapter (Hastie, 1992) in Chambers and Hastie (1992). This version of predict.Gam is adapted from the S version to match the corresponding predict methods for glm and lm objects in R. The safe.predict.Gam function in S is no longer required, primarily because a safe prediction method is in place for functions like ns, bs, and poly.

References

Hastie, T. J. (1992) *Generalized additive models.* Chapter 7 of *Statistical Models in S* eds J. M. Chambers and T. J. Hastie, Wadsworth & Brooks/Cole.

Hastie, T. and Tibshirani, R. (1990) *Generalized Additive Models.* London: Chapman and Hall.

Venables, W. N. and Ripley, B. D. (2002) *Modern Applied Statistics with S.* New York: Springer.

See Also

predict.glm, fitted, expand.grid

Examples

```
data(gam.data)
Gam.object <- gam(y ~ s(x,6) + z, data=gam.data)
predict(Gam.object) # extract the additive predictors
data(gam.newdata)
predict(Gam.object, gam.newdata, type="terms")
```

`step.Gam` *Stepwise model builder for GAM*

Description

Builds a GAM model in a step-wise fashion. For each "term" there is an ordered list of alternatives, and the function traverses these in a greedy fashion. Note: this is NOT a method for `step`, which used to be a generic, so must be invoked with the full name.

Usage

```
step.Gam(
  object,
  scope,
  scale,
  direction = c("both", "backward", "forward"),
  trace = TRUE,
  keep = NULL,
  steps = 1000,
  parallel = FALSE,
  ...
)
```

Arguments

`object`	An object of class `Gam` or any of it's inheritants.
`scope`	defines the range of models examined in the step-wise search. It is a list of formulas, with each formula corresponding to a term in the model. Each of these formulas specifies a "regimen" of candidate forms in which the particular term may enter the model. For example, a term formula might be `~1+ Income + log(Income) + s(Income)`. This means that `Income` could either appear not at all, linearly, linearly in its logarithm, or as a smooth function estimated non-parametrically. A `1` in the formula allows the additional option of leaving the term out of the model entirely. Every term in the model is described by such a term formula, and the final model is built up by selecting a component from each formula. As an alternative more convenient for big models, each list can have instead of a formula a character vector corresponding to the candidates for that term. Thus we could have `c("1","x","s(x,df=5")` rather than `~1+x+s(x,df=5)`.

The supplied model `object` is used as the starting model, and hence there is the requirement that one term from each of the term formulas be present in `formula(object)`. This also implies that any terms in `formula(object)` *not* contained in any of the term formulas will be forced to be present in every model considered. The function `gam.scope` is helpful for generating the scope argument for a large model.

`scale` an optional argument used in the definition of the AIC statistic used to evaluate models for selection. By default, the scaled Chi-squared statistic for the initial model is used, but if forward selection is to be performed, this is not necessarily a sound choice.

`direction` The mode of step-wise search, can be one of `"both"`, `"backward"`, or `"forward"`, with a default of `"both"`. If `scope` is missing, the default for `direction` is "both".

`trace` If `TRUE` (the default), information is printed during the running of `step.Gam()`. This is an encouraging choice in general, since `step.Gam()` can take some time to compute either for large models or when called with an an extensive `scope=` argument. A simple one line model summary is printed for each model selected. This argument can also be given as the binary `0` or `1`. A value `trace=2` gives a more verbose trace.

`keep` A filter function whose input is a fitted `Gam` object, and anything else passed via `...`, and whose output is arbitrary. Typically `keep()` will select a subset of the components of the object and return them. The default is not to keep anything.

`steps` The maximum number of steps to be considered. The default is 1000 (essentially as many as required). It is typically used to stop the process early.

`parallel` If `TRUE`, use parallel `foreach` to fit each trial run. Must register parallel before hand, such as `doMC` or others. See the example below.

`...` Additional arguments to be passed on to `keep`

Value

The step-wise-selected model is returned, with up to two additional components. There is an `"anova"` component corresponding to the steps taken in the search, as well as a `"keep"` component if the `keep=` argument was supplied in the call.

We describe the most general setup, when `direction = "both"`. At any stage there is a current model comprising a single term from each of the term formulas supplied in the `scope=` argument. A series of models is fitted, each corrresponding to a formula obtained by moving each of the terms one step up or down in its regimen, relative to the formula of the current model. If the current value for any term is at either of the extreme ends of its regimen, only one rather than two steps can be considered. So if there are `p` term formulas, at most `2*p - 1` models are considered. A record is kept of all the models ever visited (hence the `-1` above), to avoid repetition. Once each of these models has been fit, the "best" model in terms of the AIC statistic is selected and defines the step. The entire process is repeated until either the maximum number of steps has been used, or until the AIC criterion can not be decreased by any of the eligible steps.

Author(s)

Written by Trevor Hastie, following closely the design in the "Generalized Additive Models" chapter (Hastie, 1992) in Chambers and Hastie (1992).

References

Hastie, T. J. (1992) *Generalized additive models.* Chapter 7 of *Statistical Models in S* eds J. M. Chambers and T. J. Hastie, Wadsworth & Brooks/Cole.

Hastie, T. and Tibshirani, R. (1990) *Generalized Additive Models.* London: Chapman and Hall.

See Also

gam.scope,step,glm, gam, drop1, add1, anova.Gam

Examples

```
data(gam.data)
Gam.object <- gam(y~x+z, data=gam.data)
step.object <-step.Gam(Gam.object, scope=list("x"=~1+x+s(x,4)+s(x,6)+s(x,12),"z"=~1+z+s(z,4)))
## Not run:
# Parallel
require(doMC)
registerDoMC(cores=2)
step.Gam(Gam.object, scope=list("x"=~1+x+s(x,4)+s(x,6)+s(x,12),"z"=~1+z+s(z,4)),parallel=TRUE)

## End(Not run)
```

Index

∗ **datasets**
gam.data, 9
kyphosis, 18
∗ **effects**
gam.random, 13
∗ **mixed**
gam.random, 13
∗ **models**
anova.Gam, 2
gam, 4
gam-package, 2
gam.control, 8
gam.exact, 10
gam.lo, 11
gam.random, 13
gam.s, 15
gam.scope, 16
gam.smoothers, 17
na.gam.replace, 19
plot.Gam, 20
predict.Gam, 22
step.Gam, 24
∗ **nonparametric**
anova.Gam, 2
gam, 4
gam.control, 8
gam.exact, 10
gam.lo, 11
gam.random, 13
gam.s, 15
gam.scope, 16
gam.smoothers, 17
na.gam.replace, 19
plot.Gam, 20
predict.Gam, 22
step.Gam, 24
∗ **package**
gam-package, 2
∗ **random**
gam.random, 13
∗ **regression**
anova.Gam, 2
gam, 4
gam-package, 2
gam.control, 8
gam.exact, 10
gam.lo, 11
gam.random, 13
gam.s, 15
gam.scope, 16
gam.smoothers, 17
na.gam.replace, 19
plot.Gam, 20
predict.Gam, 22
step.Gam, 24
∗ **smooth**
anova.Gam, 2
gam, 4
gam.control, 8
gam.exact, 10
gam.lo, 11
gam.random, 13
gam.s, 15
gam.scope, 16
gam.smoothers, 17
na.gam.replace, 19
plot.Gam, 20
predict.Gam, 22
step.Gam, 24

add1, *26*
anova, *3*
anova.Gam, 2, *26*

bs, *13, 14*, *16*

drop1, *26*

expand.grid, *23*

family, *5*, *7*
fitted, *23*

gam, 4, *19*, *26*
gam-package, 2
gam.control, *5*, 8
gam.data, 9
gam.exact, 10
gam.lo, 11
gam.newdata *(*gam.data*)*, 9
gam.random, 13
gam.s, 15
gam.scope, 16, *26*
gam.smooth.list *(*gam.smoothers*)*, 17
gam.smoothers, 17
glm, *7*, *26*

kyphosis, 18

lm, *7*
lo, *14*, *16*, *23*
lo *(*gam.lo*)*, 11
loess, *13*

na.fail, *5*, *19*
na.gam.replace, *5*, 19
na.omit, *5*, *19*
ns, *13*, *14*, *16*

options, *5*

par, *21*
plot.Gam, 20
plot.preplot.Gam *(*plot.Gam*)*, 20
poly, *13*, *14*, *16*
predict.Gam, *21*, 22
predict.glm, *23*
preplot, *21*
preplot.Gam *(*plot.Gam*)*, 20

random *(*gam.random*)*, 13

s, *13*, *14*, *23*
s *(*gam.s*)*, 15
smooth.spline, *15*, *16*
step, *26*
step.Gam, *17*, 24
summary.Gam *(*anova.Gam*)*, 2