Package ‘cgaim’

October 12, 2022

Title Constrained Groupwise Additive Index Models
Version 1.0.0
Description Fits constrained groupwise additive index models and provides functions for inference and interpretation of these models. The method is described in Masselot, Chebana, Campana, Lavigne, Ouarda, Gosselin (2022) "Constrained groupwise additive index models" <doi:10.1093/biostatistics/kxac023>.
License GPL-3
Encoding UTF-8
Suggests testthat (>= 2.1.0)
RoxygenNote 7.2.0
Imports graphics, stats, scam, scar, quadprog, osqp, limSolve, Matrix, grDevices, methods, MASS, cgam, mgcv, gratia, doParallel, coneproj, TruncatedNormal, foreach
NeedsCompilation no
Author Pierre Masselot [aut, cre] (<https://orcid.org/0000-0002-7326-1290>)
Maintainer Pierre Masselot <pierre.masselot@lshtm.ac.uk>
Repository CRAN
Date/Publication 2022-07-15 15:50:02 UTC

R topics documented:

- boot.cgaim ........................................... 2
- build_constraints .................................. 3
- cgaim ................................................. 5
- cgaim.control ....................................... 8
- confint.cgaim ...................................... 10
- g ....................................................... 12
- plot.cgaim .......................................... 15
- predict.cgaim ...................................... 17
- print.cgaim ........................................ 19

Index 20
boot.cgaim

Bootstrap CGAIM

Description

Generates bootstrap replicates of a cgaim object.

Usage

boot.cgaim(object, boot.type = c("residuals", "wild", "pairs"),
bsamples = NULL, B = 100, l = 1, nc = 1)

Arguments

object
A cgaim object.

boot.type
The type of bootstrap to perform. Currently available type are "residuals", "wild" and "pairs". See details

bsamples
A numerical matrix of observation indices specifying bootstrap samples. Rows indicate observations and columns bootstrap samples. If NULL (the default), samples are generated internally.

B
Number of bootstrap samples to generate when bsamples = NULL.

l
Block length for block-bootstrap. Samples are generated by resampling block of observation of length l. The classical bootstrap corresponds to l = 1 (the default).

nc
Positive integer. If nc > 1, the function is parallelized with nc indicating the number of cores to use.

Details

This function fits the cgaim on bootstrap samples. It is called internally by the confint.cgaim function, but can also be called directly to generate various statistics.

Three types of bootstrap are currently implemented. "residuals" (the default) resamples the residuals in object to then be added to fitted values, creating alternative response vectors. The cgaim is then fitted on these newly generated y values with the original x. "wild" is similar except that residuals are multiplied by random draws from a standard normal distribution before being added to fitted values. "pairs" resamples directly pairs of y and x to create bootstrap samples.

Bootstrap samples can either be prespecified by the user through bsamples or generated internally. In the former case, the columns of bsamples indicate the number of replications B and the rows should match the original number of observations. Internally generated bootstrap samples are controlled by the number of replications B and block length l, implementing block bootstrap. The latter is particularly recommended for time series data.

As fitting a large number of cgaim models can be computationally intensive, the function can be run in parallel, using the doParallel package. This can be done by setting the argument nc to a value greater than 1, controlling the number of cores used in parallelization.
**Value**

A `boot.cgaim` object with components

- **boot**: The bootstrap result. A list that includes all \( B \) replications of \( \alpha \), \( \beta \), \( g \)fit, and \( indexfit \) organized in arrays.
- **obs**: The original object passed to the function.
- **samples**: The bootstrap samples. A matrix with indices corresponding to original observations.
- **boot.type**: The type of bootstrap performed.
- **B**: The number of bootstrap replications.
- **l**: The block length for block bootstrap.

**Examples**

```r
# A simple CGAIM
n <- 200
x1 <- rnorm(n)
x2 <- x1 + rnorm(n)
z <- x1 + x2
y <- z + rnorm(n)
df1 <- data.frame(y, x1, x2)
ans <- cgaim(y ~ g(x1, x2, acons = list(monotone = 1)), data = df1)

# Use function to compute confidence intervals (B should be increased)
set.seed(1989)
boot1 <- boot.cgaim(ans, B = 10)
ci1 <- confint(boot1)

# Produces the same result as
set.seed(1989)
ci2 <- confint(ans, type = "boot", B = 10)

# Create sampling beforehand
bsamp <- matrix(sample(1:n, n * 10, replace = TRUE), n)
boot2 <- boot.cgaim(ans, bsamples = bsamp)

# Parallel computing (two cores)
boot3 <- boot.cgaim(ans, nc = 2)
```

---

**Description**

Build a constraint matrix from common simple constraints. Internally used by `g` to construct index-specific constraint matrices.
Usage

build_constraints(p, first = 0, sign = 0, monotone = 0, convex = 0)

Arguments

p
The number of variables.

first
Indicates sign constraint for first coefficient. Recommended for identifiability if no other constraint is passed.

sign
Sign constraint applied to all coefficients. 0: no constraint,

monotone
Monotonicity constraint. 0: no constraint, -1: decreasing coefficients and 1: increasing coefficients.

convex
Convexity constraint. 0: no constraint, -1: convex coefficients and 1: concave coefficients.

Details

For monotonicity and convexity / concavity, the function assumes the coefficients are ordered. For instance, for increasing monotone coefficients, the first one will be lower than the second, which be lower than the third and so on.

The function automatically removes redundant constraints. For instance, if both sign = 1 and monotone = 1, then only the sign constraint on the first variable is kept as others are not needed.

Note that, for all arguments, any number can be passed to the function. In which case, the sign of the argument is used. Therefore passing monotone = 3.14 is the same as passing monotone = 1.

Value

A p-column constraint matrix.

Examples

# By default, produces only the identifiability constraint
build_constraints(4)

# Positive and increasing coefficients
build_constraints(4, sign = 1, monotone = 1)

# Concavity constraint
build_constraints(7, convex = -1)

# Any numeric can be passed to the function
build_constraints(5, monotone = pi)
CGAIM

Constrained groupwise additive index models

Description

Fits constrained groupwise additive index models (CGAIM) to data. CGAIM fits indices subjected to constraints on their coefficients and shape of their association with the outcome. Such constraints can be specified in the formula through g for grouped terms and s for smooth covariates.

Usage

cgaim(formula, data, weights, subset, na.action, Cmat = NULL, bvec = NULL, control = list())

Arguments

- formula: A CGAIM formula with index terms g, smooth terms s and linear terms. See details.
- data: A data.frame containing the variables of the model.
- weights: An optional vector of observation weights.
- subset: An optional vector specifying a subset of observations to be used in the fitting process.
- na.action: A function indicating how to treat NAs. The default is set by the na.action setting of options. See na.fail.
- Cmat: A constraint matrix for index coefficients alpha. Columns must match all variables entering any index through g. See details.
- bvec: A vector of lower bounds for the constraints in Cmat. Potentially recycled to match the number of constraints.
- control: A list of parameters controlling the fitting process. See cgaim.control.

Details

The CGAIM is expressed

\[ y_i = \beta_0 + \sum_j \beta_j g_j(\alpha_j^T x_{ij}) + \sum_k \gamma_k f_k(w_{ik}) + \sum_l \theta_l u_{il} + e_i \]

where the \( x_{ij} \) are variables entering grouped indices, the \( w_{ik} \) are smooth covariates and the \( u_{il} \) are linear covariates.

The formula interface considers g to identify index terms, s for smooth functions and can also include linear terms as usual. All smooth terms can be shape constrained.

The CGAIM allows for linear constraints on the alpha coefficients. Such constraints can be specified through the g interface in the formula, or through alpha.control$Cmat. The g interface is used for constraints meant for a specific index only. In this case, common constraints can easily be specified through the acons argument (see build_constraints). Alternatively, more general constraint can
be specified by passing a matrix to the \texttt{Cmat} argument. Constraints encompassing several indices can be specified through an element \texttt{Cmat} in alpha.control. Its number of columns must match the total number of index coefficients alpha to estimate. In all cases, arguments \texttt{bvec} are used to specify the bounds of constraints.

Both indices \((g)\) and smooth covariate terms \((s)\) allow shape constraints. See dedicated help for the list of constraints allowed.

The CGAIM is fitted through an iterative algorithm that alternates between estimating the ridge functions \(g_j\) (and other non-index terms) and updating the coefficients \(\alpha_j\). The smoothing of ridge functions currently supports three methods: \texttt{scam} (the default), \texttt{cgam} and \texttt{scar}. The list \texttt{smooth.control} controls the smoothing with allowed parameters defined in \texttt{cgaim.control}.

\textbf{Value}

A \texttt{cgaim} object, i.e. a list with components:

- \texttt{alpha} A named list of index coefficients.
- \texttt{gfit} A matrix containing the ridge and smooth functions evaluated at the observations. Note that column ordering puts indices first and covariates after.
- \texttt{indexfit} A matrix containing the indices evaluated at the observations.
- \texttt{beta} A vector containing the intercept and the scale coefficient of each ridge and smooth function. Includes the \(\gamma_k\) of the CGAIM model above. Note that ordering puts indices first and covariates after.
- \texttt{index} A vector identifying to which index the columns of the element \texttt{x} belong.
- \texttt{fitted} A vector of fitted responses.
- \texttt{residuals} A vector of residuals.
- \texttt{rss} The residual sum of squares of the fit.
- \texttt{flag} A flag indicating how the algorithm stopped. 1 for proper convergence, 2 when the algorithm stopped for failing to decrease the RSS and 3 when the maximum number of iterations has been reached.
- \texttt{niter} Number of iterations performed.
- \texttt{edf} Effective degrees of freedom of the estimator.
- \texttt{gcv} Generalized cross validation score.
- \texttt{dg} A matrix containing derivatives of ridge and smooth functions.
- \texttt{gse} A matrix containing standard errors of ridge and smooth functions.
- \texttt{active} A logical vector indicating which constraints are active at convergence.
- \texttt{Cmat} The constraint matrix used to fit index coefficients alpha. Will include all constraints given through \texttt{g} and the \texttt{Cmat} parameter.
- \texttt{bvec} The lower bound vector associated with \texttt{Cmat}.
- \texttt{x} A matrix containing the variables entering the indices. The variables are mapped to each index through the element \texttt{index}.
- \texttt{y} The response vector.
- \texttt{weights} The weights used for estimation.
sm_mod  
A list of model elements for the smoothing step of the estimation. Notably includes the matrix \( X_{cov} \) that includes the covariates not entering any index. Other elements depend on the method chosen for smoothing.

control  
The control list used to fit the cgaim.

terms  
The model terms.

Note
A model without intercept can only be fitted when the smoothing step is performed with scam.

See Also
confint.cgaim for confidence intervals, predict.cgaim to predict on new data, plot.cgaim to plot ridge functions.

Examples
```r
## Simulate some data
n <- 200
x1 <- rnorm(n)
x2 <- rnorm(n)
x3 <- rnorm(n)
x4 <- rnorm(n)
mu <- 4 * exp(8 * x1) / (1 + exp(8 * x1)) + exp(x3)
y <- mu + rnorm(n)
df1 <- data.frame(y, x1, x2, x3, x4)

## Fit an unconstrained the model
ans <- cgaim(y ~ g(x1, x2) + g(x3, x4), data = df1)

# Compute confidence intervals
# In practice, higher B values are warranted
cia <- confint(ans, B = 100)
cia$alpha
cia$beta

# Display ridge functions
plot(ans, ci = cia)

# Predict
newdf <- as.data.frame(matrix(rnorm(100), 25, 4))
names(newdf) <- sprintf("x%i", 1:4)
yhat <- predict(ans, newdf)

## Fit constrained model
ans2 <- cgaim(y ~ g(x1, x2, acons = list(monotone = -1)) + g(x3, x4, fcons = "cvx"), data = df1)

# Check results
ans2
plot(ans2)
```
# Same result
Cmat <- as.matrix(Matrix::bdiag(list(build_constraints(2, monotone = -1),
    build_constraints(2, first = 1))))
ans3 <- cgaim(y ~ g(x1, x2) + g(x3, x4, fcons = "cvx"), data = df1,
    Cmat = Cmat)

## A mis-specified model
ans4 <- cgaim(y ~ g(x1, x2, acons = list(monotone = 1)) +
    g(x3, x4, fcons = "dec"), data = df1)

cgaim.control

Parameters controlling the CGAIM fit

Description
Internal function setting the parameters to control the CGAIM fit. Sets default values and check parameters passed by the user. It is called internally by cgaim and should not be called directly by the user.

Usage

cgaim.control(max.iter = 50, tol = 0.001, halving = TRUE,
    min.step.len = 0.1, convergence_criterion = "rss", trace = FALSE,
    alpha.start = NULL, init.type = "regression", norm.type = "1",
    check.Cmat = TRUE, solver = "osqp", ctol = 0.001, qp_pars = list(),
    sample_pars = list(), sm_method = "scam", sm_pars = list())

Arguments

max.iter The maximum number of iteration allowed.
tol The tolerance for convergence. The algorithm stops when the convergence criterion falls below tol.
halving Logical turning on and off halving for bad steps. See details.
min.step.len Numeric between 0 and 1 giving the minimum descent step length to be considered in case of bad step. See details.
convergence_criterion Character indicating the convergence criterion for the algorithm. See details.
trace If TRUE, prints the convergence criterion and alpha coefficients at each step.
alpha.start An optional vector or list of starting alpha values. If NULL, starting values are generated internally. See the init.type argument.
init.type The type of initialization to perform if no initial value is provided. If init.type = "regression" (the default), starting values are generated by regressing the index design matrix on the response. If init.type = "random", feasible starting values are generated randomly.
norm.type The type of norm used to normalize index coefficients vectors. See norm for available norms. Default to L1 norm meaning that, for each index, absolute values of coefficients sum to 1.

check.Cmat Logical indicating whether to check for redundant constraints and remove them from Cmat.

solver The quadratic programming solver to use. One of "osqp" (the default), "quadprog" or "coneproj".

cctl Tolerance value on constraints. See details.

qp_pars A named list of parameters to be passed to the solver function.

sample_pars A named list of parameters to be passed to xsample when randomly generating initial alpha coefficients.

sm_method Character specifying which method to use for constrained smoothing. Either scam (the default), cgam or scar.

sm_pars Named list to pass specific parameters to the smoothing function of sm_method. See help pages of corresponding functions.

Details

The model is fitted by an iterative sequential quadratic programming algorithm. The algorithm iterates between updating the index alpha coefficients and updating the smoothing of indices and covariates. It stops when the criterion given in convergence_criterion is below tol, or when max.iter is reached. Convergence criteria include rss for which the algorithm stops when the relative decrease in residual sum of squares, \( (rss_{new} - rss_{old}) / rss_{old} \) is below tol, alpha for which the algorithm stops when the largest update \( \max(\text{abs}(alpha_{new} - alpha_{old}) / \text{abs}(alpha_{old})) \) is below tol, and offset when the scalar product between the RSS and current direction (measuring orthogonality) is below tol (EXPERIMENTAL, use at your own risk).

By default, when the RSS fails to decrease during a step (a "bad" step), the step length is iteratively halved until the RSS decreases. The minimum step length allowed is controlled by min.step.len as the proportion of the original step length. This is a common behaviour in non-linear least squares and is implement in nls for instance, but can be turned off by setting halving = FALSE, in which case the algorithm stops for any bad step.

The alpha updating step consists in estimating an update vector in a descent direction by a constrained regression of index derivatives on the current residuals of the model. This is fitted through a quadratic program, ensuring the updated coefficients respect the constraints at each step of the algorithm. Initial values can either be provided by the user through the argument alpha.start or be internally generated. The latter is controlled by the argument init.type allowing to initialize the weights either by regressing the index variables on the response (init.type = "regression") ensuring feasible starting values (the default), or by randomly generating feasible values (init.type = "random"). In the latter case, random generation is performed by the function xsample which can be controlled by the parameter sample_pars. When random initial values are chosen, it is recommended to fit the algorithm several time and keep the best fit, to avoid falling into a local minimum.

At the moment, three solvers are available to perform quadratic programming, which can be controlled by the argument solver. By default the function solve_osqp (solver = "osqp") is used. Alternatively the more established but slower function solve.QP (solver = "quadprog") as well as qprog (solver = "coneproj") functions can be used. Although default parameters are internally
set for these function, they can entirely be controlled through the argument `qp_pars`. See their specific help pages for details.

In some cases, minimal numerical imprecision in the repeated call to quadratic program, along with the normalization of alpha coefficients, can lead to unfeasible alphas at convergence. To avoid this, these imprecision are compensated by adding a small tolerance `ctol` to the constraints, defaulting to 0.001. If no tolerance is wanted, it can be set to 0.

By default, the package automatically checks that `Cmat` is irreducible, i.e. that no constraint is redundant. A constraint is redundant if it can be expressed as a non-negative linear combination of other constraints. If `check.Cmat = TRUE`, such constraints are removed with a warning.

**Value**

A named list containing all arguments to be used in `cgaim`.

**References**


**See Also**

These parameters control the fitting of `cgaim`.

---

### confint.cgaim

**Confidence intervals**

**Description**

Computes confidence intervals for the CGAIM components.

**Usage**

```r
## S3 method for class 'cgaim'
confint(object, parm, level = 0.95, type = c("normal", "bootstrap"), B = 100, ...)

## S3 method for class 'boot.cgaim'
confint(object, parm, level = 0.95, ...)
```
Arguments

object  A `cgaim` or `boot.cgaim` object.
parm    The model components for which to get confidence intervals. One or several of: "alpha" for index weights, "beta" for scaling coefficients and "g" for ridge function. By default, returns confidence intervals for all components.
level   The level of confidence intervals. Default to 0.95.
type    The type of confidence intervals. Either "normal" (the default) or "bootstrap". See details.
B       The number of samples to be simulated.
...     Additional parameters to be passed to `boot.cgaim` for bootstrap confidence intervals.

Details

Two types of confidence intervals are currently implemented in the function. When type = "normal", confidence intervals are computed assuming components are normally distributed. Beta coefficients are treated as regular linear regression coefficients and g as regular smooth functions estimated by (shape-constrained) generalized additive models. For alpha coefficients, we consider a linear transformation mapping them to a Truncated Multivariate Normal distribution (i.e. with only bound constraints). Simulation from the TMVN are performed (see `tmvnorm`) and transformed back into the original coefficient space (i.e. with linear constraints). The parameter B controls the number of simulations from the TMVN. Confidence intervals are computed as the percentiles of these simulated coefficients, ensuring the confidence region is entirely within the feasible region defined by the constraints.

When type = "bootstrap", confidence intervals are estimated by percentile bootstrap. `boot.cgaim` is called internally to create B samples of model components, and confidence intervals are then computed as the percentiles of bootstrap samples. Alternatively, the user can directly call `boot.cgaim` and feed the result into `confint.boot.cgaim`.

Value

A list of confidence intervals. Contains one element per model component in the parm argument.

Note

Confidence intervals for the g functions are evaluated on the same n index values as the functions in object.

References

See Also

boot.cgaim for bootstrapping.

Examples

```r
# A simple CGAIM
n <- 200
x1 <- rnorm(n)
x2 <- x1 + rnorm(n)
z <- x1 + x2
y <- z + rnorm(n)
df1 <- data.frame(y, x1, x2)
ans <- cgaim(y ~ g(x1, x2, acons = list(monotone = 1)), data = df1)

# Normal confidence intervals
set.seed(1)
ci1 <- confint(ans, B = 1000)
ci1$alpha

# Select only alphas: identical to above result
set.seed(1)
confint(ans, B = 1000, parm = "alpha")

# Select only betas: identical to above result
set.seed(1)
confint(ans, B = 1000, parm = "beta")

# Confidence intervals by bootstrap (more computationally intensive, B should be increased)
set.seed(2)

ci2 <- confint(ans, type = "boot", B = 10)

# Alternatively, bootstrap samples can be performed beforehand
set.seed(2)
boot1 <- boot.cgaim(ans, B = 10)
ci3 <- confint(boot1)
```

---

**g**

*Defining terms in CGAIM formula*

**Description**

Functions used to define terms within a cgaim formula. g defines an index with ridge function and s a smooth covariate.
Usage

\[ g(..., \text{label} = \text{NULL}, \text{acons} = \text{list}(), \text{Cmat} = \text{NULL}, \text{bvec} = 0, \]
\[ \text{fcons} = \text{NULL}, \text{s_opts} = \text{list}()) \]
\[ s(x, \text{fcons} = \text{NULL}, \text{s_opts} = \text{list}()) \]

Arguments

... Variables entering the index. May include vectors and matrices.

label Character (or any object that can coerced into one) labeling the index. By default, named after the first variable in ....

acons A list of character naming common constraints to be applied to the index weights alpha. See \texttt{build_constraints} for allowed constraints.

Cmat A constraint matrix for alpha coefficients. Number of columns must match the number of variables in the index.

bvec Numeric vector of constraint bounds. Recycled if necessary.

fcons The type of shape constraint to be applied on the smooth function. See details.

s_opts A named list of options to be passed to the smoothing of ridge functions. Depends on the method used to smooth additive models. See details.

x Covariate on which the smooth is applied.

Details

These functions define nonlinear terms in the formula, with \texttt{g} defining an index created from a collection of terms passed through the \texttt{...} argument while \texttt{s} is applied to a single variable, similarly to \texttt{s} in \texttt{mgcv}.

For indices, \texttt{g} allows the definition of constraints applied to the index only. This is a convenient alternative to passing the whole constraint matrix \texttt{Cmat} in \texttt{cgaim}. Constraints can be defined by a prespecified matrix through the argument \texttt{Cmat} or through the argument \texttt{acons} for common constraints (see \texttt{build_constraints}). Note that any provided \texttt{Cmat} must match the total number of variables in \texttt{...}, including potential matrix expansion and factors. Both \texttt{Cmat} and \texttt{acons} can be passed to the function, which will bind them internally.

Both \texttt{g} and \texttt{s} allow the definition of shape constraints for the smooth. Eight shape-constraints are currently available: monotone increasing (\texttt{fcons} = "inc"), monotone decreasing (\texttt{fcons} = "dec"), convex (\texttt{fcons} = "cvx"), concave (\texttt{fcons} = "ccv"), increasing and convex(\texttt{fcons} = "inccvx"), decreasing and convex (\texttt{fcons} = "deccvx"), increasing and concave (\texttt{fcons} = "incccv"), decreasing and concave (\texttt{fcons} = "decccv").

Smoothing can be controlled by the \texttt{s_opts} parameter. It is a list of argument depends on the method used for smoothing. See \texttt{s} for \texttt{smooth_method = "scam"}. For \texttt{smooth_method = "cgam"}, the parameters allowed may vary according to the shape-constraint chosen. The full list can be found in \texttt{cgam}, but only the constraints beginning with \texttt{s.} are allowed for now. No parameter are necessary when \texttt{smooth_method = "scar"} (see \texttt{scar}).
Value

A matrix containing the variables passed in ... with additional attributes:

- `fcons`  The shape constraint for smoothing.
- `s_opts` Arguments passed to the smoothing function.
- `label`  The label of the term.

The following attributes result from a call to `g` only:

- `term`  The terms in the index.
- `nterms` The number of variables in the index.
- `Cmat`  The constraint matrix for alpha coefficients.
- `bvec`  The associated boundary vector.

See Also

cgaim for fitting the CGAIM, build_constraints for built-in constraint matrices.

Examples

```r
## Simulate some data
n <- 200
x1 <- rnorm(n)
x2 <- rnorm(n)
x3 <- rnorm(n)
x4 <- rnorm(n)
mu <- 4 * exp(8 * x1) / (1 + exp(8 * x1)) + exp(x3)
y <- mu + rnorm(n)
df1 <- data.frame(y, x1, x2, x3, x4)

## Fit an unconstrained the model
ans <- cgaim(y ~ g(x1, x2) + g(x3, x4), data = df1)

## Fit constrained model
ans2 <- cgaim(y ~ g(x1, x2, acons = list(monotone = -1)) +
              g(x3, x4, fcons = "cvx"), data = df1)

## Pass constraint matrices instead
ans3 <- cgaim(y ~ g(x1, x2, Cmat = -diff(diag(2))) +
              g(x3, x4, fcons = "cvx"), data = df1)

## Label indices
ans4 <- cgaim(y ~ g(x1, x2, label = "foo") + g(x3, x4, label = "bar"),
              data = df1)
```
Description

Plot method for the ridge and smooth terms of a cgaim object. If provided, also plots confidence intervals.

Usage

```r
## S3 method for class 'cgaim'
plot(x, select = NULL, ci = NULL, ci.plot = c("polygon", "lines"),
     ci.args = list(), add = FALSE, xcenter = FALSE, xscale = FALSE,
     yshift = FALSE, yscale = FALSE, ...)
```

Arguments

- `x`: A cgaim object.
- `select`: A numeric or character vector indicating which terms to plot.
- `ci`: An object returned by a call to `confint.cgaim`. If NULL, no confidence interval is drawn.
- `ci.plot`: Whether to plot the confidence intervals as shaded areas `ci.plot = "polygon"` or as lines `ci.plot = "lines"`.
- `ci.args`: Additional arguments to be passed to the function used to draw confidence interval. Either `polygon` or `lines`.
- `add`: Logical. If TRUE, adds the function to the current active plot.
- `xcenter`, `xscale`: Centering and scaling values for the x axis. See `scale`.
- `yshift`, `yscale`: Either logical or numeric values to shift and scale the ridge functions. See details.
- `...`: Additional graphical parameters for the drawn function. See `par`.

Details

The values of `yshift` and `yscale` determine how ridge functions are shifted and scaled for plotting. This can be used to display the functions over data points for instance. If numeric, a vector can be passed with one value for each plotted function. The vector is recycled if necessary. This indicate the desired mean and standard deviation of plotted ridge functions. Note that this is inverse to the parameters in `scale` (and `xcenter`, `xscale`). If TRUE is passed instead, functions are shifted to the intercept and scaled to their corresponding beta coefficients, placing them on the response scale.

Value

The function is called to generate plots and returns no value.
See Also

cgaim for the main fitting function and confint.cgaim for confidence interval computation.

Examples

```r
## Simulate some data
n <- 200
x1 <- rnorm(n)
x2 <- rnorm(n)
x3 <- rnorm(n)
x4 <- rnorm(n)
mu <- 4 * exp(8 * x1) / (1 + exp(8 * x1)) + exp(x3)
y <- 5 + mu + rnorm(n)
df1 <- data.frame(y, x1, x2, x3, x4)

## Fit a model
ans <- cgaim(y ~ g(x1, x2, label = "foo") + g(x3, x4, label = "bar"),
               data = df1)

## Default plot method
plot(ans)

## Select variable
plot(ans, select = 1)
# Same as
plot(ans, select = "foo")

## Add confidence intervals
ci <- confint(ans)
plot(ans, select = 1, ci = ci)

## Change scale and location
# On the response scale
plot(ans, select = 1, ci = ci, yshift = TRUE, yscale = TRUE)

# Arbitrary scale
plot(ans, select = 1, ci = ci, yshift = 1000)

## Change look
# Main line
plot(ans, select = 1, ci = ci, col = 2, lwd = 3)

# Confidence intervals
plot(ans, select = 1, ci = ci, col = 2, lwd = 3,
     ci.args = list(col = adjustcolor(2, .5)))

# Confidence interval type
plot(ans, select = 1, ci = ci, ci.plot = "lines", col = 2, lwd = 3,
     ci.args = list(col = 2, lty = 4))
```
## Put curves on the same plot (need to shift and scale)

```r
plot(ans, select = 1, col = 2, ylim = c(-2, 3))
plot(ans, select = 2, col = 4, add = TRUE)
```

---

**predict.cgaim**

Predictions from a fitted CGAIM object

### Description

Uses a fitted cgaim object and computes prediction for the observed data or new data. Predicts the response, indices or ridge functions values at the provided data.

### Usage

```r
## S3 method for class 'cgaim'
predict(object, newdata, type = c("response", "terms", "scterms", "indices"), select = NULL, na.action = "na.pass", ...)
```

### Arguments

- **object**: A `gaim` object.
- **newdata**: A list or data.frame containing the new data to predict. If missing, fitted values from the model are returned.
- **type**: A character indicating the type of prediction to return. `type = "response"` returns the predicted response. `type = "terms"`, returns ridge and smooth functions evaluated at index predicted for `newdata`. `type = "scterms"` is the same, except that terms are postmultiplied by their scaling coefficients `beta`. `type = "indices"` returns predicted indices values.
- **select**: A numeric or character vector indicating terms to return for all types except "response".
- **na.action**: A function indicating how to treat NAs. See `na.fail`.
- **...**: For compatibility with the default `predict` method. Unused at the moment.

### Details

- `type = "terms"` returns the scaled ridge functions, i.e. before being multiplied by scaling coefficients `beta`.

### Value

- When `type = "response"` returns a vector of predicted response. When `type = "terms"` or "scterms", returns a matrix of evaluated ridge and smooth terms. When `type = "indices"`, returns a matrix of evaluated indices.
See Also

cgaim for main fitting function

Examples

## Simulate some data
n <- 200
x1 <- rnorm(n)
x2 <- rnorm(n)
x3 <- rnorm(n)
x4 <- rnorm(n)
mu <- 4 * exp(8 * x1) / (1 + exp(8 * x1)) + exp(x3)
y <- mu + rnorm(n)
df1 <- data.frame(y, x1, x2, x3, x4)

## Fit an unconstrained the model
ans <- cgaim(y ~ g(x1, x2, label = "foo") + g(x3, x4, label = "bar"),
              data = df1)

## Get fitted values
yhat <- predict(ans)

## Predict on new data
newdf <- as.data.frame(matrix(rnorm(100), 25, 4))
names(newdf) <- sprintf("x%i", 1:4)

# predicted response
ypred <- predict(ans, newdf)

# Indices
indices <- predict(ans, newdata = newdf, type = "indices")

# Ridge functions
funs <- predict(ans, newdata = newdf, type = "terms")

## Select specific terms
ind1 <- predict(ans, newdata = newdf, select = "foo", type = "indices")
fun1 <- predict(ans, newdata = newdf, select = "foo", type = "terms")

# Plot
plot(ans, select = "foo")
points(ind1, fun1)

## Scaled terms
fun2 <- predict(ans, newdata = newdf, select = "foo", type = "scterms")

# Plot
plot(ans, select = "foo", yscale = TRUE)
points(ind1, fun2)
print.cgaim

print.cgaim       Print a cgaim object

Description

Default method to print the results from a call to cgaim. Conveniently prints the formula, beta and alpha coefficients as well as the residual sum of squares.

Usage

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

Arguments

x       A cgaim object.
...

For compatibility with the default print method. Unused at the moment.

Value

Function called for its side effect of printing a cgaim object. Returns no value.

See Also

The main fitting function cgaim.
Index

boot.cgaim, 2, 11, 12
build_constraints, 3, 5, 13, 14

cgaim, 5, 8, 10, 13, 14, 16, 18, 19
cgaim.control, 5, 6, 8
cgam, 6, 9, 13
confint.boot.cgaim (confint.cgaim), 10
confint.cgaim, 2, 7, 10, 15, 16
doParallel, 2
g, 3, 5, 6, 12
lines, 15
na.fail, 5, 17
nls, 9
norm, 9
par, 15
plot.cgaim, 7, 15
polygon, 15
predict.cgaim, 7, 17
print.cgaim, 19
qprog, 9
s, 5, 6, 13
s (g), 12
scale, 15
scam, 6, 9
scar, 6, 9, 13
solve.QP, 9
solve_osqp, 9
tmvnorm, 11
xsample, 9