Package ‘qgam’

January 24, 2019

Type Package

Title Smooth Additive Quantile Regression Models

Version 1.2.3

Date 2019-01-20

Maintainer Matteo Fasiolo <matteo.fasiolo@gmail.com>

Description Smooth additive quantile regression models, fitted using the methods of Fasiolo et al. (2017) <arXiv:1707.03307>. Differently from 'quantreg', the smoothing parameters are estimated automatically by marginal loss minimization, while the regression coefficients are estimated using either PIRLS or Newton algorithm. The learning rate is determined so that the Bayesian credible intervals of the estimated effects have approximately the correct coverage. The main function is qgam() which is similar to gam() in 'mgcv', but fits non-parametric quantile regression models.

License GPL (>= 2)

Depends mgcv (>= 1.8-23)

Imports shiny, plyr, doParallel, parallel, grDevices

Suggests knitr, MASS, RhpcBLASctl, testthat

VignetteBuilder knitr

RoxygenNote 6.0.1

NeedsCompilation yes

Author Matteo Fasiolo [aut, cre],
Simon N. Wood [ctb],
Yannig Goude [ctb],
Raphael Nedellec [ctb]

Repository CRAN

Date/Publication 2019-01-24 13:00:04 UTC

R topics documented:

`check` *Generic checking function*

Description

Generic function for checking R objects which produces, for instance, convergence tests or diagnostic plots. For `qgam` objects `check.qgam()` will be used.

Usage

```
check(obj, ...)
```

Arguments

`obj`	the object to be checked.
`...`	extra arguments, mainly used by graphic functions.

Value

Reports the results of convergence tests and/or produces diagnostic plots.

Author(s)

Matteo Fasiolo <matteo.fasiolo@gmail.com>.

Examples

```
#######
# Using check.qgam
#######
library(qgam)
set.seed(0)
dat <- gamSim(1, n=200)
b<-qgam(y~s(x0)+s(x1)+s(x2)+s(x3), data=dat, qu = 0.5,
        control = list("tol" = 0.01)) # <- semi-sloppy tolerance to speed-up calibration
plot(b, pages=1)
check(b, pch=19, cex=.3)
```

---

check.learn *Visual checks for the output of tuneLearn()*

Description

Provides some visual plots showing how the calibration criterion and the effective degrees of freedom of each smooth component vary with the learning rate.

Usage

```
## S3 method for class 'learn'
check(obj, sel = 1:2, ...)
```

Arguments

obj	the output of a call to `tuneLearn`.
sel	this function produces two plots, set this parameter to 1 to plot only the first, to 2 to plot only the second or leave it to 1:2 to plot both.
...	currently not used, here only for compatibility reasons.

Details

The first plot shows how the calibrations loss, which we are trying to minimize, varies with the log learning rate. This function should look quite smooth, if it doesn't then try to increase `err` or `control$K` (the number of bootstrap samples) in the original call to `tuneLearn`. The second plot shows how the effective degrees of freedom of each smooth term vary with log(sigma). Generally as log(sigma) increases the complexity of the fit decreases, hence the slope is negative.

Value

It produces several plots.

Author(s)

Matteo Fasiolo <matteo.fasiolo@gmail.com>.

References

Fasiolo, M., Goude, Y., Nedellec, R. and Wood, S. N. (2017). Fast calibrated additive quantile regression. Available at https://arxiv.org/abs/1707.03307.

Examples

```
library(qgam)
set.seed(525)
dat <- gamSim(1, n=200)
b <- tuneLearn(lsig = seq(-0.5, 1, length.out = 10),
               y~s(x0)+s(x1)+s(x2)+s(x3),
               data=dat, qu = 0.5)
check(b)
```

---

check.learnFast *Visual checks for the output of tuneLearnFast()*

Description

Provides some visual checks to verify whether the Brent optimizer used by `tuneLearnFast()` worked correctly.

Usage

```
## S3 method for class 'learnFast'
check(obj, sel = NULL, ...)
```

Arguments

`obj`	the output of a call to `tuneLearnFast`.
`sel`	integer vector determining which of the plots will be produced. For instance if `sel = c(1, 3)` only the 1st and 3rd plots are showed. No entry of `sel` can be bigger than the number of quantiles considered in the original `tuneLearnFast()` call. That is, if estimated the learning rate for `qu = c(0.1, 0.4)`, then `max(sel)` must be <= 3.
`...`	currently not used, here only for compatibility reasons.

Details

The top plot in the first page shows the bracket used to estimate log(sigma) for each quantile. The brackets are delimited by the crosses and the red dots are the estimates. If a dot falls very close to one of the crosses, that might indicate problems. The bottom plot shows, for each quantile, the value of parameter `err` used. Sometimes the algorithm needs to increase `err` above its user-defined value to achieve convergence. Subsequent plots show, for each quantile, the value of the loss function corresponding to each value of log(sigma) explored by Brent algorithm.

Value

It produces several plots.

Author(s)

Matteo Fasiolo <matteo.fasiolo@gmail.com>.

References

Fasiolo, M., Goude, Y., Nedellec, R. and Wood, S. N. (2017). Fast calibrated additive quantile regression. Available at https://arxiv.org/abs/1707.03307.

Examples

```
library(qgam)
set.seed(525)
dat <- gamSim(1, n=200)
b <- tuneLearnFast(y ~ s(x0)+s(x1)+s(x2)+s(x3),
                   data = dat, qu = c(0.4, 0.5),
               control = list("tol" = 0.05)) # <- sloppy tolerance to speed-up calibration
check(b)
check(b, 3) # Produces only third plot
```

`check.qgam` *Some diagnostics for a fitted qgam model*

Description

Takes a fitted gam object produced by `qgam()` and produces some diagnostic information about the fitting procedure and results. It is partially based on `mgcv::gam.check`.

Usage

```
## S3 method for class 'qgam'
check(obj, nbin = 10, lev = 0.05, ...)
```

Arguments

`obj`	the output of a `qgam()` call.
`nbin`	number of bins used in the internal call to `cqcheck()`.
`lev`	the significance levels used by `cqcheck()`, which determines the width of the confidence intervals.
`...`	extra arguments to be passed to `plot()`

Details

This function provides two plots. The first shows how the number of responses falling below the fitted quantile (y-axis) changes with the fitted quantile (x-axis). To be clear: if the quantile is fixed to, say, 0.5 we expect 50% of the responses to fall below the fit. See `?cqcheck()` for details. The second plot related to `|F(hat(mu)) - F(mu0)|`, which is the absolute bias attributable to the fact that qgam is using a smoothed version of the pinball-loss. The absolute bias is evaluated at each observation, and an histogram is produced. See Fasiolo et al. (2017) for details. The function also prints out the integrated absolute bias, and the proportion of observations lying below the regression line. It also provides some convergence diagnostics (regarding the optimization), which are the same as in `mgcv::gam.check`. It reports also the maximum (k') and the selected degrees of freedom of each smooth term.

Value

Simply produces some plots and prints out some diagnostics.

Author(s)

Matteo Fasiolo <matteo.fasiolo@gmail.com>, Simon N. Wood.

References

Fasiolo, M., Goude, Y., Nedellec, R. and Wood, S. N. (2017). Fast calibrated additive quantile regression. Available at https://arxiv.org/abs/1707.03307.

Examples

```
library(qgam)
set.seed(0)
dat <- gamSim(1, n=200)
b<-qgam(y~s(x0)+s(x1)+s(x2)+s(x3), data=dat, qu = 0.5)
plot(b, pages=1)
check.qgam(b, pch=19, cex=.3)
```

---

cqcheck *Visually checking a fitted quantile model*

Description

Given an additive quantile model, fitted using `qgam`, `cqcheck` provides some plots that allow to check what proportion of responses, `y`, falls below the fitted quantile.

Usage

```
cqcheck(obj, v, X = NULL, y = NULL, nbin = c(10, 10), bound = NULL,
  lev = 0.05, scatter = FALSE, ...)
```

Arguments

`obj`	the output of a `qgam` call.
`v`	if a 1D plot is required, `v` should be either a single character or a numeric vector. In the first case `v` should be the names of one of the variables in the dataframe `X`. In the second case, the length of `v` should be equal to the number of rows of `X`. If a 2D plot is required, `v` should be either a vector of two characters or a matrix with two columns.
`X`	a dataframe containing the data used to obtain the conditional quantiles. By default it is NULL, in which case predictions are made using the model matrix in `obj$model`.
`y`	vector of responses. Its i-th entry corresponds to the i-th row of X. By default it is NULL, in which case it is internally set to `obj$y`.
`nbin`	a vector of integers of length one (1D case) or two (2D case) indicating the number of bins to be used in each direction. Used only if `bound==NULL`.
`bound`	in the 1D case it is a numeric vector whose increasing entries represent the bounds of each bin. In the 2D case a list of two vectors should be provided. `NULL` by default.
`lev`	the significance levels used in the plots, this determines the width of the confidence intervals. Default is 0.05.
`scatter`	if TRUE a scatterplot is added (using the `points` function). FALSE by default.
`...`	extra graphical parameters to be passed to `plot()`.

Details

Having fitted an additive model for, say, quantile `qu=0.4` one would expect that about 40 responses fall below the fitted quantile. This function allows to visually compare the empirical number of responses (`qu_hat`) falling below the fit with its theoretical value (`qu`). In particular, the responses are binned, which the bins being constructed along one or two variables (given be arguments `v`). Let (`qu_hat[i]`) be the proportion of responses below the fitted quantile in the ith bin. This should be approximately equal to `qu`, for every i. In the 1D case, when `v` is a single character or a numeric vector, `cqcheck` provides a plot where: the horizontal line is `qu`, the dots correspond to `qu_hat[i]` and the grey lines are confidence intervals for `qu`. The confidence intervals are based on `qbinom(lev/2, siz, qu)`, if the dots fall outside them, then `qu_hat[i]` might be deviating too much from `qu`. In the 2D case, when `v` is a vector of two characters or a matrix with two columns, we plot a grid of bins. The responses are divided between the bins as before, but now don't plot the confidence intervals. Instead we report the empirical proportions `qu_hat[i]` for the non-empty bin, and with colour the bins in red if `qu_hat[i]<qu` and in green otherwise. If `qu_hat[i]` falls outside the confidence intervals we put an * next to the numeric `qu_hat[i]` and we use more intense colours.

Value

Simply produces a plot.

Author(s)

Matteo Fasiolo <matteo.fasiolo@gmail.com>.

Examples

```
#######
# Bivariate additive model y~1+x+x^2+z+x*z/2+e, e~N(0, 1)
#######
## Not run:
library(qgam)
set.seed(15560)
n <- 500
x <- rnorm(n, 0, 1); z <- rnorm(n)
X <- cbind(1, x, x^2, z, x*z)
beta <- c(0, 1, 1, 1, 0.5)
y <- drop(X %*% beta) + rnorm(n)
dataf <- data.frame(cbind(y, x, z))
names(dataf) <- c("y", "x", "z")

#### Fit a constant model for median
qu <- 0.5
fit <- qgam(y~1, qu = qu, data = dataf)

# Look at what happens along x: clearly there is non linear pattern here
cqcheck(obj = fit, v = c("x"), X = dataf, y = y)

#### Add a smooth for x
fit <- qgam(y~s(x), qu = qu, err = 0.05, data = dataf)
cqcheck(obj = fit, v = c("x"), X = dataf, y = y) # Better!

# Lets look across x and z. As we move along z (x2 in the plot)
# the colour changes from green to red
cqcheck(obj = fit, v = c("x", "z"), X = dataf, y = y, nbin = c(5, 5))

# The effect look pretty linear
cqcheck(obj = fit, v = c("z"), X = dataf, y = y, nbin = c(10))

#### Lets add a linear effect for z
fit <- qgam(y~s(x)+z, qu = qu, data = dataf)

# Looks better!
cqcheck(obj = fit, v = c("z"))

# Lets look across x and y again: green prevails on the top-left to bottom-right
# diagonal, while the other diagonal is mainly red.
cqcheck(obj = fit, v = c("x", "z"), nbin = c(5, 5))

### Maybe adding an interaction would help?
fit <- qgam(y~s(x)+z+I(x*z), qu = qu, data = dataf)

# It does! The real model is: y ~ 1 + x + x^2 + z + x*z/2 + e, e ~ N(0, 1)
cqcheck(obj = fit, v = c("x", "z"), nbin = c(5, 5))

## End(Not run)
```

`cqcheckI` *Interactive visual checks for additive quantile fits*

Description

Given an additive quantile model, fitted using `qgam`, `cqcheck2DI` provides some interactive 2D plots that allow to check what proportion of responses, `y`, falls below the fitted quantile. This is an interactive version of the `cqcheck` function.

Usage

```
cqcheckI(obj, v, X = NULL, y = NULL, run = TRUE, width = "100%",
  height = "680px")
```

Arguments

`obj`	the output of a `qgam` call.
`v`	if a 1D plot is required, `v` should be either a single character or a numeric vector. In the first case `v` should be the names of one of the variables in the dataframe `X`. In the second case, the length of `v` should be equal to the number of rows of `X`. If a 2D plot is required, `v` should be either a vector of two characters or a matrix with two columns.
`X`	a dataframe containing the data used to obtain the conditional quantiles. By default it is NULL, in which case predictions are made using the model matrix in `obj$model`.
`y`	vector of responses. Its i-th entry corresponds to the i-th row of X. By default it is NULL, in which case it is internally set to `obj$y`.
`run`	if TRUE (default) the function produces an interactive plot, otherwise it returns the corresponding shiny app.
`width`	the width of the main plot. Default is "100%".
`height`	width the width of the main plot. Default is "680px".

Details

This is an interactive version of the `cqcheck`, see `?cqcheck` for details. The main interactive feature is that one can select an area by brushing, and then double-click to zoom in. In the 1D case the vertical part of the selected area is not use: we zoom only along the x axis. Double-clicking without brushing zooms out.

Value

Simply produces an interactive plot.

Author(s)

Matteo Fasiolo <matteo.fasiolo@gmail.com>.

Examples

```
## Not run:
#######
# Example 1: Bivariate additive model y~1+x+x^2+z+x*z/2+e, e~N(0, 1)
#######
library(qgam)
set.seed(15560)
n <- 1000
x <- rnorm(n, 0, 1); z <- rnorm(n)
X <- cbind(1, x, x^2, z, x*z)
beta <- c(0, 1, 1, 1, 0.5)
y <- drop(X %*% beta) + rnorm(n)
dataf <- data.frame(cbind(y, x, z))
names(dataf) <- c("y", "x", "z")

#### Fit a constant model for median
qu <- 0.5
fit <- qgam(y~1, qu = qu, err = 0.05, data = dataf)

# Look at what happens along x: clearly there is non linear pattern here
cqcheckI(obj = fit, v = c("x"), X = dataf, y = y)

#### Add a smooth for x
fit <- qgam(y~s(x), qu = qu, err = 0.05, data = dataf)
cqcheckI(obj = fit, v = c("x"), X = dataf, y = y) # Better!

# Lets look across across x and z. As we move along z (x2 in the plot)
# the colour changes from green to red
cqcheckI(obj = fit, v = c("x", "z"), X = dataf, y = y)

# The effect look pretty linear
cqcheckI(obj = fit, v = c("z"), X = dataf, y = y)

#### Lets add a linear effect for z
fit <- qgam(y~s(x)+z, qu = qu, err = 0.05, data = dataf)

# Looks better!
cqcheckI(obj = fit, v = c("z"))

# Lets look across x and y again: green prevails on the top-left to bottom-right
# diagonal, while the other diagonal is mainly red.
cqcheckI(obj = fit, v = c("x", "z"))

### Maybe adding an interaction would help?
fit <- qgam(y~s(x)+z+I(x*z), qu = qu, err = 0.05, data = dataf)

# It does! The real model is: y ~ 1 + x + x^2 + z + x*z/2 + e, e ~ N(0, 1)
cqcheckI(obj = fit, v = c("x", "z"))

## End(Not run)
```

elf *Extended log-F model with fixed scale*

Description

The `elf` family implements the Extended log-F density of Fasiolo et al. (2017) and it is supposed to work in conjuction with the extended GAM methods of Wood et al. (2017), implemented by `mgcv`. It differs from the `elflss` family, because here the scale of the density (sigma, aka the learning rate) is a single scalar, while in `elflss` it can depend on the covariates. At the moment the family is mainly intended for internal use, use the `qgam` function to fit quantile GAMs based on ELF.

Usage

```
elf(theta = NULL, link = "identity", qu, co)
```

Arguments

`theta`	a scalar representing the log-scale log(sigma).
`link`	the link function between the linear predictor and the quantile location.
`qu`	parameter in (0, 1) representing the chosen quantile. For instance, to fit the median choose `qu=0.5`.
`co`	positive constant used to determine parameter lambda of the ELF density (lambda = co / sigma).

Details

This function is meant for internal use only.

Value

An object inheriting from mgcv's class `extended.family`.

Author(s)

Matteo Fasiolo <matteo.fasiolo@gmail.com> and Simon N. Wood.

References

Fasiolo, M., Goude, Y., Nedellec, R. and Wood, S. N. (2017). Fast calibrated additive quantile regression. Available at https://arxiv.org/abs/1707.03307.

Wood, Simon N., Pya, N. and Safken, B. (2017). Smoothing parameter and model selection for general smooth models. Journal of the American Statistical Association.

Examples

```
library(qgam)
set.seed(2)
dat <- gamSim(1,n=400,dist="normal",scale=2)

# Fit median using elf directly: FAST BUT NOT RECOMMENDED
fit <- gam(y~s(x0)+s(x1)+s(x2)+s(x3),
           family = elf(co = 0.1, qu = 0.5), data = dat)
plot(fit, scale = FALSE, pages = 1)

# Using qgam: RECOMMENDED
fit <- qgam(y~s(x0)+s(x1)+s(x2)+s(x3), data=dat, err = 0.05, qu = 0.8)
plot(fit, scale = FALSE, pages = 1)
```

---

elflss — *Extended log-F model with variable scale*

Description

The `elflss` family implements the Extended log-F (ELF) density of Fasiolo et al. (2017) and it is supposed to work in conjuction with the general GAM fitting methods of Wood et al. (2017), implemented by `mgcv`. It differs from the `elf` family, because here the scale of the density (sigma, aka the learning rate) can depend of the covariates, while in while in `elf` it is a single scalar. At the moment the family is mainly intended for internal use, use the `qgam` function to fit quantile GAMs based on ELF.

Usage

```
elflss(link = list("identity", "log"), qu, co, theta, remInter = TRUE)
```

Arguments

`link`	vector of two characters indicating the link function for the quantile location and for the log-scale.
`qu`	parameter in (0, 1) representing the chosen quantile. For instance, to fit the median choose `qu=0.5`.
`co`	positive vector of constants used to determine parameter lambda of the ELF density (lambda = co / sigma).
`theta`	a scalar representing the intercept of the model for the log-scale log(sigma).
`remInter`	if TRUE the intercept of the log-scale model is removed.

Details

This function is meant for internal use only.

Value

An object inheriting from mgcv's class `general.family`.

Author(s)

Matteo Fasiolo <matteo.fasiolo@gmail.com> and Simon N. Wood.

References

Fasiolo, M., Goude, Y., Nedellec, R. and Wood, S. N. (2017). Fast calibrated additive quantile regression. Available at https://arxiv.org/abs/1707.03307.

Wood, Simon N., Pya, N. and Safken, B. (2017). Smoothing parameter and model selection for general smooth models. Journal of the American Statistical Association.

Examples

```
## Not run:
set.seed(651)
n <- 1000
x <- seq(-4, 3, length.out = n)
X <- cbind(1, x, x^2)
beta <- c(0, 1, 1)
sigma =  1.2 + sin(2*x)
f <- drop(X %*% beta)
dat <- f + rnorm(n, 0, sigma)
dataf <- data.frame(cbind(dat, x))
names(dataf) <- c("y", "x")

# Fit median using elf directly: NOT RECOMMENDED
fit <- gam(list(y~s(x, bs = "cr"), ~ s(x, bs = "cr")),
           family = elflss(theta = 0, co = rep(0.2, n), qu = 0.5),
           data = dataf)

plot(x, dat, col = "grey", ylab = "y")
tmp <- predict(fit, se = TRUE)
lines(x, tmp$fit[ , 1])
lines(x, tmp$fit[ , 1] + 3 * tmp$se.fit[ , 1], col = 2)
lines(x, tmp$fit[ , 1] - 3 * tmp$se.fit[ , 1], col = 2)

# Use qgam: RECOMMENDED
fit <- qgam(list(y~s(x, bs = "cr"), ~ s(x, bs = "cr")),
            data = dataf, qu = 0.5, err = 0.2, lsig = 0)

plot(x, dat, col = "grey", ylab = "y")
tmp <- predict(fit, se = TRUE)
lines(x, tmp$fit[ , 1])
lines(x, tmp$fit[ , 1] + 3 * tmp$se.fit[ , 1], col = 2)
lines(x, tmp$fit[ , 1] - 3 * tmp$se.fit[ , 1], col = 2)

## End(Not run)
```

log1pexp *Calculating log(1+exp(x)) accurately*

Description

Calculates `log(1+exp(x))` in a numerically stable fashion.

Usage

```
log1pexp(x)
```

Arguments

`x`	a numeric vector.

Details

We follow the recipe of Machler (2012), that is formula (10) page 7.

Value

A numeric vector where the i-th entry is equal to `log(1+exp(x[i]))`, but computed more stably.

Author(s)

Matteo Fasiolo <matteo.fasiolo@gmail.com>.

References

Machler, M. (2012). Accurately computing log(1-exp(-|a|)). URL: https://cran.r-project.org/package=Rmpfr/vignettes/log1mexp-note.pdf.

Examples

```
set.seed(141)
library(qgam);
x <- rnorm(100, 0, 100)
log1pexp(x) - log1p(exp(x))
```

mqgam — *Fit multiple smooth additive quantile regression models*

Description

This function fits a smooth additive regression model to several quantiles.

Usage

```
mqgam(form, data, qu, lsig = NULL, err = 0.05,
  multicore = !is.null(cluster), cluster = NULL, ncores = detectCores() -
  1, paropts = list(), control = list(), argGam = NULL)
```

Arguments

`form`	A GAM formula, or a list of formulae. See ?mgcv::gam details.
`data`	A data frame or list containing the model response variable and covariates required by the formula. By default the variables are taken from environment(formula): typically the environment from which gam is called.
`qu`	A vectors of quantiles of interest. Each entry should be in (0, 1).
`lsig`	The value of the log learning rate used to create the Gibbs posterior. By defauls `lsig=NULL` and this parameter is estimated by posterior calibration described in Fasiolo et al. (2016). Obviously, the function is much faster if the user provides a value.
`err`	An upper bound on the error of the estimated quantile curve. Should be in (0, 1). If it is a vector, it should be of the same length of `qu`. See Fasiolo et al. (2016) for details.
`multicore`	If TRUE the calibration will happen in parallel.
`cluster`	An object of class `c("SOCKcluster", "cluster")`. This allowes the user to pass her own cluster, which will be used if `multicore == TRUE`. The user has to remember to stop the cluster.
`ncores`	Number of cores used. Relevant if `multicore == TRUE`.
`paropts`	a list of additional options passed into the foreach function when parallel computation is enabled. This is important if (for example) your code relies on external data or packages: use the .export and .packages arguments to supply them so that all cluster nodes have the correct environment set up for computing.
`control`	A list of control parameters. The only one relevant here is `link`, which is the link function used (see `?elf` and `?elflss` for defaults). All other control parameters are used by `tuneLearnFast`. See `?tuneLearnFast` for details.
`argGam`	A list of parameters to be passed to `mgcv::gam`. This list can potentially include all the arguments listed in `?gam`, with the exception of `formula`, `family` and `data`.

Value

A list with entries:

- fit = a gamObject, one for each entry of qu. Notice that the slots model and smooth of each object has been removed to save memory. See ?gamObject.
- model = the model slot of the gamObjects in the fit slot. This is the same for every fit, hence only one copy is stored.
- smooth = the smooth slot of the gamObjects in the fit slot. This is the same for every fit, hence only one copy is stored.
- calibr = a list which is the output of an internal call to tuneLearnFast, which is used for calibrating the learning rate. See ?tuneLearnFast for details.

Author(s)

Matteo Fasiolo <matteo.fasiolo@gmail.com>.

References

Fasiolo, M., Goude, Y., Nedellec, R. and Wood, S. N. (2017). Fast calibrated additive quantile regression. Available at https://arxiv.org/abs/1707.03307.

Examples

```
#####
# Multivariate Gaussian example
####
library(qgam)
set.seed(2)
dat <- gamSim(1, n=300, dist="normal", scale=2)

fit <- mqgam(y~s(x0)+s(x1)+s(x2)+s(x3), data=dat, err = 0.05, qu = c(0.2, 0.8))

invisible( qdo(fit, 0.2, plot, pages = 1) )

#####
# Univariate "car" example
####
## Not run:
library(qgam); library(MASS)

# Fit for quantile 0.8 using the best sigma
quSeq <- c(0.2, 0.4, 0.6, 0.8)
set.seed(6436)
fit <- mqgam(accel~s(times, k=20, bs="ad"), data = mcycle, err = 0.05, qu = quSeq)

# Plot the fit
xSeq <- data.frame(cbind("accel" = rep(0, 1e3), "times" = seq(2, 58, length.out = 1e3)))
plot(mcycle$times, mcycle$accel, xlab = "Times", ylab = "Acceleration", ylim = c(-150, 80))
for(iq in quSeq){
```

```
  pred <- qdo(fit, iq, predict, newdata = xSeq)
  lines(xSeq$times, pred, col = 2)
}

## End(Not run)
```

---

qdo *Manipulating the output of* mqgam

Description

Contrary to qgam, mqgam does not output a standard gamObject, hence methods such as predict.gam or plot.gam cannot be used directly. qdo provides a simple wrapper for such methods.

Usage

```
qdo(obj, qu = NULL, fun = I, ...)
```

Arguments

obj	the output of a mqgam call.
qu	A vector whose elements must be in (0, 1). Each element indicates a quantile of interest, which should be an element of names(obj$fit). If left to NULL the function fun will be applied to each of the quantile fits in obj.
fun	The method or function that we want to use on the gamObject corresponding to quantile qu. For instance predict, plot or summary. By default this is the identity function (I), which means that the fitted model for quantile qu is returned.
...	Additional arguments to be passed to fun.

Value

A list where the i-th entry is the output of fun (whatever that is) corresponding to quantile qu[i].

Author(s)

Matteo Fasiolo <matteo.fasiolo@gmail.com>.

Examples

```
library(qgam); library(MASS)

quSeq <- c(0.4, 0.6)
set.seed(737)
fit <- mqgam(accel~s(times, k=20, bs="ad"), data = mcycle, err = 0.05, qu = quSeq,
          control = list("tol" = 0.01)) # <- semi-sloppy tolerance to speed-up calibration
```

```
qdo(fit, 0.4, summary)
invisible(qdo(fit, 0.4, plot, pages = 1))

# Return the object for qu = 0.6 and then plot it
tmp <- qdo(fit, 0.6)
plot(tmp)
```

---

qgam *Fit a smooth additive quantile regression model*

Description

This function fits a smooth additive regression model for a single quantile.

Usage

```
qgam(form, data, qu, lsig = NULL, err = 0.05,
  multicore = !is.null(cluster), cluster = NULL, ncores = detectCores() -
  1, paropts = list(), control = list(), argGam = NULL)
```

Arguments

`form`	A GAM formula, or a list of formulae. See ?mgcv::gam details.
`data`	A data frame or list containing the model response variable and covariates required by the formula. By default the variables are taken from environment(formula): typically the environment from which gam is called.
`qu`	The quantile of interest. Should be in (0, 1).
`lsig`	The value of the log learning rate used to create the Gibbs posterior. By defauls `lsig=NULL` and this parameter is estimated by posterior calibration described in Fasiolo et al. (2016). Obviously, the function is much faster if the user provides a value.
`err`	An upper bound on the error of the estimated quantile curve. Should be in (0, 1). See Fasiolo et al. (2016) for details.
`multicore`	If TRUE the calibration will happen in parallel.
`cluster`	An object of class `c("SOCKcluster", "cluster")`. This allowes the user to pass her own cluster, which will be used if `multicore == TRUE`. The user has to remember to stop the cluster.
`ncores`	Number of cores used. Relevant if `multicore == TRUE`.
`paropts`	a list of additional options passed into the foreach function when parallel computation is enabled. This is important if (for example) your code relies on external data or packages: use the .export and .packages arguments to supply them so that all cluster nodes have the correct environment set up for computing.
`control`	A list of control parameters. The only one relevant here is `link`, which is the link function used (see `?elf` and `?elflss` for defaults). All other control parameters are used by `tuneLearnFast`. See `?tuneLearnFast` for details.

argGam — A list of parameters to be passed to mgcv::gam. This list can potentially include all the arguments listed in ?gam, with the exception of formula, family and data.

... — additional arguments passed to mgcv::gam.

Value

A gamObject. See ?gamObject.

Author(s)

Matteo Fasiolo <matteo.fasiolo@gmail.com>.

References

Fasiolo, M., Goude, Y., Nedellec, R. and Wood, S. N. (2017). Fast calibrated additive quantile regression. Available at https://arxiv.org/abs/1707.03307.

Examples

```
#####
# Univariate "car" example
####
library(qgam); library(MASS)

# Fit for quantile 0.5 using the best sigma
set.seed(6436)
fit <- qgam(accel~s(times, k=20, bs="ad"), data = mcycle, err = 0.05, qu = 0.5)

# Plot the fit
xSeq <- data.frame(cbind("accel" = rep(0, 1e3), "times" = seq(2, 58, length.out = 1e3)))
pred <- predict(fit, newdata = xSeq, se=TRUE)
plot(mcycle$times, mcycle$accel, xlab = "Times", ylab = "Acceleration", ylim = c(-150, 80))
lines(xSeq$times, pred$fit, lwd = 1)
lines(xSeq$times, pred$fit + 2*pred$se.fit, lwd = 1, col = 2)
lines(xSeq$times, pred$fit - 2*pred$se.fit, lwd = 1, col = 2)

#####
# Multivariate Gaussian example
####
library(qgam)
set.seed(2)
dat <- gamSim(1,n=400,dist="normal",scale=2)

fit <- qgam(y~s(x0)+s(x1)+s(x2)+s(x3), data=dat, err = 0.05, qu = 0.5)
plot(fit, scale = FALSE, pages = 1)

######
# Heteroscedastic example
######
## Not run:
set.seed(651)
```

```
n <- 5000
x <- seq(-4, 3, length.out = n)
X <- cbind(1, x, x^2)
beta <- c(0, 1, 1)
sigma =  1.2 + sin(2*x)
f <- drop(X %*% beta)
dat <- f + rnorm(n, 0, sigma)
dataf <- data.frame(cbind(dat, x))
names(dataf) <- c("y", "x")

fit <- qgam(list(y~s(x, k = 30, bs = "cr"), ~ s(x, k = 30, bs = "cr")),
            data = dataf, qu = 0.95, lsig = -1.16)

plot(x, dat, col = "grey", ylab = "y")
tmp <- predict(fit, se = TRUE)
lines(x, tmp$fit[ , 1])
lines(x, tmp$fit[ , 1] + 3 * tmp$se.fit[ , 1], col = 2)
lines(x, tmp$fit[ , 1] - 3 * tmp$se.fit[ , 1], col = 2)

## End(Not run)
```

---

sigmoid *Sigmoid function and its derivatives*

Description

Calculates the sigmoid function and its derivatives.

Usage

```
sigmoid(y, deriv = FALSE)
```

Arguments

y	a numeric vector.
deriv	if TRUE alse the first three derivatives of the sigmoid function will be computed.

Value

If deriv==FALSE, it returns a numeric vector equal to 1/(1+exp(-x)). If deriv==TRUE it returns a list where the slot $D0 contains 1/(1+exp(-x)), while $D1, $D2 and $D3 contain its first three derivatives.

Author(s)

Matteo Fasiolo <matteo.fasiolo@gmail.com>.

Examples

```
library(qgam)
set.seed(90)
h <- 1e-6
p <- rnorm(1e4, 0, 1e6)
sigmoid(p[1:50]) - 1/(1+exp(-p[1:50]))

##### Testing sigmoid derivatives
e1 <- abs((sigmoid(p+h) - sigmoid(p-h)) / (2*h) - sigmoid(p, TRUE)[["D1"]]) / (2*h)
e2 <- abs((sigmoid(p+h, TRUE)$D1 - sigmoid(p-h, TRUE)$D1) /
      (2*h) - sigmoid(p, TRUE)[["D2"]]) / (2*h)
e3 <- abs((sigmoid(p+h, TRUE)$D2 - sigmoid(p-h, TRUE)$D2) /
      (2*h) - sigmoid(p, TRUE)[["D3"]]) / (2*h)

if( any(c(e1, e2, e3) > 1) ) stop("Sigmoid derivatives are not estimated accurately")
```

---

tuneLearn — *Tuning the learning rate for Gibbs posterior*

Description

The learning rate (sigma) of the Gibbs posterior is tuned either by calibrating the credible intervals for the fitted curve, or by minimizing the pinball loss on out-of-sample data. This is done by bootrapping or by k-fold cross-validation. Here the calibration loss function is evaluated on a grid of values provided by the user.

Usage

```
tuneLearn(form, data, lsig, qu, err = 0.05, multicore = !is.null(cluster),
  cluster = NULL, ncores = detectCores() - 1, paropts = list(),
  control = list(), argGam = NULL)
```

Arguments

`form`	A GAM formula, or a list of formulae. See ?mgcv::gam details.
`data`	A data frame or list containing the model response variable and covariates required by the formula. By default the variables are taken from environment(formula): typically the environment from which gam is called.
`lsig`	A vector of value of the log learning rate (log(sigma)) over which the calibration loss function is evaluated.
`qu`	The quantile of interest. Should be in (0, 1).
`err`	An upper bound on the error of the estimated quantile curve. Should be in (0, 1). See Fasiolo et al. (2016) for details.
`multicore`	If TRUE the calibration will happen in parallel.

cluster — An object of class c("SOCKcluster", "cluster"). This allowes the user to pass her own cluster, which will be used if multicore == TRUE. The user has to remember to stop the cluster.

ncores — Number of cores used. Relevant if multicore == TRUE.

paropts — a list of additional options passed into the foreach function when parallel computation is enabled. This is important if (for example) your code relies on external data or packages: use the .export and .packages arguments to supply them so that all cluster nodes have the correct environment set up for computing.

control — A list of control parameters for tuneLearn with entries:

- loss = loss function use to tune log(sigma). If loss=="cal" is chosen, then log(sigma) is chosen so that credible intervals for the fitted curve are calibrated. See Fasiolo et al. (2016) for details. If loss=="pin" then log(sigma) approximately minimizes the pinball loss on the out-of-sample data.
- sam = sampling scheme use: sam=="boot" corresponds to bootstrapping and sam=="kfold" to k-fold cross-validation. The second option can be used only if ctrl$loss=="pin".
- K = if sam=="boot" this is the number of boostrap datasets, while if sam=="kfold" this is the number of folds. By default K=50.
- b = offset parameter used by the mgcv::gauslss. By default b=0.
- vtype = type of variance estimator used to standardize the deviation from the main fit in the calibration. If set to "m" the variance estimate obtained by the full data fit is used, if set to "b" than the variance estimated produced by the bootstrap fits are used. By default vtype="m".
- epsB = positive tolerance used to assess convergence when fitting the regression coefficients on bootstrap data. In particular, if |dev-dev_old|/(|dev|+0.1)<epsB then convergence is achieved. Default is epsB=1e-5.
- verbose = if TRUE some more details are given. By default verbose=FALSE.
- link = link function to be used. See ?elf and ?elflss for defaults.
- progress = argument passed to plyr::llply. By default progress="text" so that progress is reported. Set it to "none" to avoid it.

argGam — A list of parameters to be passed to mgcv::gam. This list can potentially include all the arguments listed in ?gam, with the exception of formula, family and data.

Value

A list with entries:

- lsig = the value of log(sigma) resulting in the lowest loss.
- loss = vector containing the value of the calibration loss function corresponding to each value of log(sigma).
- edf = a matrix where the first colums contain the log(sigma) sequence, and the remaining columns contain the corresponding effective degrees of freedom of each smooth.
- convProb = a logical vector indicating, for each value of log(sigma), whether the outer optimization which estimates the smoothing parameters has encountered convergence issues. FALSE means no problem.

Author(s)

Matteo Fasiolo <matteo.fasiolo@gmail.com>.

References

Fasiolo, M., Goude, Y., Nedellec, R. and Wood, S. N. (2017). Fast calibrated additive quantile regression. Available at https://arxiv.org/abs/1707.03307.

Examples

```
library(qgam); library(MASS)

# Calibrate learning rate on a grid
set.seed(41444)
sigSeq <- seq(1.5, 5, length.out = 10)
closs <- tuneLearn(form = accel~s(times,k=20,bs="ad"),
                   data = mcycle,
                   err = 0.05,
                   lsig = sigSeq,
                   qu = 0.5)

plot(sigSeq, closs$loss, type = "b", ylab = "Calibration Loss", xlab = "log(sigma)")

# Pick best log-sigma
best <- sigSeq[ which.min(closs$loss) ]
abline(v = best, lty = 2)

# Fit using the best sigma
fit <- qgam(accel~s(times,k=20,bs="ad"), data = mcycle, qu = 0.5, err = 0.05, lsig = best)
summary(fit)

pred <- predict(fit, se=TRUE)
plot(mcycle$times, mcycle$accel, xlab = "Times", ylab = "Acceleration",
     ylim = c(-150, 80))
lines(mcycle$times, pred$fit, lwd = 1)
lines(mcycle$times, pred$fit + 2*pred$se.fit, lwd = 1, col = 2)
lines(mcycle$times, pred$fit - 2*pred$se.fit, lwd = 1, col = 2)
```

`tuneLearnFast` *Fast learning rate calibration for the Gibbs posterior*

Description

The learning rate (sigma) of the Gibbs posterior is tuned either by calibrating the credible intervals for the fitted curve, or by minimizing the pinball loss on out-of-sample data. This is done by bootrapping or by k-fold cross-validation. Here the loss function is minimized, for each quantile, using a Brent search.

Usage

```
tuneLearnFast(form, data, qu, err = 0.05, multicore = !is.null(cluster),
  cluster = NULL, ncores = detectCores() - 1, paropts = list(),
  control = list(), argGam = NULL)
```

Arguments

form
: A GAM formula, or a list of formulae. See ?mgcv::gam details.

data
: A data frame or list containing the model response variable and covariates required by the formula. By default the variables are taken from environment(formula): typically the environment from which gam is called.

qu
: The quantile of interest. Should be in (0, 1).

err
: An upper bound on the error of the estimated quantile curve. Should be in (0, 1). If it is a vector, it should be of the same length of `qu`. See Fasiolo et al. (2016) for details.

multicore
: If TRUE the calibration will happen in parallel.

cluster
: An object of class `c("SOCKcluster", "cluster")`. This allowes the user to pass her own cluster, which will be used if `multicore == TRUE`. The user has to remember to stop the cluster.

ncores
: Number of cores used. Relevant if `multicore == TRUE`.

paropts
: a list of additional options passed into the foreach function when parallel computation is enabled. This is important if (for example) your code relies on external data or packages: use the .export and .packages arguments to supply them so that all cluster nodes have the correct environment set up for computing.

control
: A list of control parameters for `tuneLearn` with entries:

  - `loss` = loss function use to tune log(sigma). If `loss=="cal"` is chosen, then log(sigma) is chosen so that credible intervals for the fitted curve are calibrated. See Fasiolo et al. (2016) for details. If `loss=="pin"` then log(sigma) approximately minimizes the pinball loss on the out-of-sample data.
  - `sam` = sampling scheme use: `sam=="boot"` corresponds to bootstrapping and `sam=="kfold"` to k-fold cross-validation. The second option can be used only if `ctrl$loss=="pin"`.
  - `vtype` = type of variance estimator used to standardize the deviation from the main fit in the calibration. If set to `"m"` the variance estimate obtained by the full data fit is used, if set to `"b"` than the variance estimated produced by the bootstrap fits are used. By default `vtype="m"`.
  - `epsB` = positive tolerance used to assess convergence when fitting the regression coefficients on bootstrap data. In particular, if `|dev-dev_old|/(|dev|+0.1)<epsB` then convergence is achieved. Default is `epsB=1e-5`.
  - `K` = if `sam=="boot"` this is the number of boostrap datasets, while if `sam=="kfold"` this is the number of folds. By default `K=50`.
  - `init` = an initial value for the log learning rate (log(sigma)). By default `init=NULL` and the optimization is initialized by other means.

- `brac` = initial bracket for Brent method. By default `brac=log(c(0.5, 2))`, so the initial search range is `(init - log(0.5), init + log(2))`.
- `tol` = tolerance used in the Brent search. By default `tol=.Machine$double.eps^0.25`. See `?optimize` for details.
- `aTol` = Brent search parameter. If the solution to a Brent get closer than `aTol * abs(diff(brac))` to one of the extremes of the bracket, the optimization is stop and restarted with an enlarged and shifted bracket. `aTol=0.05` should be > 0 and values > 0.1 don't quite make sense. By default `aTol=0.05`.
- `redWd` = parameter which determines when the bracket will be reduced. If `redWd==10` then the bracket is halved if the nearest solution falls within the central 10% of the bracket's width. By default `redWd = 10`.
- `b` = offset parameter used by the mgcv::gauslss, which we estimate to initialize the quantile fit (when a variance model is used). By default `b=0`.
- `link` = Link function to be used. See `?elf` and `?elflss` for defaults.
- `verbose` = if TRUE some more details are given. By default `verbose=FALSE`.
- `progress` = if TRUE progress in learning rate estimation is reported via printed text. `TRUE` by default.

`argGam` A list of parameters to be passed to `mgcv::gam`. This list can potentially include all the arguments listed in `?gam`, with the exception of `formula`, `family` and `data`.

Value

A list with entries:

- `lsig` = a vector containing the values of log(sigma) that minimize the loss function, for each quantile.
- `err` = the error bound used for each quantile. Generally each entry is identical to the argument `err`, but in some cases the function increases it to enhance stabily.
- `ranges` = the search ranges by the Brent algorithm to find log-sigma, for each quantile.
- `store` = a list, where the i-th entry is a matrix containing all the locations (1st row) at which the loss function has been evaluated and its value (2nd row), for the i-th quantile.

Author(s)

Matteo Fasiolo <matteo.fasiolo@gmail.com>.

References

Fasiolo, M., Goude, Y., Nedellec, R. and Wood, S. N. (2017). Fast calibrated additive quantile regression. Available at https://arxiv.org/abs/1707.03307.

Examples

```
library(qgam); library(MASS)

###
```

```
# Single quantile fit
###
# Calibrate learning rate on a grid
set.seed(5235)
tun <- tuneLearnFast(form = accel~s(times,k=20,bs="ad"),
                     data = mcycle,
                     err = 0.05,
                     qu = 0.2)

# Fit for quantile 0.2 using the best sigma
fit <- qgam(accel~s(times, k=20, bs="ad"), data = mcycle, qu = 0.2,
            err = 0.05, lsig = tun$lsig)

pred <- predict(fit, se=TRUE)
plot(mcycle$times, mcycle$accel, xlab = "Times", ylab = "Acceleration",
     ylim = c(-150, 80))
lines(mcycle$times, pred$fit, lwd = 1)
lines(mcycle$times, pred$fit + 2*pred$se.fit, lwd = 1, col = 2)
lines(mcycle$times, pred$fit - 2*pred$se.fit, lwd = 1, col = 2)

###
# Multiple quantile fits
###
## Not run:
# Calibrate learning rate on a grid
quSeq <- c(0.25, 0.5, 0.75)
set.seed(5235)
tun <- tuneLearnFast(form = accel~s(times, k=20, bs="ad"),
                     data = mcycle,
                     err = 0.05,
                     qu = quSeq)

# Fit using estimated sigmas
fit <- mqgam(accel~s(times, k=20, bs="ad"), data = mcycle, qu = quSeq,
             err = 0.05, lsig = tun$lsig)

# Plot fitted quantiles
plot(mcycle$times, mcycle$accel, xlab = "Times", ylab = "Acceleration",
     ylim = c(-150, 80))
for(iq in quSeq){
  pred <- qdo(fit, iq, predict)
  lines(mcycle$times, pred, col = 2)
}

## End(Not run)
```

UKload *UK electricity load data*

Description

Dataset on UK electricity demand, taken from the national grid (http://www2.nationalgrid.com/).

Usage

```
data(UKload)
```

Format

`UKload` contains the following variables:

NetDemand net electricity demand between 11:30am and 12am.

wM instantaneous temperature, averaged over several English cities.

wM_s95 exponential smooth of `wM`, that is `wM_s95[i] = a*wM[i] + (1-a)*wM_s95[i]` with `a=0.95`.

Posan periodic index in `[0, 1]` indicating the position along the year.

Dow factor variable indicating the day of the week.

Trend progressive counter, useful for defining the long term trend.

NetDemand.48 lagged version of `NetDemand`, that is `NetDemand.48[i] = NetDemand[i-2]`.

Holy binary variable indicating holidays.

Year should be obvious.

Date should be obvious.

Details

See Fasiolo et al. (2017) for details.

Value

matrix of replicate data series

References

Fasiolo, M., Goude, Y., Nedellec, R. and Wood, S. N. (2017). Fast calibrated additive quantile regression. Available at https://arxiv.org/abs/1707.03307.

Examples

```
library(qgam)
data(UKload)
plot(UKload$NetDemand, type = 'l')
```

Index