Package 'mgcViz'

October 5, 2021

Title Visualisations for Generalized Additive Models

Date 2021-09-30

Version 0.1.9

Description Extension of the 'mgcv' package, providing visual tools for Generalized Additive Models that exploit the additive structure of such models, scale to large data sets and can be used in conjunction with a wide range of response distributions. The focus is providing visual methods for better understanding the model output and for aiding model checking and development beyond simple exponential family regression. The graphical framework is based on the layering system provided by 'ggplot2'.

Depends R (>= 3.4), mgcv (>= 1.8-28), qgam (>= 1.2.3), ggplot2

Imports gamm4, matrixStats, viridis, GGally, KernSmooth, gridExtra, plyr, shiny, miniUI

Suggests rgl, knitr, rmarkdown, testthat, MASS, webshot2

Additional_repositories `https://dmurdoch.github.io/drat`

License GPL-3

Encoding UTF-8

RoxygenNote 7.1.1

VignetteBuilder knitr

URL `https://github.com/mfasiolo/mgcViz`

BugReports `https://github.com/mfasiolo/mgcViz/issues`

NeedsCompilation no

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

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

Repository CRAN

Date/Publication 2021-10-05 07:10:12 UTC

R topics documented:

`ALE` *Generic function for Accumulated Local Effect (ALE)*

Description

Generic function for producing ALE effects, to be plottied using the `plot` generic.

Usage

```
ALE(o, ...)
```

Arguments

`o`	the model we want to use to produce the ALE effect.
`...`	arguments to be passed to methods.

References

Apley, D.W., and Zhu, J, 2016. Visualizing the effects of predictor variables in black box supervised learning models. arXiv preprint arXiv:1612.08468.

See Also

ALE.gam

`ALE.gam` *Create Accumulated Local Effects (ALE) for GAMs*

Description

Create Accumulated Local Effects (ALE) for GAMs

Usage

```
## S3 method for class 'gam'
ALE(o, x, newdata = NULL, type = "link", nbin = 40, oind = 1, center = 1, ...)
```

Arguments

`o`	a fitted GAM model.
`x`	the name of the variable along which we want to produce the ALE effect.
`newdata`	optional argument indicating the data to be used to produce the ALE effect. If `NULL` the data contained in `o` will be used.
`type`	if set to "link" (the default option) the model output will be the linear predictor, if set to "response" the model output is on the scale of the response.
`nbin`	number of intervals into which the predictor range is divided when calculating the ALE effects.
`oind`	relevant only when the model `o` has multiple linear predictors (e.g. for GAMLSS models or for `multinom` regression). `oind` is the index of the output variable used for the ALE effect (i.e., only `predict(o)[ ,oind]`.
`center`	if set to 0 the ALE effect is not centered and the effect is equal to zero at the smallest value on x-grid. If set to 1 (default) the effect is centered as done in Apley and Zhu, 2016. That is, an estimate of the expected value of the uncentered effect is subtracted, so the effect is centered similarly to smooth effects in GAMs. If set to 2, the expected value of the model output at the smallest value on the x-grid is added to the uncentered effect.
`...`	extra arguments that will be passed to `predict` and `vcov`.

Value

An object of class `ALEXD`, where `X` is the number of dimensions, which can be plotted using `plot.ALEXD` (only `X=1` is provided at the moment).

Author(s)

Matteo Fasiolo and Christian Capezza, with some internal code having been adapted from the ALE-Plot package of Dan Apley.

References

Apley, D.W., and Zhu, J, 2016. Visualizing the effects of predictor variables in black box supervised learning models. arXiv preprint arXiv:1612.08468.

See Also

plot.ALE1D

Examples

```
# Example using Tweedie distribution
library(mgcViz)
set.seed(3)
n<-400
## Simulate data...
dat <- gamSim(1,n=n,dist="poisson",scale=.2)
dat$y <- rTweedie(exp(dat$f),p=1.3,phi=.5) ## Tweedie response

## Fit a fixed p Tweedie, with wrong link ...
b <- gam(list(y~s(x0)+s(x1)+s(x2)+s(x3),~1,~1), family=twlss(), data=dat)

plot(ALE(b, "x2", type = "response", oind = 1))
```

---

bamV *Fit a GAM model and get a gamViz object*

Description

These are wrapper that fits a GAM model using mgcv::gam or mgcv::bam and converts it to a `gamViz` object using the getViz function. It is essentially a shortcut.

Usage

```
bamV(
  formula,
  family = gaussian(),
  data = list(),
  method = "fREML",
  aGam = list(),
  aViz = list()
)

gamV(
  formula,
  family = gaussian(),
  data = list(),
  method = "REML",
  aGam = list(),
  aViz = list()
)
```

Arguments

`formula, family, data, method`	same arguments as in mgcv::gam or mgcv::bam.
`aGam`	list of further arguments to be passed to mgcv::gam or mgcv::bam.
`aViz`	list of arguments to be passed to getViz.

Value

An object of class "gamViz" which can, for instance, be plotted using plot.gamViz.

Examples

```
##### gam example
# Simulate data
library(mgcViz)
set.seed(2) ## simulate some data...
dat <- gamSim(1,n=1000,dist="normal",scale=2)

# Fit GAM and get gamViz object
b <- gamV(y~s(x0)+s(x1, x2)+s(x3), data = dat,
          aGam = list(scale = 2), aViz = list("nsim" = 20))

# This is equivalent to doing
# 1. Fit GAM
# b <- gam(y~s(x0)+s(x1, x2)+s(x3), data=dat, method="REML", scale = 2)
# 2. Convert to gamViz object
# b <- getViz(b, nsim = 20)

# Either way, we plot first and third effects by doing
print(plot(b, select = c(1, 3)), pages = 1)

##### bam example
# Simulate data
library(mgcViz)
set.seed(2) ## simulate some data...
dat <- gamSim(1,n=2000,dist="normal",scale=2)
```

```
# Fit using bam() and get gamViz object
b <- bamV(y~s(x0)+s(x1, x2)+s(x3), data = dat,
          aGam = list(discrete = TRUE), aViz = list("nsim" = 0))

# Either way, we plot first and third effects by doing
print(plot(b, select = c(2)), pages = 1)
```

---

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

Description

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

Usage

```
## S3 method for class 'gamViz'
check(
  obj,
  type = c("auto", "deviance", "pearson", "response", "tunif", "tnormal"),
  k.sample = 5000,
  k.rep = 200,
  maxpo = 10000,
  a.qq = list(),
  a.hist = list(),
  a.respoi = list(),
  ...
)
```

Arguments

obj	an object of class gamViz, the output of a getViz() call.
type	type of residuals, see residuals.gamViz, used in all plots.
k.sample	above this k testing uses a random sub-sample of data.
k.rep	how many re-shuffles to do to get p-value for k testing.
maxpo	maximum number of residuals points that will be plotted in the scatter-plots. If number of datapoints > maxpo, then a subsample of maxpo points will be plotted.
a.qq	list of arguments to be passed to qq.gamViz. See qq.gamViz.
a.hist	list of arguments to be passed to ggplot2::geom_histogram.
a.respoi	list of arguments to be passed to ggplot2::geom_point.
...	currently not used.

Details

This is a essentially a re-write of `mgcv::gam.check` using `ggplot2`. See mgcv::gam.check for details.

Value

An object of class `checkGam`, which is simply a list of `ggplot` objects.

Examples

```
library(mgcViz)
set.seed(0)
dat <- gamSim(1, n = 200)
b <- gam(y ~ s(x0) + s(x1) + s(x2) + s(x3), data = dat)
b <- getViz(b)

# Checks using default options
check(b)

# Change some algorithmic and graphical parameters
check(b,
      a.qq = list(method = "tnorm",
                  a.cipoly = list(fill = "light blue")),
      a.respoi = list(size = 0.2),
      a.hist = list(bins = 10))
```

`check0D` *Checking GAM simulated residuals or responses*

Description

This function extracts the residuals or responses of a fitted GAM model, then it compares their distribution with that of model-based simulations.

Usage

```
check0D(
  o,
  type = "auto",
  maxpo = 10000,
  na.rm = TRUE,
  trans = NULL,
  useSim = TRUE
)
```

Arguments

`o`	an object of class `gamViz`.
`type`	the type of residuals to be used. See residuals.gamViz. If `"type == y"` then the raw observations will be used.
`maxpo`	maximum number of residuals points that will be used by layers such as `l_rug()`. If number of datapoints > `maxpo`, then a subsample of `maxpo` points will be taken.
`na.rm`	if `TRUE` missing cases in `x` or `y` will be dropped out.
`trans`	function used to transform the observed and simulated residuals or responses. It must take a vector of as input, and it must either a vector of the same length or a scalar.
`useSim`	if `FALSE` then the simulated responses contained in object `o` will not be used by this function or by any of the layers that can be used with its output.

Value

An object of class `c("plotSmooth","gg")`.

Examples

```
# The variance of the response distribution changes along x2
library(mgcViz)
n  <- 400
x1 <- runif(n, -1, 1)
x2 <- runif(n, -1, 1)
dat <- data.frame("x1" = x1, "x2" = x2,
                  "y" = sin(3*x1) + 0.5 * x2^2 + pmax(x2, 0.2) * rnorm(n))

# Fit model with constant variance and perform posterior simulations (post = TRUE)
# which take into account smoothing parameter uncertainty (unconditional = TRUE)
b <- gamV(y ~ s(x1)+s(x2), data = dat,
          aViz = list(nsim = 50, post = TRUE, unconditional = TRUE))

# Histogram of simulated vs observed residuals: the latter are fat tailed
check0D(b) + l_hist() + l_rug()

# Histogram of simulated 4th central moment (~ kurtosis) of simulated residuals.
# The vertical line is the 4th moment of the observed residuals
check0D(b, trans = function(.y) mean((.y - mean(.y))^4)) + l_dens1D() + l_vline() + l_rug()
# Residuals look very fat tails, but the real problem here is the heteroscedasticity
# which can be diagnosted using check1D(b, "x2") + l_gridCheck1D(sd)
```

`check1D` *Checking GAM residuals or responses along one covariate*

Description

This function extracts the residuals of a fitted GAM model, and orders them according to the value of a single covariate. Then several visual residuals diagnostics can be plotted by adding layers.

Usage

```
check1D(
  o,
  x,
  type = "auto",
  maxpo = 10000,
  na.rm = TRUE,
  trans = NULL,
  useSim = TRUE
)
```

Arguments

`o`	an object of class `gamViz`.
`x`	it can be either a) a single character, b) a numeric vector or c) a list of characters. In case a) it should be the name of one of the variables in the dataframe used to fit `o`. In case b) its length should be equal to the length of `o$y`. In case c) it should be a list of names variables in the dataframe used to fit `o`.
`type`	the type of residuals to be used. See residuals.gamViz. If `"type == y"` then the raw observations will be used.
`maxpo`	maximum number of residuals points that will be used by layers such as `l_rug()`. If number of datapoints > `maxpo`, then a subsample of `maxpo` points will be taken.
`na.rm`	if `TRUE` missing cases in `x` or `y` will be dropped out.
`trans`	function used to transform the observed and simulated residuals or responses. It must take a vector of as input, and must return a vector of the same length.
`useSim`	if `FALSE` then the simulated responses contained in object `o` will not be used by this function or by any of the layers that can be used with its output.

Value

The function will return an object of class `c("plotSmooth","gg")`, unless argument `x` is a list. In that case the function will return an object of class `c("plotGam","gg")` containing a checking plot for each variable.

Examples

```
### Example 1: diagnosing heteroscedasticity
library(mgcViz);
set.seed(4124)
n <- 1e4
x <- rnorm(n); y <- rnorm(n);

# Residuals are heteroscedastic w.r.t. x
ob <- (x)^2 + (y)^2 + (0.2*abs(x) + 1)  * rnorm(n)
b <- bam(ob ~ s(x,k=30) + s(y, k=30), discrete = TRUE)
b <- getViz(b)

# Look at residuals along "x"
```

```
ck <- check1D(b, "x", type = "tnormal")

# Can't see that much
ck + l_dens(type = "cond", alpha = 0.8) + l_points() + l_rug(alpha = 0.2)

# Some evidence of heteroscedasticity
ck + l_densCheck()

# Compare observed residuals std dev with that of simulated data,
# heteroscedasticity is clearly visible
b <- getViz(b, nsim = 50)
check1D(b, "x") + l_gridCheck1D(gridFun = sd, showReps = TRUE)

# This also works with factor or logical data
fac <- sample(letters, n, replace = TRUE)
logi <- sample(c(TRUE, FALSE), n, replace = TRUE)
b <- bam(ob ~ s(x,k=30) + s(y, k=30) + fac + logi, discrete = TRUE)
b <- getViz(b, nsim = 50)

# Look along "fac"
ck <- check1D(b, "fac")
ck + l_points() + l_rug()
ck + l_gridCheck1D(gridFun = sd)

# Look along "logi"
ck <- check1D(b, "logi")
ck + l_points() + l_rug()
ck + l_gridCheck1D(gridFun = sd)
```

`check2D` *Checking GAM residuals along two covariates*

Description

This function extracts the residuals of a fitted GAM model, and plots them according to the values of two covariates. Then several visual residuals diagnostics can be plotted by adding layers.

Usage

```
check2D(
  o,
  x1,
  x2,
  type = "auto",
  maxpo = 10000,
  na.rm = TRUE,
  trans = NULL,
  useSim = TRUE
)
```

Arguments

o	an object of class gamViz.
x1	it can be either a) a single character, b) a numeric vector or c) a list of characters. In case a) it should be the name of one of the variables in the dataframe used to fit o. In case b) its length should be equal to the length of o$y. In case c) it should be a list of names of variables in the dataframe used to fit o.
x2	same as x2, but this will appear on the y-axis.
type	the type of residuals to be used. See residuals.gamViz. If "type == y" then the raw observations will be used.
maxpo	maximum number of residuals points that will be used by layers such as l_rug(). If number of datapoints > maxpo, then a subsample of maxpo points will be taken.
na.rm	if TRUE missing cases in x or y will be dropped out
trans	function used to transform the observed and simulated residuals or responses. It must take a vector of as input, and must return a vector of the same length.
useSim	if FALSE then the simulated responses contained in object o will not be used by this function or by any of the layers that can be used with its output.

Value

The function will return an object of class c("plotSmooth","gg"), unless arguments x1 and/or x2 are lists. If they are lists of the same length, then the function will return an object of class c("plotGam","gg") containing a checking plot for each pair of variables. If x1 is a list and x2 is not specified, the function will return an object of class c("plotGam","gg") containing a plot for each unique combination of the variables in x1.

Examples

```
## Not run:
library(mgcViz);
#### Example 1: Rosenbrock function
# Simulate data
n <- 1e4
X <- data.frame("x1"=rnorm(n, 0.5, 0.5), "x2"=rnorm(n, 1.5, 1))
X$y <- (1-X$x1)^2 + 100*(X$x2 - X$x1^2)^2 + rnorm(n, 0, 2)
b <- bam(y ~ te(x1, x2, k = 5), data = X, discrete = TRUE)
b <- getViz(b, nsim = 50)

# Plot joint density of observed covariate x1 and x2
check2D(b, x1 = "x1", x2 = "x2") + l_rug() + l_dens(type="joint", alpha=0.6) + l_points()

# Look at how mean of residuals varies across x1 and x2
check2D(b, x1 = "x1", x2 = "x2") + l_gridCheck2D() + l_points()

# Can't see much in previous plot, let's zoom in central area, where most
# data is. Here we can clearly see that the mean model is mispecified
check2D(b, x1 = "x1", x2 = "x2") + l_gridCheck2D(bw = c(0.05, 0.1)) +
                                   xlim(-1, 1) + ylim(0, 3)
# Fit can be improved by increasing k in the bam() call
```

```
#### Example 2: checking along factor variables
# Simulate data where variance changes along factor variable "fac"
n <- 1e4
X <- data.frame("x1"=rnorm(n, 0.5, 0.5), "x2"=rnorm(n, 1.5, 1))
X$fac <- as.factor( sample(letters, n, replace = TRUE) )
X$fac2 <- as.factor( sample(c("F1", "F2", "F3", "F4", "F5"), n, replace = TRUE) )
X$y <- (1-X$x1)^2 + 5*(X$x2 - X$x1^2)^2 + 0.1*as.numeric(X$fac) * rnorm(n, 0, 2)
b <- bam(y ~ te(x1, x2, k = 5) + fac + fac2, data = X, discrete = TRUE)
b <- getViz(b, nsim = 50)

# Check standard deviation of residuals along covariates "x1" and "fac"
a <- check2D(b, x1 = "x2", x2 = "fac")
a + l_gridCheck2D(gridFun = sd) + l_rug() + l_points()

# Points and rug are jittered by default, but we can over-ride this
a + l_rug(position = position_jitter(width = 0, height = 0)) +
  l_points(position = position_jitter(width = 0, height = 0))

# Check standard deviation of residuals along the two factor variables
a <- check2D(b, x1 = "fac", x2 = "fac2")
a + l_gridCheck2D(gridFun = sd, bw = c(1, 4)) + l_rug() + l_points()

## End(Not run)
```

fix.family.cdf *Getting the CDF of a gam family*

Description

Some methods implemented in mgcViz require the c.d.f. of the response distribution. This function takes a family object as input and returns the same object, but with the cdf function added to the $cdf slot. Mainly for internal use.

Usage

```
fix.family.cdf(fam)
```

Arguments

fam	an object of class family.

gamm4V *Fit a GAMM or GAMM4 model and get a gamViz object*

Description

These are wrappers that fit GAM models using mgcv::gamm or gamm4::gamm4 and convert them to a `gamViz` object using the getViz function. It is essentially a shortcut.

Usage

```
gamm4V(
  formula,
  random = NULL,
  family = gaussian(),
  data = list(),
  REML = TRUE,
  aGam = list(),
  aViz = list(),
  keepGAMObj = FALSE
)

gammV(
  formula,
  random = NULL,
  family = gaussian(),
  data = list(),
  method = "REML",
  aGam = list(),
  aViz = list(),
  keepGAMObj = FALSE
)
```

Arguments

`formula, random, family, data`	same arguments as in mgcv::gamm or gamm4::gamm4.
`REML`	same as in gamm4::gamm4
`aGam`	list of further arguments to be passed to mgcv::gamm or gamm4::gamm4.
`aViz`	list of arguments to be passed to getViz.
`keepGAMObj`	if `TRUE` a copy of the gamViz Object is kept under $gam to assure compatibility with mgcv::gamm and gamm4::gamm4. Defaults to `FALSE`.
`method`	same as in mgcv::gamm

Details

WARNING: Model comparisons (e.g. with `anova`) should only be done using the mixed model part as described in gamm4::gamm4. For mgcv::gamm please refer to the original help file.

Value

An object of class "gamViz" which can, for instance, be plotted using plot.gamViz. Also the object has the following additional elements:

- `lme` mixed model as in mgcv::gamm
- `mer` mixed model as in gamm4::gamm4
- `gam` a copy of the gamViz Object if setting `keepGAMObj = TRUE`.

Examples

```
##### gam example
library(mgcViz)
# Simulate data
dat <- gamSim(1,n=400,scale=2) ## simulate 4 term additive truth
## Now add 20 level random effect `fac'...
dat$fac <- fac <- as.factor(sample(1:20,400,replace=TRUE))
dat$y <- dat$y + model.matrix(~fac-1) %*% rnorm(20) * 0.5

br <- gammV(y~s(x0)+x1+s(x2), data=dat,random=list(fac=~1))
summary(br)
plot(br)

summary(br$lme)

## Not run:
## gamm4::gamm4 example
br4 <- gamm4V(y~s(x0)+x1+s(x2),data=dat,random=~(1|fac))
summary(br4)
plot(br4)

summary(br4$mer)

## End(Not run)
```

`getGam` *Convert gamViz object to gamObject*

Description

Function for converting a `gamViz` object to a `gamObject`. It is essentially the inverse of the getViz function.

Usage

```
getGam(o)
```

Arguments

o	a gamViz object, the output of getViz.

Examples

```
library(mgcViz)
set.seed(2) ## simulate some data...
dat <- gamSim(1,n=1000,dist="normal",scale=2)
b <- gam(y~s(x0)+s(x1, x2)+s(x3), data=dat, method="REML")
a <- getViz(b)
identical(b, getGam(a)) # Must be TRUE
```

---

getViz *Converting gam objects to gamViz objects*

Description

This function converts gam objects into gamViz objects, for which mgcViz provides several plotting methods.

Usage

```
getViz(o, nsim = 0, post = FALSE, newdata, ...)
```

Arguments

o	an object of class gam.
nsim	the number of simulated vectors of responses. A positive integer.
post	if TRUE then posterior simulation is performed. That is, we simulate nsim vectors of regression coefficients from a Gaussian approximation to the posterior, and then we simulate a vector of response using each parameter vector. If FALSE, then nsim vectors of responses are simulated using parameters fixed at the posterior mode.
newdata	Optional new data frame used to perform the simulations. To be passed to predict.gam and, if post == TRUE, to postSim.
...	extra arguments to be passed to simulate.gam (if post==FALSE) or postSim (if post==TRUE). For instance, we could pass prior weights w and offset.

Value

An object of class gamViz.

Examples

```
library(mgcViz)
set.seed(2) ## simulate some data...
dat <- gamSim(1,n=1000,dist="normal",scale=2)
b <- gam(y~s(x0)+s(x1, x2)+s(x3), data=dat, method="REML")
b <- getViz(b, nsim = 20)
str(b$store$sim) # Simulated responses now stored here

plot(sm(b,1)) + l_fitLine() + l_ciLine() + l_rug() + l_points()
plot(sm(b,2)) + l_rug() + l_fitRaster() + l_fitContour()
```

---

gridPrint *Plotting plotSmooth objects on a grid*

Description

This is a wrapper for `gridExtra::grid.arrange`, which allows to plot several `plotSmooth` objects on a grid.

Usage

```
gridPrint(...)
```

Arguments

`...`	arguments to be passed to `gridExtra::grid.arrange`.

Details

This function simply extracts the `ggplot` objects contained in any object of class `plotSmooth` and passes them to `gridExtra::grid.arrange`.

Examples

```
library(mgcViz)
n  <- 1e3
x1 <- rnorm(n)
x2 <- rnorm(n)
dat <- data.frame("x1" = x1, "x2" = x2,
                  "y" = sin(x1) + 0.5 * x2^2 + pmax(x2, 0.2) * rnorm(n))
b <- bam(y ~ s(x1)+s(x2), data = dat, method = "fREML", discrete = TRUE)
b <- getViz(b)

o1 <- plot( sm(b, 1) ) + l_fitLine() + l_ciLine()
o2 <- plot( sm(b, 2) ) + l_fitLine() + l_ciLine()
qpl <- qq(b)

# All on one page, method 1:
gridPrint(o1, o2, qpl, ncol = 2)
```

```
# All on one page, method 2:
gridPrint(grobs = list(o1, o2, qpl), ncol = 2)

# Works also when some ggplot objects are present
gridPrint(o1, o2, qpl, ggplot(), ncol = 2)
```

listLayers *Lists available layers for plotSmooth objects*

Description

This function takes as input an object of class `plotSmooth` and returns the names of all the possible visual layers that could be used with that object.

Usage

```
listLayers(o)
```

Arguments

o	an object of class `plotSmooth`.

Value

A vector containing the names of the available layers.

Examples

```
library(mgcViz)
n  <- 400
x1 <- rnorm(n)
x2 <- rnorm(n)
dat <- data.frame("x1" = x1, "x2" = x2,
                  "y" = sin(x1) + 0.5 * x2^2 + rnorm(n))
b <- gam(y ~ x1+s(x2), data = dat, method = "REML")
b <- getViz(b)

# List layers available for parametric effect plot
o <- plot( pterm(b, 1) )
listLayers(o)

# List layers available for smooth effect plot
o <- plot( sm(b, 1) )
listLayers(o)

# List layers available for checking plot
o <- check1D(b, x1)
listLayers(o)
```

`l_bound` *Add boundaries to smooth effect plot*

Description

This layer adds boundaries to a smooth effect plot.

Usage

```
l_bound(n = 200, ...)
```

Arguments

`n`	number of discrete intervals along the boundary.
`...`	graphical arguments to be passed to `ggplot2::geom_path`.

Value

An object of class `gamLayer`.

See Also

plot.sos.smooth

`l_ciBar` *Adding confidence intervals to barplots*

Description

This layer adds confidence intervals to barplots, such as those produced by factor effects GAM.

Usage

```
l_ciBar(level = 0.95, mul = NULL, ...)
```

Arguments

`level`	the level of the confidence intervals (e.g. 0.9 means 90% intervals).
`mul`	number multiplied by the standard errors when calculating standard error curves. By default `NULL`, if set to a positive number it will over-ride `level`.
`...`	graphical arguments to be passed to `ggplot2::geom_errorbar`.

Value

An object of class `gamLayer`.

See Also

See plot.ptermFactor for examples.

`l_ciLine` *Adding confidence intervals to effect plot*

Description

This layer adds confidence interval lines to smooth, random or parametric effect plots.

Usage

```
l_ciLine(level = 0.95, mul = NULL, ...)
```

Arguments

`level`	coverage level (e.g. 0.9 means 90% intervals). Should be in (0, 1).
`mul`	number multiplied by the standard errors when calculating standard error curves. By default `NULL`, if set to a positive number it will over-ride `level`.
`...`	graphical arguments to be passed to `ggplot2::geom_line`.

Value

An object of class `gamLayer`.

See Also

See plot.mgcv.smooth.1D, plot.ptermNumeric or plot.random.effect for examples.

`l_ciPoly` *Adding confidence band to effect plots*

Description

This layer adds a polygon representing the confidence band of a smooth, random or parametric effect plots.

Usage

```
l_ciPoly(level = 0.95, mul = NULL, ...)
```

Arguments

`level`	coverage level (e.g. 0.9 means 90% intervals). Should be in (0, 1).
`mul`	number multiplied by the standard errors when calculating standard error curves. By default `NULL`, if set to a positive number it will over-ride `level`.
`...`	graphical arguments to be passed to `ggplot2::geom_polygon`.

Value

An object of class `gamLayer`

See Also

See plot.mgcv.smooth.1D, plot.ptermNumeric or plot.random.effect for examples.

`l_clusterLine` *Cluster and plot smooth effects*

Description

This layers clusters several smooth effects and plots the cluster centers.

Usage

```
l_clusterLine(centers, cluFun = kmeans, a.clu = list(), ...)
```

Arguments

`centers`	the number of clusters. This is the same a the `centers` argument in stats::kmeans.
`cluFun`	the function used for clustering. I must take (at least) arguments `x`, `centers` and `data`, which have the same interpretation as in stats::kmeans (which is the default).
`a.clu`	list of further argument to be passed to `cluFun`.
`...`	graphical arguments to be passed to `ggplot2::geom_line`.

Value

An object of class `gamLayer`.

See Also

See plot.fs.interaction.1D for examples.

`l_coordContour` *Adding coordinate lines*

Description

This layers adds coordinate contours to smooth effect plots. It is mainly useful for smooth-on-the-sphere plots.

Usage

```
l_coordContour(brLO = c(-9:9 * 20), brLA = c(-8:8 * 10), ...)
```

Arguments

`brLO`	a vector of meridians to be plotted.
`brLA`	a vector of parallels to be plotted.
`...`	graphical arguments to be passed to `ggplot2::geom_contour`.

Value

An object of class `gamLayer`.

See Also

See plot.sos.smooth for examples.

`l_dens1D` *Adding density estimate to a plot*

Description

This layer adds a density estimate to a plot. It is mainly a wrapper around ggplot2::geom_density.

Usage

```
l_dens1D(...)
```

Arguments

`...`	graphical arguments to be passed to `ggplot2::geom_density`.

Value

An object of class `gamLayer`.

See Also

See check0D for examples.

`l_dens2D` *Adding density estimate heatmap*

Description

This layer adds a 2D density estimate heat-map to a plot. For 1D effect plots, it adds either the conditional density of the partial residuals, `p(r|x)`, or the joint density `p(r,x)`. For 2D effect plots it adds either `p(x1|x2)` or `p(x1,x2)`, where `x1` and `x2` are the relevant covariates.

Usage

```
l_dens2D(type, n = c(50, 50), bw = NULL, tol = 1e-06, trans = sqrt, ...)

l_dens(type, n = c(50, 50), bw = NULL, tol = 1e-06, trans = sqrt, ...)
```

Arguments

`type`	for 1D effect plots, if set to "cond" then the conditional residual density `p(r|x)` is plotted. If set to "joint" the joint density of residuals, `p(r,x)`, is plotted. The behaviour is similar for 2D effect plots, but `r` indicates the second covariate, not the residuals.
`n`	vector of two positive integers, indicating the number of grid points at which the density is evaluated on the x and y axes.
`bw`	vector with two positive entries, indicating the bandwidth to be used by the kernel density estimator of `p(x1,x2)` along x1 and x2.
`tol`	small positive numerical tolerance. The estimated density at a certain location is set to `NA` (hence it will appear white) when it falls below `tol/sqrt(2*pi*sig)`, where `sig` is the standard deviation of the residuals. Set `tol` to -1 plot the density on the whole x-y plane, no matter how low it is.
`trans`	the density on x-y is transformed using this function before being plotted.
`...`	graphical arguments to be passed to `ggplot2::geom_raster`.

Details

The density function is estimated using the fast binned kernel density estimation methods provided by the `KernSmooth` package, hence this function should be able to handle relatively large datasets (~ 10^6 observations).

Value

An object of class `gamLayer`.

See Also

See plot.mgcv.smooth.1D, plot.mgcv.smooth.2D and check1D for examples.

`l_densCheck` *Checking residuals conditional density*

Description

This layer calculates and plots how the empirical conditional density of the residuals, r, differs from its theoretical or model-based counterpart, along a covariate, x.

Usage

```
l_densCheck(n = c(80, 80), bw = NULL, tol = 1e-06, dFun = NULL, ...)
```

Arguments

`n`	vector of two positive integers, indicating the number of grid points at which the density is evaluated on the x and r axes.
`bw`	vector with two positive entries, indicating the bandwidth to be used by the kernel density estimator of `p(r|x)` along x and r.
`tol`	small positive numerical tolerance. The estimated density at a certain location is set to `NA` (hence it will appear white) when it falls below `tol/sqrt(2*pi*sig)`, where `sig` is the standard deviation of the residuals. Set `tol` to -1 plot the density on the whole x-y plane, no matter how low it is.
`dFun`	function used to compute the difference between the empirical (em) and theoretical (th) conditional density of the residuals. By default it is `(sqrt(em)-sqrt(th))^(1/3)`, where `th` is computed using either a uniform or a normal density, depending on the type of residuals used in the check1D call. It should have as arguments three vectors: `.ed` (the empirical conditional density), `.gr` (the points along the y-axis where the density is evaluated) and `.y` (the residuals).
`...`	graphical arguments to be passed to `ggplot2::geom_raster`.

Details

This layer is mainly meant to work together with the check1D function. If check1D() is called with residual type == "tunif" or "tnormal", then `l_densCheck` compares the conditional distribution of the residuals with Unif(0, 1) or N(0, 1). By changing the distance function `dFun` one could of course change both the distance metric and the reference distribution (see Examples below).

WARNING: if check1D() is called with type != "tunif" or "tnormal", then the default distance used by l_densCheck is
```
dFun <-function(.ed,.gr,.y) {
d <-dnorm(.gr,0,sd=sd(.y)) # sd=sd(.y) !!!
d <-sqrt(.ed)-sqrt(d)
return(sign(d)*abs(d)^(1/3))
}
```
so the residuals are standardized using their own std dev `sd(.y)`. Hence `l_densCheck` might not

detect that the mean estimated variance under the fitted model is different from the residuals variance. Hence it is safer to use residual types "tunif" or "tnormal", or a customized distance function dFun (see below for an example on how to do this).

Value

An object of class `gamLayer`.

Examples

```
library(mgcViz);
# Dataset where variance increases linearly with x2, for x2 > 0.2
n <- 1e3
x1 <- rnorm(1e3)
x2 <- rnorm(1e3)
dat <- data.frame("x1"=x1,
                  "x2"=x2, "y"=sin(x1) + 0.5*x2^2 + pmax(x2, 0.2)*rnorm(n))
b <- gam(y ~ s(x1)+s(x2), data=dat)
b <- getViz(b)

# (Red) Blue indicates area where the empirical density
# of the residuals is (lower) higher than it should be under
# the model (residuals should be N(0, sigma) here).
# Here there are clear signs of heteroscedasticity:
# the conditional variance is is increasing for x2 > 0.2.
check1D(b, "x2", type = "tnormal") + l_densCheck() + l_rug()

# Suppose we want to compare the conditional density of the standardized residuals
# not with a Gaussian, but with a Student-t density with 3 degree of freedom.
# We could achieve this as follows:
myDistance <- function(.ed, .gr, .y){
  d <- dt(.gr / sd(.y), df = 3)
 d <- abs( sqrt(.ed) - sqrt(d) ) # We are using absolute difference between sqrt-densities
}

check1D(b, "x2", type = "response") + l_densCheck(dFun = myDistance) + l_rug()
# NB comparing with a Student density is not useful for this example, but it illustrates
# how both the distance function and the reference density can be customized.
```

`l_fitBar` *Adding barplot to effect plots*

Description

This layer adds a barplot to an effect plots. Mainly useful for factor or binary effect plots.

Usage

```
l_fitBar(a.aes = list(), ...)
```

Arguments

`a.aes`	list of aesthetic mapping arguments that will be passed to `ggplot2::geom_bar`. For instance we could set `a.aes=list("fill"="red")` to change the colour of the barplot.
`...`	graphical arguments to be passed to ggplot2::geom_bar.

Value

an object of class `gamLayer`.

See Also

See plot.ptermFactor for examples.

`l_fitContour` *Adding fitted effect contour lines*

Description

This layer adds the contour lines corresponding to a fitted multidimensional effect.

Usage

```
l_fitContour(...)
```

Arguments

`...`	graphical arguments to be passed to `ggplot2::geom_contour`.

Value

An object of class `gamLayer`.

See Also

See plot.mgcv.smooth.2D, plot.mgcv.smooth.MD, plot.sos.smooth and plotSlice for examples.

l_fitDens *Adding density strip of fitted effect*

Description

This layer adds a conditional posterior density strip to 1D smooth effects plots. With the default colour scale, the opacity is proportional to the conditional density of the fitted effects, under the usual Gaussian approximation the posterior.

Usage

```
l_fitDens(n = 50, level = 0.95, trans = identity, ...)
```

Arguments

`n`	sqrt of the number of grid points used to compute the effect plot.
`level`	confidence level. By default the conditional density of the fit will be plotted between the Gaussian quantiles 0.025 and 0.975, hence the `level` determines the width of the y-axis.
`trans`	monotonic function to be applied to the density of the fit, which determines colour of the plot. Monotonicity is not checked.
`...`	further arguments to be passed to `ggplot2::geom_raster`.

Details

See Bowman (2018) for explanations about the advantages of density strips, relative to plots including the mean fit + confidence intervals.

Value

An object of class gamLayer.

References

Bowman, D. W (2018). Graphics for uncertainty. Journal of the Royal Statistical Society: Series A.

Examples

```
library(mgcViz)
set.seed(44)
dat <- gamSim(1,n=400,dist="normal",scale=2)
b <- gamV(y~s(x0)+x1+s(x2)+s(x3),data=dat)

plot(sm(b, 1)) + l_fitDens() + l_fitLine()
plot(pterm(b, 1)) + l_fitDens(trans = function(x) x^0.25) + l_fitLine()
```

`l_fitLine` *Add fitted smooth effect curve*

Description

This layer add lines representing a single or a group of parametric or smooth 1D effects.

Usage

```
l_fitLine(...)
```

Arguments

`...`	graphical arguments to be passed to `ggplot2::geom_line`.

Details

When used in conjuction with plot.fs.interaction.1D, which plots smooth effects of type `bs="fs"`, this function uses transparency to avoid over-plotting. This can be avoided by setting `alpha = 1` in the call to `l_fitLine`.

Value

An object of class `gamLayer`.

See Also

See plot.mgcv.smooth.1D, plot.ptermNumeric, or plot.fs.interaction.1D for examples.

`l_fitPoints` *Adding points representing the fitted effect*

Description

This function adds points representing the fitted effect. Mainly useful for plotting factor effects.

Usage

```
l_fitPoints(...)
```

Arguments

`...`	graphical arguments to be passed to `ggplot2::geom_point`.

Value

an object of class `gamLayer`.

See Also

See plot.ptermFactor for examples.

`l_fitRaster` *Adding raster representing the fitted effect*

Description

This layer adds a raster or heat-map representing a fitted multidimensional effect.

Usage

```
l_fitRaster(pTrans = function(.p) 1, noiseup = FALSE, mul = 1, ...)
```

Arguments

`pTrans`	a function from (0, 1) to (0, 1) which takes as input a p-value and returns a value, `alpha`, which will be passed on to ggplot2::geom_raster, and will determine the opacity of the heat-map. The p-value quantifies the significance of the smooth effect at each location (x1, x2). By default `pTrans` returns 1, but if we set it to, say, `pTrans = function(.p) .p<0.05` then the regions with p-values higher than 0.05 will disappear. The zto1 function can be used to specify `pTrans` in a flexible way.
`noiseup`	if `TRUE` the fitted effect, mu(x1, x2), will be perturbed with random noise before being plotted. That is, at each location (x1, x2) a random variable z(x1, x2) ~ N(0, mul * V(x1, x2)) will be added to mu(x1, x2). Here V(x1, x2) is the estimated variance of mu(x1, x2) and `mul` is a scalar multiplier (see next argument). This is useful for understanding in which areas the smooth is more uncertain, as these areas will appear more noisy.
`mul`	positive multiplier that scales the variance of the fitted effect. See the `noiseup` argument.
`...`	graphical arguments to be passed to `ggplot2::geom_raster`.

Value

An object of class `gamLayer`.

See Also

See plot.mgcv.smooth.2D, plot.sos.smooth or plotSlice for examples.

l_glyphs2D *Adding glyphs to 2D plots*

Description

This layer adds glyphs or subplots to 2D plots. It is mainly meant to be used with check2D and to produce residuals checks.

Usage

```
l_glyphs2D(
  glyFun,
  ggLay = "geom_points",
  n = c(4, 4),
  mapping = NULL,
  data = NULL,
  polar = FALSE,
  height = ggplot2::rel(0.95),
  width = ggplot2::rel(0.95),
  y_scale = I,
  x_scale = I,
  ...
)
```

Arguments

`glyFun`	the function that produces the data needed to construct the glyphs. It will take a single argument (.d), which is a `data.frame` with columns `"x"`, `"y"` and `"z"`. When `l_glyphs2D` is used with `check2D`, then `"x"` and `"y"` will be the locations of the residual `"z"` in the relevant covariates. `glyFun` needs to output a `data.frame` that will be passed to the `ggLay` function, which does the plotting.
`ggLay`	the `ggplot2` layer function (such as `"geom_point"`) used to plot the glyphs. Its mapping needs to take at least argument "x", "y" and "group". See the `mapping` argument below.
`n`	vector of two positive integers, indicating the number of 2D grid cell along x and y in which the data is divided.
`mapping`	list of aesthetic mappings to be used by `ggLay`. By default it is `aes(x=gx,y=gy,group = gid)`. Here gx and gy specify the x-y location of each data-point used to plot the glyphs, while gid specifies to which glyph each data-point belongs (there are `n[1]*n[2]` glyphs).
`data`	an optional data.frame to be used for computing the glyphs. It must have two variables called `x` and `y`. If left to `NULL` then the glyphs will be computed using the data in the `plotSmooth` object to which this layer is being added.
`polar, height, width, y_scale, x_scale`	see GGally::glyphs.
`...`	graphical arguments to be passed to `ggLay` function.

Value

An object of class gamLayer.

See Also

check2D.

Examples

```
library(mgcViz);
set.seed(4124)
n <- 1e4
dat <- data.frame("x1" = rnorm(n), "x2" = rnorm(n))

# Residuals are heteroscedastic w.r.t. x1
dat$y <- (dat$x1)^2 + (dat$x2)^2 + (1*abs(dat$x1) + 1)  * rnorm(n)
b <- bam(y ~ s(x1,k=30) + s(x2, k=30), data = dat, discrete = TRUE)
b <- getViz(b)

pl <- check2D(b, x1 = "x1", x2 = "x2", type = "tnormal") +
  l_points(colour = "blue", alpha = 0.5)

# Look at distributions of residuals across x1 and x2
# Approach 1: using binned kernel density estimate
# Colour indicates whether we have more that 50 obs in that bin
glyFun <- function(.d){
  .r <- .d$z
  .qq <- as.data.frame( density(.r)[c("x", "y")], n = 100 )
  .qq$colour <- rep(ifelse(length(.r)>50, "black", "red"), nrow(.qq))
  return( .qq )
}

pl + l_glyphs2D(glyFun = glyFun, ggLay = "geom_path", n = c(8, 8),
                mapping = aes(x=gx, y=gy, group = gid, colour = I(colour)),
                height=1.5, width = 1)

# Approach 2: using binned worm-plots. These are simply rotated QQplots.
# An horizontal plot indicates well specified residual model.
# Increasing (decreasing) worm indicates over (under) dispersion
glyFun <- function(.d){
  n <- nrow(.d)
  px <- qnorm( (1:n - 0.5)/(n) )
  py <- sort( .d$z )
  clr <- if(n > 50) { "black" } else { "red" }
  clr <- rep(clr, n)
  return( data.frame("x" = px, "y" = py - px, "colour" = clr))
}

pl + l_glyphs2D(glyFun = glyFun, ggLay = "geom_point", n = c(10, 10),
               mapping = aes(x=gx, y=gy, group = gid, colour = I(colour)),
               height=2, width = 1, size = 0.2)
```

l_gridCheck1D *Binning and checking GAM residuals*

Description

This layer bins the residuals, r, according to the value of the corresponding covariate, x. Then the residuals in each bin are summarized using a scalar-valued statistic. Confidence intervals for the statistic corresponding to each bin can be obtained by simulating residuals from the fitted GAM model, binning and summarizing them. Mainly useful in conjuction with check1D.

Usage

```
l_gridCheck1D(
  gridFun = NULL,
  n = 20,
  level = 0.8,
  stand = "none",
  showReps = TRUE,
  showObs = TRUE,
  ...
)
```

Arguments

`gridFun`	scalar-valued function used to summarize the residuals in each bin. It takes a vector as input. By default it is `mean(r)*sqrt(length(r))`, where `r` is the vector of residuals in that bin.
`n`	number of grid intervals along the relevant covariate.
`level`	the level of the confidence intervals (e.g. 0.9 means 90% intervals).
`stand`	if "none" the residuals in each bin are transformed by `gridFun` and the result statistics are plotted directly. If "sc" the statistics in each bin are scaled and centered using the mean and standard deviation of the simulated stats in that bin. If "s" we do only scaling, if "c" only centering.
`showReps`	if `TRUE` the individuals simulated statistics are also plotted using small points.
`showObs`	if `TRUE` the observed statistics are plotted using large points.
`...`	graphical arguments to be passed to `ggplot2::geom_point`.

Value

An object of class `gamLayer`

Examples

```
library(mgcViz);
set.seed(4124)
n <- 1e4
x <- rnorm(n); y <- rnorm(n);

# Residuals are heteroscedastic w.r.t. x
ob <- (x)^2 + (y)^2 + (0.2*abs(x) + 1)  * rnorm(n)
b <- bam(ob ~ s(x,k=30) + s(y, k=30), discrete = TRUE)
b <- getViz(b, nsim = 50)

# Don't see much by looking at mean
check1D(b, "x") + l_gridCheck1D()

# Heteroscedasticity clearly visible here
check1D(b, "x") + l_gridCheck1D(gridFun = sd, stand = "sc") # <- we are scaling and centering
# Last point on the right of the rug seems to indicate that a bin is missing.
# It is not an error, only on observation falls in that bin, hence the
# standard deviation is not defined there.
```

`l_gridCheck2D` *Binning and checking GAM residuals*

Description

This layer bins the residuals, r, according to the value of the corresponding covariates, x1 and x2. Then the residuals in each bin are summarized using a scalar-valued statistic. Confidence intervals for the statistic corresponding to each bin can be obtained by simulating residuals from the fitted GAM model, which are then binned and summarized. Mainly useful in conjuction with check2D.

Usage

```
l_gridCheck2D(gridFun = mean, bw = c(NA, NA), stand = TRUE, binFun = NULL, ...)
```

Arguments

`gridFun`	scalar-valued function used to summarize the residuals in each bin.
`bw`	numeric vector giving bin width in the vertical and horizontal directions. See the `binwidth` arguments in `?ggplot2::stat_summary_hex`. If left to `NA`, it will be set to 1/20 of the ranges of x1 and x2.
`stand`	if left to `TRUE` then the observed statistic in the i-th cell is normalized using the simulated statistics in that same cell. That is, we will actually plot `std_stat = (obs_stat-mean(sim_stat))/sd(sim_stat)`.
`binFun`	the ggplot2 function used to perform the binning. By default it is either ggplot2::stat_summary_2d or ggplot2::stat_summary_hex, depending on the class of the covariates x1 and x2.
`...`	graphical arguments to be passed to `ggplot2::stat_summary_hex`.

Value

An object of class gamLayer

Examples

```
library(mgcViz);
set.seed(4124)
n <- 1e4
x <- rnorm(n); y <- rnorm(n);

# Residuals are heteroscedastic w.r.t. x
ob <- (x)^2 + (y)^2 + (1*abs(x) + 1)  * rnorm(n)
b <- bam(ob ~ s(x,k=30) + s(y, k=30), discrete = TRUE)
b <- getViz(b, nsim = 50)

# Don't see much by looking at mean
check2D(b, "x", "y") + l_gridCheck2D(gridFun = mean, bw = c(0.4, 0.4))

# Variance pattern along x-axis clearer now
check2D(b, "x", "y") + l_gridCheck2D(gridFun = sd, bw = c(0.4, 0.4))
```

l_gridQCheck1D *Checking sign of residuals along one covariate*

Description

This layer is mainly useful when checking quantile GAMs fitted using the qgam package. The residuals, r, are binned according to the corresponding value of a covariate, x. Then the proportions of negative residuals within each bin are calculated, and compared with the theoretical value, qu. Confidence intervals for the proportion of negative residuals can be derived using binomial quantiles (under an independence assumption). To be used in conjuction with check1D.

Usage

```
l_gridQCheck1D(qu = NULL, n = 20, level = 0.8, ...)
```

Arguments

qu	the quantile of interest. Should be in (0, 1).
n	number of grid intervals.
level	the level of the confidence intervals plotted.
...	graphical arguments to be passed to ggplot2::geom_point.

Value

An object of class gamLayer

Examples

```
# Simulate some data
library(mgcViz)
set.seed(3841)
dat <- gamSim(1,n=400,dist="normal",scale=2)
dat$fac <- as.factor( sample(letters[1:8], nrow(dat), replace = TRUE) )
fit <- qgam(y~s(x1)+s(x2)+s(x3)+fac, data=dat, err = 0.05, qu = 0.4)
fit <- getViz(fit)

# "x0" effect is missing, but should be there. l_gridQCheck1D shows
# that fraction of negative residuals is quite different from the theoretical 0.4
# in several places along "x0".
check1D(fit, dat$x0) + l_gridQCheck1D(qu = 0.4, n = 20)
# The problem gets better if s(x0) is added to the model.

# Works also with factor variables
check1D(fit, "fac") + l_gridQCheck1D(qu = 0.4)
```

`l_gridQCheck2D` *Binning and checking QGAM residuals*

Description

This layer bins the residuals, r, according to the value of the corresponding covariates, x1 and x2. Then we calculate the proportion of negative residuals in each bin, which should not deviate too much from the theoretical proportion (eg 0.5 if we fit the median). Mainly useful in conjuction with check2D.

Usage

```
l_gridQCheck2D(qu = NULL, bw = c(NA, NA), stand = TRUE, binFun = NULL, ...)
```

Arguments

`qu`	the quantile of interest. Should be in (0, 1).
`bw`	numeric vector giving bin width in the vertical and horizontal directions. See the `binwidth` arguments in `?ggplot2::stat_summary_hex`. If left to `NA`, it will be set to 1/20 of the ranges of x1 and x2.
`stand`	if left to `TRUE` then the observed proportion of negative residuals `p_hat` in the i-th cell is normalized using the standard error `se` = `sqrt(qu(1-qu)/n)`, where `n` is the number of observation in that cell. That is, if `stand=TRUE` we plot `(p_hat-qu)/se` rather than simply `p_hat`.
`binFun`	the `ggplot2` function used to perform the binning. By default it is either ggplot2::stat_summary_2d or ggplot2::stat_summary_hex, depending on the class of the covariates x1 and x2.
`...`	graphical arguments to be passed to `ggplot2::stat_summary_hex`.

Value

An object of class gamLayer

Examples

```
library(mgcViz);
set.seed(4124)
n <- 4e2
dat <- data.frame(x = rnorm(n), y = rnorm(n))

# Simulate some data, residuals are heteroscedastic w.r.t. x
dat$ob <- (dat$x)^2 + (dat$y)^2 + (0.2*abs(dat$x) + 1)  * rnorm(n)
b <- qgamV(ob ~ x + s(y), qu = 0.3, data = dat)

# We have a residual pattern along x (increase n above to
# see the problem more clearly)
check2D(b, "x", "y") + l_gridQCheck2D(qu = 0.3, bw = c(0.4, 0.4))

# We need a smooth wrt x to make the pattern disappear
## Not run:
b1 <- qgamV(ob ~ s(x) + s(y), qu = 0.3, data = dat)

check2D(b1, "x", "y") + l_gridQCheck2D(qu = 0.3, bw = c(0.4, 0.4))

## End(Not run)
```

---

l_hist *Adding histogram to a plot*

Description

This layer adds a histogram to a plot. It is mainly a wrapper around ggplot2::geom_histogram.

Usage

```
l_hist(...)
```

Arguments

...	graphical arguments to be passed to ggplot2::geom_histogram.

Value

An object of class gamLayer.

See Also

See check0D for examples.

`l_points` *Add points to plot*

Description

This layers add points to smooth, parametric or random effect plots. It can also be used to add points to the output of `check1D` and `check2D`. The meaning of the added points, which could represent residuals or covariate values, should be clear from context.

Usage

```
l_points(...)
```

Arguments

`...`	graphical arguments to be passed to `ggplot2::geom_point`.

Value

An object of class `gamLayer`.

See Also

See plot.mgcv.smooth.1D, plot.mgcv.smooth.2D, check1D or check2D for examples.

`l_poly` *Add polygons to effect plots*

Description

This layers adds polygons to plots and it is mainly usefuls for plotting Markov random field smooths.

Usage

```
l_poly(...)
```

Arguments

`...`	graphical arguments to be passed to `ggplot2::geom_polygon`.

Value

An object of class `gamLayer`.

See Also

See plot.mrf.smooth for examples.

`l_pvContour` *Adding contour of p-values*

Description

This function adds contour lines proportional to the p-value of a multidimensional smooth effects. It is useful for checking where (across covariates x1 and x2) the fitted smooth is significantly different from zero.

Usage

```
l_pvContour(pTrans = identity, ...)
```

Arguments

`pTrans`	a transformation to be applied to the p-values before plotting.
`...`	graphical arguments to be passed to `ggplot2::geom_contour`.

Value

An object of class `gamLayer`.

See Also

See plotDiff.mgcv.smooth.2D and plotDiff.sos.smooth for examples.

`l_pvRaster` *Adding raster or heat-map of p-values*

Description

This function adds a raster or heat-map proportional to the p-value of a multidimensional smooth effects. It is useful for checking where (across covariates x1 and x2) the fitted smooth is significantly different from zero.

Usage

```
l_pvRaster(pTrans = identity, ...)
```

Arguments

`pTrans`	a transformation to be applied to the p-values before plotting.
`...`	graphical arguments to be passed to `ggplot2::geom_raster`.

Value

An object of class `gamLayer`.

See Also

See plotDiff.mgcv.smooth.2D and plotDiff.sos.smooth for examples.

`l_rug` *Adding rug to margins of a plot*

Description

This layer adds a rug plot to the margins of a plot. It is mainly a wrapper around ggplot2::geom_rug. Notice that for factor effects plots the rug is jittered by default.

Usage

```
l_rug(...)
```

Arguments

`...`	graphical arguments to be passed to `ggplot2::geom_rug`.

Value

An object of class `gamLayer`.

See Also

See plot.mgcv.smooth.1D, plot.mgcv.smooth.2D or check1D for examples.

`l_simLine` *Add simulated smooth effect curves*

Description

This layer adds curves representing smooth effects simulated from the posterior distribution.

Usage

```
l_simLine(...)
```

Arguments

`...`	graphical arguments to be passed to `ggplot2::geom_line`.

Details

This function uses transparency to avoid over-plotting. This can be avoided by setting `alpha = 1` in the call to `l_simLine`.

Value

An object of class `gamLayer`.

See Also

See plot.mgcv.smooth.1D for examples.

`l_vline` *Adding vertical line to a plot*

Description

This layer adds a vertical to a plot. It is mainly a wrapper around ggplot2::geom_vline.

Usage

```
l_vline(...)
```

Arguments

`...`	graphical arguments to be passed to `ggplot2::geom_vline`.

Value

An object of class `gamLayer`.

See Also

See check0D for examples.

mqgamV *Fit multiple QGAM models and get a mgamViz object*

Description

These are wrapper that fits multple QGAM models using qgam::mqgam and converts it to a `mgamViz` object using the getViz function. It is essentially a shortcut.

Usage

```
mqgamV(form, data, qu, lsig = NULL, err = NULL, aQgam = list(), aViz = list())
```

Arguments

`form, data, qu, lsig, err`	same arguments as in qgam::mqgam.
`aQgam`	list of further arguments to be passed to qgam::mqgam.
`aViz`	list of arguments to be passed to getViz.

Value

An object of class "mgamViz" which can, for instance, be plotted using plot.mgamViz.

Examples

```
library(mgcViz)
set.seed(2) ## simulate some data...
dat <- gamSim(2,n=500,dist="normal",scale=0.25)$data

# Fit GAM and get gamViz object
b <- mqgamV(y~s(x) + s(z) + I(x*z), data = dat, qu = c(0.25, 0.5, 0.75),
            aQgam = list(argGam = list(select = TRUE)), aViz = list("nsim" = 0))

# This is equivalent to doing
# 1. Fit QGAM
# b <- mqgam(y~s(x) + s(z) + I(x*z), data=dat,
#             qu = c(0.25, 0.5, 0.75), argGam = list(select = TRUE))
# 2. Convert to gamViz object
# b <- getViz(b, nsim = 0)

# Either way, we all effects by doing
print(plot(b, allTerms = TRUE), pages = 1)
```

`plot.ALE1D` *Plot 1D Accumulated Local Effects (ALE)*

Description

Plot 1D Accumulated Local Effects (ALE)

Usage

```
## S3 method for class 'ALE1D'
plot(x, trans = identity, maxpo = 10000, nsim = 0, ...)
```

Arguments

`x`	a 1D ALE effects, produced by the `ALE` function
`trans`	monotonic function to apply to the ALE effect, before plotting. Monotonicity is not checked.
`maxpo`	maximum number of rug lines that will be used by `l_rug`. If number of datapoints > `maxpo`, then a subsample of `maxpo` points will be taken.
`nsim`	number of ALE effect curves to be simulated from the posterior distribution. These can be plotted using the l_simLine layer. See Examples section below.
`...`	currently not used.

Value

An objects of class `plotSmooth`.

Author(s)

Matteo Fasiolo and Christian Capezza, with some internal code having been adapted from the ALEPlot package of Dan Apley.

References

Apley, D.W., and Zhu, J, 2016. Visualizing the effects of predictor variables in black box supervised learning models. arXiv preprint arXiv:1612.08468.

Examples

```
library(mgcViz)
# Here x1 and x2 are very correlated, but only
# x1 has influence of the response
set.seed(4141)
n <- 1000
X <- rmvn(n, c(0, 0), matrix(c(1, 0.9, 0.9, 1), 2, 2))
y <- X[ , 1] + 0.2 * X[ , 1]^2 + rnorm(n, 0, 0.8)
```

```
dat <- data.frame(y = y, x1 = X[ , 1], x2 = X[ , 2])
fit <- gam(y ~ te(x1, x2), data = dat)

# Marginal plot suggests that E(y) depends on x2, but
# this is due to the correlation between x1 and x2...
plot(dat$x2, fit$fitted.values)

# ... in fact ALE effect of x2 is flat ...
plot(ALE(fit, "x2")) + l_ciPoly() + l_fitLine() + l_rug()

# ... while ALE effect of x1 is strong.
plot(ALE(fit, "x1", center = 2), nsim = 20) +
  l_simLine() + l_fitLine()
```

`plot.fs.interaction.1D` *Plotting one dimensional smooth factor interactions*

Description

This method should be used to plot smooth effects of class `"fs.interaction.1D"`, that is smooth constructed using the basis `bs="tp"`. See mgcv::s.

Usage

```
## S3 method for class 'fs.interaction.1D'
plot(x, n = 100, xlim = NULL, trans = identity, ...)
```

Arguments

`x`	a smooth effect object.
`n`	number of grid points used to compute main effect and c.i. lines. For a nice smooth plot this needs to be several times the estimated degrees of freedom for the smooth.
`xlim`	if supplied then this pair of numbers are used as the x limits for the plot.
`trans`	monotonic function to apply to the smooth and residuals, before plotting. Monotonicity is not checked.
`...`	currently unused.

Value

An object of class `c("plotSmooth","gg")`.

Examples

```
library(mgcViz)
set.seed(0)
## simulate data...
f0 <- function(x) 2 * sin(pi * x)
f1 <- function(x, a = 2, b = -1) exp(a * x) + b
f2 <- function(x) 0.2 * x^11 * (10 * (1 - x))^6 + 10 *
  (10 * x)^3 * (1 - x)^10
n <- 500; nf <- 25
fac <- sample(1:nf, n, replace = TRUE)
x0 <- runif(n); x1 <- runif(n); x2 <- runif(n)
a <- rnorm(nf) * .2 + 2; b <- rnorm(nf) * .5
f <- f0(x0) + f1(x1, a[fac], b[fac]) + f2(x2)
fac <- factor(fac)
y <- f + rnorm(n) * 2
## so response depends on global smooths of x0 and
## x2, and a smooth of x1 for each level of fac.

## fit model (note p-values not available when fit
## using gamm)...
bm <- gamm(y ~ s(x0)+ s(x1, fac, bs = "fs", k = 5) + s(x2, k = 20))
v <- getViz(bm$gam)

# Plot with fitted effects and changing title
plot(sm(v, 2)) + l_fitLine(alpha = 0.6) + labs(title = "Smooth factor interactions")

# Changing plotting limits
plot(sm(v, 2)) + l_fitLine() + ylim(-0.5, 0.5) + xlim(0.25, 0.75)

# Change line type and remove legend
plot(sm(v, 2)) + l_fitLine(size = 1.3, linetype="dotted") +
                 theme(legend.position="none")

# Clustering smooth effects in 3 groups
plot(sm(v, 2)) + l_fitLine(colour = "grey") +
                 l_clusterLine(centers = 3, a.clu = list(nstart = 100))
```

---

plot.gamViz *Basic GAM plotting*

Description

This function is the mgcViz equivalent of plot.gam. It is the workhorse of the mgcViz package, and allows plotting (almost) any type of smooth, parametric or random effects. It is basically a wrapper around plotting methods that are specific to individual smooth effect classes (such as plot.mgcv.smooth.1D and plot.random.effect).

Usage

```
## S3 method for class 'gamViz'
plot(x, n = 100, n2 = 40, select = NULL, allTerms = FALSE, ...)
```

Arguments

`x`	an object of class `gamViz`, the output of a getViz call.
`n`	number of points used for each 1-d plot. For a nice smooth plot this needs to be several times the estimated degrees of freedom for the smooth.
`n2`	square root of number of grid points used for plotting 2D functions effects using contours or heatmaps.
`select`	allows plotting a subset of model terms. For instance, if you just want the plot for the second smooth term, set `select = 2`. Parametric effects always come after smooth or random effects.
`allTerms`	if `TRUE` also the parametric effects will be plotted.
`...`	other parameters, such as `maxpo` or `trans`, to be passed to the specific plotting methods for each effect (e.g. to plot.mgcv.smooth.1D).

Value

An object of class `c("plotGam","gg")`.

Examples

```
library(mgcViz)

######## Basic example
# Simulate some data and fit model
set.seed(2)
dat <- gamSim(1,n=1e3,dist="normal",scale=2)
b <- bam(y~s(x0)+s(x1, x2)+s(x3), data=dat)
b <- getViz(b)

# Default smooth effect plotting
print(plot(b), ask = FALSE)

# Now on one page and with out title on the second plot
print(plot(b) + labs(title = NULL), pages = 1)

# So far we used default layers, added in the printing phase, but
# we might want to specify our own layers. Here we is how to do it
pl <- plot(b) + l_points() + l_fitLine(linetype = 3) + l_fitContour() +
  l_ciLine(colour = 2) + theme_get() + labs(title = NULL)
print(pl, pages = 1)

# We might want to plot only the first smooth
plot(b, select = 1) + l_dens(type = "cond") + l_fitLine() + l_ciLine()

## Not run:
######## Example with "by variable" smooth effect
# Simulate data and fit model
dat <- gamSim(4)
b <- gam(y ~ fac+s(x2,by=fac)+s(x0),data=dat)
b <- getViz(b)
```

```
# print() only needed because we want to plot on a single page
print(plot(b), pages = 1)
print(plot(b, allTerms = TRUE), pages = 1) # Including also parametric effect

######## Example with 3D smooth effect which cannot be plotted
# Simulate data and fit model
n <- 5e3
x <- rnorm(n); y <- rnorm(n); z <- rnorm(n); z2 <- rnorm(n)

ob <- (x-z)^2 + (y-z)^2 + z2^3 + rnorm(n)
b1 <- bam(ob ~ s(x, y, z) + s(z2), discrete = TRUE)
b1 <- getViz(b1)

# Only second effect get plotted
plot(b1)
# In fact this does not plot anything
plot(b1, select = 1)
# For plotting effects with more than 2D, one we need specific method.
# See ?plot.mgcv.smooth.MD

######## Examples about plotting parametric effects
# 1 Gaussian GAM
set.seed(3)
dat <- gamSim(1,n=2500,dist="normal",scale=20)
dat$fac <- as.factor( sample(c("A1", "A2", "A3"), nrow(dat), replace = TRUE) )
dat$logi <- as.logical( sample(c(TRUE, FALSE), nrow(dat), replace = TRUE) )
bs <- "cr"; k <- 12
b <- bam(y ~ x0 + x1 + I(x1^2) + s(x2,bs=bs,k=k) + fac + x3:fac + I(x1*x2) + logi +
            s(x3, bs=bs),data=dat, discrete = TRUE)
b <- getViz(b)

# All effects in one page. Notably 'x3:fac' is missing: we have no methods
# for plotting second order effects.
print(plot(b, allTerms = TRUE), pages = 1)

# Plotting only parametric effects
print(plot(b, select = 3:9), pages = 1)

# 2 GAMLSS Gaussian model
library(mgcv);library(MASS)
mcycle$fac <- as.factor( sample(c("z", "k", "a", "f"), nrow(mcycle), replace = TRUE) )
b <- gam(list(accel~times + I(times^2) + s(times,k=10), ~ times + fac + s(times)),
          data=mcycle,family=gaulss())
b <- getViz(b)

# All effects on one page: effect of second linear predictor end with '.1'
print(plot(b, allTerms = TRUE), pages = 1)

## End(Not run)
```

`plot.mgamViz` *Plotting multiple quantile GAMs*

Description

This function is similar to plot.gamViz, but it is used to plot multiple quantile GAM models fitted using mqgamV or mqgam. It allows plotting standards 1D and 2D smooths, and parametric effects, It is basically a wrapper around plotting methods that are specific to individual smooth effect classes (such as plot.multi.mgcv.smooth.1D).

Usage

```
## S3 method for class 'mgamViz'
plot(x, n = 100, n2 = 40, select = NULL, allTerms = FALSE, ...)
```

Arguments

`x`	an object of class `mgamViz`, the output of a getViz call. Alternatively x can be a list of fitted GAM models, each having the same model formula.
`n`	number of points used for each 1-d plot. For a nice smooth plot this needs to be several times the estimated degrees of freedom for the smooth.
`n2`	square root of number of grid points used for plotting 2D functions effects using contours or heatmaps.
`select`	allows plotting a subset of model terms. For instance, if you just want the plot for the second smooth term, set `select = 2`. Parametric effects always come after smooth or random effects.
`allTerms`	if `TRUE` also the parametric effects will be plotted.
`...`	other parameters, such as `maxpo` or `trans`, to be passed to the specific plotting methods for each effect (e.g. to plot.multi.mgcv.smooth.1D).

Value

An object of class `c("plotGam","gg")`.

Examples

```
library(mgcViz)
set.seed(2) ## simulate some data...
dat <- gamSim(1,n=500,dist="normal",scale=2)
dat$logi <- as.logical( sample(c(TRUE, FALSE), nrow(dat), replace = TRUE) )

dat$fac <- as.factor( sample(c("A1", "A2", "A3"), nrow(dat), replace = TRUE) )

# Fit GAM and get gamViz object
fit <- mqgamV(y ~ fac + s(x0) + s(x1, x2) + x3 + logi, data = dat,
              qu = c(0.2, 0.4, 0.6, 0.8))
```

```
print(plot(fit, select = 1:4, allTerms = T), pages = 1)

## Not run:
# Example where we are fitting the same model to different datasets, but
# plotting the estimate effects together
dat <- list()
for(ii in 1:4){
  # Simulate 4 datasets, we are adding 2 factor variables "fac" and "ref" just
  # for illustrating the plotting method (the two factors have no effect on y)
  n <- 1000
  dat[[ii]] <- gamSim(1,n=n,dist="normal",scale=2)
  dat[[ii]]$fac <- as.factor( sample(c("A1", "A2", "A3"), n, replace = TRUE) )
  dat[[ii]]$ref <- as.factor( sample(letters[1:10], n, replace = TRUE) )
}

# Estimating model on each dataset
mods <- list()
for(ii in 1:4){
  mods[[ii]] <- gamV(y~s(x0)+s(x1, x2)+x3+fac+s(ref, bs = "re"), data = dat[[ii]])
}

# Names will be used to identify the four models we have fitted
names(mods) <- c("M1", "M2", "M3", "M4")
# Plotting on the same plots
print(plot.mgamViz(mods, allTerms = TRUE), pages = 1)

## End(Not run)
```

plot.mgcv.smooth.1D *Plotting one dimensional smooth effects*

Description

Plotting method for one dimensional smooth effects.

Usage

```
## S3 method for class 'mgcv.smooth.1D'
plot(
  x,
  n = 100,
  xlim = NULL,
  maxpo = 10000,
  trans = identity,
  unconditional = FALSE,
  seWithMean = FALSE,
  nsim = 0,
  ...
)

## S3 method for class 'multi.mgcv.smooth.1D'
plot(
  x,
  n = 100,
  xlim = NULL,
  maxpo = 10000,
  trans = identity,
  unconditional = FALSE,
  seWithMean = FALSE,
  asFact = NULL,
  ...
)
```

Arguments

`x`	a smooth effect object, extracted using sm.
`n`	number of grid points used to compute main effect and c.i. lines. For a nice smooth plot this needs to be several times the estimated degrees of freedom for the smooth.
`xlim`	if supplied then this pair of numbers are used as the x limits for the plot.
`maxpo`	maximum number of residuals points that will be used by layers such as `resRug()` and `resPoints()`. If number of datapoints > `maxpo`, then a subsample of `maxpo` points will be taken.
`trans`	monotonic function to apply to the smooth and residuals, before plotting. Monotonicity is not checked.
`unconditional`	if `TRUE` then the smoothing parameter uncertainty corrected covariance matrix is used to compute uncertainty bands, if available. Otherwise the bands treat the smoothing parameters as fixed.
`seWithMean`	if TRUE the component smooths are shown with confidence intervals that include the uncertainty about the overall mean. If FALSE then the uncertainty relates purely to the centred smooth itself. Marra and Wood (2012) suggests that TRUE results in better coverage performance, and this is also suggested by simulation.
`nsim`	number of smooth effect curves to be simulated from the posterior distribution. These can be plotted using the l_simLine layer. See Examples section below.
`...`	currently unused.
`asFact`	determines whether to use a factor or colour bar legend for plot.multi.mgcv.smooth.1D. For most models the default is `TRUE`. When working with QGAM models fitted with mqgamV, the default is `FALSE` for less than 10 quantiles, `TRUE` otherwise. For QGAM models there a third option, `asFact = "force"`, which forces the use of a discrete colour scale.

Value

An objects of class `plotSmooth`.

References

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

Examples

```
library(mgcViz)
n  <- 1e3
x1 <- rnorm(n)
x2 <- rnorm(n)
dat <- data.frame("x1" = x1, "x2" = x2,
                  "y" = sin(x1) + 0.5 * x2^2 + pmax(x2, 0.2) * rnorm(n))
b <- bamV(y ~ s(x1)+s(x2), data = dat, method = "fREML", aGam = list(discrete = TRUE))

o <- plot(sm(b, 1), nsim = 50) # 50 posterior simulations

## Not run:
# Plot with fitted effect + posterior simulations + rug on x axis
( o <- o + l_simLine() + l_fitLine(colour = "red") +
       l_rug(alpha = 0.8) )

# Add CI lines at 1*sigma and 5*sigma
( o <- o + l_ciLine(mul = 1) + l_ciLine(mul = 5, colour = "blue", linetype = 2) )

# Add partial residuals and change theme
( o + l_points(shape = 19, size = 1, alpha = 0.2) + theme_classic() )

# Get second effect plot
o2 <- plot( sm(b, 2) )

# Plot it with polygon for partial residuals
o2 + l_ciPoly(mul = 5, fill = "light blue") +
  l_fitLine(linetype = 2, colour = "red")

# Plot is with conditional density of partial residuals
o2 + l_dens(type = "cond", alpha = 0.9)  +
  l_fitLine(linetype = 2, colour = "red")

########
# Quantile GAM example
########
# Fit model
b <- mqgamV(y ~ s(x1) + s(x2), qu = c(0.2, 0.5, 0.8), data = dat)

plot(sm(b, 1)) + l_fitLine(linetype = 2) + l_rug(colour = "blue")

## End(Not run)
```

plot.mgcv.smooth.2D *Plotting two dimensional smooth effects*

Description

Plotting method for two dimensional smooth effects.

Usage

```
## S3 method for class 'mgcv.smooth.2D'
plot(
  x,
  n = 40,
  xlim = NULL,
  ylim = NULL,
  maxpo = 10000,
  too.far = 0.1,
  trans = identity,
  seWithMean = FALSE,
  unconditional = FALSE,
  ...
)

## S3 method for class 'multi.mgcv.smooth.2D'
plot(
  x,
  n = 30,
  xlim = NULL,
  ylim = NULL,
  maxpo = 10000,
  too.far = 0.1,
  trans = identity,
  seWithMean = FALSE,
  unconditional = FALSE,
  a.facet = list(),
  ...
)
```

Arguments

`x`	a smooth effect object, extracted using sm.
`n`	sqrt of the number of grid points used to compute the effect plot.
`xlim`	if supplied then this pair of numbers are used as the x limits for the plot.
`ylim`	if supplied then this pair of numbers are used as the y limits for the plot.

`maxpo`	maximum number of residuals points that will be used by layers such as `resRug()` and `resPoints()`. If number of datapoints > `maxpo`, then a subsample of `maxpo` points will be taken.
`too.far`	if greater than 0 then this is used to determine when a location is too far from data to be plotted. This is useful since smooths tend to go wild away from data. The data are scaled into the unit square before deciding what to exclude, and too.far is a distance within the unit square. Setting to zero can make plotting faster for large datasets, but care then needed with interpretation of plots.
`trans`	monotonic function to apply to the smooth and residuals, before plotting. Monotonicity is not checked.
`seWithMean`	if TRUE the component smooths are shown with confidence intervals that include the uncertainty about the overall mean. If FALSE then the uncertainty relates purely to the centred smooth itself. Marra and Wood (2012) suggests that TRUE results in better coverage performance, and this is also suggested by simulation.
`unconditional`	if `TRUE` then the smoothing parameter uncertainty corrected covariance matrix is used to compute uncertainty bands, if available. Otherwise the bands treat the smoothing parameters as fixed.
`...`	currently unused.
`a.facet`	arguments to be passed to ggplot2::facet_wrap or ggplot2::facet_grid. The former gets called when `fix` contains one vector, the latter when `fix` contains two vectors.

Value

An objects of class `plotSmooth`.

References

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

Examples

```
library(mgcViz)
set.seed(2) ## simulate some data...
dat <- gamSim(1, n = 700, dist = "normal", scale = 2)
b <- gam(y ~ s(x0) + s(x1, x2) + s(x3), data = dat, method = "REML")
b <- getViz(b)

# Plot 2D effect with noised-up raster, contour and rug for design points
# Opacity is proportional to the significance of the effect
plot(sm(b, 2)) + l_fitRaster(pTrans = zto1(0.05, 2, 0.1), noiseup = TRUE) +
  l_rug() + l_fitContour()

# Plot contour of effect joint density of design points
plot(sm(b, 2)) + l_dens(type = "joint") + l_points() + l_fitContour() +
  coord_cartesian(expand = FALSE) # Fill the plot
```

```
###
# Quantile GAM example
###
b <- mqgamV(y ~ s(x0) + s(x1, x2) + s(x3), qu = c(0.3, 0.7), data = dat)

plot(sm(b, 2)) + l_fitRaster(noiseup = TRUE) + l_fitContour(colour = 2)
```

`plot.mgcv.smooth.MD` *Plotting slice of higher-dimensional smooth effects*

Description

This function plots a 2D slice of a higher-dimensional smooth effects.

Usage

```
## S3 method for class 'mgcv.smooth.MD'
plot(
  x,
  fix,
  n = 40,
  xlim = NULL,
  ylim = NULL,
  maxpo = 10000,
  too.far = c(0.1, NA),
  trans = identity,
  seWithMean = FALSE,
  unconditional = FALSE,
  ...
)
```

Arguments

`x`	a smooth effect object, extracted using sm.
`fix`	a named vector indicating which variables must be kept fixed and to what values. When plotting a smooth in (d+2) dimensions, then d variables must be fixed.
`n`	sqrt of the number of grid points used to compute the effect plot.
`xlim`	if supplied then this pair of numbers are used as the x limits for the plot.
`ylim`	if supplied then this pair of numbers are used as the y limits for the plot.
`maxpo`	maximum number of residuals points that will be used by layers such as `resRug()` and `resPoints()`. If number of datapoints > `maxpo`, then a subsample of `maxpo` points will be taken.

`too.far`	a numeric vector with two entries. The first has the same interpretation as in plot.mgcv.smooth.2D and it avoids plotting the smooth effect in areas that are too far form any observation. The distance will be calculated only using the variables which are not in `fix` (see above). Hence in two dimensions, not in the full d+2 dimensions. Set it to -1 to plot the whole smooth. The second entry determines which residuals and covariates pairs are closed enough to the selected slice. If left to `NA` on the 10\ closest (in terms of scaled Euclidean distance) to the current slice will be plotted. Set it to -1 to plot all the residuals.
`trans`	monotonic function to apply to the smooth and residuals, before plotting. Monotonicity is not checked.
`seWithMean`	if TRUE the component smooths are shown with confidence intervals that include the uncertainty about the overall mean. If FALSE then the uncertainty relates purely to the centred smooth itself. Marra and Wood (2012) suggests that TRUE results in better coverage performance, and this is also suggested by simulation.
`unconditional`	if `TRUE` then the smoothing parameter uncertainty corrected covariance matrix is used to compute uncertainty bands, if available. Otherwise the bands treat the smoothing parameters as fixed.
`...`	currently unused.

Value

An objects of class `plotSmooth`.

References

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

Examples

```
## Not run:
## 3D example
library(mgcViz)
n <- 1e3
x <- rnorm(n); y <- rnorm(n); z <- rnorm(n)

ob <- (x-z)^2 + (y-z)^2 + rnorm(n)
b <- gam(ob ~ s(x, y, z))
b <- getViz(b)

# Plot one 2D slice
plot( sm(b, 1), fix = c("z"=0) ) + l_fitRaster(noiseup = TRUE, mul = 3) +
  l_fitContour(linetype = 2) + l_points(shape =  2)

## 4D
n <- 5e3
x <- rnorm(n); y <- rnorm(n); z <- rnorm(n); z2 <- rnorm(n)

ob <- (x-z)^2 + (y-z)^2 + z2^3 + rnorm(n)
```

```
b1 <- bam(ob ~ s(x, y, z, z2), discrete = TRUE)
b1 <- getViz(b1)

# Plot one 2D slice
plot(sm(b1, 1), fix = c("z"=0, "z2"=1)) + l_fitRaster() + l_fitContour()

## End(Not run)
```

plot.mrf.smooth *Plotting Markov random field smooths*

Description

This is the plotting method for Markov random field smooths.

Usage

```
## S3 method for class 'mrf.smooth'
plot(x, trans = identity, seWithMean = FALSE, unconditional = FALSE, ...)
```

Arguments

`x`	a smooth effect object, extracted using sm.
`trans`	monotonic function to apply to the smooth and residuals, before plotting. Monotonicity is not checked.
`seWithMean`	if TRUE the component smooths are shown with confidence intervals that include the uncertainty about the overall mean. If FALSE then the uncertainty relates purely to the centred smooth itself. Marra and Wood (2012) suggests that TRUE results in better coverage performance, and this is also suggested by simulation.
`unconditional`	if `TRUE` then the smoothing parameter uncertainty corrected covariance matrix is used to compute uncertainty bands, if available. Otherwise the bands treat the smoothing parameters as fixed.
`...`	currently unused.

Value

An objects of class `plotSmooth`.

References

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

Examples

```
library(mgcViz)
## Load Columbus Ohio crime data (see ?columbus for details and credits)
data(columb)       ## data frame
data(columb.polys) ## district shapes list
xt <- list(polys=columb.polys) ## neighbourhood structure info for MRF
par(mfrow=c(2,2))
## First a full rank MRF...
b <- gam(crime ~ s(district,bs="mrf",xt=xt),data=columb,method="REML")
b <- getViz(b)

# Manual plot
plot(sm(b, 1)) + l_poly(colour = 2) +
  scale_fill_gradientn(colours = heat.colors(50))

# Default plot
plot(b)
```

`plot.multi.ptermFactor`
Plotting factor or logical parametric effects

Description

These are the plotting methods for parametric factor or logical effects.

Usage

```
## S3 method for class 'multi.ptermFactor'
plot(x, a.facet = list(), asFact = TRUE, ...)

## S3 method for class 'multi.ptermLogical'
plot(x, ...)

## S3 method for class 'ptermFactor'
plot(x, maxpo = 10000, trans = identity, ...)

## S3 method for class 'ptermLogical'
plot(x, maxpo = 10000, trans = identity, ...)
```

Arguments

`x`	a factor or logical parametric effect object, extracted using pterm.
`a.facet`	arguments to be passed to ggplot2::facet_wrap or ggplot2::facet_grid. The former gets called when `fix` contains one vector, the latter when `fix` contains two vectors.

asFact	relevant only when working with models fitted with mqgamV. If FALSE quantile of interest (qu) is treated as a continuous variable, otherwise as a factor.
...	currently unused.
maxpo	maximum number of residuals points that will be used by layers such as resRug() and resPoints(). If number of datapoints > maxpo, then a subsample of maxpo points will be taken.
trans	monotonic function to apply to the fit, confidence intervals and residuals, before plotting. Monotonicity is not checked.

Value

An object of class plotSmooth.

Examples

```
# Simulate data and fit GAM
set.seed(3)
dat <- gamSim(1,n=2000,dist="normal",scale=20)
dat$fac <- as.factor( sample(c("A1", "A2", "A3"), nrow(dat), replace = TRUE) )
dat$logi <- as.logical( sample(c(TRUE, FALSE), nrow(dat), replace = TRUE) )
bs <- "cr"; k <- 12
b <- gam(y~fac + s(x0) + s(x1) + s(x2) + s(x3) + logi, data=dat)
o <- getViz(b, nsim = 0)

# Extract factor terms and plot it
pt <- pterm(o, 1)
plot(pt) + l_ciBar() + l_fitPoints(colour = 2) + l_rug(alpha = 0.2)

# Use barplot instead of points
pt <- pterm(o, 1)
plot(pt) + l_fitBar() + l_ciBar()

# Same with binary varible
pt <- pterm(o, 2)
plot(pt) + l_fitPoints() + l_ciBar()
```

---

plot.multi.random.effect

Plotting random effects

Description

This is the plotting method for random effects (simple random intercepts).

Usage

```
## S3 method for class 'multi.random.effect'
plot(x, trans = identity, ...)

## S3 method for class 'random.effect'
plot(x, trans = identity, ...)
```

Arguments

`x`	a random effect object, extracted using sm.
`trans`	monotonic function to apply to the fit, confidence intervals and residuals, before plotting. Monotonicity is not checked.
`...`	currently unused.

Value

An object of class `plotSmooth`.

Examples

```
library(mgcViz)
b <- gam(travel~s(Rail,bs="re"), data=Rail, method="REML")
b <- getViz(b)
plot(sm(b, 1)) + l_fitLine(colour = 2, linetype = 2) + l_points() +
  l_ciLine(colour = 4, linetype = 3)

plot(sm(b, 1)) + l_ciPoly() + l_points()

# Default
plot(b)

###
# Quantile GAM version
###
b <- mqgamV(travel~s(Rail,bs="re"), data=as.data.frame(Rail), qu = c(0.2, 0.4, 0.6, 0.8))

plot(sm(b, 1)) + l_ciPoly() + l_points()

# Default
plot(b)
```

`plot.ptermInteraction` *Plotting parametric interactions*

Description

This function is here only to deal with parametric interactions (eg x0:fact), which cannot be plotted at the moment.

Usage

```
## S3 method for class 'ptermInteraction'
plot(x, ...)

## S3 method for class 'multi.ptermInteraction'
plot(x, ...)
```

Arguments

`x`	a parametric interaction object, extracted using pterm.
`...`	currently unused.

Value

Currently it returns `NULL`.

---

plot.ptermMatrixNumeric

Plotting numeric parametric effects

Description

This is the plotting method for parametric numerical effects.

Usage

```
## S3 method for class 'ptermMatrixNumeric'
plot(x, n = 100, xlim = NULL, trans = identity, ...)

## S3 method for class 'multi.ptermNumeric'
plot(x, ...)

## S3 method for class 'ptermNumeric'
plot(x, n = 100, xlim = NULL, maxpo = 10000, trans = identity, ...)
```

Arguments

`x`	a numerical parametric effect object, extracted using pterm.
`n`	number of grid points used to compute main effect and c.i. lines.
`xlim`	if supplied then this pair of numbers are used as the x limits for the plot.
`trans`	monotonic function to apply to the fit, confidence intervals and residuals, before plotting. Monotonicity is not checked.
`...`	currently unused.
`maxpo`	maximum number of residuals points that will be used by layers such as `resRug()` and `resPoints()`. If number of datapoints > `maxpo`, then a subsample of `maxpo` points will be taken.

Value

An object of class `plotSmooth`.

Examples

```
# Simulate data and fit GAM
set.seed(3)
dat <- gamSim(1,n=2000,dist="normal",scale=20)
bs <- "cr"; k <- 12
b <- gam(y ~  x0 + x1 + I(x1^2) + s(x2,bs=bs,k=k) +
              I(x1*x2) + s(x3, bs=bs), data=dat)
o <- getViz(b, nsim = 0)

# Extract first terms and plot it
pt <- pterm(o, 1)
plot(pt, n = 60) + l_ciPoly() + l_fitLine() + l_ciLine()

# Extract I(x1^2) terms and plot it with partial residuals
pt <- pterm(o, 3)
plot(pt, n = 60) + l_ciPoly() + l_fitLine() + l_ciLine() + l_points()
```

`plot.sos.smooth` *Plotting smooths on the sphere*

Description

This is the plotting method for smooth effects on the sphere.

Usage

```
## S3 method for class 'sos.smooth'
plot(
  x,
  n = 40,
  xlim = NULL,
  ylim = NULL,
  maxpo = 10000,
  too.far = 0.1,
  phi = 30,
  theta = 30,
  trans = identity,
  scheme = 0,
  seWithMean = FALSE,
  unconditional = FALSE,
  ...
)
```

Arguments

`x`	a smooth effect object, extracted using sm.
`n`	sqrt of the number of grid points used to compute the effect plot.
`xlim`	if supplied then this pair of numbers are used as the x limits for the plot.
`ylim`	if supplied then this pair of numbers are used as the y limits for the plot.
`maxpo`	maximum number of residuals points that will be used by layers such as `resRug()` and `resPoints()`. If number of datapoints > `maxpo`, then a subsample of `maxpo` points will be taken.
`too.far`	if greater than 0 then this is used to determine when a location is too far from data to be plotted. This is useful since smooths tend to go wild away from data. The data are scaled into the unit square before deciding what to exclude, and too.far is a distance within the unit square. Setting to zero can make plotting faster for large datasets, but care then needed with interpretation of plots.
`phi`	one of the plotting angles, relevant only if `scheme = 0`.
`theta`	the other plotting angle, relevant only if `scheme = 0`.
`trans`	monotonic function to apply to the smooth and residuals, before plotting. Monotonicity is not checked.
`scheme`	if 0 the smooth effect is plotted on the sphere. If 1 the smooth effect is plotted on the two hemispheres.
`seWithMean`	if TRUE the component smooths are shown with confidence intervals that include the uncertainty about the overall mean. If FALSE then the uncertainty relates purely to the centred smooth itself. Marra and Wood (2012) suggests that TRUE results in better coverage performance, and this is also suggested by simulation.
`unconditional`	if `TRUE` then the smoothing parameter uncertainty corrected covariance matrix is used to compute uncertainty bands, if available. Otherwise the bands treat the smoothing parameters as fixed.
`...`	currently unused.

Value

An objects of class `plotSmooth`.

References

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

Examples

```
library(mgcViz)
set.seed(0)
n <- 400

f <- function(la,lo) { ## a test function...
```

```
  sin(lo)*cos(la-.3)
}

## generate with uniform density on sphere...
lo <- runif(n)*2*pi-pi ## longitude
la <- runif(3*n)*pi-pi/2
ind <- runif(3*n)<=cos(la)
la <- la[ind];
la <- la[1:n]

ff <- f(la,lo)
y <- ff + rnorm(n)*.2 ## test data

## generate data for plotting truth...
lam <- seq(-pi/2,pi/2,length=30)
lom <- seq(-pi,pi,length=60)
gr <- expand.grid(la=lam,lo=lom)
fz <- f(gr$la,gr$lo)
zm <- matrix(fz,30,60)

require(mgcv)
dat <- data.frame(la = la *180/pi,lo = lo *180/pi,y=y)

## fit spline on sphere model...
bp <- gam(y~s(la,lo,bs="sos",k=60),data=dat)
bp <- getViz(bp)

# Plot on sphere
plot(sm(bp, 1), scheme=0) + l_fitRaster() + l_fitContour() +
   l_points(shape = 19) + l_rug() + l_coordContour() + l_bound()

# Plotting as in standard 2D plots
plot(sm(bp, 1), scheme=1) + l_fitRaster() + l_fitContour() +
           l_points(shape = 19) + l_rug()
```

plotDiff *Generic plotting of differences*

Description

Generic function for plotting differences between objects. Useful mainly for plotting the differences between two smooth effects.

Usage

```
plotDiff(...)
```

Arguments

`...`	arguments to be passed to methods. This first one will determine which method will be called.

See Also

plotDiff.mgcv.smooth.1D, plotDiff.mgcv.smooth.2D, plotDiff.sos.smooth

---

`plotDiff.mgcv.smooth.1D` *Plotting differences between two 1D smooth effects*

Description

This method can be used to plot the difference between two 1D smooth effects. Mainly meant to be used with by-factor smooths.

Usage

```
## S3 method for class 'mgcv.smooth.1D'
plotDiff(s1, s2, n = 100, trans = identity, unconditional = FALSE, ...)
```

Arguments

`s1`	a smooth effect object, extracted using sm.
`s2`	another smooth effect object.
`n`	number of grid points used to compute main effects and c.i. lines. For a nice smooth plot this needs to be several times the estimated degrees of freedom for the smooth.
`trans`	monotonic function to apply to the smooth and residuals, before plotting. Monotonicity is not checked.
`unconditional`	if `TRUE` then the smoothing parameter uncertainty corrected covariance matrix is used to compute uncertainty bands, if available. Otherwise the bands treat the smoothing parameters as fixed.
`...`	currently unused.

Details

Let sd be the difference between the fitted smooths, that is: sd = s1 - s2. sd is a vector of length n, and its covariance matrix is Cov(sd) = X1\ where: X1 (X2) and Sig11 (Sig22) are the design matrix and the covariance matrix of the coefficients of s1 (s2), while Sig12 is the cross-covariance matrix between the coefficients of s1 and s2. To get the confidence intervals we need only diag(Cov(sd)), which here is calculated efficiently (without computing the whole of Cov(sd)).

Value

An objects of class `plotSmooth`.

References

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

Examples

```
# Simulate data and add factors uncorrelated to the response
library(mgcViz)
set.seed(6898)
dat <- gamSim(1,n=1500,dist="normal",scale=20)
dat$fac <- as.factor( sample(c("A1", "A2", "A3"), nrow(dat), replace = TRUE) )
dat$logi <- as.logical( sample(c(TRUE, FALSE), nrow(dat), replace = TRUE) )
bs <- "cr"; k <- 12
b <- gam(y ~ s(x2,bs=bs,by = fac), data=dat)
o <- getViz(b, nsim = 0)

# Extract the smooths correspoding to "A1" and "A2" and plot their differences
# with credible intervals
plotDiff(s1 = sm(o, 1), s2 = sm(o, 2)) + l_ciPoly() +
  l_fitLine() + geom_hline(yintercept = 0, linetype = 2)
```

`plotDiff.mgcv.smooth.2D`

Plotting differences between two 2D smooth effects

Description

This method can be used to plot the difference between two 2D smooth effects. Mainly meant to be used with by-factor smooths.

Usage

```
## S3 method for class 'mgcv.smooth.2D'
plotDiff(
  s1,
  s2,
  n = 40,
  too.far = 0.1,
  trans = identity,
  unconditional = FALSE,
  ...
)
```

Arguments

`s1`	a smooth effect object, extracted using sm.
`s2`	another smooth effect object.
`n`	sqrt of the number of grid points used to compute the effect plot.
`too.far`	if greater than 0 then this is used to determine when a location is too far from data to be plotted. This is useful since smooths tend to go wild away from data. The data are scaled into the unit square before deciding what to exclude, and

	too.far is a distance within the unit square. Setting to zero can make plotting faster for large datasets, but care then needed with interpretation of plots.
`trans`	monotonic function to apply to the smooth and residuals, before plotting. Monotonicity is not checked.
`unconditional`	if `TRUE` then the smoothing parameter uncertainty corrected covariance matrix is used to compute uncertainty bands, if available. Otherwise the bands treat the smoothing parameters as fixed.
`...`	currently unused.

Details

Let sd be the difference between the fitted smooths, that is: sd = s1 - s2. sd is a vector of length n, and its covariance matrix is Cov(sd) = X1\ where: X1 (X2) and Sig11 (Sig22) are the design matrix and the covariance matrix of the coefficients of s1 (s2), while Sig12 is the cross-covariance matrix between the coefficients of s1 and s2. To get the confidence intervals we need only diag(Cov(sd)), which here is calculated efficiently (without computing the whole of Cov(sd)).

Value

An objects of class `plotSmooth`.

References

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

Examples

```
# Simulate data and add factors uncorrelated to the response
library(mgcViz)
set.seed(235)
dat <- gamSim(1,n=1500,dist="normal",scale=20)
dat$fac <- as.factor( sample(c("A1", "A2", "A3"), nrow(dat), replace = TRUE) )
dat$logi <- as.logical( sample(c(TRUE, FALSE), nrow(dat), replace = TRUE) )
bs <- "cr"; k <- 12
b <- gam(y ~ s(x2, x1, by = fac), data=dat)
o <- getViz(b, nsim = 0)

# Extract the smooths correspoding to "A1" and "A2" and plot their difference
pl <- plotDiff(s1 = sm(o, 1), s2 = sm(o, 2))
pl + l_fitRaster() + l_fitContour()

# Plot p-values for differences between the two smooths
pl + l_pvRaster() + l_pvContour()
```

plotDiff.sos.smooth *Plotting differences between two smooths on the sphere*

Description

This method can be used to plot the difference between two smooth effects on the sphere. Mainly meant to be used with by-factor smooths.

Usage

```
## S3 method for class 'sos.smooth'
plotDiff(
  s1,
  s2,
  n = 40,
  too.far = 0.1,
  phi = 30,
  theta = 30,
  scheme = 0,
  trans = identity,
  unconditional = FALSE,
  ...
)
```

Arguments

`s1`	a smooth effect object, extracted using sm.
`s2`	another smooth effect object.
`n`	sqrt of the number of grid points used to compute the effect plot.
`too.far`	if greater than 0 then this is used to determine when a location is too far from data to be plotted. This is useful since smooths tend to go wild away from data. The data are scaled into the unit square before deciding what to exclude, and too.far is a distance within the unit square. Setting to zero can make plotting faster for large datasets, but care then needed with interpretation of plots.
`phi`	one of the plotting angles, relevant only if `scheme = 0`.
`theta`	the other plotting angle, relevant only if `scheme = 0`.
`scheme`	if 0 the smooth effect is plotted on the sphere. If 1 the smooth effect is plotted on the two hemispheres.
`trans`	monotonic function to apply to the smooth and residuals, before plotting. Monotonicity is not checked.
`unconditional`	if `TRUE` then the smoothing parameter uncertainty corrected covariance matrix is used to compute uncertainty bands, if available. Otherwise the bands treat the smoothing parameters as fixed.
`...`	currently unused.

Details

Let sd be the difference between the fitted smooths, that is: sd = s1 - s2. sd is a vector of length n, and its covariance matrix is Cov(sd) = X1\ where: X1 (X2) and Sig11 (Sig22) are the design matrix and the covariance matrix of the coefficients of s1 (s2), while Sig12 is the cross-covariance matrix between the coefficients of s1 and s2. To get the confidence intervals we need only diag(Cov(sd)), which here is calculated efficiently (without computing the whole of Cov(sd)).

Value

An objects of class `plotSmooth`.

References

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

Examples

```
## Not run:
#### 1) Simulate data and add factors uncorrelated to the response
library(mgcViz)
set.seed(0)
n <- 500

f <- function(la,lo) { ## a test function...
  sin(lo)*cos(la-.3)
}

## generate with uniform density on sphere...
lo <- runif(n)*2*pi-pi ## longitude
la <- runif(3*n)*pi-pi/2
ind <- runif(3*n)<=cos(la)
la <- la[ind];
la <- la[1:n]

ff <- f(la,lo)
y <- ff + rnorm(n)*.2 ## test data

## generate data for plotting truth...
lam <- seq(-pi/2,pi/2,length=30)
lom <- seq(-pi,pi,length=60)
gr <- expand.grid(la=lam,lo=lom)
fz <- f(gr$la,gr$lo)
zm <- matrix(fz,30,60)

dat <- data.frame(la = la *180/pi,lo = lo *180/pi,y=y)
dat$fac <- as.factor( sample(c("A1", "A2", "A3"), nrow(dat), replace = TRUE) )

#### 2) fit spline on sphere model...
bp <- gam(y~s(la,lo,bs="sos",k=60, by = fac),data=dat)
bp <- getViz(bp)
```

```
# Extract the smooths correspoding to "A1" and "A3" and plot their difference
# Using scheme = 0
pl0 <- plotDiff(s1 = sm(bp, 1), s2 = sm(bp, 3))
pl0 + l_fitRaster() + l_fitContour() + l_coordContour() + l_bound()

# Plot p-values for significance of differences
pl0 + l_pvRaster() + l_pvContour(breaks=c(0.05, 0.1, 0.2, 0.3, 0.5))

# Using scheme = 1
pl1 <- plotDiff(s1 = sm(bp, 1), s2 = sm(bp, 2), scheme = 1)
pl1 + l_fitRaster() + l_fitContour()

# Plot p-values for significance of differences
pl1 + l_pvRaster() + l_pvContour(breaks=c(0.05, 0.1, 0.2, 0.3, 0.5))

## End(Not run)
```

`plotRGL` *Generic RGL plotting function*

Description

Generic function for producing an interactive RGL plot.

Usage

```
plotRGL(x, ...)
```

Arguments

`x`	the object we want to plot using the `rgl` package.
`...`	arguments to be passed to methods.

See Also

plotRGL.mgcv.smooth.2D, plotRGL.mgcv.smooth.MD

plotRGL.mgcv.smooth.2D

Visualizing 2D smooth effects in 3D

Description

This method plots an interactive 3D representation of a 2D smooth effect, using the rgl package.

Usage

```
## S3 method for class 'mgcv.smooth.2D'
plotRGL(
  x,
  se = TRUE,
  n = 40,
  residuals = FALSE,
  type = "auto",
  maxpo = 1000,
  too.far = 0,
  xlab = NULL,
  ylab = NULL,
  main = NULL,
  xlim = NULL,
  ylim = NULL,
  se.mult = 1,
  trans = identity,
  seWithMean = FALSE,
  unconditional = FALSE,
  ...
)
```

Arguments

`x`	a smooth effect object, extracted using sm.
`se`	when TRUE (default) upper and lower surfaces are added to the plot at `se.mult` (see below) standard deviations for the fitted surface.
`n`	sqrt of the number of grid points used to compute the effect plot.
`residuals`	if TRUE, then the partial residuals will be added.
`type`	the type of residuals that should be plotted. See residuals.gamViz.
`maxpo`	maximum number of residuals points that will be plotted. If number of datapoints > `maxpo`, then a subsample of `maxpo` points will be taken.
`too.far`	if greater than 0 then this is used to determine when a location is too far from data to be plotted. This is useful since smooths tend to go wild away from data. The data are scaled into the unit square before deciding what to exclude, and too.far is a distance within the unit square. Setting to zero can make plotting faster for large datasets, but care is then needed when interpreting the plots.

`xlab`	if supplied then this will be used as the x label of the plot.
`ylab`	if supplied then this will be used as the y label of the plot.
`main`	used as title for the plot if supplied.
`xlim`	if supplied then this pair of numbers are used as the x limits for the plot.
`ylim`	if supplied then this pair of numbers are used as the y limits for the plot.
`se.mult`	a positive number which will be the multiplier of the standard errors when calculating standard error surfaces.
`trans`	monotonic function to apply to the smooth and residuals, before plotting. Monotonicity is not checked.
`seWithMean`	if TRUE the component smooths are shown with confidence intervals that include the uncertainty about the overall mean. If FALSE then the uncertainty relates purely to the centred smooth itself. Marra and Wood (2012) suggests that TRUE results in better coverage performance, and this is also suggested by simulation.
`unconditional`	if `TRUE` then the smoothing parameter uncertainty corrected covariance matrix is used to compute uncertainty bands, if available. Otherwise the bands treat the smoothing parameters as fixed.
`...`	currently unused.

Value

Returns `NULL` invisibly.

References

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

Examples

```
# Example 1: taken from ?mgcv::te, shows how tensor pruduct deals nicely with
# badly scaled covariates (range of x 5% of range of z )
library(mgcViz)

# Simulate some data
test1 <- function(x,z,sx=0.3,sz=0.4) {
  x <- x*20
  (pi**sx*sz)*(1.2*exp(-(x-0.2)^2/sx^2-(z-0.3)^2/sz^2)+
                 0.8*exp(-(x-0.7)^2/sx^2-(z-0.8)^2/sz^2))
}
n <- 500
old.par <- par(mfrow=c(2,2))
x <- runif(n)/20;z <- runif(n);
xs <- seq(0,1,length=30)/20;zs <- seq(0,1,length=30)
pr <- data.frame(x=rep(xs,30),z=rep(zs,rep(30,30)))
truth <- matrix(test1(pr$x,pr$z),30,30)
f <- test1(x,z)
y <- f + rnorm(n)*0.2
```

```
# Fit with t.p.r.s. basis and plot
b1 <- gam(y~s(x,z))
plotRGL(sm(getViz(b1), 1))

# Need to load rgl at this point
## Not run:
library(rgl)
rgl.close() # Close

# Fit with tensor products basis and plot (with residuals)
b2 <- gam(y~te(x,z))
plotRGL(sm(getViz(b2), 1), residuals = TRUE)

# We can still work on the plot, for instance change the aspect ratio
aspect3d(1, 2, 1)

rgl.close() # Close

## End(Not run)
```

`plotRGL.mgcv.smooth.MD`

Visualizing a 2D slice of a smooth effects in 3D

Description

This method plots an interactive 3D representation of a 2-dimensional slice of a multi-dimensional smooth effect, using the rgl package.

Usage

```
## S3 method for class 'mgcv.smooth.MD'
plotRGL(
  x,
  fix,
  se = TRUE,
  n = 40,
  residuals = FALSE,
  type = "auto",
  maxpo = 1000,
  too.far = c(0, NA),
  xlab = NULL,
  ylab = NULL,
  main = NULL,
  xlim = NULL,
  ylim = NULL,
  se.mult = 1,
  trans = identity,
  seWithMean = FALSE,
  unconditional = FALSE,
  ...
)
```

Arguments

`x`	a smooth effect object, extracted using sm.
`fix`	a named vector indicating which variables must be kept fixed and to what values. When plotting a smooth in (d+2) dimensions, then d variables must be fixed.
`se`	when TRUE (default) upper and lower surfaces are added to the plot at `se.mult` (see below) standard deviations for the fitted surface.
`n`	sqrt of the number of grid points used to compute the effect plot.
`residuals`	if TRUE, then the partial residuals will be added.
`type`	the type of residuals that should be plotted. See residuals.gamViz.
`maxpo`	maximum number of residuals points that will be plotted. If number of datapoints > `maxpo`, then a subsample of `maxpo` points will be taken.
`too.far`	a numeric vector with two entries. The first has the same interpretation as in plot.mgcv.smooth.2D and it avoids plotting the smooth effect in areas that are too far form any observation. The distance will be calculated only using the variables which are not in `fix` (see above). Hence in two dimensions, not in the full d+2 dimensions. Set it to -1 to plot the whole smooth. The second entry determines which residuals and covariates pairs are closed enough to the selected slice. If left to `NA` on the 10\ closest (in terms of scaled Euclidean distance) to the current slice will be plotted. Set it to -1 to plot all the residuals.
`xlab`	if supplied then this will be used as the x label of the plot.
`ylab`	if supplied then this will be used as the y label of the plot.
`main`	used as title for the plot if supplied.
`xlim`	if supplied then this pair of numbers are used as the x limits for the plot.
`ylim`	if supplied then this pair of numbers are used as the y limits for the plot.
`se.mult`	a positive number which will be the multiplier of the standard errors when calculating standard error surfaces.
`trans`	monotonic function to apply to the smooth and residuals, before plotting. Monotonicity is not checked.
`seWithMean`	if TRUE the component smooths are shown with confidence intervals that include the uncertainty about the overall mean. If FALSE then the uncertainty relates purely to the centred smooth itself. Marra and Wood (2012) suggests that TRUE results in better coverage performance, and this is also suggested by simulation.
`unconditional`	if `TRUE` then the smoothing parameter uncertainty corrected covariance matrix is used to compute uncertainty bands, if available. Otherwise the bands treat the smoothing parameters as fixed.
`...`	currently unused.

Value

Returns `NULL` invisibly.

References

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

Examples

```
# Example 1: taken from ?mgcv::te, shows how tensor pruduct deals nicely with
# badly scaled covariates (range of x 5% of range of z )
library(mgcViz)
n <- 1e3
x <- rnorm(n); y <- rnorm(n); z <- rnorm(n)

ob <- (x-z)^2 + (y-z)^2 + rnorm(n)
b <- gam(ob ~ s(x, y, z))
v <- getViz(b)

plotRGL(sm(v, 1), fix = c("z" = 0))

# Need to load rgl at this point
## Not run:
library(rgl)
rgl.close() # Close

plotRGL(sm(v, 1), fix = c("z" = 1), residuals = TRUE)

# We can still work on the plot, for instance change the aspect ratio
aspect3d(1, 2, 1)

rgl.close() # Close

## End(Not run)
```

`plotSlice` *Plotting sequence of slices of 2D smooth effect*

Description

This function allows to slice a multi-dimensional (D > 2) smooth effect, and to plot the resulting sequence of 2D slices in an array of plots.

Usage

```
plotSlice(x, fix, a.facet = list(), ...)
```

Arguments

`x`	a smooth effect object, extracted using sm.
`fix`	a named list of vectors, where the i-th entry of each vector indicates the value we want to use for the covariate for i-th slice. When plotting a smooth in (d+2) dimensions, we need d vectors, because d variables must be fixed. All vectors must have either the same length (the number of slices) or length 1. `fix` can contain at most 2 vectors, so if d>=5, we need to set at least one covariate to a scalar.
`a.facet`	arguments to be passed to ggplot2::facet_wrap or ggplot2::facet_grid. The former gets called when `fix` contains one vector, the latter when `fix` contains two vectors.
`...`	further arguments to be passed to plot.mgcv.smooth.MD.

Value

An objects of class `plotSmooth`.

Examples

```
## Not run:
### Example 1: plotting slices of 3D smooth
# Simulate data and fit GAM
library(mgcViz)
n <- 1e3
x <- rnorm(n); y <- rnorm(n); z <- rnorm(n)
ob <- (x-z)^2 + (y-z)^2 + rnorm(n)
b <- gam(ob ~ s(x, y, z))
v <- getViz(b)

# Get plot of slices and add layers
pl <- plotSlice(x = sm(v, 1),
                fix = list("z" = seq(-2, 2, length.out = 9)))
pl + l_fitRaster() + l_fitContour() + l_points() + l_rug()

# Over-ride default layout
pl <- plotSlice(x = sm(v, 1),
                fix = list("z" = seq(-2, 2, length.out = 9)),
                a.facet = list(nrow = 2))
pl + l_fitRaster() + l_fitContour() + theme(panel.spacing = unit(0.5, "lines"))

### Example 2: plotting slices of 4D smooth
# Simulate data and fit GAM
n <- 5e3
x <- rnorm(n); y <- rnorm(n); z <- rnorm(n); z2 <- rnorm(n)
ob <- (x-z)^2 + (y-z)^2 + z2^3 + rnorm(n)
b <- bam(ob ~ s(x, y, z, z2), discrete = TRUE)
v <- getViz(b)

# Plot slices across "z" and "x"
pl <- plotSlice(x = sm(v, 1),
                fix = list("z" = seq(-2, 2, length.out = 3), "x" = c(-1, 0, 1)))
pl + l_fitRaster() + l_fitContour() + l_points() + l_rug()

# Plot slices across "x", keeping "z" fixed
pl <- plotSlice(x = sm(v, 1),
                fix = list("z" = 0, "x" = seq(-3, 3, length.out = 9)))
pl + l_fitRaster() + l_fitContour() + l_points() + l_rug()

## End(Not run)
```

postSim *Posterior simulation from a GAM object*

Description

This method can be used to simulate vectors of responses from the Gaussian posterior approximation of a gamObject.

Usage

```
postSim(
  o,
  nsim,
  newdata,
  trans = NULL,
  method = "auto",
  w = NULL,
  offset = NULL,
  savePar = FALSE,
  ...
)
```

Arguments

`o`	the output of a `gam()` or `bam()` call.
`nsim`	the number of simulated vectors of responses. A positive integer.
`newdata`	Optional new data frame used to perform the simulations. To be passed to predict.gam.
`trans`	function used to transform or summarize each vector of simulated responses. It must take a vector as argument, but it can output a vector or a scalar. Potentially useful for saving storage (e.g. by transforming each simulated vector to a scalar). If left to `NULL` then `trans = identity` will be used.
`method`	the method used for the simulation of responses. See simulate.gam.
`w`	vector of prior weights of each response. See simulate.gam.

`offset`	numeric vector of offsets. For GAMs with multiple linear predictor (see eg gaulss) it must be a list of vectors. If `newdata!=NULL` the offsets will be assumed to be zero, unless their are explicitly provided. If `newdata==NULL` the simulations will use the offsets used during model fitting, unless offset is explicitly provided.
`savePar`	if `TRUE` than also the simulated parameters will be returned.
`...`	arguments to be passed to vcov.gam.

Value

If `savePar == FALSE` the function will return a matrix where each column is a vector of simulated responses or a transformed version of it. If `savePar == TRUE` it will return a list where the `$simY` entry will contain the simulated responses and `$simBeta` the simulated parameters.

Examples

```
library(mgcViz)
library(MASS)
b <- gam(accel~s(times, k=20), data=mcycle)

# Simulate list of 10 vectors of responses from posterior, taking into
# account smoothing parameters uncertainty (see ?vcov.gam)
n <- 10
sim <- postSim(o = b, nsim = n, unconditional = TRUE)

# Posterior simulations in grey and data in red
plot(rep(mcycle$times, n), as.vector(sim), col = "grey",
     ylab = "Acceleration", xlab = "Times")
points(mcycle$times, mcycle$accel, col = 2)

# There is clear disagreement between simulations' and data's
# conditional variance, which can be solved using flexible GAMLSS model:
b <- gam(list(accel~s(times, k=20), ~s(times)), data=mcycle, family = gaulss)
sim <- postSim(o = b, nsim = n)
plot(rep(mcycle$times, n), as.vector(sim), col = "grey",
     ylab = "Acceleration", xlab = "Times")
points(mcycle$times, mcycle$accel, col = 2)
```

`print.checkGam` *Printing the output of check.gamViz*

Description

This method prints the output of check.gamViz.

Usage

```
## S3 method for class 'checkGam'
print(x, lay = NULL, ...)
```

Arguments

`x`	the output of `check.gamViz`.
`lay`	the `layout_matrix` passed to gridExtra::grid.arrange.
`...`	further arguments to be passed to `grid.arrange`.

Value

Returns the output of `grid.arrange`, invisibly.

`print.plotGam` *Printing the output of plot.gamViz*

Description

This method prints the output of plot.gamViz.

Usage

```
## S3 method for class 'plotGam'
print(x, ask = TRUE, pages = NULL, addLay = TRUE, ...)
```

Arguments

`x`	an object of class `plotGam`.
`ask`	should we ask before moving from one page to the next?
`pages`	the number of pages over which to spread the output.
`addLay`	if TRUE, and if the `$empty` slot of the `plotGam` object is TRUE, we add some default layers to the plots, before printing. Does not have any affect if the `plotGam` object already contains some layers.
`...`	further arguments to be passed to `grid.arrange`.

Value

Returns the output of gridExtra::grid.arrange, invisibly.

print.plotSmooth *Printing plots of smooth effects*

Description

This method prints objects of class plotSmooth.

Usage

```
## S3 method for class 'plotSmooth'
print(x, addLay = TRUE, ...)
```

Arguments

x	an object of class plotSmooth.
addLay	if TRUE, and if the $empty slot of the plotSmooth object is TRUE or NULL, we add some default layers to the plots, before printing. Does not have any affect if the plotSmooth object already contains some layers (e.g. l_rug()).
...	currently unused.

Value

Returns NULL, invisibly.

print.qqGam *Printing the output of qq.gamViz*

Description

This method prints the output of qq.gamViz.

Usage

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

Arguments

x	an object of class qqGam.
...	currently unused.

Value

Returns NULL, invisibly.

pterm *Extracting parametric effects from a GAM model*

Description

This function can be used to extract a parametric effect from an object of class `gamViz`.

Usage

```
pterm(o, select)
```

Arguments

`o`	an object of class `gamViz`, the output of a `getViz()` call.
`select`	index of the selected parametric effect.

Value

An object of class "pTermSomething" where "Something" is substituted with the class of the variable of interest. For instance if this "numeric", the `pterm` will return an object of class "ptermNumeric".

Examples

```
####### 1. Gaussian GAM
library(mgcViz)
set.seed(3)
dat <- gamSim(1,n=1500,dist="normal",scale=20)
dat$fac <- as.factor( sample(c("A1", "A2", "A3"), nrow(dat), replace = TRUE) )
dat$logi <- as.logical( sample(c(TRUE, FALSE), nrow(dat), replace = TRUE) )
bs <- "cr"; k <- 12
b <- gam(y ~ x0 + x1 + I(x1^2) + s(x2,bs=bs,k=k) + fac + x3:fac + I(x1*x2) + logi,data=dat)
o <- getViz(b)

# Plot effect of 'x0'
pt <- pterm(o, 1)
plot(pt, n = 60) + l_ciPoly() + l_fitLine() + l_ciLine() + l_points()

# Plot effect of 'x3'
pt <- pterm(o, 1)
plot(pt, n = 60) + l_fitLine() + l_ciLine(colour = 2)

# Plot effect of 'fac'
pt <- pterm(o, 4)
plot(pt) + l_ciBar(colour = "blue") + l_fitPoints(colour = "red") +
           l_rug(alpha = 0.3)

# Plot effect of 'logi'
pt <- pterm(o, 6)
```

```
plot(pt) + l_fitBar(a.aes = list(fill = I("light blue"))) + l_ciBar(colour = "blue")

# Plot effect of 'x3:fac': no method available yet available for second order terms
pt <- pterm(o, 7)
plot(pt)

## Not run:
####### 1. Continued: Quantile GAMs
b <- mqgamV(y ~ x0 + x1 + I(x1^2) + s(x2,bs=bs,k=k) + x3:fac +
              I(x1*x2) + logi, data=dat, qu = c(0.3, 0.5, 0.8))

plot(pterm(b, 3)) + l_ciBar(colour = 2) + l_fitPoints()

plot(pterm(b, 4)) + l_fitBar(colour = "blue", fill = 3) + l_ciBar(colour = 2)

# Don't know how to plot this interaction
plot(pterm(b, 6))

####### 2. Gaussian GAMLSS model
library(MASS)
mcycle$fac <- as.factor( sample(c("z", "k", "a", "f"), nrow(mcycle), replace = TRUE) )
b <- gam(list(accel~times + I(times^2) + s(times,k=10), ~ times + fac + s(times)),
          data=mcycle,family=gaulss(), optimizer = "efs")
o <- getViz(b)

# Plot effect of 'I(times^2)' on mean: notice that partial residuals
# are unavailable for GAMLSS models, hence l_point does not do anything here.
pt <- pterm(o, 2)
plot(pt) + l_ciPoly() + l_fitLine() + l_ciLine() + l_points()

# Plot effect of 'times' in second linear predictor.
# Notice that partial residuals are unavailable.
pt <- pterm(o, 3)
plot(pt) + l_ciPoly() + l_fitLine() + l_ciLine(linetype = 3) + l_rug()

# Plot effect of 'fac' in second linear predictor.
pt <- pterm(o, 4)
plot(pt) + l_ciBar(colour = "blue") + l_fitPoints(colour = "red") +
           l_rug()

## End(Not run)
```

---

qgamV *Fit a QGAM model and get a gamViz object*

Description

These are wrapper that fits a QGAM model using qgam::qgam and converts it to a `gamViz` object using the getViz function. It is essentially a shortcut.

Usage

```
qgamV(form, data, qu, lsig = NULL, err = NULL, aQgam = list(), aViz = list())
```

Arguments

`form, data, qu, lsig, err`	same arguments as in qgam::qgam.
`aQgam`	list of further arguments to be passed to qgam::qgam.
`aViz`	list of arguments to be passed to getViz.

Value

An object of class "gamViz" which can, for instance, be plotted using plot.gamViz.

Examples

```
library(mgcViz)
set.seed(2) ## simulate some data...
dat <- gamSim(2,n=1000,dist="normal",scale=0.25)$data

# Fit GAM and get gamViz object
b <- qgamV(y~s(x) + s(z) + I(x*z), data = dat, qu = 0.2,
           aQgam = list(argGam = list(select = TRUE)), aViz = list("nsim" = 0))

# This is equivalent to doing
# 1. Fit QGAM
# b <- qgam(y~s(x) + s(z) + I(x*z), data=dat, qu = 0.2, argGam = list(select = TRUE))
# 2. Convert to gamViz object
# b <- getViz(b, nsim = 0)

# Either way, we all effects by doing
print(plot(b, allTerms = TRUE), pages = 1)
```

`qq` *Generic QQ plots*

Description

Generic function for producing QQ-plots.

Usage

```
qq(...)
```

Arguments

`...`	arguments to be passed to methods. This first one will determine which method will be called.

See Also

qq.gamViz

`qq.gamViz` *QQ plots for gam model residuals*

Description

Takes a fitted gam object, converted using getViz, and produces QQ plots of its residuals (conditional on the fitted model coefficients and scale parameter). If the model distributional assumptions are met then usually these plots should be close to a straight line (although discrete data can yield marked random departures from this line).

Usage

```
## S3 method for class 'gamViz'
qq(
  o,
  rep = 10,
  level = 0.8,
  method = "auto",
  type = "auto",
  CI = "none",
  worm = FALSE,
  showReps = FALSE,
  sortFun = NULL,
  discrete = NULL,
  ngr = 1000,
  xlim = NULL,
  ylim = NULL,
  a.qqpoi = list(),
  a.ablin = list(),
  a.cipoly = list(),
  a.replin = list(),
  ...
)
```

Arguments

`o`	an object of class `gamViz`, the output of a `getViz()` call.
`rep`	how many replicate datasets to generate to simulate quantiles of the residual distribution. Relevant only if `method` is set to `"simul1"` or `"simul2"`.
`level`	the level of the confidence intervals (e.g. 0.9 means 90% intervals).

`method`	the method used to calculate the QQ-plot and, possibly, the confidence intervals. If set to (`"tunif"`) `"tnormal"` the residuals are transformed to (uniform) normal, for which analytic expression for the confidence intervals are available. If set to `"simul1"` or `"simul2"` the theoretical QQ-line is constructed by simulating residuals from the model. Method `"simul2"` does not produce confidence intervals. If set to `"normal"` no simulation or transformation is performed, and a simple normal QQ-plot is produced. If set to `"auto"` the method used to produce the QQ-plot is determined automatically.
`type`	the type of residuals to be used. See residuals.gamViz.
`CI`	the type of confidence intervals to be plotted. If set to `"none"` they are not added, if set to `"normal"` they will be based on the assumption that the theoretical quantile distribution is Gaussian and if set to `"quantile"` they will be sample quantiles of simulated responses from the model.
`worm`	if `TRUE` a worm-plot (a de-trended QQ-plot) is plotted.
`showReps`	if `TRUE` all the QQ-lines corresponding to the simulated (model-based) QQ-plots.
`sortFun`	the function to be used for sorting the residuals. If left to `NULL` it will be set to `function(.x) sort(.x,method = "quick")` internally.
`discrete`	if `TRUE` the QQ-plot is discretized into `ngr` bins before plotting, in order to save plotting time (when the number of observations is large). If left to `NULL`, the discretization is used if there are more than 10^4 observations.
`ngr`	number of bins to be used in the discretization.
`xlim`	if supplied then this pair of numbers are used as the x limits for the plot.
`ylim`	if supplied then this pair of numbers are used as the y limits for the plot.
`a.qqpoi`	list of arguments to be passed to `ggplot2::geom_point`, which plots the main QQ-plot.
`a.ablin`	list of arguments to be passed to `ggplot2::geom_abline`, which adds the reference line.
`a.cipoly`	list of arguments to be passed to `ggplot2::geom_polygon`, which add the confidence intervals.
`a.replin`	list of arguments to be passed to `ggplot2::geom_line`, which adds a line for each simulated QQ-plot.
`...`	currently unused.

Details

Here `method = "simul1"` corresponds to the algorithm described in section 2.1 of Augustin et al. (2012), which involves direct simulations of residuals from the models. This requires `o$family$rd` to be defined. Setting `method = "simul2"` results in a cheaper method, described in section 2.2 of Augustin et al. (2012), which requires `o$family$qf` to be defined.

Value

An object of class `c("qqGam","plotSmooth","gg")`.

References

Augustin, N.H., Sauleau, E.A. and Wood, S.N., 2012. On quantile quantile plots for generalized linear models. Computational Statistics & Data Analysis, 56(8), pp.2404-2409.

Examples

```
######## Example: simulate binomial data
library(mgcViz)
set.seed(0)
n.samp <- 400
dat <- gamSim(1,n = n.samp, dist = "binary", scale = .33)
p <- binomial()$linkinv(dat$f) ## binomial p
n <- sample(c(1, 3), n.samp, replace = TRUE) ## binomial n
dat$y <- rbinom(n, n, p)
dat$n <- n
lr.fit <- gam(y/n ~ s(x0) + s(x1) + s(x2) + s(x3)
              , family = binomial, data = dat,
              weights = n, method = "REML")
lr.fit <- getViz(lr.fit)

# Quick QQ-plot of deviance residuals
qq(lr.fit, method = "simul2")

# Same, but changing points share and type of reference list
qq(lr.fit, method = "simul2",
       a.qqpoi = list("shape" = 1), a.ablin = list("linetype" = 2))

# Simulation based QQ-plot with reference bands
qq(lr.fit, rep = 100, level = .9, CI = "quantile")

# Simulation based QQ-plot, Pearson resids, all simulations lines shown
qq(lr.fit, rep = 100, CI = "none", showReps = TRUE, type = "pearson",
       a.qqpoi = list(shape=19, size = 0.5))

### Now fit the wrong model and check
pif <- gam(y ~ s(x0) + s(x1) + s(x2) + s(x3)
           , family = poisson, data = dat, method = "REML")
pif <- getViz(pif)

qq(pif, method = "simul2")

qq(pif, rep = 100, level = .9, CI = "quantile")

qq(pif, rep = 100, type = "pearson", CI = "none", showReps = TRUE,
               a.qqpoi = list(shape=19, size = 0.5))

######## Example: binary data model violation so gross that you see a problem
######## on the QQ plot
y <- c(rep(1, 10), rep(0, 20), rep(1, 40), rep(0, 10), rep(1, 40), rep(0, 40))
x <- 1:160
b <- glm(y ~ x, family = binomial)
class(b) <- c("gamViz", class(b)) # Tricking qq.gamViz to use it on a glm
```

```
# Note that the next two are not necessarily similar under gross
# model violation...
qq(b, method = "simul2")
qq(b, rep = 50, CI = "none", showReps = TRUE)

### alternative model
b <- gam(y ~ s(x, k = 5), family = binomial, method = "ML")
b <- getViz(b)

qq(b, method = "simul2")
qq(b, rep = 50, showReps = TRUE, CI = "none", shape = 19)

## Not run:
########  "Big Data" example:
set.seed(0)
n.samp <- 50000
dat <- gamSim(1,n=n.samp,dist="binary",scale=.33)
p <- binomial()$linkinv(dat$f) ## binomial p
n <- sample(c(1,3),n.samp,replace=TRUE) ## binomial n
dat$y <- rbinom(n,n,p)
dat$n <- n
lr.fit <- bam(y/n ~ s(x0) + s(x1) + s(x2) + s(x3)
              , family = binomial, data = dat,
              weights = n, method = "fREML", discrete = TRUE)
lr.fit <- getViz(lr.fit)

# Turning discretization off (on by default for large datasets).
set.seed(414) # Setting the seed because qq.gamViz is doing simulations
o <- qq(lr.fit, rep = 10, method = "simul1", CI = "normal", showReps = TRUE,
            discrete = F, a.replin = list(alpha = 0.1))
o # This might take some time!

# Using default discretization
set.seed(414)
o <- qq(lr.fit, rep = 10, method = "simul1", CI = "normal", showReps = TRUE,
            a.replin = list(alpha = 0.1))
o # Much faster plotting!

# Very coarse discretization
set.seed(414)
o <- qq(lr.fit, rep = 10, method = "simul1", CI = "normal", showReps = TRUE,
            ngr = 1e2, a.replin = list(alpha = 0.1), a.qqpoi = list(shape = 19))
o

# We can also zoom in at no extra costs (most work already done by qq.gamViz)
zoom(o, xlim = c(-0.25, 0.25), showReps = TRUE, discrete = TRUE, a.replin = list(alpha = 0.2))

## End(Not run)
```

`qqplots` *Quantile-Quantile Plots*

Description

This is a re-write of the QQ-plotting functions provided by `stats`, using the `ggplot2` library. `qqnorm` is a generic function the default method of which produces a normal QQ plot of the values in `y`. `qqline` adds a line to a “theoretical”, by default normal, quantile-quantile plot which passes through the `probs` quantiles, by default the first and third quartiles. `qqplot` produces a QQ plot of two datasets.

Usage

```
qqnorm(
  y,
  ylim,
  main = "Normal Q-Q Plot",
  xlab = "Theoretical Quantiles",
  ylab = "Sample Quantiles",
  datax = FALSE
)

qqplot(
  x,
  y,
  xlab = deparse(substitute(x)),
  ylab = deparse(substitute(y)),
  main = "Q-Q Plot"
)

qqline(
  y,
  datax = FALSE,
  distribution = qnorm,
  probs = c(0.25, 0.75),
  qtype = 7,
  ...
)
```

Arguments

`y,`	The second or only data sample.
`ylim, ...,`	Graphical parameters.
`main, xlab, ylab,`	Plot labels. The xlab and ylab refer to the y and x axes respectively if datax = TRUE.

`datax,`	Logical. Should data values be on the x-axis ?
`x,`	The first sample for `qqplot`.
`distribution`	quantile function for reference theoretical distribution.
`probs`	numeric vector of length two, representing probabilities. Corresponding quantile pairs define the line drawn.
`qtype`	the type of quantile computation used in quantile.

Note

Help file is mainly from `stats::qqnorm` since this is a rewrite of `stats::qqplot`, `stats::qqline` and `stats::qqnorm` using the ggplot2 library.

Examples

```
library(mgcViz)
y <- rt(500, df = 5)

# Compare new and old version of qqnorm
stats::qqnorm(y)
qqnorm(y)

# Compare new and old version of qqplot
x <- rt(200, df = 5)
y <- rt(300, df = 5)
stats::qqplot(x, y)
qqplot(x, y)
# add a qqline()
ggplot2::last_plot() + qqline(y = rt(500, df = 4.8), col = "green")

## "QQ-Chisquare" : --------------------------
y <- rchisq(500, df = 3)
## Q-Q plot for Chi^2 data against true theoretical distribution:
x <- qchisq(ppoints(500), df = 3)
stats::qqplot(qchisq(ppoints(500), df = 3), rchisq(500, df = 3),
      main = expression("Q-Q plot for" ~~ {chi^2}[nu == 3]))
qqplot(qchisq(ppoints(500), df = 3), rchisq(500, df = 3),
      main = expression("Q-Q plot for" ~~ {chi^2}[nu == 3])) + theme_bw()
```

`residuals.gamViz` *Generalized Additive Model residuals*

Description

Extension of `mgcv::residuals.gam`. Returns residuals for a fitted GAM model object. Pearson, deviance, working and response residuals are available as in the method from `mgcv`, but this version also provides residual types "tunif" and "tnormal". The former are obtained using the cdf of the response distribution (if available). The latter are obtained by further transforming the uniform residuals using the quantile function (i.e. the inverse cdf) of a standard normal variable.

Usage

```
## S3 method for class 'gamViz'
residuals(object, type = "deviance", ...)
```

Arguments

`object`	an object of class `gamViz`, the output of a `getViz()` call.
`type`	the type of residuals wanted. If should be one of "deviance", "pearson", "scaled.pearson", "working", "response", "tunif" or "tnormal". Not all are available for each family.
`...`	further arguments passed to mgcv::residuals.gam.

See Also

See also mgcv::residuals.gam for details.

`shine` *Generic shine function*

Description

Generic function for taking an object and transforming it into a `shiny` app.

Usage

```
shine(o, ...)
```

Arguments

`o`	the object we want to transform into a `shiny` app.
`...`	arguments to be passed to methods.

See Also

shine.qqGam

`shine.qqGam` *Shiny QQ-plots for GAMs*

Description

This function takes the output of qq.gamViz and transforms it into an interactive shiny app.

Usage

```
## S3 method for class 'qqGam'
shine(o, ...)
```

Arguments

`o`	the output of qq.gamViz.
`...`	currently not used.

Details

In RStudio, this function returns a call to `qq.gamViz` that reproduces the last plot rendered in the interactive shiny window.

Examples

```
## Not run:

## simulate binomial data...
library(mgcv)
library(mgcViz)
set.seed(0)
n.samp <- 400
dat <- gamSim(1,n = n.samp, dist = "binary", scale = .33)
p <- binomial()$linkinv(dat$f) ## binomial p
n <- sample(c(1, 3), n.samp, replace = TRUE) ## binomial n
dat$y <- rbinom(n, n, p)
dat$n <- n
lr.fit <- gam(y/n ~ s(x0) + s(x1) + s(x2) + s(x3)
              , family = binomial, data = dat,
              weights = n, method = "REML")
lr.fit <- getViz(lr.fit)
# launch shiny gagdet
shine(qq(lr.fit))


## End(Not run)
```

simulate.gam *Simulating responses from a GAM object*

Description

This method can be used to simulate vectors of responses from a gamObject.

Usage

```
## S3 method for class 'gam'
simulate(
  object,
  nsim = 1,
  seed = NULL,
  method = "auto",
  newdata,
  u = NULL,
  w = NULL,
  offset = NULL,
  trans = NULL,
  ...
)
```

Arguments

`object`	the output of a `gam()` or `bam()` call.
`nsim`	the number of simulated vectors of responses. A positive integer.
`seed`	currently not used.
`method`	the method used for the simulation. If set to "rd" then `o$family$rd()` will be used, if available. If set to "qf" then `o$family$qf()` (which is the inverse cdf of the response distribution) will be used to transform some uniform variates.
`newdata`	Optional new data frame or list to be passed to predict.gam.
`u`	a matrix where each row is a vector of uniform random variables in (0, 1). This will be used to simulate responses only if `method = "qf"`.
`w`	vector of prior weights to be used in the simulations. If `newdata==NULL` then `w` is set to `object$prior.weights` otherwise it is a vector of ones.
`offset`	numeric vector of offsets. For GAMs with multiple linear predictor (see eg gaulss) it must be a list of vectors. NB: if `newdata!=NULL` the offsets will be assumed to be zero, unless their are explicitly provided. If `newdata==NULL` then simulations will use the offsets used during model fitting, and `offset` argument will be ignored.
`trans`	function used to transform or summarize each vector of simulated responses. It must take a vector as argument, but it can output a vector or a scalar. Potentially useful for saving storage (e.g. by transforming each simulated vector to a scalar). If left to `NULL` then `trans = identity` will be used.
`...`	extra arguments passed to `predict.gam`.

Value

A matrix where each column is a vector of simulated responses. The number of rows is equal to the number of responses in the fitted object.

Examples

```
library(mgcViz)

set.seed(2) ## simulate some data...
dat <- gamSim(1,n=400,dist="normal",scale=2)
b <- gam(y~s(x0)+s(x1)+s(x2)+s(x3),data=dat)

# Simulate three vectors of responses
matplot(simulate(b, nsim = 3), pch = 19, col = c(1, 3, 4))
```

`sm` *Extracting a smooth effect from a GAM model*

Description

This function can be used to extract a smooth or random effect from an object of class `gamViz`.

Usage

```
sm(o, select)
```

Arguments

`o`	an object of class `gamViz`, the output of a getViz call.
`select`	index of the selected smooth or random effect.

Value

An object representing a smooth effect.

See Also

See getViz for examples.

`zoom` *Generic zooming function*

Description

Generic function for zooming, mainly meant to work with graphical objects.

Usage

```
zoom(o, ...)
```

Arguments

`o`	the object we want to zoom into.
`...`	arguments to be passed to methods.

See Also

zoom.qqGam

`zoom.qqGam` *Efficiently zooming on GAM QQ-plots*

Description

This function allows to zoom into a QQ-plot produced by qq.gamViz, in a computationally efficient manner.

Usage

```
## S3 method for class 'qqGam'
zoom(
  o,
  xlim = NULL,
  ylim = NULL,
  discrete = NULL,
  ngr = 1000,
  adGrid = TRUE,
  CI = FALSE,
  worm = FALSE,
  showReps = FALSE,
  a.qqpoi = list(),
  a.ablin = list(),
  a.cipoly = list(),
  a.replin = list(),
  ...
)
```

Arguments

`o`	the output of `mgcViz::qq.gamViz`.
`xlim`	if supplied then this pair of numbers are used as the x limits for the plot.
`ylim`	if supplied then this pair of numbers are used as the y limits for the plot.
`discrete`	if `TRUE` the QQ-plot is discretized into `ngr` bins before plotting, in order to save plotting time (when the number of observations is large). If left to `NULL`, the discretization is used if there are more than 10^4 observations.
`ngr`	number of bins to be used in the discretization.
`adGrid`	if `TRUE` the discretization grid is computed using the QQ-points falling within `xlim`. If `FALSE`, `zoom.qqGam` will compute `ngr` values using all the QQ-points used in the original `qq.gamViz` call (but only those falling within `xlim` and `ylim` will be plotted).
`CI`	if `TRUE` confidence intervals are plotted.
`worm`	if `TRUE` a worm-plot (a de-trended QQ-plot) is plotted, rather than a QQ-plot.
`showReps`	if `TRUE` all the QQ-lines corresponding to the simulated (model-based) QQ-plots.
`a.qqpoi`	list of arguments to be passed to `ggplot2::geom_point`, which plots the main QQ-plot.
`a.ablin`	list of arguments to be passed to `ggplot2::geom_abline`, which adds the reference line.
`a.cipoly`	list of arguments to be passed to `ggplot2::geom_polygon`, which add the confidence intervals.
`a.replin`	list of arguments to be passed to `ggplot2::geom_line`, which adds a line for each simulated QQ-plot.
`...`	currently unused.

Examples

```
library(mgcViz);
set.seed(0)
n.samp <- 500
dat <- gamSim(1,n=n.samp,dist="binary",scale=.33)
p <- binomial()$linkinv(dat$f) ## binomial p
n <- sample(c(1,3),n.samp,replace=TRUE) ## binomial n
dat$y <- rbinom(n,n,p)
dat$n <- n
lr.fit <- bam(y/n ~ s(x0) + s(x1) + s(x2) + s(x3)
              , family = binomial, data = dat,
              weights = n, method = "REML")
lr.fit <- getViz(lr.fit)

set.seed(414)
o <- qq(lr.fit, rep = 50, method = "simul1", CI = "normal")
o # This is the whole qqplot

# We can zoom in along x at little extra costs (most computation already done by qq.gamViz)
zoom(o, xlim = c(0, 1), showReps = TRUE,
     a.replin = list(alpha = 0.1), a.qqpoi =  list(shape = 19))
```

zto1 *Constructing a decreasing function from (0,1) to (0,1)*

Description

This function returns a non-increasing function from (0, 1) to (0, 1). It takes inputs o, a and m, and it returns the function `f(p)={z=max(0,p-o); return(max((1-z)^a,m))}`. The function f(p) can be used, for instance, for transforming p-values before plotting them.

Usage

```
zto1(o, a, m)
```

Arguments

`o, a, m`	the output function's parameters, as described above.

Value

A function whose parameters o, a and m have been fixed.

Examples

```
library(mgcViz)
x <- seq(0, 1, by = 0.01)
plot(x, zto1(0.05, 2, 0.1)(x), ylim = c(0, 1), type = 'l')
lines(x, zto1(0.05, 1, 0.2)(x), col = 2)
lines(x, zto1(0.1, 3, 0)(x), col = 3)
```

Index