Package ‘gamlss’

February 19, 2015

Description The Library For Fitting GAMLSS Models
Version 4.3-4
Date 2015-02-03
Title Generalised Additive Models for Location Scale and Shape.
Author Mikis Stasinopoulos <d.stasinopoulos@londonmet.ac.uk>, Bob Rigby <b.rigby@londonmet.ac.uk>
Vlasios Voudouris <vlasios.voudouris@abm-analytics.com>
with contributions from Calliope Akantziliotou,
and Marco Enea
Maintainer Mikis Stasinopoulos <d.stasinopoulos@londonmet.ac.uk>
Depends R (>= 2.15.0), graphics, stats, splines, utils, gamlss.data,
gamlss.dist (>= 4.3.1), nlme
LazyLoad yes
Imports MASS, survival
License GPL-2 | GPL-3
URL http://www.gamlss.org/
NeedsCompilation no
Repository CRAN
Date/Publication 2015-02-03 13:46:10

R topics documented:

  gamlss-package ......................................................... 3
  acfResid ................................................................. 4
  additive.fit ............................................................. 5
  bfp .......................................................... 7
  calibration ............................................................ 9
  centiles ............................................................. 10
  centiles.com .......................................................... 13
  centiles.pred .......................................................... 14
  centiles.split .......................................................... 17
R topics documented:

coef.gamlss ............................................................... 19
 cs ................................................................. 20
deviance.gamlss ...................................................... 23
dtop ................................................................. 24
edf ................................................................. 26
find.hyper ............................................................ 27
fitDist ............................................................... 30
fitted.gamlss .......................................................... 32
fittedPlot ............................................................ 33
formula.gamlss ....................................................... 34
gamlss ............................................................... 35
gamlss.control ........................................................ 39
gamlss.cs ............................................................ 41
gamlss.fp ............................................................ 42
gamlss.lo ............................................................. 43
gamlss.ps ............................................................. 44
gamlss.random ........................................................ 45
gamlss.scope .......................................................... 46
gamlssML .............................................................. 47
gen.likelihood ......................................................... 49
getSmo ............................................................... 51
glim.control .......................................................... 52
histDist .............................................................. 53
histSmo ............................................................... 55
IC ................................................................. 58
lms ................................................................. 59
lo ................................................................. 62
loglogSurv ............................................................ 64
lpred ............................................................... 66
LR.test ............................................................ 67
model.frame.gamlss .................................................. 68
par.plot .............................................................. 70
pdf.plot ............................................................ 71
plot.gamlss ........................................................... 73
plot.histSmo .......................................................... 75
plot2way ............................................................. 76
polyS ............................................................... 77
predict.gamlss ....................................................... 78
print.gamlss ........................................................... 80
prof.dev ............................................................. 81
prof.term ............................................................ 83
ps ................................................................. 86
Q.stats ............................................................. 90
quantSheets .......................................................... 92
random .............................................................. 94
refit ............................................................... 98
residuals.gamlss ..................................................... 100
ridge .............................................................. 101
Description

This a collection of functions to fit Generalized Additive Models for Location Scale and Shape (GAMLSS) and handled gamlss objects.

GAMLSS were introduced by Rigby and Stasinopoulos (2005). GAMLSS is a general framework for univariate regression type statistical problems using new ways of dealing with overdispersion, skewness and kurtosis in the response variable. In GAMLSS the exponential family distribution assumption used in Generalized Linear Model (GLM) and Generalized Additive Model (GAM), (see Nelder and Wedderburn, 1972 and Hastie and Tibshirani, 1990, respectively) is relaxed and replaced by a very general distribution family including highly skew and kurtotic discrete and continuous distributions. The systematic part of the model is expanded to allow modelling not only the mean (or location) but other parameters of the distribution of the response variable as linear parametric, nonlinear parametric or additive non-parametric functions of explanatory variables and/or random effects terms. Maximum (penalized) likelihood estimation is used to fit the models.

Details

Package: gamlss
Type: Package
Version: 1.5-0
Date: 2006-12-13
License: GPL (version 2 or later) See file LICENSE

This package allow the user to model the distribution of the response variable using a variety of one, two, three and four parameter families of distributions. The distributions implemented currently can be found in gamlss.family. Other distributions can be easily added. In the current implementation of GAMLSS several additive terms have been implemented including regression splines, smoothing splines, penalized splines, varying coefficients, fractional polynomials and random effects. Other additive terms can be easily added.
Author(s)

Mikis Stasinopoulos <d.stasinopoulos@londonmet.ac.uk>, Bob Rigby <r.rigby@londonmet.ac.uk>
with contributions from Calliope Akantziliotou.
Maintainer: Mikis Stasinopoulos <d.stasinopoulos@londonmet.ac.uk>

References


Examples

```r
data(abdom)
mod<-gamlss(y~pb(x),sigma.fo=~pb(x),family=BCT, data=abdom, method=mixed(1,20))
plot(mod)
rm(mod)
```

```r
acfResid
```

*ACF plot of the residuals*

Description

This plot display the ACF and PACF of the residuals of a gamlss or other fitted model (provided that they have been standardised appropriately. Is is appropriate for time series data.

Usage

```r
acfResid(obj = NULL, resid = NULL)
```

Arguments

- `obj` A gamlss model or othe fitted model where the resid() function applies exist
- `resid` if obj does not exist the argument here will be used

Details

The ACF abd PACF for the residuals $r$, squared residuals $r^2$, $r^3$ and $r^4$ are plotted
Value

The relevant plots are displayed.

Author(s)

Mikis Stasinopoulos, Bob Rigby, Vlasios Voudouris and Majid Djennad

References


See Also

*acf*

Examples

```r
library(datasets)
data(co2)
m1 <- gamlss(co2 ~ pb(as.numeric(time(co2)))+factor(cycle(co2)))
acfResid(m1)
```

---

**additive.fit**

**Implementing Backfitting in GAMLSS**

Description

This function is not to be used on its own. It is used for backfitting in the GAMLSS fitting algorithms and it is based on the equivalent function written by Trevor Hastie in the gam() S-plus implementation, (Chambers and Hastie, 1991).

Usage

```r
additive.fit(x, y, w, s, who, smooth.frame, maxit = 30, tol = 0.001, trace = FALSE, se = TRUE, ...)
```
Arguments

- \( x \): the linear part of the explanatory variables
- \( y \): the response variable
- \( w \): the weights
- \( s \): the matrix containing the smoothers
- \( who \): the current smoothers
- \( smooth.frame \): the data frame used for the smoothers
- \( maxit \): maximum number of iterations in the backfitting
- \( tol \): the tolerance level for the backfitting
- \( trace \): whether to trace the backfitting algorithm
- \( se \): whether standard errors are required
- \( ... \): for extra arguments

Details

This function should not be used on its own

Value

Returns a list with the linear fit plus the smoothers

Author(s)

Mikis Stasinopoulos

References


See Also

gamlss

additive.fit
Functions to fit fractional polynomials in GAMLSS

**Description**

The function `bfp` generates a power polynomial basis matrix which (for given powers) can be used to fit power polynomials in one x-variable. The function `fp` takes a vector and returns it with several attributes. The vector is used in the construction of the model matrix. The function `fp()` is not used for fitting the fractional polynomial curves but assigns the attributes to the vector to aid `gamlss` in the fitting process. The function doing the fitting is `gamlss.fp()` which is used at the backfitting function `additive.fit` (but never used on its own). The (experimental) function `pp` can be used to fit power polynomials as in $a + b_1x^{p_1} + b_2x^{p_2}$, where $p_1$ and $p_2$ have arbitrary values rather restricted as in the `fp` function.

**Usage**

```r
bfp(x, powers = c(1, 2), shift = NULL, scale = NULL)
fp(x, npoly = 2, shift = NULL, scale = NULL)
pp(x, start = list(), shift = NULL, scale = NULL)
```

**Arguments**

- **x**
  - the explanatory variable to be used in functions `bfp()` or `fp()`. Note that this is different from the argument `x` use in `gamlss.fp` (a function used in the backfitting but not by straight by the user)

- **powers**
  - a vector containing as elements the powers in which the x has to be raised

- **shift**
  - a number for shifting the x-variable. The default values is zero, if x is positive, or the minimum of the positive difference in x minus the minimum of x

- **scale**
  - a positive number for scaling the x-variable. The default values is $10^{\lfloor \text{sign}(log10(\text{range})) \rfloor \times \text{trunc}(abs(log10(\text{range})))}$

- **npoly**
  - a positive indicating how many fractional polynomials should be considered in the fit. Can take the values 1, 2 or 3 with 2 as default

- **start**
  - a list containing the starting values for the non-linear maximization to find the powers. The results from fitting the equivalent fractional polynomials can be used here

**Details**

The above functions are an implementation of the fractional polynomials introduced by Royston and Altman (1994). The three functions involved in the fitting are loosely based on the fractional polynomials implementation in S-plus written by Gareth Amber. The function `bfp` generates the right design matrix for the fitting a power polynomial of the type $a + b_1x^{p_1} + b_2x^{p_2} + \ldots + b_kx^{p_k}$. For given powers $p_1, p_2, \ldots, p_k$ given as the argument `powers` in `bfp()` the function can be used to fit power polynomials in the same way as the functions `poly()` or `bs()` (of package `splines`) are used to fit orthogonal or piecewise polynomials respectively. The function `fp()`, which is working as a smoother in `gamlss`, is used to fit the best fractional polynomials within a set of power values.
Its argument `npoly` determines whether one, two or three fractional polynomials should be used in the fitting. For a fixed number `npoly` the algorithm looks for the best fitting fractional polynomials in the list `-2, -1, -0.5, 0, 0.5, 1, 2, 3`. Note that `npoly=3` is rather slow since it fits all possible combinations 3-way combinations at each backfitting interaction. The function `gamlss.fp()` is an internal function of GAMLSS allowing the fractional polynomials to be fitted in the backfitting cycle of `gamlss`, and should be not used on its own.

**Value**

The function `bfp` returns a matrix to be used as part of the design matrix in the fitting.

The function `fp` returns a vector with values zero to be included in the design matrix but with attributes useful in the fitting of the fractional polynomials algorithm in `gamlss.fp`.

**Warning**

Since the model constant is included in both the design matrix `X` and in the backfitting part of fractional polynomials, its values is wrongly given in the `summary`. Its true values is the model constant minus the constant from the fractional polynomial fitting. What happens if more than one fractional polynomials are fitted?

**Author(s)**

Mikis Stasinopoulos <d.stasinopoulos@londonmet.ac.uk>, Bob Rigby <b.rigby@londonmet.ac.uk>

**References**


**See Also**

`gamlss`, `gamlss.family`

**Examples**

```r
data(abdom)
# fits polynomials with power 1 and .5
mod1<-gamlss(y~bfp(x,c(1,0.5)),data=abdom)
# fit the best of one fractional polynomial
mod1<-gamlss(y~fp(x,1),data=abdom)
```
calibration

# fit the best of two fractional polynomials
m2 <- gamlss(y ~ fp(x,2), data = abdom)
# fit the best of three fractional polynomials
m3 <- gamlss(y ~ fp(x,3), data = abdom)
# get the coefficient for the second model
m2$mu.coef$mo
# now power polynomials using the best 2 fp c()
m4 <- gamlss(y ~ pp(x, c(1,3)), data = abdom)
# This is not good idea in this case because
# if you look at the fitted values you see what it went wrong
plot(y ~ x, data = abdom)
lines(fitted(m2, "mu") ~ abdom$x, col = "red")
lines(fitted(m4, "mu") ~ abdom$x, col = "blue")

---

**calibration**  
*Calibrating centile curves*

**Description**

This function can be used when the fitted model centiles do not coincide with the sample centiles.

**Usage**

```r
calibration(object, xvar, cent = 100 * pnorm((-4:4) * 2/3),
              legend = FALSE, fan = FALSE, ...)
```

**Arguments**

- `object`: a `gamlss` fitted object
- `xvar`: The explanatory variable
- `cent`: a vector with elements the % centile values for which the centile curves have to be evaluated
- `legend`: whether legend is required
- `fan`: whether to use the fan version of centiles
- `...`: other argument pass on to `centiles()` function

**Details**

The function finds the sample quantiles of the residuals of the fitted model (the z-scores) and use them as sample quantile in the argument `cent` of the `centiles()` function. This procedure is appropriate if the fitted model centiles do not coincide with the sample centiles and when this failure is the same in all values of the explanatory variable `xvar`.

**Value**

A centile plot is produced and the sample centiles below each centile curve are printed (or saved)
centiles

Author(s)
Mikis Stasinopoulos <d.stasinopoulos@londonmet.ac.uk>, Bob Rigby <r.rigby@londonmet.ac.uk> and Vlasios Voudouris <vlasios.voudouris@abm-analytics.com>

References


See Also
centiles, centiles.fan

Examples
```r
data(abdom)
m1<-gamlss(y~pb(x), sigma.fo~ pb(x), family=L0, data=abdom)
calibration(m1, xvar=abdom$x, fan=TRUE)
```

centiles

Plots the centile curves for a GAMLSS object

Description
This function `centiles()` plots centile curves for distributions belonging to the GAMLSS family of distributions. The function also tabulates the sample percentages below each centile curve (for comparison with the model percentages given by the argument `cent`). The function `centiles.fan()` plots a fan-chart of the centile curves. A restriction of the functions is that it applies to models with one explanatory variable only.

Usage
```r
centiles(obj, xvar = NULL, cent = c(0.4, 2, 10, 25, 50, 75, 90, 98, 99.6),
   legend = TRUE, ylab = "y", xlab = "x", main = NULL,
   main.gsub = "@", xleg = min(xvar), yleg = max(obj$y),
   xlim = range(xvar), ylim = range(obj$y), save = FALSE,
   plot = TRUE, points = TRUE, pch = 15, cex = 0.5, col = gray(0.7),
   col.centiles = 1:length(cent) + 2, lty.centiles = 1, lwd.centiles = 1,...)
centiles.fan(obj, xvar = NULL, cent = c(0.4, 2, 10, 25, 50, 75, 90, 98, 99.6),
   ylab = "y", xlab = "x", main = NULL, main.gsub = "@",
   xleg = min(xvar), yleg = max(obj$y), xlim = range(xvar),
```

centiles

ylim = range(obj$y), points = FALSE, median = TRUE, pch = 15,
cex = 0.5, col = gray(0.7),
colors = c("cm", "gray", "rainbow", "heat", "terrain", "topo"), ...)

Arguments

obj a fitted gamlss object from fitting a gamlss distribution
xvar the unique explanatory variable
cent a vector with elements the % centile values for which the centile curves have to
be evaluated
legend whether a legend is required in the plot or not, the default is legend=TRUE
ylab the y-variable label
xlab the x-variable label
main the main title here as character. If NULL the default title "centile curves using
NO" (or the relevant distributions name) is shown
main.gsub if the main.gsub (with default "@") appears in the main title then it is substi-
tuted with the default title.
xleg position of the legend in the x-axis
yleg position of the legend in the y-axis
xlim the limits of the x-axis
ylim the limits of the y-axis
save whether to save the sample percentages or not with default equal to FALSE. In
this case the sample percentages are printed but are not saved
plot whether to plot the centiles. This option is useful for centile.split
pch the character to be used as the default in plotting points see par
cex size of character see par
col plotting colour see par
col.centiles Plotting colours for the centile curves
lty.centiles line type for the centile curves
lwd.centiles The line width for the centile curves
colors the different colour schemes to be used for the fan-chart. The following are
available c("cm", "gray", "rainbow", "heat", "terrain", "topo"),
points whether the data points should be plotted, default is TRUE for centiles() and
FALSE for centiles.fan()
median whether the median should be plotted (only in centiles.fan())
... for extra arguments

Details

Centiles are calculated using the fitted values in obj and xvar must correspond exactly to the pre-
dictor in obj to plot correctly.
col.centiles, lty.centiles and lwd.centiles may be vector arguments and are recycled to
the length cent if necessary.
A centile plot is produced and the sample centiles below each centile curve are printed (or saved)

This function is appropriate only when one continuous explanatory variable is fitted in the model

Mikis Stasinopoulos <d.stasinopoulos@londonmet.ac.uk>, Bob Rigby <r.rigby@londonmet.ac.uk>
with contribution from Steve Ellison

References

Rigby, R. A. and Stasinopoulos D. M. (2005). Generalized additive models for location, scale and

GAMLSS package in R. Accompanying documentation in the current GAMLSS help files, (see
also http://www.gamlss.org/).

org/v23/i07.

See Also

gamlss, centiles.split, centiles.com

Examples

data(abdom)
h<-gamlss(y~pb(x), sigma.formula=~pb(x), family=BCT, data=abdom)
# default plot
centiles(h,xvar=abdom$x)
# control of colours and lines
centiles(h, xvar=abdom$x, col.cent=c(2,3,4,5,1,5,4,3,2,1),
         lwd.cent=c(1,1,1,1,2,1,1,1,1,1))
#Control line types
centiles(h, xvar=abdom$x, col.cent=1, cent=c(.5,2.5,50,97.5,99.5),
         lty.centiles=c(3,2,1,2,3),lwd.cent=c(1,1,1,1,1))
# control of the main title
centiles(h, xvar=abdom$x, main="Abdominal data \n @")
# the fan-chart
centiles.fan(h,xvar=abdom$x, colors="rainbow")
rm(h)
Description

This function compares centiles curves for more than one GAMLSS objects. It is based on the centiles function. The function also tabulates the sample percentages below each centile curve (for comparison with the model percentages given by the argument cent.) A restriction of the function is that it applies to models with one explanatory variable only.

Usage

```r
centiles.com(obj, ..., xvar = NULL, cent = c(0.4, 10, 50, 90, 99.6),
  legend = TRUE, ylab = "y", xlab = "x", xleg = min(xvar),
  yleg = max(obj$y), xlim = range(xvar), ylim = NULL,
  no.data = FALSE, color = TRUE, main = NULL, plot = TRUE)
```

Arguments

- **obj**: a fitted gamlss object from fitting a gamlss continuous distribution
- **...**: optionally more fitted GAMLSS model objects
- **xvar**: the unique explanatory variable
- **cent**: a vector with elements the % centile values for which the centile curves have to be evaluated
- **legend**: whether a legend is required in the plot or not, the default is legend=TRUE
- **ylab**: the y-variable label
- **xlab**: the x-variable label
- **xleg**: position of the legend in the x-axis
- **yleg**: position of the legend in the y-axis
- **xlim**: the limits of the x-axis
- **ylim**: the limits of the y-axis
- **no.data**: whether the data should plotted, default no.data=FALSE or not no.data=TRUE
- **color**: whether the fitted centiles are shown in colour, color=TRUE (the default) or not color=FALSE
- **main**: the main title
- **plot**: whether to plot the centiles

Value

Centile plots are produced for the different fitted models and the sample centiles below each centile curve are printed.
Warning

This function is appropriate only when one continuous explanatory variable is fitted in the model.

Author(s)

Mikis Stasinopoulos <d.stasinopoulos@londonmet.ac.uk> and Bob Rigby <r.rigby@londonmet.ac.uk>

References


See Also

gamlss, centiles, centiles.split

Examples

data(abdom)

h1 <- gamlss(y ~ cs(x, df=3), sigma.formula = ~cs(x, 1), family = BCT, data = abdom)

h2 <- gamlss(y ~ pb(x), sigma.formula = ~pb(x), family = BCT, data = abdom)

centiles.com(h1, h2, xvar = abdom$x)

rm(h1, h2)

centiles.pred

Creating predictive centiles values

Description

This function creates predictive centiles curves for new x-values given a GAMLSS fitted model. The function has three options: i) for given new x-values and given percentage centiles calculates a matrix containing the centiles values for y, ii) for given new x-values and standard normalized centile values calculates a matrix containing the centiles values for y, iii) for given new x-values and new y-values calculates the z-scores. A restriction of the function is that it applies to models with only one explanatory variable.

Usage

centiles.pred(obj, type = c("centiles", "z-scores", "standard-centiles"),
               xname = NULL, xvalues = NULL, power = NULL, yval = NULL,
               cent = c(0.4, 2, 10, 25, 50, 75, 90, 98, 99.6),
               dev = c(-4, -3, -2, -1, 0, 1, 2, 3, 4),
               plot = FALSE, legend = TRUE,
               ...)
Arguments

- **obj**: A fitted gamlss object from fitting a gamlss continuous distribution
- **type**: The default, "centiles", gets the centiles values given in the option cent. type="standard-centiles" gets the standard centiles given in the dev. type="z-scores" gets the z-scores for given y and x new values
- **xname**: The name of the unique explanatory variable (it has to be the same as in the original fitted model)
- **xvalues**: The new values for the explanatory variable where the prediction will take place
- **power**: If power transformation is needed (but read the note below)
- **yval**: The response values for a given x required for the calculation of "z-scores"
- **cent**: A vector with elements the % centile values for which the centile curves have to be evaluated
- **dev**: A vector with elements the standard normalized values for which the centile curves have to be evaluated in the option type="standard-centiles"
- **plot**: Whether to plot the "centiles" or the "standard-centiles", the default is plot=FALSE
- **legend**: Whether a legend is required in the plot or not, the default is legend=TRUE
- **...**: For extra arguments

Value

A vector (for option type="z-scores") or a matrix for options type="centiles" or type="standard-centiles" containing the appropriate values

Warning

See example below of how to use the function when power transformation is used for the x-variables

Note

The power option should be only used if the model

Author(s)

Mikis Stasinopoulos, <d.stasinopoulos@londonmet.ac.uk>, based on ideas of Elaine Borghie from the World Health Organization

References


See Also
gamlss, centiles, centiles.split

Examples

```r
## bring the data and fit the model
data(abdom)
a <- gamlss(y~pb(x), sigma.fo=-pb(x), data=abdom, family=BCT)
## plot the centiles
centiles(a, xvar=abdom$x)
# first use of centiles.pred()
# to calculate the centiles at new x values
newx <- seq(12,40,2)
mat <- centiles.pred(a, xname="x", xvalues=newx)
mat
## now plot the centile curves
mat <- centiles.pred(a, xname="x", xvalues=newx, plot=TRUE)

## second use of centiles.pred()
## to calculate (normalised) standard-centiles for new x
## values using the fitted model
newx <- seq(12,40,2)
mat <- centiles.pred(a, xname="x", xvalues=newx, type="standard-centiles")
mat
## now plot the standard centiles
mat <- centiles.pred(a, xname="x", xvalues=newx, type="standard-centiles",
plot = TRUE)

## third use of centiles.pred()
## if we have new x and y values what are their z-scores?
newy <- c(130,121,123,125,140,145,150)
for(i in 1:7) points(newx[i],newy[i],col="blue")
## now calculate their z-scores
znewx <- centiles.pred(a, xname="x", xvalues=newx, yval=newy, type="z-scores")
znewx
## Not run:
## What we do if the x variables is transformed?
## case 1 : transformed x-variable within the formula
## fit model
aa <- gamlss(y~pb(x*0.5), sigma.fo=-pb(x^0.5), data=abdom, family=BCT)
## centiles works
centiles(aa,xvar=abdom$x, legend = FALSE)
newx <- seq(12,40,2)
```

centiles.split

Plots centile curves split by x for a GAMLSS object

Description

This function plots centiles curves for separate ranges of the unique explanatory variable x. It is similar to the centiles function but the range of x is split at a user defined values xcut.points into r separate ranges. The functions also tabulates the sample percentages below each centile curve for each of the r ranges of x (for comparison with the model percentage given by cent). The model should have only one explanatory variable.

Usage

centiles.split(obj, xvar = NULL, xcut.points = NULL, n.inter = 4,
cent = c(0.4, 2, 10, 25, 50, 75, 90, 98, 99.6),
legend = FALSE, main = NULL, main.gsub = "@",
ylab = "y", xlab = "x", ylim = NULL, overlap = 0,
save = TRUE, plot = TRUE, ...)

Arguments

obj a fitted gamlss object from fitting a gamlss continuous distribution
xvar the unique explanatory variable
xcut.points the x-axis cut off points e.g. c(20, 30). If xcut.points=NULL then the n.inter argument is activated
n.inter if xcut.points=NULL this argument gives the number of intervals in which the x-variable will be splited, with default 4
cent a vector with elements the % centile values for which the centile curves are to be evaluated

---

mat <- centiles.pred(aa, xname="x", xvalues=newx, plot=TRUE)
mat <- centiles.pred(aa, xname="x", xvalues=c(20, 30))
mat
xx <- rep(mat[,1],9)
yy<mat[,2:10]
points(xx,yy,col="red")
## case 2 : x-variable previously transformed
nx<abdom$x^0.5
aa<-gamlss(y=pb(nx),sigma.fo=pb(nx), data=abdom, family=BCT)
centiles(aa, xvar=abdom$x)
newd<-data.frame( abdom, nx=abdom$x^0.5)
mat <- centiles.pred(aa, xname="nx", xvalues=c(30), power=0.5, data=newd)
xxx<rep(mat[,1],9)
yyy<mat[,2:10]
points(xxx,yyy,col="red")
## End(Not run)
centiles.split

legend
main
main.ngsub
ylab
xlab
ylim
overlap
save
plot

whether a legend is required in the plots or not, the default is legend=FALSE
the main title as character. If NULL the default title (shown the intervals) is shown
if the main.ngsub (with default "@") appears in the main title then it is substituted with the default title.
the y-variable label
the x-variable label
the range of the y-variable axis
how much overlapping in the xvar intervals. Default value is overlap=0 for non overlapping intervals
whether to save the sample percentages or not with default equal to TRUE. In this case the functions produce a matrix giving the sample percentages for each interval
whether to plot the centiles. This option is useful if the sample statistics only are to be used

... for extra arguments

Value

Centile plots are produced and the sample centiles below each centile curve for each of the r ranges of x can be saved into a matrix.

Warning

This function is appropriate when only one continuous explanatory variable is fitted in the model

Author(s)

Mikis Stasinopoulos, <d.stasinopoulos@londonmet.ac.uk>, Bob Rigby <r.rigby@londonmet.ac.uk>, with contributions from Elaine Borghie

References


See Also

gamlss centiles, centiles.com
Examples

```r
data(abdom)
h <- gamlss(y~pb(x), sigma.formula=-pb(x), family=BCT, data=abdom)
mout <- centiles.split(h,xvar=abdom$x)
mout
rm(h,mout)
```

Description

coeff.gamlss is the GAMLSS specific method for the generic function `coef` which extracts model coefficients from objects returned by modelling functions. ‘coefficients’ is an alias for `coef`.

Usage

```r
### S3 method for class 'gamlss'
coef(object, what = c("mu", "sigma", "nu", "tau"), ...)
```

Arguments

- `object`: a GAMLSS fitted model
- `what`: which parameter coefficient is required, default `what="mu"`
- `...`: for extra arguments

Value

Coefficients extracted from the GAMLSS model object.

Author(s)

Mikis Stasinopoulos <d.stasinopoulos@londonmet.ac.uk>

References


See Also

gamlss, deviance.gamlss, fitted.gamlss

Examples

data(aids)
h <- gamlss(y~poly(x,3)+qrt, family=PO, data=aids) #
coef(h)
rm(h)

---

*cs*

**Specify a Smoothing Cubic Spline Fit in a GAMLSS Formula**

**Description**

The functions `cs()` and `scs()` are using the cubic smoothing splines function `smooth.spline()` to do smoothing. They take a vector and return it with several attributes. The vector is used in the construction of the model matrix. The functions do not do the smoothing, but assigns the attributes to the vector to aid gamlss in the smoothing. The function doing the smoothing is `gamlss.cs()`. This function use the R function `smooth.spline()` which is then used by the backfitting function `additive.fit()` which is based on the original GAM implementation described in Chambers and Hastie (1992). The function `gamlss.scs()` differs from the function `cs()` in that allows cross validation of the smoothing parameters unlike the `cs()` which fixes the effective degrees of freedom, `df`. Note that the recommended smoothing function is now the function `pb()` which allows the estimation of the smoothing parameters using a local maximum likelihood. The function `pb()` is based on the penalised beta splines (P-splines) of Eilers and Marx (1996).

The (experimental) function `vc` is now defunct. For fitting varying coefficient models, Hastie and Tibshirani (1993) use the function `pvc()`.

**Usage**

```r
cs(x, df = 3, spar = NULL, c.spar = NULL, control = cs.control(...), ...)
scs(x, df = NULL, spar = NULL, control = cs.control(...), ...)
cs.control(cv = FALSE, all.knots = TRUE, nknots = NULL, keep.data = TRUE,
          df.offset = 0, penalty = 1.4, control.spar = list(), ...)
```

**Arguments**

- **x**: the univariate predictor, (or expression, that evaluates to a numeric vector). For the function `vc` the `x` argument is the vector which has its (linear) coefficient change with `r`
- **df**: the desired equivalent number of degrees of freedom (trace of the smoother matrix minus two for the constant and linear fit). The real smoothing parameter (spar below) is found such that `df=tr(S)-2`, where `S` is the implicit smoother matrix. Values for `df` should be greater than 0, with 0 implying a linear fit.
spar smoothing parameter, typically (but not necessarily) in (0,1]. The coefficient
lambda of the integral of the squared second derivative in the fit (penalised
log likelihood) criterion is a monotone function of 'spar', see the details in
smooth.spline.

c.spar This is an option to be used when the degrees of freedom of the fitted gamlss
object are different from the ones given as input in the option df. The de-
default values used are the ones given the option control.spar in the R func-
tion smooth.spline() and they are c.spar=c(-1.5, 2). For very large data
sets e.g. 10000 observations, the upper limit may have to increase for example
to c.spar=c(-1.5, 2.5). Use this option if you have received the warning
'The output df are different from the input, change the control.spar'. c.spar
can take both vectors or lists of length 2, for example c.spar=c(-1.5, 2.5) or
c.spar=list(-1.5, 2.5) would have the same effect.

control control for the function smooth.spline(), see below
cv see the R function smooth.spline()
all.knots see the R function smooth.spline()
nknots see the R function smooth.spline()
keep.data see the R function smooth.spline()
df.offset see the R function smooth.spline()
penalty see the R function smooth.spline(), here the default value is 1.4
control.spar see above c.spar or the equivalent argument in the function smooth.spline
... for extra arguments

Details

Note that cs itself does no smoothing; it simply sets things up for the function gamlss() which in
turn uses the function additive.fit() for backfitting which in turn uses gamlss.cs()

Note that cs() and scs() functions behave differently at their default values that is if df and lambda
are not specified. cs(x) by default will use 3 extra degrees of freedom for smoothing for x. scs(x)
by default will estimate lambda (and the degrees of freedom) automatically using generalised cross
validation (GCV). Note that if GCV is used the convergence of the gamlss model can be less stable
compared to a model where the degrees of freedom are fixed. This will be true for small data sets.

Value

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

Warning

For a user who wishes to compare the gamlss() results with the equivalent gam() results in S-plus:
make sure when using S-plus that the convergence criteria epsilon and bf.epsilon in control.gam() are
decreased sufficiently to ensure proper convergence in S-plus. Also note that the degrees of
freedom are defined on top of the linear term in gamlss, but on top of the constant term in S-plus,
(so use an extra degrees of freedom in S-plus in order to obtain comparable results to those in 
gamlss).

Change the upper limit of spar if you received the warning 'The output df are different from the 
input, change the control.spar'.

For large data sets do not use expressions, e.g. cs(x^0.5) inside the gamlss function command 
but evaluate the expression, e.g. nx=x^0.5, first and then use cs(nx).

Note

The degrees of freedom df are defined differently from that of the gam() function in S-plus. Here df 
are the additional degrees of freedom excluding the constant and the linear part of x. For example 
df=4 in gamlss() is equivalent to df=5 in gam() in S-plus.

Author(s)

Mikis Stasinopoulos and Bob Rigby (see also the documentation of the function 
smoothNspline() for the original authors of the cubic spline function.)

References


Eilers, P. H. C. and Marx, B. D. (1996). Flexible smoothing with B-splines and penalties (with 

Soc. B., 55, 757-796.

Rigby, R. A. and Stasinopoulos D. M. (2005). Generalized additive models for location, scale and 

GAMLSS package in R. Accompanying documentation in the current GAMLSS help files, (see 
also http://www.gamlss.org/).

org/v23/i07.

See Also

gamlss, gamlss.cs, pb, pvc

Examples

# cubic splines example
data(aids)
# fitting a smoothing cubic spline with 7 degrees of freedom 
# plus the a quarterly effect
aids1<-gamlss(y~cs(x,df=7)+qrt,data=aids,family=PO) 

aids2<-gamlss(y~scs(x,df=5)+qrt,data=aids,family=PO) 

aids3<-gamlss(y~scs(x)+qrt,data=aids,family=PO) # using GCV 
with(aids, plot(x,y))
deviance.gamlss

Description

Returns the global, \(-2\times\log(\text{likelihood})\), or the penalized, \(-2\times\log(\text{likelihood})+\) penalties, deviance of a fitted GAMLSS model object.

Usage

```r
## S3 method for class 'gamlss'
deviance(object, what = c("G", "P"), ...)
```

Arguments

- `object`: a GAMLSS fitted model
- `what`: put "G" for Global or "P" for Penalized deviance
- `...`: for extra arguments

Details

deviance is a generic function which can be used to extract deviances for fitted models. deviance.gamlss is the method for a GAMLSS object.

Value

The value of the global or the penalized deviance extracted from a GAMLSS object.

Author(s)

Mikis Stasinopoulos `<d.stasinopoulos@londonmet.ac.uk>`

References


See Also
gamlss.family, coef.gamlss, fitted.gamlss

Examples

data(aids)
h<-gamlss(y=poly(x,3)+qrt, family=PO, data=aids) #
deviance(h)
rm(h)

dtop                Detrended transformed Owen's plot

Description

Provides single or multiple detrended transformed Owen’s plot, Owen (1995), for a GAMLSS fitted objects or any other fitted object which has the method resid(). This is a diagnostic tool for checking whether the normalised quantile residuals are coming from a normal distribution or not. This could be true if the horizontal line is within the confidence intervals.

Usage

dtop(object = NULL, xvar = NULL, resid = NULL, conf.level = c("95", "99"), n.inter = 4, xcut.points = NULL, overlap = 0, show.given = TRUE, cex = 1, pch = 21, line = TRUE, ...)

Arguments

object          a GAMLSS fitted object or any other fitted object which has the method resid().
xvar            the explanatory variable against which the detrended Owen’s plots will be plotted
resid           if the object is not specified the residual vector can be given here
conf.level      95 (default) or 99 percent confidence interval for the plots
n.inter         the number of intervals in which the explanatory variable xvar will be cut
xcut.points     the x-axis cut off points e.g. c(20,30). If xcut.points=NULL then the n.inter argument is activated
overlap         how much overlapping in the xvar intervals. Default value is overlap=0 for non overlapping intervals
show.given      whether to show the x-variable intervals in the top of the graph, default is show.given=TRUE
cex             the cex plotting parameter with default cex=1
pch             the pch plotting parameter with default pch=21
line            whether the detrended empirical cdf should be plotted or not
...             for extra arguments
Details

If the `xvar` argument is not specified then a single detrended Owen’s plot is used, see Owen (1995). In this case the plot is a detrended nonparametric likelihood confidence band for a distribution function. That is, if the horizontal lines lies within the confidence band then the normalised residuals could have come from a Normal distribution and consequently the assumed response variable distribution is reasonable. If the `xvar` is specified then we have as many plots as `n.iter`. In this case the x-variable is cut into `n.iter` intervals with an equal number observations and detrended Owen’s plots for each interval are plotted. This is a way of highlighting failures of the model within different ranges of the explanatory variable.

Value

A plot is returned.

Author(s)

Mikis Stasinopoulos, Bob Rigby and Vlassios Voudouris

References


See Also

wp

Examples

data(abdom)
a<-gamlss(y~pb(x),sigma.fo=-pb(x,1),family=LO,data=abdom)
dtop(a)
dtop(a, xvar=abdom$x)
rm(a)
Effective degrees of freedom from gamlss model

Description

The functions `edf()` and `edfAll()` can be used to obtained the effective degrees of freedom for different additive terms for the distribution parameters in a gamlss model.

Usage

```r
edf(obj, what = c("mu", "sigma", "nu", "tau"), print = TRUE, ...)
edfAll(obj, ...)
```

Arguments

- `obj` A gamlss fitted model
- `what` which of the four parameters `mu`, `sigma`, `nu` or `tau`.
- `print` whether to print the label
- `...` for extra arguments

Value

The function `edfAll()` returns a list of edf for all the fitted parameters. The function `edf()` a vector of edf.

Note

The edf given are the ones fitted in the backfitting so the usually contained (depending on the additive term) the contatnt and the linear part.

Author(s)

Mikis Stasinopoulos <d.stasinopoulos@londonmet.ac.uk>

References


A function to select values of hyper-parameters in a GAMLSS model

Description

This function selects the values of hyper parameters and/or non-linear parameters in a GAMLSS model. It uses the R function `optim` which then minimises the generalised Akaike information criterion (GAIC) with a user defined penalty.

Usage

```r
find.hyper(model = NULL, parameters = NULL, other = NULL, k = 2,
            steps = c(0.1), lower = -Inf, upper = Inf, method = "L-BFGS-B",
            ...)```

Arguments

- `model`: this is a GAMLSS model in `quote()`. e.g. `quote(gamlss(y~cs(x,df=p[1]),sigma.fo=cs(x,df=p[2]),data=abdom))` where `p[1]` and `p[2]` denote the parameters to be estimated
- `parameters`: the starting values in the search of the optimum hyper-parameters and/or non-linear parameters e.g. `parameters=c(3,3)`
- `other`: this is used to optimise other non-parameters, for example a transformation of the explanatory variable of the kind `x^p[3]`, `others=quote(nx<-x^p[3])` where `nx` is now in the model formula
- `k`: specifies the penalty in the GAIC, (the default is 2) e.g. `k=3`
- `steps`: the steps taken in the optimisation procedure [see the `nsteps` option in `optim()`], by default is set to 0.1 for all hyper parameters and non-linear parameters
- `lower`: the lower permissible level of the parameters i.e. `lower=c(1,1)` this does not apply if a method other than the default method "L-BFGS-B" is used
- `upper`: the upper permissible level of the parameters i.e. `upper=c(3,10)`, this is not apply if a method other than the default method "L-BFGS-B" is used
- `method`: the method used in `optim()` to numerically minimise the GAIC over the hyper-parameters and/or non-linear parameters. By default this is "L-BFGS-B" to allow box-restriction on the parameters

Examples

```r
library(gamlss.data)
data(usair)
m1<- gamlss(y~pb(x1)+pb(x2)+pb(x6), data=usair)
edfAll(m1)
edf(m1)
```
for extra arguments to be passed to the R function \texttt{optim()} used in the optimisation.

Details

This historically was an experimental function which worked well for the search of the optimum degrees of freedom and non-linear parameters (e.g. power parameter $\lambda$ used to transform $x$ to $x^\lambda$). With the introduction of the P-Spline smoothing function \texttt{pb()} the function \texttt{find.hyper()} became almost redundant. \texttt{find.hyper()} takes lot longer than \texttt{pb()} to find automatically the hyper parameters while both method produce similar results. See below the examples for a small demonstration.

Value

The function turns the same output as the function \texttt{optim()}.

- \texttt{par} the optimum hyper-parameter values
- \texttt{value} the minimised value of the GAIC
- \texttt{counts} A two-element integer vector giving the number of calls to ‘fn’ and ‘gr’ respectively
- \texttt{convergence} An integer code. ‘0’ indicates successful convergence. see the function \texttt{optim()} for other errors
- \texttt{message} A character string giving any additional information returned by the optimiser, or ‘NULL’

Warning

It may be slow to find the optimum.

Author(s)

Mikis Stasinopoulos

References


See Also

\texttt{gamlss}, \texttt{plot.gamlss}, \texttt{optim}
Examples

```r
## Not run:
data(abdom)
# Example estimating the smoothing parameters for mu and
# the transformation parameters for x
# declare the model
mod1 <- quote(gamlss(y~cs(nx,df=p[1]),family=BCT,data=abdom,
control=gamlss.control(trace=FALSE)));
# since we want also to find the transformation for x
# we use the "other" option
op <- find.hyper(model=mod1, other=quote(nx~x*p[2]), parameters=c(3,0.5),
lower=c(1,0.001), steps=c(0.1,0.001))

op
# the optimum parameters found are
# p = (p[1],p[2]) = (3.113218 0.001000) = (df for mu, lambda)
# so it needs df = 3 on top of the constant and linear
# in the cubic spline model for mu since p[1] is approximately 3
# and log transformation for x since p[2] is approximately 0
# here is an example with no data declaration in define the model
# we have to attach the data
attach(abdom)
mod2 <- quote(gamlss(y~cs(nx,df=p[1]),family=BCT,
control=gamlss.control(trace=FALSE)));
op2<find.hyper(model=mod2, other=quote(nx~x*p[2]), parameters=c(3,0.5),
lower=c(1,0.001), steps=c(0.1,0.001))
op2

attach(abdom)
# showing different ways of estimating the smoothing parameter
# get the df using local ML (PQL)
m0 <- gamlss(y~pb(x), data=abdom)
# get the df using local GAIC
m1<gamlss(y~pb(x, method="GAIC", k=2), data=abdom)
# fitting cubic splines with fixed df's at 3
m2<gamlss(y~cs(x, df=3), data=abdom)
# fitting cubic splines using find hyper (global GAIC)
mod1 <- quote(gamlss(y~cs(x, df=p[1]),family=BCT,data=abdom,control=gamlss.control(trace=FALSE)));
op <- find.hyper(model=mod1, parameters=c(3), lower=c(1,0.001), steps=c(0.1,0.001))

m3 <- gamlss(y~cs(x, df=op$par), data=abdom)
# effective degrees of freedom for the 4 models
edf(m0);edf(m1); m2$mu.df; m3$mu.df
# deviances for the four models
deviance(m0); deviance(m1); deviance(m2); deviance(m3)
# their GAIC
GAIC(m0,m1,m2,m3)
# plotting the models
plot(y~x, data=abdom, type="n")
lines(fitted(m3)-abdom$x, col="red")
lines(fitted(m1)-abdom$x, col="green")
lines(fitted(m0)-abdom$x, col="blue")
# almost identical
```
fitDist

fits Different Parametric `gamlss.family` distributions to data

Description

This function is using the function `gamlssML()` to fit all relevant parametric `gamlss.family` distributions to a data vector. The final model is the one which is selected by the generalised Akaike information criterion with penalty k.

Usage

```r
fitDist(y, k = 2,
        type = c("realAll", "realline", "realplus", "real0to1", "counts", "binom"),
        try.gamlss = FALSE, extra = NULL, data = NULL, ...)
```

Arguments

- **y**: the data vector
- **k**: the penalty for the GAIC with default values k=2 the standard AIC
- **type**: the type of distribution to be tried see details
- **try.gamlss**: if `gamlssML()` failed whether should try `gamlss` instead. This will slow up things for big data.
- **extra**: whether extra distribution should be tried which are not in the type list
- **data**: the data frame where y can be found
- **...**: for extra arguments to be passed to `gamlssML()` to `gamlss()`

Details

The following are the different type argument:

- **realAll**: all the `gamlss.family` continuous distributions defined on the real line, i.e. `realline` plus `realplus`
- **realplus**: the `gamlss.family` continuous distributions in the positive real line: "EXP","GA","IG","LNO", "WEB3", "BCCGo", "exGAUS", "GG", "GIG", "BCTo", "BCEvo"
- **real0to1**: the `gamlss.family` continuous distributions from 0 to 1: "BE", "BEINF", "BE-INF0", "BEINF1", "BEIO1", "BEZI", "GB1"
- **counts**: the `gamlss.family` distributions for counts: "PO", "LG", "NBI", "NBII", "PIG", "DEL", "SI", "ZIP", "ZAP", "ZALG", "ZANBI", "ZIP2", "ZIPIG"
- **binom**: the `gamlss.family` distributions for binomial type data :"BI", "BB", "ZIBI", "ZIBB", "ZABI", "ZABB"
fitDist

Value

A `gamlssML` object with two extra components:

- `fits` an ordered list according to the GAIC of the fitted distribution
- `failed` the distributions where the `gamlssML()` (or `gamlss()`) fits have failed

Author(s)

Mikis Stasinopoulos <d.stasinopoulos@londonmet.ac.uk>, Bob Rigby <r.rigby@londonmet.ac.uk>, Vlasis Voudouris <v.voudouris@londonmet.ac.uk> and Majid Djennad <m.djennad.londonmet.ac.uk>

References


See Also

`gamlss`, `gamlssML`

Examples

```r
y <- rt(100, df=1)
m1<-fitDist(y, type="realline")
m1$fits
m1$failed
# an example of using extra
## Not run:
library(gamlss.tr)
data(tensile)
gen.trun(par=1,family="GA", type="right")
gen.trun(par=1,"LOGNO", type="right")
gen.trun(par=c(0,1),"TF", type="both")
ma<-fitDist(str, type="real@tol", extra=c("GAt", "LOGNOTr", "TFtr"), data=tensile)
## End(Not run)
```
Description

`fitted.gamlss` is the GAMLSS specific method for the generic function `fitted` which extracts fitted values for a specified parameter from a GAMLSS objects. `fitted.values` is an alias for it. The function `fv()` is similar to `fitted.gamlss()` but allows the argument what not to be character

Usage

```r
## S3 method for class 'gamlss'
fitted(object, what = c("mu", "sigma", "nu", "tau"), ...)
fv(obj, what = "mu", ...)
```

Arguments

- `object` a GAMLSS fitted model
- `obj` a GAMLSS fitted model
- `what` which parameter fitted values are required, default what="mu"
- `...` for extra arguments

Value

Fitted values extracted from the GAMLSS object for the given parameter.

Author(s)

Mikis Stasinopoulos <d.stasinopoulos@londonmet.ac.uk>

References


See Also

`print.gamlss`, `summary.gamlss`, `fitted.gamlss`, `coef.gamlss`, `residuals.gamlss`, `update.gamlss`, `plot.gamlss`, `deviance.gamlss`, `formula.gamlss`
**Examples**

```r
data(aids)
h <- gamlss(y~poly(x,3)+qrt, family=PO, data=aids) #
fitted(h)
r(m(h)
```

---

**fittedPlot**

*Plots The Fitted Values of a GAMLSS Model*

**Description**

This function, applicable only to a models with a single explanatory variable, plots the fitted values for all the parameters of a GAMLSS model against the (one) explanatory variable. It is also useful for comparing the fits for more than one model.

**Usage**

```r
fittedPlot(object, ..., x = NULL, color = TRUE, line.type = FALSE, xlab = NULL)
```

**Arguments**

- `object`: a fitted GAMLSS model object(with only one explanatory variable)
- `...`: optionally more fitted GAMLSS model objects
- `x`: The unique explanatory variable
- `color`: whether the fitted lines plots are shown in colour, `color=TRUE` (the default) or `not color=FALSE`
- `line.type`: whether the line type should be different or not. The default is `color=FALSE` or `not line.type=FALSE`
- `xlab`: the x-label

**Value**

A plot of the fitted values against the explanatory variable

**Author(s)**

Mikis Stasinopoulos <d.stasinopoulos@londonmet.ac.uk>, Bob Rigby <r.rigby@londonmet.ac.uk>
and Calliope Akantziliotou

**References**


See Also

gamlss, centiles, centiles.split

Examples

data(abdom)
h1<-gamlss(y~pb(x), sigma.formula=-x, family=BCT, data=abdom)
h2<-gamlss(y~pb(x), sigma.formula=-pb(x), family=8CT, data=abdom)
fittedPlot(h1,h2,x=abdom$x)
rm(h1,h2)

formula.gamlss

Extract the Model Formula in a GAMLSS fitted model

Description

formula.gamlss is the GAMLSS specific method for the generic function formula which extracts
the model formula from objects returned by modelling functions.

Usage

## S3 method for class 'gamlss'
formula(x, what = c("mu", "sigma", "nu", "tau"), ...)

Arguments

x                     a GAMLSS fitted model
what                   which parameter coefficient is required, default what="mu"
...                    for extra arguments

Value

Returns a model formula

Author(s)

Mikis Stasinopoulos <d.stasinopoulos@londonmet.ac.uk>

References

Rigby, R. A. and Stasinopoulos D. M. (2005). Generalized additive models for location, scale and

GAMLSS package in R. Accompanying documentation in the current GAMLSS help files, (see
also [http://www.gamlss.org/](http://www.gamlss.org/)).

See Also
gamlss, deviance.gamlss, fitted.gamlss

Examples
data(aids)
h<-gamlss(y~poly(x,3)+qrt, family=PO, data=aids) #
formula(h, "mu")
rm(h)

gamlss

Generalized Additive Models for Location Scale and Shape

Description

Returns an object of class "gamlss", which is a generalized additive model for location scale and shape (GAMLSS). The function gamlss() is very similar to the gam() function in S-plus (now also in R in package gam), but can fit more distributions (not only the ones belonging to the exponential family) and can model all the parameters of the distribution as functions of the explanatory variables (e.g. using linear, non-linear, smoothing, loess and random effects terms).

This implementation of gamlss() allows modelling of up to four parameters in a distribution family, which are conventionally called mu, sigma, nu and tau.

The function gamlssNews() shows what is new in the current implementation.

Usage
gamlss(formula = formula(data), sigma.formula = ~1,
  nu.formula = ~1, tau.formula = ~1, family = NO(),
  data = sys.parent(), weights = NULL,
  contrasts = NULL, method = RS(), start.from = NULL,
  mu.start = NULL, sigma.start = NULL,
  nu.start = NULL, tau.start = NULL,
  mu.fix = FALSE, sigma.fix = FALSE, nu.fix = FALSE,
  tau.fix = FALSE, control = gamlss.control(...),
  i.control = glim.control(...), ...)
is.gamlss(x)
gamlssNews()

Arguments
formula a formula object, with the response on the left of an ~ operator, and the terms, separated by + operators, on the right. Nonparametric smoothing terms are indicated by pb() for penalised beta splines, cs for smoothing splines, lo for loess smooth terms and random or ra for random terms, e.g. y~cs(x, df=5)+x1+x2+x3. Additional smoothers can be added by creating the appropriate interface. Interactions with nonparametric smooth terms are not fully supported, but will not produce errors; they will simply produce the usual parametric interaction
sigma.formula a formula object for fitting a model to the sigma parameter, as in the formula above, e.g. sigma.formula=~cs(x, df=5). It can be abbreviated to sigma.formula=~cs(x, df=5).

nu.formula a formula object for fitting a model to the nu parameter, e.g. nu.formula=~x

tau.formula a formula object for fitting a model to the tau parameter, e.g. tau.formula=~cs(x, df=2)

family a gamlss.family object, which is used to define the distribution and the link functions of the various parameters. The distribution families supported by gamlss() can be found in gamlss.family. Functions such as BI() (binomial) produce a family object. Also can be given without the parentheses i.e. BI. Family functions can take arguments, as in BI(mu.link=probit)

data a data frame containing the variables occurring in the formula. If this is missing, the variables should be on the search list. e.g. data=aids

weights a vector of weights. Note that this is not the same as in the glm() or gam() function. Here weights can be used to weight out observations (like in subset) or for a weighted likelihood analysis where the contribution of the observations to the likelihood differs according to weights. The length of weights must be the same as the number of observations in the data. By default, the weight is set to one. To set weights to vector w use weights=w

contrasts list of contrasts to be used for some or all of the factors appearing as variables in the model formula. The names of the list should be the names of the corresponding variables. The elements should either be contrast-type matrices (matrices with as many rows as levels of the factor and with columns linearly independent of each other and of a column of ones), or else they should be functions that compute such contrast matrices.

method the current algorithms for GAMLSS are RS(), CG() and mixed(). i.e. method=RS() will use the Rigby and Stasinopoulos algorithm, method=CG() will use the Cole and Green algorithm and mixed(2,10) will use the RS algorithm twice before switching to the Cole and Green algorithm for up to 10 extra iterations

start.from a fitted GAMLSS model which the fitted values will be used as staring values for the current model

mu.start vector or scalar of initial values for the location parameter mu e.g. mu.start=4

sigma.start vector or scalar of initial values for the scale parameter sigma e.g. sigma.start=1

nu.start vector or scalar of initial values for the parameter nu e.g. nu.start=3

tau.start vector or scalar of initial values for the location parameter tau e.g. tau.start=2

mu.fix whether the mu parameter should be kept fixed in the fitting processes e.g. mu.fix=FALSE

sigma.fix whether the sigma parameter should be kept fixed in the fitting processes e.g. sigma.fix=FALSE

nu.fix whether the nu parameter should be kept fixed in the fitting processes e.g. nu.fix=FALSE

tau.fix whether the tau parameter should be kept fixed in the fitting processes e.g. tau.fix=FALSE

control this sets the control parameters of the outer iterations algorithm. The default setting is the gamlss.control function

i.control this sets the control parameters of the inner iterations of the RS algorithm. The default setting is the glm.control function
... for extra arguments

Details

The Generalized Additive Model for Location, Scale and Shape is a general class of statistical models for a univariate response variable. The model assumes independent observations of the response variable $y$ given the parameters, the explanatory variables and the values of the random effects. The distribution for the response variable in the GAMLSS can be selected from a very general family of distributions including highly skew and/or kurtotic continuous and discrete distributions, see gamlss.family. The systematic part of the model is expanded to allow modelling not only of the mean (or location) parameter, but also of the other parameters of the distribution of $y$, as linear parametric and/or additive nonparametric (smooth) functions of explanatory variables and/or random effects terms. Maximum (penalized) likelihood estimation is used to fit the (non)parametric models. A Newton-Raphson/Fisher scoring algorithm is used to maximize the (penalized) likelihood. The additive terms in the model are fitted using a backfitting algorithm.

Value

Returns a gamlss object with components

- family: the distribution family of the gamlss object (see gamlss.family)
- parameters: the name of the fitted parameters i.e. mu, sigma, nu, tau
- call: the call of the gamlss function
- y: the response variable
- control: the gamlss fit control settings
- weights: the vector of weights
- G.deviance: the global deviance
- N: the number of observations in the fit
- rrqres: a function to calculate the normalized (randomized) quantile residuals of the object
- iter: the number of external iterations in the fitting process
- type: the type of the distribution or the response variable (continuous or discrete)
- method: which algorithm is used for the fit, RS(), CG() or mixed()
- converged: whether the model fitting has have converged
- residuals: the normalized (randomized) quantile residuals of the model
- mu.fv: the fitted values of the mu model, also sigma.fv, nu.fv, tau.fv for the other parameters if present
- mu.lp: the linear predictor of the mu model, also sigma.lp, nu.lp, tau.lp for the other parameters if present
- mu.wv: the working variable of the mu model, also sigma.wv, nu.wv, tau.wv for the other parameters if present
mu.wt the working weights of the mu model, also sigma.wt, nu.wt, tau.wt for the other parameters if present
mu.link the link function for the mu model, also sigma.link, nu.link, tau.link for the other parameters if present
mu.terms the terms for the mu model, also sigma.terms, nu.terms, tau.terms for the other parameters if present
mu.x the design matrix for the mu, also sigma.x, nu.x, tau.x for the other parameters if present
mu.qr the QR decomposition of the mu model, also sigma.qr, nu.qr, tau.qr for the other parameters if present
mu.coefficients the linear coefficients of the mu model, also sigma.coefficients, nu.coefficients, tau.coefficients for the other parameters if present
mu.formula the formula for the mu model, also sigma.formula, nu.formula, tau.formula for the other parameters if present
mu.df the mu degrees of freedom also sigma.df, nu.df, tau.df for the other parameters if present
mu.nl.df the non linear degrees of freedom, also sigma.nl.df, nu.nl.df, tau.nl.df for the other parameters if present
df.fit the total degrees of freedom use by the model
df.residual the residual degrees of freedom left after the model is fitted
aic the Akaike information criterion
sbc the Bayesian information criterion

Warning

Respect the parameter hierarchy when you are fitting a model. For example a good model for mu should be fitted before a model for sigma is fitted

Note

The following generic functions can be used with a GAMLSS object: print, summary, fitted, coef, residuals, update, plot, deviance, formula

Author(s)

Mikis Stasinopoulos <d.stasinopoulos@londonmet.ac.uk>, Bob Rigby <r.rigby@londonmet.ac.uk>, Calliope Akantziliotou and Vlasios Voudouris <vlasios.voudouris@abm-analytics.com>

References


See Also
gamlss.family, pdf.plot, find.hyper

Examples

data(abdom)
mod<-gamlss(y~pb(x), sigma.fo~pb(x), family=BCT, data=abdom, method=mixed(1,20))
plot(mod)
rm(mod)

<table>
<thead>
<tr>
<th>gamlss.control</th>
<th>Auxiliary for Controlling GAMLSS Fitting</th>
</tr>
</thead>
</table>

Description

Auxiliary function as user interface for gamlss fitting. Typically only used when calling gamlss function with the option control.

Usage

gamlss.control(c.crit = 0.001, n.cyc = 20, mu.step = 1, sigma.step = 1, nu.step = 1, tau.step = 1, gd.tol = 5, iter = 0, trace = TRUE, autostep = TRUE, save = TRUE, ...)

Arguments

c.crit the convergence criterion for the algorithm
n.cyc the number of cycles of the algorithm
mu.step the step length for the parameter mu
sigma.step the step length for the parameter sigma
nu.step the step length for the parameter nu
tau.step the step length for the parameter tau
gd.tol global deviance tolerance level
iter starting value for the number of iterations, typically set to 0 unless the function refit is used
trace whether to print at each iteration (TRUE) or not (FALSE)
autostep whether the steps should be halved automatically if the new global deviance is greater that the old one, the default is autostep=TRUE
save TRUE, (the default), saves all the information on exit. save=FALSE saves only limited information as the global deviance and AIC. For example fitted values, design matrices and additive terms are not saved. The latest is useful when gamlss() is called several times within a procedure.

... for extra arguments

Details

The step length for each of the parameters mu, sigma, nu or tau is very useful to aid convergence if the parameter has a fully parametric model. However using a step length is not theoretically justified if the model for the parameter includes one or more smoothing terms, (even thought it may give a very approximate result).

The c.crit can be increased to speed up the convergence especially for a large set of data which takes longer to fit. When ‘trace’ is TRUE, calls to the function cat produce the output for each outer iteration.

Value

A list with the arguments as components.

Author(s)

Mikis Stasinopoulos <d.stasinopoulos@londonmet.ac.uk>, Bob Rigby <r.rigby@londonmet.ac.uk>

References


See Also

gamlss

Examples

data(aids)
h<-gamlss(y=poly(x,3)+qrt, family=PO, data=aids) #
con<-gamlss.control(mu.step=0.1)
h<-gamlss(y=poly(x,3)+qrt, family=PO, data=aids, control=con) #
rm(h,con)
**gamlss.cs**

---

**Support for Function cs() and scs()**

---

**Description**

This is support for the functions cs(), and scs(). It is not intended to be called directly by users. The function `gamlss.cs` is using the R function `smooth.spline`.

**Usage**

```r
gamlss.cs(x, y, w, df = NULL, spar = NULL, xeval = NULL, ...)```

**Arguments**

- `x`  
  the design matrix
- `y`  
  the response variable
- `w`  
  prior weights
- `df`  
  effective degrees of freedom
- `spar`  
  spar the smoothing parameter
- `xeval`  
  used in prediction
- `...`  
  for extra arguments

**Value**

Returns a class "smooth.spline" object with

- `residuals`  
  The residuals of the fit
- `fitted.values`  
  The smoothing values
- `var`  
  the variance for the fitted smoother
- `lambda`  
  the final value for spar
- `nl.df`  
  the smoothing degrees of freedom excluding the constant and linear terms, i.e. (df-2)
- `coefSmo`  
  this is a list containing among others the knots and the coefficients

**Author(s)**

Mikis Stasinopoulos <d.stasinopoulos@londonmet.ac.uk>, Bob Rigby <r.rigby@londonmet.ac.uk>

**See Also**

`gamlss.cs`
Support for Function `fp()`

**Description**

Those are support for the functions `fp()` and `pp`. It is not intended to be called directly by users.

**Usage**

```r
gamlss.fp(x, y, w, npoly = 2, xeval = NULL)
gamlss.pp(x, y, w)
```

**Arguments**

- `x` the x for function `gamlss.fp` is referred to the design matrix of the specific parameter model (not to be used by the user)
- `y` the y for function `gamlss.fp` is referred to the working variable of the specific parameter model (not to be used by the user)
- `w` the w for function `gamlss.fp` is referred to the iterative weight variable of the specific parameter model (not to be used by the user)
- `npoly` a positive indicating how many fractional polynomials should be considered in the fit. Can take the values 1, 2 or 3 with 2 as default
- `xeval` used in prediction

**Value**

Returns a list with

- `fitted.values` fitted
- `residuals` residuals
- `var`
- `nl.df` the trace of the smoothing matrix
- `lambda` the value of the smoothing parameter
- `coefSmo` the coefficients from the smoothing fit
- `varcoeff` the variance of the coefficients

**Author(s)**

Mikis Stasinopoulos <d.stasinopoulos@londonmet.ac.uk>, Bob Rigby <r.rigby@londonmet.ac.uk>
References


See Also

gamlss, fp

---

gamlss.lo  Support for Function lo()

Description

This is support for the `loess` function `lo()`. It is not intended to be called directly by users. The function `gamlss.lo` is calling the R function `loess`.

Usage

gamlss.lo(x, y, w, xeval = NULL, ...)

Arguments

- **x**: the design matrix
- **y**: the response variable
- **w**: prior weights
- **xeval**: used in prediction
- **...**: further arguments passed to or from other methods.

Value

Returns an object

- **fitted**: the smooth values
- **residuals**: the residuals
- **var**: the variance of the smoother
- **nl.df**: the non-linear degrees of freedom
- **coeffSmo**: with value NULL
- **lambda**: the value of span
Author(s)
Mikis Stasinopoulos based on Brian Ripley implementation of loess function in R

See Also
gamlss, lo

Description
Those functions are support for the functions pb(), pbo(), ps(), ridge(), ri(), cy(), pvc(), and pbm(). The functions are not intended to be called directly by users.

Usage
gamlss.pb(x, y, w, xeval = NULL, ...)
gamlss.pbo(x, y, w, xeval = NULL, ...)
gamlss.ps(x, y, w, xeval = NULL, ...)
gamlss.ridge(x, y, w, xeval = NULL, ...)
gamlss.ri(x, y, w, xeval = NULL, ...)
gamlss.cy(x, y, w, xeval = NULL, ...)
gamlss.pvc(x, y, w, xeval = NULL, ...)
gamlss.pbm(x, y, w, xeval = NULL, ...)

Arguments

x the x for function gamlss.fp is referred to the design matric of the specific parameter model (not to be used by the user)

y the y for function gamlss.fp is referred to the working variable of the specific parameter model (not to be used by the user)

w the w for function gamlss.fp is referred to the iterative weight variable of the specific parameter model (not to be used by the user)

xeval used in prediction

... further arguments passed to or from other methods.

Value
All function return fitted smoothers.

Author(s)
Mikis Stasinopoulos <d.stasinopoulos@londonmet.ac.uk>, Bob Rigby <r.rigby@londonmet.ac.uk>
References


See Also

`gamlss`, `pb`, `ps`, `ri`, `ridge`, `cy`, `pvc`, `pbm`

gamlss.random  Support for Functions random() and re()

Description

This is support for the functions `random()` and `re()` respectively. It is not intended to be called directly by users.

Usage

```r
\texttt{gamlss.random(x, y, w)}
\texttt{gamlss.re(x, y, w, xeval = NULL, \ldots)}
```

Arguments

- `x` the explanatory design matrix
- `y` the response variable
- `w` iterative weights
- `xeval` it used internally for prediction
- `\ldots` for extra arguments

Value

Returns a list with

- `y` the fitted values
- `residuals` the residuals
- `var` the variance of the fitted values
- `lambda` the final lambda, the smoothing parameter
- `coefSmo` with value NULL
gamlss.scope

**Generate a Scope Argument for Stepwise GAMLSS**

**Description**
Generate a scope argument for a stepwise GAMLSS.

**Usage**
```r
gamlss.scope(frame, response = 1, smoother = "cs", arg = NULL, form = TRUE)
```

**Arguments**
- `frame` a data or model frame
- `response` which variable is the response; the default is the first
- `smoother` what smoother to use; default is `cs`
- `arg` any additional arguments required by the smoother
- `form` should a formula be returned (default), or else a character version of the formula

**Details**
Each formula describes an ordered regimen of terms, each of which is eligible on their own for inclusion in the gam model. One of the terms is selected from each formula by step.gam. If a 1 is selected, that term is omitted.
**Value**

A list of formulas is returned, one for each column in frame (excluding the response). For a numeric variable, say x1, the formula is

\~ 1 + x1 + cs(x1)

If x1 is a factor, the last smooth term is omitted.

**Author(s)**

Mikis Stasinopoulos: a modified function from Statistical Models in S

**References**


**See Also**

*stepGAIC*

**Examples**

```r
data(usair)
gs1<-gamlss.scope(model.frame(y~x1+x2+x3+x4+x5+x6, data=usair))
gs2<-gamlss.scope(model.frame(usair))
gs1
gs2
gs3<-gamlss.scope(model.frame(usair), smooth="fp", arg="3")
gs3
```

---

**gamlssML**

*Maximum Likelihood estimation of a simple GAMLSS model*

**Description**

This is a function for fitting a `gamlss.family` distribution to single data set using a non linear maximisation algorithm in R. This is relevant only when there are not explanatory variables.
Usage

gamlssML(y, family = NO, weights = NULL, mu.start = NULL, sigma.start = NULL, nu.start = NULL, tau.start = NULL, mu.fix = FALSE, sigma.fix = FALSE, nu.fix = FALSE, tau.fix = FALSE, data = NULL, start.from = NULL, ...)

Arguments

y a vector of data requiring the fit of a \texttt{gamlss.family} distribution or a formula, for example, \texttt{y~1} (explanatory variables are ignored).

family \texttt{gamlss.family} object, which is used to define the distribution and the link functions of the various parameters. The distribution families supported by \texttt{gamlssML()} can be found in \texttt{gamlss.family}

weights a vector of weights. Here weights can be used to weight out observations (like in \texttt{subset}) or for a weighted likelihood analysis where the contribution of the observations to the likelihood differs according to weights. The length of \texttt{weights} must be the same as the number of observations in the data. By default, the weight is set to one. To set weights to vector \texttt{w} use \texttt{weights\[w}

mu.start a scalar of initial values for the location parameter \(\mu\) e.g. \texttt{mu.start=4}

sigma.start a scalar of initial values for the scale parameter \(\sigma\) e.g. \texttt{sigma.start=1}

nu.start scalar of initial values for the parameter \(\nu\) e.g. \texttt{nu.start=3}

tau.start scalar of initial values for the parameter \(\tau\) e.g. \texttt{tau.start=3}

mu.fix whether the \(\mu\) parameter should be kept fixed in the fitting processes e.g. \texttt{mu.fix=FALSE}

sigma.fix whether the \(\sigma\) parameter should be kept fixed in the fitting processes e.g. \texttt{sigma.fix=FALSE}

nu.fix whether the \(\nu\) parameter should be kept fixed in the fitting processes e.g. \texttt{nu.fix=FALSE}

tau.fix whether the \(\tau\) parameter should be kept fixed in the fitting processes e.g. \texttt{tau.fix=FALSE}

data a data frame containing the variable \(y\). If this is missing, the variable should be on the search list e.g. \texttt{data=aids}

start.from a \texttt{gamlss} object to start from the fitting or vector of length as many parameters in the distribution

... for extra arguments

Details

This function which fits a \texttt{gamlss.family} distribution to a single data set is using a non linear maximisation. in fact it uses the internal function \texttt{MLE()} which is a copy of the \texttt{mle()} function of package \texttt{stat4}. The function \texttt{gamlssML()} could be for large data faster than the equivalent \texttt{gamlss()} function which is designed for regression type of models.

Value

Returns a \texttt{gamlssML} object which behaves like a \texttt{gamlss} fitted object.
gen.likelihood

Author(s)

Mikis Stasinopoulos <d.stasinopoulos@londonmet.ac.uk>, Bob Rigby <r.rigby@londonmet.ac.uk>, Vlasis Voudouris <v.voudouris@londonmet.ac.uk> and Majid Djennad <m.djennad.londonmet.ac.uk>

References


See Also
gamlss.family, gamlss

Examples

```
#-------- negative binomial 1000 observations
y<- rNBI(1000)
system.time(m1<-gamlss(y~1, family=NBI))
  system.time(m1a<-gamlss(y~1, family=NBI, trace=FALSE))
  system.time(m1k<-gamlssML(y, family=NBI))
AIC(m1,m1a,m1k, k=0)
# neg. binomial  n=10000
y<- rNBI(10000)
system.time(m2<-gamlss(y~1, family=NBI))
  system.time(m2a<-gamlss(y~1, family=NBI, trace=FALSE))
  system.time(m2k<-gamlssML(y, family=NBI))
AIC(m2,m2a,m2k, k=0)
# binomial type data
data(aep)
m1 <- gamlssML(aep$y, family=BB) # ok
m2 <- gamlssML(y, data=aep, family=BB) # ok
m3 <- gamlssML(y-1, data=aep, family=BB) # ok but not
  #gamlssML(aep$y-1, family=BB)
```
Usage

`gen.likelihood(object)`

Arguments

object A gamlss fitted model

Details

The purpose of this function is to help the function `vcov()` to get the right Hessian matrix after a model has fitted. Note that at the moment smoothing terms are considered as fixed.

Value

A function of the log-likelihood

Author(s)

Mikis Stasinopoulos <d.stasinopoulos@londonmet.ac.uk> Bob Rigby <r.rigby@londonmet.ac.uk> and Vlasios Voudouris

References


See Also

`vcov`

Examples

data(aids)
m1 <- gamlss(y~x+qrt, data=aids, family=NBI)
logl<-gen.likelihood(m1)
logl()
loglik(m1)
getSmo

Extracting Smoother information from a GAMLSS fitted object

Description

The function `getSmo()` extracts information from a fitted smoothing additive term.

Usage

```r
getSmo(object, what = c("mu", "sigma", "nu", "tau"),
       which = 1)
```

Arguments

- `object`: a GAMLSS fitted model
- `what`: which distribution parameter is required, default `what="mu"`
- `which`: which smoothing term

Details

This function facilitates the extraction of information from a fitted additive terms. For example `getSmo(m1,"sigma",2)` is equivalent of `m1$coefSmo[[2]]`. To get the actual fitted values type `m1$sigma.s[[2]]`

Value

A list containing information about a fitted smoother or a fitted objects

Author(s)

Mikis Stasinopoulos and Bob Rigby

References


glim.control

Examples

data(usair)
t1<-gamlss(y-x1+pb(x5)+pb(x6), data=usair, family=GAMLSS)
# get the value for lambda for the second fitted term in mu
getsmo(t1, "mu", 2)$lambda

---

glim.control

Auxiliary for Controlling the inner algorithm in a GAMLSS Fitting

Description

Auxiliary function used for the inner iteration of gamlss algorithm. Typically only used when calling gamlss function through the option i.control.

Usage

glim.control(cc = 0.001, cyc = 50, glm.trace = FALSE,
bf.cyc = 30, bf.tol = 0.001, bf.trace = FALSE,
...)

Arguments

- cc: the convergence criterion for the algorithm
- cyc: the number of cycles of the algorithm
- glm.trace: whether to print at each iteration (TRUE) or not (FALSE)
- bf.cyc: the number of cycles of the backfitting algorithm
- bf.tol: the convergence criterion (tolerance level) for the backfitting algorithm
- bf.trace: whether to print at each iteration (TRUE) or not (FALSE, the default)
- ...: for extra arguments

Value

A list with the arguments as components

Author(s)

Mikis Stasinopoulos <d.stasinopoulos@londonmet.ac.uk>, Bob Rigby <r.rigby@londonmet.ac.uk>
References


See Also

`gamlss`

Examples

```r
data(aids)
con<-glm.control(glm.trace=TRUE)
h<-gamlss(y~poly(x,3)+qrt, family=PO, data=aids, i.control=con) #
rm(h,con)
```

**histDist**

This function plots the histogram and a fitted (GAMLSS family) distribution to a variable

Description

This function fits constants to the parameters of a GAMLSS family distribution and them plot the histogram and the fitted distribution.

Usage

```r
histDist(y, family = NO, freq = NULL,
       density = FALSE, nbins = 10, xlim = NULL,
       ylim = NULL, main = NULL, xlab = NULL,
       ylab = NULL, data = NULL, ...)
```

Arguments

- `y` a vector for the response variable
- `family` a `gamlss.family` distribution
- `freq` the frequencies of the data in `y` if exist. `freq` is used as weights in the `gamlss` fit
density
density default value is FALSE. Change to TRUE if you would like a non-parametric
density plot together with the parametric fitted distribution plot (for continuous
variable only)

nbins
The suggested number of bins (argument passed to truehist() of package
MASS). Either a positive integer, or a character string naming a rule: "Scott"
or "Freedman-Diaconis" or "FD". (Case is ignored.)

xlim
the minimum and the maximum x-axis value (if the default values are out of
range)

ylim
the minimum and the maximum y-axis value (if the default values are out of
range)

main
the main title for the plot

xlab
the label in the x-axis

ylab
the label in the y-axis

data
the data.frame

... for extra arguments to be passed to the gamlss function

Details
This function first fits constants for each parameters of a GAMLSS distribution family using the
gamlss function and them plots the fitted distribution together with the appropriate plot according
to whether the y variable is of a continuous or discrete type. Histogram is plotted for continuous
and barplot for discrete variables. The function truehist of Venables and Ripley’s MASS package
is used for the histogram plotting.

Value
returns a plot

Author(s)
Mikis Stasinopoulos

References
Rigby, R. A. and Stasinopoulos D. M. (2005). Generalized additive models for location, scale and

GAMLSS package in R. Accompanying documentation in the current GAMLSS help files, (see
also http://www.gamlss.org/).

org/v23/i07.

See Also
gamlss, gamlss.family
### Examples

```r
data(abdom)
histDist(y, family="NO", data=abdom)
  # use the ylim
histDist(y, family="NO", ylim=c(0, 0.005), data=abdom)
  # bad fit use PE
histDist(y, family="PE", ymax=0.005, data=abdom)
  # discrete data counts
  # Hand at al. p150  Leptinotarsa decemlineata
y <- c(0, 1, 2, 3, 4, 6, 7, 8, 10, 11)
freq <- c(3, 12, 5, 6, 5, 2, 2, 1, 2)
histDist(y, "NB1", freq=freq)
  # the same as
histDist(rep(y, freq), "NB1")
```

### Description

This set of functions use the old Poisson trick of discretising the data and then fitting a Poisson error model to the resulting frequencies (Lindsey, 1997). Here the model fitted is a smooth cubic spline curve. The result is a density estimator for the data.

### Usage

```r
histSmo(y, lambda = NULL, df = NULL, order = 3, lower = NULL,
        upper = NULL, type = c("freq", "prob"),
        save = FALSE, plot = TRUE, breaks = NULL,
        discrete = FALSE, ...)
histSmoC(y, df = 10, lower = NULL, upper = NULL, type = c("freq", "prob"),
        save = FALSE, plot = TRUE, breaks = NULL,
        discrete = FALSE, ...)
histSmoO(y, lambda = 1, order = 3, lower = NULL, upper = NULL,
        type = c("freq", "prob"), save = FALSE,
        plot = TRUE, breaks = NULL,
        discrete = FALSE, ...)
histSmoP(y, lambda = NULL, df = NULL, order = 3, lower = NULL,
        upper = NULL, type = c("freq", "prob"), save = FALSE,
        plot = TRUE, breaks = NULL, discrete = FALSE,
        ...)
```

### Arguments

- **y**: the variable of interest
- **lambda**: the smoothing parameter
- **df**: the degrees of freedom
order the order of the P-spline
lower the lower limit of the y-variable
upper the upper limit of the y-variable
type the type of histogram
save whether to save the results
plot whether to plot the resulting density estimator
breaks the number of break points to be used in the histogram and consequently the number of observations in the Poisson fit
discrete whether to treat the fitting density as a discrete distribution or not
... further arguments passed to or from other methods.

Details

Here are the methods used here:

i) The function histSmoO() uses Penalised discrete splines (Eilers, 2003). This function is appropriate when the smoothing parameter is fixed.

ii) The function histSmoC() uses smooth cubic splines and fits a Poisson error model to the frequencies using the cs() additive function of GAMLSS. This function is appropriate if the effective degrees of freedom are fixed in the model.

iii) The function histSmoP() uses Penalised cubic splines (Eilers and Marx 1996). It is fitting a Poisson model to the frequencies using the pb() additive function of GAMLSS. This function is appropriate if automatic selection of the smoothing parameter is required.

iv) The function histSmo() combines all the above functions in the sense that if lambda is fixed it uses histSmoO(), if the df’s are fixed it uses codehistSmoC() and if none of these is specified it uses histSmoP().

Value

Returns a histSmo S3 object. The object has the following components:

x the middle points of the discretise data
counts how many observation are on the discretise intervals
density the density value for each discrete interval
hist the hist object used to discretise the data
cdf The resulting cumulative distribution function useful for calculating probabilities from the estimate density
nvCDF The inverse cumulative distribution function
model The fitted Poisson model only for histSmoP() and histSmoC()

Author(s)

Mikis Stasinopoulos, Paul Eilers, Bob Rigby and Vlasios Voudouris
References


See Also

pb, cs

Examples

```r
# creating data from Pareto 2 distribution
y <- rPARETO2(1000)
## Not run:
# getting the density
histSmo(y, lower=0)
# more breaks a bit slower
histSmo(y, breaks=200, lower=0)
# quick fit using lambda
histSmo(y, lambda=1, breaks=200, lower=0)
# or
histSmo(y, lambda=1, breaks=200, lower=0)
# quick fit using df
histSmo(y, df=15, breaks=200, lower=0)
# or
histSmo(y, df=15, breaks=200, lower=0)
# saving results
m1<- histSmo(y, lower=0, save=TRUE)
plot(m1)
plot(m1, "cdf")
plot(m1, "invcdf")
# now generate from SHASH
YY <- rSHASH(1000)
m1<- histSmo(YY, save=TRUE)
# calculate Pr(YY>10)
1-m1$cdf(10)
# calculate Pr(-10<YY<10)
1-(1-m1$cdf(10))-m1$cdf(-10)
YYY <- rNBI(1000, mu=5, sigma=4)
histDist(YYY, discrete=TRUE, family=NBI())
## End(Not run)
```
IC

Gives the GAIC for a GAMLSS Object

Description

IC is a function to calculate the Generalised Akaike information criterion (GAIC) for a given penalty k for a fitted GAMLSS object. The function AIC,gamlss is the method associated with a GAMLSS object of the generic function AIC. The function GAIC is a synonymous of the function AIC,gamlss. The function extractAIC is a the method associated a GAMLSS object of the generic function extractAIC and it is mainly used in the stepAIC function. The function Rsq compute a generalisation of the R-squared for not normal models.

Usage

IC(object, k = 2)
## S3 method for class 'gamlss'
AIC(object, ..., k = 2, c = FALSE)
GAIC(object, ..., k = 2, c = FALSE )
## S3 method for class 'gamlss'
extractAIC(fit, scale, k = 2, c = FALSE, ...)

Arguments

- object: an gamlss fitted model
- fit: an gamlss fitted model
- ...: allows several GAMLSS object to be compared using a GAIC
- k: the penalty with default k=2.5
- c: whether the corrected AIC, i.e. AICc, should be used, note that it applies only when k=2
- scale: this argument is not used in gamlss

Value

The function IC returns the GAIC for given penalty k of the GAMLSS object. The function AIC returns a matrix contains the df’s and the GAIC’s for given penalty k. The function GAIC returns identical results to AIC. The function extractAIC returns vector of length two with the degrees of freedom and the AIC criterion.

Author(s)

Mikis Stasinopoulos <d.stasinopoulos@londonmet.ac.uk>
References


See Also

`gamlss`

Examples

```r
data(abdom)
mod1<-gamlss(y~pb(x),sigma.fo=pb(x),family=BCT, data=abdom)
IC(mod1)
mod2<-gamlss(y~pb(x),sigma.fo=x,family=BCT, data=abdom)
AIC(mod1,mod2,k=3)
GAIC(mod1,mod2,k=3)
extractAIC(mod1,mod2)
rm(mod1,mod2)
```

```r
lms
A function to fit LMS curves for centile estimation
```

Description

This function is design to help the user to easily construct growth curve centile estimation. It is applicable when only "one" explanatory variable is available (usually age).

Usage

```r
lms(y, x, families = LMS, data = NULL, k = 2,
    cent = 100 * pnorm((-4:4) * 2/3),
    calibration = TRUE, trans.x = FALSE,
    fix.power = NULL, lim.trans = c(0, 1.5),
    prof = FALSE, step = 0.1, legend = FALSE,
    mu.df = NULL, sigma.df = NULL, nu.df = NULL,
    tau.df = NULL, c.crit = .01,
    method.pb = c("ML", "GAIC"), ...)
```
Arguments

y
The response variable
x
The unique explanatory variable
families
a list of gamlss.families with default LMS=c("BCCGo", "BCPEo", "BCTo")
data
the data frame
k
the penalty to be used in the GAIC
cent
a vector with elements the % centile values for which the centile curves have to be evaluated
calibration
whether calibration is required with default TRUE
trans.x
whether to check for transformation in x with default FALSE
fix.power
if set it fix the power of the transformation for x
lim.trans
the limits for the search of the power parameter for x
prof
whether to use the profile GAIC of the power tranformation
step
if codeprof=TRUE is used this determine the step for the profile GAIC
legend
whether a legend is required in the plot with default FALSE
mu.df
mu effective degrees of freedom if required otherwise are estimated
sigma.df
sigma effective degrees of freedom if required otherwise are estimated
nu.df
nu effective degrees of freedom if required otherwise are estimated
tau.df
tau effective degrees of freedom if required otherwise are estimated
c.crit
the convergence criterion to be pass to gamlss()
method.pb
the method used in the pb() for estimating the smoothing parameters. The default is local maximum likelihood "ML". "GAIC" is also permitted where k is taken from the k argument of the function.

... extra argument which can be passed to gamlss()

Details

This function should be used if the construction of the centile curves involves only one explanatory variable.

The model assumes that the response variable has a flexible distribution i.e. \( y \sim D(\mu, \sigma, \nu, \tau) \) where the parameters of the distribution are smooth functions of the explanatory variable i.e. \( g(\mu) = s(x) \), where \( g() \) is a link function and \( s() \) is a smooth function. Occasionally a power transformation in the x-axis helps the construction of the centile curves. That is, in this case the parameters are modelled by \( x^p \) rather than just x, i.e.\( g(\mu) = s(x^p) \). The function lms() uses P-splines (pb()) as a smoother.

If a transformation is needed for x the function lms() starts by finding an optimum value for p using the simple model \( NO(\mu = s(x^p)) \). (Note that this value of p is not the optimum for the final chosen model but it works well in practice.)

After fitting a Normal error model for staring values the function proceeds by fitting several "appropriate" distributions for the response variable. The set of gamlss.familiy distributions to fit is specified by the argument families. The default families arguments is LMS=c("BCCGo", "BCPEo", "BCTo")
that is the LMS class of distributions, Cole and Green (1992). Note that this class is only appropriate when \( y \) is positive (with no zeros). If the response variable contains negative values and zeros then use the argument `families=theSHASH` where `theSHASH <- c("NO", "SHASHo")` or add any other list of distributions which you may think is appropriate. Justification of using the specific centile (0.38 2.27 9.1211220 25.25, 50, 74.75, 90.88, 97.72, 99.62) is given in Cole (1994).

**Value**

It returns a `gamlss` fitted object

**Note**

The function is fitting several models and for large data can be slow

**Author(s)**

Mikis Stasinopoulos `<d.stasinopoulos@londonmet.ac.uk>`, Bob Rigby `<r.rigby@londonmet.ac.uk>` and Vlasios Voudouris `<vlasios.voudouris@abm-analytics.com>`

**References**


**See Also**

gamlss, centiles, calibration

**Examples**

```r
## Not run:
data(abdom)
m1 <- lms(y,x, data=abdom, n.cyc=30)
m2 <- lms(y,x ,data=abdom, method.pb="GAIC", k=log(610))
# this example takes time
data(db)
m1 <- lms(y=head, x=age, data=db, trans.x=TRUE)
```

## End(Not run)
Specify a loess fit in a GAMLSS formula

Description

Allows the user to specify a loess fit in a GAMLSS model. This function is similar to the lo function in the gam implementation of package gam see Chambers and Hastie (1991).

Usage

lo(formula, control = lo.control(...), ...)
lo.control(span = 0.75, enp.target = NULL,
    degree = 2, parametric = FALSE, drop.square = FALSE,
    normalize = TRUE, family = c("gaussian", "symmetric"),
    method = c("loess", "model.frame"),
    surface = c("interpolate", "direct"),
    statistics = c("approximate", "exact"),
    trace.hat = c("exact", "approximate"),
    cell = 0.2, iterations = 4, ...)

Arguments

formula a formula specifying the explanatory variables
control a control to be passed to the loess function
... extra arguments
span the number of observations in a neighbourhood. This is the smoothing parameter for a loess fit.
enp.target an alternative way to specify span, as the approximate equivalent number degrees of freedom to be used. See also the help file of the R function loess. For consistency with the older version of lo the effective degrees of freedom df can be also specified instead of span, e.g. df=5
degree the degree of local polynomial; can be 1 or 2. See also the help file of loess
parametric should any terms be fitted globally rather than locally? See the help file of loess
drop.square for fits with more than one predictor and degree=2, should the quadratic term be dropped for particular predictors?. See also help file of loess
normalize should the predictors be normalized to a common scale if there is more than one? See the help file of loess
family if "gaussian" fitting is by least-squares, and if "symmetric" a re-descending M estimator is used with Tukey's biweight function. See the help file of loess
method fit the model or just extract the model frame. See the help file of loess
surface should the fitted surface be computed exactly or via interpolation from a kd tree? See also the help file of loess.control
statistics should the statistics be computed exactly or approximately? See the help file of loess.control
trace.hat  should the trace of the smoother matrix be computed exactly or approximately?  
See the help file of loess.control

cell  if interpolation is used this controls the accuracy of the approximation via 
the maximum number of points in a cell in the kd tree. See the help file of loess.control

iterations  the number of iterations used in robust fitting. See the help file of loess.control

Details

Note that lo itself does no smoothing; it simply sets things up for the function gamlss.lo() which 
is used by the backfitting function gamlss.add().

Value

a loess object is returned.

Warning

In this version the first argument is a formula NOT a list as in the previous one

Note

Note that lo itself does no smoothing; it simply sets things up for gamlss.lo() to do the backfitting.

Author(s)

Mikis Stasinopoulos <d.stasinopoulos@londonmet.ac.uk>, Bob Rigby <r.rigby@londonmet.ac.uk>, 
(The original lo() function was based on the Trevor Hastie’s S-plus lo() function. See also the 
documentation of the loess function for the authorship of the function.

References


Rigby, R. A. and Stasinopoulos D. M. (2005). Generalized additive models for location, scale and 

GAMLSS package in R. Accompanying documentation in the current GAMLSS help files, (see 
also http://www.gamlss.org/).


See Also

cs, random,
Examples

data(aids)
attach(aids)
# fitting a loess curve with span=0.4 plus the a quarterly effect
aids1<-gamlss(y~lo(~x,span=0.4)+qrt,data=aids,family=PO) #
plot(x,y)
lines(x,fitted(aids1))
rm(aids1)
detach(aids)

loglogSurv

Log-Log Survival function plots for checking the tail behaviour of the data

Description

The log-log Survival functions are design for checking the tails of a single response variable (no explanatory should be involved). There are three different function:

a) the function loglogSurv1() which plot the (left or right) tails of the empirical log-log Survival function against loglog(y), where y is the variable of interest. The coefficient of a linear fit to the plot can be used an estimated for Type I tails.

b) the function loglogSurv2() which plot the (left or right) tails of the empirical log-log Survival function against log(y). The coefficient of a linear fit to the plot can be used an estimated for Type II tails.

c) the function loglogSurv3() which plot the (left or right) tails of the empirical log-log Survival function against y. The coefficient of a linear fit to the plot can be used an an estimated for Type III tails.

The function loglogSurv() combines all the above functions.

The function logSurv() is also design for exploring the tails of a single response variable. It plots the empirical log-survival function against log(y) for specified percentage of the tail and fits a linear, quadratic and exponential curve to the points of the plot. For distributions defined on the positive real line a good linear fit would indicate a Pareto type tail, a good quadratic fit a log-normal type tail and good exponential fit a Weibull type tail. Note that this function is only appropriate to investigate rather heavy tails and it is not very good to discriminate between different type of tails, as the loglogSurv().

Usage

loglogSurv(y, percentage = 10, howmany = NULL, type = c("right", "left"),
plot = TRUE, print = TRUE, save = FALSE)
loglogSurv1(y, percentage = 10, howmany = NULL, type = c("right", "left"),
plot = TRUE, print = TRUE, save = FALSE)
loglogSurv2(y, percentage = 10, howmany = NULL, type = c("right", "left"),
plot = TRUE, print = TRUE, save = FALSE)
loglogSurv3(y, percentage = 10, howmany = NULL, type = c("right", "left"),
plot = TRUE, print = TRUE, save = FALSE)
loglogSurv(y, percentage = 10, howmany = NULL, type = c("right", "left"),
plot = TRUE, print = TRUE, save = FALSE)

Arguments

- **y**: a vector, the variable of interest
- **percentage**: what percentage of the tail need to be modelled, default is 10%
- **howmany**: how many observations in the tail needed. This is an alternative to percentage. If it specified it take over from the percentage argument otherwise percentage is used.
- **type**: which tall needs checking the right (default) of the left
- **plot**: whether to plot with default equal TRUE
- **print**: whether to print the coefficients with default equal TRUE
- **save**: whether to save the fitted linear model with default equal FALSE

Details

The functions loglogSurv1(), loglogSurv3() and loglogSurv3() take the upper (or lower) part of an ordered variable create its empirical survival function and plot the log-log of this functions against log(log(y)), log(y) and y respectively. Then they fit a line to the plot. The coefficients of the line can be interpreted as parameters determined the behaviour of the tail. More details can be found in Chapter 6 of "The Distribution Toolbox of GAMLSS" book which can be found in http://www.gamlss.org/

Value

A plot

Author(s)

Bob Rigby, Mikis Stasinopoulos and Vlassios Voudouris

References


Examples

data(film90)
F90 <- film90$borev0
op<-par(mfrow=c(3,1))
loglogSurv1(F90)
loglogSurv2(F90)
loglogSurv3(F90)
par(op)
loglogSurv(F90)

logSurv(F90)

lpred

Extract Linear Predictor Values and Standard Errors For A GAMLSS Model

Description

lpred is the GAMLSS specific method which extracts the linear predictor and its (approximate) standard errors for a specified parameter from a GAMLSS objects. The lpred can be also used to extract the fitted values (with its approximate standard errors) or specific terms in the model (with its approximate standard errors) in the same way that the predict.lm() and predict.glm() functions can be used for lm or glm objects. The function lp extract only the linear predictor. If prediction is required for new data values then use the function predict.gamlss().

Usage

lpred(obj, what = c("mu", "sigma", "nu", "tau"),
      type = c("link", "response", "terms"),
      terms = NULL, se.fit = FALSE, ...)

lp(obj, what = "mu", ...)

Arguments

obj a GAMLSS fitted model
what which distribution parameter is required, default what="mu"


type
type="link" (the default) gets the linear predictor for the specified distribution parameter. type="response" gets the fitted values for the parameter while type="terms" gets the fitted terms contribution

terms if type="terms", which terms to be selected (default is all terms)


se.fit if TRUE the approximate standard errors of the appropriate type are extracted

... for extra arguments

Value

If se.fit=FALSE a vector (or a matrix) of the appropriate type is extracted from the GAMLSS object for the given parameter in what. If se.fit=TRUE a list containing the appropriate type, fit, and its (approximate) standard errors, se.fit.
Author(s)
Mikis Stasinopoulos

References


See Also
predict.gamlss

Examples

data(aids)
mod<-gamlss(y=poly(x,3)+qrt, family=PO, data=aids) #
mod.t <- lpred(mod, type = "terms", terms = "qrt")
mod.t
mod.lp <- lp(mod)
mod.lp
rm(mod, mod.t,mod.lp)

LR.test  

Likelihood Ratio test for nested GAMLSS models

Description
The function performs a likelihood ratio test for two nested fitted model.

Usage
LR.test(null, alternative, print = TRUE)

Arguments

null The null hypothesis (simpler) fitted model
alternative The alternative hypothesis (more complex) fitted model
print whether to print or save the result
Details

Warning: no checking whether the models are nested is performed.

Value

If print=FALSE a list with chi, df and p.val is produced.

Author(s)

Mikis Stasinopoulos <d.stasinopoulos@londonmet.ac.uk>

References


See Also

gamlss, dropterm

Examples

data(usair)
m0<-gamlss(y~x1+x2, data=usair)
m1<-gamlss(y~x1+x2+x3+x4, data=usair)
LR.test(m0,m1)

model.frame.gamlss

Extract a model.frame, a model matrix or terms from a GAMLSS object for a given distributional parameter

Description

model.frame.gamlss, model.matrix.gamlss and terms.gamlss are the gamlss versions of the generic functions model.frame, model.matrix and terms respectively.
Usage

```r
## S3 method for class 'gamlss'
model.frame(formula, what = c("mu", "sigma", "nu", "tau"), ...)
## S3 method for class 'gamlss'
terms(x, what = c("mu", "sigma", "nu", "tau"), ...)
## S3 method for class 'gamlss'
model.matrix(object, what = c("mu", "sigma", "nu", "tau"), ...)
```

Arguments

- `formula`: a gamlss object
- `x`: a gamlss object
- `object`: a gamlss object
- `what`: for which parameter to extract the model.frame, terms or model.frame
- `...`: for extra arguments

Value

a model.frame, a model.matrix or terms

Author(s)

Mikis Stasinopoulos

References


See Also

- `gamlss`

Examples

```r
data(aids)
mod <- gamlss(y~poly(x,3)+qrt, family=P0, data=aids) #
model.frame(mod)
model.matrix(mod)
terms(mod, "mu")
rm(mod)
```
par.plot  

A function to plot parallel plot for repeated measurement data

Description

This function can be used to plot parallel plots for each individual in a repeated measurement study. It is based on the coplot() function of R.

Usage

par.plot(formula = NULL, data = NULL, subjects = NULL, color = TRUE, show.given = TRUE, ...)

Arguments

- **formula**: a formula describing the form of conditioning plot. A formula of the form \( y \sim x | a \) indicates that plots of \( y \) versus \( x \) should be produced conditional on the variable \( a \). A formula of the form \( y \sim x | a \times b \) indicates that plots of \( y \) versus \( x \) should be produced conditional on the two variables \( a \) and \( b \).
- **data**: a data frame containing values for any variables in the formula. This argument is compulsory.
- **subjects**: a factor which distinguishes between the individual participants.
- **color**: whether the parallel plot are shown in colour, color=TRUE (the default) or not color=FALSE
- **show.given**: logical (possibly of length 2 for 2 conditioning variables): should conditioning plots be shown for the corresponding conditioning variables (default 'TRUE')
- **...**: for extra arguments

Value

It returns a plot.

Note

Note that similar plot can be found in the library nlme by Pinheiro and Bates

Author(s)

Mikis Stasinopoulos <d.stasinopoulos@londonmet.ac.uk>
References


See Also
gamlss

Examples

```r
library(nlme)
data(Orthodont)
par.plot(distance~age, data=Orthodont, subject=Subject)
par.plot(distance~age, data=Orthodont, subject=Subject)
par.plot(distance~age, data=Orthodont, subject=Subject, show.given=FALSE)
```

pdf.plot

*Plots Probability Distribution Functions for GAMLSS Family*

Description

A function to plot probability distribution functions (pdf) belonging to the gamlss family of distributions. This function allows either plotting of the fitted distributions for up to eight observations or plotting specified distributions belonging in the gamlss family

Usage

```r
pdf.plot(obj = NULL, obs = c(1), family = NO(), mu = NULL,
    sigma = NULL, nu = NULL, tau = NULL, min = NULL,
    max = NULL, step = NULL, allinone = FALSE,
    no.title = FALSE, ...)
```

Arguments

- **obj**  
  An gamlss object e.g. `obj=model1` where `model1` is a fitted gamlss object
- **obs**  
  A number or vector of up to length eight indicating the case numbers of the observations for which fitted distributions are to be displayed, e.g. `obs=c(23, 58)` will display the fitted distribution for the 23th and 58th observations
family This must be a gamlss family i.e. family=NO
mu The value(s) of the location parameter mu for which the distribution has to be evaluated e.g mu=c(3,7)
sigma The value(s) the scale parameter sigma for which the distribution has to be evaluated e.g sigma=c(3,7)
nu The value(s) the parameter nu for which the distribution has to be evaluated e.g. nu=3
tau The value(s) the parameter tau for which the distribution has been evaluated e.g. tau=5
min Minimum value of the random variable y e.g. min=0
max Maximum value of y e.g. max=10
step Steps for the evaluation of y e.g. step=0.5
allinone This will go
no.title Whether you need title in the plot, default is no.title=FALSE
... for extra arguments

Details
This function can be used to plot distributions of the GAMLSS family. If the first argument obj is specified and it is a GAMLSS fitted object, then the fitted distribution of this model at specified observation values (given by the second argument obs) is plotted for a specified y-variable range (arguments min, max, and step).

If the first argument is not given then the family argument has to be specified and the pdf is plotted at specified values of the parameters mu, sigma, nu, tau. Again the range of the y-variable has to be given.

Value
plot(s) of the required pdf(s) are returned

Warning
The range of some distributions depends on the fitted parameters

Note
The range of the y values given by min, max and step are very important in the plot

Author(s)
Mikis Stasinopoulos <d.stasinopoulos@londonmet.ac.uk> and Calliope Akantziliotou
plot.gamlss

References


See Also

gamlss

Examples

```r
pdf.plot(family=BCT, min=1, max=20, step=.05, mu=10, sigma=0.15, nu=-1, tau=c(4,10,20,40) )
# now using an gamlss object
# library(gamlss)
#data(abdom)
#h=gamlss(y~pb(x), sigma.formula=-pb(x), family=BCT, data=abdom) # fits
#pdf.plot(obj=h , obs=c(23,67), min=50, max=150, step=.5)
```

plot.gamlss  
*Plot Residual Diagnostics for an GAMLSS Object*

Description

This function provides four plots for checking the normalized (randomized for a discrete response distribution) quantile residuals of a fitted GAMLSS object, referred to as residuals below : a plot of residuals against fitted values, a plot of the residuals against an index or a specific explanatory variable, a density plot of the residuals and a normal Q-Q plot of the residuals. If argument ts=TRUE then the first two plots are replaced by the autocorrelation function (ACF) and partial autocorrelation function (PACF) of the residuals

Usage

```r
## S3 method for class 'gamlss'
plot(x, xvar = NULL, parameters = NULL, ts = FALSE,
     summaries = TRUE, ...)
```

Arguments

- `x` a GAMLSS fitted object
- `xvar` an explanatory variable to plot the residuals against
- `parameters` plotting parameters can be specified here
plot.gamlss

ts set this to TRUE if ACF and PACF plots of the residuals are required

summarizes set this to FALSE if no summary statistics of the residuals are required

... further arguments passed to or from other methods.

Details

This function provides four plots for checking the normalized (randomized) quantile residuals (called residuals) of a fitted GAMLSS object. Randomization is only performed for discrete response variables. The four plots are

- residuals against the fitted values (or ACF of the residuals if ts=TRUE)
- residuals against an index or specified x-variable (or PACF of the residuals if ts=TRUE)
- kernel density estimate of the residuals
- QQ-normal plot of the residuals

For time series response variables option ts=TRUE can be used to plot the ACF and PACF functions of the residuals.

Value

Returns four plots related to the residuals of the fitted GAMLSS model and prints summary statistics for the residuals if the summary=T

Author(s)

Mikis Stasinopoulos <d.stasinopoulos@londonmet.ac.uk>, Bob Rigby <r.rigby@londonmet.ac.uk> and Kalliope Akantziliotou

References


See Also

gamlss
plot.histSmo

Examples

data(aids)
a<-gamlss(y~pb(x)+qrt,family=P0,data=aids)
plot(a)
rm(a)

plot.histSmo A Plotting Function for density estimator object histSmo

Description

Plots the estimated density or its c.d.f function or its inverse cdf function

Usage

## S3 method for class 'histSmo'
plot(x, type = c("hist", "cdf", "invcdf"), ...)

Arguments

x An histSmo object

type Different plots: a histogram and density estimator, a cdf function or an inverse

cdf function.

... for further arguments

Value

returns the relevant plot

Author(s)

Mikis Stasinopoulos, Paul Eilers, Bob Rigby, Vlasios Voudouris and Majid Djennad

References

Eilers, P. H. C. and Marx, B. D. (1996). Flexible smoothing with B-splines and penalties (with
0-387-98218-3
Rigby, R. A. and Stasinopoulos D. M. (2005). Generalized additive models for location, scale and
See Also

histSmo

Examples

```r
Y <- rPARETO2(1000)
m1 <- histSmo(Y, lower=0, save=TRUE)
plot(m1)
plot(m1, "cdf")
plot(m1, "invcdf")
```

plot2way

Function to plot two interaction in a GAMLSS model

Description

This function is designed to plot a factor to factor interaction in a GAMLSS model.

Usage

```r
plot2way(obj, terms = list(), what = c("mu", "sigma", "nu", "tau"),
         show.legend = TRUE, ...)
```

Arguments

- `obj` A gamlss model
- `terms` this should be a character vector with the names of the two factors to be plotted
- `what` which parameters? mu, sigma, nu, or tau
- `show.legend` whether to show the legend in the two way plot
- `...` Further arguments

Details

This is an experimental function which should be use with prudence since no other check is done on whether this interaction interfere with other terms in the model

Value

The function creates a 2 way interaction plot

Author(s)

Mikis Stasinopoulos
polyS

References


See Also
term.plot,

Examples

data(aids)
ti <- factor(c(rep(1,18),rep(2,27)))
m1 <- gamlss(y+x+qrt*ti, data=aids, family=NBI)
m2 <- gamlss(y+x+qrt*ti, data=aids, family=NO)
plot2way(m1, c("qrt","ti"))
plot2way(m1, c("ti", "qrt"))

polyS

Auxiliary support for the GAMLSS

Description

These two functions are similar to the poly and polym in R. Are needed for the gamlss.lo function of GAMLSS and should not be used on their own.

Usage

polyS(x, ...)
poly.matrix(m, degree = 1)

Arguments

x a variable
m a variable
degree the degree of the polynomial
... for extra arguments

Value

Returns a matrix of orthogonal polynomials
Warning

Not be use by the user

Author(s)

Mikis Stasinopoulos <d.stasinopoulos@londonmet.ac.uk>

References


See Also

gamlss, gamlss.lo

predict.gamlss

Extract Predictor Values and Standard Errors For New Data In a GAMLSS Model

Description

predict.gamlss is the GAMLSS specific method which produce predictors for a new data set for a specified parameter from a GAMLSS objects. The predict.gamlss can be used to extract the linear predictors, fitted values and specific terms in the model at new data values in the same way that the predict.lm() and predict.glm() functions can be used for lm or glm objects. Note that linear predictors, fitted values and specific terms in the model at the current data values can also be extracted using the function lpred() (which is called from predict if new data is NULL).

Usage

```r
## S3 method for class 'gamlss'
predict(object, what = c("mu", "sigma", "nu", "tau"),
       newdata = NULL, type = c("link", "response", "terms"),
       terms = NULL, se.fit = FALSE, data = NULL, ...)
predictAll(object, newdata = NULL, type = c("response", "link", "terms"),
          terms = NULL, se.fit = FALSE, ...)
```
Arguments

- **object**: a GAMLSS fitted model
- **what**: which distribution parameter is required, default what = "mu"
- **newdata**: a data frame containing new values for the explanatory variables used in the model
- **type**: the default, gets the linear predictor for the specified distribution parameter. type = "response" gets the fitted values for the parameter while type = "terms" gets the fitted terms contribution
- **term**: if type = "terms", which terms to be selected (default is all terms)
- **se.fit**: if TRUE the approximate standard errors of the appropriate type are extracted if exist
- **data**: the data frame used in the original fit if is not defined in the call
- ... for extra arguments

Details

The predict function assumes that the object given in newdata is a data frame containing the right x-variables used in the model. This could possible cause problems if transformed variables are used in the fitting of the original model. For example, let us assume that a transformation of age is needed in the model i.e. nage = age^.5. This could be fitted as mod = gamlss(y ~ cs(age^.5), data = mydata) or as nage = age^.5; mod = gamlss(y ~ cs(nage), data = mydata). The later could more efficient if the data are in thousands rather in hundreds. In the first case, the code predict(mod, newdata = data.frame(age = c(34, 56))) would produce the right results. In the second case a new data frame has to be created containing the old data plus any new transform data. This data frame has to be declared in the data option. The option newdata should contain a data.frame with the new names and the transformed values in which prediction is required, (see the last example).

Value

A vector or a matrix depending on the options.

Note

This function is under development

Author(s)

Mikis Stasinopoulos

References


See Also

lp, lpred

Examples

```r
data(aids)
a <- gamlss(y=poly(x,3)+qrt, family=PO, data=aids) #
newaids <- data.frame(x=c(45,46,47), qrt=c(2,3,4))
ap <- predict(a, newdata=newaids, type = "response")
ap # now getting all the parameters
predictAll(a, newdata=newaids)
rm(a, ap)
data(abdom)
# transform x
aa <- gamlss(y=cs(x+.5), data=abdom)
# predict at old values
predict(aa)[610]
# predict at new values
predict(aa,newdata=data.frame(x=42.43))
# now transform x first
nx <- abdom$x^.5
aaa <- gamlss(y=cs(nx), data=abdom)
# create a new data frame
newd <- data.frame(abdom, nx=abdom$x^0.5)
# predict at old values
predict(aaa)[610]
# predict at new values
predict(aaa,newdata=data.frame(nx=42.43^.5), data=newd)
```

---

**print.gamlss**  
Prints a GAMLSS fitted model

**Description**

`print.gamlss` is the GAMLSS specific method for the generic function `print` which prints objects returned by modelling functions.

**Usage**

```r
## S3 method for class 'gamlss'
print(x, digits = max(3, getOption("digits") - 3), ...)
```
Arguments

x                a GAMLSS fitted model
digits          the number of significant digits to use when printing
...             for extra arguments

Value

Prints a gamlss object

Author(s)

Mikis Stasinopoulos <d.stasinopoulos@londonmet.ac.uk>, Bob Rigby <r.rigby@londonmet.ac.uk> and Calliope Akantziliotou

References


See Also

gamlss, deviance.gamlss, fitted.gamlss

Examples

data(aids)
h<-gamlss(y~poly(x,3)+qrt, family=PO, data=aids)
print(h) # or just h
rm(h)

prof.dev

Plotting the Profile Deviance for one of the Parameters in a GAMLSS model

Description

This functions plots the profile deviance of one of the (four) parameters in a GAMLSS model. It can be used if one of the parameters \mu, \sigma, \nu or \tau is a constant (not a function of explanatory variables) to obtain a profile confidence intervals.
Usage

prof.dev(object, which = NULL, min = NULL, max = NULL, step = NULL, length = 7, startlastfit = TRUE, plot = TRUE, perc = 95, ...)

Arguments

object A fitted GAMLSS model
which which parameter to get the profile deviance e.g. which="tau"
min the minimum value for the parameter e.g. min=1
max the maximum value for the parameter e.g. max=20
step how often to evaluate the global deviance (defines the step length of the grid for the parameter) e.g. step=1
length the length if step is not set, default equal 7
startlastfit whether to start fitting from the last fit or not, default value is startlastfit=TRUE
plot whether to plot, plot=TRUE or save the results, plot=FALSE
perc what % confidence interval is required
... for extra arguments

Details

This function can be use to provide likelihood based confidence intervals for a parameter for which a constant model (i.e. no explanatory model) is fitted and consequently for checking the adequacy of a particular values of the parameter. This can be used to check the adequacy of one distribution (e.g. Box-Cox Cole and Green) nested within another (e.g. Box-Cox power exponential). For example one can test whether a Box-Cox Cole and Green (Box-Cox-normal) distribution or a Box-Cox power exponential is appropriate by plotting the profile of the parameter tau. A profile deviance showing support for tau=2 indicates adequacy of the Box-Cox Cole and Green (i.e. Box-Cox normal) distribution.

Value

Return a profile plot (if the argument plot=TRUE) and an ProfLikelihood.gamlss object if saved. The object contains:

values the values at the grid where the parameter was evaluated
fun the function which approximates the points using splines
min the minimum values in the grid
max the maximum values in the grid
max.value the value of the parameter maximising the Profile deviance (or GAIC)
CI the profile confidence interval (if global deviance is used)
criterion which criterion was used
Warning

A dense grid (i.e. small step) evaluation of the global deviance can take a long time, so start with a sparse grid (i.e. large step) and decrease gradually the step length for more accuracy.

Author(s)

Calliope Akantziliotou, Mikis Stasinopoulos <d.stasinopoulos@londonmet.ac.uk> and Bob Rigby <r.rigby@londonmet.ac.uk>

References


See Also

gamlss, prof.term

Examples

```r
## Not run:
data(abdom)
h <- gamlss(y~pb(x), sigma.formula=~pb(x), family=BCT, data=abdom)
prof.dev(h, "nu", min=-2.000,max=2)
rm(h)
## End(Not run)
```

prof.term

*Plotting the Profile: deviance or information criterion for one of the terms (or hyper-parameters) in a GAMLSS model*

Description

This function plots the profile deviance for a chosen parameter included in the linear predictor of any of the mu,sigma, nu or tau models so profile confidence intervals can be obtained. It can also be used to plot the profile of a specified information criterion for any hyper-parameter when smooth additive terms are used.
Usage

prof.term(model = NULL, criterion = c("GD", "GAIC"), penalty = 2.5,
other = NULL, min = NULL, max = NULL, step = NULL,
length = 7, xlabel = NULL, plot = TRUE, perc = 95,
start.prev = TRUE, ...)

Arguments

model this is a GAMLSS model, e.g.
model=gamlss(y~cs(x,df=3), sigma.fo=~cs(x,df=3), data=abdom), where
this indicates the (hyper)parameter to be profiled

criterion whether global deviance ("GD") or information criterion ("GAIC") is profiled.
The default is global deviance criterion="GD"

penalty The penalty value if information criterion is used in criterion, default penalty=2.5

other this can be used to evaluate an expression before the actual fitting of the model
(Make sure that those expressions are well define in the global environment)

min the minimum value for the parameter e.g. min=1

max the maximum value for the parameter e.g. max=20

step how often to evaluate the global deviance (defines the step length of the grid for the parameter) e.g. step=1

length if the step is left NULL then length is considered for evaluating the grid for the parameter. It has a default value of 11

xlabel if a label for the axis is required

plot whether to plot, plot=TRUE the resulting profile deviance (or GAIC)

perc what % confidence interval is required

start.prev whether to start from the previous fitted model parameters values or not (default is TRUE)

... for extra arguments

Details

This function can be use to provide likelihood based confidence intervals for a parameter involved in terms in the linear predictor(s). These confidence intervals are more accurate than the ones obtained from the parameters’ standard errors. The function can also be used to plot a profile information criterion (with a given penalty) against a hyper-parameter. This can be used to check the uniqueness in hyper-parameter determination using for example find.df.

Value

Return a profile plot (if the argument plot=TRUE) and an ProfLikelihood.gamlss object if saved. The object contains:

values the values at the grid where the parameter was evaluated

fun the function which approximates the points using splines
prof.term

min   the minimum values in the grid
max   the maximum values in the grid
max.value   the value of the parameter maximising the Profile deviance (or GAIC)
CI   the profile confidence interval (if global deviance is used)
criterion which criterion was used

Warning

A dense grid (i.e. small step) evaluation of the global deviance can take a long time, so start with a sparse grid (i.e. large step) and decrease gradually the step length for more accuracy.

Author(s)

Mikis Stasinopoulos <d.stasinopoulos@londonmet.ac.uk> and Bob Rigby <r.rigby@londonmet.ac.uk>

References


See Also

gamlss, prof.dev

Examples

data(aids)
# fitting a linear model
gamlss(y~x+qrt,family=NBI,data=aids)
# testing the linear beta parameter
mod<-'quote(gamlss(y ~ offset(this * x) + qrt, data = aids, family = NBI))
prof.term(mod, min=0.06, max=0.11)
# find the hyper parameter using cubic splines smoothing
mod1<-quote(gamlss(y ~ cs(x,df=this) + qrt, data = aids, family = NBI))
prof.term(mod1, min=1, max=15, step=1, criterion="GAIC", penalty=log(45))
# find a break point in x
mod2 <- 'quote(gamlss(y ~ x+I((x>this)*(x-this)))+qrt,family=NBI,data=aids))
prof.term(mod2, min=1, max=45, step=1, criterion="GD")
rm(mod,mod1,mod2)
Specify a Penalised B-Spline Fits in a GAMLSS Formula

Description

There are several functions operating using penalised B-splines: `pb()`, `pbo()`, `cy()`, `pvc()`, `pbm()` and `ps()`.

The functions take a vector and return it with several attributes. The vector is used in the construction of the design matrix $X$ used in the fitting. The functions do not do the smoothing, but assign the attributes to the vector to aid `gamlss` in the smoothing. The functions doing the smoothing are `gamlss.pb()`, `gamlss.pbo()`, `gamlss.cy()` and `gamlss.pvc()`, `gamlss.pbm()` and `gamlss.ps()` which are used in the backfitting function `additive.fit`.

The function `pb()` is more efficient and faster than the original penalized smoothing function `ps()`. After December 2014 the `pb()` has changed radically to improved performance. The older version of the function `pb()` is called now `pbo()`. `pb()` allows the estimation of the smoothing parameters using different local (performance iterations) methods. The methods are "ML", "ML-1", "EM", "GAIC" and "GCV". The function `cy()` fits a cycle penalised beta regression spline such as the last fitted value of the smoother is equal to the first fitted value. The function `pvc()` fits varying coefficient models see Hastie and Tibshirani (1993) and it is more general and flexible than the old `vc()` function which is based on cubic splines.

Usage

```
pb(x, df = NULL, lambda = NULL, control = pb.control(...), ...) npbo(x, df = NULL, lambda = NULL, control = pbo.control(...), ...) npbo.control(inter = 20, degree = 3, order = 2, start = 10, quantiles = FALSE, method = c("ML", "GAIC", "GCV", "EM", "ML-1"), k = 2, ...) npb.control(inter = 20, degree = 3, order = 2, start = 10, quantiles = FALSE, method = c("ML", "GAIC", "GCV"), k = 2, ...)ncy(x, df = NULL, lambda = NULL, control = cy.control(...), ...) cy.control(inter = 20, degree = 3, order = 2, start = 10, quantiles = FALSE, method = c("ML", "GAIC", "GCV", "EM", "ML-1"), k = 2, ts=FALSE, ...) npvc(x, df = NULL, lambda = NULL, by = NULL, control = pvc.control(...), ...) npvc.control(inter = 20, degree = 3, order = 2, start = 10, quantiles = FALSE, method = c("ML", "GAIC", "GCV", "EM", "ML-1"), k = 2, ...) npm(x, df = NULL, lambda = NULL, mono="up", "down"), control = pbm.control(...), ...) npbm.control(inter = 20, degree = 3, order = 2, start = 10, quantiles = FALSE, method = c("ML", "GAIC"), k = 2, ...) nps(x, df = 3, lambda = NULL, ps.intervals = 20, degree = 3, order = 3)
```

Arguments

- **x**
  - the univariate predictor

- **df**
  - the desired equivalent number of degrees of freedom (trace of the smoother matrix minus two for the constant and linear fit)
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>lambda</td>
<td>the smoothing parameter</td>
</tr>
<tr>
<td>control</td>
<td>setting the control parameters</td>
</tr>
<tr>
<td>by</td>
<td>a factor, for fitting different smoothing curves to each level of the factor or a continuous explanatory variable in which case the coefficients of the by variable change smoothly according to ( x ) i.e. ( \beta(x)z ) where ( z ) is the by variable.</td>
</tr>
<tr>
<td>...</td>
<td>for extra arguments</td>
</tr>
<tr>
<td>inter</td>
<td>the no of break points (knots) in the x-axis</td>
</tr>
<tr>
<td>degree</td>
<td>the degree of the piecewise polynomial</td>
</tr>
<tr>
<td>order</td>
<td>the required difference in the vector of coefficients</td>
</tr>
<tr>
<td>start</td>
<td>the lambda starting value if the local methods are used, see below</td>
</tr>
<tr>
<td>quantiles</td>
<td>if TRUE the quantile values of ( x ) are use to determine the knots</td>
</tr>
<tr>
<td>ts</td>
<td>if TRUE assumes that it is a seasonal factor</td>
</tr>
<tr>
<td>method</td>
<td>The method used in the (local) performance iterations. Available methods are &quot;ML&quot;, &quot;ML-1&quot;, &quot;EM&quot;, &quot;GAIC&quot; and &quot;GCV&quot;</td>
</tr>
<tr>
<td>k</td>
<td>the penalty used in &quot;GAIC&quot; and &quot;GCV&quot;</td>
</tr>
<tr>
<td>mono</td>
<td>for monotonic P-splines whether going &quot;up&quot; or &quot;down&quot;</td>
</tr>
<tr>
<td>ps.intervals</td>
<td>the no of break points in the x-axis</td>
</tr>
</tbody>
</table>

### Details

The `ps()` function is based on Brian Marx function which can be found in [http://www.stat.lsu.edu/faculty/marx/](http://www.stat.lsu.edu/faculty/marx/). The `pb()`, `cy()`, `pvc()` and `pbm()` functions are based on Paul Eilers’s original R functions. Note that `ps()` and `pb()` functions behave differently at their default values if `df` and `lambda` are not specified. `ps(x)` by default uses 3 extra degrees of freedom for smoothing \( x \). `pb(x)` by default estimates lambda (and therefore the degrees of freedom) automatically using a "local" method. The local (or performance iterations) methods available are: (i) local Maximum Likelihood, "ML", (ii) local Generalized Akaike information criterion, "GAIC", (iii) local Generalized Cross validation "GCV" (iv) local EM-algorithm, "EM" (which is very slow) and (v) a modified version of the ML, "ML-1" which produce identical results with "EM" but faster.

The function `pb()` fits a P-spline smoother.

The function `pbm()` fits a monotonic (going up) P-spline smoother.

The function `cy()` fits a P-spline smoother where the beginning and end are the same.

The `pvc()` fits a varying coefficient model.

Note that the local (or performance iterations) methods can occasionally make the convergence of `gamlss` less stable compared to models where the degrees of freedom are fixed.

### Value

The vector `x` is returned, endowed with a number of attributes. The vector itself is used in the construction of the model matrix, while the attributes are needed for the backfitting algorithms `additive.fit()`.
Warning

There are occasions where the automatic local methods do not work. One accusation which came to our attention is when the range of the response variable values is very large. Calling the response variable will solve the problem.

Author(s)

Mikis Stasinopoulos <d.stasinopoulos@londonmet.ac.uk>, Bob Rigby <r.rigby@londonmet.ac.uk> and Paul Eilers

References

http://www.stat.lsu.edu/faculty/marx/


See Also
gamlss, gamlss.ps.cs

Examples

#============================
# pb() and ps() functions
data(aids)
# fitting a smoothing cubic spline with 7 degrees of freedom
# plus the a quarterly effect
aids1<-gamlss(y~ps(x,df=7)+qrt,data=aids,family=PO) # fix df's
aids2<-gamlss(y~pb(x,df=7)+qrt,data=aids,family=PO) # fix df's
aids3<-gamlss(y~pb(x)+qrt,data=aids,family=PO) # estimate lambda
with(aids, plot(x,y))
with(aids, lines(x,fitted(aids1),col="red"))
with(aids, lines(x,fitted(aids2),col="green"))
with(aids, lines(x,fitted(aids3),col="yellow"))
rm(aids1, aids2, aids3)
#============================
## Not run:
# cy()
# simulate data
set.seed(5555)
x = seq(0, 1, length = 100)
y = sign(cos(1 * x * 2 * pi + pi / 4)) + rnorm(length(x)) * 0.2
```r
plot(y~x)
m1<-gamlss(y~cy(x))
lines(fitted(m1)-x)
rm(y,x,m1)

# the pvc() function
# function to generate data
genData <- function(n=200)
{
  f1 <- function(x)-60+15*x-0.10*x^2
  f2 <- function(x)-120+10*x+0.08*x^2
  set.seed(1441)
  x1 <- runif(n/2, min=0, max=55)
  x2 <- runif(n/2, min=0, max=55)
  y1 <- f1(x1)+rNO(n=n/2, mu=0, sigma=20)
  y2 <- f2(x2)+rNO(n=n/2, mu=0, sigma=30)
  y <- c(y1,y2)
  x <- c(x1,x2)
  f <- gl(2,n/2)
  da<-data.frame(y,x,f)
da }
da<-genData(500)
plot(y~x, data=da, pch=21, bg=c("gray","yellow3")[unclass(f)])
# fitting models
# smoothing x
m1 <- gamlss(y~pb(x), data=da)
# parallel smoothing lines
m2 <- gamlss(y~pb(x)+f, data=da)
# linear interaction
m3 <- gamlss(y~pb(x)+f*x, data=da)
# varying coefficient model
m4 <- gamlss(y~pvc(x, by=f), data=da)
GAIC(m1,m2,m3,m4)
# plotting the fit
lines(fitted(m4)[da$f==1][order(da$x[da$f==1])]-da$x[da$f==1]
  ,order(da$x[da$f==1]), col="blue", lwd=2)
lines(fitted(m4)[da$f==2][order(da$x[da$f==2])]-da$x[da$f==2]
  ,order(da$x[da$f==2]), col="red", lwd=2)
rm(da,m1,m2,m3,m4)

# the rent data
# first with a factor
data(rent)
data(rent)
plot(R~F1, data=rent, pch=21, bg=c("gray","blue")[unclass(rent$B)])
r1 <- gamlss(R~pb(F1), data=rent)
# identical to model
r11 <- gamlss(R~pvc(F1), data=rent)
# now with the factor
r2 <- gamlss(R~pvc(F1, by=B), data=rent)
lines(fitted(r2)[rent$B==1][order(rent$F1[rent$B==1])]-rent$F1[rent$B==1]
  ,order(rent$F1[rent$B==1]), col="blue", lwd=2)
lines(fitted(r2)[rent$B==0][order(rent$F1[rent$B==0])]-rent$F1[rent$B==0]
```

# probably not very sensible model
rm(r1,r11,r2)

# now with a continuous variable
# additive model
h1 <- gamlss(R~pb(F1)+pb(A), data=rent)

# varying-coefficient model
h2 <- gamlss(R~pb(F1)+pb(A)+pvc(A,by=F1), data=rent)
AIC(h1,h2)
rm(h1,h2)

# monotone function
set.seed(1334)
x <- seq(0, 1, length = 100)
p <- 0.4
y <- sin(2 * pi * p * x) + rnorm(100) * 0.1
plot(y~x)
m1 <- gamlss(y~pbm(x))
points(fitted(m1)-x, col="red")

yy <- -y
plot(yy~x)
m2 <- gamlss(yy~pbm(x, mono="down"))
points(fitted(m2)-x, col="red")

## End(Not run)

# A function to calculate the Q-statistics

Q.stats

Description

This function calculates and prints the Q-statistics (or Z-statistics) which are useful to test normality of the residuals within a range of an independent variable, for example age in centile estimation, see Royston and Wright (2000).

Usage

Q.stats(obj = NULL, xvar = NULL, resid = NULL, xcut.points = NULL, n.inter = 10, zvals = TRUE, save = TRUE, plot = TRUE, digits.xvar = getOption("digits"), ...)

Arguments

obj : a GAMLSS object
xvar : a unique explanatory variable
resid : quantile or standardised residuals can be given here instead of a GAMLSS object in obj. In this case the function behaves differently (see details below)
\texttt{Q.stats} 91

\texttt{xcut.points} the x-axis cut off points e.g. \((20, 30)\). If \texttt{xcut.points=NULL} then the \texttt{n.inter} argument is activated

\texttt{n.inter} if \texttt{xcut.points=NULL} this argument gives the number of intervals in which the x-variable will be split, with default 10

\texttt{zvals} if \texttt{TRUE} the output matrix contains the individual Z-statistics rather that the Q statistics

\texttt{save} whether to save the Q-statistics or not with default equal to \texttt{TRUE}. In this case the functions produce a matrix giving individual Q (or z) statistics and the final aggregate Q’s

\texttt{plot} whether to plot a visual version of the Q statistics (default is \texttt{TRUE})

\texttt{digits.xvar} to control the number of digits of the \texttt{xvar} in the plot

\texttt{...} for extra arguments

**Details**

Note that the function \texttt{Q.stats} behaves differently depending whether the \texttt{obj} or the \texttt{resid} argument is set. The \texttt{obj} argument produces the Q-statistics (or Z-statistics) table appropriate for centile estimation (therefore it expect a reasonable large number of observations). The argument \texttt{resid} allows any model residuals, (not necessary GAMLSS), suitable standardised and is appropriate for any size of data. The resulting table contains only the individuals Z-statistics.

**Value**

A table containing the Q-statistics or Z-statistics. If \texttt{plot=TRUE} it produces also an graphical representation of the table.

**Author(s)**

Mikis Stasinopoulos <d.stasinopoulos@londonmet.ac.uk>, Bob Rigby <r.rigby@londonmet.ac.uk>, with contributions from Elaine Borghie

**References**


**See Also**

\texttt{gamlss}, \texttt{centiles.split}, \texttt{wp}
Examples

data(abdom)
h<-gamlss(y~pb(x), sigma.formula=-pb(x), family=BCT, data=abdom)
Q.stats(h,xvar=abdom$x,n.inter=8)
Q.stats(h,xvar=abdom$x,n.inter=8,zvals=FALSE)
Q.stats(resid=resid(h), xvar=abdom$x, n.inter=5)
rm(h)

quantSheets(Quan
tile Sheets)

Description

The quantile sheets function quantSheets() is based on the work of Sabine Schnabe and Paul Eiler (see references below). The estimation of the quantile curves is done simultaneously by also smoothing in the direction of y as well as x. This avoids (but do not eliminate completely) the problem of crossing quantiles.

Usage

quantSheets(y, x, x.lambda = 1, p.lambda = 1, data = NULL, 
            cent = 100 * pnorm((-4:4) * 2/3),
            control = quantSheets.control(...), print = TRUE, ...)
quantSheets.control(x.inter = 10, p.inter = 10, degree = 3, logit = FALSE,
                    order = 2, kappa = 0, n.cyc = 100, c.crit = 1e-05, plot = TRUE,
                    power = NULL, ...)
findPower(y, x, data = NULL, lim.trans = c(0, 1.5), prof = FALSE,
          k = 2, c.crit = 0.01, step = 0.1)

z.scoresQS(object, y, x, plot = FALSE, tol = NULL)

Arguments

y the y variable
x the x variable
x.lambda smoothing parameter in the direction of x
p.lambda smoothing parameter in the direction of y (probabilities)
data the data frame
cent the centile values where the quantile sheets is evaluated
control for the parameters controlling the algorithm
print whether to print the sample percentages
x.inter number of intervals in the x direction for the B-splines
quantSheets 93

- `p.inter`: number of intervals in the probabilities (y-direction) for the B-splines
- `degree`: the degree for the B-splines
- `logit`: whether to use `logit(p)` instead of `p` (probabilities) for the y-axis
- `order`: the order of the penalty
- `kappa`: is a ridge parameter set to zero (for no ridge effect)
- `n.cyc`: number of cycles of the algorithm
- `c.crit`: convergence criterion of the algorithm
- `plot`: whether to plot the resulting quantile sheets
- `power`: The value of the power transformation in the x axis if needed
- `lim.trans`: the limits for looking for the power transformation parameter using `findPower()`
- `prof`: whether to use the profile GAIC or `optim()` to the parameter the power transformation
- `k`: the GAIC penalty
- `step`: the steps for the profile GAIC if the argument `prof` of `findPower()` is TRUE
- `object`: a fitted quantSheets object
- `tol`: how far out from the range of the y variable should go for estimating the distribution of y using the `flexDist()` function

... for further arguments

**Details**

The advantage of quantile sheets is that they estimates simultaneously all the quantiles. This almost eliminates the problem of crossing quantiles. The method is very fast and useful for exploratory tool. The function needs two smoothing parameters. Those two parameters have to specified by the user. They are not estimated automatically. They can be selected by visual inspection.

The disadvantages of quantile sheets comes from the fact that like all non-parametric techniques do not have a goodness of fit measure to change how good is the models and the residuals based diagnostics are not existence since it is difficult to define residuals in this set up.

In this implementation we do provide residuals by using the `flexDist()` function from package `gamlss.dist`. This is based on the idea that by knowing the quantiles of the distribution we can reconstruct non parametrically the distribution itself and this is what `flexDist()` is doing. As a word of caution, such a construct is based on several assumptions and depends on several smoothing parameters. Treat those residuals with caution. The same caution should apply to the function `z.scoresQS()`.

**Value**

Using the function `quantSheets()` a quantSheets object is returned having the following methods: `print()`, `fitted()`, `predict()` and `resid()`.

Using `findPower()` a single values of the power parameter is returned.

Using `z.scoresQS` a vector of z-scores is returned.
random

Specify random effects in a GAMLSS formula

Description

They are two functions for fitting random effects within a GAMLSS model, `random()` and `re()`. The function `random()` is based on the original `random()` function of Trevor Hastie in the package `gam`. In our version the function has been modified to allow a "local" maximum likelihood estimation of the smoothing parameter `lambda`. This method is equivalent to the PQL method of Breslow and Clayton (1993) applied at the local iterations of the algorithm. In fact for a GLM model and a simple random effect it is equivalent to `glmmPQL()` function in the package `MASS` see Venables and Ripley (2002). Venables and Ripley (2002) claimed that this iterative method was first introduced by Schall (1991). Note that in order for the "local" maximum likelihood estimation procedure to operate both argument `df` and `lambda` has to be `NULL`.

The function `re()` is an interface for calling the `lme()` function of the package `nlme`. This gives the user the ability to fit complicated random effect models while the assumption of the normal distribution for the response variable is relaxed. The theoretical justification comes again from the fact that this is a PQL method, Breslow and Clayton (1993).
random

Usage
random(x, df = NULL, lambda = NULL, start=10)
re(formula = fixed = ~1, random = NULL, correlation = NULL, method = "ML", ...)

Arguments
- x: a factor
- df: the target degrees of freedom
- lambda: the smoothing parameter lambda which can be viewed as a shrinkage parameter.
- start: starting value for lambda if local Maximum likelihood is used.
- fixed: a formula specifying the fixed effects of the lme() model. This, in most cases can be also included in the gamlss parameter formula
- random: a formula or list specifying the random effect part of the model as in lme() function
- correlation: the correlation structure of the lme() model
- method: which method, "ML" (the default), or "REML"
- ... this can be used to pass arguments for lmeControl()

Details
The function random() can be seen as a smoother for use with factors in gamlss(). It allows the fitted values for a factor predictor to be shrunk towards the overall mean, where the amount of shrinking depends either on lambda, or on the equivalent degrees of freedom or on the estimated sigma parameter (default). Similar in spirit to smoothing splines, this fitting method can be justified on Bayesian grounds or by a random effects model. Note that the behavior of the function is different from the original Hastie function. Here the function behaves as follows: i) if both df and lambda are NULL then the PQL method is used ii) if lambda is not NULL, lambda is used for fitting iii) if lambda is NULL and df is not NULL then df is used for fitting.

Since factors are coded by model.matrix() into a set of contrasts, care has been taken to add an appropriate "contrast" attribute to the output of random(). This zero contrast results in a column of zeros in the model matrix, which is aliased with any column and is hence ignored.

The use of the function re() requires knowledge of the use of the function lme() of the package nlme for the specification of the appropriate random effect model. Some care should be taken whether the data set is

Value
x is returned with class "smooth", with an attribute named "call" which is to be evaluated in the backfitting additive.fit() called by gamlss()

Author(s)
For re() Mikis Stasinopoulos and Marco Enea and for random() Trevor Hastie (amended by Mikis Stasinopoulos),
References


See Also

gamlss, gamlss.random

Examples

```r
#------------ Example 1 from Pinheiro and Bates (2000) page 15---------------------
# bring nlme
library(nlme)
data(ergoStool)
# lme model
l1<-lme(effort~Type, data=ergoStool, random=~1|Subject, method="ML")
# use random()
rl1<-gamlss(effort~Type+random(Subject), data=ergoStool )
# use re() with fixed effect within re()
rl2=gamlss(effort~re(fixed=~Type, random=~1|Subject), data=ergoStool )
# use re() with fixed effect in gamlss formula
rl3=gamlss(effort~Type+re(random=~1|Subject), data=ergoStool )
# compare lme fitted values with random
plot(fitted(l1), fitted(rl1))
# compare lme fitted values with random
plot(fitted(l1), fitted(rl2))
lines(fitted(l1), fitted(rl3), col=2)
# getting the fitted coefficients
getSmo(rl2)
#----------------------------------------------------------------------------------
## Not run:
#------------Example 2 Hodges data-----------------------------------------------------
data(hodges)
plot(prind~state, data=hodges)
m1<- gamlss(prind~random(state), sigma.fo=~random(state), nu.fo=~random(state),
```
random

    tau.fo=random(state), family=BCT, data=hodges)
m2<- gamlss(prind=re(random=-1|state), sigma.fo=re(random=-1|state),
        nu.fo=re(random=-1|state), tau.fo=re(random=-1|state), family=BCT,
        data=hodges)

    # comparing the fitted effective degrees of freedom
m1$mu.df
m2$mu.df
m1$sigma.df
m2$sigma.df
m1$nu.df
m2$nu.df
m1$tau.df
m2$tau.df

    # random effect for tau is not needed
m3<- gamlss(prind=random(state), sigma.fo=random(state), nu.fo=random(state),
        family=BCT, data=hodges, start.from=m1)
plot(m3)

    # term plots work for random but not at the moment for re()
op <- par(mfrow=c(2,2))
term.plot(m3, se=TRUE)
term.plot(m3, se=TRUE, what="sigma")
term.plot(m3, se=TRUE, what="nu")
par(op)

    # getting information from a fitted lme object
coef(getSmo(m2))
ranef(getSmo(m2))
VarCorr(getSmo(m2))
summary(getSmo(m2))
intervals(getSmo(m2))
fitted(getSmo(m2))

    # plotting
plot(getSmo(m2))
qqnorm(getSmo(m2))

--------------------------Example 3 from Pinheiro and Bates (2000) page 42--------------------------
data(Pixel)
l1 <- lme(pixel~ day+I(day^2), data=Pixel, random=list(Dog="day", Side="-1"),
    method="ML")

    # this will fail
#t1<-gamlss(pixel~re(fixed=-day+I(day^2), random=list(Dog="-day", Side="-1")),
#    data=Pixel)
    # but this is working
t1<-gamlss(pixel~re(fixed=-day+I(day^2), random=list(Dog="-day", Side="-1"),
        opt="optim"), data=Pixel)
plot(fitted(l1)-fitted(t2))

--------------------------Example 4 from Pinheiro and Bates (2000) page 146--------------------------
data(Orthodont)
l1 <- lme(distance~ I(age-11), data=Orthodont, random=~I(age-11)|Subject,
    method="ML")

t1<-gamlss(distance~I(age-11)+re(random=~I(age-11)|Subject), data=Orthodont)
plot(fitted(l1)-fitted(t1))

    # checking the model
Refit a GAMLSS model

```r
plot(t1)
wp(t1, ylim.all=2)
# two observation fat try LO
t2<-gamlss(distance~I(age-11)+re(random=-1|Subject, opt="optim", numIter=100), data=Orthodont, family=LO)
plot(t2)
wp(t2, ylim.all=2)
# a bit better but not satisfactory Note that 3 parameters distributions fail
library(MASS)
data(bacteria)
summary(glmPQL(y~trt + I(week > 2), random = ~ 1 | ID, family = binomial, data = bacteria))
s1 <- gamlss(y~trt + I(week > 2)+random(ID), family = BI, data = bacteria)
s2 <- gamlss(y~trt + I(week > 2)+re(random=-1|ID), family = BI, data = bacteria)
s3 <- gamlss(y~trt + I(week > 2)+re(random=-1|ID, method="REML"), family = BI, data = bacteria)
# the estimate of the random effect sd sigma_b
sqrt(getSmo(s1)$tau2)
getSmo(s2)
getSmo(s3)
library(lme4)
data(Ovary)
# AR1
l1 <- lme(folllices~sin(2*pi*Time)+cos(2*pi*Time), data=Ovary, random=pdDiag(~sin(2*pi*Time)), correlation=corAR1())
# ARMA
l2 <- lme(folllices~sin(2*pi*Time)+cos(2*pi*Time), data=Ovary, random=pdDiag(~sin(2*pi*Time)), correlation=corARMA(q=2))
# now gamlss
# AR1
t1 <- gamlss(folllices~re(fixed=-sin(2*pi*Time)+cos(2*pi*Time), random=pdDiag(~sin(2*pi*Time)), correlation=corAR1()), data=Ovary)
plot(fitted(l1)-fitted(t1))
# ARMA
t2 <- gamlss(folllices~re(fixed=-sin(2*pi*Time)+cos(2*pi*Time), random=pdDiag(~sin(2*pi*Time)), correlation=corARMA(q=2)), data=Ovary)
plot(fitted(l2)-fitted(t2))
AIC(t1, t2)
wp(t2, ylim.all=1)
#-----------------------------
## End(Not run)
```
Description

This function refits a GAMLSS model. It is useful when the algorithm has not converged after 20 outer iteration (the default value)

Usage

refit(object, ...)

Arguments

- `object`: a GAMLSS fitted model which has not converged
- `...`: for extra arguments

Details

This function is useful when the iterations have reach the maximum value set by the code(n.cyc) of the `gamlss.control` function and the model has not converged yet

Value

Returns a GAMLSS fitted model

Note

The function update does a very similar job

Author(s)

Mikis Stasinopoulos <d.stasinopoulos@londonmet.ac.uk>, Bob Rigby <r.rigby@londonmet.ac.uk>

References


See Also

gamlss, update.gamlss
**Examples**

data(aids)
h<-gamlss(y=poly(x,3)+qrt, family=PO, data=aids) #
refit(h)
rm(h)

**Description**

residuals.gamlss is the GAMLSS specific method for the generic function residuals which extracts the residuals for a fitted model. The abbreviated form resid is an alias for residuals.

**Usage**

```r
## S3 method for class 'gamlss'
residuals(object, what = c("z-scores", "mu", "sigma", "nu", "tau"),
          type = c("simple", "weighted", "partial"),
          terms=NULL, ...)
```

**Arguments**

- `object` a GAMLSS fitted model
- `what` specify whether the standardized residuals are required, called here the "z-scores", or residuals for a specific parameter
- `type` the type of residual if residuals for a parameter are required
- `terms` if type is "partial" this specifies which term is required
- `...` for extra arguments

**Details**

The "z-scores" residuals saved in a GAMLSS object are the normalized (randomized) quantile residuals (see Dunn and Smyth, 1996). Randomization is only needed for the discrete family distributions, see also `rqres.plot`. Residuals for a specific parameter can be "simple" = (working variable - linear predictor), "weighted" = sqrt(working weights)*(working variable - linear predictor) or "partial" = (working variable - linear predictor)+contribution of specific terms.

**Value**

a vector or a matrix of the appropriate residuals of a GAMLSS model. Note that when weights are used in the fitting the length of the residuals can be different from N the length of the fitted values. Observations with weights equal to zero are not appearing in the residuals. Also observations with frequencies as weights will appear more than once according to their frequencies.
Note
The "weighted" residuals of a specified parameter can be zero and one if the square of first derivative have been used in the fitting of this parameter.

Author(s)
Mikis Stasinopoulos <d.stasinopoulos@londonmet.ac.uk>, Bob Rigby <r.rigby@londonmet.ac.uk>

References

See Also
print.gamlss, summary.gamlss, fitted.gamlss, coef.gamlss, residuals.gamlss, update.gamlss, plot.gamlss, deviance.gamlss, formula.gamlss

Examples
```r
data(aids)
h <- gamlss(y~poly(x,3)+qrt, family=NB1, data=aids) #
plot(aids$x,resid(h))
plot(aids$x,resid(h,"sigma"))
rm(h)
```

Specifying Ridge Regression in a GAMMSS Formula

Description
The function `ridge()` allows the user to fit a ridge regression within GAMLSS. It allows the coefficients of a set of explanatory variables to be shrunk towards an overall zero, where the amount of shrinking depends either on lambda, or on the equivalent degrees of freedom (df). The function `ridge()` can estimate lambda using a local REML-algorithm.

Usage
```r
ridge(X, df = NULL, lambda = NULL, order = 0)
ridge(X, df = NULL, lambda = NULL, order = 0, start = 10)
```
Arguments

- **X**: A standardized (mean=0, sd=1) matrix of explanatory variables
- **df**: the effective degrees of freedom df
- **lambda**: the smoothing parameter lambda
- **order**: the order of the difference applied to the coefficients
- **start**: the lambda starting value if the Schall’s EM-algorithm is used

Value

- x is returned with class "smooth", with an attribute named "call" which is to be evaluated in the backfitting additive.fit() called by gamlss()

Author(s)

Mikis Stasinopoulos <-d.stasinopoulos@londonmet.ac.uk>, Bob Rigby <-r.rigby@londonmet.ac.uk> and Paul Eilers

References


See Also

gamlss

Examples

data(usair)
# standarized the x's
# for (i in 1:7) usair[,i] <- (usair[,i]-mean(usair[,i]))/sd(usair[,i])
# usair$X <- as.matrix(usair[,c("x1","x2", "x3","x4", "x5", "x6")])
X <- with(usair, scale(cbind(x1,x2, x3,x4, x5, x6)))
# create a matrix in the data
usair$X <- X
ml<- gamlss(y~ridge(X, df=3), data=usair, family=GA)
m2<- gamlss(y~ridge(X, lambda=10), data=usair, family=GA)
# plotting the coefficients as function of the df
df <- seq(0,6,0.5)
resmat<-matrix(0, nrow=length(df), ncol=6)
for (i in 1:length(df))
{
    resmat[i,] <- gamlss(y~ridge(X, df=df[i]), data=usair)$mu.coefSmo[[1]][["coef"]]
}

colnames(resmat)<-colnames(usair$X)
plot(1:length(df), type="n", xlim=c(-.5,6.5), ylim=c(-23, 38))
for (i in 1:6)
{
    lines(resmat[i,]~df, col="purple")
    points(resmat[i,]~df)
}
lines(rep(0,length(df))~df, col="black")
# now estimating the lambda
gl<-gamlss(y~r1(X), data=usair)
ml$mu.coefSmo

---

rqres.plot

Plotting Randomized Quantile Residuals

Description

This function plots worm plots, van Buuren and Fredriks M. (2001), or QQ-plots of the normalized randomized quantile residuals (Dunn and Smyth, 1996) for a model using a discrete GAMLSS family distribution.

Usage

rqres.plot(obj = NULL, howmany = 6, plot = c("all", "average"),
            type = c("wp", "QQ"), ...)

Arguments

obj  a fitted GAMLSS model object from a "discrete" type of family
howmany  The number of QQ-plots required up to ten i.e. howmany=6
plot  whether to plot all plots all the residual realisations "all" or just the mean "average"
type  whether to plot worm plots "wp" or QQ plots "QQ" with default worm plots
...  for extra arguments

Details

For discrete family distributions, the `gamlss()` function saves on exit one realization of randomized quantile residuals which can be plotted using the generic function `plot` which calls the `plot.gamlss`. Looking at only one realization can be misleading, so the current function creates QQ-plots for several realizations. The function allows up to 10 QQ-plots to be plotted. Occasionally one wishes to create a lot of realizations and then take a median of them (separately for each ordered value) to create a single median realization. The option all in combinations with the option howmany creates a QQ-plot of the medians of the normalized randomized quantile residuals. These 'median' randomized quantile residuals can be saved using the option (save=TRUE).
Value

If save it is TRUE then the vector of the median residuals is saved.

Author(s)

Mikis Stasinopoulos <d.stasinopoulos@londonmet.ac.uk>

References


See Also

plot.gamlss, gamlss

Examples

data(aids) # fitting a model from a discrete distribution
h<-gamlss(y~pb(x)+qrt, family=NBI, data=aids) #
plot(h)
# plot qq- plots from 6 realization of the randomized quantile residuals
rqres.plot(h)
# a qq-plot from the medians from 40 realizations
rqres.plot(h,howmany=40,plot="average") #

Rsq Generalised (Pseudo) R-squared for GAMLSS models

Description

This function gives the generalised R-squared of Nagelkerke (1991) for a GAMLSS model.

Usage

Rsq(object, type = c("Cox Snell","Cragg Uhler","both"))
Arguments

- **object**: a GAMLSS object
- **type**: which definition of R squared. Can be the "Cox Snell" or the Nagelkerke, "Cragg Uhler" or "both".

Details

The Rsq() function uses the definition for R-squared:

\[ R^2 = 1 - \left( \frac{L(0)}{L(\hat{\theta})} \right)^{2/n} \]

where \( L(0) \) is the null model (only a constant is fitted to all parameters) and \( L(\hat{\theta}) \) is the current fitted model. This definition sometimes is referred to as the Cox & Snell R-squared. The Nagelkerke /Cragg & Uhler’s definition divides the above with

\[ 1 - L(0)^{2/n} \]

Value

The Rsq() produces a single value if type="Cox Snell" or "Cragg Uhler" and a list if type="both".

Note

The null model is fitted using the function gamlssML() which can create warning messages.

Author(s)

Mikis Stasinopoulos <d.stasinopoulos@londonmet.ac.uk>

References


See Also

- GAIC

Examples

data(aids)
m1 <- gamlss(y=x+qrt, data=aids, family=NBI)Rsq(m1)Rsq(m1, type="both")rm(m1)
rvcov

Robust Variance-Covariance matrix of the parameters from a fitted GAMLSS model

Description

The function rvcov() is designed for providing robust standard errors for the parameters estimates of a GAMLSS fitted model. The same result can be achieved by using vcov(fitted_model, robust=TRUE). The function get.() gets the $K$ matrix (see details below).

Usage

rvcov(object, type = c("vcov", "cor", "se", "coef", "all"),
      hessian.fun = c("R", "PB") )
get.K(object, what = c("K", "Deriv"))

Arguments

- object: a GAMLSS fitted object
- type: this argument for rvcov() function whether variance-covariance matrix, correlation matrix, standard errors or all of them
- what: this an argument for the function get.K() allowing to get either $K$ or the first derivative of the likelihood with respect to the parameters (the $\beta$'s in the GAMLSS notation).
- hessian.fun: How to obtain numerically the Hessian i) using optimHess(), option "R" ii) using a function by Pinheiro and Bates taken from package nlme, option "PB".

Details

The robust standard errors are calculated for the robust sandwich estimator of the variance-covariance given by $S = VKV$ where $V$ is the standard variance-covariance matrix (the inverse of the information matrix) and $K$ is an estimate of the variance of the first derivatives of the likelihood. The function get.K() is used to get the required $K$ matrix.

Value

A variance covariance matrix or other relevant output

Author(s)

Mikis Stasinopoulos <d.stasinopoulos@londonmet.ac.uk>, Bob Rigby <r.rigby@londonmet.ac.uk> and Vlasios Voudouris
References


See Also

vcov, ~~~

Examples

```r
# generate from a gamma distribution
Y <- rGA(200, mu=1, sigma=2)
hist(Y)
# fitting the wrong model i.e. sigma=1
m1 <- gamlss(Y~1, family=EXP)
# the conventional se is too precise
vcov(m1, type="se")
# the sandwich se is wider
rvcov(m1, type="se")
# fitting the correct model
m2 <- gamlss(Y~1, family=GA)
vcov(m2, type="se")
rvcov(m2, type="se")
# similar standard errors
# also obtained using
vcov(m2, type="se", robust=TRUE)
```

stepGAIC

Choose a model by GAIC in a Stepwise Algorithm

Description

The function `stepGAIC()` performs stepwise model selection using a Generalized Akaike Information Criterion. The function `stepGAIC()` calls one of the two functions `stepGAIC.VR()` or `stepGAIC.CH()` depending on the argument `additive`. The function `stepGAIC.VR()` is based on the function `stepAIC()` given in the library MASS of Venables and Ripley (2002). The function `stepGAIC.CH` is based on the S function `step.gam()` (see Chambers and Hastie (1991)) and it is more suited for model with smoothing additive terms, (see below comments for the additive function `pb()`). Both functions have been adapted to work with gamlss objects. The main difference
for the user is the scope argument, see below. If the `stepGAIC()` is called with the argument `additive=FALSE` then the `stepGAIC.VR()` is called else the `stepGAIC.CH()`.

While the functions `stepGAIC.VR()` and `stepGAIC.CH()` are used to build models for individual parameters of the distribution of the response variable, the functions `stepGAICA11.A()` and `stepGAICA11.B()` are building a model for all the parameters. Both the functions `stepGAICA11.A()` and `stepGAICA11.B()` are based on `stepGAIC.VR()`. The use two different strategies for selecting a appropriate final model. `stepGAICA11.A()` has the following strategy:

**Strategy A:**

i) build a model for $\mu$ using a forward approach.

ii) given the model for $\mu$ build a model for $\sigma$ (forward)

iii) given the models for $\mu$ and $\sigma$ build a model for $\nu$ (forward)

iv) given the models for $\mu$, $\sigma$, and $\nu$ build a model for $\tau$ (forward)

v) given the models for $\mu$, $\sigma$, $\nu$, and $\tau$ check whether the terms for $\nu$ are needed using backward elimination.

vi) given the models for $\mu$, $\sigma$, $\nu$, and $\tau$ check whether the terms for $\sigma$ are needed (backward).

vii) given the models for $\mu$, $\sigma$, $\nu$, and $\tau$ check whether the terms for $\mu$ are needed (backward).

Note for this strategy to work the scope argument should be set appropriately.

`stepGAICA11.B()` uses the same procedure as the function `stepGAIC.VR()` but each term in the scope is fitted to ALL the parameters of the distribution, rather than the one specified by the argument `what` of `stepGAIC.VR()`.

**Usage**

```r
stepGAIC.VR(object, scope, direction = c("both", "backward", "forward"),
    trace = T, keep = NULL, steps = 1000, scale = 0,
    what = c("mu", "sigma", "nu", "tau"), k = 2, ...)

stepGAIC.CH(object, scope = gamlss.scope(model.frame(object)),
    direction = c("both", "backward", "forward"), trace = T, keep = NULL,
    steps = 1000, what = c("mu", "sigma", "nu", "tau"), k = 2, ...)

stepGAIC(object, scope = gamlss.scope(model.frame(object)),
    direction = c("both", "backward", "forward"),
    trace = T, keep = NULL, steps = 1000,
    what = c("mu", "sigma", "nu", "tau"), k = 2,
    additive = FALSE, ...)

stepGAICA11.A(object, scope = NULL, sigma.scope = NULL, nu.scope = NULL,
    tau.scope = NULL, mu.try = TRUE, sigma.try = TRUE,
    nu.try = TRUE, tau.try = TRUE, ...)
```
Arguments

object

ana gamlss object. This is used as the initial model in the stepwise search.

scope

defines the range of models examined in the stepwise search. For the function
stepAIC() this should be either a single formula, or a list containing compo-
nents upper and lower, both formulae. See the details for how to specify the
formulae and how they are used. For the function stepGAIC the scope defines
the range of models examined in the step-wise search. It is a list of formulas,
with each formula corresponding to a term in the model. A 1 in the formula
allows the additional option of leaving the term out of the model entirely. +

direction

the mode of stepwise search, can be one of both, backward, or forward, with a
default of both. If the scope argument is missing the default for direction is
backward.

trace

if positive, information is printed during the running of stepAIC. Larger values
may give more information on the fitting process.

keep

a filter function whose input is a fitted model object and the associated 'AIC'
statistic, and whose output is arbitrary. Typically 'keep' will select a subset
of the components of the object and return them. The default is not to keep
anything.

steps

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

scale

scale is nor used in gamlss

what

which distribution parameter is required, default what="mu"

k

the multiple of the number of degrees of freedom used for the penalty. Only 'k =
2' gives the genuine AIC: 'k = log(n)' is sometimes referred to as BIC or SBC.

additive

if additive=TRUE then stepGAIC.CH is used else stepGAIC.CH, default value
is FALSE

sigma.scope

scope for sigma if different to scope in stepGAICA11.A()

nu.scope

scope for nu if different to scope in stepGAICA11.A()

tau.scope

scope for tau if different to scope in stepGAICA11.A()

mu.try

The default value is is TRUE, set to FALSE if no model for mu is needed

sigma.try

The default value is TRUE, set to FALSE if no model for sigma is needed

nu.try

The default value is TRUE, set to FALSE if no model for nu is needed

tau.try

The default value is TRUE, set to FALSE if no model for tau is needed

newdata

The new data set where the Test Global Deviance (TGD) will be evaluated

... any additional arguments to 'extractAIC'. (None are currently used.)
Details

The set of models searched is determined by the scope argument.

For the function `stepGAIC.VR()` the right-hand-side of its lower component is always included in the model, and right-hand-side of the model is included in the upper component. If scope is a single formula, it specifies the upper component, and the lower model is empty. If scope is missing, the initial model is used as the upper model.

Models specified by scope can be templates to update object as used by `update.formula`.

For the function `stepGAIC.CH()` each of the formulas in scope specifies a "regimen" of candidate forms in which the particular term may enter the model. For example, a term formula might be

\[- x1 + \log(x1) + cs(x1, df=3)\]

This means that x1 could either appear linearly, linearly in its logarithm, or as a smooth function estimated non-parametrically. Every term in the model is described by such a term formula, and the final model is built up by selecting a component from each formula.

The function `gamlss.scope` similar to the S `gam.scope()` in Chambers and Hastie (1991) can be used to create automatically term formulae from specified data or model frames.

The supplied model object is used as the starting model, and hence there is the requirement that one term from each of the term formulas of the parameters be present in the formula of the distribution parameter. This also implies that any terms in formula of the distribution parameter not contained in any of the term formulas will be forced to be present in every model considered.

When the smoother used in `gamlss` modeling belongs to the new generation of smoothers allowing the determination of the smoothing parameters automatically (i.e. `pb()`, `cy()` then the function `stepGAIC.VR()` can be used for model selection (see example below).

The function `stepTGD` is a clone function of `stepGAIC.VR()` where the selection criterion is not anymore a GAIC but the Test Global Deviance, that is, the deviance (-2log(Likelihood)) evaluated at the test data rather than the training ones.

Value

the stepwise-selected model is returned, with up to two additional components. There is an "anova" component corresponding to the steps taken in the search, as well as a "keep" component if the 'keep=' argument was supplied in the call. The "Resid. Dev" column of the analysis of deviance table refers to a constant minus twice the maximized log likelihood.

The function `stepGAICall.A()` returns with a component "anovaAll" containing all the different anova tables used in the process.

Author(s)

Mikis Stasinopoulos based on functions in MASS library and in Statistical Models in S

References


See Also
gamlss.scope

Examples

```r
## Not run:
data(usair)
# Note default of additive=FALSE
# fitting all variables linearly
mod1 <- gamlss(y~., data=usair, family=GA)
# find the best subset for the mu
mod2 <- stepGAIC(mod1)
mod2$anova
# find the best subset for sigma
mod3 <- stepGAIC(mod2, what="sigma", scope=-x1+x2+x3+x4+x5+x6)
mod3$anova
# find the best model using pb() smoother
only three variables are used here for simplicity
mod10 <- gamlss(y~1, data=usair, family=GA)
mod20 <- stepGAIC(mod10, scope=list(lower=-1, upper=pb(x1)+pb(x2)+pb(x5)))
edf(mod20)
# x1 and x2 enter linearly
# now use the stepGAIC.CH function
# creating a scope from the usair model frame
gs <- gamlss.scope(model.frame(y~x1+x2+x3+x4+x5+x6, data=usair))
gs
mod4 <- gamlss(y~1, data=usair, family=GA)
mod5 <- stepGAIC(mod4, gs, additive=TRUE)
mod5$anova
mod6 <- stepGAIC(mod5, what="sigma", scope=-x1+x2+x3+x4+x5+x6)
mod6$anova
mod6
# now stepGAICall.A
mod7 <- stepGAICall.A(mod4, scope=list(lower=-1,upper=-x1+x2+x3+x4+x5+x6))
# now stepGAICall.B
mod8 <- stepGAICall.B(mod4, scope=list(lower=-1,upper=-x1+x2+x3+x4+x5+x6))
# now stepTGID()
data(aep)
# sampling from the data
rand <- sample(2, dim(aep)[1], replace=TRUE, prob=c(0.6,0.4))
# the proportions in the sample
table(rand)/dim(aep)[1]
```
oldaep<-aep[rand==1,]
nuevaep<-aep[rand==2,]
m0<-gamlss(y=ward+year+loglos, data=oldaep, family=BB)
# checking mu
m1 <- stepTDG(m0, newdata=nuevaep)
# checking sigma
m2 <- stepTDG(m0, scope=ward+year, newdata=nuevaep, what="sigma")

## End(Not run)

summary.gamlss

Summarizes a GAMLSS fitted model

Description

summary.gamlss is the GAMLSS specific method for the generic function summary which summarize objects returned by modelling functions.

Usage

## S3 method for class 'gamlss'
summary(object, type = c("vcov", "qr"),
         robust=FALSE, save = FALSE,
         hessian.fun = c("R", "PB"),
         digits = max(3, getOption("digits") - 3),...)

Arguments

object a GAMLSS fitted model

type the default value vcov uses the vcov() method for gamlss to get the variance-covariance matrix of the estimated beta coefficients, see details below. The alternative qr is the original method used in gamlss to estimated the standard errors but it is not reliable since it do not take into the account the inter-correlation between the distributional parameters mu, sigma, nu and tau.

robust whether robust (sandwich) standard errors are required

save whether to save the environment of the function so to have access to its values

hessian.fun whether when calculate the Hessian should use the "R" function optimHess() or a function based on Pinheiro and Bates nlme package, "PB".

digits the number of digits in the output

... for extra arguments
Details

Using the default value type="vcov", the vcov() method for gamlss is used to get the variance covariance matrix (and consequently the standard errors) of the beta parameters. The variance covariance matrix is calculated using the inverse of the numerical second derivatives of the observed information matrix. This is a more reliable method since it take into the account the inter-correlation between the all the parameters. The type="qr" assumes that the parameters are fixed at the estimated values. Note that both methods are not appropriate and should be used with caution if smoothing terms are used in the fitting.

Value

Print summary of a GAMLSS object

Author(s)

Mikis Stasinopoulos <d.stasinopoulos@londonmet.ac.uk>, Bob Rigby <r.rigby@londonmet.ac.uk> and Calliope Akantziliotou

References


See Also

gamlss, deviance.gamlss, fitted.gamlss

Examples

data(aids)
h<-gamlss(y=poly(x,3)+qrt, family=PO, data=aids) #
summary(h)
rm(h)
term.plot

Plot regression terms for a specified parameter of a fitted GAMLSS object

Description

Plots regression terms against their predictors, optionally with standard errors and partial residuals added. It is based on the R function `term.plot` but is suitably changed to apply to GAMLSS objects.

Usage

```r
term.plot(object, what = c("mu", "sigma", "nu", "tau"), data = NULL,
          envir = environment(formula(object)), partial.resid = FALSE,
          rug = FALSE, terms = NULL, se = TRUE, ylim = c("common", "free"),
          scheme = c("shaded", "lines"), xlabs = NULL, ylabs = NULL,
          main = NULL, pages = 0, col.term = "darkred",
          col.se = "orange", col.shaded = "gray", col.res = "lightblue",
          col.rug = "gray", lwd.term = 1.5, lty.se = 2, lwd.se = 1,
          cex.res = 1, pch.res = par("pch"),
          ask = interactive() & nb.fig < n.tms & & .Device != "postscript",
          use.factor.levels = TRUE, surface.gam = FALSE, ...)
```

Arguments

object a fitted GAMLSS object
what the required parameter of the GAMLSS distribution i.e. "mu"
data data frame in which variables in `object` can be found
envir environment in which variables in `object` can be found
partial.resid logical; should partial residuals be plotted or not
rug add rug plots (jitter 1-d histograms) to the axes?
terms which terms to be plotted (default `NULL` means all terms)
se plot point-wise standard errors?
ylim there are two options here a) "common" and b) "free". The "common" option plots all figures with the same ylim range and therefore allows the viewer to check the relative contribution of each terms compare to the rest. In the "free" option the limits are computed for each plot seperately.
scheme whether the se's should appear shaded or as lines
xlabs vector of labels for the x axes
ylabs vector of labels for the y axes
main logical, or vector of main titles; if 'TRUE', the model's call is taken as main title, 'NULL' or 'FALSE' mean no titles.
**term.plot**

- **pages**: in how many pages the plot should appear. The default is 0 which allows different page for each plot.

- **col.term**: the colour of the term line.

- **col.se**: the colour of the se's lines.

- **col.shaded**: the colour of the shaded area.

- **col.res**: the colour of the partial residuals.

- **col.rug**: the colour of the rug.

- **lwd.term**: line width of the fitted terms.

- **lty.se**: line type for standard errors.

- **lwd.se**: line width for the standard errors.

- **cex.res**: plotting character expansion for the partial residuals.

- **pch.res**: characters for points in the partial residuals.

- **ask**: logical; if 'TRUE', the user is asked before each plot, see 'par(ask=.)'.

- **use.factor.levels**: Should x-axis ticks use factor levels or numbers for factor terms?

- **surface.gam**: whether to use surface plot if a ga() term is fitted.

- **...**: other graphical parameters.

**Details**

The function uses the `lpred` function of GAMLSS. The 'data' argument should rarely be needed, but in some cases 'termplot' may be unable to reconstruct the original data frame. Using 'na.action=na.exclude' makes these problems less likely. Nothing sensible happens for interaction terms.

**Value**

- a plot of fitted terms.

**Author(s)**

Mikis Stasinopoulos based on the existing `termplot()` function.

**References**


See Also

termplot

Examples

data(aids)
a <- gamlss(y ~ pb(x) + qrt, data = aids, family = NBI)
term.plot(a, pages = 1)
rm(a)

update.gamlss  

Description

update.gamlss is the GAMLSS specific method for the generic function update which updates and (by default) refits a GAMLSS model.

Usage

## S3 method for class 'gamlss'
update(object, formula, ..., what = c("mu", "sigma", "nu", "tau", "All"),
evaluate = TRUE)

Arguments

object
formula.
...  
what
evaluate

a GAMLSS fitted model
the formula to update
for updating argument in gamlss()
the parameter in which the formula needs updating for example "mu", "sigma", "nu" "tau" or "All". If "All" all the formulae are updated. Note that the what argument has an effect only if only if the argument formula. is set
whether to evaluate the call or not

Value

Returns a GAMLSS call or fitted object.

Author(s)

Mikis Stasinopoulos <d.stasinopoulos@londonmet.ac.uk>, Bob Rigby <r.rigby@londonmet.ac.uk>
References


See Also

`print.gamlss`, `summary.gamlss`, `fitted.gamlss`, `coef.gamlss`, `residuals.gamlss`, `plot.gamlss`, `deviance.gamlss`, `formula.gamlss`

Examples

data(aids)
# fit a poisson model
h.po <- gamlss(y~pb(x)+qrt, family=PO, data=aids)
# update with a negative binomial
h.nb <- update(h.po, family=NBI)
# update the smoothing
h.nb1 <- update(h.nb, ~cs(x,8)+qrt)
# remove qrt
h.nb2 <- update(h.nb1, ~-.qrt)
# put back qrt take log of y and fit a normal distribution
h.nb3 <- update(h.nb1, log(.)~+.qrt, family=NO)
# verify that it is the same
h.no <- gamlss(log(y)~cs(x,8)+qrt, data=aids )

---

**VGD**  
.Validation (or test) data global deviance

Description

These function are useful for model selection. The function VGD helps to validate a GAMLSS model by randomly splitting the data into training and validation sets. It minimizes the global deviance for the training data set and then uses the validation set to calculate the prediction global deviance. The function VGD1 is identical to VGD but the output is a list rather than a values as in VGD. The function VGD2 is identical to VGD1 but it takes as argument the new data, (newdata), rather than a factor which split the combined data in two as in VGD or VGD1. The function TGD takes a fitted gamlss object and produce the test global deviance for the new data. The resulting prediction errors can be used for selecting the distribution of the model, the terms in the model or degrees of freedom for smoothing terms.
Usage

VGD(formula = NULL, sigma.formula = ~1, nu.formula = ~1, tau.formula = ~1,
data = NULL, family = NO,
control = gamlss.control(trace = FALSE), rand = NULL, ...)
VGD1(formula = NULL, sigma.formula = ~1, nu.formula = ~1, tau.formula = ~1,
data = NULL, family = NO,
control = gamlss.control(trace = FALSE), rand = NULL, ...)
VGD2(formula = NULL, sigma.formula = ~1, nu.formula = ~1, tau.formula = ~1,
data = NULL, family = NO,
control = gamlss.control(trace = FALSE),
newdata = NULL, ...)
TGD(object, newdata = NULL, ...)

Arguments

object a gamlss object
formula a gamlss formula for mu (including the response on the left)
sigma.formula a formula for sigma
nu.formula a formula for nu
tau.formula a formula for tau
data the data set used for the fitting
newdata the data set to be used for validation or test
family a gamlss.family object
control gamlss.control to be passed to gamlss
rand a random vector of one and two indicating whether is the training set (1) or the
validation set (2) i.e. created in advance using something like rand <- sample(2, N, replace=T, prob=c(...)
where N is the length of the data
...
for extra arguments to be passed in the gamlss fit

Value

The function VGD returns a validated global deviance, that is, the global deviance evaluated at the
validation data. The functions VGD1, VGD2 and TGD are returning a list with component oldGD,
ewGD, oldPE, newPE. oldGD is the deviance of the training data, newGD is the deviance for the
validation or (test) data. oldPE is the mean prediction error for the old data (oldGD divided by the
number of observations in the training data). newPE is the mean prediction error for the new data,
(newGD divided by the number of observations in the validation (or test) data).

Author(s)

Mikis Stasinopoulos <d.stasinopoulos@londonmet.ac.uk> and Bob Rigby <r.rigby@londonmet.ac.uk>
References


See Also
gamlss.family, gamlss, deviance.gamlss

Examples

data(abdom)
# generate the random split of the data
rand <- sample(2, 610, replace=TRUE, prob=c(0.6, 0.4))
# the proportions in the sample
table(rand)/610
# VGD needs a factor
VDG(y~pb(x,df=2),sigma.formula=-pb(x,df=1), data=abdom, family=L0, rand=rand)
# VGD1 needs a factor and has different output
VDG1(y~pb(x,df=2),sigma.formula=-pb(x,df=1), data=abdom, family=L0, rand=rand)
# using VGD2 needs two different sets of data
VDG2(y~pb(x,df=2),sigma.formula=-pb(x,df=1), data=abdom[rand==1,], family=L0,
newdata=abdom[rand==2, ] )
# fit in the old data
olddata<-abdom[rand==1,]
m1<-gamlss(y~pb(x,df=2),sigma.formula=-pb(x,df=1), data=olddata, family=L0)
# get the global deviance in the new data
newdata<-abdom[rand==2,]
TG(m1, newdata=newdata)

Definition

Provides a single plot or multiple worm plots for a GAMLSS fitted or more general for any fitted models where the method resid() exist and the residuals are defined sensibly. The worm plot (a detrended QQ-plot), van Buuren and Fredriks M. (2001), is a diagnostic tool for checking the residuals within different ranges (by default not overlapping) of the explanatory variable(s).
Usage

wp(object=NULL, xvar = NULL, resid = NULL, n.inter = 4, xcut.points = NULL, overlap = 0, xlim.all = 4, xlim.worm = 3.5, show.given = TRUE, line = TRUE, ylim.all = 12 * sqrt(1/length(resid)), ylim.worm = 12 * sqrt(n.inter/length(resid)), cex = 1, pch = 21, ...)

Arguments

- **object**: a GAMLSS fitted object or any other fitted model where the resid() method works (preferably it should be standarised or quantile residuals)
- **xvar**: the explanatory variable(s) against which the worm plots will be plotted. If only one variable is involved use xvar=x1 if two variables are involved use xvar=x1+x2. See also note below for use of formula if the data argument is not found in the fitted model
- **resid**: if object is missing this argument can be used to specify the residual vector (again it should a quantile residuals or it be assumed to come from a normal distribution)
- **n.inter**: the number of intervals in which the explanatory variable xvar will be cut
- **xcut.points**: the x-axis cut off points e.g. c(20,30). If xcut.points=NULL then the n.inter argument is activated
- **overlap**: how much overlapping in the xvar intervals. Default value is overlap=0 for non overlapping intervals
- **xlim.all**: for the single plot, this value is the x-variable limit, default is xlim.all=4
- **xlim.worm**: for multiple plots, this value is the x-variable limit, default is xlim.worm=3.5
- **show.given**: whether to show the x-variable intervals in the top of the graph, default is show.given=TRUE
- **line**: whether to plot the polynomial line in the worm plot, default value is line=TRUE
- **ylim.all**: for the single plot, this value is the y-variable limit, default value is ylim.all=12*sqrt(1/length(fitted(object)))
- **ylim.worm**: for multiple plots, this value is the y-variable limit, default value is ylim.worm=12*sqrt(n.inter/length(fitted(object)))
- **cex**: the cex plotting parameter with default cex=1
- **pch**: the pch plotting parameter with default pch=21
- ... for extra arguments

Details

If the xvar argument is not specified then a single worm plot is used. In this case a worm plot is a de-trended normal QQ-plot so departure from normality is highlighted.

If a single xvar is specified (with or without the use of a formula) i.e. xvar=x1 or xvar=x1) then we have as many worm plot as n.inter. In this case the x-variable is cut into n.inter intervals with an equal number observations and de-trended normal QQ (i.e. worm) plots for each interval are plotted. This is a way of highlighting failures of the model within different ranges of the the single
explanatory variable. The fitted coefficients from fitting cubic polynomials to the residuals (within each x-variable interval) can be obtain by e.g. `coeffs<-wp(model1,xvar=x,n.inter=9)`. van Buuren and Fredriks M. (2001) used these residuals to identify regions (intervals) of the explanatory variable within which the model does not fit adequately the data (called "model violation")

Two variables can be displayed with the use of a formula, i.e. `xvar=x1+x2`. In this case the `n.inter` can be a vector with two values.

Value

For multiple plots the `xvar` intervals and the coefficients of the fitted cubic polynomials to the residuals (within each `xvar` interval) are returned.

Note

Note that the `wp()` function, if the argument object is used, is looking for the data argument of the object. If the argument data exists it uses its enviroment to find `xvar` (whether it is a formula or not). As a result if data exists withing object `xvar=x*f` can be used (assuming thet x and f are in the data) otherwise the variable should be explicitly defined i.e. `xvar=data$x*data$f`.

Author(s)

Mikis Stasinopoulos and Bob Rigby

References


See Also

gamlss, plot.gamlss

Examples

data(abdom)
# with data
a<-gamlss(y~pb(x),sigma.fo=~pb(x,1),family=L0,data=abdom)
wp(a)
coeffI<-wp(a,xvar=x)
coeffI
## Not run:
z.scores

Z-scores for lms objects

Description

This creates z-scores for new values of y and x given a fitted lms object.

Usage

z.scores(object, y, x)
z.scores

**Arguments**

- **object**: a lms fitted object
- **y**: new y values
- **x**: new x values

**Details**

This is simply a job that can be also done by `centiles.pred()`.

**Value**

the required z-scores

**Author(s)**

Mikis Stasinopoulos

**References**


**See Also**

`centiles.pred`

**Examples**

```r
## Not run:
IND<-sample.int(7040, 1000, replace=FALSE)
db1 <- db[IND,]
plot(head~age, data=db1)
m0 <- lms(head, age, data=db1,trans.x=TRUE )
z.scores(m0, x=c(2,15,30,40),y=c(45,50,56,63))
## End(Not run)
```
Index

*Topic `textasciitilde`dtop, 24
*Topic `distribution`
  fitDist, 30
  histSmo, 55
  loglogSurv, 64
  plot.histSmo, 75
*Topic `package`
  gamlss-package, 3
*Topic `regression`
  additive.fit, 5
  bfp, 7
  calibration, 9
  centiles, 10
  centiles.com, 13
  centiles.pred, 14
  centiles.split, 17
  coef.gamlss, 19
  cs, 20
  deviance.gamlss, 23
  edf, 26
  find.hyper, 27
  fitDist, 30
  fitted.gamlss, 32
  fittedPlot, 33
  formula.gamlss, 34
  gamlss, 35
  gamlss.control, 39
  gamlss.cs, 41
  gamlss.fp, 42
  gamlss.lo, 43
  gamlss.ps, 44
  gamlss.random, 45
  gamlss.scope, 46
  gamlssML, 47
  gen.likelihood, 49
  getSmo, 51
  glim.control, 52
  histDist, 53
  IC, 58
  lms, 59
  lo, 62
  lpred, 66
  LR.test, 67
  model.frame.gamlss, 68
  par.plot, 70
  pdf.plot, 71
  plot.gamlss, 73
  plot2way, 76
  polyS, 77
  predict.gamlss, 78
  print.gamlss, 80
  prof.dev, 81
  prof.term, 83
  ps, 86
  Q.stats, 90
  quantSheets, 92
  random, 94
  refit, 98
  residuals.gamlss, 100
  ridge, 101
  rqres.plot, 103
  Rsq, 104
  rvcov, 106
  stepGAIC, 107
  summary.gamlss, 112
  term.plot, 114
  update.gamlss, 116
  VGD, 117
  wp, 119
  z.scores, 122
*Topic `regression`
  acfResid, 4
*Topic `ts`
  acfResid, 4
  acf, 5
  acfResid, 4
  additive.fit, 5, 7, 86
par.plot, 70
pb, 22, 45, 57
pb (ps), 86
pbo (ps), 86
pdf.plot, 39, 71
plot.gamlss, 28, 32, 73, 101, 104, 117, 121
plot2way, 76
poly.matrix (polyS), 77
polyS, 77
pp (bfp), 7
predict.gamlss, 67, 78
predictAll (predict.gamlss), 78
print.gamlss, 32, 80, 101, 117
prof.dev, 81, 85
prof.term, 83, 83
ps, 45, 86
pvc, 22, 45
pvc (ps), 86
Q.stats, 90
quantSheets, 92
random, 46, 63, 94
re (random), 94
refit, 39, 98
residuals.gamlss, 32, 100, 101, 117
ri, 45
ri (ridge), 101
ridge, 45, 101
rqres.plot, 100, 103
Rsq, 104
rvcov, 106
scs (cs), 20
stepGAIC, 47, 107
stepGAICall.A (stepGAIC), 107
stepGAICall.B (stepGAIC), 107
stepTGD (stepGAIC), 107
summary.gamlss, 32, 101, 112, 117
term.plot, 77, 114
termplot, 116
terms.gamlss (model.frame.gamlss), 68
TGD (VGD), 117
update, 99
vcov, 50, 107
VGD, 117
VGD1 (VGD), 117
VGD2 (VGD), 117
wp, 25, 91, 119
z.scores, 122
z.scoresQS (quantSheets), 92
update.gamlss, 32, 99, 101, 116