Package ‘gamlss’

October 6, 2019

Description  Functions for fitting the Generalized Additive Models for Location Scale and Shape introduced by Rigby and Stasinopoulos (2005), <doi:10.1111/j.1467-9876.2005.00510.x>. The models use a distributional regression approach where all the parameters of the conditional distribution of the response variable are modelled using explanatory variables.

Version 5.1-5

Date 2019-10-06

Title Generalised Additive Models for Location Scale and Shape

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

Depends R (>= 3.3.0), graphics, stats, splines, utils, grDevices, gamlss.data (>= 5.0-0), gamlss.dist (>= 4.3.1), nlme, parallel

LazyLoad yes

Imports MASS, survival, methods

License GPL-2 | GPL-3

URL http://www.gamlss.org/

NeedsCompilation yes

Author Mikis Stasinopoulos [aut, cre, cph],
Bob Rigby [aut],
Vlasios Voudouris [ctb],
Calliope Akantziliotou [ctb],
Marco Enea [ctb],
Daniil Kiose [ctb]

Repository CRAN

Date/Publication 2019-10-06 08:00:02 UTC

R topics documented:

gamlss-package ................................................................................ 3
acfResid ......................................................................................... 7
additive.fit .................................................................................. 8
bfp ............................................................................................... 9
calibration ................................................................................... 11
<table>
<thead>
<tr>
<th>R topics documented:</th>
</tr>
</thead>
<tbody>
<tr>
<td>centiles</td>
</tr>
<tr>
<td>centiles.com</td>
</tr>
<tr>
<td>centiles.pred</td>
</tr>
<tr>
<td>centiles.split</td>
</tr>
<tr>
<td>coef.gamlss</td>
</tr>
<tr>
<td>cs</td>
</tr>
<tr>
<td>deviance.gamlss</td>
</tr>
<tr>
<td>devianceIncr</td>
</tr>
<tr>
<td>dtop</td>
</tr>
<tr>
<td>edf</td>
</tr>
<tr>
<td>find.hyper</td>
</tr>
<tr>
<td>fitDist</td>
</tr>
<tr>
<td>fitted.gamlss</td>
</tr>
<tr>
<td>fittedPlot</td>
</tr>
<tr>
<td>formula.gamlss</td>
</tr>
<tr>
<td>gamlss</td>
</tr>
<tr>
<td>gamlss.control</td>
</tr>
<tr>
<td>gamlss.cs</td>
</tr>
<tr>
<td>gamlss.fp</td>
</tr>
<tr>
<td>gamlss.lo</td>
</tr>
<tr>
<td>gamlss.ps</td>
</tr>
<tr>
<td>gamlss.random</td>
</tr>
<tr>
<td>gamlss.scope</td>
</tr>
<tr>
<td>gamlssML</td>
</tr>
<tr>
<td>gamlssVGD</td>
</tr>
<tr>
<td>gen.likelihood</td>
</tr>
<tr>
<td>getPEF</td>
</tr>
<tr>
<td>getQuantile</td>
</tr>
<tr>
<td>getSmo</td>
</tr>
<tr>
<td>glm.control</td>
</tr>
<tr>
<td>histDist</td>
</tr>
<tr>
<td>histSmo</td>
</tr>
<tr>
<td>IC</td>
</tr>
<tr>
<td>lms</td>
</tr>
<tr>
<td>lo</td>
</tr>
<tr>
<td>loglogSurv</td>
</tr>
<tr>
<td>lpred</td>
</tr>
<tr>
<td>LR.test</td>
</tr>
<tr>
<td>model.frame.gamlss</td>
</tr>
<tr>
<td>numeric.deriv</td>
</tr>
<tr>
<td>par.plot</td>
</tr>
<tr>
<td>pcat</td>
</tr>
<tr>
<td>pdf.plot</td>
</tr>
<tr>
<td>plot.gamlss</td>
</tr>
<tr>
<td>plot.histSmo</td>
</tr>
<tr>
<td>plot2way</td>
</tr>
<tr>
<td>polyS</td>
</tr>
<tr>
<td>predict.gamlss</td>
</tr>
</tbody>
</table>
Description

Functions for fitting the Generalized Additive Models for Location Scale and Shape introduced by Rigby and Stasinopoulos (2005), <doi:10.1111/j.1467-9876.2005.00510.x>. The models use a distributional regression approach where all the parameters of the conditional distribution of the response variable are modelled using explanatory variables.

Details

The DESCRIPTION file:

```
Packages:    gamlss
Description: Functions for fitting the Generalized Additive Models for Location Scale and Shape introduced by Rigby and Stasinopoulos
Version:    5.1-5
Date:       2019-10-06
Authors@R:  c(person("Mikis", "Stasinopoulos", role = c("aut", "cre", "cph"), email = "d.stasinopoulos@londonmet.ac.uk"),
Title:      Generalised Additive Models for Location Scale and Shape
Maintainer: Mikis Stasinopoulos <d.stasinopoulos@londonmet.ac.uk>
Depends:    R (>= 3.3.0), graphics, stats, splines, utils, grDevices, gamlss.data (>= 5.0-0), gamlss.dist (>= 4.3.1), nlme, parallel,
LazyLoad:   yes
Imports:    MASS, survival, methods
License:    GPL-2 | GPL-3
```
Index of help topics:

IC           Gives the GAIC for a GAMLSS Object
LR.test      Likelihood Ratio test for nested GAMLSS models
Q.stats      A function to calculate the Q-statistics
Rsq          Generalised (Pseudo) R-squared for GAMLSS models
VC.test      Vuong and Clarke tests
acfResid     ACF plot of the residuals
additive.fit  Implementing Backfitting in GAMLSS
bfp          Functions to fit fractional polynomials in GAMLSS
calibration  Calibrating centile curves
centiles     Plots the centile curves for a GAMLSS object
centiles.com  Comparing centiles from different GAMLSS models
centiles.pred Creating predictive centiles values
centiles.split Plots centile curves split by x for a GAMLSS object
coeff.gamlss Extract Model Coefficients in a GAMLSS fitted model
cs           Specify a Smoothing Cubic Spline Fit in a GAMLSS Formula
deviance.gamlss Global Deviance of a GAMLSS model
devianceIncr  The global deviance increment
dtop          Detrended transformed Owen's plot
edf           Effective degrees of freedom from gamlss model
find.hyper    A function to select values of hyper-parameters in a GAMLSS model
fitDist       Fitting Different Parametric 'gamlss.family' Distributions.
fitted.gamlss Extract Fitted Values For A GAMLSS Model
fittedPlot    Plots The Fitted Values of a GAMLSS Model
formula.gamlss Extract the Model Formula in a GAMLSS fitted model
gamlss        Generalized Additive Models for Location Scale and Shape
gamlss-package Generalised Additive Models for Location Scale and Shape
gamlss.control Auxiliary for Controlling GAMLSS Fitting
gamlss.cs     Support for Function cs() and scs()
gamlss.fp     Support for Function fp()
gamlss.lo     Support for Function lo()
gamlss.ps     Support for Functions for smoothers
gamlss.random Support for Functions random() and re()
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>gamlss.scope</code></td>
<td>Generate a Scope Argument for Stepwise GAMLSS</td>
</tr>
<tr>
<td><code>gamlssML</code></td>
<td>Maximum Likelihood estimation of a simple GAMLSS model</td>
</tr>
<tr>
<td><code>gamlssVGD</code></td>
<td>A Set of Functions for selecting Models using Validation or Test Data Sets and Cross Validation</td>
</tr>
<tr>
<td><code>gen.likelihood</code></td>
<td>A function to generate the likelihood function from a GAMLSS object</td>
</tr>
<tr>
<td><code>getPEF</code></td>
<td>Getting the partial effect function from a continuous term in a GAMLSS model</td>
</tr>
<tr>
<td><code>getQuantile</code></td>
<td>Getting the partial quantile function for a term</td>
</tr>
<tr>
<td><code>getSmo</code></td>
<td>Extracting Smoother information from a GAMLSS fitted object</td>
</tr>
<tr>
<td><code>glim.control</code></td>
<td>Auxiliary for Controlling the inner algorithm in a GAMLSS Fitting</td>
</tr>
<tr>
<td><code>histDist</code></td>
<td>This function plots the histogram and a fitted (GAMLSS family) distribution to a variable</td>
</tr>
<tr>
<td><code>histSmo</code></td>
<td>Density estimation using the Poisson trick</td>
</tr>
<tr>
<td><code>lms</code></td>
<td>A function to fit LMS curves for centile estimation</td>
</tr>
<tr>
<td><code>lo</code></td>
<td>Specify a loess fit in a GAMLSS formula</td>
</tr>
<tr>
<td><code>loglogSurv</code></td>
<td>Survival function plots for checking the tail behaviour of the data</td>
</tr>
<tr>
<td><code>lpred</code></td>
<td>Extract Linear Predictor Values and Standard Errors For A GAMLSS Model</td>
</tr>
<tr>
<td><code>model.frame.gamlss</code></td>
<td>Extract a model.frame, a model matrix or terms from a GAMLSS object for a given distributional parameter</td>
</tr>
<tr>
<td><code>numeric.deriv</code></td>
<td>An internal GAMLSS function for numerical derivatives</td>
</tr>
<tr>
<td><code>par.plot</code></td>
<td>A function to plot parallel plot for repeated measurement data</td>
</tr>
<tr>
<td><code>pcat</code></td>
<td>Reduction for the Levels of a Factor.</td>
</tr>
<tr>
<td><code>pdf.plot</code></td>
<td>Plots Probability Distribution Functions for GAMLSS Family</td>
</tr>
<tr>
<td><code>plot.gamlss</code></td>
<td>Plot Residual Diagnostics for an GAMLSS Object</td>
</tr>
<tr>
<td><code>plot.histSmo</code></td>
<td>A Plotting Function for density estimator object histSmo</td>
</tr>
<tr>
<td><code>plot2way</code></td>
<td>Function to plot two interaction in a GAMLSS model</td>
</tr>
<tr>
<td><code>polyS</code></td>
<td>Auxiliary support for the GAMLSS</td>
</tr>
<tr>
<td><code>predict.gamlss</code></td>
<td>Extract Predictor Values and Standard Errors For New Data In a GAMLSS Model</td>
</tr>
<tr>
<td><code>print.gamlss</code></td>
<td>Prints a GAMLSS fitted model</td>
</tr>
<tr>
<td><code>prof.dev</code></td>
<td>Plotting the Profile Deviance for one of the Parameters in a GAMLSS model</td>
</tr>
<tr>
<td><code>prof.term</code></td>
<td>Plotting the Profile: deviance or information</td>
</tr>
</tbody>
</table>
criterion for one of the terms (or hyper-parameters) in a GAMLSS model

- `ps` P-Splines Fits in a GAMLSS Formula
- `quantSheets` Quantile Sheets
- `random` Specify a random intercept model in a GAMLSS formula
- `refit` Refit a GAMLSS model
- `residuals.gamlss` Extract Residuals from GAMLSS model
- `ri` Specify ridge or lasso Regression within a GAMLSS Formula
- `rqres.plot` Creating and Plotting Randomized Quantile Residuals
- `rvcov` Robust Variance-Covariance matrix of the parameters from a fitted GAMLSS model
- `stepGAIC` Choose a model by GAIC in a Stepwise Algorithm
- `summary.gamlss` Summarizes a GAMLSS fitted model
- `term.plot` Plot regression terms for a specified parameter of a fitted GAMLSS object
- `update.gamlss` Update and Re-fit a GAMLSS Model
- `wp` Worm plot
- `z.scores` Z-scores for lms objects

**Author(s)**

NA

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

**References**


(see also [http://www.gamlss.com/](http://www.gamlss.com/)).

**See Also**

`gamlss.dist`
acfResid

Examples

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

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

acfResid(obj = NULL, resid = NULL)

Arguments

obj A gamlss model or other 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 displayied

Author(s)

Mikis Stasinopoulos. Bob Rigby. Vlasios Voudouris and Majid Djennad

References

additive.fit

See Also

acf

Examples

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

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 [http://www.gamlss.com/](http://www.gamlss.com/)).

**See Also**

gamlss

---

**bfp**  
*Functions to fit fractional polynomials in GAMLSS*

**Description**

The function `bfp` generate 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 use to fit power polynomials as in \( a + b_1 x^{p_1} + b_2 x^{p_2} \), 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)
fpm(x, npoly = 2, shift = NULL, scale = NULL)
ppm(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 \(10^{\text{sign}(\log_{10}(\text{range}))}*\text{trunc}(\text{abs}(\log_{10}(\text{range})))}\)

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 in 1999. (unfortunately the URL link for his work no longer exist). 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 used in the fitting. For a fixed number npoly the algorithm looks for the best fitting fractional polynomials in the list c(-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 that one fractional polynomials are fitted?

Author(s)

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


(see also [http://www.gamlss.com/](http://www.gamlss.com/)).

See Also

`gamlss, gamlss.family`

Examples

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
m1<-gamlss(y~fp(x,1),data=abdom)
# 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$mu.coefSmo
# now power polynomials using the best 2 fp c()
# This is not good idea in this case because
m4 <- gamlss(y ~ pp(x, c(1,3)), data = abdom)
# if you look at the fitted values you see what it went wrong
plot(y~x,data=abdom)
lines(fitted(m2,“mu”)~abdom$x,col=“red“)
lines(fitted(m4,“mu”)~abdom$x,col=“blue“)

calibration

**Calibrating centile curves**

Description

This function can 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, ...)
```
calibration

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)

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 [http://www.gamlss.com/](http://www.gamlss.com/)).

See Also

- `centiles`, `centiles.fan`

Examples

```r
data(abdom)
m1<-gamlss(y~pb(x), sigma.fo=~pb(x), family=LO, 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, cent = c(0.4, 2, 10, 25, 50, 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, cent = c(0.4, 2, 10, 25, 50, 90, 98, 99.6),
             ylab = "y", xlab = "x", main = NULL, main.gsub = "@",
             xleg = min(xvar), yleg = max(obj$y), xlim = range(xvar),
             ylim = range(obj$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 substituted 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 predictor 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.

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

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

References
(see also http://www.gamlss.com/).
centiles.com

Comparing centiles from different GAMLSS models

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, 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
The function `centiles.com` produces centile plots for the different fitted models and the sample centiles below each centile curve are printed.

**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 [http://www.gamlss.com/](http://www.gamlss.com/)).

**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)
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

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 http://www.gamlss.com/).

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)
## the first use of the function 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 )
## the second use of the function centiles.pred()
## to calculate (nornalised) 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 )

# the third use of the function centiles.pred()
## if we have new x and y values what are their z-scores?
# create new y and x values and plot them in the previous plot
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 is working in this case
centiles(aa, xvar=abdom$x, legend = FALSE)
## get predict for values of x at 12, 14, ..., 40
mat <- centiles.pred(aa, xname="x", xvalues=seq(12,40,2), plot=TRUE )
mat
# plot all prediction points
xx <- rep(mat[,1],9)
yy <- unlist(mat[,2:10])
points(xx,yy,col="red")
## case 2 : the x-variable is previously transformed
## equivalent to fitting
newd<-data.frame( abdom, nx=abdom$x^0.5)
aa1 <- gamlss(y~pb(nx),sigma.fo=~pb(nx), family=BCT, data=newd)
centiles(aa1, xvar=abdom$x)
## getting the centiles at x equal to 12, 14, ...40
mat <- centiles.pred(aa1, xname="nx", xvalues=seq(12,40,2), power=0.5,
data=newd, plot=TRUE)
# plot all prediction points
xxx <- rep(mat[,1],9)
yyy <- unlist(mat[,2:10])
points(xxx,yyy,col="red")
# the idea is that if the transformed x-variable is used in the fit
# the power argument has to used in centiles.pred()
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.point 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

```r
centiles.split(obj, xvar, 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 split, with default 4
- `cent` a vector with elements the % centile values for which the centile curves are to be evaluated
- `legend` whether a legend is required in the plots or not, the default is `legend=FALSE`
- `main` the main title as character. If NULL the default title (shown the intervals) is shown
- `main.gsub` if the `main.gsub` (with default "@") appears in the `main` title then it is substituted with the default title.
- `ylab` the y-variable label
- `xlab` the x-variable label
- `ylim` the range of the y-variable axis
- `overlap` how much overlapping in the xvar intervals. Default value is `overlap=0` for non overlapping intervals
- `save` 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
- `plot` 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 [http://www.gamlss.com/](http://www.gamlss.com/)).

See Also

gamlss centiles, centiles.com

Examples

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)
Usage

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

Arguments

- `object`: a GAMLSS fitted model
- `what`: which parameter coefficient is required, default `what="mu"`
- `parameter`: equivalent to `what` (more obvious name)
- `...`: for extra arguments

Value

Coefficients extracted from the GAMLSS model object.

Author(s)

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

References


(see also [http://www.gamlss.com/](http://www.gamlss.com/)).

See Also

- `gamlss`, `deviance.gamlss`, `fitted.gamlss`

Examples

```r
data(aids)
h<-gamlss(y~poly(x,3)+qrt, family=PO, data=aids) #
coef(h)
rm(h)
```
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 default values used are the ones given the option control.spar in the R function 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 smooth.spline() for the original authors of the cubic spline function.)

References


(see also http://www.gamlss.com/).

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))
lines(aids$x, fitted(aids1), col="red")
lines(aids$x, fitted(aids3), col="green")
rm(aids1, aids2, aids3)

deviance.gamlss

Global Deviance of a GAMLSS model

Description

Returns the global, -2*log(likelihood), or the penalized, -2*log(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 http://www.gamlss.com/).

See Also

- `gamlss.family`
- `coef.gamlss`
- `fitted.gamlss`

Examples

```r
data(aids)
h<-gamlss(y~poly(x,3)+qrt, family=PO, data=aids) #
deviance(h)
rm(h)
```
The global deviance increment is the contribution of each individual observation to the global deviance. The function `devianceIncr()` can be used to extract the global deviance increment for a fitted gamlss model or for a new (test/validation) data set. Large values for global deviance increment indicate a bad fit and for new data a bad prediction.

**Usage**

```r
devianceIncr(obj, newdata = NULL)
```

**Arguments**

- `obj`: a `gamlss` object
- `newdata`: test data set to check the global deviance increment.

**Value**

Returns a vector of the global deviance increments for each observation.

**Author(s)**

Mikis Stasinopoulos

**References**


(see also [http://www.gamlss.com/](http://www.gamlss.com/)).

**See Also**

`deviance`
Examples

```r
# Count data set
# fit Poisson model
h1 <- gamlss(Claims~L_Population+L_Accidents+L_KI+L_Popdensity,
             data=LGAclaims, family=PO)
p1<-devianceIncr(h1)
# fit negative binomial model
h2 <- gamlss(Claims~L_Population+L_Accidents+L_KI+L_Popdensity,
             data=LGAclaims, family=NBI)
p2<-devianceIncr(h2)
# comparing using boxplots
boxplot(cbind(p1,p2))
# comparing using empirical cdf
plot(ecdf(p1))
lines(ecdf(p2), col=2)
# comparing against the y-values
plot(p1~LGAclaims$Claims, pch=20, col="gray")
points(p2~LGAclaims$Claims, pch="-", col="orange")
```

```r
# Continuous data sets
## Not run:
m1 <- gamlss(head~pb(age), data=db[1:6000,])
p1<-devianceIncr(m1)
m2 <- gamlss(head~pb(age), sigma.fo~pb(age), nu.fo=-pb(age),
             tau.fo~pb(age), data=db[1:6000,], family=BCT)
p2<d.evianceIncr(m2)
# comparing using summaries
summary(p1); summary(p2)
# comparing using boxplots
boxplot(cbind(p1,p2))
# comparing using histograms
hist(p1, col=rgb(1,0,0,0.5), xlim=c(0,50), breaks=seq(0,50,2))
hist(p2, col=rgb(0,0,1,0.5), add=T)
# comparing using empirical cdf
plot(ecdf(p1))
lines(ecdf(p2), col=2)
## End(Not run)
```

```

dtop

Detrended transformed Owen’s plot

dtop

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,
    type = c("Owen", "JW"),
    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 he 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 http://www.gamlss.com/).

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)

edf

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

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

Arguments

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

Value

The function edfAll() returns a list of edf for all the fitted parameters. The function edf() returns 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 constant and the linear part.

Author(s)

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

References


(see also http://www.gamlss.com/).

See Also

gamlss

Examples

library(gamlss.data)
data(usair)
m1<- gamlss(y~pb(x1)+pb(x2)+pb(x6), data=usair)
edfAll(m1)
edf(m1)

find.hyper 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.
**find.hyper**

**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.
  ```r
  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 `ndeps` 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(30,10)`, 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

- **...**
  - for extra arguments to be passed to the R function `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 `pb()` the function `find.hyper()` became almost redundant. `find.hyper()` takes lot longer than `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 `optim()`

- **par**
  - the optimum hyper-parameter values

- **value**
  - the minimised value of the GAIC

- **counts**
  - A two-element integer vector giving the number of calls to ‘fn’ and ‘gr’ respectively

- **convergence**
  - An integer code. ‘0’ indicates successful convergence. see the function `optim()` for other errors
find.hyper

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 http://www.gamlss.com/).

See Also

gamlss.plot.gamlss.optim

Examples

## 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))
fitDist

Fitting Different Parametric `gamlss.family` Distributions.

Description

The function `fitDist()` is using the function `gamlssML()` to fit all relevant parametric `gamlss.family` distributions, specified by the argument `type`, to a single data vector (with no explanatory variables). The final marginal distribution is the one selected by the generalised Akaike information criterion with penalty `k`. The default is `k=2` i.e AIC.

The function `fitDistPred()` is using the function `gamlssMLpred()` to fit all relevant (marginal) parametric `gamlss.family` distributions to a single data vector (similar to `fitDist()`) but the final model is selected by the minimum prediction global deviance. The user has to specify the training and validation/test samples.

The function `chooseDist()` is using the function `update.gamlss()` to fit all relevant parametric (conditional) `gamlss.family` distributions to a given fitted `gamlss` model. The output of the function is a matrix with rows the different distributions (from the argument `type`) and columns the different GAIC's (`k`). The default argument for `k` are 2, for AIC, 3.84, for Chi square, and log(n) for BIC. No final model is given by the function like for example in `fitDist()`. The function
getOrder() can be used to rank the columns of the resulting table (matrix). The final model can be refitted using update(), see the examples.

Usage

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

```
fitDistPred(y,
    type = c("realAll", "realline", "realplus", "real0to1", "counts", "binom"),
    try.gamlss = FALSE, extra = NULL, data = NULL, rand = NULL,
    newdata = NULL, trace = FALSE, ...)
```

```
chooseDist(object, k = c(2, 3.84, round(log(length(object$y)), 2)), type =
    c("realAll", "realline", "realplus", "real0to1", "counts", "binom", "extra"),
    extra = NULL, trace = FALSE,
    parallel = c("no", "multicore", "snow"), ncpus = 1L, cl = NULL, ...)
```

```
chooseDistPred(object,
    type = c("realAll", "realline", "realplus", "real0to1", "counts", "binom", "extra"),
    extra = NULL, trace = FALSE,
    parallel = c("no", "multicore", "snow"),
    ncpus = 1L, cl = NULL, newdata = NULL, rand = NULL, ...)
```

```
getOrder(obj, column = 1)
```

Arguments

- **y**: the data vector
- **object, obj**: a GAMLSS fitted model
- **k**: the penalty for the GAIC with default values $k=2$ the standard AIC. In the case of the function chooseDist() $k$ can be a vector i.e. $k = c(2,4,6)$ so more than one GAIC are saved.
- **type**: the type of distribution to be tried see details
- **try.gamlss**: this applies to functions fitDist() and fitDistPred(). It allows if gamlssML() fail to fit the model to try gamlss instead. This will slow up things for big data.
- **extra**: whether extra distributions should be tried, which are not in the type list. Note that the function chooseDist() allows the fitting of only the 'extra' distributions. This can be achieved if extra is set i.e. `extra = c("GA", "IG", "GG")` and type is set to extra i.e. `type = "extra"`.
- **data**: the data frame where $y$ can be found, only for functions fitDist() and fitDistPred()
- **rand**: For fitDistPred() a factor with values 1 (for fitting) and 2 (for predicting).
- **newdata**: The prediction data set (validation or test).
- **trace**: whether to print during fitting. Note that when parallel is 'multicore' or "snow" "trace" is not produce any output.
- **parallel**: The type of parallel operation to be used (if any). If missing, the default is "no".
ncpus integer: number of processes to be used in parallel operation: typically one would chose this to the number of available CPUs.

c1 This is useful for snow clusters, i.e. parallel = "snow", when the clusters are created in advance. If not supplied, a cluster on the local machine is created for the duration of the call.

column which column of the output matrix to be ordered according to best GAIC

... 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 and the real positive line i.e. realplus
- real0to1: The gamlss.family continuous distributions from 0 to 1: "BE", "BEo", "BE-INF0", "BEINF1", "BEOI", "BEZI", "BEINF", "GB1"
- binom: The gamlss.family distributions for binomial type data: "BI", "BB", "DB", "ZIBI", "ZIBB", "ZABI", "ZABB"

The function fitDist() uses the function gamlssML() to fit the different models, the function fitDistPred() uses gamlssMLpred() and the function chooseDist() used update.gamlss().

Value

For the functions fitDist() and fitDistPred() a gamlssML object is return (the one which minimised the GAIC or VDEV respectively) 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

For the function chooseDist() a matrix is returned, with rows the different distributions and columns the different GAIC’s set by k.

Author(s)

Mikis Stasinopoulos <d.stasinopoulos@londonmet.ac.uk>, Bob Rigby, Vlasis Voudouris and Majid Djennad.
References


(see also http://www.gamlss.com/).

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:
#---------------------------------------
# Example of using the argument extra
data(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="real0to1", trace=T,
extra=c("GAtr", "LOGN0tr", "TFtr"),
data=tensile)
ma$fits
ma$failed
#-------------------------------------
# selecting model using the prediction global deviance
# Using fitDistPred
# creating training data
y <- rt(1000, df=2)
m1 <- fitDist(y, type="realline")
m1$fits
m1$fails
# create validation data
yn <- rt(1000, df=2)
# choose distribution which fits the new data best
p1 <- fitDistPred(y, type="realline", newdata=yn)
p1$fits
p1$failed
# using the function chooseDist()
```

```r
# fitting normal distribution model
m1 <- gamlss(y~pb(x), sigma.fo=~pb(x), family=NO, data=abdom)
# choose a distribution on the real line
# and save GAIC(k=c(2,4,6.4), i.e. AIC, Chi-square and BIC.
t1 <- chooseDist(m1, type="realline", parallel="snow", ncpus=4)
# the GAIC's
# the distributions which failed are with NA's
# ordering according to BIC
gOrder(t1,3)
fm<-update(m1, family=names(getOrder(t1,3)[1]))
```

---

**fitted.gamlss**

*Extract Fitted Values For A GAMLSS Model*

**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.gamlls() but allows the argument what not to be character

**Usage**

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

**Arguments**

- `object` a GAMLSS fitted model
- `obj` a GAMLSS fitted model
- `what` which parameter fitted values are required, default what="mu"
- `parameter` equivalent to what
- `...` 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 http://www.gamlss.com/).

See Also

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

Examples

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

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
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 and Calliope Akantziliotou

References


(see also http://www.gamlss.com/).

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=BCT, 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"),
         parameter = NULL, ... )
Arguments

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

Value

Returns a model formula

Author(s)

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

References


(see also [http://www.gamlss.com/](http://www.gamlss.com/)).

See Also

- `gamlss`, `deviance.gamlss`, `fitted.gamlss`

Examples

```r
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

```r
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.fo=cs(x,df=5)`.

- `nu.formula`: a formula object for fitting a model to the nu parameter, e.g. `nu.fo=cs(x,df=5)`.

- `tau.formula`: a formula object for fitting a model to the tau parameter, e.g. `tau.fo=cs(x,df=5)`.

- `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 starting 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 glim.control function

... for extra arguments

x an object

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.

is.gamlss is a short version is is(object,"gamlss")

**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
- `rqres` 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, Calliope Akantziliotou and Vlasios Voudouris <vlasios.voudouris@abm-analytics.com>

References


(see also http://www.gamlss.com/).

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)

---

**gamlss.control**  
*Auxiliary for Controlling GAMLSS Fitting*

**Description**

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

**Usage**

```r
gamlss.control(c.crit = 0.001, n.cyc = 20, mu.step = 1, sigma.step = 1, nu.step = 1,
               tau.step = 1, gd.tol = Inf, 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 (set more recently to Inf to allow the algorithm to
  converged even if the global deviance change dramatically during the iterations)
- `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`  
  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 though 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 \( \text{cat} \) 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

References


(see also \url{http://www.gamlss.com/}).

See Also

\texttt{gamlss}

Examples

\begin{verbatim}
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)
\end{verbatim}
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

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

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

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**: the trace of the smoothing matrix
- **nl.df**: 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
References


(see also [http://www.gamlss.com/](http://www.gamlss.com/)).

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
coefSmo with value NULL
lambda the value of span
Author(s)

Mikis Stasinopoulos based on Brian Ripley implementation of loess function in \texttt{R}

See Also

gamlss, lo

---

Support for Functions for smoothers

Description

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

Usage

\begin{verbatim}
gamlss.pb(x, y, w, xeval = NULL, ...)  
gamlss.pbo(x, y, w, xeval = NULL, ...)  
gamlss.ps(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, ...)  
gamlss.pbz(x, y, w, xeval = NULL, ...)  
gamlss.pbc(x, y, w, xeval = NULL, ...)  
gamlss.pbp(x, y, w, xeval = NULL, ...)
\end{verbatim}

Arguments

\begin{description}
\item[x] the \texttt{x} for function \texttt{gamlss.fp} is referred to the design matric of the specific parameter model (not to be used by the user)
\item[y] the \texttt{y} for function \texttt{gamlss.fp} is referred to the working variable of the specific parameter model (not to be used by the user)
\item[w] the \texttt{w} for function \texttt{gamlss.fp} is referred to the iterative weight variable of the specific parameter model (not to be used by the user)
\item[xeval] used in prediction
\item[...] further arguments passed to or from other methods.
\end{description}

Value

All function return fitted smoothers.

Author(s)

Mikis Stasinopoulos <d.stasinopoulos@londonmet.ac.uk>, Bob Rigby
gamlss.random

References


(see also [http://www.gamlss.com/](http://www.gamlss.com/)).

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

gamlss.random(x, y, w, xeval = NULL, ...)  
gamlss.re(x, y, w, xeval = NULL, ...)

Arguments

x  the explanatory design matrix
y  the response variable
w  iterative weights
xeval  it used internally for prediction
...  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
coeffsma  with value NULL
gamlss.scope

Author(s)
Mikis Stasinopoulos, based on Trevor Hastie function gam.random

References
(see also http://www.gamlss.com/).

See Also
gamlss, random

gamlss.scope Generate a Scope Argument for Stepwise GAMLSS

Description
Generates a scope argument for a stepwise GAMLSS.

Usage
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 http://www.gamlss.com/).

See Also

stepGAIC

Examples

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

The function gamlssML() fits a gamlss.family distribution to single data set using a non linear maximisation algorithm in R. This is relevant only when explanatory variables do not exist.

The function gamlssMLpred() is similar to gamlssML() but it saves the predictive global deviance for the newdata. The newdata in gamlssMLpred() can be given with the arguments newdata or defining the factor rand. rand should be a binary factor rand splitting the original data set into a training set (value 1) and a validation/test set (values 2), see also gamlssVDG.
Usage

gamlssML(formula, 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 = sys.parent(), start.from = NULL, ...) 

gamlssMLpred(response = NULL, data = NULL, family = NO, 
  rand = NULL, newdata = NULL, ...)

Arguments

formula, response
  a vector of data requiring the fit of a gamlss.family distribution or (only for the 
  function gamlssML) a formula, for example, y~1, with no explanatory variables 
  because they are ignored).

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

weights
  a vector of weights. Here weights can be used to weight out observations (like in 
  subset) or for a weighted likelihood analysis where the contribution of the ob-
  servations 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 say w use weights=w

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

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

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

tau.start
  scalar of initial values for the parameter tau e.g. tau.start=3

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

data
  a data frame containing the variable y. If this is missing, the variable should be 
  on the search list. e.g. data=aids

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

rand
  For gamlssMLpred() a factor with values 1 (for fitting) and 2 (for predicting).

newdata
  The prediction data set (validation or test).

... 
  for extra arguments
Details

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

The function `gamlssMLpred()` uses the function `gamlssML()` to fit the model but then uses `predict.gamlssML()` to predict for newdata and saves the prediction i) deviance increments, ii) global deviance iii) residuals.

Value

Returns a `gamlssML` object which behaves like a `gamlss` fitted objected

Author(s)

Mikis Stasinopoulos <d.stasinopoulos@londonmet.ac.uk>, Bob Rigby, Vlasis Voudouris and Majid Djennd

References


(see also [http://www.gamlss.com/](http://www.gamlss.com/)).

See Also

`gamlss.family`, `gamlss`

Examples

```r
#-------- 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(m11<-gamlssML(y, family=NBI))
AIC(m1,m1a,m11, k=0)
# neg. binomial  n=10000
y<- rNBI(10000)
  system.time(m1<-gamlss(y~1, family=NBI))
  system.time(m1a<-gamlss(y~1, family=NBI, trace=FALSE))
  system.time(m11<-gamlssML(y, family=NBI))
AIC(m1,m1a,m11, k=0)
# binomial type data
data(aep)
```
m1 <- gamlssML(aep$y, family=BB) # ok
m2 <- gamlssML(y, data=aep, family=BB) # ok
m3 <- gamlssML(y~1, data=aep, family=BB) # ok
m4 <- gamlssML(aep$y~1, family=BB) # ok
AIC(m1,m2,m3,m4)

# Not run:
#-----------------------------------------------------------
# neg. binomial n=10000
y<- rNBI(10000)
rand <- sample(2, length(y), replace=TRUE, prob=c(0.6,0.4))
table(rand)
Y <- subset(y, rand==1)
YVal <- subset(y, rand==2)
length(Y)
length(YVal)
da1 <- data.frame(y=y)
dim(da1)
da2 <- data.frame(y=Y)
dim(da2)
danew <- data.frame(y=YVal)
# using gamlssVGD to fit the models
g1 <- gamlssVGD(y~1, rand=rand, family=NBI, data=da1)
g2 <- gamlssVGD(y~1, family=NBI, data=da2, newdata=dan)
AIC(g1,g2)
VGD(g1,g2)
# using gamlssMLpred to fit the models
p1 <- gamlssMLpred(y, rand=rand, family=NBI)
p2 <- gamlssMLpred(Y, family=NBI, newdata=YVal)
# AIC and VGD should produce identical results
AIC(p1,p2,g1,g2)
VGD(p1,p2, g1,g2)
# the fitted residuals
wp(p1, ylim.all=1)
# the prediction residuals
wp(resid=p1$residVal, ylim.all=.5)
#-------------------------------------------------------------
# chossing between distributions
p2<-gamlssMLpred(y, rand=rand, family=PO)
p3<-gamlssMLpred(y, rand=rand, family=PIG)
p4<-gamlssMLpred(y, rand=rand, family=BNB)
AIC(p1, p2, p3, p4)
VGD(p1, p2, p3, p4)

## End(Not run)
Description

This is a set of function useful for selecting appropriate models.

The functions gamlssVGD, VGD, getTGD, TGD can be used when a subset of the data is used for validation or testing.

The function stepVGD() is a stepwise procedure for selecting an appropriate model for any of the parameters of the model minimising the test global deviance. The function stepVGDAll.A() can select a model using strategy A for all the parameters.

The functions gamlssCV, CV can be used for a k-fold cross validation.

Usage

```r
# gamlssVGD
form = NULL, sigma.formula = ~1, nu.formula = ~1,
      tau.formula = ~1, data = NULL, family = NO,
      control = gamlss.control(trace = FALSE),
      rand = NULL, newdata = NULL, ...)

# VGD
object, ...)

# getTGD
object, newdata = NULL, ...)

# TGD
object, ...)

# gamlssCV
form = NULL, sigma.formula = ~1, nu.formula = ~1,
      tau.formula = ~1, data = NULL, family = NO,
      control = gamlss.control(trace = FALSE),
      K.fold = 10, set.seed = 123, rand = NULL,
      parallel = c("no", "multicore", "snow"),
      ncpus = 1L, cl = NULL, ...)

# CV
object, ...)

# drop1TGD
object, scope, newdata, parameter = c("mu", "sigma", "nu", "tau"),
      sorted = FALSE, trace = FALSE,
      parallel = c("no", "multicore", "snow"),
      ncpus = 1L, cl = NULL, ...)

# add1TGD
object, scope, newdata, parameter = c("mu", "sigma", "nu", "tau"),
      sorted = FALSE, trace = FALSE,
      parallel = c("no", "multicore", "snow"),
      ncpus = 1L, cl = NULL, ...)

# stepTGD
object, scope, newdata,
      direction = c("both", "backward", "forward"),
      trace = TRUE, keep = NULL, steps = 1000,
      parameter = c("mu", "sigma", "nu", "tau"),
      parallel = c("no", "multicore", "snow"),
      ncpus = 1L, cl = NULL, ...)
```
stepTGDAll.A(object, scope = NULL, newdata = NULL, steps = 1000, sigma.scope = NULL, nu.scope = NULL, tau.scope = NULL, mu.try = TRUE, sigma.try = TRUE, nu.try = TRUE, tau.try = TRUE, parallel = c("no", "multicore", "snow"), ncpus = 1L, cl = NULL, ...)

Arguments

formula A gamlss mu formula.
sigma.formula Formula for sigma.
nu.formula Formula for nu.
tau.formula Formula for tau.
data The data frame required for the fit.
family The gamlss.family distribution.
control The control for fitting the gamlss model.
rand For gamlssVGD a variable with values 1 (for fitting) and 2 (for predicting). For gamlssCV a variable with k values indicating the cross validation sets.
newdata The new data set (validation or test) for prediction.
object A relevant R object.
scope defines the range of models examined in the stepwise selection similar to stepGAIC() where you can see examples
sigma.scope defines the range of models examined in the stepwise selection for sigma
nu.scope defines the range of models examined in the stepwise selection for nu
tau.scope defines the range of models examined in the stepwise selection for tau
mu.try whether should try fitting models for mu
sigma.try whether should try fitting models for sigma
nu.try whether should try fitting models for nu
tau.try whether should try fitting models for tau
parameter which distribution parameter is required, default what="mu"
sorted should the results be sorted on the value of TGD
trace if TRUE additional information may be given on the fits as they are tried.
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
keep see stepGAIC() for explanation
steps the maximum number of steps to be considered. The default is 1000.
K.fold the number of subsets of the data used
set.seed the seed to be used in creating rand
parallel The type of parallel operation to be used (if any). If missing, the default is "no".
ncpus integer: number of processes to be used in parallel operation: typically one would chose this to the number of available CPUs.
c1 An optional parallel or snow cluster for use if parallel = "snow". If not supplied, a cluster on the local machine is created for the duration of the call.

Details
The function gamlssVGD() fits a gamlss model to the training data set determined by the arguments rand or newdata. The results is a gamlssVGD objects which contains the gamlss fit to the training data plus three extra components: i) VGD the global deviance applied to the validation data sets. ii) predictError which is VGD divided with the number of observations in the validation data set and iii) residVal the residuals for the validation data set.

The function VGD() extract the validated global deviance from one or more fitted gamlssVGD objects and can be used foe model comparison.

The function getTGD() operates different from the function gamlssVGD(). It assumes that the users already have fitted models using gamlss() and now he/she wants to evaluate the global deviance at a new (validation or test) data set.

The function TGD() extract the validated/test global deviance from one or more fitted gamlssTGD objects and can be use to compare models.

The gamlssCV() performs a k-fold cross validation on a gamlss models.

The function CV() extract the cross validated global deviance from one or more fitted gamlssCV objects and can be use to compare models.

The functions add1TGD(), drop1TGD() and stepTGD() behave similar to add1(), drop1() and stepGAIC() functions respectively but they used validation or test deviance as the selection criterion rather than the GAIC.

Value
A fitted models of a set of global deviances.

Author(s)
Mikis Stasinopoulos

References


See Also

stepGAIC

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
olddata<-abdom[rand==1,] # training data
newdata<-abdom[rand==2,] # validation data
#---------------------------------------
# gamlssVGD
#---------------------------------------
# Using rand
v1 <- gamlssVGD(y~pb(x,df=2),sigma.formula=~pb(x,df=1), data=abdom, family=NO, rand=rand)
v2 <- gamlssVGD(y~pb(x,df=2),sigma.formula=~pb(x,df=1), data=abdom, family=LO, rand=rand)
v3 <- gamlssVGD(y~pb(x,df=2),sigma.formula=~pb(x,df=1), data=abdom, family=TF, rand=rand)
VGD(v1,v2,v3)
#-------------------------------------------------------------------------------------------
## Not run:
#-------------------------------------------------------------------------------------------
# using two data set
v11 <- gamlssVGD(y~pb(x,df=2),sigma.formula=~pb(x,df=1), data=olddata, family=NO, newdata=newdata)
v12 <- gamlssVGD(y~pb(x,df=2),sigma.formula=~pb(x,df=1), data=olddata, family=LO, newdata=newdata)
v13 <- gamlssVGD(y~pb(x,df=2),sigma.formula=~pb(x,df=1), data=olddata, family=TF, newdata=newdata)
VGD(v11,v12,v13)
#-------------------------------------------------------------------------------------------
# function getTGD
#-------------------------------------------------------------------------------------------
# fit gamlss models first
g1 <- gamlss(y~pb(x,df=2),sigma.formula=~pb(x,df=1), data=olddata, family=NO)
g2 <- gamlss(y~pb(x,df=2),sigma.formula=~pb(x,df=1), data=olddata, family=LO)
g3 <- gamlss(y~pb(x,df=2),sigma.formula=~pb(x,df=1), data=olddata, family=TF)
# and then use
gg1 <-getTGD(g1, newdata=newdata)
gg2 <-getTGD(g2, newdata=newdata)
gg3 <-getTGD(g3, newdata=newdata)
TGD(gg1, gg2, gg3)
# function gamlssCV
set.seed(123)
rand1 <- sample (10 , 610, replace=TRUE)
g1 <- gamlssCV(y~pb(x,df=2), sigma.formula=~pb(x,df=1), data=abdom, family=NO, rand=rand1)
g2 <- gamlssCV(y~pb(x,df=2), sigma.formula=~pb(x,df=1), data=abdom, family=LO, rand=rand1)
g3 <- gamlssCV(y~pb(x,df=2), sigma.formula=~pb(x,df=1), data=abdom, family=TF, rand=rand1)
CV(g1,g2,g3)
CV(g1)
# using parallel
set.seed(123)
rand1 <- sample (10 , 610, replace=TRUE)
nc <- detectCores()
system.time(g21 <- gamlssCV(y~pb(x,df=2), sigma.formula=~pb(x,df=1), data=abdom, family=NO, rand=rand1, parallel = "no", ncpus = nc ))

system.time(g22 <- gamlssCV(y~pb(x,df=2), sigma.formula=~pb(x,df=1), data=abdom, family=LO, rand=rand1, parallel = "multicore", ncpus = nc ))

system.time(g23 <- gamlssCV(y~pb(x,df=2), sigma.formula=~pb(x,df=1), data=abdom, family=TF, rand=rand1, parallel = "snow", ncpus = nC ))

CV(g21,g22,g23)
# functions add1TGD() drop1TGD() and stepTGD()
# the data
data(rent)
rand <- sample(2, dim(rent)[1], replace=TRUE, prob=c(0.6,0.4))
# the proportions in the sample
table(rand)/dim(rent)[1]
oldrent<-rent[rand==1,] # training set
newrent<-rent[rand==2,] # validation set

# null model
v0 <- gamlss(R~1, data=oldrent, family=GA)
# complete model
v1 <- gamlss(R~pb(Fl)+pb(A)+H+loc, sigma.fo=~pb(Fl)+pb(A)+H+loc, data=oldrent, family=GA)

# drop1TGD
system.time(v3<- drop1TGD(v1, newdata=newrent, parallel="no"))
system.time(v4<- drop1TGD(v1, newdata=newrent, parallel="multicore", ncpus=nC ) )
system.time(v5<- drop1TGD(v1, newdata=newrent, parallel="snow", ncpus=nC))
gen.likelihood

A function to generate the likelihood function from a GAMLSS object

description

This function generate a function with argument the parameters of the GAMLSS model which can evaluate the log-likelihood function.

usage

gen.likelihood(object)

arguments

object A gamlss fitted model

details

The purpose of this function is to help the function vcov() to get he 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 and Vlasios Voudouris
References


(see also \url{http://www.gamlss.com/}).

See Also

\texttt{vcov}

Examples

```r
data(aids)
ml <- gamlss(y~x+qrt, data=aids, family=NBI)
logl<-gen.likelihood(ml)
logL()
logLik(ml)
```

---

\texttt{getPEF} \hspace{1cm} \textit{Getting the partial effect function from a continuous term in a GAMLSS model}

Description

This function can be used to calculate the partial effect and the elasticity of a continuous explanatory variable \( x \).

By ‘partial effect’ function we mean how \( x \) is influence the parameter of interest given that the rest of explanatory terms for this parameter are on (specified) fixed values.

The function takes a GAMLSS object and for the range of the continuous variable \( x \), (by fixing the rest of the explanatory terms at specified values), calculates the effect that \( x \) has on the specific distribution parameter (or its predictor). The resulting function shows the effect that \( x \) has on the distribution parameter. The partial effect function which is calculated on a finite grit is then approximated using the \texttt{splinefun()} in R and its is saved.

The saved function can be used to calculate the elasticity of \( x \). The elasticity is the first derivative of the partial effect function and shows the chance of the parameter of interest for a small change in \( x \), by fixing the rest of the explanatory variables at specified values.

Usage

```
getPEF(obj = NULL, term = NULL, data = NULL, n.points = 100,
       parameter = c("mu", "sigma", "nu", "tau"),
       type = c("response", "link"), how = c("median", "last"),
       fixed.at = list(), plot = FALSE)
```
getPEF

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>obj</code></td>
<td>A gamlss object</td>
</tr>
<tr>
<td><code>term</code></td>
<td>the continuous explanatory variable</td>
</tr>
<tr>
<td><code>data</code></td>
<td>the data.frame (not needed if is declared on <code>obj</code>)</td>
</tr>
<tr>
<td><code>n.points</code></td>
<td>the number of points in which the influence function for x need to be evaluated</td>
</tr>
<tr>
<td><code>parameter</code></td>
<td>which distribution parameter</td>
</tr>
<tr>
<td><code>type</code></td>
<td>whether against the parameter, &quot;response&quot;, or the predictor &quot;link&quot;</td>
</tr>
<tr>
<td><code>how</code></td>
<td>whether for continuous variables should use the median or the last observation in the data</td>
</tr>
<tr>
<td><code>fixed.at</code></td>
<td>a list indicating at which values the rest of the explanatory terms should be fixed</td>
</tr>
<tr>
<td><code>plot</code></td>
<td>whether to the plot the influence function and its first derivatives</td>
</tr>
</tbody>
</table>

Value

A function is created which can be used to evaluate the partial effect function at different values of x.

Author(s)

Mikis Stasinopoulos, Vlasios Voudouris, Daniil Kiose

References


(see also [http://www.gamlss.com/](http://www.gamlss.com/)).

See Also

[gamlss](http://www.gamlss.com/)

Examples

```r
m1 <- gamlss(R~pb(Fl)+pb(A), data=rent, family=GA)
# getting the Partial Efect function
pef <- getPEF(obj=m1,term="A", plot=TRUE)
# the value at 1980
pef(1980)
# the first derivative at 1980
```
getQuantile

Description

This function can be used to calculate the partial effect that an explanatory variable has on a specific quantile.

By ‘partial effect’ function we mean how the term influence the quantile given that the rest of explanatory terms are constant.

The function takes a GAMLSS object and for the range of a specified explanatory (by fixing the rest of the terms at specified values), calculates the effect that this term has on the a quantile of the distribution. That is, it shows the effect that the particular term has on the quantile. The ‘partial’ quantile is calculated on a finite grid of values and then the function is approximated (using the splinefun()) and saved.

The saved function can be used to calculate the first derivative. This first derivatives shows the chance of the quantile function for a small change in the explanatory variable, by fixing the rest of the explanatory variables at a constant values.

Usage

getQuantile(obj = NULL, term = NULL, quantile = 0.9, data = NULL, n.points = 100, how = c("median", "last"), fixed.at = list(), plot = FALSE)

Arguments

obj          A gamlss object
term         an explanatory variable (at the moment works with with continuous)
quantile     the required quantile of the distribution
data          the data.frame (not needed if is declared on obj)
n.points      the number of points in which the quantile function needs evaluation
how           whether for extra continuous explanatory variables should fixed at the median or the last observation in the data
fixed.at      a list indicating at which values the rest of the explanatory terms should be fixed
plot          whether to the plot the partial quantile function and its first derivatives
Details

The function `getQuantile()` relies on the `predictAll()` function to evaluate the distribution parameters at a grid (default 100 points) of the specified term (given that the the rest of the terms are fixed). Then the inverse cdf is used to calculate the partial quantile. The function then is approximated using `splinefun()` and saved.

Value

A function is created which can be used to evaluate the partial effect of the explanatory variable on a specified quantile.

Author(s)

Mikis Stasinopoulos

References


(see also http://www.gamlss.com/).

See Also

gamlss, getPEF

Examples

```r
library(gamlss)
data(rent)
m1 <- gamlss(R~pb(Fl)+pb(A)+B+loc, data=rent, family=GA)
FF<-getQuantile(m1, quantile=0.9, term="A", plot=TRUE)
FF(1960)
FF(1060, deriv=1)
FF(1060, deriv=2)
## Not run:
# plotting partial quantile
# .05, 0.25, 0.5, 0.75, 0.95
# at the default values
# Fl = median(Fl), B=0, and loc=2
plot(R~A, data=rent, col="lightgray", pch=20)
for (i in c(.05, 0.25, 0.5, 0.75, 0.95))
{
  Qua <- getQuantile(m1, quantile=i,term="A")
  curve(Qua, 1900, 1985, lwd=1, lty=1, add=T)
}
```
# plotting at values Fl=60, B=1, and loc=1.
for (i in c(.05, 0.25, 0.5, 0.75, 0.95))
{
Qua <- getQuantile(m1, quantile=i, term="A",
                   fixed.at=list(Fl=60, B=1, loc=1))
curve(Qua, 1900, 1985, lwd=1, lty=2, col="red", add=T)
}
# plotting at Fl=60, B=1 and loc=1.
for (i in c(.05, 0.25, 0.5, 0.75, 0.95))
{
Qua <- getQuantile(m1, quantile=i, term="A",
                   fixed.at=list(Fl=120, B=0, loc=3))
curve(Qua, 1900, 1985, lwd=1, lty=3, col="blue", add=T)
}

## End(Not run)

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"),
        parameter = NULL, which = 1)
```

**Arguments**

- `object`: a GAMLSS fitted model
- `what`: which distribution parameter is required, default what="mu"
- `parameter`: equivalent to what
- `which`: which smoothing term i.e. 1, 2 etc. Note that 0 means all.

**Details**

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

**Value**

A list containing information about a fitted smoother or a fitted objects
Author(s)
Mikis Stasinopoulos and Bob Rigby

References


(see also [http://www.gamlss.com/](http://www.gamlss.com/)).

Examples
```r
data(usair)
t1<-gamlss(y~x1+pb(x5)+pb(x6), data=usair, family=GA)
# get the value for lambda for the second fitted term in mu
getSmo(t1, parameter="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

References


(see also [http://www.gamlss.com/](http://www.gamlss.com/)).

See Also

gamlss

Examples

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

```r
histDist(y, family = NO, freq = NULL, density = FALSE, nbins = 10, xlim = NULL, ylim = NULL, main = NULL, xlab = NULL, ylab = NULL, data = NULL, col.hist = "gray", border.hist = "blue", fg.hist = rainbow(12)[9], line.wd = 2, line.ty = c(1, 2), line.col = c(2, 3),
```

**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 then 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, col.hist = "gray", border.hist = "blue", fg.hist = rainbow(12)[9], line.wd = 2, line.ty = c(1, 2), line.col = c(2, 3),
```
histDist

```r
col.main = "blue4", col.lab = "blue4",
col.axis = "blue", ...)
```

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>y</code></td>
<td>a vector for the response variable</td>
</tr>
<tr>
<td><code>family</code></td>
<td>a <code>gamlss.family</code> distribution</td>
</tr>
<tr>
<td><code>freq</code></td>
<td>the frequencies of the data in <code>y</code> if exist. <code>freq</code> is used as weights in the <code>gamlss</code> fit</td>
</tr>
<tr>
<td><code>density</code></td>
<td>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)</td>
</tr>
<tr>
<td><code>nbins</code></td>
<td>The suggested number of bins (argument passed to <code>truehist()</code> of package MASS). Either a positive integer, or a character string naming a rule: &quot;Scott&quot; or &quot;Freedman-Diaconis&quot; or &quot;FD&quot;. (Case is ignored.)</td>
</tr>
<tr>
<td><code>xlim</code></td>
<td>the minimum and the maximum x-axis value (if the default values are out of range)</td>
</tr>
<tr>
<td><code>ylim</code></td>
<td>the minimum and the maximum y-axis value (if the default values are out of range)</td>
</tr>
<tr>
<td><code>main</code></td>
<td>the main title for the plot</td>
</tr>
<tr>
<td><code>xlab</code></td>
<td>the label in the x-axis</td>
</tr>
<tr>
<td><code>ylab</code></td>
<td>the label in the y-axis</td>
</tr>
<tr>
<td><code>data</code></td>
<td>the data.frame</td>
</tr>
<tr>
<td><code>col.hist</code></td>
<td>the colour of the histogram or barplot</td>
</tr>
<tr>
<td><code>border.hist</code></td>
<td>the colour of the border of the histogram or barplot</td>
</tr>
<tr>
<td><code>fg.hist</code></td>
<td>the colour of axis in the histogram or barplot</td>
</tr>
<tr>
<td><code>line.wd</code></td>
<td>the line width of the fitted distribution</td>
</tr>
<tr>
<td><code>line.ty</code></td>
<td>the line type of the fitted distribution</td>
</tr>
<tr>
<td><code>line.col</code></td>
<td>the line color of the fitted distribution</td>
</tr>
<tr>
<td><code>col.main</code></td>
<td>the colour for the main title</td>
</tr>
<tr>
<td><code>col.lab</code></td>
<td>the colour of the labels</td>
</tr>
<tr>
<td><code>col.axis</code></td>
<td>the color of the axis</td>
</tr>
<tr>
<td><code>...</code></td>
<td>for extra arguments to be passed to the <code>gamlss</code> function</td>
</tr>
</tbody>
</table>

**Details**

This function first fits constants for each parameters of a GAMLSS distribution family using the `gamlss` function and then 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


(see also http://www.gamlss.com/).

See Also

gamlss, gamlss.family

Examples

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, line.col="blue")
# discere data counts
# Hand at al. p150 Leptinotarsa decemlineata
y <- c(0,1,2,3,4,6,7,8,10,11)
freq <- c(33,12,5,6,5,2,2,2,1,2)
histDist(y, "NBI", freq=freq)
# the same as
histDist(rep(y,freq), "NBI")

Description

Density estimation using the Poisson trick

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.
histSmo

Usage

histSmo(y, lambda = NULL, df = NULL, order = 3, lower = NULL, upper = NULL, type = c("freq", "prob"), plot = FALSE, breaks = NULL, discrete = FALSE, ...)

histSmoC(y, df = 10, lower = NULL, upper = NULL, type = c("freq", "prob"), plot = FALSE, breaks = NULL, discrete = FALSE, ...)

histSmoO(y, lambda = 1, order = 3, lower = NULL, upper = NULL, type = c("freq", "prob"), plot = FALSE, breaks = NULL, discrete = FALSE, ...)

histSmoP(y, lambda = NULL, df = NULL, order = 3, lower = NULL, upper = NULL, type = c("freq", "prob"), plot = FALSE, 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
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 Poison 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 [http://www.gamlss.com/](http://www.gamlss.com/)).

See Also

- pb, cs

Examples

```r
# creating data from Pareto 2 distribution
set.seed(153)
Y <- rPARETO2(1000)
## Not run:
# getting the density
histSmo(Y, lower=0, plot=TRUE)
# more breaks a bit slower
histSmo(Y, breaks=200, lower=0, plot=TRUE)
```
# quick fit using lambda
def histSmoO(Y, lambda=1, breaks=200, lower=0, plot=TRUE)
# or
histSmo(Y, lambda=1, breaks=200, lower=0, plot=TRUE)
# quick fit using df
def histSmoC(Y, df=15, breaks=200, lower=0, plot=TRUE)
# or
histSmo(Y, df=15, breaks=200, lower=0)
# saving results
m1 <- histSmo(Y, lower=0, plot=T)
plot(m1)
plot(m1, "cdf")
plot(m1, "invcdf")
# using with a histogram
library(MASS)
truehist(Y)
lines(m1, col="red")
#---------------------------
# now generate from SHASH distribution
YY <- rSHASH(1000)
m1 <- histSmo(YY)
# calculate P(YY>10)
1-m1$cdf(10)
# calculate P(-10<YY<10)
1-(1-m1$cdf(10))-m1$cdf(-10)
#---------------------------
# from discrete distribution
YYY <- rNBI(1000, mu=5, sigma=4)
histSmo(YYY, discrete=TRUE, plot=T)
#
YYY <- rPO(1000, mu=5)
histSmo(YYY, discrete=TRUE, plot=T)
#
YYY <- rNBI(1000, mu=5, sigma=.1)
histSmo(YYY, discrete=TRUE, plot=T)
# generating from beta distribution
YYY <- rBE(1000, mu=.1, sigma=.3)
histSmo(YYY, lower=0, upper=1, plot=T)
# from truncated data
Y <- with(stylo, rep(word, freq))
histSmo(Y, lower=1, discrete=TRUE, plot=T)
histSmo(Y, lower=1, discrete=TRUE, plot=T, type="prob")
## End(Not run)

IC

*IC*: Gives the GAIC for a GAMLSS Object

**Description**

The function `IC()` calculates 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 GAIC.table() produces a table for different models and different penalties, k.

The function extractAIC is a 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 )
GAIC.table(object, ..., k = c(2, 3.84, round(log(length(object$y)), 2)))
## 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 GAIC.table() returns a table which its rows showing different models and its columns different k’s. 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 http://www.gamlss.com/).

See Also
gamlss

Examples
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)
GAIC.table(mod1,mod2)
extractAIC(mod1,k=3)
rm(mod1,mod2)

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

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 = 0.01,
    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.family 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 <-c("NO","SHASH0") 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 and Vlasios Voudouris <vlasios.voudouris@abm-analytics.com>

References

(see also http://www.gamlss.com/).

See Also
gamlss, centiles, calibration

Examples

## 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)

# l0
Specify a loess fit in a GAMLSS formula

Description

Allows the user to specify a loess fit within a GAMLSS model. This function is similar to the l0 function in the gam implementation of package gam see Chambers and Hastie (1991).
The function vis.l0() allows plotting the results.
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", "none"),  
trace.hat = c("exact", "approximate"),  
cell = 0.2, iterations = 4, iterTrace = FALSE, ...)

vis.lo(obj, se=-1, rug = FALSE, partial.resid = FALSE,  
col.term = "darkred", col.shaded = "gray",  
col.res = "lightblue", col.rug = "gray", lwd.term = 1.5,  
cex.res = 1, pch.res = par("pch"),  
type = c("persp", "contour"), col.surface = "gray",  
nlevels = 30, n.grid = 30, image = TRUE, ...)

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
iterTrace logical (or integer) determining if tracing information during the robust iterations (iterations>= 2) is produced. See the help file of loess.control
obj an lowss object fitted within gamlss
se if se>0 then standard errors surfaces are drawn in the 3-dimentional plot. Set se at the required level i.e se=1.96 will be an approximated 95% CI.
rug whether to plot a rug in the plot
partial.resid whether to plot the partial residuals
col.term the colour of the line of fitted term
cex.res the shading of standard
col.shaded the shading of standard error intervals
col.res the colour of partial residuals
col.rug the colour of the rug
lwd.term the width of the line
pch.res The character for the partial residuals
type The type of the plot if the x’s are two dimensional
col.surface the colour of the fitted surface
nlevels the number of levels used in contour() plot.
n.grid The number of points to evaluate the surface
image whether to use image() or just contour

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, (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.)
loglogSurv

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 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 as an estimated for Type I tails (see Chapter 17 in Rigby et al. (2019) for definition of the different types of tails.)

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

References


(see also http://www.gamlss.com/).

See Also

cs, random,

Examples

# 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) #
term.plot(aids1, page=1)
## Not run:
r1 <- gamlss(R~lo(~Fl)+lo(~A), data=rent, family=GA)
term.plot(r1, pages=1)
vis.lo(getSmo(r1, which=1), partial=T)
r2 <- gamlss(R~lo(~Fl+A), data=rent, family=GA)
term.plot(r2, pages=1)
vis.lo(getSmo(r2, which=1))
vis.lo(getSmo(r2, which=1), se=1.97)
vis.lo(getSmo(r2, which=1), partial.res=T)
## End(Not run)
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 as an estimated for Type III tails.

The function loglogSurv() combines all the above functions.

The function logSurv() is design for exploring the heavy tails of a single response variable. It plots the empirical log-survival function of the right tail of the distribution or the empirical log-cdf function of the left tail against log(y) for a specified probability of the tail. Then fits a linear, a quadratic and an 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(). The function logSurv() plots but do not fit the curves.

The function loglogplot() plot the empirical log-survival function of all data against log(y). The function ECDF() calculates the empirical commutative distribution function. It is similar to ecdf() but divides by n+1 rather n, the number of conservations.

Usage

loglogSurv(y, prob = 0.9, print = TRUE, title = NULL, lcol = gray(0.1), ltype = 1, plot = TRUE, ...)

loglogSurv1(y, prob = 0.9, print = TRUE, title = NULL, lcol = gray(0.1), ltype = 1, ...)

loglogSurv2(y, prob = 0.9, print = TRUE, title = NULL, lcol = gray(0.1), ltype = 1, ...)

loglogSurv3(y, prob = 0.9, print = TRUE, title = NULL, lcol = gray(0.1), ltype = 1, ...)

logSurv(y, prob = 0.9, tail = c("right", "left"), plot = TRUE, lines = TRUE, print = TRUE, title = NULL, lcol = c(gray(0.1), gray(0.2), gray(0.3)), ltype = c(1, 2, 3), ...)

logSurv0(y, prob = 0.9, tail = c("right", "left"), plot = TRUE, title = NULL, ...)

ECDF(y)

loglogplot(y, nplus1 = TRUE, ...)

Arguments

y a vector, the variable of interest

prob what probability. The defaul is 0.90 which means 10% for "right" tail 90% for "left" tail
The functions `loglogSurv1()`, `loglogSurv3()` and `loglogSurv3()` take the upper 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. The function `loglogSurv()` fits all three models and displays the best.

The function `logSurv()` takes the upper (or lower) part of an ordered variable and plots the log empirical survival function against `log(y)`. Also display three curves i) linear ii) quadratic and iii) exponential to determine what kind of tail relationship exist. Plotting the log empirical survival function against `log(y)` often call in the literature the "log-log plot".

The function `loglogplot()` plots the whole log empirical survival function against `log(y)` (not just the tail). The function `ECDF()` calculate the step function of the empirical cumulative distribution function.

More details can be found in Chapter 17 of "Rigby et al. (2019) book an old version on which can be found in https://www.gamlss.com/")

The functions create plots.

Bob Rigby, Mikis Stasinopoulos and Vlassios Voudouris


Examples

data(film90)
y <- film90$lborev1
op<-par(mfrow=c(3,1))
loglogSurv1(y)
loglogSurv2(y)
loglogSurv3(y)
par(op)
loglogSurv(y)

logSurv(y)

loglogplot(y)

plot(ECDF(y), main="ECDF")

---

**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**

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

**Arguments**

- **obj**: a GAMLSS fitted model
- **what**: which distribution parameter is required, default what="mu"
- **parameter**: equivalent to what
`lpred`  

- **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

**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 [http://www.gamlss.com/](http://www.gamlss.com/)).

**See Also**

predict.gamlss

**Examples**

```r
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 models.

**Usage**

```r
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 [http://www.gamlss.com/](http://www.gamlss.com/)).

**See Also**

`gamlss`, `dropterm`
Examples

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

**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
- `parameter` equivalent to `what`
- `...` 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=PO, data=aids) #
model.frame(mod)
model.matrix(mod)
terms(mod, "mu")
rm(mod)
```

---

**numeric.deriv**

*An internal GAMLSS function for numerical derivatives*

**Description**

A function to calculate numerical derivatives.

**Usage**

```r
numeric.deriv(expr, theta, delta = NULL,
    rho = sys.frame(sys.parent()))
```

**Arguments**

- `expr` The expression to be differentiated
- `theta` A character vector
- `delta` constant for the accuracy
- `rho` environment

**Details**

This function is use by several GAMLSS functions but it is not for general use since there are more reliable function to do that in R.
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 ~ x | a indicates that plots of y versus x should be produced conditional on the variable a. A formula of the form y ~ x | a * 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 distinguish between the individual participants

color whether the parallel plot are shown in colour, color=TRUE (the default) or not

Value

A vector of numerical derivatives

Note

Do not use this function unless you know what you are doing

Author(s)

Mikis Stasinopoulos

References


(see also http://www.gamlss.com/).
par.plot

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 [http://www.gamlss.com/](http://www.gamlss.com/)).

See Also

gamlss

Examples

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

Reduction for the Levels of a Factor.

Description

The function is trying to merged similar levels of a given factor. Its based on ideas given by Tutz (2013).

Usage

```r
pcat(fac, df = NULL, lambda = NULL, method = c("ML", "GAIC"), start = 0.001,
Lp = 0, kappa = 1e-05, iter = 100, c.crit = 1e-04, k = 2)
gamlss.pcat(x, y, w, xeval = NULL, ...)
plotDF(y, factor = NULL, formula = NULL, data, along = seq(0, nlevels(factor)),
kappa = 1e-06, Lp = 0, ...)
plotLambda(y, factor = NULL, formula = NULL, data, along = seq(-2, 2, 0.1),
kappa = 1e-06, Lp = 0, ...)
```

Arguments

- `fac, factor`: a factor to reduce its levels
- `df`: the effective degrees of freedom df
- `lambda`: the smoothing parameter
- `method`: which method is used for the estimation of the smoothing parameter, "ML" or "GAIC" are allowed.
- `start`: starting value for lambda if it estimated using "ML" or "GAIC"
- `Lp`: The type of penalty required, Lp=0 is the default. Use Lp=1 for lasso type and different values for different required penalty.
- `kappa`: a regulation parameters used for the weights in the penalties.
- `iter`: the number of internal iteration allowed
- `c.crit`: the convergent criterion
- `k`: the penalty if "GAIC" method is used.
- `x`: explanatory factor
- `y`: the response or iterative response variable
- `w`: iterative weights
- `xeval`: indicator whether to predict
- `formula`: A formula
- `data`: A data frame
- `along`: a sequence of values
- `...`: for extra variables
Details

The `pcat()` is used for the fitting of the factor. The function shrinks the levels of the categorical factor (not towards the overall mean as the function `random()` is doing) but towards each other. This results to a reduction of the number if levels of the factors. Different norms can be used for the shrinkage by specifying the argument `Lp`.

Value

The function `pcat` reruns a vector 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()`. The backfitting is done in `gamlss.pcat`.

Note

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

Author(s)

Mikis Stasinopoulos <d.stasinopoulos@londonmet.ac.uk>, Paul Eilers and Marco Enea

References


(see also http://www.gamlss.com/).

See Also

`random`

Examples

```r
# Simulate data 1
n <- 10 # number of levels
m <- 200 # number of observations
set.seed(2016)
level <- as.factor(floor(runif(m) * n) + 1)
a0 <- rnorm(n)
sigma <- 0.4
mu <- a0[level]
y <- mu + sigma * rnorm(m)
plot(y~level)
points(1:10,a0, col="red")
da1 <- data.frame(y, level)
#------------------
mn <- gamlss(y~1, data=da1 ) # null model
ms <- gamlss(y~level-1, data=da1) # saturated model
```
m1 <- gamlss(y~pcat(level), data=da1)  # calculating lambda ML
AIC(mn, ms, m1)
## Not run:
m11 <- gamlss(y~pcat(level, method="GAIC", k=log(200)), data=da1)  # GAIC
AIC(mn, ms, m1, m11)
# getting the fitted object  -----------------------------------------------
getSmo(m1)
coef(getSmo(m1))
fitted(getSmo(m1))[[1:10]]
plot(getSmo(m1))  #
# After the fit a new factor is created this factor has the reduced levels
levels(getSmo(m1)$factor)
# --------------------------------------------------------------------------
## End(Not run)

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

df.plot(obj = NULL, obs = c(1), family = NO(), mu = NULL,
sigma = NULL, nu = NULL, tau = NULL, from = 0,
to = 10, min = NULL, max = NULL, no.points = 201,
no.title = FALSE, col = gray(0.4), y.axis.lim = 1.1,
frame.plot = TRUE, ...)

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 be evaluated e.g. tau=5
from
  Minimum value of the random variable y (identical to min)
to
  Maximum value of the random variable y (identical to max)
min
  Minimum value of the random variable y e.g. min=0
max
  Maximum value of y e.g. max=10
no.points
  the number of points in which the function will be evaluated
no.title
  Whether you need title in the plot, default is no.title=FALSE
col
  the color of the lines
y.axis.lim
  the limits for the y-axis
frame.plot
  whether to frame the individual plots
...
  for extra arguments, Note that usufull argument can be col.axis, col.lab, cex.axis, cex.lab etc.

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

References


(see also [http://www.gamlss.com/](http://www.gamlss.com/)).

**See Also**

`gamlss`

**Examples**

```r
pdf.plot(family=BCT, min=1, max=20, mu=10, sigma=0.15, nu=-1, tau=c(4,10,20,40) )
```

```r
## Not run:
# now using an gamlss object
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)
## End(Not run)
```

---

**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
- `ts`: set this to `TRUE` if ACF and PACF plots of the residuals are required
- `summaries`: set this to `FALSE` if no summary statistics of the residuals are required
- `...`: further arguments passed to or from other methods.
plot.gamlss

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 \texttt{ts=TRUE})
- residuals against an index or specified x-variable (or PACF of the residuals if \texttt{ts=TRUE})
- kernel density estimate of the residuals
- QQ-normal plot of the residuals

For time series response variables option \texttt{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 \texttt{summary=T}

Author(s)

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

References


(see also \url{http://www.gamlss.com/}).

See Also

gamlss

Examples

data(aids)
a<-gamlss(y~pb(x)+qrt,family=PO,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, Vlasis Voudouris and Majid Djennad

References

(see also http://www.gamlss.com/).

See Also

histSmo
Examples

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

plot2way(obj, terms = list(), what = c("mu", "sigma", "nu", "tau"),
         parameter = NULL, 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
parameter equivalent to what
show.legend whether to show the legend in the two way plot
... Further arguments

Details

This is an experimental function which should be used 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
References


(see also [http://www.gamlss.com/](http://www.gamlss.com/)).

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 [http://www.gamlss.com/](http://www.gamlss.com/)).


**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"),
parameter = NULL,
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, use.weights = FALSE,
data = NULL, y.value = "median",
set.to = .Machine$double.xmin,
output = c("list", "matrix"), ...)```

Arguments

object: a GAMLSS fitted model
what: which distribution parameter is required, default what="mu"
parameter: equivalent to what
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
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 if exist
use.weights: if use.weights=TRUE the old data and the newdata are merged and the model is refitted with weights equal to the prior weights for the old data observations and equal to a very small value (see option set.to) for the .newdata values. This trick allows to obtain standard errors for all parameters
data: the data frame used in the original fit if is not defined in the call
y.value: how to get the response values for the newdata if they do not exist. The default is taking the median, y.value="median". Other function like "max", "min" are allowed. Also numerical values.
set.to: what values the weights for the newdata should take
output: whether the output to be a 'list' (default) or a 'matrix'
...: 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^0.5). This could be fitted as mod<-gamlss(y~cs(age^0.5), data=mydata) or as nage<-(age^0.5); 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 http://www.gamlss.com/).

See Also

lp, lpred

Examples

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^.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 and Calliope Akantziliotou

References


(see also [http://www.gamlss.com/](http://www.gamlss.com/)).

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 function 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, col="darkgreen")

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
col The colour of the profile line

Details

This function can be used 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 \texttt{ProfLikelihood.gamlss} object if saved. The object contains:

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

References


(see also \url{http://www.gamlss.com/}).

See Also

\texttt{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)
```
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, col="darkgreen")

Arguments

model this is a GAMLSS model, e.g. model=gamlss(y~cs(x,df=this),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)

col the color of the profile line
Details

This function can be used 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 a `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)

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

References


(see also [http://www.gamlss.com/](http://www.gamlss.com/)).

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)

ps

P-Splines Fits in a GAMLSS Formula

Description

There are several function which use P-spline methodology:

a) pb(), the current version of P-splines which uses SVD in the fitting and therefore is the most reliable

b) pbo() and pbp(), older versions of P-splines. The first uses a simple matrix algebra in the fits. The second is the last version of pb() with SVD but uses different method for prediction.

c) pbc() the new version of cycle P-splines (using SVD)

d) cy() the older version of cycle P-splines.

e) pbm() for fitting monotonic P-splines (using SVD)

f) pbz() for fitting P-splines which allow the fitted curve to shrink to zero degrees of freedom

 g) ps() the original P-splines with no facility of estimating the smoothing parameters and

j) pvc() penalised varying coefficient models.

k) pvp() older version of pb() where the prediction was different (it is here in case someone would like to compare the results).

Theoretical explanation of the above P-splines can be found in Eilers et al. (2016)
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.pbc() gamlss.cy() gamlss.pvc(), gamlss.pbm(), gamlss.pbz and gamlss.ps() which are used in the backfitting function additive.fit.
The function pb() is more efficient and faster than the original penalised smoothing function ps(). After December 2014 the pb() has changed radically to improved performance. The older version of the pb() function is called now pbo(). pb() allows the estimation of the smoothing parameters using different local (performance iterations) methods. The method are "ML", "ML-1", "EM", "GAIC" and "GCV".
The function \texttt{pbm()} fits monotonic smooth functions, that is functions which increase or decrease monotonically depending on the value of the argument \texttt{mono} which takes the values "up" or "down".

The function \texttt{pbz()} is similar to \texttt{pb()} with the extra property that when \texttt{lambda} becomes very large the resulting smooth function goes to a constant rather than to a linear function. This is very useful for model selection. The function is based on Maria Durban idea of using a double penalty, one of order 2 and one of order 1. The second penalty only applies if the effective df are close to 2 (that is if a linear is already selected).

The function \texttt{pbc()} fits a cycle penalised beta regression spline such as the last fitted value of the smoother is equal to the first fitted value. \texttt{cy()} is the older version.

The function \texttt{pvc()} fits varying coefficient models see Hastie and Tibshirani(1993) and it is more general and flexible than the old \texttt{vc()} function which was based on cubic splines.

The function \texttt{getZmatrix()} creates a (random effect) design matrix \texttt{Z} which can be used to fit a P-splines smoother using the \texttt{re()} function. (The \texttt{re()} is an interface with the random effect function \texttt{lme} of the package \texttt{nlme}.)

\textbf{Usage}

\begin{verbatim}
pb(x, df = NULL, lambda = NULL, max.df=NULL, control = pb.control(...), ...)
pbo(x, df = NULL, lambda = NULL, control = pbo.control(...), ...)
ppb(x, df = NULL, lambda = NULL, control = ppb.control(...), ...)
pbo.control(inter = 20, degree = 3, order = 2, start = 10, quantiles = FALSE,
    method = c("ML", "GAIC", "GCV", "EM", "ML-1"), k = 2, ...)
ppb.control(inter = 20, degree = 3, order = 2, start = 10, quantiles = FALSE,
    method = c("ML", "GAIC", "GCV"), k = 2, ...)
pbc(x, df = NULL, lambda = NULL, max.df=NULL, control = pbc.control(...), ...)
pbc.control(inter = 20, degree = 3, order = 2, start = 10, method = c("ML", "GAIC", "GCV"), k = 2, sin = TRUE, ...)
cy(x, df = NULL, lambda = NULL, control = cy.control(...), ...)
cy.control(inter = 20, degree = 3, order = 2, start = 10, method = c("ML", "GAIC", "GCV", "EM", "ML-1"), k = 2, ts=FALSE, ...)
pvc(x, df = NULL, lambda = NULL, by = NULL, control = pvc.control(...), ...)
pvc.control(inter = 20, degree = 3, order = 2, start = 10, quantiles = FALSE,
    method = c("ML", "GAIC", "GCV"), k = 2, ...)
pbm(x, df = NULL, lambda = NULL, mono=c("up", "down"), control = pbm.control(...), ...)
pbm.control(inter = 20, degree = 3, order = 2, start = 10, quantiles = FALSE,
    method=c("ML", "GAIC", "GCV"), k=2, kappa = 1e10, ...)
pbz(x, df = NULL, lambda = NULL, control = pbz.control(...), ...)
pbz.control(inter = 20, degree = 3, order = 2, start = c(1e-04, 1e-04),
    quantiles = FALSE, method = c("ML", "GAIC", "GCV"), k = 2, lim = 3, ...)
ps(x, df = 3, lambda = NULL, ps.intervals = 20, degree = 3, order = 3)
getZmatrix(x, xmin = NULL, xmax = NULL, inter = 20, degree = 3, order = 2)
\end{verbatim}
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)
- **lambda** the smoothing parameter
- **max.df** the limit of how large the effective degrees of freedom should be allowed to be
- **control** setting the control parameters
- **by** 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.

... for extra arguments

- **inter** the no of break points (knots) in the x-axis
- **degree** the degree of the piecewise polynomial
- **order** the required difference in the vector of coefficients
- **start** the lambda starting value if the local methods are used, see below
- **quantiles** if TRUE the quantile values of \( x \) are use to determine the knots
- **ts** if TRUE assumes that it is a seasonal factor
- **method** The method used in the (local) performance iterations. Available methods are "ML", "ML-1", "EM", "GAIC" and "GCV"
- **k** the penalty used in "GAIC" and "GCV"
- **mono** for monotonic P-splines whether going "up" or "down"
- **kappa** the smoothing hyper-parameter for the monotonic part of smoothing
- **ps.intervals** the no of break points in the x-axis
- **xmin** minimum value for creating the B-spline
- **xmax** maximum value for creating the B-spline
- **sin** whether to use the sin penalty or not
- **lim** at which level the second penalty of order 1 should start

Details

The `ps()` function is based on Brian Marx function which can be found in his website. 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 or down) P-spline smoother.
The function `pbc()` 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. Scaling the response variable will solve the problem.

**Author(s)**

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

**References**


(see also [http://www.gamlss.com/](http://www.gamlss.com/)).

**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(aids1),col="yellow"))
rm(aids1, aids2, aids3)

#=============================  
## Not run:
# pbc()  
# simulate data
set.seed(555)
x = seq(0, 1, length = 100)
y = sign(cos(1 * x * 2 * pi + pi / 4)) + rnorm(length(x)) * 0.2
plot(y~x)
m1<-gamlss(y~pbc(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)
plot(R~Fl, data=rent, pch=21, bg=c(“gray”, “blue”)[unclass(rent$B)])
r1 <- gamlss(R-pb(Fl), data=rent)
# Identical to model
r11 <- gamlss(R-pvc(Fl), data=rent)
# Now with the factor
r2 <- gamlss(R-pvc(Fl, by=B), data=rent)
lines(fitted(r2)[rent$B==1][order(rent$Fl[rent$B==1])]-rent$Fl[rent$B==1]
[order(rent$Fl[rent$B==1])], col="blue", lwd=2)
lines(fitted(r2)[rent$B==0][order(rent$Fl[rent$B==0])]-rent$Fl[rent$B==0]
[order(rent$Fl[rent$B==0])], col="red", lwd=2)
# Probably not very sensible model
rm(r1,r11,r2)

# Now with a continuous variable
# Additive model
h1 <- gamlss(R-pb(Fl)+pb(A), data=rent)
# Varying-coefficient model
h2 <- gamlss(R-pb(Fl)+pb(A)+pvc(A, by=Fl), 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")

# The pbz() function
# Creating uncorrelated data
set.seed(123)
y<-rNO(100)
x<-1:100
plot(y~x)
Q.stats

#----------------------
# ML estimation
m1<-gamlss(y~pbz(x))
m2 <-gamlss(y~pb(x))
AIC(m1,m2)
op <- par( mfrow=c(1,2))
term.plot(m1, partial=T)
term.plot(m2, partial=T)
par(op)

# GAIC estimation
m11<-gamlss(y~pbz(x, method="GAIC", k=2))
m21 <-gamlss(y~pb(x, method="GAIC", k=2))
AIC(m11,m21)
op <- par( mfrow=c(1,2))
term.plot(m11, partial=T)
term.plot(m21, partial=T)
par(op)

# GCV estimation
m12<-gamlss(y~pbz(x, method="GCV"))
m22 <-gamlss(y~pb(x, method="GCV"))
AIC(m12,m22)
op <- par( mfrow=c(1,2))
term.plot(m12, partial=T)
term.plot(m22, partial=T)
par(op)

# fixing df is more tricky since df are the extra df
m13<-gamlss(y~pbz(x, df=0))
m23 <-gamlss(y~pb(x, df=0))
AIC(m13,m23)

# here the second penalty is not take effect therefore identical results
m14<-gamlss(y~pbz(x, df=1))
m24 <-gamlss(y~pb(x, df=1))
AIC(m14,m24)

# fixing lambda
m15<-gamlss(y~pbz(x, lambda=1000))
m25 <-gamlss(y~pb(x, lambda=1000))
AIC(m15,m25)

#----------------------
# prediction
m1<-gamlss(y~pbz(x), data=data.frame(y,x))
m2 <-gamlss(y~pb(x), data=data.frame(y,x))
AIC(m1,m2)
predict(m1, newdata=data.frame(x=c(80, 90, 100, 110))
predict(m2, newdata=data.frame(x=c(80, 90, 100, 110))

## End(Not run)

Q.stats

A function to calculate the Q-statistics
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)
- **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 split, with default 10
- **zvals**: if TRUE the output matrix contains the individual Z-statistics rather that the Q statistics
- **save**: whether to save the Q-statistics or not with default equal to TRUE. In this case the functions produce a matrix giving individual Q (or z) statistics and the final aggregate Q's
- **plot**: whether to plot a visual version of the Q statistics (default is TRUE)
- **digits.xvar**: to control the number of digits of the xvar in the plot
- **...**: for extra arguments

Details
Note that the function Q.stats behaves differently depending whether the obj or the resid argument is set. The 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 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 plot=TRUE it produces also an graphical representation of the table.

Author(s)
Mikis Stasinopoulos <d.stasinopoulos@londonmet.ac.uk>, Bob Rigby with contributions from Elaine Borghie
References


(see also http://www.gamlss.com/).

See Also
gamlss, centiles.split, 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

Quantile 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,
$k = 2, \ c.crit = 0.01, \ step = 0.1)$

\[ z.\text{scoresQS}(\text{object, } y, \ x, \ \text{plot} = \text{FALSE, } \text{tol} = \text{NULL}) \]

**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
- **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.

**Author(s)**

Mikis Stasinopoulos based on function provided by Paul Eiler and Sabine Schnabe

**References**


(see also http://www.gamlss.com/).

**See Also**

`lms`: for a parametric equivalent results.

**Examples**

```r
data(abdom)
m1 <- quantSheets(y,x, data=abdom)
head(fitted(m1))
p1 <- predict(m1, newdata=c(20,30,40))
matpoints(c(20,30,40), p1)
z.scoresQS(m1,y=c(150, 300),x=c(20, 30) )
# If we needed a power transformation not appropriate for this data
findPower(y,x, data=abdom)
```
**random**  
*Specify a random intercept model 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 more 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).

**Usage**

```r
random(x, df = NULL, lambda = NULL, start=10)
```

```r
re(fixed = ~1, random = NULL, correlation = NULL, method = "ML",
    level = NULL, ...)
```

**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 Maximul likelihood is used.
- `fixed`  
a formula specify 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"
- `level`  
this argument has to be set to zero (0) if when use `predict()` you want to get the marginal contribution
- `...`  
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 betaken 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

#---------- 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()
t1<-gamlss(effort~Type+random(Subject), data=ergoStool )
# use re() with fixed effect within re()
t2<-gamlss(effort=re(fixed=-Type, random=-1|Subject), data=ergoStool )
# use re() with fixed effect in gamlss formula
t3<-gamlss(effort~Type+re(random=-1|Subject), data=ergoStool )
# compare lme fitted values with random
plot(fitted(l1), fitted(t1))
# compare lme fitted values with random
plot(fitted(l1), fitted(t2))
lines(fitted(l1), fitted(t3), col=2)
# getting the fitted coefficients
getSmo(t2)
#----------
## 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),
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))
fixef(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(t1))

#---------------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
plot(t1)
wp(t1, ylim.all=2)
# two observation fat try LO

# a bit better but not satisfactory Note that 3 parameter distributions fail
library(MASS)
data(bacteria)
summary(glmmPQL(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)

#----------------Example 6 from Pinheiro and Bates (2000) page 239-244----------------
# using corAR1()
data(Ovary)
# AR1
refit <- lme(follicles~sin(2*pi*Time)+cos(2*pi*Time), data=Ovary, 
  random=pdDiag(~sin(2*pi*Time)), correlation=corAR1())
# ARMA
l2 <- lme(follicles~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(follicles~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(follicles~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)

refit

Refit a GAMLSS model

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
residuals.gamlss

Note

The function `update` does a very similar job

Author(s)

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

References


(see also [http://www.gamlss.com/](http://www.gamlss.com/)).

See Also

`gamlss`, `update.gamlss`

Examples

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

residuals.gamlss Extract Residuals from GAMLSS model

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

References


(see also http://www.gamlss.com/).

See Also

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

data(aids)

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

ri

Specify ridge or lasso Regression within a GAMLSS Formula

Description

The function ri() allows the user to fit a ridge regression within GAMLSS. It allows the coefficients of a set of explanatory variables to be shrunk towards zero. The amount of shrinking depends either on lambda, or on the equivalent degrees of freedom (df). The type of shrinking depends on the argument Lp see example.

Usage

ri(X = NULL, x.vars = NULL, df = NULL, lambda = NULL, method = c("ML", "GAIC"), order = 0, start = 10, Lp = 2, kappa = 1e-05, iter = 100, c.crit = 1e-06, k = 2)

Arguments

X A matrix of explanatory variables X which is standardised (mean=0, sd=1) automatically. Note that in order to get predictions you should use the option x.vars

x.vars which variables from the data.frame declared in data needs to be included. This is a way to fit the model if predictions are required.

df the effective degrees of freedom df

lambda the smoothing parameter lambda

method which method is used for the estimation of the smoothing parameter, ‘ML’ or ‘GAIC’ are allowed.

order the order of the difference applied to the coefficients with default zero. (Do not change this unless there is some ordering in the explanatory variables.)

start starting value for lambda if it estimated using ‘ML’ or ‘GAIC’

Lp The type of penalty required, Lp=2 a proper ridge regression is the default. Use codeLp=1 for lasso and different values for different penalties.

kappa a regulation parameters used for the weights in the penalties.

iter the number of internal iteration allowed see details.

c.crit c.crit is the convergent criterion

k k is the penalty if ‘GAIC’ method is used.
Details

This implementation of ridge and related regressions is based on an idea of Paul Eilers which used weights in the penalty matrix. The type of weights are defined by the argument \( L_p \). \( L_p=2 \) is the standard ridge regression, \( L_p=1 \) fits a lasso regression while \( L_p=0 \) allows a "best subset" regression see Hastie et al (2009) page 71.

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 and Paul Eilers

References


(see also http://www.gamlss.com/).

See Also

gamlss

Examples

# USAIR DATA
# standarise data 1-----------------------------------------------
# ridge
m1<- gamlss(y~ri(x.vars=c("x1","x2","x3","x4","x5","x6")),
           data=usair)
# lasso
m2<- gamlss(y~ri(x.vars=c("x1","x2","x3","x4","x5","x6"), Lp=1),
           data=usair)
# best subset
m3<- gamlss(y~ri(x.vars=c("x1","x2","x3","x4","x5","x6"), Lp=0),
           data=usair)
#------- plotting the coefficients
op <- par(mfrow=c(3,1))
plot(getSmo(m1)) #
plot(getSmo(m2))
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.type = c("few", "all"), type = c("wp", "QQ"), xlim = NULL, ylim = NULL, ...)

get.rqres(obj = NULL, howmany = 10, order = FALSE)

Arguments

obj a fitted GAMLSS model object from a "discrete" type of family
howmany The number randomise quantile residuals required i.e. howmany=6
plot.type whether to plot few of the randomised quantile residual realisations, "few" in a separate plots (there must be less than 8) or all "all" in one plot (with their median)
type whether to plot worm plots "wp"or QQ plots "QQ" with default worm plots
xlim setting manually the xlim of the graph
ylim setting manually the ylim of the graph
order whether to order the ealization of randomised quantile residuals
... for extra arguments to be passed to wp()

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
```r
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 worm-plot of the medians from 10 realizations
rqres.plot(h,howmany=40,plot="all") #
```

Rsq

Generalised (Pseudo) R-squared for GAMLSS models

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

Usage
```r
Rsq(object, type = c("Cox Snell","Cragg Uhler","both"))
```

Arguments
```r
object    a GAMLSS object
```
```r
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 http://www.gamlss.com/).

See Also

GAIC

Examples

data(aids)
m1 <- gamlss(y~x+qrt, data=aids, family=NBI)
Rsq(m1)
Rsq(m1, type="both")
rm(m1)
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.K()` gets the $K$ matrix (see details below).

**Usage**

```r
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 $S = V K V$ 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 use the 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 and Vlasios Voudouris
References


(see also [http://www.gamlss.com/](http://www.gamlss.com/)).

See Also

vcov, ~~~

Examples

# 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 conventinal 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 stardard 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 (GAIC). It is based on the function `stepAIC()` given in the library MASS of Venables and Ripley (2002). The function has been changed recently to allow parallel computation. The parallel computations are similar to the ones performed in the function `boot()` of the `boot` package. Note that since version 4.3-5 of `gamlss` the `stepGAIC()` do not have the option of using the function `stepGAIC.CH` through the argument `additive`.
Note that \texttt{stepGAIC()} is relying to the \texttt{dropterm()} and \texttt{addterm()} methods applied to \texttt{gamlss} objects. \texttt{drop1()} and \texttt{add1()} are equivalent methods to the \texttt{dropterm()} and \texttt{addterm()} respectively but with different default arguments (see the examples).

The function \texttt{stepGAIC.VR()} is the old version of \texttt{stepGAIC()} with no parallel computations.

The function \texttt{stepGAIC.CH} is based on the S function \texttt{step.gam()} (see Chambers and Hastie (1991)) and it is more suited for models with smoothing additive terms when the degrees of freedom for smoothing are fixed in advance. This is something which rarely used these days, as most of the smoothing functions allow the calculation of the smoothing parameter, see for example the additive function \texttt{pb()}.

The functions \texttt{stepGAIC.VR()} and \texttt{stepGAIC.CH()} have been adapted to work with \texttt{gamlss} objects and the main difference is the \texttt{scope} argument, see below.

While the functions \texttt{stepGAIC()} is used to build models for individual parameters of the distribution of the response variable, the functions \texttt{stepGAICAll.A()} and \texttt{stepGAICAll.A()} are building models for all the parameters.

The functions \texttt{stepGAICAll.A()} and \texttt{stepGAICAll.B()} are based on the \texttt{stepGAIC()} function but use different strategies for selecting a appropriate final model. \texttt{stepGAICAll.A()} has the following strategy:

Strategy A:

i) build a model for \texttt{mu} using a forward approach.

ii) given the model for \texttt{mu} build a model for \texttt{sigma} (forward)

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

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

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

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

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

Note for this strategy to work the \texttt{scope} argument should be set appropriately.

\texttt{stepGAICAll.B()} uses the same procedure as the function \texttt{stepGAIC()} but each term in the scope is fitted to all the parameters of the distribution, rather than the one specified by the argument \texttt{what} of \texttt{stepGAIC()}. The \texttt{stepGAICAll.B()} relies on the \texttt{add1All()} and \texttt{drop1All()} functions for the selection of variables.

\textbf{Usage}

\begin{verbatim}
stepGAIC(object, scope, direction = c("both", "backward", "forward"),
         trace = T, keep = NULL, steps = 1000, scale = 0,
         what = c("mu", "sigma", "nu", "tau"), parameter= NULL, k = 2,
         parallel = c("no", "multicore", "snow"), ncpus = 1L, cl = NULL,
         ...)  
stepGAIC.VR(object, scope, direction = c("both", "backward", "forward"),
            trace = T, keep = NULL, steps = 1000, scale = 0,
            what = c("mu", "sigma", "nu", "tau"), parameter= NULL, k = 2,
            parallel = c("no", "multicore", "snow"), ncpus = 1L, cl = NULL,
            ...)  
\end{verbatim}
trace = T, keep = NULL, steps = 1000, scale = 0,
what = c("mu", "sigma", "nu", "tau"), parameter= NULL, 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"),
p parameter= NULL, k = 2, ...

stepGAICAll.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,
parallel = c("no", "multicore", "snow"), ncpus = 1L,
c l = NULL, ...)

stepGAICAll.B(object, scope, direction = c("both", "backward", "forward"),
trace = T, keep = NULL, steps = 1000, scale = 0, k = 2,
parallel = c("no", "multicore", "snow"), ncpus = 1L,
c l = NULL, ...)

drop1All(object, scope, test = c("Chisq", "none"), k = 2, sorted = FALSE,
trace = FALSE, parallel = c("no", "multicore", "snow"),
cpus = 1L, cl = NULL, ...)

add1All(object, scope, test = c("Chisq", "none"), k = 2, sorted = FALSE,
trace = FALSE, parallel = c("no", "multicore", "snow"),
cpus = 1L, cl = NULL, ...)

Arguments

object an 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"

**parameter**
equivalent to what

**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.

**parallel**
The type of parallel operation to be used (if any). If missing, the default is "no".

**ncpus**
integer: number of processes to be used in parallel operation: typically one would chose this to the number of available CPUs.

**cl**
An optional parallel or snow cluster for use if parallel = "snow". If not supplied, a cluster on the local machine is created for the duration of the call.

**sigma.scope**
scope for sigma if different to scope in stepGAICall.A()

**nu.scope**
scope for nu if different to scope in stepGAICall.A()

**tau.scope**
scope for tau if different to scope in stepGAICall.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

**test**
whether to print the chi-square test or not

**sorted**
whether to sort the results

... 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 modelling 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).

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


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

org/v23/i07.

Regression and Smoothing: Using GAMLSS in R, Chapman and Hall/CRC. (see also http://www.
gamlss.com/).


See Also

gamlss.scope

Examples

## Not run:
data(usair)
#------------------------------------------------------------
# null model
mod0<-gamlss(y~1, data=usair, family=GA)
# all the explanatory variables x1:x6 fitted linearly
mod1<-gamlss(y~., data=usair, family=GA)
#------------------------------------------------------------
# dropping terms
dropterm(mod1)
# with chi-square information
drop1(mod1)
# for parallel computations use something like
nC <- detectCores()
drop1(mod1, parallel="snow", ncpus=nC)
drop1(mod1, parallel="multicore", ncpus=nC)

# adding terms
addterm(mod0, scope=as.formula(paste("-", paste(names(usair[-1]),
collapse="+",sep="")))
# with chi-square information
add1(mod0, scope=as.formula(paste("-", paste(names(usair[-1]),
collapse="+",sep="")))
# for parallel computations
nC <- detectCores()
add1(mod0, scope=as.formula(paste("-", paste(names(usair[-1]),
collapse="+",sep=""))), parallel="snow", ncpus=nC)

# stepGAIC
# find the best subset for the mu
mod2 <- stepGAIC(mod1)
mod2$anova

# for parallel computations
mod21 <- stepGAIC(mod1, , parallel="snow", ncpus=nC)

# 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
mod20<-stepGAIC(mod0, scope=list(lower=~1, upper=~pb(x1)+pb(x2)+pb(x5)))
edf(mod20)
# note that x1 and x2 enter linearly

# the stepGAIC.CH function (no parallel here)
# creating a scope from the usair model frame
gs<-gamlss.scope(model.frame(y~x1+x2+x3+x4+x5+x6, data=usair))
gs
mod5<-stepGAIC.CH(mod0,gs)
mod5$anova

# now stepGAICAll.A
mod7<-stepGAICAll.A(mod0, scope=list(lower=~1,upper=~x1+x2+x3+x4+x5+x6))

# now stepGAICAll.B
drop1All(mod1, parallel="snow", ncpus=nC)
add1All(mod0, scope=as.formula(paste("-", paste(names(usair[-1]),
collapse="+"))), parallel="snow", ncpus=nC)
mod8<-stepGAICAll.B(mod0, scope=list(lower=~1,upper=~x1+x2+x3+x4+x5+x6))
#--------------------------------------------------------------
#--------------------------------------------------------------
## End(Not run)

summary.gamlss  

**Summary**

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

**Usage**

```r
## 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 and Calliope Akantziliotou

References


(see also http://www.gamlss.com/).

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

term.plot(object, what = c("mu", "sigma", "nu", "tau"),
          parameter = NULL, 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,
          polys = NULL, polys.scheme = "topo", ...)

Arguments

object a fitted GAMLSS object
what the required parameter of the GAMLSS distribution i.e. "mu"
parameter equivalent to what
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 compared to the rest. In the 'free'
        option the limits are computed for each plot separately.
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.
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
polys        The polygon information file for MRF models
polys.scheme  Color scheme for polygons for RMF models
...          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 [http://www.gamlss.com/](http://www.gamlss.com/)).

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

Update and Re-fit a GAMLSS Model

Description

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

Usage

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

Arguments

- `object`: a GAMLSS fitted model
- `formula.`: the formula to update
- `...`: for updating argument in `gamlss()`
- `what`: 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
- `parameter`: equivalent to `what`
- `evaluate`: 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

References


(see also [http://www.gamlss.com/](http://www.gamlss.com/)).
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(y))~.+qrt, family=NO)
# verify that it is the same
h.no <- gamlss(log(y)~cs(x,8)+qrt, data=aids)

VC.test

Vuong and Clarke tests

Description

The Vuong and Clarke tests for GAMLSS fitted models.

Usage

VC.test(obj1, obj2, sig.lev = 0.05)

Arguments

obj1
The first fitted gamlss object

obj2
The second fitted gamlss object

sig.lev
Significance level used for testing.

Details

The Vuong (1989) and Clarke (2007) tests are likelihood-ratio-based tests for model selection that use the Kullback-Leibler information criterion. The implemented tests can be used for choosing between two bivariate models which are not necessarily nested.

In the Vuong test, the null hypothesis is that the two models are equally close to the actual model, whereas the alternative is that one model is closer. The test follows asymptotically a standard normal distribution under the null. Assume that the critical region is \((-c, c)\), where \(c\) is typically set to 1.96. If the value of the test is greater than \(c\) then we reject the null hypothesis that the models are equivalent in favour of the model in \(obj1\). Vice-versa if the value is smaller than \(-c\) we reject
the null hypothesis that the models are equivalent in favour of the model in `obj2`. If the value falls within \((-c, c_0)\) then we cannot discriminate between the two competing models given the data.

In the Clarke test, if the two models are statistically equivalent then the log-likelihood ratios of the observations should be evenly distributed around zero and around half of the ratios should be larger than zero. The test follows asymptotically a binomial distribution with parameters \(n\) and 0.5. Critical values can be obtained as shown in Clarke (2007). Intuitively, the model in `obj1` is preferred over that in `obj2` if the value of the test is significantly larger than its expected value under the null hypothesis (`coden/2`), and vice versa. If the value is not significantly different from \(n/2\) then `obj1` can be thought of as equivalent to `obj2`.

**Value**

For the Vuong test it returns its value and the decision and for the Clarke test returns the value the p-value and the decision. Decisions criteria are as discussed above.

**Author(s)**

Mikis Stasinopoulos and Giampiero Marra

**References**


**See Also**

`LR.test`

**Examples**

```r
library(gamlss)
# fitting different models
m0 <- gamlss(y~x+qrt, data=aids, family=PO)
m1 <- gamlss(y~pb(x)+qrt, data=aids, family=PO)
m2 <- gamlss(y~pb(x)+qrt, data=aids, family=NBI)
# comparison of the models
VC.test(m0,m2)
VC.test(m0,m1)
VC.test(m1,m2)
```
Description

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 de-trended 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, cex.lab = 1, pch = 21, bg = "wheat", 
col = "red", bar.bg = c(num = "light blue"), ...)

Arguments

object a GAMLSS fitted object or any other fitted model where the resid() method works (preferably it should be standardised 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 for changing the side of worm with default cex=1

cex.lab  the cex plotting parameter for changing the size of the axis labels

pch  the pch plotting parameter with default pch=21

bg  The background colour of the worm plot points

col  the colour of the fitted (and horizontal and vertical) lines

bar.bg  the colour of the bars when xvar is used

...  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.iter. In this case the x-variable is cut into n.iter 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.iter=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.iter 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 environment to find xvar (whether it is a formula or not). As a result if data exists within object xvar=~x*f can be used (assuming that 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**

```r
data(abdom)
# with data
a<-gamlss(y~pb(x),sigma.fo=~pb(x),family=L0,data=abdom)
wp(a)
coeff1<-wp(a,xvar=x)
coeff1
## Not run:
# no data argument
b <- gamlss(abdom$y~pb(abdom$x),sigma.fo=~pb(abdom$x),family=L0)
wp(b)
wp(b, xvar=abdom$x)# not wp(b, xvar=x)
# using the argument resid
# this will work
wp(resid=resid(a), xvar=abdom$x)
# not this
# wp(resid=resid(a), xvar=x)
# this example uses the rent data
m1 <- gamlss(R~pb(Fl)+pb(A)+loc, sigma.fo=~pb(Fl)+pb(A), data=rent, family=GA)
# a single worm plot
wp(m1, ylim.all=0.5)
# a single continuous x variable
wp(m1, xvar=Fl, ylim.worm=.8)
# a single x variable changing the default number of intervals
wp(m1, xvar=Fl, ylim.worm=1.5, n.inter=9)
# different x variable changing the default number of intervals
B1<-wp(m1, xvar=A, ylim.worm=1.2, n.inter=9)
B1
# the number five plot has intervals
# [5,1957.5,1957.5]
# rather disappointing
# try formula for xvar
wp(m1, xvar=~A, ylim.worm=1.2, n.inter=9)
# better in this case using formula
# now using a factor included in the model
wp(m1, xvar=~loc, ylim.worm=1.2, n.inter=9)
# using a factor not in the model
wp(m1, xvar=~B, ylim.worm=1.5, n.inter=9)
# level 2 (with B=1) did not fit well
# trying two continuous variable
wp(m1, xvar=~Fl*A, ylim.worm=1.5, n.inter=4)
```
## z.scores

### Z-scores for lms objects

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

### Usage

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

### 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 [http://www.gamlss.com/](http://www.gamlss.com/)).
See Also

centiles.pred

Examples

## 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 Statistical Models
  getQuantile, 66
*Topic distribution
  fitDist. 34
  gamlss-package, 3
  histSmo, 72
  loglogSurv, 82
  plot.histSmo, 98
*Topic package
  gamlss-package, 3
*Topic regression
  pcat, 92
*Topic regression
  additive.fit, 8
  bfp, 9
  calibration, 11
  centiles, 13
  centiles.com, 15
  centiles.pred, 17
  centiles.split, 20
  coef.gamlss, 21
  cs, 23
  deviance.gamlss, 25
  devianceIncr, 27
  dtop, 28
  edf, 30
  find.hyper, 31
  fitDist, 34
  fitted.gamlss, 38
  fittedPlot, 39
  formula.gamlss, 40
  gamlss, 41
  gamlss-package, 3
  gamlss.control, 46
  gamlss.cs, 48
  gamlss.fp, 49
  gamlss.lo, 50
  gamlss.ps, 51
  gamlss.random, 52
  gamlss.scope, 53
  gamlssML, 54
  gamlssVGD, 57
  gen.likelihood, 63
  getPEF, 64
  getQuantile, 66
  getSmo, 68
  glim.control, 69
  histDist, 70
  IC, 75
  lms, 77
  lo, 79
  lpred, 85
  LR.test, 87
  model.frame.gamlss, 88
  numeric.deriv, 89
  par.plot, 90
  pdf.plot, 94
  plot.gamlss, 96
  plot2way, 99
  polyS, 100
  predict.gamlss, 101
  print.gamlss, 104
  prof.dev, 105
  prof.term, 107
  ps, 109
  Q.stats, 115
  quantSheets, 117
  random, 120
  refit, 124
  residuals.gamlss, 125
  ri, 127
  rqres.plot, 129
  Rsq, 130
  rvcov, 132
  stepGAIC, 133
  summary.gamlss, 139
  term.plot, 140
  update.gamlss, 143
INDEX

gen.likelihood, 63
get.K (rvcov), 132
get.rqres (rqres.plot), 129
getOrder (fitDist), 34
getPEF, 64, 67
getQuantile, 66
getSmo, 68
getTGD (gamlssVDG), 57
getzmatrix (ps), 109
glim.control, 43, 69
histDist, 70
histSmo, 72, 98
histSmoC (histSmo), 72
histSmoO (histSmo), 72
histSmoP (histSmo), 72
ic, 75
is.gamlss (gamlss), 41
lms, 77, 119
lo, 51, 79
loglogplot (loglogSurv), 82
loglogplot0 (loglogSurv), 82
loglogSurv, 82
loglogSurv1 (loglogSurv), 82
loglogSurv2 (loglogSurv), 82
loglogSurv3 (loglogSurv), 82
logSurv (loglogSurv), 82
logSurv0 (loglogSurv), 82
lp, 103
lp (lpred), 85
lpred, 85, 103
LR.test, 87, 145
model.frame.gamlss, 88
model.matrix.gamlss
  (model.frame.gamlss), 88
numeric.deriv, 89
optim, 33
par.plot, 90
pb, 25, 52, 74
pb (ps), 109
pbc (ps), 109
pbm, 52
pbm (ps), 109
pbo (ps), 109
php (ps), 109
pbz (ps), 109
pcat, 92
pdf.plot, 45, 94
plot.gamlss, 33, 39, 96, 126, 130, 144, 148
plot.histSmo, 98
plot2way, 99
plotDF (pcat), 92
plotLambda (pcat), 92
poly.matrix (pcat), 100
polyS, 100
pp (bfp), 9
predict.gamlss, 86, 101
predictAll (predict.gamlss), 101
print.gamlss, 39, 104, 126, 144
prof.dev, 105, 108
prof.term, 106, 107
ps, 52, 109
pvc, 25, 52
pvc (ps), 109
Q.stats, 115
quantSheets, 117
random, 53, 82, 93, 120
re (random), 120
refit, 46, 124
residuals.gamlss, 39, 125, 126, 144
ri, 52, 127
ridge, 52
rqres.plot, 126, 129
Rsq, 130
rvcov, 132
scs (cs), 23
stepGAIC, 54, 61, 133
stepGAICA1.A (stepGAIC), 133
stepGAICA1.B (stepGAIC), 133
stepTGD (gamlssVDG), 57
stepTGA1.A (gamlssVDG), 57
summary.gamlss, 39, 126, 139, 144
term.plot, 100, 140
termplot, 142
terms.gamlss (model.frame.gamlss), 88
TGD (gamlssVDG), 57
update, 125
update.gamlss, 39, 125, 126, 143
VC.test, 144
vcov, 64, 133
VGD (gamlssVDG), 57
vis.lo (lo), 79

wp, 30, 117, 146

z.scores, 149
z.scoresQS (quantSheets), 117