Package ‘gamlss.nl’

February 19, 2015

Type Package
Title Fitting non linear parametric GAMLSS models
Version 4.1-0
Date 2012-02-15
Author Mikis Stasinopoulos <d.stasinopoulos@londonmet.ac.uk>, Bob Rigby <r.rigby@londonmet.ac.uk> with contributions from Philippe Lambert.
Maintainer Mikis Stasinopoulos <d.stasinopoulos@londonmet.ac.uk>
LazyLoad yes
Depends R (>= 2.2.1), gamlss, survival
Description This is an add on package to GAMLSS. It allows one extra
   method for fitting GAMLSS models. The main function nlgamlss()
   can fit any parametric (up to four parameter) GAMLSS
   distribution.
License GPL-2 | GPL-3
URL http://www.gamlss.org/
Repository CRAN
Date/Publication 2012-02-15 12:56:58
NeedsCompilation no

R topics documented:

  gamlss-nl-package .................................................. 2
  finterp ............................................................. 3
  la ................................................................. 8
  nl ................................................................. 9
  NL.control ......................................................... 11
  nlgamlss ......................................................... 13

Index 17
The GAMLSS add on package for fitting parametric non-linear models

Description

The main purpose of this package is to allow non-linear fitting within a GAMLSS model. The main function `nlgamlss()` can fit any parametric (up to four distribution parameters) GAMLSS family of distributions.

Details

Package: gamlss-nl
Type: Package
Version: 1.5.0
Date: 2005-12-12
License: GPL (version 2 or later)

Author(s)

Mikis Stasinopoulos <d.stasinopoulos@londonmet.ac.uk>, Bob Rigby <r.rigby@londonmet.ac.uk> based on work of Jim Lindsey and Philippe Lambert.

References


See Also

gamlss

Examples

data(la)
# fitting the Johnson's Su distribution to the data
modJSU <- nlgamlss(y=PET60, mu.formula=-bflow*(1-p1*exp(-p2/bflow)), sigma.formula=-1, nu.formula=-1, mu.start=c(.6, 110), sigma.start=3, nu.start=1, tau.start=0.6, family=JSU, data=la)
plot(modJSU)
summary(modJSU)
vcov(modJSU)
**finterp**  
*Formula Interpreter*

**Description**

This function is taken from Jim Lindsey’s R package `rmutil`.

What follows is taken from the help file of `rmutil`. Note that not all the functionalities of finterp are implemented in nlgamlss.

finterp translates a model formula into a function of the unknown parameters or of a vector of them. Such language formulae can either be in Wilkinson and Rogers notation or be expressions containing both known (existing) covariates and unknown (not existing) parameters. In the latter, factor variables cannot be used and parameters must be scalars.

The covariates in the formula are sought in the environment or in the data object provided. If the data object has class, 'repeated' or 'response', then the key words, 'times' will use the response times from the data object as a covariate, 'individuals' will use the index for individuals as a factor covariate, and 'nesting' the index for nesting as a factor covariate. The latter two only work for W&R notation.

Note that, in parameter displays, formulae in Wilkinson and Rogers notation use variable names whereas those with unknowns use the names of these parameters, as given in the formulae, and that the meaning of operators (*, /, :, etc.) is different in the two cases.

The function `fmobj` inspects a formula and returns a list containing the objects referred to, with indicators as to which are unknown parameters, covariates, factor variables, and functions.

**Usage**

```r
## Default S3 method:
finterp(.z, .envir = parent.frame(), .formula = FALSE, .vector = TRUE, .args = NULL, .start = 1, .name = NULL, .expand = TRUE, .intercept = TRUE, .old = NULL, .response = FALSE, ...)
## S3 method for class 'data.frame'
finterp(.z, .envir = NULL, .formula = FALSE, .vector = TRUE, .args = NULL, .start = 1, .name = NULL, .expand = NULL, .intercept = TRUE, .old = NULL, ...)  
finterp(.z, ...)
fmobj(z, envir = parent.frame())
```

**Arguments**

- `.z`  
  A model formula beginning with `\~`, either in Wilkinson and Rogers notation or containing unknown parameters. If it contains unknown parameters, it can have several lines so that, for example, local variables can be assigned temporary values. In this case, enclose the formula in curly brackets.

- `.envir`  
  The environment in which the formula is to be interpreted or a data object of class, 'repeated', 'tccov', or 'tvcov'.

---

**Description**

This function is taken from Jim Lindsey’s R package `rmutil`.

What follows is taken from the help file of `rmutil`. Note that not all the functionalities of finterp are implemented in nlgamlss.

finterp translates a model formula into a function of the unknown parameters or of a vector of them. Such language formulae can either be in Wilkinson and Rogers notation or be expressions containing both known (existing) covariates and unknown (not existing) parameters. In the latter, factor variables cannot be used and parameters must be scalars.

The covariates in the formula are sought in the environment or in the data object provided. If the data object has class, 'repeated' or 'response', then the key words, 'times' will use the response times from the data object as a covariate, 'individuals' will use the index for individuals as a factor covariate, and 'nesting' the index for nesting as a factor covariate. The latter two only work for W&R notation.

Note that, in parameter displays, formulae in Wilkinson and Rogers notation use variable names whereas those with unknowns use the names of these parameters, as given in the formulae, and that the meaning of operators (*, /, :, etc.) is different in the two cases.

The function `fmobj` inspects a formula and returns a list containing the objects referred to, with indicators as to which are unknown parameters, covariates, factor variables, and functions.

**Usage**

```r
## Default S3 method:
finterp(.z, .envir = parent.frame(), .formula = FALSE, .vector = TRUE, .args = NULL, .start = 1, .name = NULL, .expand = TRUE, .intercept = TRUE, .old = NULL, .response = FALSE, ...)
## S3 method for class 'data.frame'
finterp(.z, .envir = NULL, .formula = FALSE, .vector = TRUE, .args = NULL, .start = 1, .name = NULL, .expand = NULL, .intercept = TRUE, .old = NULL, ...)  
finterp(.z, ...)
fmobj(z, envir = parent.frame())
```

**Arguments**

- `.z`  
  A model formula beginning with `\~`, either in Wilkinson and Rogers notation or containing unknown parameters. If it contains unknown parameters, it can have several lines so that, for example, local variables can be assigned temporary values. In this case, enclose the formula in curly brackets.

- `.envir`  
  The environment in which the formula is to be interpreted or a data object of class, 'repeated', 'tccov', or 'tvcov'.
.formula  If TRUE and the formula is in Wilkinson and Rogers notation, just returns the formula.

.vector  If FALSE and the formula contains unknown parameters, the function returned has them as separate arguments. If TRUE, it has one argument, the unknowns as a vector, unless certain parameter names are specified in `.args`. Always TRUE if `.envir` is a data object.

.args  If `.vector` is TRUE, names of parameters that are to be function arguments and not included in the vector.

.start  The starting index value of the parameter vector in the function returned when `.vector` is TRUE.

.name  Character string giving the name of the data object specified by `.envir`. Ignored unless the latter is such an object and only necessary when `finterp` is called within other functions.

.expand  If TRUE, expand functions with only time-constant covariates to return one value per observation instead of one value per individual. Ignored unless `.envir` is an object of class, ‘repeated’.

.intercept  If W&R notation is supplied and `.intercept=F`, a model function without intercept is returned.

.old  The name of an existing object of class ‘formulafn’ which has common parameters with the one being created, or a list of such objects. Only used if `.vector`=TRUE. The value of `.start` should ensure that there is no conflict in indexing the vector.

.response  If TRUE, any response variable can be used in the function. If FALSE, checks are made that the response is not also used as a covariate.

z  A model formula beginning with ~, either in Wilkinson and Rogers notation or containing unknown parameters.

.envir  The environment in which the formula is to be interpreted.

...  for extra arguments

Value

A function, of class `formulafn`, of the unknown parameters or of a vector of them is returned. Its attributes give the formula supplied, the model function produced, the covariate names, the parameter names, and the range of values of the index of the parameter vector. If 'formula' is TRUE and a Wilkinson and Rogers formula was supplied, it is simply returned instead of creating a function.

For `fmobj` a list, of class ‘fmoobj’, containing a character vector (‘objects’) with the names of the objects used in a formula, and logical vectors indicating which are unknown parameters (‘parameters’), covariates (‘covariates’), factor variables (‘factors’), and functions (‘functions’) is returned.

Author(s)

J.K. Lindsey
Examples

# From Jim Lindsey
x1 <- rpois(20,2)
x2 <- rnorm(20)

# Wilkinson and Rogers formula with three parameters
fn1 <- finterp(~x1+x2)
fn1
fn1(rep(2,3))

# the same formula with unknowns
fn2 <- finterp(~b0+b1*x1+b2*x2)
fn2
fn2(rep(2,3))

# nonlinear formulae with unknowns
# log link
fn2a <- finterp(~exp(b0+b1*x1+b2*x2))
fn2a
fn2a(rep(0,2,3))

# parameters common to two functions
fn2b <- finterp(~c0+c1*exp(b0+b1*x1+b2*x2), .old=fn2a, .start=4)
fn2b

# function returned also depends on values of another function
fn2c <- finterp(~fn2+c1*exp(b0+b1*x1+b2*x2), .old=fn2a, .start=4, .args="fn2")
fn2c
args(fn2c)
fn2c(rep(0,2,4),fn2(rep(2,3)))

# compartment model
times <- 1:20
# exp() parameters to ensure that they are positive
fn3 <- finterp(~exp(absorption-volume)/(exp(absorption)-
               exp(elimination))*(exp(-exp(elimination)*times)-
               exp(-exp(absorption)*times))
fn3
fn3(log(c(0.3,3,0.2)))

# a more efficient way
# (note that parameters do not appear in the same order)
form <- ~{
  ka <- exp(absorption)
  ke <- exp(elimination)
  ka*exp(-volume)/(ka-ke)*(exp(-ke*times)-exp(-ka*times))
}
fn3a <- finterp(form)
fn3a(log(c(0.3,0.2,3)))

# Poisson density
y <- rpois(20,5)
fn4 <- function(p) sum(fn5(mu=p))
nlm(fn4,p=1)
# canonical parameter

# likelihood for Poisson log linear regression
y <- rpois(20,fn2a(c(0.2,1,0.4)))
nlm(fn2a,p=c(1,0,0))
# or

# likelihood for Poisson nonlinear regression
y <- rpois(20,fn3(log(c(3,0.3,0.2))))
nlm(fn3,p=log(c(1,0.4,0.1)))

# envir as data objects
# y <- matrix(rnorm(20),ncol=5)
# x1[3,3] <- y[2,2] <- NA
# xx <- tcctomat(x1)
# xx2 <- tcctomat(data.frame(x1,x2))
# z1 <- matrix(rnorm(20),ncol=5)
# z2 <- matrix(rnorm(20),ncol=5)
# z3 <- matrix(rnorm(20),ncol=5)
# zz <- tvctomat(z1)
# zz <- tvctomat(z2,old=zz)
# reps <- rmna(resp, ccov=xx, tvcov=zz)
# reps2 <- rmna(resp, ccov=xx2, tvcov=zz)
# rm(y, x1, x2, z1, z2)
# repeated objects
# time-constant covariates
# Wilkinson and Rogers notation
#form1 <- ~x1
#fn1 <- finterp(form1, .envir=reps)
#fn1(2:3)
#fn1a <- finterp(form1, .envir=xx)
#fn1a(2:3)
#formlb <- ~x1+x2
#fnlb <- finterp(formlb, .envir=reps2)
#fnlb(2:6)
#fnlc <- finterp(formlc, .envir=xx2)
#fnlc(2:6)
# with unknown parameters
#form2 <- ~a+b*x1
#fn2 <- finterp(form2, .envir=reps)
#fn2(2:3)
#fn2a <- finterp(form2, .envir=xx)
#fn2a(2:3)

# time-varying covariates
# Wilkinson and Rogers notation
#form3 <- ~z1+z2
#fn3 <- finterp(form3, .envir=reps)
#fn3(2:4)
#fn3a <- finterp(form3, .envir=zz)
#fn3a(2:4)
# with unknown parameters
#form4 <- ~a+b*z1+c*z2
#fn4 <- finterp(form4, .envir=reps)
#fn4(2:4)
#fn4a <- finterp(form4, .envir=zz)
#fn4a(2:4)

# note: lengths of x1 and z2 differ
# Wilkinson and Rogers notation
#form5 <- ~x1+z2
#fn5 <- finterp(form5, .envir=reps)
#fn5(2:4)
# with unknown parameters
#form6 <- ~a+b*x1+c*z2
#fn6 <- finterp(form6, .envir=reps)
#fn6(2:4)

# with times
# Wilkinson and Rogers notation
#form7 <- ~x1+z2+times
#fn7 <- finterp(form7, .envir=reps)
#fn7(2:5)
#form7a <- ~x1+x2+z2+times
#fn7a <- finterp(form7a, .envir=reps2)
#fn7a(2:8)
# with unknown parameters
The blood flow dataset.

Description
The blood flow dataset.

Usage
data(la)

Format
A data frame with 251 observations on the following 4 variables.

bflow the blood flow measured invasively using radioactively labelled micro-spheres
PET60 is the blood flow measured non-invasively by positron emission tomography using a scan up to 60 seconds
PETother is the blood flow measured non-invasively by positron emission tomography using a scan up to 510 seconds.

Details

The blood flow data were analyzed by Lange et al. (1989), Jones and Faddy (2003) and Rigby and Stasinopoulos (2006). As response variables y the variables PET60, PETother, PET510 can be used representing blood flow measured non-invasively by positron emission tomography using a scan up to 60 seconds (PET60) or 510 second (PET510) respectively. The explanatory variable x is the blood flow measured invasively using radioactively labelled micro-spheres. The distribution of y has previously been modelled by a normal (NO) and a \( t \) family distribution (TF), Lange et al. (1989), and by a skew \( t \) distribution (ST), Jones and Faddy (2003) and by Rigby and Stasinopoulos (2006) using several three and four parameter distributions, including the Box-Cox power exponential (BCPE), Rigby and Stasinopoulos (2004) and the Box-Cox \( t \) distribution (BCT).

References


Examples

data(la)
plot(PET60~bflow,data=la)

nl

Functions to fit nonlinear additive models in GAMLLS

Description

The function nl.obj generate a nonlinear object which can be used to fit a nonlinear additive model within the gamlss algorithm. The function nl takes the nonlinear object created by nl.obj and returns it with several attributes which are used in the function gamlss.nl() which is doing the actual fitting within the backfitting function additive.fit. The actual fit is done by the R function nlm. The function gamlss.nl() is never used on its own.

Usage

nl.obj(formula, start, data)
nl(obj)
gamlss.nl(x, y, w, xeval = NULL)
Arguments

- **formula**: a non-linear formula or function
- **start**: starting values for the parameters in the formula
- **data**: data where the formula can be interpreted
- **obj**: a non-linear object created by `nl.obj`
- **x**: the nl object from `nl`
- **y**: iterative y variable
- **w**: iterative weights
- **xeval**: used in prediction if implemented

Details

The function `gamlss.nl()` is an internal function of GAMLSS allowing the use of the `nlinlm` function to be used within the backfitting cycle of `gamlss`, and should be not used on its own.

Value

The function `nl.obj` returns a non-linear object by using the Jim Lindsey's function `finterp` found in the R package `rmutil`.

The function `nl` returns a vector with values zero to be included in the design matrix but with attributes useful in the fitting the non-linear model.

Author(s)

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

References

- [Jim Lindsey web page](http://popgen.unimaas.nl/~jlindsey/index.html)

See Also

- `nl.gamlss`

Examples

```r
data(la)
nl0<-nl.obj(formula=-bflow*(1-(1-exp(p1))*exp(-p2/bflow)), start=c(-.9, 90), data=la)
mod1<-gamlss(PET~nl(nl0)-1, data=la )
```
Description

This is an auxiliary function used to control the iterations for \texttt{nlgamlss} fitting. Typically only used when calling \texttt{nlgamlss} function with the option \texttt{control}. Since the \texttt{nlgamlss} uses \texttt{nlm} for fitting all of the \texttt{NL.control} argument are passed to \texttt{nlm}.

Usage

\begin{verbatim}
NL.control(fscale = 1L, typsize = NULL, stepmax = NULL, iterlim = 100,
          ndigit = 10, steptol = 1e-05,
          gradtol = 1e-05, print.level = 0, check.analyticals = TRUE,
          hessian = TRUE)
\end{verbatim}

Arguments

- \texttt{fscale} an estimate of the size of log-likelihood at the minimum with default equal 1.
- \texttt{typsize} this argument is passed to \texttt{nlm} and it is an estimate of the size of each parameter at the minimum. If its value is \texttt{NULL} (the default value) the \texttt{typsize} is set within the \texttt{nlgamlss} function to \texttt{typsize} = abs(\texttt{p0}) where \texttt{p0} is the vector containing the starting values of all the parameters to be maximized. \texttt{p0} is defined within \texttt{nlgamlss}
- \texttt{stepmax} this argument is passed to \texttt{nlm} and it is a positive scalar which gives the maximum allowable scaled step length. \texttt{stepmax} is used to prevent steps which would cause the optimization function to overflow, to prevent the algorithm from leaving the area of interest in parameter space, or to detect divergence in the algorithm. \texttt{stepmax} would be chosen small enough to prevent the first two of these occurrences, but should be larger than any anticipated reasonable step. If its value is \texttt{NULL} (the default value) it is defined within \texttt{nlgamlss} as \texttt{stepmax} = sqrt(\texttt{p0} \%\% \texttt{p0})
- \texttt{iterlim} a positive integer specifying the maximum number of iterations to be performed before the program is terminated. The default is 100
- \texttt{ndigit} the number of significant digits in the log-likelihood function. The default is 10
- \texttt{steptol} A positive scalar providing the minimum allowable relative step length. The defaults is 1e-05
- \texttt{gradtol} a positive scalar giving the tolerance at which the scaled gradient is considered close enough to zero to terminate the algorithm. The scaled gradient is a measure of the relative change in log-likelihood in each direction \texttt{'p[i]'} divided by the relative change in \texttt{'p[i]'} . The default is 1e-05
- \texttt{print.level} this argument determines the level of printing which is done during the minimization process. The default value of \texttt{0} means that no printing occurs, a value of \texttt{1} means that initial and final details are printed and a value of \texttt{2} means that full tracing information is printed.
check.analyticals

A logical scalar specifying whether the analytic gradients and Hessians, if they are supplied, should be checked against numerical derivatives at the initial parameter values. This can help detect incorrectly formulated gradients or Hessians.

hessian

If TRUE, the hessian of the log likelihood at the maximum is returned, the default is hessian = TRUE.

Details

See the R function `nlm` and the first two references below for details of the algorithm.

Value

A list with the arguments as components.

Note

This function supports the function `nlgamlss`.

Author(s)

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

References


See Also

`nlgamlss, nlm`
Description

This function is an additional method for fitting GAMLSS models. It is suitable to fit linear or non-linear parametric models using distributions available in the GAMLSS package. It is based on the function stablereg() of the R package stable created by Philippe Lambert and Jim Lindsey which can be found in Jim Lindsey’s web page [http://popgen.unimaas.nl/~jlindsey/index.html](http://popgen.unimaas.nl/~jlindsey/index.html) (see also Lambert, P. and Lindsey, J.K. (1999)). The method is very general but requires starting values for all the parameters. For parametric models it can also be used to give the exact (that is taking into the account the correlation between the distributional parameters) asymptotic standard errors.

Usage

```r
nlgamls(y = NULL, mu.formula = ~1, sigma.formula = ~1,
       mu.formula = ~1, tau.formula = ~1,
       mu.fix = FALSE, sigma.fix = FALSE,
       mu.fix = FALSE, tau.fix = FALSE, all.fix = FALSE,
       mu.start = NULL, sigma.start = NULL,
       mu.start = NULL, tau.start = NULL,
       family = NO(), weights = 1,
       exact = FALSE, delta = 1,
       data = parent.frame(),
       control = NL.control(),
       llik.output = FALSE)
```

Arguments

- **y**: the response variable `y`. Note the difference between `gamlss` and `nlgamss` in declaring the `y` variable In `gamlss`, `y ~ x` is used for modelling the location parameters while here you need `y = y` and `mu = x`.
- **mu.formula**: a formula object for fitting a model to the location `mu` parameter, e.g. `mu.formula = ~ x`. The `mu.formula` is a linear or nonlinear language expression beginning with `~` or an R function, describing the regression function for the predictor of the location parameter (i.e. after transformation of `mu` by its link function). `mu.start` is a vector of initial conditions for the parameters in the regression for this parameter. `mu.fix` is a boolean indicating if an optimization of the likelihood has to be carried out on these parameters. If no optimization is desired on the location parameters `mu`, i.e. when the likelihood has to be evaluated or optimized at a fixed location, then `mu.fix = TRUE` has to be explicitly specified with `mu.start` indicating the fixed value for the predictor of `mu`.
- **sigma.formula**: a formula object for fitting a model to the sigma parameter, as in the `mu.formula` above, e.g. `sigma.formula = ~ x1 + x2`. It can be abbreviated to `sigma.formula = ~ x1 + x2`. 
nu.formula  a formula object for fitting a model to the nu parameter, e.g. nu.formula=~x

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

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

all.fix  whether all the parameters should be fixed at their starting values. This is a way of evaluating the likelihood function

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

family  the distribution family of the gamlss object (see `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 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

exact  If TRUE, fits the exact likelihood function for continuous data by integration over y observation intervals usually determined by the rounding used in the measurement of y, see delta below

delta  Scalar or vector giving the unit of measurement for each response value, set to unity by default. For example, if a response is measured to two decimals, delta=0.01. If the response is transformed, this must be multiplied by the Jacobian. For example, with a log transformation, delta=1/y. The transformation cannot contain unknown parameters. The delta values are used only if exact=TRUE

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

control  this sets the control parameters for the nlm() iterations algorithm. The default setting is the `nlcontrol` function

llik.output  is TRUE when the likelihood has to be displayed at each iteration of the optimization

Value

Returns a nlgamlss object with components

family  the distribution family of the nlgamlss object (see `gamlss.family`)

parameters  the name of the fitted parameters i.e. mu, sigma, nu, tau
call the call of the nlgamllss function
y the response variable
control the nlgamllss 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, discrete or mixture)
method which algorithm is used for the fit, JL() in this case
aic the Akaike information criterion
sbc the Schwatz Bayesian information criterion
df.residual the residual degrees of freedom left after the model is fitted
df.fit the total degrees of freedom use by the model
converged whether the model fitting has have converged as in nlm()
iter the number of iterations as in nlm()
residuals the normalized (randomized) quantile residuals of the model
coefficients all the fitted coefficients of the model
se the standard errors of all the fitted coefficients of the model
cov the covariance matrix of all the fitted coefficients of the model
corr the correlation matrix of all the fitted coefficients 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.link the link function for the mu model, also sigma.link, nu.link, tau.link 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.coeficients the estimated coefficients of the mu model, also sigma.coeficients, nu.coeficients, tau.coeficients for the other parameters if present
mu.coeficients the standard errors of the coefficients of the mu model, also sigma.coeficients, nu.coeficients, tau.coeficients 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
Note

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

Author(s)

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

References

http://popgen.unimaas.nl/~jlindsey/index.html: Jim Lindsey web page


See Also

gamlss, gamlss.family

Examples

data(la)

# fitting a BCPE distribution to the data
modBCPE<- nlgamlss(y=PET60, mu.formula=bflow*(1-(1-exp(p1))*exp(-p2/bflow)),
                 sigma.formula=1, mu.start = c(-.9, 90),
                 sigma.start= -2.3, mu.start=0, tau.start=log(2.5),
                 family=BCPE, data=la)

modBCPE

plot(modBCPE)
Index

*Topic datasets
  la, 8
*Topic package
  gamlss-nl-package, 2
*Topic regression
  finterp, 3
  nl, 9
  NL.control, 11
  nlgamlss, 13
  finterp, 3
  fmobj(finterp), 3

gamlss, 2, 16
gamlss-nl(gamlss-nl-package), 2
gamlss-nl-package, 2
gamlss.family, 14, 16
gamlss.nl, 9
gamlss.nl(nl), 9

la, 8

nl, 9
NL.control, 11
nlgamlss, 10, 12, 13
nlm, 12