

# Package ‘gamlss.nl’

November 22, 2009

**Type** Package

**Title** Fitting non linear parametric GAMLSS models

**Version** 3.1-0

**Date** 2009-11-22

**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.com/>

**Repository** CRAN

**Date/Publication** 2009-11-22 16:17:26

## R topics documented:

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

<b>Index</b>	<b>17</b>
--------------	-----------

---

gamlss-nl-package *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

Rigby, R. A. and Stasinopoulos D. M. (2005). Generalized additive models for location, scale and shape, (with discussion), *Appl. Statist.*, **54**, part 3, pp 507-554.

Stasinopoulos D. M., Rigby R.A. and Akantziliotou C. (2003) Instructions on how to use the GAMLSS package in R. Accompanying documentation in the current GAMLSS help files, (see also <http://www.gamlss.com/>).

## See Also

[gamlss](#)

## Examples

```
data(la)
# fitting the Johnson's Su distribution to the data
modJSU <- nlgamlss(y=PET60, mu.fo= ~bflow*(1-p1*exp(-p2/bflow)), sigma.formula=~1,
                  nu.fo=~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)
```

## 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 `nlgam1ss`.

`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 `fmbobj` 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

```
finterp.default(.z, .envir = parent.frame(), .formula = FALSE,
               .vector = TRUE, .args = NULL, .start = 1,
               .name = NULL, .expand = TRUE, .intercept = TRUE,
               .old = NULL, .response = FALSE, ...)
finterp(.z, ...)
fmbobj(z, envir = parent.frame())
```

## Arguments

- |                       |   |
|-----------------------|---|
| <code>.z</code>       | A model formula beginning with <code>\~</code> , 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 |
| <code>.envir</code>   | The environment in which the formula is to be interpreted or a data object of class, 'repeated', 'tccov', or 'tvcov'.   |
| <code>.formula</code> | If TRUE and the formula is in Wilkinson and Rogers notation, just returns the formula.  |

<code>.vector</code>	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 <code>'args'</code> . Always TRUE if <code>'envir'</code> is a data object.
<code>.args</code>	If <code>'vector'</code> is TRUE, names of parameters that are to be function arguments and not included in the vector.
<code>.start</code>	The starting index value of the parameter vector in the function returned when <code>'vector'</code> is TRUE.
<code>.name</code>	Character string giving the name of the data object specified by <code>'envir'</code> . Ignored unless the latter is such an object and only necessary when <code>'finterp'</code> is called within other functions.
<code>.expand</code>	If TRUE, expand functions with only time-constant covariates to return one value per observation instead of one value per individual. Ignored unless <code>'envir'</code> is an object of class, <code>'repeated'</code> .
<code>.intercept</code>	If W&R notation is supplied and <code>'intercept=F'</code> , a model function without intercept is returned.
<code>.old</code>	The name of an existing object of class <code>'formulafn'</code> which has common parameters with the one being created, or a list of such objects. Only used if <code>'vector'=TRUE</code> . The value of <code>'start'</code> should ensure that there is no conflict in indexing the vector.
<code>.response</code>	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.
<code>z</code>	A model formula beginning with <code>~</code> , either in Wilkinson and Rogers notation or containing unknown parameters.
<code>envir</code>	The environment in which the formula is to be interpreted.
<code>...</code>	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 `'fmobj'`, 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

### References

<http://popgen.unimaas.nl/~jlindsey/index.html>:JimLindseywebpage

**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 <- finterp(~mu^y*exp(-mu)/gamma(y+1))
fn4
fn4(5)
dpois(y,5)

```

```

#
# Poisson likelihood
# mean parameter
fn5 <- finterp(~-y*log(mu)+mu+lgamma(y+1),.vector=FALSE)
fn5
likefn1 <- function(p) sum(fn5(mu=p))
nlm(likefn1,p=1)
mean(y)
# canonical parameter
fn5a <- finterp(~-y*theta+exp(theta)+lgamma(y+1),.vector=FALSE)
fn5a
likefn1a <- function(p) sum(fn5a(theta=p))
nlm(likefn1a,p=1)
#
# likelihood for Poisson log linear regression
y <- rpois(20,fn2a(c(0.2,1,0.4)))
nlm(likefn1,p=1)
mean(y)
likefn2 <- function(p) sum(fn5(mu=fn2a(p)))
nlm(likefn2,p=c(1,0,0))
# or
likefn2a <- function(p) sum(fn5a(theta=fn2(p)))
nlm(likefn2a,p=c(1,0,0))
#
# likelihood for Poisson nonlinear regression
y <- rpois(20,fn3(log(c(3,0.3,0.2))))
nlm(likefn1,p=1)
mean(y)
likefn3 <- function(p) sum(fn5(mu=fn3(p)))
nlm(likefn3,p=log(c(1,0.4,0.1)))
#
# envir as data objects
# y <- matrix(rnorm(20),ncol=5)
#y[3,3] <- y[2,2] <- NA
#x1 <- 1:4
#x2 <- c("a","b","c","d")
#resp <- restovec(y)
#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

```

```

#print(fn1 <- finterp(form1, .envir=reprs))
#fn1(2:3)
#print(fn1a <- finterp(form1, .envir=xx))
#fn1a(2:3)
#form1b <- ~x1+x2
#print(fn1b <- finterp(form1b, .envir=reprs2))
#fn1b(2:6)
#print(fn1c <- finterp(form1b, .envir=xx2))
#fn1c(2:6)
# with unknown parameters
#form2 <- ~a+b*x1
#print(fn2 <- finterp(form2, .envir=reprs))
#fn2(2:3)
#print(fn2a <- finterp(form2, .envir=xx))
#fn2a(2:3)
#
# time-varying covariates
# Wilkinson and Rogers notation
#form3 <- ~z1+z2
#print(fn3 <- finterp(form3, .envir=reprs))
#fn3(2:4)
#print(fn3a <- finterp(form3, .envir=zz))
#fn3a(2:4)
# with unknown parameters
#form4 <- ~a+b*z1+c*z2
#print(fn4 <- finterp(form4, .envir=reprs))
#fn4(2:4)
#print(fn4a <- finterp(form4, .envir=zz))
#fn4a(2:4)
#
# note: lengths of x1 and z2 differ
# Wilkinson and Rogers notation
#form5 <- ~x1+z2
#print(fn5 <- finterp(form5, .envir=reprs))
#fn5(2:4)
# with unknown parameters
#form6 <- ~a+b*x1+c*z2
#print(fn6 <- finterp(form6, .envir=reprs))
#fn6(2:4)
#
# with times
# Wilkinson and Rogers notation
#form7 <- ~x1+z2+times
#print(fn7 <- finterp(form7, .envir=reprs))
#fn7(2:5)
#form7a <- ~x1+x2+z2+times
#print(fn7a <- finterp(form7a, .envir=reprs2))
#fn7a(2:8)
# with unknown parameters
#form8 <- ~a+b*x1+c*z2+e*times
#print(fn8 <- finterp(form8, .envir=reprs))
#fn8(2:5)
#

```

```

# with a variable not in the data object
#form9 <- ~a+b*z1+c*z2+e*z3
#print(fn9 <- finterp(form9, .envir=reps))
#fn9(2:5)
# z3 assumed to be an unknown parameter:
#fn9(2:6)
#
# multiline formula
#form10 <- ~{
#     tmp <- exp(b)
#     a+tmp*z1+c*z2+d*times}
#print(fn10 <- finterp(form10, .envir=reps))
#fn10(2:5)
# for fmobj
x1 <- rpois(20,2)
x2 <- rnorm(20)
x3 <- gl(2,10)
#
# W&R formula
fmobj(~x1+x2+x3)
#
# formula with unknowns
fmobj(~b0+b1*x1+b2*x2)
#
# nonlinear formulae with unknowns
# log link
fmobj(~exp(b0+b1*x1+b2*x2))

```

---

la

*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** ?

**PET510** 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  $t$  diastribution (BCT) .

## References

- Jones, M. C. and Faddy, M. J. (2003). A skew extension of the  $t$  distribution, with applications. *J. Roy. Statist. Soc B*, **65**, 159-174.
- Lange, K. L., Little, R. J. A. and Taylor, J. M. G. (1989). Robust statistical modelling using the  $t$  distribution. *J. Am. Statist. Ass.*, **84**, 881-896.
- Rigby, R.A. Stasinopoulos, D.M. (2006). Using the Box-Cox  $t$  distribution in GAMLSS to mode skewnees and and kurtosis. to appear in *Statistical Modelling*.

## Examples

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

---

 nl

---

*Functions to fit nonlinear additive models in GAMLSS*


---

## Description

The function `nl.obs` 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.obs` 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**

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

**Details**

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

**Value**

The function `nl.obs` 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**

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

Rigby, R. A. and Stasinopoulos D. M. (2005). Generalized additive models for location, scale and shape,(with discussion), *Appl. Statist.*, **54**, part 3, pp 507-554.

Stasinopoulos D. M., Rigby R.A. and Akantziliotou C. (2003) Instructions on how to use the GAMLSS package in R. Accompanying documentation in the current GAMLSS help files, (see also <http://www.gamlss.com/>).

**See Also**

[nlgamlss](#)

**Examples**

```
data(la)
nlo<-nl.obj(formula=~bflow*(1-(1-exp(p1))*exp(-p2/bflow)), start=c(-.9, 90), data=la)
mod1<-gamlss(PET60~nl(nlo)-1, data=la )
```

NL.control

*Auxiliary for Controlling non linear GAMLSS Fitting***Description**

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

**Usage**

```
NL.control(fscale = 1,  typsize = NULL, stepmax = NULL, iterlim = 100,
          ndigit = 10, steptol = 1e-05,
          gradtol = 1e-05, print.level = 0, check.analyticals = TRUE,
          hessian = TRUE)
```

**Arguments**

<code>fscale</code>	an estimate of the size of log-likelihood at the minimum with default equal 1.
<code>typsize</code>	this argument is passed to <code>nlm</code> and it is an estimate of the size of each parameter at the minimum. If its value is <code>NULL</code> (the default value) the <code>typsize</code> is set within the <code>nlgamlss</code> function to <code>typsize=abs(p0)</code> where <code>p0</code> is the vector containing the starting values of all the parameters to be maximized. <code>p0</code> is defined within <code>nlgamlss</code>
<code>stepmax</code>	this argument is passed to <code>nlm</code> and it is a positive scalar which gives the maximum allowable scaled step length. <code>stepmax</code> 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. <code>stepmax</code> 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 <code>NULL</code> (the default value) it is defined within <code>nlgamlss</code> as <code>stepmax=sqrt(p0 %*% p0)</code>
<code>iterlim</code>	a positive integer specifying the maximum number of iterations to be performed before the program is terminated. The default is 100
<code>ndigit</code>	the number of significant digits in the log-likelihood function. The default is 10
<code>steptol</code>	A positive scalar providing the minimum allowable relative step length. The default is 1e-05
<code>gradtol</code>	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 ' <code>p[i]</code> ' divided by the relative change in ' <code>p[i]</code> '. The default is 1e-05
<code>print.level</code>	this argument determines the level of printing which is done during the minimization process. The default value of '0' means that no printing occurs, a value of '1' means that initial and final details are printed and a value of 2 means that full tracing information is printed.

<code>check.analyticals</code>	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.
<code>hessian</code>	if TRUE, the hessian of the log likelihood at the maximum is returned ,the default is <code>hessian=TRUE</code>

### 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 functions supports the function `nlgamlss`

### Author(s)

Mikis Stasinopoulos <[d.stasinopoulos@londonmet.ac.uk](mailto:d.stasinopoulos@londonmet.ac.uk)>, Bob Rigby <[r.rigby@londonmet.ac.uk](mailto:r.rigby@londonmet.ac.uk)>

### References

Dennis, J. E. and Schnabel, R. B. (1983) *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*. Prentice-Hall, Englewood Cliffs, NJ.

Schnabel, R. B., Koontz, J. E. and Weiss, B. E. (1985) A modular system of algorithms for unconstrained minimization. *ACM Trans. Math. Software*, **11**, 419-440.

Rigby, R. A. and Stasinopoulos D. M. (2005). Generalized additive models for location, scale and shape,(with discussion), *Appl. Statist.*, **54**, part 3, pp 507-554.

Stasinopoulos D. M., Rigby R.A. and Akantziliotou C. (2003) Instructions on how to use the GAMLSS package in R. Accompanying documentation in the current GAMLSS help files, (see also <http://www.gamlss.com/>).

### See Also

[nlgamlss](#), [nlm](#)

nlgamlss

*Fitting non linear Generalized Additive Models for Location Scale and Shape (GAMLSS)*

## Description

This function is an additional method for fitting GAMLSS models. It 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> (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

```
nlgamlss(y = NULL, mu.formula = ~1, sigma.formula = ~1,
         nu.formula = ~1, tau.formula = ~1,
         mu.fix = FALSE, sigma.fix = FALSE,
         nu.fix = FALSE, tau.fix = FALSE, all.fix = FALSE,
         mu.start = NULL, sigma.start = NULL,
         nu.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`,  $y \sim x$  is used for modelling the location parameters while here you need  $y=y$  and  $\mu . fo=x$
- `mu.formula` a formula object for fitting a model to the location  $\mu$  parameter, e.g. `mu . fo=~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.fo=~x1+x2`.

<code>nu.formula</code>	a formula object for fitting a model to the nu parameter, e.g. <code>nu.fo=~x</code>
<code>tau.formula</code>	a formula object for fitting a model to the tau parameter, e.g. <code>tau.fo=~x</code>
<code>mu.fix</code>	whether the mu parameter should be kept fixed in the fitting processes e.g. <code>mu.fix=FALSE</code>
<code>sigma.fix</code>	whether the sigma parameter should be kept fixed in the fitting processes e.g. <code>sigma.fix=FALSE</code>
<code>nu.fix</code>	whether the nu parameter should be kept fixed in the fitting processes e.g. <code>nu.fix=FALSE</code>
<code>tau.fix</code>	whether the tau parameter should be kept fixed in the fitting processes e.g. <code>tau.fix=FALSE</code>
<code>all.fix</code>	whether all the parameters should be fixed at their starting values. This is a way of evaluating the likelihood function
<code>mu.start</code>	vector or scalar of initial values for the location parameter mu e.g. <code>mu.start=4</code>
<code>sigma.start</code>	vector or scalar of initial values for the scale parameter sigma e.g. <code>sigma.start=1</code>
<code>nu.start</code>	vector or scalar of initial values for the parameter nu e.g. <code>nu.start=3</code>
<code>tau.start</code>	vector or scalar of initial values for the location parameter tau e.g. <code>tau.start=2</code>
<code>family</code>	the distribution family of the gamlss object (see <a href="#">gamlss.family</a> )
<code>weights</code>	a vector of weights. Here weights can be used to weight out observations (like in <code>subset</code> ) or for a weighted likelihood analysis where the contribution of the observations to the likelihood differs according to <code>weights</code> . The length of <code>weights</code> 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 <code>w</code> use <code>weights=w</code>
<code>exact</code>	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 <code>delta</code> below
<code>delta</code>	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, <code>delta=0.01</code> . If the response is transformed, this must be multiplied by the Jacobian. For example, with a log transformation, <code>delta=1/y</code> . The transformation cannot contain unknown parameters. The delta values are used only if <code>exact=TRUE</code>
<code>data</code>	a data frame containing the variables occurring in the formula. If this is missing, the variables should be on the search list. e.g. <code>data=aids</code>
<code>control</code>	this sets the control parameters for the <code>nlm()</code> iterations algorithm. The default setting is the <code>NL.control</code> function
<code>llik.output</code>	is TRUE when the likelihood has to be displayed at each iteration of the optimization

### Value

Returns a `nlgamlss` object with components

<code>family</code>	the distribution family of the <code>nlgamlss</code> object (see <a href="#">gamlss.family</a> )
<code>parameters</code>	the name of the fitted parameters i.e. <code>mu</code> , <code>sigma</code> , <code>nu</code> , <code>tau</code>

<code>call</code>	the call of the <code>nlgamlss</code> function
<code>y</code>	the response variable
<code>control</code>	the <code>nlgamlss</code> fit control settings
<code>weights</code>	the vector of weights
<code>G.deviance</code>	the global deviance
<code>N</code>	the number of observations in the fit
<code>rqres</code>	a function to calculate the normalized (randomized) quantile residuals of the object
<code>iter</code>	the number of external iterations in the fitting process
<code>type</code>	the type of the distribution or the response variable (continuous , discrete or mixture)
<code>method</code>	which algorithm is used for the fit, <code>JL()</code> in this case
<code>aic</code>	the Akaike information criterion
<code>sbc</code>	the Schwatz Bayesian information criterion
<code>df.residual</code>	the residual degrees of freedom left after the model is fitted
<code>df.fit</code>	the total degrees of freedom use by the model
<code>converged</code>	whether the model fitting has have converged as in <code>nlm()</code>
<code>iter</code>	the number of iterations as in <code>nlm()</code>
<code>residuals</code>	the normalized (randomized) quantile residuals of the model
<code>coefficients</code>	all the fitted coefficients of the model
<code>se</code>	the standard errors of all the fitted coefficients of the model
<code>cov</code>	the covariance matrix of all the fitted coefficients of the model
<code>corr</code>	the correlation matrix of all the fitted coefficients of the model
<code>mu.fv</code>	the fitted values of the mu model, also <code>sigma.fv</code> , <code>nu.fv</code> , <code>tau.fv</code> for the other parameters if present
<code>mu.lp</code>	the linear predictor of the mu model, also <code>sigma.lp</code> , <code>nu.lp</code> , <code>tau.lp</code> for the other parameters if present
<code>mu.link</code>	the link function for the mu model, also <code>sigma.link</code> , <code>nu.link</code> , <code>tau.link</code> for the other parameters if present
<code>mu.formula</code>	the formula for the mu model, also <code>sigma.formula</code> , <code>nu.formula</code> , <code>tau.formula</code> for the other parameters if present
<code>mu.coefficients</code>	the estimated coefficients of the mu model, also <code>sigma.coefficients</code> , <code>nu.coefficients</code> , <code>tau.coefficients</code> for the other parameters if present
<code>mu.coefficients</code>	the standard errors of the coefficients of the mu model, also <code>sigma.coefficients</code> , <code>nu.coefficients</code> , <code>tau.coefficients</code> for the other parameters if present
<code>mu.df</code>	the mu degrees of freedom also <code>sigma.df</code> , <code>nu.df</code> , <code>tau.df</code> 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
- Lambert, P. and Lindsey, J.K. (1999) Analysing financial returns using regression models based on non-symmetric stable distributions. *Applied Statistics* **48**, 409-424.
- Rigby, R. A. and Stasinopoulos D. M. (2005). Generalized additive models for location, scale and shape,(with discussion), *Appl. Statist.*, **54**, part 3, pp 507-554.
- Rigby, R.A. Stasinopoulos, D.M. (2006). Using the Box-Cox  $t$  distribution in GAMLSS to mode skewnees and and kurtosis. to appear in *Statistical Modelling*.
- Stasinopoulos D. M., Rigby R.A. and Akantziliotou C. (2006) Instructions on how to use the GAMLSS package in R. Accompanying documentation in the current GAMLSS help files, (see also <http://www.gamlss.com/>).

**See Also**

[gamlss](#), [gamlss.family](#)

**Examples**

```
data(la)
# fitting a BCPE distribtion to the data
modBCPE<- nlgamlss(y=PET60, mu.fo=~bflow*(1-(1-exp(p1))*exp(-p2/bflow)),
                  sigma.formula=~1, mu.start = c(-.9, 90),
                  sigma.start= -2.3, nu.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