Package: gamlss2 (via r-universe)

September 17, 2024

Title GAMLSS Infrastructure for Flexible Distributional Regression

Version 0.1-0

Date 2024-09-15

Description Next generation infrastructure for generalized additive models for location, scale, and shape (GAMLSS) and distributional regression more generally. The package provides a fresh reimplementaton of the classic 'gamlss' package while being more modular and facilitating the creation of advanced terms and models.

License GPL-2 | GPL-3

URL https://gamlss-dev.github.io/gamlss2/

BugReports https://github.com/gamlss-dev/gamlss2/issues

Depends R (>= 4.1.0), gamlss.dist, mgcv

Imports parallel, Formula

Suggests gamlss, gamlss.data, colorspace, knitr, scoringRules, partykit, Matrix, nnet, lattice, rpart, distributions3, nlme

LazyLoad yes

VignetteBuilder knitr

Repository https://gamlss-dev.r-universe.dev

RemoteUrl https://github.com/gamlss-dev/gamlss2

RemoteRef HEAD

RemoteSha f8c26836967a5892a2280f9b9de16709648680f5

Contents

`gamlss2-package` *GAMLSS Modeling with Advanced Flexible Infrastructures*

Description

Next generation infrastructure for generalized additive models for location, scale, and shape (GAMLSS) and distributional regression more generally. The package provides a fresh reimplementaton of the classic 'gamlss' package while being more modular and facilitating the creation of advanced terms and models.

Details

The primary purpose of this package is to facilitate the creation of advanced infrastructures designed to enhance the Generalized Additive Models for Location Scale and Shape (GAMLSS, Rigby and Stasinopoulos 2005) modeling framework. Notably, the **gamlss2** package represents a significant overhaul of its predecessor, **gamlss**, with a key emphasis on improving estimation speed and incorporating more adaptable infrastructures. These enhancements enable the seamless integration of various algorithms into GAMLSS, including gradient boosting, Bayesian estimation, regression trees, and forests, fostering a more versatile and powerful modeling environment.

Moreover, the package expands its compatibility by supporting all model terms from the base R **mgcv** package. Additionally, the **gamlss2** package introduces the capability to accommodate more than four parameter families. Essentially, this means that users can now specify any type of model using these new infrastructures, making the package highly flexible and accommodating to a wide range of modeling requirements.

Index: This package was not yet installed at build time.

Author(s)

Mikis Stasinopoulos [aut, cph] (<https://orcid.org/0000-0003-2407-5704>), Robert Rigby [aut] (<https://orcid.org/0000-0003-3853-1707>), Nikolaus Umlauf [aut, cre] (<https://orcid.org/0000-0003-2160-9803>), Achim Zeileis [ctb] (<https://orcid.org/0000-0003-0918-3766>), Reto Stauffer [aut] (<https://orcid.org/0000-0002-3798-5507>)

Maintainer: Nikolaus Umlauf <Nikolaus.Umlauf@uibk.ac.at>

References

Rigby RA, Stasinopoulos DM (2005). "Generalized Additive Models for Location, Scale and Shape (with Discussion)." *Journal of the Royal Statistical Society, Series C (Applied Statistics)*, **54**, 507–554. doi:10.1111/j.14679876.2005.00510.x

Rigby RA, Stasinopoulos DM, Heller GZ, De Bastiani F (2019). *Distributions for Modeling Location, Scale, and Shape: Using GAMLSS in R*, Chapman and Hall/CRC. doi:10.1201/9780429298547

Stasinopoulos DM, Rigby RA (2007). "Generalized Additive Models for Location Scale and Shape (GAMLSS) in R." *Journal of Statistical Software*, **23**(7), 1–46. doi:10.18637/jss.v023.i07

Stasinopoulos DM, Rigby RA, Heller GZ, Voudouris V, De Bastiani F (2017). *Flexible Regression and Smoothing: Using GAMLSS in R*, Chapman and Hall/CRC. doi:10.1201/b21973

See Also

gamlss2, fake_formula

---

fake_formula *Extended Processing of "Fake" Formulas*

Description

Create a "fake" formula from a formula, a Formula, or a list of formulas. The function extracts all necessary variables (transformation of variables), to build a model.frame. The function also extracts all special model terms within the formulas, the information can be used to setup any special model term specification list.

Usage

```
fake_formula(formula, specials = NULL,
  nospecials = FALSE, onlyspecials = FALSE)
```

Arguments

formula	A formula, Formula, or a list of formulas.
specials	Character, vector of names of special functions in the formula, see terms.formula.
nospecials	Logical, should variables of special model terms be part of the "fake formula"?
onlyspecials	Logical, should only the special model terms be returned?

Value

Depending on the input formula, the function returns a formula or Formula. If onlyspecials = TRUE a vector or list of special model term names is returned.

See Also

gamlss2

Examples

```
## basic formula
f <- y ~ x1 + x2 + log(x3)
ff <- fake_formula(f)
print(ff)

## including special model terms
f <- y ~ x1 + s(x2) + x3 + te(log(x3), x4)
ff <- fake_formula(f)
print(ff)

## multiple parts on the right-hand side
f <- y ~ x1 + s(x2) + x3 + te(log(x3), x4) | x2 + sqrt(x5)
ff <- fake_formula(f)
print(ff)

## collapse all formula parts
print(formula(ff, collapse = TRUE))
print(formula(ff, collapse = TRUE, update = TRUE))

## list of formulas
f <- list(
  y ~ x1 + s(x2) + x3 + te(log(x3), x4),
    ~ x2 + sqrt(x5),
    ~ z2 + x1 + exp(x3)
)
ff <- fake_formula(f)
print(ff)

## extract separate parts on the right-hand side
formula(ff, rhs = 1)
formula(ff, rhs = 2)
formula(ff, rhs = 3)

## formula with multiple responses and multiple parts
f <- y1 | y2 | y3 ~ x1 + s(x2) + x3 + te(log(x3), x4) | x2 + ti(x5)
ff <- fake_formula(f)
print(ff)

## list of formulas with multiple responses
f <- list(
  y1 ~ x1 + s(x2) + x3 + te(log(x3), x4),
  y2 ~ x2 + sqrt(x5),
```

```
  y3 ~ z2 + x1 + exp(x3) + s(x10)
)
ff <- fake_formula(f)

## extract only without special terms
ff <- fake_formula(f, nospecials = TRUE)
print(ff)

## extract only special terms
ff <- fake_formula(f, onlyspecials = TRUE)
print(ff)
```

`find_family` *Find and Fit GAMLSS Families*

Description

These functions provide useful infrastructures for finding suitable GAMLSS families for a response variable.

Usage

```
## List of available families from gamlss.dist package.
available_families(type = c("continuous", "discrete"), families = NULL)

## Find suitable response distribution.
find_family(y, families = NULL, k = 2, verbose = TRUE, ...)

## Fit distribution parameters.
fit_family(y, family = NO, plot = TRUE, ...)
```

Arguments

`type`	Character, is the reponse continuous or discrete?
`families`	Character, the names of the family objects of the **gamlss.dist** package that should be returned.
`y`	The reponse vector or matrix.
`k`	Numeric, the penalty factor that should be used for the `GAIC`.
`verbose`	Logical, should runtime information be printed?
`family`	A famnily object that should be used for estimation, see also `gamlss2.family`.
`plot`	Logical, should a plot of the fitted density be provided?
`...`	Further arguments to be passed to `gamlss2` when using `find_family()`, or arguments `legend = TRUE/FALSE`, `pos = "topright"` (see also function `legend`), `main`, `xlab` and `ylab` when argument `plot = TRUE` using function `fit_family()`.

Details

The function `find_family()` employs `gamlss2` to estimate intercept-only models for each specified family object in the `families` argument. Note that model estimation occurs within a `try` block with warnings suppressed. Additionally, the function calculates the `GAIC` for each family whenever feasible and returns the sorted values in descending order.

Function `fit_family()` fits a single intercept-only model using the specified family and creates a plot of the fitted density.

Value

Function `find_family()` returns a vector of `GAIC` values for the different fitted families. Function `fit_family()` returns the fitted intercept-only model.

See Also

`gamlss2`.

Examples

```
## Not run: ## load data
data("rent", package = "gamlss.data")

## find a suitable response to the response
ic <- find_family(rent$R)
print(ic)

## fit parameters using the BCCG family
fit_family(rent$R, family = BCCG)

## count data
data("polio", package = "gamlss.data")

## search best count model
ic <- find_family(polio, k = 0,
  families = available_families(type = "discrete"))
print(ic)

## fit parameters using the ZASICHEL family
fit_family(polio, family = ZASICHEL)

## End(Not run)
```

Description

Estimation of generalized additive models for location scale and shape (GAMLSS). The model fitting function `gamlss2()` provides flexible infrastructures to estimate the parameters of a response distribution. The number of distributional parameters is not fixed, see `gamlss2.family`. Moreover, `gamlss2()` supports all smooth term constructors from the **mgcv** package in addition to the classical model terms as provided by **gamlss** and `gamlss.add`.

Usage

```
gamlss2(x, ...)

## S3 method for class 'formula'
gamlss2(formula, data, family = NO,
  subset, na.action, weights, offset, start = NULL,
  control = gamlss2_control(...), ...)

## S3 method for class 'list'
gamlss2(x, ...)
```

Arguments

`formula`	A GAM-type `formula` or `Formula`. All smooth terms of the **mgcv** package are supported, see also `formula.gam`.
`x`	For `gamlss.list()` `x` is a list of `formula`s.
`data`	A data frame or list or environment containing the variables in the model. If not found in `data`, the variables are taken from `environment(formula)`, typically the environment from which `gamlss2` is called.
`family`	A `gamlss.family` or `gamlss2.family` object used to define distribution and the link functions of the parameters.
`subset`	An optional vector specifying a subset of observations to be used in the fitting process.
`na.action`	`NA` processing for setting up the `model.frame`.
`weights`	An optional vector of prior `weights` to be used in the fitting process. Should be `NULL` or a numeric vector.
`offset`	This can be used to specify an *a priori* known components to be included in the linear predictors during fitting. Please note that if only a single numeric vector is provided, the offset will be assigned to the first specified parameter of the distribution. In the case of multiple offsets, a data frame or list must be supplied. Each offset is assigned in the same order as the parameters of the distribution specified in the family object.
`start`	Starting values for estimation algorithms.
`control`	A list of control arguments, see `gamlss2_control`.
`...`	Arguments passed to `gamlss2_control`.

Details

The model fitting function `gamlss2()` provides flexible infrastructures for the estimation of GAMLSS.

- Distributional models are specified using family objects, either from the **gamlss.dist** package or using `gamlss2.family` objects.
- Estimation is carried out through a Newton-Raphson/Fisher scoring algorithm, see function `RS`. The estimation algorithms can also be exchanged using `gamlss2_control`. Additionally, if an `optimizer` is specified by the family object, this optimizer function will be employed for estimation.
- The return value is determined by the object returned from the optimizer function, typically an object of class `"gamlss2"`. Default methods and extractor functions are available for this class. Nevertheless, users have the flexibility to supply their own optimizer function, along with user-specific methods tailored for the returned object.

Value

The return value is determined by the object returned from the optimizer function. By default, the optimization is performed using the `RS` optimizer function (see `gamlss2_control`), yielding an object of class `"gamlss2"`. Default methods and extractor functions are available for this class.

References

Rigby RA, Stasinopoulos DM (2005). "Generalized Additive Models for Location, Scale and Shape (with Discussion)." *Journal of the Royal Statistical Society, Series C (Applied Statistics)*, **54**, 507–554. doi:10.1111/j.14679876.2005.00510.x

Rigby RA, Stasinopoulos DM, Heller GZ, De Bastiani F (2019). *Distributions for Modeling Location, Scale, and Shape: Using GAMLSS in R*, Chapman and Hall/CRC. doi:10.1201/9780429298547

Stasinopoulos DM, Rigby RA (2007). "Generalized Additive Models for Location Scale and Shape (GAMLSS) in R." *Journal of Statistical Software*, **23**(7), 1–46. doi:10.18637/jss.v023.i07

Stasinopoulos DM, Rigby RA, Heller GZ, Voudouris V, De Bastiani F (2017). *Flexible Regression and Smoothing: Using GAMLSS in R*, Chapman and Hall/CRC. doi:10.1201/b21973

See Also

`RS`, `gamlss2_control`, `gamlss2.family`

Examples

```
## load the abdominal circumference data
data("abdom", package = "gamlss.data")

## specify the model Formula
f <- y ~ s(x) | s(x) | s(x) | s(x)

## estimate model
b <- gamlss2(f, data = abdom, family = BCT)

## model summary
summary(b)
```

```
## plot estimated effects
plot(b, which = "effects")

## plot diagnostics
plot(b, which = "resid")

## predict parameters
par <- predict(b)

## predict quantiles
pq <- sapply(c(0.05, 0.5, 0.95), function(q) family(b)$q(q, par))

## visualize
plot(y ~ x, data = abdom, pch = 19,
  col = rgb(0.1, 0.1, 0.1, alpha = 0.3))
matplot(abdom$x, pq, type = "l", lwd = 2,
  lty = 1, col = 4, add = TRUE)

## use of starting values
m <- gamlss2(f, data = abdom, family = BCT,
  start = c(mu = 200, sigma = 0.1, nu = 0, tau = 10))

## fix some parameters
m <- gamlss2(f, data = abdom, family = BCT,
  start = c(mu = 200, sigma = 0.1, nu = 0, tau = 10),
  fixed = c(nu = TRUE, tau = TRUE))

## estimated coefficients (intercepts)
coef(m)

## starting values using full predictors
m <- gamlss2(f, data = abdom, family = BCT,
  start = fitted(m))

## same with
m <- gamlss2(f, data = abdom, family = BCT,
  start = m)
```

`gamlss2.family` *Family Objects in* **gamlss2**

Description

Family objects within the package **gamlss2** are used to specify the information required to use a model fitting function. This includes details such as parameter names, corresponding link functions, the density function, log-likelihood function and derivatives of the log-likelihood with respect to the predictors. In addition, family objects are used in the calculation of post-modeling statistics, such as residual diagnostics and random number generation. An overview can be found in the accompanying details and examples.

Details

The following lists the minimum requirements on a **gamlss2** family object to be used with `gamlss2`:

- The family object is expected to return a `list` of class `"gamlss2.family"`.
- The object must contain the family name as a character string.
- The object must contain the names of the parameters as a character string, as well as the corresponding link functions as character string.
- The family object must contain a `$d()` function to evaluate the (log-)density.

Furthermore, it is assumed that the density function in a family object has the following arguments:

`d(y, par, log = FALSE, ...)`

where argument `y` is the response (possibly a matrix) and `par` is a named list holding the evaluated parameters of the distribution, e.g., using a normal distribution `par` has two elements, one for the mean `par$mu` and one for the standard deviation `par$sigma`. The dots argument is for passing special internally used objects, depending on the type of model this feature is usually not needed.

Optionally, the family object holds derivative functions evaluating derivatives of the log-likelihood w.r.t. the predictors (or expectations of derivatives). For each parameter, these functions must have the following arguments:

`function(y, par, ...)`

for computing the first derivative of the log-likelihood w.r.t. a predictor and

`function(y, par, ...)`

for computing the _negative_ second derivatives. Within the family object these functions are organized in a named list, see the examples below. If these functions are not specified, all derivatives will be approximated numerically. Note that also cross derivatives can be implemented, e.g., when using the `CG` algorithm for fitting a GAMLSS.

In addition, for the cumulative distribution function (`p(y, par, ...)`), for the quantile function (`q(y, par, ...)`) or for creating random numbers (`r(n, par, ...)`) the same structure is assumed.

Using function `gamlss2` the family objects may also specify the `optimizer()`er function that should be used with this family.

See Also

`gamlss2`

Examples

```
## Not run: ## new family object for the normal distribution
Normal <- function(...) {
  fam <- list(
    "family" = "Normal",
    "names" = c("mu", "sigma"),
    "links" = c("mu" = "identity", "sigma" = "log"),
    "score" = list(
      "mu" = function(y, par, ...) {
        (y - par$mu) / (par$sigma^2)
      },
```

```
      "sigma" = function(y, par, ...) {
        -1 + (y - par$mu)^2 / (par$sigma^2)
      }
    ),
    "hess" = list(
      "mu" = function(y, par, ...) {
        1 / (par$sigma^2)
      },
      "sigma" = function(y, par, ...) {
        rep(2, length(y))
      },
      "mu.sigma" = function(y, par, ...) {
        rep(0, length(y))
      }
    ),
    "loglik" = function(y, par, ...) {
      sum(dnorm(y, par$mu, par$sigma, log = TRUE))
    },
    "mu" = function(par, ...) {
      par$mu
    },
    "d" = function(y, par, log = FALSE) {
      dnorm(y, mean = par$mu, sd = par$sigma, log = log)
    },
    "p" = function(y, par, ...) {
      pnorm(y, mean = par$mu, sd = par$sigma, ...)
    },
    "r" = function(n, par) {
      rnorm(n, mean = par$mu, sd = par$sigma)
    },
    "q" = function(p, par) {
      qnorm(p, mean = par$mu, sd = par$sigma)
    },
    "initialize" = list(
      "mu"    = function(y, ...) { (y + mean(y)) / 2 },
      "sigma" = function(y, ...) { rep(sd(y), length(y)) }
    ),
    "mean"      = function(par) par$mu,
    "variance"  = function(par) par$sigma^2,
    "valid.response" = function(x) {
      if(is.factor(x) | is.character(x))
        stop("the response should be numeric!")
      return(TRUE)
    }
  )

  class(fam) <- "gamlss2.family"

  return(fam)
}

## load the abdominal circumference data
data("abdom", package = "gamlss.data")
```

```
## specify the model Formula
f <- y ~ s(x) | s(x)

## estimate model
b <- gamlss2(f, data = abdom, family = Normal)

## plot estimated effects
plot(b, which = "effects")

## plot diagnostics
plot(b, which = "resid")

## predict parameters
par <- predict(b)

## predict quantiles
pq <- sapply(c(0.05, 0.5, 0.95), function(q) family(b)$q(q, par))

## visualize
plot(y ~ x, data = abdom, pch = 19,
  col = rgb(0.1, 0.1, 0.1, alpha = 0.3))
matplot(abdom$x, pq, type = "l", lwd = 2,
  lty = 1, col = 4, add = TRUE)

## another example using only the density
## function, all derivatives are approximated
## in this case; for residual diagnostics,
## the $p() and $q() function is needed, too.
Gamma <- function(...) {
  fam <- list(
    "names" = c("mu", "sigma"),
    "links" = c("mu" = "log", "sigma" = "log"),
    "d" = function(y, par, log = FALSE, ...) {
      shape <- par$sigma
      scale <- par$mu/par$sigma
      dgamma(y, shape = shape, scale = scale, log = log)
    },
    "p" = function(y, par, lower.tail = TRUE, log.p = FALSE) {
      shape <- par$sigma
      scale <- par$mu/par$sigma
      pgamma(y, shape = shape, scale = scale,
        lower.tail = lower.tail, log.p = log.p)
    },
    "q" = function(p, par, lower.tail = TRUE, log.p = FALSE) {
      shape <- par$sigma
      scale <- par$mu/par$sigma
       qgamma(p, shape = shape, scale = scale,
         lower.tail = lower.tail, log.p = log.p)
    }
  )

  class(fam) <- "gamlss2.family"
```

```
  return(fam)
}

## example using the Munich rent data
data("rent", package = "gamlss.data")

## model formula
f <- R ~ ti(Fl) + ti(A) + ti(Fl, A, bs = "ps") |
  ti(Fl) + ti(A) + ti(Fl, A, bs = "ps")

## estimate model
b <- gamlss2(f, data = rent, family = Gamma)

## visualize estimated effects
plot(b, which = "effects")

## diagnostics, needs the $p() and $q() function!
plot(b, which = "resid")

## End(Not run)
```

`gamlss2_control` *Control Parameters*

Description

Various parameters that control fitting of GAMLSS using `gamlss2`.

Usage

```
gamlss2_control(optimizer = RS, trace = TRUE,
  flush = TRUE, light = FALSE, expand = TRUE,
  model = TRUE, x = TRUE, y = TRUE,
  fixed = FALSE, ...)
```

Arguments

`optimizer`	Function, the optimizer function that should be used for fitting.
`trace`	Logical, should information be printed while the algorithm is running?
`flush`	Logical, use `flush.console` for displaying the current output in the console.
`light`	Logical, if set to `light = TRUE`, no model frame, response, model matrix and other design matrices will be part of the return value.
`expand`	Logical, if fewer formulas are supplied than there are parameters of the distribution, should formulas with intercept only formulas be added?
`model`	Logical, should the *model frame* be included as component of the returned object.

x	Logical, indicating whether the model matrix should be included as component of the returned object.
y	Logical, should the response be included as component of the returned object.
fixed	Logical, a named vector of parameters that should be fixed during estimation. See the examples for gamlss2.
...	Further control parameters to be part of the return value, e.g., used within optimizer function RS.

Details

The control parameters in gamlss2_control can also be extended, e.g., if another optimization function is used, newly specified control parameters are automatically passed on to this function.

Value

A list with the arguments specified.

See Also

RS, gamlss2

Examples

```
## Not run: ## load the abdominal circumference data
data("abdom", package = "gamlss.data")

## specify the model Formula
f <- y ~ s(x) | s(x)

## estimate model with different step length
## control in the RS algorithm
b1 <- gamlss2(f, data = abdom, family = BCT, step = 1)
b2 <- gamlss2(f, data = abdom, family = BCT, step = 0.9)

## End(Not run)
```

HarzTraffic *Traffic Counts at Sonnenberg in the Harz Region*

Description

This dataset contains daily traffic counts close to Sonnenberg, located in the Harz region in Germany. It covers a period of nearly three years, from 2021-01-01 to 2023-11-30.

Usage

```
data("HarzTraffic", package = "gamlss2")
```

Format

A data frame containing 1057 observations on 16 variables.

date Date, the date of the record.

yday Integer, the day of the year.

bikes Integer, the number of motorcycles on that day.

cars Integer, the number of cars on that day.

trucks Integer, the number of trucks on that day.

others Integer, the number of other vehicles on that day.

tempmin Numeric, minimum temperature in $°C$.

tempmax Numeric, maximum temperature in $°C$.

temp Numeric, mean temperature in $°C$.

humidity Numeric, mean relative humidity in percent.

tempdew Numeric, average dewpoint temperature in $°C$.

cloudiness Numeric, average cloud cover in percent.

rain Numeric, amount of precipitation in mm (snow and rain).

sunshine Numeric, sunshine duration in minutes.

wind Numeric, mean wind speed in m/s.

windmax Numeric, maximum wind speed in m/s.

Source

Weather Data:

Data Source: Deutscher Wetterdienst (DWD), Climate Data Center (CDC).

Licence: CC BY 4.0

URL: https://opendata.dwd.de/climate_environment/CDC/

Station: Wernigerode (5490; Sachsen-Anhalt)

Position: 10.7686/51.8454/233 (lon, lat, alt, EPSG 4326)

Traffic Data:

Data Source: Bundesanstalt für Strassenwesen (BASt)

Licence: CC BY 4.0

URL: https://www.bast.de, https://www.bast.de/DE/Verkehrstechnik/Fachthemen/v2-verkehrszaehlung/Verkehrszaehlung.html

Examples

```
## seasonal variation of motorcycle counts at Sonnenberg/Harz
data("HarzTraffic", package = "gamlss2")
plot(bikes ~ yday, data = HarzTraffic)

## count distribution
barplot(table(HarzTraffic$bikes))

## negative binomial seasonal model using cyclic splines
m <- gamlss2(bikes ~ s(yday, bs = "cc") | s(yday, bs = "cc"),
  data = HarzTraffic, family = NBI)

## visualize effects
plot(m)

## residual diagnostics
plot(m, which = "resid")

## fitted parameters for each day of the year
nd <- data.frame(yday = 1:365)
par <- predict(m, newdata = nd)

## corresponding quantiles
p <- sapply(c(0.05, 0.5, 0.95), function(q) family(m)$q(q, par))

## visualization
plot(bikes ~ yday, data = HarzTraffic, pch = 19, col = gray(0.1, alpha = 0.3))
matplot(nd$yday, p, type = "l", lty = c(2, 1, 2), lwd = 2, col = 4, add = TRUE)
```

---

pb *P-Splines for GAMLSS*

Description

Estimation of P-splines using an efficient local maximum likelihood approach to automatically select the smoothing parameter. According to the inventors of P-splines, pb stands for "penalized beta" splines or "Paul and Brian".

Usage

```
pb(x, k = 20, ...)
```

Arguments

x	The variable that should be used for estimation.
k	The dimension of the B-spline basis to represent the smooth term.
...	Further arguments passed to function s.

Details

Function pb() is an **internal** wrapper function that calls s to set up a smooth specification object that can be used for model fitting with gamlss2. Using pb(), an efficient local maximum likelihood approach is used to estimate the smoothing parameter. See the reference for details.

Value

The function returns a smooth specification object of class "ps.smooth.spec", see also smooth.construct.ps.smooth.spe

References

Eilers PHC, Marx BD (1996). "Flexible Smoothing with B-Splines and Penalties." *Statistical Science*, **11**(2), 89–121. doi:10.1214/ss/1038425655

Rigby RA, Stasinopoulos DM (2014). "Automatic Smoothing Parameter Selection in GAMLSS with an Application to Centile Estimation." *Statistical Methods in Medical Research*, **23**(4), 318–332. doi:10.1177/0962280212473302

See Also

gamlss2, smooth.construct.ps.smooth.spec

Examples

```
## load head circumference data
data("dbhh", package = "gamlss.data")

## specify the model Formula
f <- head ~ pb(age) | pb(age) | pb(age) | pb(age)

## estimate model
b <- gamlss2(f, data = dbhh, family = BCT)

## visualize estimated effects
plot(b, which = "effects")

## plot diagnostics
plot(b, which = "resid")

## predict parameters
par <- predict(b)

## predict quantiles
pq <- sapply(c(0.05, 0.5, 0.95), function(q) family(b)$q(q, par))

## plot
plot(head ~ age, data = dbhh, pch = 19,
  col = rgb(0.1, 0.1, 0.1, alpha = 0.3))
matplot(dbhh$age, pq, type = "l",
  lty = 1, col = 4, add = TRUE)
```

plot.gamlss2 *Plotting GAMLSS*

Description

Plotting methods for objects of class "gamlss2" and "gamlss2.list", which can be used for effect plots of model terms or residual plots. Note that effect plots of model terms with more than two covariates are not supported, for this purpose use the predict method.

Usage

```
## S3 method for class 'gamlss2'
plot(x, parameter = NULL,
  which = "effects", terms = NULL,
  scale = TRUE, spar = TRUE, ...)

## S3 method for class 'gamlss2.list'
plot(x, parameter = NULL, which = "effects",
  terms = NULL, spar = TRUE, legend = TRUE, ...)
```

Arguments

x	An object of class "gamlss2" or "gamlss2.list", which can be created by using the c() method combining "gamlss2" objects. See th examples.
parameter	Character or integer. For which parameter/model/what should the plots be created? Note that instead of argument parameter plots can also be specified passing argument model and what to ....
which	Character or integer, selects the type of plot: "effects" produces effect plots of (special) model terms, "hist-resid" shows a histogram of residuals, "qq-resid" shows a quantile-quantile plot of residuals, "scatter-resid" shows a scatter plot of residuals with fitted values for the distribution mean (or median, if available in the family object).
terms	Character or integer. For which model term should the plot(s) be created?
scale	If set to 1, effect plots all have the same scale on the y-axis. If set to 0 each effect plot has its own scale for the y-axis.
spar	Should graphical parameters be set?
legend	Should a legend be added using multiple model plots?
...	Arguments such as lwd, lty, col, legend = TRUE (for multiple model plots), a.o., depending on the type of plot. See the examples.

See Also

gamlss2.

Examples

```
## Not run: ## load data
data("film90", package = "gamlss.data")

## model formula
f <-  ~ s(lboopen) + s(lnosc)
f <- rep(list(f), 4)
f[[1]] <- update(f[[1]], lborev1 ~ .)

## estimate model
b1 <- gamlss2(f, data = film90, family = BCCG)

## plot effects (default)
plot(b1)

## plot specific effect
plot(b1, parameter = "sigma")
plot(b1, model = "sigma")
plot(b1, model = "nu", term = 1)
plot(b1, model = "nu", term = 2)
plot(b1, model = "nu", term = "lnosc")
plot(b1, term = "lnosc")

## plot all residual diagnostics
plot(b1, which = "resid")

## single diagnostic plots
plot(b1, which = "hist-resid")
plot(b1, which = "qq-resid")
plot(b1, which = "wp-resid")
plot(b1, which = "scatter-resid")

## estimate another model
b2 <- gamlss2(f, data = film90, family = BCPE)

## compare estimated effects
plot(c(b1, b2))
plot(c(b1, b2), term = "lboopen",
  col = c(1, 4), lwd = 3, lty = 1,
  pos = c("topleft", "topright", "bottomleft", "bottomright"))
plot(c(b1, b2), model = "sigma")
plot(c(b1, b2), model = "sigma", term = 2)
plot(c(b1, b2), model = c("mu", "nu"))

## End(Not run)
```

`predict.gamlss2` *Extracting Fitted or Predicted Parameters or Terms from gamlss2 Models*

Description

Methods for **gamlss2** model objects for extracting fitted (in-sample) or predicted (out-of-sample) parameters, terms, etc.

Usage

```
## S3 method for class 'gamlss2'
predict(object, model = NULL, newdata = NULL,
  type = c("parameter", "link", "response", "terms"), terms = NULL,
  se.fit = FALSE, drop = TRUE, ...)
```

Arguments

object	model object of class gamlss2.
model	character. Which model part(s) should be predicted? Can be one or more of "mu", "sigma", etc. By default all model parts are included.
newdata	data.frame. Optionally, a new data frame in which to look for variables with which to predict. If omitted, the original observations are used.
type	character. Which type of prediction should be computed? Can be the full additive predictor(s) ("link", before applying the link function(s)), the corresponding parameter ("parameter", after applying the link function(s)), the individual terms of the additive predictor(s) ("terms"), or the corresponding mean of the response distribution ("response").
terms	character. Which of the terms in the additive predictor(s) should be included? By default all terms are included.
se.fit	logical. Should standard errors for the predictions be included? *(not implemented yet)*.
drop	logical. Should the predictions be simplified to a vector if possible (TRUE) or always returned as a data.frame (FALSE)?
...	currently only used for catching what as an alias for model.

Details

Predictions for gamlss2 model objects are obtained in the following steps: First, the original data is extracted or some newdata is set up. Second, all of the terms in the additive predictors of all model parameters ("mu", "sigma", ...) are computed. Third, the full additive predictor(s) are obtained by adding up all individual terms. Fourth, the parameter(s) are obtained from the additive predictor(s) by applying the inverse link function(s). In a final step, the mean of the associated probability distribution can be computed.

See also prodist.gamlss2 for setting up a full **distributions3** object from which moments, probabilities, quantiles, or random numbers can be obtained.

Value

If drop = FALSE a data.frame. If drop = TRUE (the default), the data.frame might be simplified to a numeric vector, if possible.

See Also

predict, prodist.gamlss2

Examples

```
## fit heteroscedastic normal GAMLSS model
## stopping distance (ft) explained by speed (mph)
data("cars", package = "datasets")
m <- gamlss2(dist ~ s(speed) | s(speed), data = cars, family = NO)

## new data for predictions
nd <- data.frame(speed = c(10, 20, 30))

## default: additive predictors (on link scale) for all model parameters
predict(m, newdata = nd)

## mean of the response distribution
predict(m, newdata = nd, type = "response")

## model parameter(s)
predict(m, newdata = nd)
predict(m, newdata = nd, model = "sigma")
predict(m, newdata = nd, model = "sigma", drop = FALSE)

## individual terms in additive predictor(s)
predict(m, newdata = nd, type = "terms", model = "sigma")
predict(m, newdata = nd, type = "terms", model = "sigma", terms = "s(speed)")
```

---

prodist.gamlss2 *Extracting Fitted or Predicted Probability Distributions from gamlss2 Models*

Description

Methods for **gamlss2** model objects for extracting fitted (in-sample) or predicted (out-of-sample) probability distributions as **distributions3** objects.

Usage

```
## S3 method for class 'gamlss2'
prodist(object, ...)
```

Arguments

object	A model object of class gamlss2.
...	Arguments passed on to predict.gamlss2, e.g., newdata.

Details

To facilitate making probabilistic forecasts based on gamlss2 model objects, the prodist method extracts fitted or predicted probability distribution objects. Internally, the predict.gamlss2 method is used first to obtain the distribution parameters (mu, sigma, tau, nu, or a subset thereof). Subsequently, the corresponding distribution object is set up using the GAMLSS class from the **gamlss.dist** package, enabling the workflow provided by the **distributions3** package (see Zeileis et al. 2022).

Note that these probability distributions only reflect the random variation in the dependent variable based on the model employed (and its associated distributional assumption for the dependent variable). This does not capture the uncertainty in the parameter estimates.

Value

An object of class GAMLSS inheriting from distribution.

References

Zeileis A, Lang MN, Hayes A (2022). "distributions3: From Basic Probability to Probabilistic Regression." Presented at *useR! 2022 - The R User Conference*. Slides, video, vignette, code at https://www.zeileis.org/news/user2022/.

See Also

GAMLSS, predict.gamlss2

Examples

```
## packages, code, and data
library("distributions3")
data("cars", package = "datasets")

## fit heteroscedastic normal GAMLSS model
## stopping distance (ft) explained by speed (mph)
m <- gamlss2(dist ~ s(speed) | s(speed), data = cars, family = NO)

## obtain predicted distributions for three levels of speed
d <- prodist(m, newdata = data.frame(speed = c(10, 20, 30)))
print(d)

## obtain quantiles (works the same for any distribution object 'd' !)
quantile(d, 0.5)
quantile(d, c(0.05, 0.5, 0.95), elementwise = FALSE)
quantile(d, c(0.05, 0.5, 0.95), elementwise = TRUE)

## visualization
plot(dist ~ speed, data = cars)
nd <- data.frame(speed = 0:240/4)
nd$dist <- prodist(m, newdata = nd)
nd$fit <- quantile(nd$dist, c(0.05, 0.5, 0.95))
matplot(nd$speed, nd$fit, type = "l", lty = 1, col = "slategray", add = TRUE)
```

```
## moments
mean(d)
variance(d)

## simulate random numbers
random(d, 5)

## density and distribution
pdf(d, 50 * -2:2)
cdf(d, 50 * -2:2)

## Poisson example
data("FIFA2018", package = "distributions3")
m2 <- gamlss2(goals ~ s(difference), data = FIFA2018, family = PO)
d2 <- prodist(m2, newdata = data.frame(difference = 0))
print(d2)
quantile(d2, c(0.05, 0.5, 0.95))

## note that log_pdf() can replicate logLik() value
sum(log_pdf(prodist(m2), FIFA2018$goals))
logLik(m2)
```

`quantile.gamlss2` *Quantiles for GAMLSS*

Description

The function computes estimated quantiles and optionally produces a plot.

Usage

```
## S3 method for class 'gamlss2'
quantile(x, probs = c(0.025, 0.25, 0.50, 0.75, 0.975),
  variable = NULL, newdata = NULL,
  plot = FALSE, data = TRUE,
  n = 100L, ...)
```

Arguments

`x`	An object of class `"gamlss2"`.
`probs`	Numeric vector of probabilities with values in [0,1].
`variable`	Logical or integer, should quantiles be plotted using the covariate data? Note that the variable option is only possible for single covariate models.
`newdata`	Data frame that should be used for computing the quantiles.
`plot`	Logical, should a plot be shown?
`data`	Logical, should the raw data be added to the plot?

n	Integer, number of observations that should be used to compute an equidistant grid for the selected variable.
...	Arguments such as col, legend = TRUE/FALSE. See the examples.

Details

The function applies the predict method to determine the parameters of the response distribution. It then computes the quantiles as specified in the argument probs.

Value

A data frame of the estimated quantiles.

See Also

gamlss2.

Examples

```
## Not run: ## load data
data("film90", package = "gamlss.data")

## model formula
f <-  ~ s(lboopen)
f <- rep(list(f), 4)
f[[1]] <- update(f[[1]], lborev1 ~ .)

## estimate model
b <- gamlss2(f, data = film90, family = BCPE)

## compute quantiles using "newdata"
nd <- film90[1:10, ]
print(quantile(b, newdata = nd))

## plot sorted quantiles
quantile(b, plot = TRUE)

## quantile plot using covariate data
quantile(b, plot = TRUE, variable = TRUE)

## plot without raw data
quantile(b, plot = TRUE, variable = TRUE, data = FALSE)

## End(Not run)
```

re *Random Effects*

Description

There are two ways of fitting a random effect within gamlss2. The first, using s(), is for a simple random effect, that is, when only one factor is entered the model as a smoother. This method uses the function s() of the package **mgcv** with argument bs = "re". For example, if area is factor with several levels, s(area, bs = "re") will sringh the levels of area towards their mean level. The second, more general way, allows to fit more complicated random effect models using the function re(). The function re() is an interface connecting gamlss2 with the specialised package for random effects **nlme**.

Here we document only the re() function only but we also give examples using s(..., bs = "re").

Usage

```
re(fixed =~ 1, random = NULL, ...)
```

Arguments

fixed	A formula that specifies the fixed effects of the nlme{lme} model. In most cases, this can also be included in the gamlss2 parameter formula.
random	A formula specifying the random effect part of the model, as in the nlme{lme()} function.
...	For the re() function, the dots argument is used to specify additional control arguments for the nlme{lme} function, such as the method and correlation arguments.

Details

Both functions set up model terms that can be estimated using a backfitting algorithm, e.g., the default RS algorithm.

Value

Function s with bs = "re" returns a smooth specification object of class "re.smooth.spec", see also smooth.construct.re.smooth.spec.

The re() function returns a special model term specification object, see specials for details.

See Also

gamlss2, smooth.construct.re.smooth.spec, s, lme

Examples

```
## orthdontic measurement data
data("Orthodont", package = "nlme")

## model using lme()
m <- lme(distance ~ I(age-11), data = Orthodont,
  random =~ I(age-11) | Subject, method = "ML")

## using re(), function I() is not supported,
## please transform all variables in advance
Orthodont$age11  <- Orthodont$age - 11

## estimation using the re() constructor
b <- gamlss2(distance ~ s(age,k=3) + re(random =~ age11 | Subject),
  data = Orthodont)

## compare fitted values
plot(fitted(b, model = "mu"), fitted(m))
abline(0, 1, col = 4)

## extract summary for re() model term
st <- specials(b, model = "mu", elements = "model")
summary(st)

## random intercepts and slopes with s() using AIC
a <- gamlss2(distance ~ s(age,k=3) + s(Subject, bs = "re") + s(Subject, age11, bs = "re"),
  data = Orthodont)

## compare fitted values
plot(fitted(b, model = "mu"), fitted(m))
points(fitted(a, model = "mu"), fitted(m), col = 2)
abline(0, 1, col = 4)

## more complicated correlation structures.
data("Ovary", package = "nlme")

## ARMA model
m <- lme(follicles ~ sin(2 * pi * Time) + cos(2 * pi * Time), data = Ovary,
  random = pdDiag(~sin(2*pi*Time)), correlation = corARMA(q = 2))

## now with gamlss2(), transform in advance
Ovary$sin1 <- sin(2 * pi * Ovary$Time)
Ovary$cos1 <- cos(2 * pi * Ovary$Time)

## model formula
f <- follicles ~ s(Time) + re(random =~ sin1 | Mare,
  correlation = corARMA(q = 2), control = lmeControl(maxIter = 100))

## estimate model
b <- gamlss2(f, data = Ovary)
```

```
## smooth random effects
f <- follicles ~ ti(Time) + ti(Mare, bs = "re") +
  ti(Mare, Time, bs = c("re", "cr"), k = c(11, 5))

g <- gamlss2(f, data = Ovary)

## compare fitted values
par(mfrow = n2mfrow(nlevels(Ovary$Mare)), mar = c(4, 4, 1, 1))

for(j in levels(Ovary$Mare))
  {
  ds <- subset(Ovary, Mare == j)

  plot(follicles ~ Time, data = ds)

  f <- fitted(b, model = "mu")[Ovary$Mare == j]
  lines(f ~ ds$Time, col = 4, lwd = 2)

  f <- fitted(g, model = "mu")[Ovary$Mare == j]
  lines(f ~ ds$Time, col = 3, lwd = 2)

  f <- fitted(m)[Ovary$Mare == j]
  lines(f ~ ds$Time, col = 2, lwd = 2)
 }
```

`response_name` *Auxiliary Functions for Formulas and Model Objects*

Description

Various auxiliary functions to facilitate the work with formulas and fitted model objects.

Usage

```
response_name(formula)
```

Arguments

`formula`	A `formula`, `Formula`, or a fitted model object.

Value

Function `response_name` extracts the response name as a character vector.

See Also

`gamlss2`

Examples

```
## basic formula
f <- y ~ x1 + x2 + log(x3)
response_name(f)

## formula with multiple responses
f <- y1 | y2 | y3 ~ x1 + s(x2) + x3 + te(log(x3), x4) | x2 + ti(x5)
response_name(f)

## list of formulas
f <- list(
  y1 ~ x1 + s(x2) + x3 + te(log(x3), x4),
  y2  ~ x2 + sqrt(x5),
  y3  ~ z2 + x1 + exp(x3) + s(x10)
)
response_name(f)
```

RS *Rigby and Stasinopoulos (RS) & Cole and Green (CG) Algorithm*

Description

The function `RS()` implements the algorithm of Rigby and Stasinopoulos, the function `CG()` implements the algorithm of Cole and Green for estimating a GAMLSS with `gamlss2`.

Usage

```
## Rigby and Stasinopoulos algorithm.
RS(x, y, specials, family, offsets,
  weights, start, xterms, sterms, control)

## Cole and Green algorithm.
CG(x, y, specials, family, offsets,
  weights, start, xterms, sterms, control)
```

Arguments

`x`	The full model matrix to be used for fitting.
`y`	The response vector or matrix.
`specials`	A named list of special model terms, e.g., including design and penalty matrices for fitting smooth terms using `smooth.construct`.
`family`	A family object, see `gamlss2.family`.
`offsets`	If supplied, a list or data frame of possible model offset.
`weights`	If supplied, a numeric vector of weights.

start	Starting values, either for the parameters of the response distribution or, if specified as a named list in which each element of length one is named with "(Intercept)", the respective intercepts are initialized. If starting values are specified as a named list, data frame or matrix, where each element/column is a vector with the same length as the number of observations in the data, the respective predictors are initialized with these. See the examples for gamlss2.
xterms	A named list specifying the linear model terms. Each named list element represents one parameter as specified in the family object.
sterms	A named list specifying the special model terms. Each named list element represents one parameter as specified in the family object.
control	Further control arguments as specified within the call of gamlss2. See the details.

Details

Functions RS() and CG() are called within gamlss2. Both functions implement a backfitting algorithm for estimating GAMLSS. For algorithm details see Rigby and Stasinopoulos (2005).

The functions use the following control arguments:

- eps: Numeric vector of length 2, the stopping criterion. Default is eps = c(1e-05, 1e-05) for the outer and the inner backfitting loop.
- maxit: Integer vector of length 2, the maximum number of iterations of the outer and inner backfitting loop. Default is maxit = c(100, 10).
- step: Numeric, the step length control parameter. Default is step = 1. Note that if step is set smaller than 1, it might be appropriate to lower the stopping criterion eps, too.
- CG: Integer, the number of iterations when to start the CG correction. Default is CG = Inf.
- trace: Logical, should information be printed while the algorithm is running?
- flush: Logical, use flush.console for displaying the current output in the console.
- ridge: Logical, should automatic ridge penalization be applied only to linear effects, without penalizing the intercept? For each parameter of the distribution the optimum ridge penalty is estimated using an information criterion. Possible options are criterion = c("aic", "aicc", "bic", "gaic", "gcv"). The default is criterion = "gaic" and argument K = 2, which can be set in gamlss2_control.

To facilitate the development of new algorithms for **gamlss2**, users can exchange them using the optimizer argument in gamlss2_control. Users developing new model fitting functions are advised to use these functions as templates and pass them to gamlss2_control. Alternatively, users can replace the optimizer function by adding a named list element, "optimizer", to the family object. For instructions on setting up new families in **gamlss2**, see gamlss2.family.

Value

Functions RS() and CG() return a named list of class "gamlss2" containing the following objects:

fitted.values	A data frame of the fitted values of the modeled parameters of the selected distribution.

`fitted.specials`
: A named list, one element for each parameter of the distribution, containing the fitted model object information of special model terms.

`fitted.linear`
: A named list, one element for each parameter of the distribution, containing the information on fitted linear effects.

`coefficients`
: A named list, one element for each parameter of the distribution, containing the estimated parameters of the linear effects.

`elapsed`
: The elapsed runtime of the algorithm.

`iterations`
: How many iterations the algorithm performed.

`logLik`
: The final value of the log-likelihood of the model.

`control`
: All control arguments used as supplied from function `gamlss2_control`.

References

Rigby RA, Stasinopoulos DM (2005). "Generalized Additive Models for Location, Scale and Shape (with Discussion)." *Journal of the Royal Statistical Society, Series C (Applied Statistics)*, **54**, 507–554. doi:10.1111/j.14679876.2005.00510.x

See Also

`gamlss2`, `gamlss2_control`, `gamlss2.family`

Examples

```
## Not run: ## load the abdominal circumference data
data("abdom", package = "gamlss.data")

## specify the model Formula
f <- y ~ s(x) | s(x) | s(x) | s(x)

## estimate model using RS (default)
b <- gamlss2(f, data = abdom, family = BCT, optimizer = RS)

## now with CG
b <- gamlss2(f, data = abdom, family = BCT, optimizer = CG)

## first 2 RS iterations and afterwards switch to CG
b <- gamlss2(f, data = abdom, family = BCT, CG = 2)

## End(Not run)
```

Rsq *GAIC and Generalised (Pseudo) R-squared for GAMLSS Models*

Description

Functions to compute the GAIC and the generalised R-squared of Nagelkerke (1991) for a GAMLSS models.

Usage

```
## Information criteria.
GAIC(object, ...,
  k = 2, corrected = FALSE)

## R-squared.
Rsq(object, ...,
  type = c("Cox Snell", "Cragg Uhler", "both", "simple"),
  newdata = NULL)
```

Arguments

object	A fitted model object
...	Optionally more fitted model objects.
k	Numeric, the *penalty* to be used. The default k = 2 corresponds to the classical AIC.
corrected	Logical, whether the corrected AIC should be used? Note that it applies only when k = 2.
type	which definition of R squared. Can be the "Cox Snell" or the Nagelkerke, "Cragg Uhler" or "both", and "simple", which computes the R-squared based on the median. In this case also newdata may be supplied.
newdata	Only for type = "simple" the R-squared can be evaluated using newdata.

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

Numeric vector or data frame, depending on the number of fitted model objects.

References

Nagelkerke NJD (1991). "A Note on a General Definition of the Coefficient of Determination." *Biometrika*, **78**(3), 691–692. doi:10.1093/biomet/78.3.691

See Also

gamlss2

Examples

```
## load the aids data set
data("aids", package = "gamlss.data")

## estimate negative binomial count models
b1 <- gamlss2(y ~ x + qrt, data = aids, family = NBI)
b2 <- gamlss2(y ~ s(x) + s(qrt, bs = "re"), data = aids, family = NBI)

## compare models
Rsq(b1)
Rsq(b1, type = "both")
Rsq(b1, b2)
GAIC(b1, b2)
AIC(b1, b2)
BIC(b1, b2)

## plot estimated effects
plot(b2)
```

`softplus` *Softplus Link Object*

Description

Link object (with link function, inverse link function, etc.) that assures positivity of parameters based on the softplus function.

Usage

```
softplus(a = 1)
```

Arguments

a	Extra parameter of the generalized softplus function

Details

The softplus link function with parameter a is given by:

$$\frac{\log(1 + \exp(a \cdot x))}{a}$$

This is an approximation of the linear spline $\max\{0, x\}$ where the discrepancy between the two functions decreases with increasing a.

Wiemann et al. (2023) propose to employ the softplus function as the inverse link function where positivity of a parameter needs to be assured, e.g., in count data regressions. This is in particular of interest as an alternative to the exponential inverse link function because the exponential implies multiplicative effects of the regressors while the softplus function does not.

Value

An object of class "link-glm".

References

Wiemann PFV, Kneib T, Hambuckers J (2023). "Using the Softplus Function to Construct Alternative Link Functions in Generalized Linear Models and Beyond." *Statistical Papers*, forthcoming. doi:10.1007/s0036202301509x

See Also

make.link, gamlss2

Examples

```
## visualization of softmax function from Wiemann et al. (2003, Figure 1)
x <- -200:200/50
plot(x, softplus(1)$linkinv(x), ylab = expression(softplus[a](x)),
  type = "l", col = 2, lwd = 2)
grid()
lines(x, softplus(5)$linkinv(x), col = 3, lwd = 2)
lines(x, softplus(10)$linkinv(x), col = 4, lwd = 2)
lines(x, pmax(0, x), lty = 3, lwd = 2)
legend("topleft", c("a = 1", "a = 5", "a = 10", "linear spline"),
  col = c(2, 3, 4, 1), lty = c(1, 1, 1, 3), lwd = 2, bty = "n")

## Poisson regression example with different links
data("FIFA2018", package = "distributions3")
m_exp <- glm(goals ~ difference, data = FIFA2018, family = poisson(link = "log"))
m_splus <- glm(goals ~ difference, data = FIFA2018, family = poisson(link = softplus(1)))
AIC(m_exp, m_splus)

## comparison of fitted effects
nd <- data.frame(difference = -15:15/10)
nd$mu_exp <- predict(m_exp, newdata = nd, type = "response")
nd$mu_splus <- predict(m_splus, newdata = nd, type = "response")
plot(mu_exp ~ difference, data = nd, ylab = expression(mu),
  type = "l", col = 4, lwd = 2, ylim = c(0, 2.5))
lines(mu_splus ~ difference, data = nd, col = 2, lwd = 2)
legend("topleft", c("exp", "softplus"), col = c(4, 2), lwd = 2, lty = 1, bty = "n")
```

special_terms *Special Model Terms for GAMLSS*

Description

The **gamlss2** package provides infrastructure to include special model terms for the optimizer functions RS and CG, e.g., such as neural networks, trees and forests. The infrastructure assumes that such special model terms provide their own fitting and predict method.

Usage

```
## Generic fitting method.
special_fit(x, ...)

## Generic predict method.
special_predict(x, ...)

## Extractor function for fitted special terms.
specials(object, model = NULL, terms = NULL, elements = NULL, ...)
```

Arguments

x	A model term object as supplied in the formula in the gamlss2 call.
object	A fitted gamlss2 object.
model	Character or integer, specifies the model for which fitted special terms should be extracted.
terms	Character or integer, specifies the special model terms that should be extracted.
elements	Character, specifies which elements of a fitted special term should be extracted. If elements = "names", the corresponding element names are extracted.
...	Arguments needed for the special_fit() function to facilitate the fitting of the model term, see the details. Similarly, for the special_predict() function, the ... argument encompasses the objects for computing predictions for the model term.

Details

To implement a new special term, the first step is to write a formula constructor function for the new model term. For example, consider the implementation below, which demonstrates how to create a neural network model term. Additionally, the name of the new model term constructor must be passed to the specials argument of the function fake_formula. Please note that in the provided example, no new special name is passed because "n" is already registered in fake_formula.

Afterwards, a fitting and a predict method for the new special model term needs to be implemented. Please also refer to the example below, implementing these functions for a neural network model term.

The following describes the detailed arguments and return values.

A method for special_fit() has the following arguments:

- x: The special model term object, containing all the data for fitting.
- z: The current working response/residual from the backfitting step.
- w: The current working weights from the backfitting step.
- y: The response vector/matrix, e.g., used to evaluate the log-likelihood.
- eta: The current named list of predictors.
- j: Character, the parameter name for which the model term needs to be updated.
- family: The family object of the model, see gamlss2.family.
- control: A named list of control arguments, see gamlss2_control.

Note that for setting up a special model term only the first three arguments a mandatory, all other arguments are optional. The function must at least return a named list containing the "fitted.values" to work with RS and CG.

A method for special_predict() has the following arguments:

- x: Depending on the return value of function special_fit(), the fitted model term object, see the examples.
- data: The data for which predictions should be computed.
- se.fit: Logical, should standard errors of the predictions be computed.

Note that function special_predict() should return a data frame with named colums "fit", "lower" and "upper", "lower" and "upper" are optional.

See Also

gamlss2, RS, gamlss2_control, gamlss2.family

Examples

```
## example special term for neural networks,
## the constructor function is used in the formula
## when calling gamlss2()
n <- function(formula, ...)
{
  stopifnot(requireNamespace("nnet"))

  ## list for setting up the special model term
  st <- list()

  ## list of control arguments
  ctr <- list(...)
  if(is.null(ctr$size))
    ctr$size <- 50
  if(is.null(ctr$maxit))
    ctr$maxit <- 1000
  if(is.null(ctr$decay))
    ctr$decay <- 0.1
  if(is.null(ctr$trace))
    ctr$trace <- FALSE
  if(is.null(ctr$MaxNWts))
```

```
  ctr$MaxNWts <- 10000
 if(is.null(ctr$scale))
  ctr$scale <- TRUE

 ## put all information together
 st$control <- ctr
 st$formula <- formula
 st$term <- all.vars(formula)
st$label <- paste0("n(", paste0(gsub(" ", "", as.character(formula)), collapse = ""), ")")
 st$data <- model.frame(formula)

 ## scale per default!
 if(ctr$scale) {
  sx <- list()
  for(j in colnames(st$data)) {
   if(!is.factor(st$data[[j]])) {
    sx[[j]] <- range(st$data[[j]])
    st$data[[j]] <- (st$data[[j]] - sx[[j]][1]) / diff(sx[[j]])
   }
  }
  st$scalex <- sx
 }

 ## assign the "special" class and the new class "n"
 class(st) <- c("special", "n")

 return(st)
}

## set up the special "n" model term fitting function
special_fit.n <- function(x, z, w, control, ...)
{
 ## model formula needs to be updated
 .fnns <- update(x$formula, response_z ~ .)

 ## assign current working response
 x$data$response_z <- z
 x$data$weights_w <- w

 ## possible weights from last iteration
 Wts <- list(...)$transfer$Wts

 ## estimate model
nnc <- parse(text = paste0('nnet::nnet(formula = .fnns, data = x$data, weights = weights_w,',
     'size = x$control$size, maxit = x$control$maxit, decay = x$control$decay,',
     'trace = x$control$trace, MaxNWts = x$control$MaxNWts, linout = TRUE',
     if(!is.null(Wts)) ', Wts = Wts)' else ')'))

 rval <- list("model" = eval(nnc))

 ## get the fitted.values
 rval$fitted.values <- predict(rval$model)
```

```
  ## transferring the weights for the next backfitting iteration
  ## note, "transfer" can be used to transfer anything from one
  ## iteration to the next
  rval$transfer <- list("Wts" = rval$model$wts)

  ## center fitted values
  rval$shift <- mean(rval$fitted.values)
  rval$fitted.values <- rval$fitted.values - rval$shift

  ## degrees of freedom
  rval$edf <- length(coef(rval$model))

  ## possible scaling
  rval$scalex <- x$scalex

  ## assign class for predict method
  class(rval) <- "n.fitted"

  return(rval)
}

## finally, the predict method
special_predict.n.fitted <- function(x, data, se.fit = FALSE, ...)
{
  if(!is.null(x$scalex)) {
    for(j in names(x$scalex)) {
      data[[j]] <- (data[[j]] - x$scalex[[j]][1]) / diff(x$scalex[[j]])
    }
  }
  p <- predict(x$model, newdata = data, type = "raw")
  p <- p - x$shift
  if(se.fit)
    p <- data.frame("fit" = p)
  return(p)
}

## Not run: ## example with data
data("abdom", package = "gamlss.data")

## specify the model Formula
f <- y ~ n(~x) | n(~x) | n(~x) | n(~x)

## estimate model,
## set the seed for reproducibility
## note, data should be scaled!
set.seed(123)
b <- gamlss2(f, data = abdom, family = BCT)

## visualize estimated effects
plot(b, which = "effects")

## plot diagnostics
plot(b, which = "resid")
```

```
## predict parameters
par <- predict(b)

## predict quantiles
pq <- sapply(c(0.05, 0.5, 0.95), function(q) family(b)$q(q, par))

## plot
plot(y ~ x, data = abdom, pch = 19,
  col = rgb(0.1, 0.1, 0.1, alpha = 0.3))
matplot(abdom$x, pq, type = "l", lwd = 2,
  lty = 1, col = 4, add = TRUE)

## another example using the Munich rent data
data("rent", package = "gamlss.data")

## model Formula
f <- R ~ n(~Fl+A,size=10,decay=0.7) | n(~Fl+A,size=10,decay=0.7)

## estimate model
set.seed(456)
b <- gamlss2(f, data = rent, family = GA)

## plot estimated effects
plot(b, which = "effects", persp = FALSE)

## diagnostics
plot(b, which = "resid")

## predict using new data
n <- 50
nd <- with(rent, expand.grid(
  "Fl" = seq(min(Fl), max(Fl), length = n),
  "A" = seq(min(A), max(A), length = n)
))

## predict parameters of the GA distribution
par <- predict(b, newdata = nd)

## compute median rent R estimate
nd$fit <- family(b)$q(0.5, par)

## visualize
library("lattice")

p1 <- wireframe(fit ~ Fl + A, data = nd,
  screen = list(z = 50, x = -70, y = -10),
  aspect = c(1, 0.9), drape = TRUE,
  main = "n(~Fl+A)",
  xlab = "Floor", ylab = "YoC",
  zlab = "Rent")

p2 <- levelplot(fit ~ Fl + A, data = nd,
```

```
  contour = TRUE,
  main = "n(~Fl+A)", xlab = "Floor", ylab = "YoC")

print(p1, split = c(1, 1, 2, 1), more = TRUE)
print(p2, split = c(2, 1, 2, 1), more = FALSE)

## extract fitted special terms,
## fitted NN for parameter mu
specials(b, model = "mu", elements = "model")

## same for sigma
specials(b, model = "sigma", elements = "model")

## return element names of fitted special term list
specials(b, model = "sigma", elements = "names")

## End(Not run)
```

---

stepwise *Stepwise Model Term Selection Using GAIC*

Description

The optimizer function `stepwise()` performs stepwise model term selection using a Generalized Akaike Information Criterion (GAIC). Estimation is based on the Rigby and Stasinopoulos (RS) & Cole and Green (CG) algorithm as implemented in function RS.

Usage

```
## Wrapper function for stepwise GAMLSS estimation.
step_gamlss2(formula, ..., K = 2,
  strategy = c("both.linear", "both"), keeporder = FALSE,
  cores = 1L)

## After stepwise search, extract the new formula.
new_formula(object)

## Stepwise optimizer function.
stepwise(x, y, specials, family, offsets,
  weights, start, xterms, sterms, control)
```

Arguments

formula	A model formula for gamlss2.
...	Arguments passed to gamlss2.
K	Numeric, the penalty for the GAIC.

strategy	Character, the strategy that should be applied for the stepwise algorithm. Possible options are "forward.linear", "forward", "backward", "backward.linear", "replace", "replace.linear", "both", "both.linear". See the details.
keeporder	Logical, For the different strategies of the stepwise algorithm, should the updates be performed sequentially according to the order of the parameters of the response distribution as specified in the family (see gamlss2.family), or should the selection search be performed across all parameters?
cores	Integer, if cores > 1L, function mclapply function is used to speed up computations using multiple cores within the selection steps.
object	An object fitted using step_gamlss2().
x	The full model matrix to be used for fitting.
y	The response vector or matrix.
specials	A named list of special model terms, e.g., including design and penalty matrices for fitting smooth terms using smooth.construct.
family	A family object, see gamlss2.family.
offsets	If supplied, a list or data frame of possible model offset.
weights	If supplied, a numeric vector of weights.
start	Starting values, either for the parameters of the response distribution or, if specified as a named list in which each element of length one is named with "(Intercept)", the respective intercepts are initialized. If starting values are specified as a named list, data frame or matrix, where each element/column is a vector with the same length as the number of observations in the data, the respective predictors are initialized with these. See the examples for gamlss2.
xterms	A named list specifying the linear model terms. Each named list element represents one parameter as specified in the family object.
sterms	A named list specifying the special model terms. Each named list element represents one parameter as specified in the family object.
control	Further control arguments as specified within the call of gamlss2.

Details

The wrapper function step_gamlss2() calls gamlss2 using the stepwise() optimizer function.

The stepwise algorithm can apply the following strategies:

1. Each predictor must include an intercept.
2. In a forward selection step, model terms with the highest improvement on the GAIC are selected.
3. In a replacement step, each model term is tested to see if an exchange with a model term not yet selected will improve the GAIC.
4. In a backward step, model terms are deselected, if the GAIC can be further improved.
5. In a bidirectional step, model terms can be either added or removed.
6. In addition, the forward, backward and replace selection step can be combined.

The selected strategies are iterated until no further improvement is achieved.

The different strategies can be selected using argument `strategy`. Please see the examples. Possible values are `strategy = c("both", "forward", "backward", "replace", "all")`. Here, `strategy = "all"` combines the forward, backward and replace selection step.

In addition, each of the steps 2-4 can be applied to linear model terms only, prior to performing the steps for all model terms. This can be done by additionally setting `strategy = c("both.linear", "forward.linear", "backward.linear", "replace.linear", "all.linear")`.

The default is `strategy = c("both.linear", "both")` and `keeporder = FALSE`.

Note that each of the steps 2-4 can be performed while maintaining the order of the parameters of the response distribution, i.e., if the `keeporder = TRUE` argument is set, then the parameters will be updated in the order specified in the `gamlss2.family`. Using backward elimination, the model terms are deselected in reverse order.

Value

The optimizer function `stepwise()` returns the final model as named list of class `"gamlss2"`. See the return value of function `RS`. The wrapper function `step_gamlss2()` also returns the final model.

See Also

`gamlss2`, `gamlss2_control`, `RS`

Examples

```
## Not run: ## load the Munich rent data
data("rent", package = "gamlss.data")

## because of possible linear interactions,
## scale the covariates first
rent$Fl <- scale(rent$Fl)
rent$A <- scale(rent$A)

## the Formula defines the searching scope
f <- R ~ Fl + A + Fl:A + loc + s(Fl) + s(A) + te(Fl, A) |
  Fl + A + loc + Fl:A + s(Fl) + s(A) + te(Fl, A)

## estimate a Gamma model using the stepwise algorithm
b <- step_gamlss2(f, data = rent, family = GA, K = 2)

## same with
## b <- gamlss2(f, data = rent, family = GA, optimizer = stepwise, K = 2)

## show the new formula of selected model terms
new_formula(b)

## final model summary
summary(b)

## effect plots
plot(b)
```

```
## diagnostic plots
plot(b, which = "resid")

## plot GAIC
plot(b, which = "selection")

## use forward linear, replace and backward strategy
b <- step_gamlss2(f, data = rent, family = GA, K = 2,
  strategy = c("forward.linear", "replace", "backward"))

## more complex model
## note, the third parameter
## nu does not include any model terms
f <- R ~ Fl + A + Fl:A + loc + s(Fl) + s(A) + te(Fl, A) |
  Fl + A + loc + Fl:A + s(Fl) + s(A) + te(Fl, A) |
  1 |
  Fl + A + loc + Fl:A + s(Fl) + s(A) + te(Fl, A)

## model using the BCT family
b <- step_gamlss2(f, data = rent, family = BCT,
  K = 2, strategy = c("forward.linear", "both"),
  keeporder = TRUE)

## plot GAIC
plot(b, which = "selection")

## End(Not run)
```

Index