

Package: gamlss2 (via r-universe)

October 14, 2026

Title GAMLSS Infrastructure for Flexible Distributional Regression

Version 0.1-0

Description Next generation infrastructure for generalized additive models for location, scale, and shape (GAMLSS) and distributional regression more generally. The package provides a fresh reimplementaion 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, distributions3 (>= 0.3.0)

Imports Formula, mvtnorm

Suggests colorspace, future, future.apply, gamlss, glmnet, lattice, Matrix, MASS, nlme, nnet, partykit, quarto, rpart, scoringRules, sf, survival, VGAM

VignetteBuilder quarto

Config/roxygen2/version 8.0.0

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

Date/Publication 2026-10-14 10:09:11 UTC

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

RemoteRef HEAD

RemoteSha 550759daa8443f8a7c84ae1fa8715468592b525a

Contents

gamlss2-package	3
ALD	4
bamlss2	6
BS	8

calibration	9
cf	11
ct	12
cv_gamlss2	13
deviance.gamlss2	15
discretize_family	17
elm	19
fake_formula	22
family.distribution	24
family_extensions	26
find_family	29
fk	31
gamlss2	33
GAMLSS2	36
gamlss2.family	38
gamlss2_control	42
gamlss2_smoothing	44
gamlss2_start	45
gnet	47
HarzTraffic	48
im	50
JR	53
Kumaraswamy	55
la	57
lin	61
lo	63
logLik.gamlss2	65
make.link2	66
marginal_predict	67
mcmc	69
mixture_family	71
MN	75
ms	77
n	79
new_formula	81
OL	82
pb	84
plot.gamlss2	85
predict.gamlss2	87
prodist.gamlss2	90
quantile.gamlss2	92
re	93
residuals.gamlss2	96
response_name	99
results	100
RS	101
Rsq	103
select_gamlss2	105

si	108
softplus	111
special_terms	112
SpirometryUS	118
stepwise	120
tree	122
user	123
vcov.gamlss2	125
Index	127

gamlss2-package	<i>GAMLSS Modeling with Advanced Flexible Infrastructures</i>
-----------------	---

Description

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

Details

gamlss2 fits an additive predictor for each parameter of a response distribution, using linear terms, smooth terms from **mgcv**, and user-defined model terms (see [specials](#)). Response distributions are specified through `gamlss2.family` objects, including families with more than four parameters. Fitting algorithms can be selected through `gamlss2.control`. Fitted models provide methods for summaries, prediction, plotting, and diagnostics. Predictions can be returned as **distributions3** objects for evaluating fitted distributions (see `predict.gamlss2`).

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

Author(s)

Nikolaus Umlauf, Mikis Stasinopoulos, Robert Rigby, Achim Zeileis, Reto Stauffer, Gillian Heller, Fernanda de Bastiani, Andreas Mayr.
Maintainer: Nikolaus Umlauf <Nikolaus.Umlauf@uibk.ac.at>.

References

To cite `gamlss2` in publications use:
Umlauf N, Stasinopoulos M, De Bastiani F, Heller G, Kneib T, Mayr A, Rigby R, Stauffer R, Zeileis A (2025). “gamlss2: The Next Generation of Distributional Regression”. In O’Sullivan D, Sweeney J, Moghaddam S, Burke K (eds.), *Proceedings of the 39th International Workshop on Statistical Modelling*, 312–317. Department of Mathematics and Statistics, University of Limerick, Limerick, Ireland. https://iws2025.ie/wp-content/uploads/2025/07/IWSM2025_Limerick_Proceedings.pdf

For the underlying methods cite:

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**(3), 507–554. doi:10.1111/j.14679876.2005.00510.x

See Also

[gamlss2](#), [fake_formula](#)

ALD

Asymmetric Laplace Quantile Regression

Description

Asymmetric Laplace family for quantile regression and a wrapper for estimating one or more conditional quantile models.

Usage

```
ALD(tau = 0.5)
```

```
fit_ALD(formula, data, tau, anchor = 0.5, max_step = 0.05,
  restarts = c("always", "on_failure", "never"), start = NULL,
  monotone = FALSE, verbose = interactive(), ...)
```

Arguments

tau	For ALD, one finite quantile probability strictly between zero and one. For fit_ALD, a numeric vector of such probabilities.
formula	A model formula accepted by gamlss2 .
data	A data frame containing the variables in formula.
anchor	Quantile probability from which the sequence of fits starts.
max_step	Maximum distance between successive quantiles along the fitting path. Intermediate bridge models are fitted when necessary. Use Inf to disable bridge models.
restarts	Character. With "always", each warm-started fit is compared with a cold fit; "on_failure" uses a cold fit only when the warm fit fails or does not converge; and "never" uses continuation only.
start	Optional starting values for the anchor model, as accepted by gamlss2 .
monotone	Logical. If TRUE, returned fitted values and predictions are rearranged pointwise to be nondecreasing in tau.
verbose	Logical. Should progress information be printed?
...	Further arguments passed to gamlss2 .

Details

The asymmetric Laplace density is proportional to $\exp\{-\rho_\tau((y - \mu)/\sigma)\}$, where $\rho_\tau(u) = u\{\tau - I(u < 0)\}$. Thus, μ is the conditional τ -quantile and $\sigma > 0$ is a scale parameter.

`fit_ALD` first estimates the anchor quantile and then proceeds toward the lower and upper requested quantiles. Coefficients and smoothing parameters are transferred between successive fits. Converged candidate fits are preferred and compared using their asymmetric Laplace log-likelihood. If no candidate converges, up to two retries halve and quarter the step length, retaining the smoothing criterion, iteration limits, and convergence tolerances. These retries use cold starts unless `restarts = "never"`. If no candidate converges after these retries, a finite candidate is returned with its convergence flag unchanged; callers should check the individual models or the diagnostics.

Monotone rearrangement affects only the values returned by `fitted()` and `predict()`; the individual fitted models are retained unchanged.

The ALD can be used as a working likelihood when the response distribution is not asymmetric Laplace. With constant scale, its data likelihood targets pinball loss for the requested quantile. The location score is nonsmooth, and working-likelihood intervals are not automatically calibrated under misspecification. Ordinary observed-information Gaussian inference for smooth likelihoods should not be interpreted as calibrated quantile confidence intervals here; a suitable calibration or resampling procedure requires separate justification.

Value

ALD returns an object of class `"gamlss2.family"`.

`fit_ALD` returns an object of class `"ALDfit"`, containing the individual models, fitted values, requested quantiles, and fitting diagnostics.

References

- Yu K, Moyeed RA (2001). "Bayesian Quantile Regression". *Statistics & Probability Letters*, **54**(4), 437–447. doi:[10.1016/S01677152\(01\)001249](https://doi.org/10.1016/S01677152(01)001249)
- Chernozhukov V, Fernandez-Val I, Galichon A (2010). "Quantile and Probability Curves Without Crossing". *Econometrica*, **78**(3), 1093–1125. doi:[10.3982/ECTA7880](https://doi.org/10.3982/ECTA7880)
- Sriram K (2015). "A Sandwich Likelihood Correction for Bayesian Quantile Regression Based on the Misspecified Asymmetric Laplace Density". *Statistics & Probability Letters*, **107**, 18–26. doi:[10.1016/j.spl.2015.07.035](https://doi.org/10.1016/j.spl.2015.07.035)

See Also

[gamlss2](#), [predict](#), [mcmc](#).

Examples

```
## Not run: ## load data
data("SpirometryUS", package = "gamlss2")

## subset for females
d <- subset(SpirometryUS, gender == "female")
```

```
## fit one conditional median model
m50 <- gamlss2(fev1 ~ s(age), data = d, family = ALD(tau = 0.5))

## estimate several conditional quantiles
qu <- c(0.025, 0.5, 0.975)
m <- fit_ALD(fev1 ~ s(age), data = d, tau = qu,
  max_step = 0.1, monotone = TRUE, verbose = FALSE)

## plot fitted quantile curves
fit <- predict(m)
i <- order(d$age)
plot(fev1 ~ age, data = d)
matlines(d$age[i], fit[i, ],
  lty = 1, lwd = 2, col = 4)

## estimate lower limits model using full MCMC
set.seed(1328)
m2.5 <- mcmc(m$models[["tau=0.025"]],
  n.iter = 1200, burnin = 200, thin = 1)

summary(m2.5)

## predict credible intervals
nd <- data.frame(age = seq(6, 80, length = 300))
nd$p <- predict(m2.5, parameter = "mu", newdata = nd,
  FUN = function(x) quantile(x, probs = c(0.025, 0.5, 0.975)))

## plot
plot(fev1 ~ age, data = d, col = adjustcolor(1, 0.3))
matlines(nd$age, nd$p, lty = c(2, 1, 2), lwd = 2, col = 2)

## End(Not run)
```

bamlss2

Bayesian GAMLSS Wrapper

Description

A wrapper around [gamlss2](#) for fitting Bayesian GAMLSS models by MCMC sampling. Starting values are computed using `maxit` backfitting iterations before sampling. The controls `n.iter`, `burnin`, and `thin` can be specified directly; weights and offset are passed through to `gamlss2`.

Usage

```
bamlss2(formula, n.iter = 1200, burnin = 200, thin = 1, maxit = 2, ...)
```

Arguments

formula	A GAM-type formula or Formula . The model can include smooth terms as provided by the mgcv package.
---------	--

n.iter	Integer, the total number of MCMC iterations.
burnin	Integer, the burn-in period.
thin	Integer, the thinning parameter for saved samples.
maxit	Integer, the number of backfitting iterations used to compute starting values before running MCMC.
...	Further arguments passed to gamlss2 and/or gamlss2_control .

Value

An object of class "bamlss2" inheriting from "gamlss2" containing the fitted model and posterior samples.

References

Umlauf N, Klein N, Zeileis A (2018). "BAMLSS: Bayesian Additive Models for Location, Scale, and Shape (and Beyond)". *Journal of Computational and Graphical Statistics*, **27**(3), 612–627. [doi:10.1080/10618600.2017.1407325](https://doi.org/10.1080/10618600.2017.1407325)

See Also

[gamlss2](#), [mcmc](#), [BS](#)

Examples

```
## Not run:
## air quality data
air <- subset(airquality, !is.na(Ozone))

## set the seed
set.seed(123)

## estimate Gamma model with MCMC
m <- bamlss2(
  Ozone ~ s(Temp) + s(Wind) | s(Temp),
  data = air,
  family = GA
)

## posterior summary
summary(m)

## plot estimated effects
plot(m)

## plot samples
plot(m, which = "samples", ask = TRUE)

## predict parameters using samples
pm <- predict(m, type = "parameter", FUN = mean)
print(head(pm))
```

```
psd <- predict(m, type = "parameter", FUN = sd)
print(head(psd))

## End(Not run)
```

BS

Bayesian Sampler for GAMLSS Models

Description

The function `BS()` implements an MCMC sampler for generalized additive models for location, scale, and shape (GAMLSS) as fitted by [gamlss2](#). The sampler uses efficient Metropolis-Hastings updates for regression coefficients of linear and smooth terms.

Usage

```
BS(x, y, specials, family, offsets,
   weights, start, xterms, sterms, control)
```

Arguments

<code>x</code>	The full model matrix to be used for fitting.
<code>y</code>	The response vector or matrix.
<code>specials</code>	A named list of special model terms, for example including design and penalty matrices for fitting smooth terms using smooth.construct .
<code>family</code>	A family object, see gamlss2.family .
<code>offsets</code>	If supplied, a list or data frame of model offsets. See gamlss2 for details on the offset interface.
<code>weights</code>	If supplied, a numeric vector of prior weights used in the fitting process.
<code>start</code>	Starting values for the parameters of the response distribution. See the examples for gamlss2 .
<code>xterms</code>	A named list specifying the linear model terms. Each named list element represents one parameter as specified in the family object.
<code>sterms</code>	A named list specifying the special model terms. Each named list element represents one parameter as specified in the family object.
<code>control</code>	Further control arguments as specified in the call to gamlss2 .

Details

Function `BS()` is typically called via [mcmc](#) or via the wrapper function [bamlss2](#). It implements a blocked Metropolis-Hastings sampler using working responses and working weights. Smooth terms are updated using penalized weighted least squares proposals and their smoothing variances are sampled using univariate slice sampling.

The function uses the following control arguments:

- `n.iter`: integer, the total number of MCMC iterations;
- `burnin`: integer, the burn-in period;
- `thin`: integer, the thinning parameter for saved samples;
- `trace`: logical, should information be printed while the sampler is running;
- `flush`: logical, whether to use [flush.console](#) to display current output in the console;
- `nullmodel`: logical, should an intercept-only null model be evaluated to compute a null deviance and deviance reduction?;
- `binning`: logical, should binning be used for smooth terms if available?

Value

A fitted model object containing posterior samples and summary information. Typically it includes:

- `samples`: posterior draws for coefficients and smoothing variances;
- `coefficients`: posterior means of coefficients;
- `fitted.values`: posterior mean fitted values for each distributional parameter;
- `edf`: estimated degrees of freedom for smooth terms;
- `logLik`: log-likelihood at the posterior mean predictor.

References

Umlauf N, Klein N, Zeileis A (2018). “BAMLSS: Bayesian Additive Models for Location, Scale, and Shape (and Beyond)”. *Journal of Computational and Graphical Statistics*, **27**(3), 612–627. [doi:10.1080/10618600.2017.1407325](https://doi.org/10.1080/10618600.2017.1407325)

See Also

[gamlss2](#), [RS](#), [CG](#), [mcmc](#), [bamlss2](#)

Examples

```
## see ?mcmc
```

calibration

Calibration Plots for Binary Responses

Description

Compute and plot calibration curves for models with 0/1 responses. The function can handle one or several fitted [gamlss2](#) models and compares their calibration in a single plot.

Usage

```
calibration(..., newdata = NULL,
  y = NULL, parameter = NULL, breaks = seq(0, 1, by = 0.1),
  minn = 20, main = "Calibration plot",
  xlab = "Predicted probability",
  ylab = "Observed proportion", plot = TRUE,
  add_loess = TRUE, smooth_n = 200,
  col = NULL, lty = NULL, legend = TRUE, pos = "topleft",
  xlim = NULL, ylim = NULL)
```

Arguments

<code>...</code>	One or several fitted gamlss2 model objects to be assessed.
<code>newdata</code>	Optional data frame for out-of-sample calibration.
<code>y</code>	Optional numeric or factor vector with the binary response.
<code>parameter</code>	Character, parameter name(s) to predict. The default NULL selects all parameters in the family's names order.
<code>breaks</code>	Numeric vector of break points used to construct bins for the predicted probabilities.
<code>minn</code>	Integer, the minimum number of observations required in each bin.
<code>main</code>	Character, main title for the plot.
<code>xlab</code>	Character, label for the x-axis.
<code>ylab</code>	Character, label for the y-axis.
<code>plot</code>	Logical, should a calibration plot be produced?
<code>add_loess</code>	Logical, should a loess smooth be added for each model?
<code>smooth_n</code>	Integer, number of evaluation points for the loess curve.
<code>col</code>	Colors used for the models.
<code>lty</code>	Line types used for the loess curves.
<code>legend</code>	Logical, should a legend be added when more than one model is supplied?
<code>pos</code>	Character, legend position passed to legend .
<code>xlim</code>	The x limits of the plot.
<code>ylim</code>	The y limits of the plot.

Details

For each fitted model, predicted probabilities are obtained via [predict.gamlss2](#) using `type = "parameter"` and the selected parameter (all family parameters when NULL). If the family provides a `probabilities()` function, it converts these predictions to probabilities; otherwise, the predicted parameters are used directly. If the result is a matrix or data frame, only its first column is used, after any conversion. Thus, without `probabilities()`, multiple predicted parameters are reduced to the first parameter column. The values used for calibration must represent the probability of $y = 1$.

Predicted probabilities are grouped into bins defined by `breaks`. Within each bin, the mean predicted probability, the observed proportion of 1s, and the number of observations are computed. Bins with fewer than `minn` observations are dropped.

Value

Invisibly returns a data frame with the columns `interval`, `probs`, `y`, `n`, and `model`. For a single model, the `model` column is dropped.

References

Van Calster B, McLernon DJ, van Smeden M, Wynants L, Steyerberg EW (2019). “Calibration: The Achilles Heel of Predictive Analytics”. *BMC Medicine*, **17**, 230. doi:10.1186/s1291601914667

See Also

[gamlss2](#), [predict](#)

Examples

```
## Not run: set.seed(123)
n <- 400
x1 <- runif(n, -3, 3)
x2 <- runif(n, -3, 3)
eta <- sin(x1) + 0.5 * x2
p <- plogis(eta)
y <- rbinom(n, size = 1, prob = p)
d <- data.frame(y = y, x1 = x1, x2 = x2)

m1 <- gamlss2(y ~ x1, family = BI, data = d)
m2 <- gamlss2(y ~ s(x1), family = BI, data = d)
m3 <- gamlss2(y ~ s(x1) + x2, family = BI, data = d)

calibration(m1, m2, m3)
head(calibration(m1, m2, m3, plot = FALSE))

## End(Not run)
```

cf

Conditional Inference Forest Model Terms

Description

Constructs a conditional inference forest model term using **partykit**.

Usage

```
cf(formula, ...)
```

Arguments

<code>formula</code>	A vector or formula specifying the covariates that should be used for modeling.
<code>...</code>	Control arguments passed to cforest .

Details

The forest is refitted with the current working response and weights during backfitting. The fitted contribution is centered for identifiability.

Value

An object of classes "special" and "cf".

See Also

[special_terms](#)

Examples

```
## Not run: ## air quality data
air <- subset(airquality, !is.na(Ozone))

set.seed(6020)
m <- gamlss2(
  Ozone ~ cf(Temp, minsplit = 2L, minbucket = 2L, alpha = 0.1) | .,
  data = air,
  family = GA
)

plot(m)

## predicted distributions and quantiles
d <- predict(m, newdata = air)
head(d, 3)
pq <- quantile(d, probs = c(0.025, 0.5, 0.975))

plot(Ozone ~ Temp, data = air,
     ylim = range(air$Ozone, pq))
i <- order(air$Temp)
matlines(air$Temp[i], pq[i, ], lty = 1, lwd = 2, col = 4)

## End(Not run)
```

ct

Conditional Inference Tree Model Terms

Description

Constructs a conditional inference tree model term using **partykit**.

Usage

```
ct(formula, ...)
```

Arguments

`formula` A vector or formula specifying the covariates that should be used for modeling.
`...` Control arguments passed to [ctree](#).

Details

The tree is refitted with the current working response and weights during backfitting. The fitted contribution is centered for identifiability.

Value

An object of classes "special" and "ct".

See Also

[special_terms](#)

Examples

```
## Not run:
m <- gamlss2(
  mag ~ ct(depth) | .,
  data = quakes,
  family = truncate_family(NO, lower = 3.99)
)

plot(m)

## predicted distributions and quantiles
d <- predict(m, newdata = quakes)
pq <- quantile(d, probs = c(0.025, 0.5, 0.975))

plot(mag ~ depth, data = quakes,
     ylim = range(quakes$mag, pq),
     col = adjustcolor(1, 0.3))
i <- order(quakes$depth)
matlines(quakes$depth[i], pq[i, ], lty = 1, lwd = 2, col = 4)

## End(Not run)
```

Description

`cv_gamlss2()` implements K-fold cross validation for models fitted with [gamlss2](#). Different scoring rules can be supplied via the `metric` argument. Convenience metric functions (`log_pdf_metric()`, `rqres_metric()`, `mse_metric()`) are provided.

Usage

```
## k-fold cross-validation
cv_gamlss2(..., data, folds = 5,
  metric = log_pdf_metric, simplify = TRUE, cores = 1L)

## log-pdf for each observation
log_pdf_metric(model, data)

## randomized quantile residuals
rqres_metric(model, data)

## mean squared error
mse_metric(model, data)
```

Arguments

<code>...</code>	model specification passed to gamlss2 such as formula, family, etc.
<code>data</code>	a <code>data.frame</code> containing the variables in the model. For functions supplied to argument <code>metric</code> , a <code>data.frame</code> for evaluating predictions or residuals.
<code>folds</code>	either an integer specifying the number of folds, or a list, matrix, or data frame of index sets for test folds. Defaults to 5.
<code>metric</code>	a function of the form <code>metric(model, data)</code> returning a score for the given fitted model and test data. Defaults to <code>log_pdf_metric</code> .
<code>simplify</code>	logical. If <code>TRUE</code> , results are returned in a simplified vector or data frame depending on the metric output.
<code>cores</code>	Positive integer, number of workers for evaluating folds. The default <code>cores = 1L</code> runs sequentially; larger values use future_lapply with a temporary multisession plan. The caller's full future plan is restored on exit, including on errors. Parallel evaluation requires the suggested packages future and future.apply .
<code>model</code>	a fitted <code>gamlss2</code> model.

Details

`cv_gamlss2()` splits the data into training and test folds. For each fold the model is fitted on the training data, and the chosen `metric` is evaluated on the held-out test data. By default, the scoring rule is the log predictive density (`log_pdf_metric`), but other metrics can be used, such as randomized quantile residuals (`rqres_metric`) or mean squared error of the conditional mean (`mse_metric`).

The function returns either a list of fold-wise results or, if `simplify = TRUE`, a named vector or data frame aligned with the original observations.

Set the random seed before calling `cv_gamlss2()` to reproduce randomly generated folds. Parallel evaluation also uses reproducible random-number streams for stochastic fits or metrics.

Value

If `simplify = TRUE` and the metric returns scalars, a named numeric vector of fold scores is returned. Otherwise a data frame with fold membership and scores per observation is returned. The convenience metrics return a numeric vector of scores or residuals.

See Also

[gamlss2](#), [log_pdf](#)

Examples

```
## Not run: ## air quality data
air <- subset(airquality, !is.na(Ozone))

## set the seed
set.seed(123)

## cross-validation using the GA distribution
## only model mu parameter with s(Temp)
cv1 <- cv_gamlss2(
  Ozone ~ s(Temp),
  data = air,
  family = GA, maxit = 100
)

## now, add effects
cv2 <- cv_gamlss2(
  Ozone ~ s(Temp) + s(Wind) | s(Temp),
  data = air,
  family = GA,
  folds = cv1$fold, maxit = 100
)

## evaluate log-likelihood
sum(cv1$score)
sum(cv2$score)

## evaluate folds with two workers, reusing the same fold assignments
cv_parallel <- cv_gamlss2(
  Ozone ~ s(Temp), data = air, family = GA,
  folds = cv1$fold, maxit = 100, cores = 2L
)
sum(cv_parallel$score)

## End(Not run)
```

Description

Extracts the deviance from fitted `gamlss2` objects. By default, the total deviance is returned. Optionally, observation-wise deviance contributions are returned.

Usage

```
## S3 method for class 'gamlss2'
deviance(object, ..., newdata = NULL, sum = TRUE)
```

Arguments

<code>object</code>	An object of class "gamlss2".
<code>...</code>	Optionally, further fitted <code>gamlss2</code> objects.
<code>newdata</code>	An optional data frame in which to evaluate the deviance. If omitted, the deviance is evaluated for the data used to fit the model.
<code>sum</code>	Should the observation-wise deviance contributions be summed? If <code>TRUE</code> , the default, the total deviance is returned. If <code>FALSE</code> , the individual deviance contributions are returned.

Details

The deviance of a fitted `gamlss2` model is defined as minus twice the fitted log-likelihood,

$$-2 \log L$$

. If `sum = TRUE`, one deviance value is returned for each supplied model. If more than one model is supplied, the deviances are ordered increasingly.

If `sum = FALSE`, the function returns the observation-wise deviance contributions,

$$-2 \log f(y_i | \hat{\theta}_i)$$

, where $f()$ is the fitted probability density or probability mass function and $\hat{\theta}_i$ denotes the fitted distribution parameters for observation i . These values sum to the total deviance and are useful, for example, in cross-validation.

Value

If `sum = TRUE`, a numeric vector with the total deviance of the supplied model or models. If `sum = FALSE`, a numeric vector of observation-wise deviance contributions for one model, or a matrix with one column per model if several models are supplied.

See Also

[gamlss2](#)

Examples

```
## Not run: ## specify a distributional model:
## location and scale vary with x,
## shape parameters remain constant
f <- dist ~ s(speed) | 1

## estimate model
m1 <- gamlss2(f, data = cars, family = GA)

## total deviance
deviance(m1)

## observation-wise deviance contributions
di <- deviance(m1, sum = FALSE)

## the contributions sum to the total deviance
sum(di)
deviance(m1)

## compare two models
m2 <- gamlss2(dist ~ s(speed) | s(speed), data = cars, family = GA)
deviance(m1, m2)

## compare observation-wise deviance contributions
di <- deviance(m1, m2, sum = FALSE)
head(di)

## deviance on new data
nd <- cars[1:5, ]
deviance(m1, newdata = nd)
deviance(m1, newdata = nd, sum = FALSE)

## End(Not run)
```

discretize_family

Discretize a Distribution Family into Fixed Bins

Description

Constructs a discrete distribution family by assigning the probability of each fixed interval of a base distribution to a corresponding response value.

Usage

```
discretize_family(family = NO, breaks,
  values = seq_len(length(breaks) - 1L), right = TRUE)
```

Arguments

family	A distribution family accepted by the gamlss2 family infrastructure.
breaks	A strictly increasing numeric vector of interval boundaries. The first and last elements must be <code>-Inf</code> and <code>Inf</code> .
values	A strictly increasing vector of finite numeric response values, one for each interval defined by breaks. By default, the intervals are labeled consecutively starting at one.
right	Logical. Should the intervals be right-closed, $(a, b]$, rather than left-closed, $[a, b)$?

Details

For each interval defined by adjacent elements of breaks, `discretize_family()` assigns the interval probability under the base distribution to the corresponding element of values. Thus, with `right = TRUE`, response value y_j has probability

$$P(Y \in (b_j, b_{j+1}])$$

under the base distribution. With `right = FALSE`, the intervals are left-closed instead.

The returned family provides probability mass, cumulative distribution, quantile, random-number, mean, and variance functions. Its distributional parameters and link functions are inherited from the base family.

Value

An object of class `"gamlss2.family"` representing the discretized distribution.

See Also

[family_extensions](#), [round_family](#), [gamlss2](#), [gamlss2.family](#)

Examples

```
## Not run:
## fixed intervals and their observed values
breaks <- c(-Inf, -1, 0, 1, Inf)
values <- 0:3

## discretize a normal family
fam <- discretize_family(N0, breaks = breaks, values = values)

## simulate data whose location depends on x
set.seed(6020)
n <- 500
x <- sort(runif(n, -3, 3))
par <- data.frame(mu = sin(x), sigma = exp(-0.5 + cos(x)))
y <- fam$random(par, n)
head(y)
df <- data.frame(y = y, x = x)
```

```
## plot count data
plot(y ~ x, data = df)

## fitted pdf
fit_family(df$y, family = fam)

## fit a regression model using the discretized family
m <- gamlss2(y ~ s(x) | ., data = df, family = fam)

## inspect the estimated effect and fitted probabilities
plot(m)
d <- predict(m)
head(pdf(d, y))

## predict median
p <- quantile(d, probs = 0.5)
plot(y ~ x, data = df)
lines(p ~ x, lwd = 2, col = 4)

## End(Not run)
```

elm

*Extreme Learning Machine Model Terms***Description**

Constructor function for Extreme Learning Machine (ELM) model terms for GAMLSS.

Usage

```
## Model term constructor function.
elm(x, k = 50, a = "tanh", ...)
```

Arguments

x	A numeric vector or matrix, a factor, or a formula. If x is a formula, a design matrix is created using <code>model.matrix</code> . See the examples.
k	Integer, number of hidden units (random features) used to build the ELM design matrix.
a	Character, activation function for the hidden units. Supported options are "logistic", "tanh" (default), "relu", "leaky_relu", "elu", "softplus", "atan", "softsign", "gaussian", "laplace", "sine", and "identity".
...	Further control arguments can be passed: <code>criterion = "bic"</code> (default) for shrinkage parameter selection and <code>scale = TRUE</code> (default) for internal scaling of the design matrix. The hidden-layer initialization can be controlled via <code>sampler = "halton"</code> (default), "pca", or "random", and via further tuning arguments such as <code>whiten</code> , <code>target_sd</code> , <code>slope_sd_1d</code> , <code>q_range</code> , and <code>orthogonalize</code> . Further arguments are passed to <code>model.matrix</code> if x is specified using a formula.

Details

The ELM term constructs a randomized single-hidden-layer representation of the covariate(s) in `x`. Internally, a design matrix `Z` is built (including an intercept column), random weights are sampled, and the linear predictors $Z \%*\% W$ are transformed through the activation function `a` to obtain the hidden-layer design matrix `X`. Columns of `X` are centered before estimation.

For numerical stability and comparability across terms, `Z` can be scaled internally (`scale = TRUE`). Intercept columns are left unchanged. If `x` is a factor, a QR-based group scaling is applied; otherwise, a column-wise center-and-scale normalization is used.

The activation is applied element-wise to $Z \%*\% W$. For numerical stability, bounded activations are evaluated on clipped inputs (currently $[-35, 35]$).

By default, hidden-unit weights are generated using a low-discrepancy Halton design in a whitened input space. This tends to produce more stable and less redundant features than purely random draws in many practical applications. With `sampler = "pca"`, directions are additionally aligned with principal component directions of the scaled design. The historical stochastic behavior can be recovered with `sampler = "random"`.

The default `sampler = "halton"` is deterministic given the input data. Use `set.seed` only if you explicitly request `sampler = "random"`.

Value

An object representing a `specials` model term, used internally by `gamlss2` during model fitting and prediction.

References

Huang GB, Zhu QY, Siew CK (2006). "Extreme Learning Machine: Theory and Applications". *Neurocomputing*, **70**(1–3), 489–501. doi:10.1016/j.neucom.2005.12.126

Huang GB, Zhu QY, Siew CK (2004). "Extreme Learning Machine: A New Learning Scheme of Feedforward Neural Networks". In *2004 IEEE International Joint Conference on Neural Networks*, **2**, 985–990. IEEE, Budapest, Hungary. doi:10.1109/IJCNN.2004.1380068

See Also

`gamlss2`, `specials`.

Examples

```
## Not run: ## load data
data("SpirometryUS")

## subset for female
d <- subset(SpirometryUS, gender == "female")

## the default sampler is deterministic; set
## a seed only for sampler = "random"
## formula for all 4 parameters
f <- fev1 ~ elm(age,k=100,a="tanh") | . | . | .
```

```

## estimate model
m <- gamlss2(f, data = d, family = BCT)

## estimated effects
plot(m)

## predict quantiles
qu <- c(0.025, seq(0.1, 0.9, by = 0.1), 0.975)
fit <- quantile(m, probs = qu)

## plot
plot(fev1 ~ age, data = d)
i <- order(d$age)
matlines(d$age[i], fit[i, ],
  lty = 1, lwd = 2, col = c(2, rep(4, ncol(fit) - 1)))

## main effects and interactions
f <- fev1 ~ s(age) + s(height) + s(weight) +
  elm(~age+height+weight,k=200) | . | . | .

m <- gamlss2(f, data = d, family = BCT)

## summary to inspect effect of interactions
## of the elm()s
summary(m)

## plot main effects
plot(m)

## quantile residuals
plot(m, which = "resid")

## prediction is handled automatically via
## the special term interface
n <- 50
nd <- with(d, expand.grid(
  "age" = seq(min(age), max(age), length = n),
  "weight" = seq(min(weight), max(weight), length = n)
))
nd$height <- mean(d$height)

## compute lower 2.5
nd$fit <- quantile(m, newdata = nd, probs = 0.025)

## visualize
n <- length(unique(nd$age))
age <- sort(unique(nd$age))
weight <- sort(unique(nd$weight))

z <- matrix(nd$fit, n, n)

image(age, weight, z,
  col = hcl.colors(100, "YlOrRd"),

```

```

      xlab = "age", ylab = "weight")
contour(age, weight, z, add = TRUE)

## End(Not run)

```

fake_formula

Extended Processing of "Fake" Formulas

Description

Create a simplified formula from a [formula](#), a [Formula](#), or a [list](#) of [formulas](#). The result retains the variables and transformations needed to build a [model.frame](#) while separating out special model terms. This is mainly used internally when **gamlss2** prepares formulas for fitting.

Usage

```

fake_formula(formula, specials = NULL,
             nospecials = FALSE, onlspecialchars = 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"?
onlspecialchars	Logical, should only the special model terms be returned?

Details

The function is primarily an internal helper for formula preprocessing in **gamlss2**, but it can also be useful for inspecting how complex formulas are decomposed before fitting.

Special model terms such as `s()`, `te()`, `ti()`, `re()`, or package-specific specials are identified and handled separately from ordinary variables. Depending on the control arguments:

- if `nospecials = FALSE`, variables used inside registered special terms are reintroduced into the returned formula so they remain available in the model frame;
- if `nospecials = TRUE`, the returned formula excludes the special terms and their variables;
- if `onlspecialchars = TRUE`, the function returns the names of the detected special terms instead of a formula object.

For multi-part or multi-parameter formulas, the output preserves the overall formula structure.

Value

Depending on the input, the function returns a [formula](#) or [Formula](#). If `onlspecialchars = TRUE`, a character 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, onllyspecials = TRUE)
print(ff)
```

family.distribution *Generate a 'gamlss2' Family from a 'distributions3' Object*

Description

Set up a [gamlss2.family](#) object based on an object inheriting from "distribution" (as provided by the **distributions3** package).

Usage

```
## S3 method for class 'distribution'
family(
  object,
  links,
  score = TRUE,
  hessian = FALSE,
  update = FALSE,
  expected = FALSE,
  ...
)

## S3 method for class 'GAMLSS'
family(object, ...)

## S3 method for class 'gamlss.family'
family(object, score = TRUE, hessian = FALSE, update = FALSE, ...)

distributions3_family(
  distribution,
  links,
  score = TRUE,
  hessian = FALSE,
  update = FALSE,
  expected = FALSE,
  create_distribution = "structure",
  type = NULL,
  valid.response = NULL,
  initialize = NULL,
  ...
)
```

Arguments

object	an object specifying a distribution, typically created by <code>Normal</code> or <code>Binomial</code> or <code>GAMLSS</code> . Additionally, a <code>gamlss.family</code> from the <code>gamlss.dist</code> package is also supported.
links	named vector or list. Specification of one link per parameter to be modeled, either via a character string to be passed to <code>make.link2</code> or a <code>link-gamlss2</code> object (inheriting from <code>link-glm</code>).
score	logical. Should the <code>score</code> method for the distribution object be used to compute the scores? If set to <code>FALSE</code> they are approximated numerically.
hessian	logical. Should the <code>hessian</code> method for the distribution object be used to compute the Hessians? If set to <code>FALSE</code> they are approximated numerically.
update	logical. Should the <code>score</code> and <code>hessian</code> methods for the distribution object be used to compute the updates of the linear predictor and the corresponding weights? If set to <code>FALSE</code> they are approximated numerically.
expected	logical. Should the expected rather than the observed Hessian be used?
...	arguments passed from the methods to the workhorse function <code>distributions3_family</code> .
distribution	character. Name of the class and distribution object creator, e.g., <code>"Normal"</code> or <code>"Binomial"</code> .
create_distribution	either a function or one of the character strings <code>"structure"</code> or <code>"do.call"</code> . This specifies how to create a distribution object from a (fitted) parameter list (typically from <code>gamlss2</code>). Either a dedicated function can be specified or such a function is set up internally. If <code>create_distribution = "structure"</code> then only the class attribute is added in a <code>structure()</code> call. If set to <code>"do.call"</code> then the distribution object is created via <code>do.call()</code> to the class creator (such as <code>Normal</code> etc.). The latter is safer but requires much more overhead.
type	character. Specification of the type of distribution (<code>"continuous"</code> vs. <code>"discrete"</code>).
valid.response	function. A function(<code>x</code>) returning a logical indicating whether <code>x</code> is a valid response for the distribution.
initialize	list of functions. A named list with one function(<code>y, ...</code>) per parameter to be modeled, returning an initialization for this parameter.

Details

`distributions3_family` maps fitted parameter lists to **distributions3** objects and uses their methods for fitting and assessing the model. The family methods call this function and infer some family settings automatically.

Value

A list inheriting from `gamlss2.family`.

Examples

```
library("distributions3")
f <- family(Poisson(), links = c(lambda = "log"))
str(f)
```

family_extensions	<i>Extend Distribution Families</i>
-------------------	-------------------------------------

Description

Extends an existing distribution family by transforming, conditioning, coarsening, or adding point masses. The resulting "gamlss2.family" objects can be used directly in [gamlss2](#) models, and family extensions can be nested to compose several operations.

Usage

```
transform_family(family = NO, transform, inverse, log_jacobian,
  increasing = TRUE, support = c(-Inf, Inf), valid.response = NULL,
  name = NULL)
```

```
logit_family(family = NO)
```

```
truncate_family(family = NO, lower = -Inf, upper = Inf,
  include = c(FALSE, TRUE))
```

```
censor_family(family = NO, lower = -Inf, upper = Inf)
```

```
inflate_family(family = NO, at = 0)
```

```
hurdle_family(family = PO, at = 0)
```

```
round_family(family = NO, digits = 0L)
```

Arguments

family	A distribution family accepted by the gamlss2 family infrastructure, such as NO or PO, or another modified family.
transform	A fixed monotone function mapping observations from the base scale to the response scale.
inverse	The inverse function of transform.
log_jacobian	A function returning the logarithm of the absolute derivative of inverse, evaluated on the response scale.
increasing	Logical. Is transform increasing?
support	Numeric vector with the two response-scale support bounds. These bounds are also used when evaluating the CDF outside the transformed support.
valid.response	Optional function returning logical values that indicate whether observations lie in the transformed support.
name	Optional character string used as the extended family label.

lower, upper	Fixed scalar lower and upper truncation or censoring boundaries. For <code>censor_family()</code> , leaving both at their infinite defaults constructs an observation-specific right-censored family for a <code>Surv</code> response.
include	Logical vector of length two indicating whether the lower and upper truncation boundaries are retained. The default interval is $(lower, upper]$.
at	One or more fixed inflation points, or one fixed hurdle point.
digits	Integer number of decimal places retained by rounding.

Details

For a fixed monotone transformation $Y = g(X)$, `transform_family` uses

$$\log f_Y(y) = \log f_X(g^{-1}(y)) + \log |dg^{-1}(y)/dy|.$$

Increasing and decreasing transformations are supported. The convenience function `logit_family` constructs the logistic push-forward, which has strict support $0 < Y < 1$. Exact zeros and ones are not valid observations for this continuous transformation.

The function `truncate_family` conditions the base distribution on the interval selected by `lower`, `upper`, and `include`. Its density or probability mass is divided by the probability of the retained interval. With at least one finite boundary, `censor_family` observes

$$Y = \min(\max(X, lower), upper)$$

and consequently creates point masses at finite censoring boundaries. With both boundaries left infinite, it instead accepts a right-censored survival::`Surv(time, status)` response. An observed event contributes the base density or mass at time; a censored observation contributes the base survival probability beyond time.

For inflation points c_j , `inflate_family` constructs

$$F(y) = \pi_0 F_0(y) + \sum_j \pi_j I(c_j \leq y).$$

Natural base-distribution mass at an inflation point is retained. The appended parameters `pi1`, `pi2`, and so on have log links and represent positive odds w_j relative to the base component, not probabilities. Thus,

$$\pi_0 = \frac{1}{1 + \sum_j w_j}, \quad \pi_j = \frac{w_j}{1 + \sum_\ell w_\ell}.$$

This is the baseline-odds parameterization also used by `mixture_family`. Quantiles account for every probability jump.

The function `hurdle_family` instead constructs

$$F(y) = \pi_1 I(c \leq y) + \pi_0 F_0(y \mid X \neq c).$$

Here `pi1` is also the odds w_1 , so the hurdle probability is $w_1/(1 + w_1)$. This is the complete mass at c , including when the base distribution already assigns positive mass to that point.

The function `round_family` uses intervals centered on a decimal grid and is restricted to continuous base distributions. Its quantile and random-number functions follow R's `round()` convention. For fixed, user-defined intervals, see `discretize_family`.

Inflated, hurdle, and fixed-bin families provide exact means and variances when the required base-family moments are available. Censored and rounded moments requiring unavailable partial moments report an error.

Continuous distributions with added point masses provide randomized quantile residuals. Bounds, atom locations, breaks, and transformations are fixed when the family is constructed. Other extensions requiring numerical normalization or derivatives during fitting are not provided.

Value

An object of class "gamlss2.family". Existing base-family parameters retain their names, order, and links. Inflation and hurdle families append positive-odds parameters named `pi1`, `pi2`, and so on.

See Also

[gamlss2](#), [gamlss2.family](#), [discretize_family](#), [mixture_family](#), [distributions3_family](#)

Examples

```
## Not run: ## simulate data
set.seed(6020)
n <- 300
x <- sort(runif(n, -3, 3))

## logistic transformation of a normal distribution
fam <- logit_family(N0)
y <- fam$random(list(mu = sin(x), sigma = exp(-1 + cos(x))), n)
hist(y)
m <- gamlss2(y ~ s(x) | s(x), family = fam)
plot(m)

## zero-truncated Poisson distribution
fam <- truncate_family(P0, lower = 0)
y <- fam$random(list(mu = exp(1/2 + sin(x))), n)
m <- gamlss2(y ~ s(x), family = fam)
plot(x, y)

## small helper for plotting fitted quantiles
fq <- function(model) {
  matlines(x,
    quantile(model, probs = c(0.05, 0.5, 0.95)),
    lty = 1, lwd = 2, col = 4
  )
}

fq(m)

## normal distribution censored at both boundaries
fam <- censor_family(N0, lower = -1, upper = 1)
y <- fam$random(list(mu = sin(x), sigma = exp(-1 + cos(x))), n)
hist(y)
m <- gamlss2(y ~ s(x) | s(x), family = fam)
```

```

plot(m)
plot(x, y)
fq(m)

## zero-inflated Poisson distribution
fam <- inflate_family(P0, at = 0)
## pi1 is the added-mass odds; its mixture probability is plogis(cos(x))
y <- fam$random(list(mu = exp(1/2 + sin(x)), pi1 = exp(-1 + cos(x))), n)
barplot(table(y))
m <- gamlss2(y ~ s(x) | s(x), family = fam)
plot(m)
plot(x, y)
fq(m)

## zero-hurdle Poisson distribution
fam <- hurdle_family(P0, at = 0)
## pi1 is the hurdle odds; the probability of zero is plogis(cos(x))
y <- fam$random(list(mu = exp(1/2 + sin(x)), pi1 = exp(cos(x))), n)
barplot(table(y))
m <- gamlss2(y ~ s(x) | s(x), family = fam)
plot(m)
plot(x, y)
fq(m)

## decimal rounding
fam <- round_family(N0, digits = 1)
y <- fam$random(list(mu = sin(x), sigma = exp(-1 + cos(x))), n)
m <- gamlss2(y ~ s(x) | s(x), family = fam)
plot(m)
plot(m, which = "resid")

## End(Not run)

```

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, ...)

## find gamlss2 model
find_gamlss2(formula, families = NULL, k = 2,
  select = FALSE, verbose = TRUE, ...)
```

Arguments

type	Character, is the response continuous or discrete?
families	Character, the names of the family objects that should be returned.
y	The response vector or matrix.
k	Numeric, the penalty factor that should be used for the GAIC .
verbose	Logical, should runtime information be printed?
family	A family object that should be used for estimation, see also gamlss2.family .
plot	Logical, should a plot of the fitted density be provided?
formula	A model formula, see gamlss2 .
select	Logical, if set to select = TRUE, model term selection is enforced using model fitting function select_gamlss2 .
...	For function available_families(), arguments passed to the family objects, e.g., for setting link functions mu.link = "log". 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(). For function find_gamlss2(), arguments are passed to available_families() and gamlss2 .

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 values sorted in decreasing order of the selected fit criterion, so smaller values indicate a better fit.

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. The best model is the model with the lowest [GAIC](#).

Function `fit_family()` returns the fitted intercept-only model.

Function `find_gamlss2()` returns the final selected model.

See Also

[gamlss2](#).

Examples

```
## Not run: ## find a suitable response to the response
ic <- find_family(airquality$Ozone)
print(ic)

## fit parameters using the WEI family
fit_family(airquality$Ozone, family = WEI)

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

## fit parameters using the ZANBI family
fit_family(InsectSprays$count, family = ZANBI)

## search complete model
## m <- find_gamlss2(Ozone ~ s(Temp) + s(Wind) | . | . | .,
##   data = airquality, select = TRUE, mu.link = "log")

## End(Not run)
```

fk

Free-Knot Regression Splines for GAMLSS

Description

Constructs a univariate regression spline whose interior knot locations, and optionally the number of interior knots, are selected from the current weighted working response.

Usage

```
fk(x, k = 10L, nk = NULL, degree = 1L,
  criterion = "bic", knots = NULL, K = NULL,
  reselect = FALSE, optim.control = list(), ...)
```

Arguments

x	A numeric covariate.
k	Integer, the largest number of free interior knots considered when nk = NULL. Candidate models from one through k interior knots are compared.
nk	Integer, optional fixing the number of free interior knots. The locations are still estimated.
degree	Integer, the polynomial degree of the B-spline basis. The default 1 gives a piecewise-linear spline.

criterion	Character selection criterion. Available choices are "bic", "aic", "gaic", "aicc", and "gcv".
knots	Optional numeric vector of fixed interior knot locations. The values must be distinct and strictly inside the range of x. When x is transformed, locations are specified on the transformed scale. Supplying knots overrides automatic location optimization.
K	Optional nonnegative penalty multiplier used by criterion = "gaic". The model-level value, or 2, is used when NULL.
reselect	Logical. If FALSE, the number of knots selected during the first backfitting update is retained and its locations are warm-started in later updates. If TRUE, the number is selected again in every update.
optim.control	Named list passed as the control argument to optim for knot-location optimization.
...	Additional entries merged into optim.control.

Details

For each candidate number of interior knots, `fk()` uses a B-spline basis with fixed, repeated boundary knots. Interior knots are represented by positive ordered gaps. This removes permutation ambiguity and prevents knot crossings during optimization.

Conditional on the knot locations, spline coefficients are profiled out by weighted least squares. Knot optimization therefore only operates on the interior locations. The fitted model term is centered.

When `nk = NULL`, models with one through `k` interior knots are compared using `criterion`. Effective degrees of freedom include both the centered spline coefficients and the estimated interior knot locations. Fixed locations supplied through `knots` do not add knot-location degrees of freedom.

Value

The constructor returns an object of classes "special" and "fk".

See Also

[gamlss2](#), [special_terms](#)

Examples

```
## Not run: ## load data
data("SpirometryUS")

## subset for female
d <- subset(SpirometryUS, gender == "male")

## Automatically select one free interior knot
m <- gamlss2(fev1 ~ fk(age, nk = 1) | ., data = d, family = NO)

## estimated effects
plot(m)
```

```
## predict quantiles
qu <- c(0.025, seq(0.1, 0.9, by = 0.1), 0.975)
fit <- quantile(m, probs = qu)

## plot
plot(fev1 ~ age, data = d)
i <- order(d$age)
matlines(d$age[i], fit[i, ],
  lty = 1, lwd = 2, col = c(2, rep(4, ncol(fit) - 1)))

## End(Not run)
```

gamlss2

Generalized Additive Models for Location Scale and Shape

Description

The function `gamlss2()` fits generalized additive models for location, scale, and shape (GAMLSS). It estimates one additive predictor for each parameter of the chosen response distribution, such as μ , σ , ν , or τ . The number and meaning of these parameters are determined by the selected family object, see [gamlss2.family](#).

In addition to ordinary linear terms, `gamlss2()` supports smooth terms from **mgcv** as well special user-defined model-terms, see [specials](#).

Usage

```
gamlss2(formula, ...)

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

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

Arguments

formula	A GAM-type formula or Formula describing the additive predictors for the distribution parameters. All smooth terms from mgcv are supported, see also formula.gam . For <code>gamlss2.list()</code> , formula is a list of formulas .
data	A data frame, list, or environment containing the variables in the model. If not found in data, the variables are taken from <code>environment(formula)</code> , typically the environment from which <code>gamlss2()</code> is called.
family	A <code>gamlss.family</code> or <code>gamlss2.family</code> object defining the response distribution and the link functions of its 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 <code>model.frame</code> .
weights	An optional vector of prior weights to be used in the fitting process. Should be NULL or a numeric vector.
offset	Optional offset terms to be included in the linear predictors during fitting. If a single numeric vector is supplied, it is assigned to the first parameter of the distribution. If offsets are needed for several parameters, provide a data frame or list in the same order as the parameter names in the family object.
start	Optional starting values for the estimation algorithm, see <code>gamlss2_start</code> .
knots	An optional list of user-specified knots, see <code>smoothCon</code> .
control	A list of control arguments, see <code>gamlss2_control</code> .
...	Arguments passed to <code>gamlss2_control</code> .

Details

The function `gamlss2()` is the main entry point for fitting distributional regression models in **gamlss2**.

- The response distribution is specified through a family object, either from **gamlss.dist** or created as a `gamlss2.family` object.
- By default, estimation is carried out with the **RS** algorithm. The optimizer can be changed through `gamlss2_control`. A family object may also provide its own optimizer.
- The returned object depends on the chosen optimizer. In the standard case, this is an object of class "gamlss2", for which summary, plotting, prediction, and extractor methods are available.

Value

The object returned by the selected optimizer. By default, this is an object of class "gamlss2" produced by **RS**. Standard methods and extractor functions are available for this class.

See Also

[RS](#), [gamlss2_control](#), [gamlss2.family](#), [gamlss2_start](#)

Examples

```
## air quality data
air <- subset(airquality, !is.na(Ozone))

## positive skewed response
hist(air$Ozone)

## specify a distributional model
f <- Ozone ~ s(Temp) + s(Wind) | s(Temp)

## estimate model
```

```
m1 <- gamlss2(f, data = air, family = GA)

## model summary
summary(m1)

## plot parameter-specific effects
plot(m1, which = "effects")

## plot diagnostics
plot(m1, which = "resid")

## predict distribution parameters
par <- predict(m1, parameter = c("mu", "sigma"))
print(head(par))

## create new data:
## vary temperature while holding wind constant
nd <- with(air, data.frame(
  Temp = seq(min(Temp), max(Temp), length.out = 100),
  Wind = median(Wind)
))

## fitted conditional quantiles
pq <- quantile(
  m1,
  newdata = nd,
  probs = c(0.05, 0.5, 0.95)
)

## visualize
plot(Ozone ~ Temp, data = air, pch = 19,
  col = rgb(0.1, 0.1, 0.1, alpha = 0.3),
  xlab = "Temperature (degrees F)",
  ylab = "Ozone (ppb)")
matlines(nd$Temp, pq,
  lwd = 2,
  lty = c(2, 1, 2),
  col = 4)

## exceedance probabilities
nd5 <- with(air, data.frame(
  Temp = seq(min(Temp), max(Temp), length.out = 5),
  Wind = min(Wind)
))

## use distributions3 methods
d <- predict(m1, newdata = nd5)
nd5$Probs <- 1 - cdf(d, 90)

barplot(
  nd5$Probs,
  names.arg = round(nd5$Temp),
  xlab = "Temperature (degrees F)",
```

```

    ylab = "Pr(Ozone > 90 ppb)",
    ylim = c(0, 1)
  )

  ## count response example using HarzTraffic data
  data("HarzTraffic", package = "gamlss2")

  ## seasonal count model for motorcycles
  m2 <- gamlss2(
    bikes ~ s(yday, bs = "cc") | s(yday, bs = "cc"),
    data = HarzTraffic,
    family = NBI
  )

  ## summary
  summary(m2)

  ## effect plots
  plot(m2)

  ## residual diagnostics
  plot(m2, which = "resid")

  ## quantiles over the day of the year
  nd <- data.frame(yday = 1:365)
  d <- predict(m2, newdata = nd)
  pq <- quantile(d, probs = c(0.05, 0.5, 0.95))

  ## visualize fitted quantiles
  plot(bikes ~ yday, data = HarzTraffic, pch = 19,
    col = gray(0.1, alpha = 0.3))
  matlines(nd$yday, pq, lty = c(2, 1, 2),
    lwd = 2, col = 4)

```

Description

The GAMLSS2 family is a unified class with corresponding methods that represent all distributional families supported by the **gamlss2** package. It enables seamless integration with the **distributions3** workflow and provides a consistent interface for model fitting and distributional computations.

Usage

```
GAMLSS2(family, parameters)
```

Arguments

family	character. Name of a <code>gamlss2.family</code> or a family provided by the <code>gamlss.dist</code> package, e.g. <code>NO</code> or <code>BI</code> for the normal or binomial distribution, respectively.
parameters	numeric, matrix, list or data frame, see the examples.

Details

The S3 class `GAMLSS2` is a slightly more general implementation of the S3 class `GAMLSS` tailored for `gamlss2`. For details please see the documentation of `GAMLSS`

Value

A `GAMLSS2` object, 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

`gamlss2.family`

Examples

```
## package and random seed
library("distributions3")
set.seed(6020)

## one normal distribution
X <- GAMLSS2("NO", c(mu = 1, sigma = 2))
X

## two normal distributions
X <- GAMLSS2("NO", cbind(c(1, 1.5), c(0.6, 1.2)))
X

## three Weibull distributions
X <- GAMLSS2("WEI", list(mu = c(1, 1, 2), sigma = c(1, 2, 2)))
X

## see ?gamlss.dist::GAMLSS for the remainder of this example
```

Description

A "gamlss2.family" object describes a response distribution for use with **gamlss2**. It tells the fitting algorithm which distributional parameters exist, which link functions they use, and which distributional functions are available, such as densities, quantiles, random-number generators, or score functions.

Usage

```
## S3 method for class 'gamlss2'
family(object, ...)
## S3 method for class 'gamlss2.family'
print(x, full = TRUE, ...)
```

Arguments

object	A fitted "gamlss2" model object.
x	A "gamlss2.family" object.
full	Logical; print available family-specific and derivative functions?
...	Not used currently.

Details

A **gamlss2** family object is a list of class "gamlss2.family". It defines the response distribution and the functions needed to fit and use that distribution in a model.

The minimum required elements are:

- family: the name of the distribution (character string).
- names: a character vector of parameter names (e.g., c("mu", "sigma")).
- links: a named character vector specifying the link function for each parameter (e.g., c(mu = "identity", sigma = "log")), or a list of link functions, see [softplus](#).
- pdf(par, y, log = FALSE, ...): a function to evaluate the (log-)density.

The pdf() function must accept the response y, a named list par of evaluated parameter values (e.g., par\$mu, par\$sigma), a logical log, and optional additional arguments.

Further elements can be added depending on how fully the family should support fitting, prediction, diagnostics, and simulation. Common optional elements are:

- score: a named list of functions (one per parameter), each computing the first derivative of the log-likelihood with respect to the linear predictor: score[[param]](par, y, ...).
- hessian: a named list of functions computing second derivatives (the negative Hessian). For parameters mu and sigma, this includes: hessian[["mu"]](par, y, ...), hessian[["sigma"]](par, y, ...), and optionally cross derivatives like hessian[["mu:sigma"]](par, y, ...).

- `log_likelihood(par, y, ...)`: a function computing the total log-likelihood.
- `cdf(par, y, ...)`: cumulative distribution function.
- `quantile(par, p, ...)`: quantile function.
- `support(par, ...)`: lower and upper response support. It must return a numeric matrix with columns `min` and `max`, with one row per parameter combination. A fixed numeric vector `c(min, max)` can be supplied instead. If support is omitted but a quantile function is available, it is inferred from the quantiles at probabilities zero and one. Without either, a discrete family is assumed to be a count distribution on the nonnegative integers. For a continuous family with a density, a parameter-dependent support is inferred heuristically by probing the log-density. Explicit support is recommended for shifted or truncated counts and for unusual, disconnected, or numerically extreme continuous supports.
- `random(par, n, ...)`: random number generator.
- `mean(par, ...)`: mean function.
- `variance(par, ...)`: variance function.
- `skewness(par, ...)`, `kurtosis(par, ...)`: optional higher-order moment functions.
- `initialize`: a named list of initialization functions, one for each parameter (e.g., `initialize$mu(y, ...)`), used to generate starting values.
- `valid.response(x)`: a function that checks whether the response is valid (e.g., numeric, non-factor).
- `optimizer()`: an optional function defining a custom optimization method for use with `gamlss2`, see also [RS](#).

If analytical score or Hessian functions are not provided, they are approximated numerically. If `cdf()` is omitted for a family with an available density and support, it is constructed by numerical integration for a continuous family or by summing the probability mass over the integer support for a discrete family. Analytical CDFs are retained unchanged. Numerical CDFs accept `lower.tail`, `log.p`, `rel.tol`, and `abs.tol`; continuous CDFs additionally accept subdivisions, while discrete CDFs accept `max.terms`.

If `quantile()` is omitted but a CDF and support are available, it is constructed by numerical inversion. Continuous CDFs are inverted by bracketing and bisection. For discrete families, the generated function returns the smallest integer whose CDF is at least the requested probability. Numerical quantiles accept `lower.tail`, `log.p`, `tol`, and `maxiter`; quantiles generated directly from a count PMF additionally accept `max.terms`. Analytical quantile functions are retained unchanged.

If `mean()` or `variance()` is omitted for a family with a density and support, the missing functions are constructed numerically. Continuous moments use adaptive integration; the variance is integrated as a centered second moment. Discrete moments are obtained by batched summation over the integer support. Numerical moments accept `rel.tol` and `abs.tol`; continuous moments additionally accept subdivisions, while discrete moments accept `max.terms`. A failure to establish a finite moment is reported as an error. Analytical moment functions are retained unchanged.

If `random()` is omitted but a quantile function is available, random values are generated by inverse transform sampling. Its `...` arguments are forwarded to the quantile function. Existing random-number generators are retained unchanged.

Value

For `family.gamlss2()`, the family object stored in a fitted "gamlss2" model is returned. A "gamlss2.family" object itself is a list describing the response distribution, its parameters, their link functions, and the distribution-specific methods used during fitting, prediction, diagnostics, and simulation.

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(par, y, ...) {
        (y - par$mu) / (par$sigma^2)
      },
      "sigma" = function(par, y, ...) {
        -1 + (y - par$mu)^2 / (par$sigma^2)
      }
    ),
    "hessian" = list(
      "mu" = function(par, y, ...) {
        1 / (par$sigma^2)
      },
      "sigma" = function(par, y, ...) {
        rep(2, length(y))
      },
      "mu:sigma" = function(par, y, ...) {
        rep(0, length(y))
      }
    ),
    "log_likelihood" = function(par, y, ...) {
      sum(dnorm(y, par$mu, par$sigma, log = TRUE))
    },
    "pdf" = function(par, y, log = FALSE) {
      dnorm(y, mean = par$mu, sd = par$sigma, log = log)
    },
    "cdf" = function(par, y, ...) {
      pnorm(y, mean = par$mu, sd = par$sigma, ...)
    },
    "random" = function(par, n) {
      rnorm(n, mean = par$mu, sd = par$sigma)
    },
    "quantile" = function(par, p) {
      qnorm(p, mean = par$mu, sd = par$sigma)
    },
  )
}
```

```

    "support" = c(-Inf, Inf),
    "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)
}

## specify the model formula
f <- Ozone ~ s(Wind) | s(Wind)

## estimate model
m1 <- gamlss2(f, data = airquality, family = Normal)

## plot estimated effects
plot(m1, which = "effects")

## plot diagnostics
plot(m1, which = "resid")

## predict distributions
nd <- model.frame(m1)
nd <- nd[order(nd$Wind), ]
d <- predict(m1, newdata = nd)

## predict quantiles
pq <- quantile(d, probs = c(0.05, 0.5, 0.95), elementwise = FALSE)

## visualize
plot(Ozone ~ Wind, data = nd,
     pch = 19, col = rgb(0.1, 0.1, 0.1, alpha = 0.3),
     ylim = range(nd$Ozone, pq))
matlines(nd$Wind, pq, lwd = 2,
        lty = 1, col = 4)

## another example that defines only the density
## function; in this case all derivatives are
## approximated numerically. For residual diagnostics,
## the $cdf() and $quantile() functions are needed as well
Gamma <- function(...) {
  fam <- list(
    "family" = "Gamma",

```

```

    "names" = c("mu", "sigma"),
    "links" = c("mu" = "log", "sigma" = "log"),
    "pdf" = function(par, y, log = FALSE, ...) {
      shape <- par$sigma
      scale <- par$mu/par$sigma
      dgamma(y, shape = shape, scale = scale, log = log)
    },
    "cdf" = function(par, y, lower.tail = TRUE, log.p = FALSE) {
      shape <- par$sigma
      scale <- par$mu/par$sigma
      pgamma(y, shape = shape, scale = scale,
        lower.tail = lower.tail, log.p = log.p)
    },
    "quantile" = function(par, p, 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)
}

## model formula
f <- Ozone ~ s(Temp) + s(Wind) | .

## estimate model
m2 <- gamlss2(f, data = airquality, family = Gamma)

## visualize estimated effects
plot(m2, which = "effects")

## diagnostics, needs the $cdf() and $quantile() function!
plot(m2, which = "resid")

## End(Not run)

```

gamlss2_control

Control Parameters

Description

Control parameters for fitting GAMLSS models with [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 to be used for fitting. Use JR for joint REML estimation.
trace	Logical, should information be printed while the algorithm is running?
flush	Logical, whether to use flush.console to display current output in the console.
light	Logical, if set to <code>light = TRUE</code> , 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 intercept-only formulas be added automatically?
model	Logical, should the <i>model frame</i> be included as a component of the returned object.
x	Logical, indicating whether the model matrix should be included as a component of the returned object.
y	Logical, should the response be included as a component of the returned object.
fixed	Named logical vector indicating which parameters should be fixed during estimation. See gamlss2_start for examples.
...	Further control parameters to be included in the return value, for example parameters used by the optimizer function RS .

Details

The set of control parameters can be extended. For example, if a different optimizer is used, newly specified control parameters are passed on to that optimizer automatically.

Optimizer-specific controls and algorithm details are documented in [RS](#) and [CG](#), and in [JR](#) for joint REML estimation. Smoothing parameter selection is described in [gamlss2_smoothing](#).

Value

A list with the arguments specified.

See Also

[RS](#), [CG](#), [JR](#), [gamlss2](#), [gamlss2_start](#), [gamlss2_smoothing](#)

Examples

```
## Not run: ## specify the model formula
f <- dist ~ s(speed) | s(speed)

## specify fitting controls
ctrl <- gamlss2_control(trace = FALSE, maxit = c(50, 100))
m <- gamlss2(f, data = cars, family = GA, control = ctrl)

## End(Not run)
```

Description

Smoothing parameter selection for smooth terms in models fitted with [gamlss2](#).

Details

For **mgcv**-based smooth terms, smoothing parameters are selected by local REML updates by default (`criterion = "reml"`). These updates use the current working response and weights. With one penalty matrix this is the usual Schall update; with multiple penalty matrices all smoothing parameters are updated together using the extended Fellner–Schall trace equations. This is a local working-model REML update. Explicit choices such as `"gcv"`, `"aic"`, `"aicc"`, `"bic"`, and `"ncv"` retain their respective selection methods.

These local updates are used by the [RS](#) and [CG](#) optimizers. With `optimizer = JR`, coefficients and smoothing parameters are estimated jointly by Laplace approximate REML; see [JR](#).

Neighbourhood cross-validation can be selected with `criterion = "ncv"`. If `ncv` is omitted, singleton neighbourhoods are used, corresponding to leave-one-observation-out cross-validation. For ordered data, a numeric `ncv = k` uses the observation together with up to `k` neighbours on either side. Larger neighbourhoods generally provide more smoothing and can account for short-range dependence, at increased computational cost. For ordered data, small values such as `ncv = 1` or `ncv = 2` are often preferable to the singleton default. Custom neighbourhoods can also be supplied as a list through a smooth term's `xt` argument, for example `s(x, xt = list(ncv = neighbourhoods))`, where `neighbourhoods` has one integer index vector per observation and each vector contains its target observation.

All methods return coefficients and covariance at the reported smoothing parameters.

For the [RS](#) and [CG](#) optimizers, argument `demmler.reinsch` controls the use of a weighted Demmler–Reinsch reparameterization when selecting the smoothing parameter of an eligible single-penalty smooth, including multivariate smooths. The default, `"auto"`, uses the native reparameterization for local REML updates, and for quadratic criteria after the initial outer iteration. Set `demmler.reinsch = TRUE` to force it for eligible updates, or `FALSE` to disable it. For quadratic criteria, the initial outer iteration continues to use the direct solver; with numerical gradients or `native.wfit = FALSE`, automatic reparameterization is delayed until the current search reaches the edge of its one-decade search window.

Within an [RS](#) or [CG](#) fit, the reparameterization is cached while the design matrix, weights, penalty, and binning are unchanged. A new working response only requires updating its projection into the cached basis. The declared penalty rank keeps the null space unpenalized even at large smoothing parameters.

Analytic gradients are used by default for quadratic smoothing criteria (GCV, AIC, GAIC, AICc, and BIC), including smooths with multiple penalties. Set `analytic.gradient = FALSE` to use numerical gradients. Criteria using `logLik = TRUE` continue to use numerical gradients, with native direct solves and family likelihood evaluation in R. Nearly exact fits use observation-space residuals to avoid cancellation in the quadratic RSS.

Smooths with fixed smoothing parameters and multiple-penalty smooths use the direct solver. For local REML, the common range of the penalty matrices is cached and the coupled update is evaluated in native code. Single-penalty updates use the constrained penalty null space and the Demmler–Reinsch basis when available.

References

Demmler A, Reinsch C (1975). “Oscillation Matrices with Spline Smoothing”. *Numerische Mathematik*, **24**(5), 375–382. doi:10.1007/BF01437406

Wood SN (in press). “On Neighbourhood Cross Validation”. *Statistical Science*. Preprint: doi:10.48550/arXiv.2404.16490

See Also

[gamlss2](#), [gamlss2_control](#), [RS](#), [CG](#), [JR](#), [pb](#)

Examples

```
## Not run: ## specify the model formula
f <- dist ~ s(speed) | s(speed)

## estimate models with different smoothing criteria
m0 <- gamlss2(f, data = cars, family = GA, criterion = "reml")
m1 <- gamlss2(f, data = cars, family = GA, criterion = "aic")
m2 <- gamlss2(f, data = cars, family = GA, criterion = "bic")
m3 <- gamlss2(f, data = cars, family = GA, criterion = "gaic")
m4 <- gamlss2(f, data = cars, family = GA, criterion = "gaic", K = 4)
m5 <- gamlss2(f, data = cars, family = GA, criterion = "ncv")
m6 <- gamlss2(f, data = cars, family = GA, criterion = "ncv", ncv = 2)

plot(c(m0, m1, m2, m3, m4, m5, m6))

## End(Not run)
```

gamlss2_start

Model Initialization and Parameter Constraints

Description

Guidance on supplying starting values through argument `start` in [gamlss2](#) and on fixing distribution parameters via `fixed` in [gamlss2_control](#).

Details

The `start` argument can be used to initialize estimation in several ways. Supported inputs include:

- a named numeric vector or list giving one starting value per distribution parameter, such as `mu`, `sigma`, `nu`, or `tau`;

- a data frame or matrix with one column per distribution parameter and one row per observation, providing starting values for the full additive predictors on the link scale;
- a fitted "gamlss2" object, in which case fitted() values are used as starting values for all distribution parameters;
- a coef.gamlss2 object, allowing coefficient-wise initialization of linear and smooth terms.

Parameters can be fixed during estimation using the fixed control argument. This is useful, for example, when shape parameters should remain constant while the remaining parameters are estimated.

Starting values from a previous fit can reduce optimizer iterations. For coefficient-wise initialization, coef(object, full = TRUE) includes smooth coefficients; lambdas = TRUE also includes smoothing parameters. Compare iteration counts using the same model, observations, and controls.

See Also

[gamlss2](#), [gamlss2_control](#), [RS](#)

Examples

```
## Not run: air <- subset(airquality, !is.na(Ozone))

## hold the smoothing parameter fixed for comparable refits
f <- Ozone ~ s(Wind, sp = 1) | 1
ctrl <- gamlss2_control(maxit = 100, eps = 1e-7)

## scalar starting values for the distribution parameters
m1 <- gamlss2(f, data = air, family = GA,
  start = c(mu = 200, sigma = 0.1), control = ctrl)

## reuse fitted additive predictors on the link scale
m2 <- gamlss2(f, data = air, family = GA,
  start = fitted(m1, type = "link"), control = ctrl)

## coef() without full = TRUE initializes only the linear coefficients
m3 <- gamlss2(f, data = air, family = GA,
  start = coef(m1), control = ctrl)

## reuse all linear and smooth coefficients from the previous fit
m4 <- gamlss2(f, data = air, family = GA,
  start = coef(m1, full = TRUE, lambdas = TRUE, dropall = FALSE),
  control = ctrl)

## full coefficient starts reduce iterations in this example
## check convergence and compare the final log-likelihoods as well
fits <- list(scalar = m1, predictors = m2, linear = m3, full = m4)
data.frame(
  iterations = sapply(fits, function(m) m$iterations),
  converged = sapply(fits, function(m) m$converged),
  logLik = sapply(fits, function(m) as.numeric(logLik(m)))
)
```

```
## fix BCT shape parameters at their supplied starting values
m5 <- gamlss2(Ozone ~ s(Wind, sp = 1) | 1 | 1 | 1,
  data = air, family = BCT,
  start = c(mu = 200, sigma = 0.1, nu = 1, tau = 10),
  fixed = c(nu = TRUE, tau = TRUE), maxit = 100)

## fixed flags must also be supplied when reusing coefficients
m6 <- gamlss2(Ozone ~ s(Wind, sp = 1) | 1 | 1 | 1,
  data = air, family = BCT,
  start = coef(m5, full = TRUE, dropall = FALSE),
  fixed = c(nu = TRUE, tau = TRUE), maxit = 100)
unique(fitted(m6, parameter = c("nu", "tau")))

## End(Not run)
```

gnet

*Model Terms with glmnet***Description**

Constructor for estimating penalized model terms using the **glmnet** package.

Usage

```
gnet(formula, ...)
```

Arguments

formula	A formula specifying the covariates for the penalized term.
...	criterion selects the information criterion: "bic" (default), "aic", "aicc", "gaic", or "gcv". Other arguments are passed to glmnet , including alpha (elastic-net mixing parameter), lambda (penalty sequence), nlambdas (number of penalties), and standardize (whether to standardize covariates).

Details

`gnet()` constructs a [model.matrix](#) from the formula, removes its intercept column, and fits the term using [glmnet](#) within backfitting. The penalty lambda minimizes the selected criterion. For `criterion = "gaic"`, the multiplier K is set in [gamlss2_control](#) (default: 2).

Value

A special model-term object used internally by [gamlss2](#).

See Also

[la](#), [gamlss2](#), [specials](#).

Examples

```
## Not run: ## transform to categorical
air <- subset(airquality, !is.na(Ozone))

air$Tc <- cut(air$Temp, breaks = seq(50, 100, by = 10),
  include.lowest = TRUE)
air$Wc <- cut(air$Wind, breaks = seq(2, 22, by = 4),
  include.lowest = TRUE)

## model formula, for each parameter we use
## the same gnet() model term
f <- Ozone ~ gnet(~Tc + Wc) | .

## estimate model
m <- gamlss2(f, data = air, family = NO)

## summary with edf
summary(m)

## plot coefficient paths
sp <- specials(m, parameter = "mu", terms = "gnet(~Tc+Wc)", elements = "model")
print(class(sp))
plot(sp)

## End(Not run)
```

HarzTraffic

Traffic Counts at Sonnenberg in the Harz Region

Description

This dataset contains daily traffic counts near Sonnenberg in the Harz region of Germany. It covers nearly three years, from 2021-01-01 to 2023-11-30, and combines traffic counts with weather covariates. The data are useful for illustrating seasonal count regression models, for example with cyclic smoothers over the day of the year.

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 extracted from date. This is convenient for modeling within-year seasonality.

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 $^{\circ}C$.
tempmax Numeric, maximum temperature in $^{\circ}C$.
temp Numeric, mean temperature in $^{\circ}C$.
humidity Numeric, mean relative humidity in percent.
tempdew Numeric, average dewpoint temperature in $^{\circ}C$.
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.

Details

The traffic variables record daily counts by vehicle type. The weather variables summarize daily conditions measured at a nearby station. In package examples, bikes is often modeled as a seasonal response because motorcycle traffic shows a pronounced annual cycle and substantial variation across weather conditions.

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>

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)

## corresponding quantiles
p <- quantile(m, newdata = nd, probs = c(0.05, 0.5, 0.95))

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

im

*Convolutional Image Model Terms for GAMLSS***Description**

Constructs a convolutional neural-network image model term for use with `gamlss2`. The term learns convolutional filters, applies a ReLU activation and spatial-pyramid pooling, and fits a penalized linear output layer to the current working response.

Usage

```
im(x, dim = NULL, ...)
```

Arguments

<code>x</code>	A single image variable. Images can be supplied as an n by p matrix of flattened images, an n by height by width (optionally by channel) array, or a list of equally sized image matrices or arrays.
<code>dim</code>	Optional integer vector <code>c(height, width)</code> for grayscale images or <code>c(height, width, channels)</code> for multi-channel images. It is required for flattened matrix input and can otherwise be inferred from the data.
<code>...</code>	Additional named CNN controls. Unknown controls are rejected: <code>filters</code> Integer number of convolutional filters. The default is 4. <code>kernel</code> One integer or two integers giving the convolutional kernel height and width. The default is <code>c(3, 3)</code> . <code>pool</code> Positive integer spatial-pyramid pooling levels. The default is <code>c(1, 2, 4)</code> , truncated to the image dimensions.

epochs Number of optimization epochs for the first backfitting update. The default is 8.

warm_epochs Number of optimization epochs for subsequent updates. The default is 3.

learning_rate Positive Adam learning rate. The default is 0.003.

decay Non-negative ridge penalty for the convolutional and output parameters. The default is 0.001.

beta1, beta2 Adam exponential-decay parameters in $[0, 1)$. Their defaults are 0.9 and 0.999.

eps Positive numerical stabilization constant for Adam. The default is $1e-8$.

clip Optional positive bound for gradient clipping. The default is 5.

scale Logical; should each image channel be centered and scaled using the training images? The default is TRUE.

seed Optional finite numeric seed used when initializing the filters.

trace Logical; should the CNN optimization progress be printed? The default is FALSE.

Details

The image term is intended for images stored as one observation per row. For a flattened matrix, use `dim` to specify the image dimensions; for an array or image list, the dimensions are checked and retained. The same image representation and dimensions must be supplied when predicting on new data.

The model applies a same-padded convolution, a rectified-linear activation, and spatial-pyramid pooling. The pooled features are mapped to the current working response by a penalized linear output layer. Convolutional parameters are initialized on the first update and warm-started on later backfitting updates. The fitted contribution is centered before it is added to the distributional predictor.

This term is a CNN-based alternative to [si](#), which represents images through a penalized tensor-product coefficient surface. The fitted model can be inspected with [plot.gamlss2](#) and used for new images with [predict.gamlss2](#).

Value

An object of classes "special" and "im". During fitting, the corresponding `special_fit.im()` method returns the fitted CNN, centered fitted values, transfer information for warm starts, and an effective degrees-of-freedom estimate.

See Also

[si](#), [special_terms](#), [gamlss2](#)

Examples

```
## Not run:
set.seed(1328)
n <- 120
nr <- 16
```

```

nc <- 16

## images with a three-by-three pattern at random locations
images <- array(rnorm(n * nr * nc, sd = 0.25), c(n, nr, nc))
locations <- data.frame(
  row = sample(2:(nr - 2), n, replace = TRUE),
  col = sample(2:(nc - 2), n, replace = TRUE)
)
amplitude <- runif(n, 0.5, 1.5)
for(i in seq_len(n)) {
  images[i, locations$row[i]:(locations$row[i] + 2),
    locations$col[i]:(locations$col[i] + 2)] <-
    images[i, locations$row[i]:(locations$row[i] + 2),
      locations$col[i]:(locations$col[i] + 2)] + amplitude[i]
}

## known spatial response surface
truth <- outer(seq_len(nr), seq_len(nc), function(row, col)
  exp(-((row - 5) / 3)^2 - ((col - 11) / 4)^2))
truth <- truth / max(truth)
d <- data.frame(y = vapply(seq_len(n), function(i) {
  amplitude[i] * mean(truth[locations$row[i]:(locations$row[i] + 2),
    locations$col[i]:(locations$col[i] + 2)])
}, 0) + rnorm(n, sd = 0.2))
image(images[1, , ], col = hcl.colors(20, "Inferno"),
  main = "Example image")
d$image <- I(matrix(images, nrow = n))

m <- gamlss2(
  y ~ im(image, dim = c(nr, nc), filters = 3, epochs = 15,
    warm_epochs = 2, seed = 1904) | 1,
  data = d,
  family = NO
)

plot(m)
## predicted distributions
predict(m, newdata = d[1:5, , drop = FALSE])

## predicted response map for a moving image pattern
map.grid <- expand.grid(row = 2:(nr - 2), col = 2:(nc - 2))
map.images <- array(0, c(nrow(map.grid), nr, nc))
for(i in seq_len(nrow(map.grid))) {
  map.images[i, map.grid$row[i]:(map.grid$row[i] + 2),
    map.grid$col[i]:(map.grid$col[i] + 2)] <- 1
}
map.data <- data.frame(y = rep(0, nrow(map.grid)))
map.data$image <- I(matrix(map.images, nrow = nrow(map.grid)))
map.pred <- predict(m, parameter = "mu", newdata = map.data)
true.map <- vapply(seq_len(nrow(map.grid)), function(i) {
  sum(truth[map.grid$row[i]:(map.grid$row[i] + 2),
    map.grid$col[i]:(map.grid$col[i] + 2)]) / 9
}, numeric(1))

```

```

true.map <- matrix(true.map,
  nrow = length(unique(map.grid$row)))
pred.map <- matrix(map.pred,
  nrow = length(unique(map.grid$row)))

op <- par(mfrow = c(1, 2))
image(unique(map.grid$row), unique(map.grid$col), true.map,
  col = hcl.colors(20, "Inferno"),
  xlab = "image row", ylab = "image column",
  main = "True response map")
image(unique(map.grid$row), unique(map.grid$col), pred.map,
  col = hcl.colors(20, "Inferno"),
  zlim = range(true.map, pred.map),
  xlab = "image row", ylab = "image column",
  main = "Predicted response map")
par(op)

## End(Not run)

```

JR

*Joint REML Estimation***Description**

Fit a GAMLSS model by joint Laplace approximate REML.

Usage

```

## optimizer function
JR(x, y, specials, family, offsets,
  weights, start, xterms, sterms, control)

## refit function
jr(object, trace = TRUE, ...)

```

Arguments

<code>x, y, specials, family, offsets, weights, start, xterms, sterms, control</code>	Model components passed by <code>gamlss2()</code> .
<code>object</code>	A fitted <code>gamlss2</code> model retaining its design matrices.
<code>trace</code>	Print the final joint REML result.
<code>...</code>	Named joint REML control overrides.

Details

Use `optimizer = JR` to fit a model or `jr(fit)` to continue from an existing fit. The direct solver jointly estimates coefficients and smoothing parameters from cached designs and quadratic penalties. It requires a family density, a predictor-to-parameter map, and an identifiable finite joint mode.

Selected penalties and fixed distributional parameters are unsupported. Duplicate unpenalized directions in overlapping smooths are removed from the joint solve. The stored coefficients and covariance use the minimum-norm representation in the original coefficient space, and effective degrees of freedom exclude these redundant directions. If joint REML fails, the input fit is retained and `reml.failure` records the reason. Check `isTRUE(fit$jr$converged)` before using joint uncertainty estimates. Power-exponential families may have a non-smooth REML profile when a fitted location coincides with an observation. Such fits retain RS/CG and report that smoothing-parameter curvature could not be verified. Fully penalized smooths that start nearly suppressed are also checked at less penalized starting values, to avoid stopping in a flat profile tail. If the outer optimizer reports false convergence, a stricter independent profile Newton check may verify the fit. In that case `fit$jr$outter.verified` is true; the original optimizer status is retained in `fit$jr$outter`. Iteration and evaluation limits still retain the initial fit.

The following controls can be supplied through `gamlss2_control` or as named arguments to `jr()`:

- `jr.inner`: Function, [RS](#) or [CG](#), used to obtain the initial fit. Default is RS.
- `jr.outer.maxit`: Integer, the maximum number of outer smoothing parameter iterations. Default is 60.
- `jr.outer.maxeval`: Integer, the maximum number of outer objective evaluations. Default is 150.
- `jr.eps`: Positive number, the relative coefficient step tolerance in the joint mode solve. Default is 1e-8. The solver may refine further to stabilize power-exponential curvature.
- `jr.maxit`: Positive integer, the maximum number of coefficient iterations per objective evaluation. Default is 100.

The initial RS/CG fit uses `eps` and `maxit`, as documented in [RS](#). Fully and partially fixed smoothing parameters and unpenalized `fx = TRUE` smooths are supported. At extreme smoothing parameters, penalty roots and an equilibrated solve preserve accuracy in the unpenalized coefficient directions.

For local REML updates with RS/CG and other smoothing parameter selection methods, see [gamlss2_smoothing](#).

Value

A fitted `gamlss2` model with joint REML results in `fit$jr`.

See Also

[RS](#), [gamlss2_control](#), [gamlss2_smoothing](#), [vcov.gamlss2](#)

Examples

```
## Not run:
m <- gamlss2(dist ~ s(speed), data = cars, family = NO)
m <- jr(m)
vcov(m, unconditional = TRUE)

## End(Not run)
```

Description

The two-parameter Kumaraswamy family for continuous responses strictly between 0 and 1, together with a shifted logarithmic link specification.

Usage

```
## the Kumaraswamy family
Kumaraswamy(a.link = shiftlog, b.link = shiftlog, ...)
KS(a.link = shiftlog, b.link = shiftlog, ...)

## the log(mu - shift) link specification
shiftlog(shift = 1)
```

Arguments

<code>a.link</code>	A character string, link object, or function specifying the link for shape parameter a .
<code>b.link</code>	A character string, link object, or function specifying the link for shape parameter b .
<code>shift</code>	A single finite number specifying the lower bound imposed by the shifted logarithmic link.
<code>...</code>	Currently not used.

Details

The Kumaraswamy distribution is continuous on $(0, 1)$ with strictly positive shape parameters a and b . Its probability density function is

$$f(y; a, b) = aby^{a-1}(1 - y^a)^{b-1},$$

and its cumulative distribution function is

$$F(y; a, b) = 1 - (1 - y^a)^b.$$

Consequently, the quantile function is

$$F^{-1}(p; a, b) = \left(1 - (1 - p)^{1/b}\right)^{1/a}.$$

For a parameter μ , `shiftlog(shift = s)` defines the link and inverse link

$$g(\mu) = \log(\mu - s), \quad g^{-1}(\eta) = \exp(\eta) + s,$$

with domain $\mu > s$. The default `shift = 1` therefore restricts both shape parameters to be greater than 1. In that case the distribution has the unique interior mode

$$\left(\frac{a-1}{ab-1} \right)^{1/a}.$$

When alternative links allow $a \leq 1$ or $b \leq 1$, the family returns NA for the interior mode.

Value

`Kumaraswamy()` and `KS()` return the same "gamlss2.family" object. It provides probability density, cumulative distribution, quantile, and random-generation functions, as well as the mean, variance, and interior mode. Its random generator follows the family convention `random(par, n)`.

`shiftlog()` returns a link specification of class "link-glm".

References

Kumaraswamy P (1980). "A Generalized Probability Density Function for Double-Bounded Random Processes". *Journal of Hydrology*, **46**(1–2), 79–88. doi:10.1016/00221694(80)900360

See Also

[gamlss2.family](#), [gamlss2](#)

Examples

```
## create family object with different link specifications
fam <- Kumaraswamy(a.link = shiftlog, b.link = "log")

## simulate data
set.seed(1328)
n <- 1000
d <- data.frame("x" = sort(runif(n, -pi, pi)))

## true parameters
par <- data.frame(
  "a" = exp(1.2 + sin(d$x)) + 1,
  "b" = 1
)

## sample one response for every parameter row
d$y <- fam$random(par, 1)

## estimate model using the Kumaraswamy family
m <- gamlss2(y ~ s(x), data = d, family = fam)

## plot estimated effect
plot(m)

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

```
## predict quantiles
p <- quantile(m, probs = seq(0.05, 0.95, by = 0.05))

## plot
plot(d, pch = 19, col = adjustcolor(1, 0.3))
matlines(d$x, p, lty = 1, col = 4, lwd = 2)
```

la	<i>Lasso Model Terms</i>
----	--------------------------

Description

Constructor function and plotting for Lasso penalized model terms for GAMLSS.

Usage

```
## model term constructor function
la(x, type = 1, const = 1e-05, ...)

## plotting function
plot_lasso(x, terms = NULL,
  which = c("criterion", "coefficients"),
  zoom = c(3, 4), spar = TRUE, ...)
```

Arguments

x	For function <code>la()</code> , a numeric vector or matrix, or a formula. See the examples. For function <code>plot_lasso()</code> , an object returned from gamlss2 .
type	Integer or character, the type of the Lasso penalty. <code>type = 1</code> or <code>type = "normal"</code> uses the normal penalty, <code>type = 2</code> or <code>type = "group"</code> the group penalty, <code>type = 3</code> or <code>type = "ordinal"</code> the ordinal fusion penalty and <code>type = 4</code> or <code>type = "nominal"</code> the nominal fusion penalty.
const	Numeric, the constant that is used for approximating the absolute function.
terms	Character or integer, the model term that should be plotted. The default <code>terms = NULL</code> is plotting all model terms.
which	Character, should the information criterion or the coefficient paths be plotted? See the examples.
zoom	Numeric vector of length 2, the zooming factors for plotting information criteria curves and coefficient paths. The first element sets the distance from the optimum shrinkage parameter λ to the left side, and the second element to the right side, respectively.
spar	Logical, should plotting parameters be automatically set in par ?
...	For function <code>la()</code> further control arguments can be passed: The <code>criterion = "bic"</code> for shrinkage parameter selection, arguments for creating the model.matrix

if the model term is specified using a formula, and `scale = TRUE/FALSE` to control the internal scaling of the design matrix. For function `plot_lasso()` arguments like `lwd`, `col`, `main`, etc., that control plotting parameters can be supplied. An additional ridge penalty (elastic net) can be added to each `la()` term by setting `add_ridge = TRUE` in the `gamlss2` call.

Details

To implement the Lasso penalty, an approximation of the absolute value function is used, following the approach by Oelker and Tutz (2017). This enables the use of standard Newton-Raphson-type algorithms for estimation. Each Lasso model term has its own shrinkage parameter, allowing a mix of different penalty types within the model. The framework builds on the methodology of Groll et al. (2019), where coefficients are updated through iteratively reweighted least squares (IWLS). This is feasible due to the absolute function approximation, which results in a quadratic penalty matrix similar to that used in penalized splines. By default, the shrinkage parameters are selected using the Bayesian Information Criterion (BIC).

The choice of type determines the structure that is penalized:

- "normal" penalizes individual coefficients and is suitable for numeric covariates or general design matrices.
- "group" penalizes one complete block of coefficients jointly and is mainly useful for factor effects or other grouped design matrices.
- "ordinal" uses an adjacent-differences fusion penalty and is intended for ordered factor effects or matrices whose columns represent ordered levels.
- "nominal" uses a pairwise-differences fusion penalty and is intended for unordered factor effects or matrices whose columns represent unordered levels.

For numerical stability and comparable shrinkage across terms, `la()` can scale the underlying design matrix internally. The exact scaling depends on the selected penalty type. For "ordinal" and "nominal" terms, an additional ridge component is included by default.

Value

The `la()` function is used internally within `gamlss2` and provides the necessary details for estimating Lasso-type model terms. Essentially, it serves as a special model term, as outlined in `specials`. Currently, the `plot_lasso()` function does not return any output.

References

- Groll A, Hambuckers J, Kneib T, Umlauf N (2019). "LASSO-Type Penalization in the Framework of Generalized Additive Models for Location, Scale and Shape". *Computational Statistics & Data Analysis*, **140**, 59–73. doi:10.1016/j.csda.2019.06.005
- Oelker MR, Tutz G (2017). "A Uniform Framework for the Combination of Penalties in Generalized Structured Models". *Advances in Data Analysis and Classification*, **11**(1), 97–120. doi:10.1007/s116340150205y

See Also

`gamlss2`, `specials`.

Examples

```
## Not run: ## transform numeric to factor variables
air <- subset(airquality, !is.na(Ozone))

air$Tc <- cut(air$Temp, breaks = seq(50, 100, by = 10),
  include.lowest = TRUE)
air$Wc <- cut(air$Wind, breaks = seq(2, 22, by = 4),
  include.lowest = TRUE)

## set up the model formula for a BCT model
f <- Ozone ~ la(Tc,type=3) + la(Wc,type=3) | . | . | .

## estimation
m1 <- gamlss2(f, data = air, family = BCT)

## summary, shows the estimated degrees of freedom
## for each model term
summary(m1)

## plot estimated coefficients
plot(m1)

## plot information criteria curves
## for each model term
plot_lasso(m1)

## plot parameter paths
plot_lasso(m1, which = "coefficients")

## plot a single model term
plot_lasso(m1, which = "coefficients", term = 2)

## same with
plot_lasso(m1, which = "coefficients", term = "mu.la(Wc)")

## zoom out
plot_lasso(m1, which = "coefficients", term = 2,
  zoom = c(8, 7))

## set names
plot_lasso(m1, which = "coefficients", term = 2,
  zoom = c(8, 7), names = c("A", "B", "C", "D"))

## set title
plot_lasso(m1, which = "coefficients", term = 2,
  zoom = c(8, 7), main = "Fused Lasso")

## simulated example using the nominal fused lasso
## and a matrix as argument for la()
set.seed(1328)

## number of observations and covariates
```

```

n <- 500
k <- 50

## model matrix
X <- matrix(rnorm(n * k), n, k)
colnames(X) <- paste0("x", 1:k)

## true coefficients
beta <- list(
  "mu" = rbinom(k, 1, 0.1),
  "sigma" = rbinom(k, 1, 0.1) * 0.3
)

## parameters
mu <- X %>% beta$mu
sigma <- exp(-1 + X %>% beta$sigma)

## response
y <- rnorm(n, mean = mu, sd = sigma)

## model formula with nominal fused lasso
f <- y ~ la(X,type=4) | la(X,type=4)

## estimate model incl. extra ridge penalty
## for all la() model terms
m2 <- gamlss2(f, add_ridge = TRUE)

## plot information criteria curves
plot_lasso(m2)

## coefficient paths
plot_lasso(m2, which = "coefficients")

## zoom out
plot_lasso(m2, which = "coefficients",
  zoom = c(8, 9))

## extract coefficients
cb <- coef(m2, full = TRUE)

## compare (without intercept)
cb_mu <- cb[grep("mu.", names(cb))][,-1]
cb_sigma <- cb[grep("sigma.", names(cb))][,-1]

## true positive rate
tp <- mean(c(cb_mu[beta$mu > 0] > 0,
  cb_sigma[beta$sigma > 0] > 0))

## false positive rate, needs threshold
thres <- 0.01
fp <- mean(c(abs(cb_mu[beta$mu == 0]) > thres,
  abs(cb_sigma[beta$sigma == 0]) > thres))

```

```
## End(Not run)
```

lin

Bundled Linear and Ridge-Penalized Model Terms

Description

Constructs a model-matrix block for use as a single model term in a `gamlss2` formula. Function `lin()` fits the block with only a negligible numerical penalty, whereas `ridge()` standardizes its columns and estimates a ridge penalty for the complete block.

Usage

```
lin(x, ..., ridge = FALSE, scale = FALSE)

ridge(...)
```

Arguments

<code>x</code>	A one-sided formula or the first unquoted predictor expression. A formula is recommended when transformations, factors, or interactions are used, for example <code>~ x1 + x2</code> , <code>~ x1:x2</code> , or <code>~ 0 + f</code> .
<code>...</code>	In <code>lin()</code> , additional unquoted predictor expressions that are combined additively with <code>x</code> when <code>x</code> is not a formula. In <code>ridge()</code> , all arguments are forwarded to <code>lin()</code> ; consequently, calls such as <code>ridge(x1, x2)</code> and <code>ridge(~ x1 + x2)</code> are both supported. For precise control of formula expansion, a one-sided formula is recommended.
<code>ridge</code>	Logical. Should the identity-penalty smoothing parameter be estimated? This is primarily a low-level argument; <code>ridge()</code> is the convenient interface for <code>ridge = TRUE</code> with scaling enabled.
<code>scale</code>	Logical. Should the columns of the constructed design matrix be centered and divided by their sample standard deviations? Columns with zero or non-finite standard deviation are left unchanged. Function <code>ridge()</code> always uses <code>scale = TRUE</code> .

Details

The supplied predictors are converted to a design matrix using `model.matrix`. Any column named "(Intercept)" is removed because the distributional predictor has its own intercept. Apart from that removal, standard R formula rules and contrasts are used. Thus, `~ x1 * x2` expands to two main effects and their interaction, while `~ x1:x2` produces a pure interaction without adding the marginal terms. Pure interactions are therefore supported. For a factor `f`, `~ f` uses the current contrast coding (normally treatment coding), whereas `~ 0 + f` requests one indicator column per level.

Supplying several predictors without a formula is a convenience shortcut: `lin(x1, x2)` is equivalent to `lin(~ x1 + x2)`, and similarly for `ridge()`. A formula should be used whenever operators such as `:`, `*`, `-`, or `0 +` need to be interpreted explicitly.

Function `lin()` fixes the identity-penalty multiplier at a negligible value. It therefore behaves like weighted linear regression for the current working response, while entering all columns as one model term rather than as separate ordinary linear terms. This can be useful for keeping a block of columns together and for using the same model-matrix and prediction machinery as penalized terms. Ordinary linear effects can still be specified directly in the surrounding `gamlss2` formula.

When every distributional predictor using the block has an intercept, nonconstant columns are centered before fitting, without changing their scale. This avoids competition between the block and the separate intercept during backfitting. The intercept then refers to the centered block, and the stored column means are reused for prediction. If a predictor using the block has no intercept, the unscaled design is retained.

For `ridge()`, coefficients are estimated from the current weighted working response using an identity penalty. Ignoring the small numerical stabilization used by the fitting algorithm, the update minimizes

$$(z - X\beta)^T W (z - X\beta) + \lambda \beta^T \beta,$$

where z and W are the current working response and weights. A large λ shrinks the whole coefficient block toward zero.

Before fitting, every nonconstant column of X is standardized using its training-data mean and standard deviation. The stored transformation is reused for prediction. Consequently, the identity penalty is applied on a comparable scale across numeric predictors, indicator columns, and interaction columns.

The smoothing parameter λ is selected during backfitting using the smooth-term criterion supplied to `gamlss2()`. The default is a local REML update (`criterion = "reml"`); supported alternatives include `"gcv"`, `"aic"`, `"gaic"`, `"aicc"`, and `"bic"`. For example, use `gamlss2(..., criterion = "bic")` to select the ridge penalty by BIC. The previous estimate is used to warm-start subsequent backfitting updates.

The effective degrees of freedom are computed from the penalized weighted hat matrix.

Value

`lin()` and `ridge()` return a smooth specification object of class `"lin.smooth.spec"`, which is completed to an object of class `"lin.effect"` during model setup.

See Also

[gamlss2](#), [specials](#), [model.matrix](#), [smooth.construct](#)

Examples

```
## Not run:
## longley data example with
## strongly correlated predictors
data("longley")

## fit all predictors linearly
m1 <- gamlss2(
  Employed ~ lin(~ GNP.deflator + GNP + Unemployed +
```

```

    Armed.Forces + Population + Year),
  data = longley,
  family = NO
)

## fit the same model using a ridge-penalized block,
## ridge() standardizes the model-matrix columns before applying the
## common ridge penalty.
m2 <- gamlss2(
  Employed ~ ridge(~ GNP.deflator + GNP + Unemployed +
    Armed.Forces + Population + Year),
  data = longley,
  family = NO
)

## Compare model summaries.
summary(m1)
summary(m2)

## extract coefficients
cm <- cbind(
  "linear" = specials(m1, parameter = "mu", elements = "coefficients"),
  "ridge" = specials(m2, parameter = "mu", elements = "coefficients")
)
print(cm)

## ridge terms combined with ordinary linear effects,
## Year is left unpenalized, while the remaining correlated
## economic variables are shrunk jointly
m3 <- gamlss2(
  Employed ~ Year +
    ridge(~ GNP.deflator + GNP + Unemployed +
      Armed.Forces + Population),
  data = longley,
  family = NO, maxit = 100
)

summary(m3)

## interactions
m4 <- gamlss2(
  Employed ~ ridge(~ GNP + Unemployed + GNP:Unemployed),
  data = longley,
  family = NO
)

specials(m4, parameter = "mu", elements = "coefficients")

## End(Not run)

```

Description

Constructs a local polynomial regression fitting (LOESS) model term for use with `gamlss2`. The neighborhood size and polynomial degree can be supplied by the user. Neighborhood parameter tuning can be performed automatically using various information criteria during backfitting.

Usage

```
lo(formula, ...)
```

Arguments

<code>formula</code>	A formula specifying the predictors to be included in the local polynomial regression (e.g., <code>~ x1</code> or <code>~ x1 + x2</code>).
<code>...</code>	Further arguments passed directly to the underlying <code>loess</code> function or its control structures. Typical arguments include <code>span</code> (the smoothing parameter controlling neighborhood size), <code>degree</code> (the degree of the local polynomials, usually 1 or 2), and <code>parametric</code> or <code>drop.square</code> .

Details

The term is fitted using `loess` within backfitting. Effective degrees of freedom (edf) for information criteria are taken from the fitted object's `trace.hat`. The fitted smooth is centered by subtracting its mean to ensure identifiability.

Value

The constructor `lo()` returns an object of classes `"special"` and `"lo"` containing the covariate matrix, formula definitions, and control arguments required for fitting and prediction.

After estimation with `gamlss2`, the corresponding fitted special term contains the following main components:

- `model`: the underlying fitted `loess` object.
- `fitted.values`: centered fitted contribution of the term.
- `edf`: effective degrees of freedom, taken from `trace.hat`.
- `shift`: the mean value used to center the fitted smooth.
- `term`: character vector of variable names involved in the smooth.
- `formula`: the internal updated formula matching the working response.

References

Cleveland WS, Grosse E, Shyu WM (1992). "Local Regression Models". In Chambers JM, Hastie TJ (eds.), *Statistical Models in S*, chapter 8, 309–376. Wadsworth & Brooks/Cole, Pacific Grove, CA. doi:[10.1201/97802037385358](https://doi.org/10.1201/97802037385358)

See Also

[gamlss2](#), [special_terms](#), [loess](#)

Examples

```
## Not run: ## air quality data
air <- subset(airquality, !is.na(Ozone))
air <- air[order(air$Temp), ]

## Fit a distribution where both location (mu) and scale (sigma)
## depend nonlinearly on Temp via local regression
f <- Ozone ~ lo(~Temp+Wind, span=0.5, degree=2) | lo(Temp, span=0.7)

## estimate model
m <- gamlss2(f, data = air, family = GA)

## model summary
summary(m)

## visualize estimated effects
plot(m)

## End(Not run)
```

logLik.gamlss2

*Log-Likelihood of GAMLSS Models***Description**

Extracts the fitted log-likelihood from gamlss2 objects. Optionally, the log-likelihood can be evaluated on new data. Several fitted models can be supplied for comparison.

Usage

```
## S3 method for class 'gamlss2'
logLik(object, ..., newdata = NULL)
```

Arguments

object	An object of class "gamlss2".
...	Optionally, further fitted gamlss2 objects.
newdata	An optional data frame in which to evaluate the log-likelihood. If omitted, the log-likelihood stored in the fitted model object is returned. If supplied, newdata must contain the response variable.

Details

For the original model data, logLik() returns the log-likelihood value stored in the fitted gamlss2 object. If newdata is supplied, the distribution parameters are first predicted for the observations in newdata; the response is then extracted from newdata and the log-likelihood is recomputed using the model family.

The returned object follows the usual `logLik` convention: it has class `"logLik"` and carries attributes `"df"` and `"nobs"`. The degrees of freedom are taken from the fitted model object, while the number of observations is either the number of observations used for fitting or the number of rows evaluated in `newdata`.

If more than one fitted model is supplied, a named list of `"logLik"` objects is returned. The list is ordered by decreasing log-likelihood.

Value

For a single model, an object of class `"logLik"`. For several supplied models, a named list of `"logLik"` objects, ordered by decreasing log-likelihood.

See Also

`logLik`, `deviance.gamlss2`, `gamlss2`

Examples

```
## fit two normal GAMLSS models
## stopping distance (ft) explained by speed (mph)
data("cars", package = "datasets")

m1 <- gamlss2(dist ~ speed | 1, data = cars, family = NO)
m2 <- gamlss2(dist ~ speed | speed, data = cars, family = NO)

## fitted log-likelihood
logLik(m1)

## attributes follow the usual logLik convention
ll <- logLik(m1)
attr(ll, "df")
attr(ll, "nobs")

## compare several fitted models
logLik(m1, m2)

## log-likelihood evaluated on new data
nd <- cars[1:10, ]
logLik(m2, newdata = nd)

## relation to the deviance
-2 * as.numeric(logLik(m2))
deviance(m2)
```

Description

This function is used with the family functions in `gamlss2()`. Given the name of a link, it returns a link function, an inverse link function, the derivative $d\mu/d\eta$, and a function for domain checking. Note that `make.link2()` is slightly more flexible and also allows functions as arguments.

Usage

```
make.link2(link)
```

Arguments

`link` A character string, see function `make.link`, or function.

Value

A list of class "link-gamlss2" containing the following components:

<code>linkfun</code>	Link function <code>function(mu)</code> .
<code>linkinv</code>	Inverse link function <code>function(eta)</code> .
<code>mu.eta</code>	Derivative function(eta): $d\mu/d\eta$.
<code>valideta</code>	Function <code>function(eta)</code> that returns TRUE if eta is in the domain of <code>linkinv</code> .
<code>name</code>	A character string representing the name of the link function.

See Also

`make.link`, `gamlss2`, `gamlss2.family`.

Examples

```
## character specification
utils::str(make.link2("logit"))

## functions
utils::str(make.link2(softplus))
```

marginal_predict

Marginal Prediction Profiles for gamlss2 Models

Description

Compute one-variable-at-a-time prediction profiles from a fitted **gamlss2** model. Each selected covariate is varied over a grid while the other model covariates are held at representative values.

Usage

```
marginal_predict(object, newdata = NULL, variables = NULL,
  n = 100L, continuous = median, at = NULL, values = NULL, ...)
```

Arguments

object	A fitted object inheriting from class "gamlss2" or "bamlss2".
newdata	An optional data frame defining the observed ranges, levels, and representative values of the model covariates. By default, the model frame of object is used.
variables	A character vector naming the covariates for which profiles should be computed. By default, all covariates used by the model are included.
n	A single integer greater than one giving the grid size for numeric, Date, and POSIXct covariates. A covariate that is constant has a one-point grid.
continuous	A function, or the name of a function, used to compute the representative value of covariates not currently being varied. The default is median; mean is another common choice. The function must return one value.
at	An optional named list specifying values at which covariates should be held fixed. Each element must contain one value.
values	An optional named list specifying the prediction grid for one or more of the selected variables.
...	Further arguments passed to <code>predict.gamlss2</code> , such as <code>type</code> , <code>parameter</code> , <code>terms</code> , or <code>se.fit</code> .

Details

For each selected covariate, numeric and date-time covariates are varied over an equally spaced grid spanning their observed range. Factors are varied over their levels, and character and logical covariates over their observed values. The `values` argument can be used to replace any of these grids.

Covariates not being varied are fixed at `at`, when supplied. Otherwise, numeric and date-time covariates are summarized by `continuous`, factors are set to their first level, character variables to their first observed value, and logical variables to `FALSE` when that value is observed.

Here, *marginal prediction* means a conditional one-variable profile: predictions are evaluated at one representative combination of the remaining covariates. They are not averages over the empirical joint distribution of those covariates. In models with interactions, the profile is conditional on the interacting covariates' fixed values; use `at` to choose scientifically meaningful values.

Value

A named list with one data frame per selected covariate. Each data frame contains the focal covariate followed by the output from `predict(object, newdata = ..., ...)`. Vector predictions, such as those from `type = "response"`, are stored in a column named `.fit`. The representative values computed for the covariates are available in the `"fixed"` attribute of the returned list. Each is used whenever that covariate is not the focal variable.

See Also

`predict.gamlss2`, `prodist.gamlss2`, `quantile.gamlss2`

Examples

```
## Same air-quality model as in the first gamlss2() example.
air <- subset(airquality, !is.na(Ozone))
m <- gamlss2(
  Ozone ~ s(Temp) + s(Wind) | s(Temp),
  data = air,
  family = GA
)

## Conditional distributions for temperature and wind.
mp <- marginal_predict(
  m,
  variables = c("Temp", "Wind"),
  n = 50,
  type = "distribution"
)
head(mp$Temp)
attr(mp, "fixed")

## plot quantiles
mp$Temp$qfit <- quantile(mp$Temp$fit, probs = seq(0.01, 0.99, by = 0.01))

matplot(mp$Temp$Temp, mp$Temp$qfit, type = "l", lty = 1, col = 1,
  xlab = "Temperature (degrees F)", ylab = "Fitted ozone quantiles")

## Supply a custom grid and condition on a particular wind speed.
mp2 <- marginal_predict(
  m,
  variables = "Temp",
  values = list(Temp = seq(60, 95, by = 5)),
  at = list(Wind = 5),
  type = "response"
)
mp2$Temp
```

mcmc

Run MCMC Sampling for a Fitted gamlss2 Model

Description

The function `mcmc()` runs MCMC sampling for a fitted [gamlss2](#) model. It reuses the fitted model object and merges the MCMC output back into the original object so that standard methods such as [summary](#) and [plot](#) work as expected.

Usage

```
mcmc(object, n.iter = 1200, burnin = 200, thin = 1,
  trace = interactive())
```

Arguments

<code>object</code>	An object of class "gamlss2" or "bamlss2" as returned by gamlss2 .
<code>n.iter</code>	Integer, the total number of MCMC iterations.
<code>burnin</code>	Integer, the burn-in period.
<code>thin</code>	Integer, the thinning parameter for saved samples.
<code>trace</code>	Logical. Should progress information be printed while sampling?

Details

The function sets `n.iter`, `burnin`, `thin`, and `trace` in the model control list and obtains starting values from the fitted object, including smoothing parameters. The MCMC sampler [BS](#) is then called and the resulting samples and posterior summaries are merged into the original model object.

This approach ensures that the result retains the `terms`, `call`, and related components required by extractor and plotting methods. If the fitted model object does not contain the design information required for MCMC sampling, for example because a reduced control setting was used, sampling cannot be performed.

Applying `mcmc()` to an object that already contains posterior samples continues from the terminal state of the preceding chain. Stored posterior summaries and DIC are then recomputed from all retained samples.

Value

A model object inheriting from "gamlss2" containing posterior samples and summary information. The object also contains:

- `samples`: posterior draws for coefficients and smoothing variances;
- `logLik.samples`: log-likelihood values for the retained draws;
- `results`: model results derived from posterior means;
- `elapsed`: elapsed sampling time.

References

Umlauf N, Klein N, Zeileis A (2018). "BAMLSS: Bayesian Additive Models for Location, Scale, and Shape (and Beyond)". *Journal of Computational and Graphical Statistics*, **27**(3), 612–627. [doi:10.1080/10618600.2017.1407325](https://doi.org/10.1080/10618600.2017.1407325)

See Also

[gamlss2](#), [bamlss2](#), [BS](#)

Examples

```
## Not run: ## specify the model formula
f <- Ozone ~ s(Temp) | . | 1

## estimate model using backfitting
m1 <- gamlss2(f, data = airquality, family = GIG)
```

```
## run MCMC sampling
set.seed(1328)
m2 <- mcmc(m1)

## generate more samples
m2 <- mcmc(m2)

## posterior summary
summary(m2)

## plot estimated effects
plot(m2)

## plot samples
plot(m2, which = "samples")
plot(m2, which = "samples", ask = TRUE)
plot(m2, which = "samples", ask = TRUE,
     parameter = "mu", lag.max = 120, term = "s(Temp)")

## samples can be extracted for further inspection
print(head(m2$samples))

## compare estimated mean effects
plot(c(m1, m2))

## predict parameters using samples
pm <- predict(m2, type = "parameter", FUN = mean)
print(head(pm))
psd <- predict(m2, type = "parameter", FUN = sd)
print(head(psd))

## End(Not run)
```

mixture_family

Construct a Finite Mixture Distribution Family

Description

Constructs a "gamlss2.family" representing a finite mixture of two or more component distribution families. Component distribution parameters and the mixing probabilities may all be modeled using separate additive predictors.

Usage

```
mixture_family(families = NO, k = NULL, reference = 1L,
               prefix = "c", initialize = TRUE)
```

Arguments

families	A single distribution family or a list of component families. Each entry may be a family-generating function, a family object, or a character family name accepted by the gamlss2 family infrastructure. If a single family is supplied, it is repeated k times.
k	Number of mixture components. For a single family, NULL defaults to two components. For a list of families, k is inferred from its length and, if supplied, must agree with that length.
reference	Integer identifying the reference component for the mixing-weight parameterization. Its unnormalized odds are fixed at one.
prefix	Either one non-empty character prefix, from which component labels such as "c1" and "c2" are formed, or one distinct label per component. Component parameter names are formed by joining the label and original parameter name with a dot.
initialize	Starting-value strategy. TRUE (the default) or "separated" uses separated empirical-quantile starts for recognized component location parameters, the component-family initializers for other parameters, and equal mixing probabilities. "cluster" uses deterministic response-based clusters to obtain component-specific parameter and mixing-probability starts. FALSE or "component" uses the unmodified component-family initializers and equal mixing probabilities. "modal" uses separated density modes when available, with response-scaled lower bounds for recognized scale parameters.

Details

For component densities $f_j(y)$ and probabilities π_j , the mixture density is

$$f(y) = \sum_{j=1}^k \pi_j f_j(y).$$

It is evaluated using a log-sum-exp calculation to avoid numerical underflow.

Component parameters are ordered component by component. For example, a two-component normal mixture has parameters c1.mu, c1.sigma, c2.mu, c2.sigma, and pi2. Consequently, a [gamlss2](#) formula for this family has five right-hand sides in that order.

Mixing probabilities use a baseline-logit parameterization analogous to [MN](#). For every non-reference component j , the parameter pi j has a log link and represents positive odds w_j relative to the reference component. With the reference odds fixed at one,

$$\pi_j = \frac{w_j}{1 + \sum_{\ell \neq r} w_\ell}, \quad \pi_r = \frac{1}{1 + \sum_{\ell \neq r} w_\ell}.$$

The probabilities() family function converts these odds to normalized probabilities.

Component scores are weighted by posterior responsibilities; missing Hessians are computed numerically. Working-response updates reuse component densities.

The default initialize = "separated" starts recognized locations at ordered empirical midpoint quantiles, uses component initializers for other parameters, and starts with equal mixing probabilities. "cluster" starts from response-based groups and their proportions; "component" uses

unmodified component initializers and equal probabilities. "modal" uses separated density modes and their nearest observations, falling back to separated starts if no suitable modes are found.

When modal starts are found, log-linked sigma has a lower bound of 0.8 times the density bandwidth and log-linked JSU tau has a lower bound of one. These bounds remain in effect during fitting. Invalid or unusable response-based starts fall back to full-response component initializers and then to the inverse-link value at zero.

The returned family includes `cdf()`, `quantile()`, `random()`, `mean()`, and `variance()` whenever all component families provide the corresponding operation. All components must use the same distribution type, for example all continuous or all discrete. Mixture quantiles are obtained by numerically inverting the mixture CDF.

As with finite mixtures generally, the likelihood can have local optima and equivalent solutions obtained by exchanging component labels. Separated initialization helps but does not impose an ordering constraint. Multiple fits, including fits using the optional "cluster" strategy or explicitly supplied starting values, can still be useful for difficult data.

Value

An object of class "gamlss2.family". In addition to the standard family elements, it contains:

- `probabilities(par, n = NULL)`: normalized prior mixing probabilities.
- `responsibilities(par, y)`: posterior component probabilities conditional on observations.
- `score_matrix(par, y)`: observation-wise scores for all mixture parameters, evaluated with shared responsibilities.
- `components`: the completed component families.
- `component_map`: mappings from original component parameter names to mixture parameter names.

See Also

[gamlss2](#), [gamlss2.family](#)

Examples

```
## Not run: ## waiting times in the faithful data are distinctly bimodal
data("faithful", package = "datasets")

## sort for plotting
d <- faithful[order(faithful$eruptions), ]

## construct and estimate a two-normal mixture
fam <- mixture_family(NO, k = 2)

## fit mixture pdf
fit_family(d$waiting, family = fam)

## mixture of experts model
m <- gamlss2(
  waiting ~ s(eruptions) | . | . | . | .,
  data = d,
```

```

    family = fam,
    criterion = "aicc"
  )

  ## estimated effects
  plot(m)

  ## predict parameters
  par <- predict(m, type = "parameter")
  head(par)

  ## prior and posterior component probabilities
  prior <- family(m)$probabilities(par)
  head(prior)
  posterior <- family(m)$responsibilities(par, d$waiting)
  head(posterior)

  ## classify observations by their largest posterior probability
  ## and show quantiles of the fitted distributions
  pc <- max.col(as.matrix(posterior))
  ccols <- c(2, 4)
  fq <- quantile(m)

  plot(
    waiting ~ eruptions,
    data = d,
    col = ccols[pc],
    pch = 19,
    xlab = "Eruption duration (minutes)",
    ylab = "Waiting time (minutes)",
    ylim = range(d$waiting, fq)
  )

  matlines(d$eruptions, fq, col = 1, lty = 1)

  legend(
    "topleft", paste("Component", 1:2),
    col = ccols, pch = 19, bty = "n"
  )

  ## mixture regression with a numeric and a binary predictor.
  data("mtcars", package = "datasets")
  cars <- transform(
    mtcars,
    am = factor(am, levels = 0:1, labels = c("automatic", "manual"))
  )

  ## both component means depend on weight and transmission,
  ## component scales and the mixing probability are constant
  m <- gamlss2(
    mpg ~ s(wt) + am | 1 | s(wt) + am | 1 | 1,
    data = cars,
    family = fam
  )

```

```

)

cars_par <- predict(m, type = "parameter")
cars_posterior <- family(m)$responsibilities(cars_par, cars$mpg)
cars_component <- max.col(as.matrix(cars_posterior))
table(cars$am, cars_component)

## plot estimated classes and component mean regressions,
## point shape and line type distinguish automatic
## from manual transmission.
wt_grid <- seq(min(cars$wt), max(cars$wt), length.out = 100)
newcars <- expand.grid(wt = wt_grid, am = levels(cars$am))
newcars$am <- factor(newcars$am, levels = levels(cars$am))
fit <- predict(m, newdata = newcars, type = "parameter")

plot(
  mpg ~ wt,
  data = cars,
  col = ccols[cars_component],
  pch = c(1, 16)[cars$am],
  xlab = "Weight (1000 lbs)",
  ylab = "Miles per gallon",
  ylim = range(cars$mpg, fit)
)

for(j in 1:2) {
  for(a in seq_along(levels(cars$am))) {
    keep <- newcars$am == levels(cars$am)[a]
    lines(
      newcars$wt[keep], fit[keep, paste0("c", j, ".mu")],
      col = ccols[j], lty = a, lwd = 2
    )
  }
}

legend(
  "topright", paste("Component", 1:2),
  col = ccols, lty = 1, lwd = 2, bty = "n"
)
legend(
  "bottomleft", levels(cars$am),
  col = 1, pch = c(1, 16), lty = 1:2, bty = "n"
)

## End(Not run)

```

Description

The `MN()` family implements a multinomial logit model for categorical responses with $k \geq 2$ unordered categories.

Usage

```
MN(k)
```

Arguments

`k` An integer specifying the number of response categories.

Details

The response must be a factor with exactly `k` levels. The first level is treated as the reference category.

For categories $j = 2, \dots, k$, each predictor models the log-odds relative to the reference category.

The family provides analytical first and second derivatives of the log-likelihood with respect to the predictors and a `probabilities()` method for fitted category probabilities.

Value

A "gamlss2.family" object to be used with [gamlss2](#).

See Also

[gamlss2](#), [gamlss2.family](#)

Examples

```
## Not run: data("marital.nz", package = "VGAM")

## multinomial model
m <- gamlss2(mstatus ~ s(age) | . | ., data = marital.nz,
  family = MN(4), maxit = 100)

## plot effects
plot(m)

## predict probabilities
par <- predict(m, type = "parameter")
probs <- family(m)$probabilities(par)
names(probs) <- levels(marital.nz$mstatus)

## plot
i <- order(marital.nz$age)
matplot(marital.nz$age[i], probs[i, ], type = "l", lty = 1, lwd = 2,
  xlab = "Age", ylab = "Probabilities")
legend("center", names(probs), lty = 1, lwd = 2, col = 1:4)

## End(Not run)
```

ms

*Monotonic P-Splines for GAMLSS***Description**

Constructs a monotonic penalized B-spline (P-spline) model term for use with `gamlss2`. The smooth can be constrained to be monotonically increasing or decreasing. The smoothing parameter is either supplied by the user or selected automatically during estimation.

Usage

```
ms(x, mono = c("up", "down"), k = 10,
   criterion = "aicc", lambda = NULL,
   lambda.min = 1e-8, lambda.max = 1e8,
   lambda.tol = 1e-4, ...)
```

Arguments

<code>x</code>	A numeric covariate to be represented by a monotonic P-spline.
<code>mono</code>	Character specifying the direction of monotonicity. Use "up" for a monotonically increasing effect or "down" for a monotonically decreasing effect.
<code>k</code>	Integer giving the dimension of the B-spline basis. The default is 10.
<code>criterion</code>	Character specifying the criterion used for automatic smoothing parameter selection. Available criteria are "aicc", "aic", "gaic", "bic", and "gcv". The default is "aicc".
<code>lambda</code>	Optional positive smoothing parameter. If NULL, the smoothing parameter is selected automatically. If supplied, <code>lambda</code> is kept fixed during fitting.
<code>lambda.min, lambda.max</code>	Positive lower and upper limits for automatic smoothing parameter selection. The search is performed on the logarithmic scale.
<code>lambda.tol</code>	Tolerance used in the one-dimensional optimization of the smoothing parameter.
<code>...</code>	Further arguments passed to <code>s</code> when constructing the P-spline basis.

Details

Function `ms()` defines a special model term for `gamlss2`. The underlying basis is an mgcv P-spline basis constructed with `bs = "ps"`.

For an increasing smooth, adjacent B-spline coefficients are constrained to satisfy

$$\beta_{r+1} - \beta_r \geq 0,$$

whereas for a decreasing smooth the inequalities are reversed. Because B-spline basis functions are non-negative and form a partition of unity, ordering the spline coefficients in this way produces a monotonic fitted effect.

Estimation is based on penalized weighted least squares. For a given smoothing parameter λ , the coefficient vector minimizes

$$(z - X\beta)^\top W(z - X\beta) + \lambda\beta^\top S\beta$$

subject to the monotonicity constraints. The constrained problem is solved with an active-set algorithm. Weighted cross-products are computed once per backfitting update, so smoothing parameter selection only requires operations on matrices having dimension determined by k .

If `lambda = NULL`, the smoothing parameter is selected by a one-dimensional search on `log(lambda)` using the criterion specified by `criterion`.

The fitted term is centered before it is returned to the additive backfitting algorithm.

Value

The constructor `ms()` returns an object of classes "special" and "ms" containing the P-spline design and penalty matrices and information required for fitting and prediction.

After fitting with `gamlss2`, the corresponding fitted special term contains the following main components:

- `coefficients`: estimated B-spline coefficients.
- `fitted.values`: centered fitted contribution of the term.
- `edf`: effective degrees of freedom.
- `lambdas`: selected or fixed smoothing parameter.
- `vcov`: covariance/influence matrix conditional on the active constraints.
- `df`: residual degrees of freedom for the term, $n - \text{edf}$.

Additional components are stored for prediction and for warm starts in subsequent backfitting iterations.

References

- Eilers PHC, Marx BD (1996). "Flexible Smoothing with B-Splines and Penalties". *Statistical Science*, **11**(2), 89–121. doi:[10.1214/ss/1038425655](https://doi.org/10.1214/ss/1038425655)
- Bollaerts K, Eilers PHC, van Mechelen I (2006). "Simple and Multiple P-Splines Regression with Shape Constraints". *British Journal of Mathematical and Statistical Psychology*, **59**(2), 451–469. doi:[10.1348/000711005X84293](https://doi.org/10.1348/000711005X84293)

See Also

[gamlss2](#), [special_terms](#)

Examples

```
## Not run: ## air quality data
air <- subset(airquality, !is.na(Ozone))
air <- air[order(air$Temp), ]

## monotone Temp effect for mu and sigma
```

```
## Wind is modeled by an unconstrained smooth
f <- Ozone ~ s(Wind) + ms(Temp, "up") | ms(Temp, "down")

## estimate model
m <- gamlss2(f, data = air, family = GA)

## model summary
summary(m)

## visualize estimated effects
plot(m)

## End(Not run)
```

n

Neural Network Model Terms for GAMLSS

Description

Constructs a feed-forward neural network model term for use with `gamlss2`. The term is fitted using `nnet` with a linear output unit and is integrated into the iterative backfitting algorithm for distributional regression.

Usage

```
n(formula, ...)
```

Arguments

formula	A one-sided formula specifying the predictors entering the neural network, for example <code>~ x1 + x2</code> . Numeric and factor predictors can be used.
...	Control arguments for the neural network fit. The currently supported arguments are <code>size</code> , the number of units in the hidden layer; <code>maxit</code> , the maximum number of iterations; <code>decay</code> , the weight-decay parameter; <code>trace</code> , indicating whether optimization progress should be printed; <code>MaxNWts</code> , the maximum allowable number of weights; and <code>scale</code> , indicating whether numeric predictors should be scaled before fitting. Local decay selection uses criterion (" <code>aic</code> ", " <code>aicc</code> ", " <code>bic</code> ", " <code>gaic</code> ", or " <code>gcv</code> "), an optional <code>decay.grid</code> , and <code>K</code> , the EDF penalty for " <code>gaic</code> " (default 2). Defaults are <code>size = 50</code> , <code>maxit = 1000</code> , <code>decay = 0.1</code> , <code>trace = FALSE</code> , <code>MaxNWts = 10000</code> , and <code>scale = TRUE</code> .

Details

Function `n()` defines a special neural network model term for `gamlss2`. At each backfitting update, the current working response and weights are passed to `nnet` and a feed-forward neural network is fitted with `linout = TRUE`. Hence, the neural network contributes a continuous additive predictor to the corresponding distributional parameter.

The formula supplied to `n()` defines the inputs to the network. For example, `n(~ Temp + Wind)` fits a neural network using Temp and Wind as inputs.

By default, each numeric predictor is scaled to the interval $[0, 1]$ using its observed range before the network is fitted. Factor predictors are left unchanged. The same scaling information is stored with the fitted term and is applied automatically when predicting from new data. Scaling can be disabled with `scale = FALSE`.

The fitted effect is centered by subtracting its mean before it is returned to the additive predictor.

The effective degrees of freedom (edf) use a Gauss–Newton approximation with the current working weights and decay penalty. The calculation accounts for shrinkage and centering of the fitted effect.

To select decay during backfitting, supply a supported criterion in `n()` or `gamlss2()`. Selection minimizes the local criterion based on the centered working residual sum of squares and EDF, using the same criterion formulas as other smooth terms. This is a working-model criterion rather than the full model likelihood. Without a supported criterion, decay remains fixed; `criterion = "fixed"` in `n()` also overrides a global criterion. A vector of per-weight decay penalties remains fixed under global criteria.

Selection requires a scalar decay, which supplies the starting value. By default, each update compares the current decay with values ten times smaller and larger, bounded by 10^{-8} and 10^8 ; zero decay starts the positive grid at 0.1. Alternatively, `decay.grid` supplies finite nonnegative candidates. The current decay is always included. All candidates start from the same network weights, and the selected decay and fitted weights are carried to the next update.

Value

The constructor `n()` returns an object of classes "special" and "n" containing the model formula, input data, neural network control arguments, and optional scaling information required for fitting.

See Also

[gamlss2](#), [special_terms](#), [nnet](#)

Examples

```
## Not run:
## air quality data
air <- subset(airquality, !is.na(Ozone))
air <- air[order(air$Temp), ]

## Neural network effect of Temp and Wind for mu.
## A smaller hidden layer is used here for illustration.
set.seed(1328)
f <- Ozone ~ n(~ Temp + Wind, size = 10, decay = 0.05) | 1

## estimate model
m <- gamlss2(f, data = air, family = GA)

## model summary
summary(m)
```

```

## visualize estimated effects
plot(m)

## create new data:
## vary temperature while holding wind constant
nd <- with(air, data.frame(
  Temp = seq(min(Temp), max(Temp), length.out = 100),
  Wind = median(Wind)
))

## fitted conditional quantiles
pq <- quantile(
  m,
  newdata = nd,
  probs = c(0.05, 0.5, 0.95)
)

## visualize
plot(Ozone ~ Temp, data = air, pch = 19,
  col = rgb(0.1, 0.1, 0.1, alpha = 0.3),
  xlab = "Temperature (degrees F)",
  ylab = "Ozone (ppb)")
matlines(nd$Temp, pq,
  lwd = 2,
  lty = c(2, 1, 2),
  col = 4)

## End(Not run)

```

new_formula

Extracting a New Formula After Selection Algorithms

Description

The generic function extracts the selected model terms from a fitted selection procedure and returns the corresponding reduced model formula.

Usage

```
new_formula(object, ...)
```

Arguments

object	A fitted model object produced by a selection routine such as step_gamlss2 or select_gamlss2 .
...	Not used yet.

Details

The function is intended for fitted models that store the outcome of an automatic term-selection step in `object$selection$formula`. It extracts the selected right-hand side terms and restores the original response on the left-hand side.

For multi-parameter distributional models, the returned object is a [Formula](#) with one right-hand side part for each distribution parameter. This makes it convenient to inspect the final model or to refit it without repeating the selection step.

Value

A [Formula](#) containing the selected model terms.

See Also

[step_gamlss2](#), [select_gamlss2](#), [gamlss2](#)

Examples

```
## no runnable example; see \code{\link{step_gamlss2}} and
## \code{\link{select_gamlss2}} for end-to-end usage
```

OL

Ordered Logistic Family for Ordinal Regression

Description

Defines the ordered logistic (cumulative logit) family for ordinal responses in `gamlss2`.

Usage

`OL(k)`

`ologit(k)`

Arguments

`k` A single integer specifying the number of response categories. Must be $k \geq 2$.

Details

The response must be coded as the integers $1, \dots, k$, or as a factor with exactly these levels.

Let γ_j denote the cutpoint between categories j and $j + 1$. The cutpoints are parameterized as

$$\gamma_1 = \theta_1$$

and, recursively,

$$\gamma_j = \gamma_{j-1} + \exp(\delta_j), \quad j = 2, \dots, k - 1.$$

Equivalently,

$$\gamma_j = \gamma_1 + \sum_{\ell=2}^j \exp(\delta_\ell),$$

which ensures strictly ordered cutpoints.

The cumulative probabilities are

$$\Pr(Y > j) = \text{logit}^{-1}(\text{location} - \gamma_j), \quad j = 1, \dots, k - 1.$$

Each parameter can use an additive predictor through the `"|"` formula interface. The family also provides category and transition probabilities, and the mean and variance of numeric category labels.

`ologit()` is retained as a compatibility alias for `OL()`.

Value

A `"gamlss2.family"` object with parameters `location`, `theta1` (θ_1), and, for $k > 2$, `deltaj` (δ_j) for $j = 2, \dots, k - 1$.

See Also

[gamlss2](#), [gamlss2.family](#), [polr](#)

Examples

```
## example using the housing data from the MASS package
library("MASS")

## fit standard cumulative logit model using polr()
m1 <- polr(Sat ~ Infl + Type + Cont, weights = Freq, data = housing)
summary(m1)

## OL() uses integer category labels 1, ..., k
housing$Satint <- as.integer(housing$Sat)

## fit the corresponding model using gamlss2
m2 <- gamlss2(Satint ~ Infl + Type + Cont,
  data = housing, weights = Freq,
  family = OL(k = 3))
summary(m2)

## compare coefficients
coef(m1)
coef(m2)

## predict class probabilities
pm1 <- predict(m1, type = "p")
pm2 <- family(m2)$probabilities(predict(m2, type = "parameter"))

print(head(pm1))
print(head(pm2))
```

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.spec](#)

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

```
## Not run: ## air quality data
air <- subset(airquality, !is.na(Ozone))
air <- air[order(air$Temp), ]

## specify the model formula
f <- Ozone ~ pb(Temp) | . | 1 | 1

## estimate model
m <- gamlss2(f, data = air, family = BCPE)

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

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

## predict quantiles
pq <- quantile(m, probs = c(0.05, 0.5, 0.95))

## plot
plot(Ozone ~ Temp, data = air, pch = 19,
     col = rgb(0.1, 0.1, 0.1, alpha = 0.3))
matlines(air$Temp, pq,
        lty = 1, col = 4, lwd = 2)

## End(Not run)
```

plot.gamlss2

*Plotting GAMLSS***Description**

Plotting methods for objects of class "gamlss2" and "gamlss2.list". These methods support effect plots for model terms and residual diagnostics. Effect plots for model terms with more than two covariates are not supported; in such cases, use [predict](#) and plot the results manually.

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

<code>x</code>	An object of class "gamlss2" or "gamlss2.list". Objects of class "gamlss2.list" can be created by combining "gamlss2" objects with <code>c()</code> . See the examples.
<code>parameter</code>	Character or integer specifying for which distribution parameter the plots should be created. Defaults to all parameters.
<code>which</code>	Character or integer selecting the type of plot. "effects" produces effect plots of model terms, "resid" shows the available residual diagnostics, "hist-resid" produces a histogram of residuals, "qq-resid" a quantile-quantile plot of residuals, "wp-resid" a worm plot of residuals, and "scatter-resid" a scatter plot of residuals against fitted values for the distribution median. In addition, "selection" plots the GAIC path stored for a fitted stepwise or selection object, and "samples" visualizes posterior or simulation draws for objects that contain samples.
<code>terms</code>	Character or integer. Which model term(s) should be plotted?
<code>scale</code>	Logical. If TRUE, effect plots all have the same scale on the y-axis. If FALSE, each effect plot has its own y-axis scale.
<code>spar</code>	Logical. Should graphical parameters be set automatically?
<code>legend</code>	Logical. Should a legend be added for plots with multiple models?
<code>...</code>	Additional graphical arguments such as <code>lwd</code> , <code>lty</code> , <code>col</code> , <code>legend = TRUE</code> (for multiple model plots), among others, depending on the type of plot.

Details

For objects of class "gamlss2", `which = "effects"` is the default and plots the partial effects stored in `results(object)`. If no effect information is available, residual diagnostics are shown instead. Effect uncertainty uses fast term-local covariance by default. Pass `method = "joint"` through `...` to rebuild effects from the full joint covariance.

Linear effects are constructed separately for each formula term, so a main effect does not include columns belonging to an interaction. A linear interaction between two numeric covariates is displayed as an image and contour plot, in the same way as a bivariate smooth effect. Linear terms involving more than two covariates are not plotted automatically.

For objects of class "gamlss2.list", plots compare effect estimates across several fitted models. Residual diagnostics are only available for single-model objects.

See Also

[gamlss2](#).

Examples

```
## Not run: ## model formula
f <- Ozone ~ s(Temp) + s(Wind) + s(Solar.R) | .

## estimate model
m1 <- gamlss2(f, data = airquality, family = GA)
```

```

## plot effects (default)
plot(m1)

## plot specific effect
plot(m1, parameter = "sigma")
plot(m1, parameter = "sigma")
plot(m1, parameter = "mu", terms = 1)
plot(m1, parameter = "mu", terms = 2)
plot(m1, parameter = "mu", terms = "Solar")
plot(m1, terms = "Solar.R")

## plot all residual diagnostics
plot(m1, which = "resid")

## single diagnostic plots
plot(m1, which = "hist-resid")
plot(m1, which = "qq-resid")
plot(m1, which = "wp-resid")
plot(m1, which = "scatter-resid")

## if available, plot stored step_gamlss2()-selection path
## plot(m1, which = "selection")

## End(Not run)

```

predict.gamlss2	<i>Predictions from gamlss2 Models</i>
-----------------	--

Description

Predict response distributions, parameters, additive predictors, terms, or response means from a **gamlss2** model.

Usage

```

## S3 method for class 'gamlss2'
predict(object, parameter = NULL, newdata = NULL,
  type = c("distribution", "parameter", "link", "response", "terms"),
  terms = NULL, se.fit = FALSE, drop = TRUE, ...,
  level = NULL, interval = c("none", "wald"), unconditional = FALSE)

```

Arguments

object	A gamlss2 model.
parameter	Distribution parameter name(s) or index(es), such as "mu" or "sigma"; defaults to all parameters.
newdata	Data frame for predictions; defaults to the fitted data.

type	"distribution" (default) returns distributions3 objects; "parameter" returns parameters on their natural scale; "link" returns additive predictors; "response" returns response means; "terms" returns individual additive terms. Specifying parameter without type selects "parameter".
terms	Additive terms to include; defaults to all. Names are partially matched unless <code>nogrep = TRUE</code> is passed in . . .
se.fit	For ordinary predictions, TRUE computes standard errors from coefficient draws; R in . . . sets the number of draws (default 200).
drop	Simplify the result when possible.
level	Numeric confidence level(s) between zero and one; supplying levels selects pointwise Wald intervals.
interval	"none" (default) returns ordinary predictions; "wald" returns analytic pointwise confidence intervals, at the 95% confidence level by default.
unconditional	If TRUE, include approximate smoothing-parameter uncertainty in Wald intervals and coefficient draws. This requires a likelihood fit without stored posterior samples.
. . .	Further controls, including FUN (a function summarizing predictions across coefficient draws), R (number of Gaussian draws, default 200), seed (default 123; FALSE continues the current random-number stream), and burnin for simulation. For likelihood fits, FUN or <code>se.fit = TRUE</code> requests simulation; R alone does not.

Details

Coefficient draws stored in the fit are used when available; R does not replace stored posterior samples. Otherwise, Gaussian draws use the full joint coefficient covariance, retaining dependence among terms and distributional parameters. For penalized likelihood fits, this approximation is motivated by the Bayesian interpretation of smoothing penalties. Nonlinear transformations can produce asymmetric uncertainty distributions.

To propagate uncertainty to a quantile or event probability, retain the joint distribution draws with `FUN = identity`, evaluate the quantity within each draw, and then summarize across draws at each prediction location. These intervals describe estimation uncertainty in the chosen quantity. Quantiles of the fitted response distribution instead give plug-in prediction intervals for a future observation.

Wald intervals are approximate pointwise *confidence* intervals for fitted values, not *prediction* intervals for future observations. They are computed without simulation and may be asymmetric on transformed scales. They condition on fitted smoothing parameters by default and exclude model-selection uncertainty; see [vcov.gamlss2](#).

Wald intervals support maximum-likelihood fits with linear terms and coefficient-linear **mgcv** smooths, but not nonlinear custom special terms. They cannot be combined with FUN, R, seed, or burnin. An indefinite joint information matrix or non-estimable coefficient causes an error rather than a working-information approximation.

Value

Ordinary predictions return a distribution object or values in the selected form. Term predictions are data frames or, for multiple parameters, lists of matrices. With `se.fit = TRUE`, simulation

summaries use names `fit` (simulation mean) and `se` (simulation standard deviation), prefixed by parameter or term names when needed. A supplied `FUN` determines the simulation summary; its center can differ from the prediction at the estimated coefficients. With `FUN = identity`, simulated distribution predictions form a list of distribution objects, one per joint coefficient draw.

Wald predictions return `fit`, `se.fit`, `lower`, and `upper`. For multiple levels, `lower` and `upper` are lists named by confidence percentage (for example, "80%" and "95%"). `fit` and each bound retain the structure selected by `parameter`, `terms`, and `drop`; `se.fit` is shared across levels.

See Also

[predict](#), [marginal_predict](#), [prodist.gamlss2](#), [quantile.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: a "distribution" object
d <- predict(m, newdata = nd)
print(d)

## can be used with distributions3 methods
pdf(d, 20)
cdf(d, 20)
quantile(d, 0.8)
mean(d)
variance(d)
random(d, 3)

## additive predictors on the link scale
predict(m, newdata = nd, type = "link")

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

## same as
mean(d)

## all model parameters
predict(m, newdata = nd, type = "parameter")

## parameter selects parameter predictions if type is omitted
predict(m, newdata = nd, parameter = "sigma")
predict(m, newdata = nd, parameter = "sigma", drop = FALSE)

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

```

predict(m, newdata = nd, type = "terms", parameter = "sigma", terms = "s(speed)")

## predict quantiles
quantile(m, newdata = nd, probs = c(0.1, 0.5, 0.9))

## same as
quantile(d, probs = c(0.1, 0.5, 0.9), elementwise = FALSE)

## standard errors
predict(m, type = "parameter", newdata = nd, se.fit = TRUE, R = 200)

## fast pointwise confidence bands, without sampling
bands <- predict(m, type = "parameter", newdata = nd, level = c(0.8, 0.95))
bands$lower[["80%"]]
bands$upper[["95%"]]

## prediction uncertainty using multiple draws
d <- predict(m, newdata = cars, R = 1000, FUN = identity)

## predict 1000 quantiles
q <- sapply(d, quantile, probs = 0.95)

i95 <- function(x) {
  setNames(quantile(x, c(0.025, 0.5, 0.975), names = FALSE),
    c("lower", "fit", "upper"))
}

## quantiles of quantile samples
q <- t(apply(q, 1, i95))

## plot quantile with uncertainty bands
plot(dist ~ speed, data = cars,
  ylim = range(cars$dist, q))
matlines(cars$speed, q,
  lty = c(2, 1, 2), col = 4, lwd = 2)

```

prodist.gamlss2	<i>Extracting Fitted or Predicted Probability Distributions from gamlss2 Models</i>
-----------------	---

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

For ordinary predictions, `prodist(object, newdata = ...)` returns the same distributions as the default `predict(object, newdata = ...)`. Use `prodist` when code needs a common interface for extracting probability distributions from different supported model classes. Omit `newdata` to obtain fitted distributions.

The returned objects support **distributions3** methods such as `pdf()`, `cdf()`, `quantile()`, and `random()`. They describe response variation conditional on the fitted parameters; they do not include parameter-estimation uncertainty.

Value

A **distributions3** object inheriting from `distribution`. Its concrete class depends on the family: a **distributions3** family retains its distribution class (e.g., `Normal`); families without a distribution constructor use `GAMLSS2`.

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

`GAMLSS2`, `family.distribution`, `predict.gamlss2`

Examples

```
## Not run: ## fit a Gamma model for stopping distance given speed
library("distributions3")
data("cars", package = "datasets")
m <- gamlss2(dist ~ s(speed) | s(speed),
  data = cars, family = GA)

## prodist() and default predict() return the same distributions
nd <- data.frame(speed = c(10, 20, 30))
d <- prodist(m, newdata = nd)
all.equal(d, predict(m, newdata = nd))
print(d)

## evaluate the predicted distributions
quantile(d, c(0.05, 0.5, 0.95), elementwise = FALSE)
mean(d)
variance(d)
pdf(d, c(10, 50, 100))
cdf(d, c(10, 50, 100))
```

```

random(d, 5)

## fitted distributions reproduce the model log-likelihood
sum(log_pdf(prodist(m), cars$dist))
logLik(m)

## End(Not run)

```

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

<code>x</code>	An object of class "gamlss2".
<code>probs</code>	Numeric vector of probabilities with values in [0,1].
<code>variable</code>	Covariate selector for single-covariate models: TRUE uses the sole covariate; an integer or character selects by column position or name. NULL (default) or FALSE computes quantiles at the supplied or fitted observations without a covariate grid.
<code>newdata</code>	Data frame that should be used for computing the quantiles.
<code>plot</code>	Logical, should an ordered quantile plot or a covariate plot be shown?
<code>data</code>	Logical, should observed responses be added to the plot when available?
<code>n</code>	Integer, number of observations that should be used to compute an equidistant grid for the selected variable.
<code>...</code>	Arguments such as <code>col</code> , <code>legend = TRUE/FALSE</code> . See the examples.

Details

Quantiles are computed from the predicted response distributions.

With `plot = TRUE` and no selected variable, the x-axis shows observation ranks as percentages, ordered by fitted median; the y-axis is on the response scale. Lines show the fitted conditional quantiles at `probs`, and points show observed responses when `data = TRUE`. The same ordering is used for the responses and all quantiles, so each position refers to one observation. If `newdata` does not contain the response, only the fitted quantile curves are shown.

Value

A data frame with one row per prediction and one column per probability. With `variable = NULL`, a single probability returns a numeric vector. When a covariate is selected, it is included as the first column. With `plot = TRUE`, the result is returned invisibly.

See Also

[gamlss2](#).

Examples

```
## Not run: ## load data
data("SpirometryUS", package = "gamlss2")

## subset for female
d <- subset(SpirometryUS, gender == "female")

## model formula
f <- fev1 ~ s(age) | . | . | .

## estimate model
m <- gamlss2(f, data = d, family = BCPE)

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

## plot quantiles ordered by the fitted median
quantile(m, plot = TRUE)

## quantile plot using covariate data
quantile(m, plot = TRUE,
  variable = TRUE, pos = "topright")

## plot without raw data
quantile(m, plot = TRUE,
  variable = TRUE, pos = "topright", data = FALSE)

## End(Not run)
```

Description

Random effects can be included in `gamlss2` models in two ways. The first uses `s()` with `bs = "re"` for simple random effects, i.e., when a single factor is entered into the model as a smoother. This approach relies on the `s()` function from the **mgcv** package. For example, if `x` is a factor with

several levels, then `s(x, bs = "re")` shrinks the level-specific effects of `x` towards their overall mean.

The second, more general approach uses the model term constructor `re()`, which provides an interface to the specialised random-effects functionality in the **nlme** package. This allows fitting more complex random-effects structures.

This documentation focuses on the `re()` function, but we also provide examples using `s(..., bs = "re")`.

Usage

```
re(random, correlation = NULL, ...)
```

Arguments

<code>random</code>	A formula specifying the random effect part of the model, as in the lme function.
<code>correlation</code>	An optional correlation object, see lme .
<code>...</code>	For the <code>re()</code> function, the dots argument is used to specify additional control arguments for the lme function, such as the <code>method</code> and <code>correlation</code> 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

```
## Not run: ## orthodontic 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,
```

```

## setup formula first
f <- distance ~ age11 + re(~age11|Subject)

## estimate model
m1 <- gamlss2(f, data = Orthodont)

## compare fitted values
plot(fitted(m1, parameter = "mu"), fitted(m))
abline(0, 1, col = 4)

## extract summary for re() model term
st <- specials(m1, parameter = "mu", term = "age11", 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(m1, parameter = "mu"), fitted(m))
points(fitted(a, parameter = "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 ~ sin1 + cos1 +
  re(~ 1 | Mare) +
  re(~ sin1 - 1 | Mare, correlation = corARMA(q = 2))

## estimate model
m2 <- 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(m2, parameter = "mu")[Ovary$Mare == j]
lines(f ~ ds$Time, col = 4, lwd = 2)

f <- fitted(g, parameter = "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)
}

## simulated data
set.seed(1328)

n <- 10000
k <- 500

## generate random effects
f <- as.factor(sample(1:k, size = n, replace = TRUE))
ref <- rnorm(k, sd = 0.6)

## random effects only for sigma
y <- rBCT(n, mu = 10, sigma = exp(-1 + ref[f]), tau = 10)

## estimate model
m3 <- gamlss2(y ~ 1 | re(~ 1 | f), family = BCT)

## extract fitted random effects
cb <- coef(m3, full = TRUE)
cb <- cb[grep1("re", names(cb), fixed = TRUE)]

## compare
plot(cb, ref, main = round(mean((cb - ref)^2), 2))
abline(0, 1, lwd = 2, col = 4)

## End(Not run)

```

residuals.gamlss2

Residuals for GAMLSS Models

Description

Extracts residuals from fitted `gamlss2` objects. By default, normalized quantile residuals are returned. Response residuals can be obtained as observed responses minus fitted distribution means. A plotting method for residual diagnostic plots is also provided.

Usage

```
## S3 method for class 'gamlss2'
residuals(object,
  type = c("quantile", "response", "parameter"), newdata = NULL, ...)

## S3 method for class 'gamlss2.residuals'
plot(x, which = NULL, spar = TRUE, ...)
```

Arguments

<code>object</code>	An object of class "gamlss2".
<code>type</code>	Character specifying the type of residuals to be computed. The default "quantile" computes normalized quantile residuals. "response" computes observed responses minus fitted distribution means. "parameter" is reserved for parameter-specific residuals and is currently not implemented.
<code>newdata</code>	An optional data frame in which to evaluate the residuals. If omitted, residuals are evaluated for the data used to fit the model. When supplied, <code>newdata</code> must also contain the response variable.
<code>...</code>	Additional arguments passed on to methods, family-specific residual functions, or predict.gamlss2 .
<code>x</code>	An object of class "gamlss2.residuals", typically produced by <code>residuals.gamlss2</code> .
<code>which</code>	Character or integer selecting the diagnostic plot(s). Available plots are "hist-resid" for a histogram with kernel density estimate, "qq-resid" for a normal quantile-quantile plot, and "wp-resid" for a worm plot. If omitted, all three plots are produced. Integers 1, 2, and 3 select these plots in the same order.
<code>spar</code>	Logical. Should graphical parameters be set automatically for multiple plots?

Details

Normalized quantile residuals are obtained by evaluating the fitted distribution function at the observed responses and transforming the resulting probabilities to the standard normal scale. For continuous response distributions this is

$$r_i = \Phi^{-1}\{F(y_i|\hat{\theta}_i)\},$$

where $F(\cdot)$ is the fitted cumulative distribution function and $\hat{\theta}_i$ denotes the fitted distribution parameters for observation i .

For discrete response distributions, randomized quantile residuals are used: a uniform random probability is drawn between the fitted distribution probabilities just below and at the observed response before applying the standard normal quantile function. Thus, repeated calls may differ for discrete families unless the random-number seed is fixed.

If the family object supplies a family-specific residual function, this is used. Otherwise, quantile residuals are computed from the family cumulative distribution function. If no cumulative distribution function is available, the function falls back to response residuals and issues a warning.

Response residuals are computed as

$$y_i - \mu(\hat{\theta}_i),$$

where $\mu()$ is the family mean function if available. Otherwise the first distribution parameter is used as the fitted mean.

Non-finite residuals are replaced by NA. The plotting method provides simple residual diagnostics for checking whether quantile residuals are approximately standard normal.

Value

A numeric vector of residuals with class "gamlss2.residuals" and an attribute "type" indicating the residual type. The plotting method is called for its side effect and returns NULL invisibly.

References

Dunn PK, Smyth GK (1996). "Randomized Quantile Residuals". *Journal of Computational and Graphical Statistics*, **5**(3), 236–244. doi:[10.1080/10618600.1996.10474708](https://doi.org/10.1080/10618600.1996.10474708)

See Also

[gamlss2](#), [predict.gamlss2](#), [plot.gamlss2](#)

Examples

```
## data
data("cars", package = "datasets")

## fit heteroscedastic normal GAMLSS model
m <- gamlss2(dist ~ s(speed) | s(speed), data = cars, family = NO)

## plot estimated quantiles
p <- quantile(m, probs = c(0.05, 0.5, 0.95))
plot(dist ~ speed, data = cars, ylim = range(cars$dist, p))
matlines(cars$speed, p, lty = 1, col = 4, lwd = 2)

## normalized quantile residuals
rq <- residuals(m)
summary(rq)
attr(rq, "type")

## response residuals
rr <- residuals(m, type = "response")
summary(rr)

## residuals for new data
nd <- cars[1:10, ]
residuals(m, newdata = nd)

## diagnostic plots
plot(rq)
plot(rq, which = "qq-resid")
```

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)
```

 results

*Extract Results from GAMLSS Models***Description**

Extracts partial effects and corresponding uncertainty intervals from a fitted `gamlss2` model. The results are mainly used for plotting.

Usage

```
results(x, data = NULL, ...)

## S3 method for class 'gamlss2'
results(x, data = NULL, ...)
```

Arguments

<code>x</code>	An object of class "gamlss2".
<code>data</code>	An optional data frame used to construct the effects. By default, the model frame of <code>x</code> is used.
<code>...</code>	Further arguments controlling uncertainty intervals, including <code>interval</code> , <code>level</code> , <code>method</code> , and <code>nsim</code> .

Value

A list of class "results" whose `effects` component is a named list of partial-effect data frames. Each data frame contains covariate values and fitted effects (`fit`), with interval bounds (lower and upper) when available.

See Also

[gamlss2](#), [plot.gamlss2](#)

Examples

```
## Not run: ## estimate model
m <- gamlss2(dist ~ s(speed), data = cars, family = GA)

## extract results
r <- results(m)
names(r$effects)

## plot estimated effects
plot(r)

## End(Not run)
```

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

<code>x</code>	The full model matrix to be used for fitting.
<code>y</code>	The response vector or matrix.
<code>specials</code>	A named list of special model terms, e.g., including design and penalty matrices for fitting smooth terms using smooth.construct .
<code>family</code>	A family object, see gamlss2.family .
<code>offsets</code>	If supplied, a list or data frame of possible model offset.
<code>weights</code>	If supplied, a numeric vector of weights.
<code>start</code>	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 .
<code>xterms</code>	A named list specifying the linear model terms. Each named list element represents one parameter as specified in the family object.
<code>sterms</code>	A named list specifying the special model terms. Each named list element represents one parameter as specified in the family object.
<code>control</code>	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 1 to 3, the stopping criterion for the global, per-parameter, and CG correction loops. The CG tolerance defaults to the per-parameter tolerance.
- `maxit`: Integer vector of length 1 to 3, the maximum number of global parameter sweeps and per-parameter IRLS/backfitting iterations. Default is `maxit = c(20, 50)`. For the CG algorithm, an optional third element limits the internal correction sweeps and defaults to the second element. RS checks global convergence after each parameter sweep and ignores the third element.
- `step`: Numeric, the step length control parameter. Default is `step = 1`. With fixed smoothing parameters, it scales complete RS parameter updates or CG correction sweeps after the first update. Smoothing-parameter changes and their coefficients are accepted or rejected together. Smaller steps may require a smaller `eps`.
- `autostep`: Logical, should an update that decreases its objective be halved automatically? Default is TRUE. Updates use the likelihood, penalized likelihood, or smoothing-selection criterion as appropriate; see `gamlss2_smoothing`.
- `CG`: Integer, the outer iteration in which to start the CG correction. Thus, `CG = 5` uses RS for the first four outer iterations and CG in the fifth. `CG = 0` starts with CG immediately; the default is `CG = Inf`.
- `trace`: Logical, should information be printed while the algorithm is running?
- `rs.cache`: Logical, reuse generated family evaluations and weighted matrix calculations with identical inputs within a fit? Default is TRUE; set FALSE to disable caching for comparisons.
- `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`.

For smoothing parameter selection, numerical solvers, and examples, see `gamlss2_smoothing`.

Use these functions as templates for custom fitting algorithms, supplied via `optimizer` in `gamlss2_control` or the family object's "optimizer" element. For new families, see `gamlss2.family`.

Value

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

- | | |
|------------------------------|--|
| <code>fitted.values</code> | A data frame of the fitted values of the predictors of the selected distribution. |
| <code>fitted.specials</code> | A named list, one element for each parameter of the distribution, containing the fitted model object information of special model terms. |
| <code>fitted.linear</code> | 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.
converged	Logical, did the global iteration satisfy the stopping criterion?
logLik	The final value of the log-likelihood of the model.
control	All effective control arguments used by the optimizer.

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**(3), 507–554. doi:10.1111/j.14679876.2005.00510.x

See Also

[gamlss2](#), [gamlss2_control](#), [gamlss2.family](#), [gamlss2_smoothing](#)

Examples

```
## Not run: ## count response example using HarzTraffic data
data("HarzTraffic", package = "gamlss2")

## specify the model formula
f <- bikes ~ s(yday, bs = "cc") + s(temp) + s(rain) | .

## estimate model using RS (default)
m1 <- gamlss2(f, data = HarzTraffic, family = NBI, optimizer = RS)

## now with CG
m2 <- gamlss2(f, data = HarzTraffic, family = NBI, optimizer = CG)

## first 2 RS iterations and afterwards switch to CG
m3 <- gamlss2(f, data = HarzTraffic, family = NBI, CG = 3)

## End(Not run)
```

Rsq

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

Description

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

Usage

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

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

Arguments

<code>object</code>	A fitted model object.
<code>...</code>	Optionally more fitted model objects.
<code>k</code>	Numeric, the <i>penalty</i> to be used. The default <code>k = 2</code> corresponds to the classical AIC.
<code>corrected</code>	Logical, whether the corrected AIC should be used? Note that it applies only when <code>k = 2</code> .
<code>type</code>	Which definition of R-squared should be used? Possible values are "Cox Snell", "Cragg Uhler", "both", and "simple". The option "simple" computes an R-squared based on the mean; in this case <code>newdata</code> may also be supplied.
<code>newdata</code>	For <code>type = "simple"</code> , the R-squared can also be evaluated on <code>newdata</code> .

Details

The `Rsqr()` function uses the following definition of 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 fitted model under consideration. This definition is sometimes referred to as the Cox and Snell R-squared. The Nagelkerke or Cragg and Uhler definition divides the above by

$$1 - L(0)^{2/n}$$

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**

```
## Not run: ## load HarzTraffic data
data("HarzTraffic", package = "gamlss2")

## estimate negative binomial count models
m1 <- gamlss2(
  bikes ~ s(yday, bs = "cc"),
  data = HarzTraffic,
  family = NBI
)

## model with additional model term for sigma
m2 <- gamlss2(
  bikes ~ s(yday, bs = "cc") | .,
  data = HarzTraffic,
  family = NBI
)

## compare models
Rsqr(m1)
Rsqr(m1, type = "both")
Rsqr(m1, m2)
GAIC(m1, m2)
AIC(m1, m2)
BIC(m1, m2)

## plot estimated effects
plot(m2)

## End(Not run)
```

select_gamlss2

*Smooth Model Term Selection with Additional Penalties***Description**

The function `select_gamlss2()` applies an additional shrinkage penalty to all **mgcv** model terms. This can shrink some terms to zero and thereby remove them from the model.

Usage

```
## wrapper function
select_gamlss2(formula, ..., criterion = "BIC", thres = c(0.9, 0.1))

## modified RS optimizer function
```

```
sRS(x, y, specials, family, offsets,
    weights, start, xterms, sterm, control)
```

Arguments

formula	A GAM-type formula or Formula . All smooth terms of the mgcv package are supported, see also formula.gam . For <code>gamlss2.list()</code> , formula is a list of formulas .
...	Arguments passed to gamlss2 and to argument <code>control</code> in function <code>sRS()</code> . For <code>criterion = "GAIC"</code> , the penalty can be supplied via <code>K</code> .
criterion	The information criterion used to estimate the shrinkage parameters. This can also be a vector of length 2, where the first element specifies the criterion to be used during the selection step, and the second element specifies the criterion to be used during the refitting step. Possible options are "BIC", "GCV", "AIC", "AICc" and "GAIC" with user-defined penalty <code>K</code> ; the default is <code>K = 2</code> .
thres	A vector of thresholds used for model term selection. The first element controls the minimum allowed estimated degrees of freedom for a model term to enter the model. The second element specifies the minimum percentage of the total estimated predictor range required for a term to be included in the model.
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 model offsets.
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

Model terms are retained if their estimated degrees of freedom exceed `thres[1]` and the proportion of the total predictor range they cover exceeds `thres[2]`. These thresholds mimic selection based on summaries and effect plots. After the selection step, the model is refitted using only the selected terms, excluding the additional penalty applied during the initial fitting process. The additional penalty for automatic term selection is described in [gam.selection](#). This approach is experimental, so model selection should still be assessed carefully.

Value

An object of class "gamlss2".

See Also

[gamlss2](#), [new_formula](#).

Examples

```
## Not run: ## simulate data
set.seed(123)

## number of observations
n <- 1000

## covariates
k <- 100
x <- matrix(runif(n * k, -3, 3), ncol = k)
colnames(x) <- paste0("x", 1:k)
d <- as.data.frame(x)

## true effects
d$f1 <- sin(d$x1)
d$f2 <- exp(d$x2)/15 - 1
d$f3 <- -d$x3 / 3
d$f4 <- d$x4^2/5 - 1
d$f5 <- cos(d$x5)

## true parameters
mu <- with(d, f1 + f3 + f4)
sigma <- with(d, exp(1.5 + f2 + f4 + f5))

## simulate response
d$y <- rnorm(n, mean = mu, sd = sigma)

## model formula
f <- paste("y ~", paste0("s(x", 1:k, ")", collapse = "+"), "| .")
f <- as.formula(f)

## estimate model (takes some time)
m <- select_gamlss2(f, data = d, family = NO)

## plot selected estimated effects
plot(m)

## final model
new_formula(m)

## example taken from ?gam.selection
library("mgcv")
set.seed(3)
n <- 200
```

```
## simulate data
dat <- gamSim(1, n = n, scale = .15, dist = "poisson")

## spurious
dat$x4 <- runif(n, 0, 1)
dat$x5 <- runif(n, 0, 1)

## formula
f <- y ~ s(x0) + s(x1) + s(x2) + s(x3) + s(x4) + s(x5)

## estimate model
m1 <- gam(f, data = dat, family = poisson,
  select = TRUE, method = "REML")
summary(m1)
plot(m1, pages = 1)

## same with gamlss2
m2 <- select_gamlss2(f, data = dat, family = P0)
summary(m2)
plot(m2)

## final model
new_formula(m2)

## End(Not run)
```

si

Smooth Image Model Terms for GAMLSS

Description

Constructs a smooth image model term for `gamlss2`. The image coefficient surface is represented by a tensor product of marginal B-spline bases with difference penalties.

Usage

```
si(x, dim = NULL, k = c(8, 8), degree = c(3, 3),
  m = c(2, 2), sp = NULL, ...)
```

Arguments

x	A matrix-valued data-frame column containing one flattened image per row. The number of columns must equal the product of dim.
dim	Integer vector <code>c(height, width)</code> for grayscale images or <code>c(height, width, channels)</code> for multi-channel images.
k	One or two integers giving the marginal B-spline basis dimensions. The default is <code>c(8, 8)</code> .

degree	One or two non-negative integers giving the marginal B-spline degrees. The default is cubic splines, <code>c(3, 3)</code> .
m	One or two difference-penalty orders, each smaller than the corresponding value of <code>k</code> . The default is <code>c(2, 2)</code> .
sp	Optional vector of one or two non-negative smoothing parameters. If supplied, the two marginal smoothing parameters are kept fixed.
...	Further controls passed to the smooth specification.

Details

The image contribution is a linear functional of a tensor-product coefficient surface. For image $x_i(h, w)$ and coefficient surface $\beta(h, w)$, the fitted contribution is

$$f_i = \sum_h \sum_w x_i(h, w) \beta(h, w).$$

The surface is expanded in marginal B-spline bases, and the image values are projected onto that basis without explicitly constructing the full image-by-coefficient Kronecker matrix.

For multi-channel images, one coefficient surface is fitted per channel with common marginal smoothing parameters. The fitted term is centered before it is added to the distributional predictor.

The compact data representation is usually created with `dat$image <- I(matrix(images, nrow = n))`, where `images` is an `n` by height by width (and optionally channels) array.

Value

The constructor returns a smooth specification object of class `"si.smooth.spec"`. After construction by **mgcv**, the corresponding effect contains the tensor-product design and penalty matrices used for fitting and prediction.

See Also

[gamlss2](#), [special_terms](#), [smooth.construct](#)

Examples

```
## Not run:
set.seed(1328)
n <- 365*3
nr <- 30
nc <- 40
lat <- seq(40, 55, length.out = nr)
lon <- seq(-10, 20, length.out = nc)

## artificial daily temperature fields
clim <- outer(lat, lon,
  function(lat, lon) 20 - 0.35 * (lat - 40) + 0.04 * lon)
pw <- outer(lat, lon,
  function(lat, lon) exp(-((lat - 48) / 4)^2 - ((lon + 3) / 6)^2))
```

```

pe <- outer(lat, lon,
  function(lat, lon) exp(-((lat - 46) / 4)^2 - ((lon - 14) / 5)^2))
pn <- outer(as.numeric(scale(lat)), rep(1, nc))

temp <- array(NA_real_, c(n, nr, nc))
for(i in seq_len(n)) {
  temp[i, , ] <- clim +
    rnorm(1, sd = 2.5) +
    rnorm(1, sd = 3.0) * pw +
    rnorm(1, sd = 3.0) * pe +
    rnorm(1, sd = 1.0) * pn +
    matrix(rnorm(nr * nc, sd = 0.25), nr, nc)
}

## ozone also depends on humidity, wind,
## and two temperature regions
hum <- runif(n, 35, 95)
wind <- rgamma(n, shape = 4, scale = 1.2)
tw <- apply(temp[, 4:7, 2:5, drop = FALSE], 1L, mean)
te <- apply(temp[, 3:6, 10:13, drop = FALSE], 1L, mean)
f_im <- 1.8 * (te - mean(te)) - 1.2 * (tw - mean(tw))
f_hum <- -0.006 * (hum - 55)^2
f_wind <- 7 * exp(-0.5 * ((wind - 3) / 2)^2)

d <- data.frame(
  ozone = 45 + f_hum + f_wind + f_im + rnorm(n, sd = 2),
  hum = hum,
  wind = wind
)
d$temp <- I(matrix(temp, nrow = n))

## three temperature fields
zlim <- range(temp[1:3, , ])
op <- par(mfrow = c(1, 3))
for(i in 1:3) {
  image(lon, lat, t(temp[i, , ]),
    zlim = zlim, asp = 1,
    col = hcl.colors(30, "Inferno", rev = TRUE),
    xlab = "lon", ylab = "lat",
    main = paste("Temperature, day", i))
}
par(op)

m <- gamlss2(
  ozone ~ s(hum) + s(wind) +
    si(temp, dim = c(nr, nc), k = 10),
  data = d,
  family = NO
)

## plot the fitted temperature-to-ozone image effect
ss <- m$specials[["si(temp)"]]
cf <- m$fitted.specials$mu[["si(temp)"]]$coefficients

```

```

## each pixel is evaluated as a one-unit image perturbation,
## this visualizes the fitted coefficient surface,
## not a reconstructed field
P <- kronecker(ss$Bw, ss$Bh)
q <- attr(ss, "qrc")
P <- t(qr.qty(q, t(P))[(attr(ss, "nCons") + 1L):ncol(P), , drop = FALSE])
eff <- drop(P %*% cf)
image(lon, lat, t(matrix(eff, nr, nc)),
      col = hcl.colors(30, "Inferno", rev = TRUE),
      xlab = "lon", ylab = "lat", main = "Fitted si() image effect")

## predicted distributions and quantiles
nd <- d[1:5, , drop = FALSE]
predict(m, newdata = nd)
quantile(m, newdata = nd)

## End(Not run)

```

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 inverse 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. (2024) 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 (2024). "Using the Softplus Function to Construct Alternative Link Functions in Generalized Linear Models and Beyond". *Statistical Papers*, **65**(5), 3155–3180. doi:[10.1007/s0036202301509](https://doi.org/10.1007/s0036202301509)

See Also

[make.link](#), [gamlss2](#)

Examples

```
## visualization of the softplus function from Wiemann et al. (2024, 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")
```

Description

The **gamlss2** package provides an interface for special model terms whose fitting is not handled by the standard linear or smooth-term machinery. Examples include neural networks, trees, or forests. A special term is expected to provide its own fitting and prediction methods so that it can be updated within the **RS** or **CG** backfitting algorithms.

Usage

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

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

## extractor function for fitted special terms
specials(object, parameter = 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.
parameter	Character or integer selecting the distribution parameter whose 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 <code>elements = "names"</code> , the corresponding element names are extracted.
...	Arguments needed for the <code>special_fit()</code> function to facilitate the fitting of the model term, see the details. Similarly, for the <code>special_predict()</code> function, the ... argument encompasses the objects for computing predictions for the model term.

Details

To implement a new special term, define:

1. a constructor function that is used inside the model formula,
2. a corresponding `special_fit()` method,
3. and a corresponding `special_predict()` method.

The constructor function should collect the information needed for fitting, such as the model formula, the variables, and any control arguments. If the constructor has a new special name, this name must be registered via the `specials` argument of [fake_formula](#). In the example below, this is not necessary because "n" is already registered.

The following paragraphs describe the expected arguments and return values of the fitting and prediction methods.

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](#).

Only the first three arguments are required to set up a special model term; all remaining arguments are optional. The function must at least return a named list containing the component `"fitted.values"` in order to work with [RS](#) and [CG](#). In practice, useful return components are:

- `"fitted.values"`: Required. Numeric vector of fitted contributions of the special term.
- Additional fitted objects: Optional. For example, the fitted model object itself, degrees of freedom, or quantities needed for prediction.
- `"transfer"`: Optional. Information that should be passed from one backfitting iteration to the next.

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.

The function `special_predict()` should return fitted contributions for the supplied data. The minimal valid return value is a numeric vector. If `se.fit = TRUE`, the method may instead return a data frame with a column named `"fit"` and optional columns such as `"lower"` and `"upper"` when interval information is available.

See Also

[gamlss2](#), [cf](#), [ct](#), [elm](#), [fk](#), [gnet](#), [im](#), [la](#), [lin](#), [lo](#), [ms](#), [n](#), [pb](#), [re](#), [si](#), [tree](#), [user](#)

Examples

```
## note: this is a quite long example,
##       to showcase the complete workflow
##
## 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("HarzTraffic", package = "gamlss2")

## specify the model formula
f <- bikes ~ n(yday,decay=0.01) | .

## estimate model,
## set the seed for reproducibility
## note, data should be scaled!
set.seed(123)
m1 <- gamlss2(f, data = HarzTraffic, family = NBI)

```

```

## visualize estimated effects
plot(m1, which = "effects")

## plot diagnostics
plot(m1, which = "resid")

## predict quantiles
nd <- data.frame("yday" = 0:364)
pq <- quantile(m1, newdata = nd, probs = c(0.05, 0.5, 0.95))

## plot
plot(bikes ~ yday, data = HarzTraffic, pch = 19,
     col = rgb(0.1, 0.1, 0.1, alpha = 0.3))
matlines(nd$yday, pq, lwd = 2,
        lty = 1, col = 4)

## include another covariate
f <- bikes ~ n(~yday+rain,size=100,decay=0.1) | .

## estimate model
set.seed(456)
m2 <- gamlss2(f, data = HarzTraffic, family = NBI)

## plot estimated effects
plot(m2, which = "effects")

## diagnostics
plot(m2, which = "resid")

## predict using new data
n <- 50
nd <- with(HarzTraffic, expand.grid(
  "yday" = seq(0, 364, length = n),
  "rain" = seq(min(rain), max(rain), length = n)
))

## median rent R estimate
nd$fit <- quantile(m2, newdata = nd, probs = 0.5)

## visualize
library("lattice")

p1 <- wireframe(
  fit ~ yday + rain, data = nd,
  screen = list(z = 50, x = -70, y = -10),
  aspect = c(1, 0.9), drape = TRUE,
  main = "n(~yday+rain)",
  xlab = "Day of the year", ylab = "Precipitation [mm]",
  zlab = "#Bikes"
)

p2 <- levelplot(

```

```

fit ~ yday + rain, data = nd,
contour = TRUE,
main = "n(~yday+rain)",
xlab = "Day of the year",
ylab = "Precipitation [mm]"
)

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(m2, parameter = "mu", elements = "model")

## same for sigma
specials(m2, parameter = "sigma", elements = "model")

## return element names of fitted special term list
specials(m2, parameter = "sigma", elements = "names")

## End(Not run)

```

SpirometryUS

Spirometry Measurements from NHANES 2007–2012

Description

Various spirometry measurements from the National Health and Nutrition Examination Survey (NHANES) 2007–2012 along with covariates providing demographics and basic body measurements.

Usage

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

Format

A data frame containing 16596 observations on 13 variables.

fvc Numeric. Forced vital capacity (FVC) in liters, i.e., the volume of air that can forcibly be blown out after full inspiration.

fev1 Numeric. Forced expiratory volume in 1 second (FEV1) in liters, i.e., the volume of air that can forcibly be blown out in the first second, after full inspiration.

ratio Numeric. Ratio of FEV1 to FVC.

pef Numeric, peak expiratory flow (PEF) in liters per second, i.e., the maximal flow (or speed) achieved during the maximally forced expiration initiated at full inspiration.

fef Numeric. Forced expiratory flow (FEF) in liters per second, i.e., the flow (or speed) of air coming out of the lung during the middle portion (25% to 75%) of a forced expiration.

- volume** Numeric. Extrapolated volume.
- fet** Numeric. Forced expiratory time (FET) in seconds, i.e., the length of the expiration.
- gender** Factor. Binary gender information with levels female and male.
- age** Numeric. Age in years (rounded to quarters).
- weight** Numeric. Body weight in kilograms.
- height** Numeric. Body height in centimeters.
- bmi** Numeric. Body mass index in kilograms per meter-squared, rounded to 2 decimal places.
- ethnicity** Factor. Self-reported race and ethnicity information with levels white, black, mexican American, other hispanic, and other (including multi-racial).

Details

In order to establish lung function reference equations, Zavorsky (2025) studies the dependence of three spirometry measurements (FVC, FEV1, and the FEV1/FVC ratio) on age, adjusted for height and weight and separately for females and males. He intends to show that a simple normally-distributed model with (piecewise) linear mean equation and (piecewise) constant variance suffices for obtaining an adequate distributional fit from which the 5% quantile can be obtained as the so-called lower limit of normal (LLN). Actually, his comparison with GAMLSS – using flexible predictors for both mean and variance along with a Box-Cox-transformed normal distribution – shows that GAMLSS leads to a similar fit for the mean but a much better fit for the LLN.

Zavorsky's (2025) analyses are based on a data set that he derived from the National Health and Nutrition Examination Survey (NHANES) in the United States 2007–2012. From the entire available data from <https://www.cdc.gov/nchs/nhanes/> he included those observations which met or exceeded the technical acceptability of the measurements for forced expiratory volume in 1 second (FEV1) and forced vital capacity (FVC). The data are described in a short communication published in the *Data in Brief* journal and the accompanying spreadsheet in CSV format (comma-separated values) is available from *Mendeley Data*.

The data comprises observations from NHANES' "Examination Data", in particular in "Spirometry – Pre and Post-Bronchodilator" and "Body Measures", plus accompanying "Demographics Data". See the variable descriptions above for more details. Basic information about spirometry can be found for example in the Wikipedia at <https://en.wikipedia.org/wiki/Spirometry>.

Source

Zavorsky GS (2024). "Refined NHANES 2007–2012 Spirometry Dataset for the Comparison of Segmented (Piecewise) Linear Models to That of GAMLSS", *Mendeley Data*, V1. doi:10.17632/dwjykg3xww.1

References

- Zavorsky GS (2024). "A Refined Spirometry Dataset for Comparing Segmented (Piecewise) Linear Models to That of GAMLSS". *Data in Brief*, **57**, 111062. doi:10.1016/j.dib.2024.111062
- Zavorsky GS (2025). "Debunking the GAMLSS Myth: Simplicity Reigns in Pulmonary Function Diagnostics". *Respiratory Medicine*, **236**, 107836. doi:10.1016/j.rmed.2024.107836

Examples

```
data("SpirometryUS", package = "gamlss2")
summary(SpirometryUS)
```

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) and Cole and Green (CG) algorithms, as implemented in function [RS](#).

Usage

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

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

Arguments

formula	A model formula for gamlss2 .
...	Arguments passed to gamlss2 .
fixed	An optional one-sided formula, Formula object, or list of formulas specifying model terms that must always remain in the model. Terms must also occur in the search scope given by formula. For multi-parameter families, right-hand sides or list elements correspond to the distributional parameters in their usual order.
K	Numeric, the penalty for the GAIC .
strategy	Character vector of selection strategies; see the details.
keeporder	Logical. Update distributional parameters sequentially rather than search across all parameters? See the details for the order.
cores	Integer, number of workers for candidate fits. The default cores = 1L runs sequentially. Larger values use future_lapply with a temporary multisession plan, including on Windows. The caller's future plan is restored on exit. This requires the suggested packages future and future.apply .
x, y, specials, family, offsets, weights, start, xterms, sterm	Model components and starting values passed to the optimizer, as documented for RS .
control	Further control arguments as specified within the call of gamlss2 .

Details

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

Argument `strategy` selects one or more strategies:

"forward" Add the term that most improves the GAIC.

"backward" Remove the term that most improves the GAIC.

"replace" Exchange a selected term for an unselected term if this improves the GAIC.

"both" Consider both additions and removals.

"all" Combine forward, backward, and replacement steps.

Append `".linear"` to any strategy to restrict it to linear terms. These strategies run before those applied to all model terms. The default is `strategy = c("both.linear", "both")` and `keeporder = FALSE`. Selected strategies are iterated until the GAIC no longer improves.

With `keeporder = TRUE`, parameters are updated in the order specified by `gamlss2.family`, or in reverse order for backward elimination. Otherwise, selection searches across all parameters.

Terms supplied in `fixed` are included in the initial model and are never eligible for removal or replacement. The intercept is always retained and need not be included in `fixed`. Fixing a linear interaction also fixes all of its available lower-order marginal terms.

Linear selection is performed on formula terms, not individual columns of the model matrix. Thus, all columns representing a factor or an interaction with a factor are added, removed, or replaced together. Linear interactions obey strong hierarchy: an interaction can enter only after its available lower-order terms have entered, and a lower-order term cannot be removed while an included interaction depends on it.

Value

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

See Also

[new_formula](#), [gamlss2](#), [gamlss2_control](#), [RS](#)

Examples

```
## Not run: ## because of possible linear interactions,
## scale the covariates first
air <- subset(airquality, !is.na(Ozone))
air$Wind <- scale(air$Wind)
air$Temp <- scale(air$Temp)

## the formula defines the searching scope
f <- Ozone ~ Wind + Temp + Wind:Temp +
  ti(Wind) + ti(Temp) + ti(Wind, Temp) | .

## estimate a Gamma model using the stepwise algorithm
m1 <- step_gamlss2(f, data = air, family = GA, K = 2)
```

```
## always retain both linear main effects
m2 <- step_gamlss2(f, data = air, family = GA, K = 2,
  fixed = ~ Wind + Temp | .)

## same with
## m <- gamlss2(f, data = air, family = NO,
##   optimizer = stepwise, K = 2)

## show the new formula of selected model terms
new_formula(m1)

## final model summary
summary(m1)

## effect plots
plot(m1)

## diagnostic plots
plot(m1, which = "resid")

## plot GAIC
plot(m1, which = "selection")

## use forward linear, replace and backward strategy
m2 <- step_gamlss2(f, data = air, family = GA, K = 2,
  strategy = c("forward.linear", "replace", "backward"))

## plot GAIC
plot(m2, which = "selection")

## End(Not run)
```

tree

Regression Tree Model Terms

Description

Constructs an `rpart` regression-tree model term for use with `gamlss2`.

Usage

```
tree(formula, ...)
```

Arguments

<code>formula</code>	Vector or one-sided formula specifying the predictors in the tree.
<code>...</code>	Arguments passed to rpart.control .

Details

The term is refitted with [rpart](#) to the current working response and weights at each backfitting update. Its contribution is centered before being returned to the additive predictor.

Value

An object of classes "special" and "tree".

See Also

[special_terms](#)

Examples

```
## Not run:
data("mcycle", package = "MASS")

m <- gamlss2(
  accel ~ tree(times, minsplitt=5) | .,
  data = mcycle
)

plot(m)

## predicted distributions and quantiles
predict(m, newdata = mcycle[1:5, , drop = FALSE])
pq <- quantile(m, probs = c(0.025, 0.5, 0.975))

plot(accel ~ times, data = mcycle,
     ylim = range(mcycle$accel, pq))
matlines(mcycle$times, pq, lty = 1, lwd = 2, col = 4)

## End(Not run)
```

user

Reserved User Special Model Term

Description

The model term constructor `user()` in [gamlss2](#) formulae may be used to set up user-defined model terms. See the example below.

Value

The value is determined by the user-defined constructor and should inherit from `c("special", "user")`.

See Also

[special_terms](#)

Examples

```
## Not run:
## a custom weighted local smoother
user <- function(formula, span = 0.75, degree = 2) {
  if(!inherits(formula, "formula"))
    formula <- as.formula(paste("~", deparse1(substitute(formula))))
  st <- list(
    formula = formula,
    data = model.frame(formula),
    control = list(span = span, degree = degree),
    term = all.vars(formula),
    label = paste0("user(", paste(all.vars(formula), collapse = "+"), ")")
  )
  class(st) <- c("special", "user")
  st
}

special_fit.user <- function(x, z, w, control, ...) {
  fit_formula <- update(x$formula, response_z ~ .)
  dat <- x$data
  dat$response_z <- z
  dat$weights_w <- w
  fit <- loess(
    fit_formula,
    data = dat,
    weights = weights_w,
    span = x$control$span,
    degree = x$control$degree
  )
  fitted <- as.numeric(predict(fit, newdata = dat))
  shift <- mean(fitted)
  out <- list(
    model = fit,
    fitted.values = fitted - shift,
    shift = shift,
    edf = fit$trace.hat
  )
  class(out) <- "user.fitted"
  out
}

special_predict.user.fitted <- function(x, data, se.fit = FALSE, ...) {
  fit <- as.numeric(predict(x$model, newdata = data)) - x$shift
  if(se.fit) return(data.frame(fit = fit))
  fit
}

air <- subset(airquality, !is.na(Ozone))

m <- gamlss2(
  Ozone ~ user(~ Temp, span = 0.7) | s(Temp),
  data = air,
```

```

    family = GA,
    trace = FALSE
  )

plot(m)

## End(Not run)

```

vcov.gamlss2

*Joint Covariance Matrix for gamlss2 Models***Description**

Computes coefficient covariance from the joint penalized information of a fitted gamlss2 model.

Usage

```

## S3 method for class 'gamlss2'
vcov(object,
      type = c("vcov", "cor", "se", "coef"), full = FALSE,
      method = c("joint", "working", "numeric"), unconditional = FALSE, ...)

```

Arguments

object	A fitted gamlss2 model.
type	The covariance matrix, correlation matrix, standard errors, or corresponding coefficient vector to return.
full	Logical. If FALSE, return ordinary linear coefficients. Their covariance is still extracted from the complete joint inverse. If TRUE, include supported smooth coefficients.
method	Information calculation. "joint" uses the raw joint observed information, "working" uses the final family working curvature with zero cross-parameter blocks, and "numeric" uses a slow coefficient-dimensional numerical Hessian for validation.
unconditional	Logical. If TRUE, a correction for uncertainty in selected smoothness parameters is added. For JR fits this uses the fitted joint REML curvature. Other fits use a post-fit local approximation. This requires method = "joint".
...	Control arguments passed to the numerical Hessian when method = "numeric".

Details

The default covariance is the inverse of $H + P$, where H is the joint likelihood information for every supported distributional predictor and P is the complete penalty at the fitted smoothing and regularization parameters. Thus it retains covariance among linear and smooth terms and among distributional parameters. Prior observation weights and offsets in the final linear predictors are included.

Joint information is computed by differentiating the linked family scores. Mixed derivatives are checked in both orientations. If finite differences disagree, additional step sizes and Richardson extrapolation are used to check numerical convergence; persistent inconsistencies raise an error.

For monotonic [ms](#) terms, inversion is restricted to coefficient directions that preserve the fitted active constraints and term centering. This local approximation conditions on the active constraint set and does not include uncertainty about which constraints are active. Unconditional and sandwich corrections are currently unavailable for these terms.

The default covariance conditions on fitted smoothing parameters. Use `unconditional = TRUE` to include their approximate estimation uncertainty. Fixed smoothing parameters receive no correction, and model-selection uncertainty is excluded. Adaptive smooth penalties, selected ridge penalties, and non-estimable coefficients are unsupported.

Special terms that have no verifiable coefficient-linear design are rejected explicitly. Objects fitted with `light = TRUE` do not retain the final predictors and must be refitted for joint covariance. Aliased or numerically unidentified coefficient variances are returned as NA.

The ordinary `summary()` and effect-result paths use the covariance stored with each fitted model term by default, which is substantially faster for large models. Use `summary(object, method = "joint")` or `results(object, method = "joint")` to request the complete joint information. Use `summary(object, unconditional = TRUE)` for the smoothness correction. This option selects the joint method unless another method is specified explicitly. Simulation-based `predict()` calls always use the complete joint covariance. No consumer silently substitutes `method = "working"` when observed joint information is indefinite.

Value

A matrix for type = "vcov" or "cor"; a named numeric vector for type = "se" or "coef".

See Also

[gamlss2](#), [predict.gamlss2](#)

Examples

```
## Not run:
data("cars", package = "datasets")
m <- gamlss2(dist ~ s(speed) | s(speed), data = cars, family = NO)

## linear-coefficient covariance,
## marginalized over smooth coefficients.
vcov(m)

## complete linear and smooth covariance
V <- vcov(m, full = TRUE)
head(diag(V))

## include estimated smoothness uncertainty
Vc <- vcov(m, full = TRUE, unconditional = TRUE)

## End(Not run)
```

Index

- * **aplot**
 - plot.gamlss2, 85
 - quantile.gamlss2, 92
- * **categorical**
 - MN, 75
- * **datasets**
 - HarzTraffic, 48
 - SpirometryUS, 118
- * **discretization**
 - discretize_family, 17
- * **distribution**
 - discretize_family, 17
 - family_extensions, 26
 - find_family, 29
 - gamlss2.family, 38
 - Kumaraswamy, 55
 - la, 57
 - mixture_family, 71
 - MN, 75
 - OL, 82
 - prodist.gamlss2, 90
 - softplus, 111
- * **mixture**
 - mixture_family, 71
- * **models**
 - ALD, 4
 - bamlss2, 6
 - BS, 8
 - cf, 11
 - ct, 12
 - deviance.gamlss2, 15
 - elm, 19
 - family_extensions, 26
 - find_family, 29
 - fk, 31
 - gamlss2, 33
 - gamlss2.family, 38
 - gamlss2_control, 42
 - gamlss2_smoothing, 44
 - gamlss2_start, 45
 - im, 50
 - JR, 53
 - Kumaraswamy, 55
 - la, 57
 - lin, 61
 - lo, 64
 - logLik.gamlss2, 65
 - marginal_predict, 67
 - mcmc, 69
 - ms, 77
 - n, 79
 - OL, 82
 - residuals.gamlss2, 96
 - results, 100
 - RS, 101
 - select_gamlss2, 105
 - si, 108
 - softplus, 111
 - special_terms, 112
 - stepwise, 120
 - tree, 122
 - user, 123
 - vcov.gamlss2, 125
- * **multinomial**
 - MN, 75
- * **neural network**
 - n, 79
- * **neural**
 - im, 50
- * **ordinal**
 - OL, 82
- * **package**
 - gamlss2-package, 3
- * **regression**
 - ALD, 4
 - bamlss2, 6
 - BS, 8
 - deviance.gamlss2, 15

- discretize_family, 17
- elm, 19
- fake_formula, 22
- family_extensions, 26
- find_family, 29
- gamlss2, 33
- gamlss2.family, 38
- gamlss2_control, 42
- gamlss2_smoothing, 44
- gamlss2_start, 45
- Kumaraswamy, 55
- la, 57
- lin, 61
- logLik.gamlss2, 65
- marginal_predict, 67
- mcmc, 69
- mixture_family, 71
- MN, 75
- new_formula, 81
- OL, 82
- pb, 84
- predict.gamlss2, 87
- quantile.gamlss2, 92
- re, 93
- residuals.gamlss2, 96
- response_name, 99
- results, 100
- RS, 101
- Rsq, 103
- select_gamlss2, 105
- softplus, 111
- special_terms, 112
- stepwise, 120
- vcov.gamlss2, 125
- * smooth**
 - fk, 31
 - lin, 61
 - lo, 64
 - ms, 77
 - si, 108
- * utilities**
 - fake_formula, 22
 - new_formula, 81
 - pb, 84
 - re, 93
 - response_name, 99
- ALD, 4
- available_families (find_family), 29
- bamlss2, 6, 8, 9, 70
- BI, 37
- Binomial, 25
- BS, 7, 8, 70
- calibration, 9
- cdf.GAMLSS2 (GAMLSS2), 36
- cdf.gamlss2 (GAMLSS2), 36
- censor_family (family_extensions), 26
- cf, 11, 114
- cforest, 11
- CG, 9, 43–45, 54, 112, 114
- CG (RS), 101
- ct, 12, 114
- ctree, 13
- cv_gamlss2, 13
- deviance.gamlss2, 15, 66
- discretize_family, 17, 27, 28
- distribution_modifiers
 - (family_extensions), 26
- distributions3_family, 28
- distributions3_family
 - (family.distribution), 24
- elm, 19, 114
- fake_formula, 4, 22, 113
- family.distribution, 24, 91
- family.GAMLSS (family.distribution), 24
- family.gamlss.family
 - (family.distribution), 24
- family.gamlss2 (gamlss2.family), 38
- family_extensions, 18, 26
- find_family, 29
- find_gamlss2 (find_family), 29
- fit_ALD (ALD), 4
- fit_family (find_family), 29
- fk, 31, 114
- flush.console, 9, 43, 102
- format.GAMLSS2 (GAMLSS2), 36
- Formula, 6, 22, 33, 82, 99, 106
- formula, 6, 22, 33, 99, 106
- formula.gam, 33, 106
- future_lapply, 14, 120
- GAIC, 30, 120
- GAIC (Rsq), 103
- gam.selection, 106

- GAMLSS, [25, 37](#)
- `gamlss.family`, [25](#)
- GAMLSS2, [36, 91](#)
- `gamlss2`, [4–11, 13–16, 18, 20, 23, 25, 26, 28, 30, 32, 33, 38–40, 42–47, 51, 56–58, 62, 64, 66, 67, 69, 70, 72, 73, 76, 78, 80, 82–84, 86, 87, 91, 93, 94, 98–103, 105–107, 109, 112–114, 120, 121, 123, 125, 126](#)
- `gamlss2-package`, [3](#)
- `gamlss2.family`, [3, 8, 18, 24, 25, 28, 30, 33, 34, 37, 38, 56, 67, 73, 76, 83, 101–103, 106, 114, 121](#)
- `gamlss2_control`, [3, 7, 34, 42, 45–47, 54, 102, 103, 114, 121](#)
- `gamlss2_smoothing`, [43, 44, 54, 102, 103](#)
- `gamlss2_start`, [34, 43, 45](#)
- `glmnet`, [47](#)
- `gnet`, [47, 114](#)

- HarzTraffic, [48](#)
- `hessian`, [25](#)
- `hurdle_family` (`family_extensions`), [26](#)

- `im`, [50, 114](#)
- `inflate_family` (`family_extensions`), [26](#)
- `is_continuous.GAMLSS2` (`GAMLSS2`), [36](#)
- `is_continuous.gamlss2` (`GAMLSS2`), [36](#)
- `is_discrete.GAMLSS2` (`GAMLSS2`), [36](#)
- `is_discrete.gamlss2` (`GAMLSS2`), [36](#)

- JR, [43–45, 53](#)
- `jr` (JR), [53](#)

- KS (Kumaraswamy), [55](#)
- Kumaraswamy, [55](#)
- `kurtosis.GAMLSS2` (`GAMLSS2`), [36](#)
- `kurtosis.gamlss2` (`GAMLSS2`), [36](#)

- `la`, [47, 57, 114](#)
- `legend`, [10, 30](#)
- `lin`, [61, 114](#)
- `list`, [22](#)
- `lme`, [94](#)
- `lo`, [63, 114](#)
- `loess`, [64](#)
- `log_pdf`, [15](#)
- `log_pdf.GAMLSS2` (`GAMLSS2`), [36](#)
- `log_pdf.gamlss2` (`GAMLSS2`), [36](#)
- `log_pdf_metric` (`cv_gamlss2`), [13](#)
- `logit_family` (`family_extensions`), [26](#)
- `logLik`, [66](#)
- `logLik.gamlss2`, [65](#)

- `make.link`, [67, 112](#)
- `make.link2`, [25, 66](#)
- `marginal_predict`, [67, 89](#)
- `mcmc`, [5, 7–9, 69](#)
- `mean.GAMLSS2` (`GAMLSS2`), [36](#)
- `mean.gamlss2` (`GAMLSS2`), [36](#)
- `mixture_family`, [27, 28, 71](#)
- MN, [72, 75](#)
- `model.frame`, [22, 34](#)
- `model.matrix`, [19, 47, 57, 61, 62](#)
- `ms`, [77, 114, 126](#)
- `mse_metric` (`cv_gamlss2`), [13](#)
- `multisession`, [14, 120](#)

- `n`, [79, 114](#)
- `new_formula`, [81, 107, 121](#)
- `nnet`, [79, 80](#)
- NO, [37](#)
- Normal, [25](#)

- OL, [82](#)
- `ologit` (OL), [82](#)
- `optim`, [32](#)

- `par`, [57](#)
- `pb`, [45, 84, 114](#)
- `pdf.GAMLSS2` (`GAMLSS2`), [36](#)
- `pdf.gamlss2` (`GAMLSS2`), [36](#)
- `plot`, [69](#)
- `plot.gamlss2`, [51, 85, 98, 100](#)
- `plot.gamlss2.residuals` (`residuals.gamlss2`), [96](#)
- `plot_lasso` (`la`), [57](#)
- `polr`, [83](#)
- `predict`, [5, 11, 85, 89](#)
- `predict.gamlss2`, [3, 10, 51, 68, 87, 91, 97, 98, 126](#)
- `Predict.matrix.lin.effect` (`lin`), [61](#)
- `Predict.matrix.si.effect` (`si`), [108](#)
- `print.GAMLSS2` (`GAMLSS2`), [36](#)
- `print.gamlss2.family` (`gamlss2.family`), [38](#)
- `prodist`, [91](#)
- `prodist.gamlss2`, [68, 89, 90](#)

- quantile.GAMLSS2 (GAMLSS2), 36
- quantile.gamlss2, 68, 89, 92
- random.GAMLSS2 (GAMLSS2), 36
- random.gamlss2 (GAMLSS2), 36
- re, 93, 114
- residuals.gamlss2, 96
- response_name, 99
- results, 100
- ridge (lin), 61
- round_family, 18
- round_family (family_extensions), 26
- rpart, 123
- rpart.control, 122
- rqres_metric (cv_gamlss2), 13
- RS, 9, 34, 39, 43–46, 54, 94, 101, 112, 114, 120, 121
- Rsqr, 103
- s, 77, 84, 94
- score, 25
- select_gamlss2, 30, 81, 82, 105
- set.seed, 20
- shiftlog (Kumaraswamy), 55
- si, 51, 108, 114
- skewness.GAMLSS2 (GAMLSS2), 36
- skewness.gamlss2 (GAMLSS2), 36
- smooth.construct, 8, 62, 101, 106, 109
- smooth.construct.lin.smooth.spec (lin), 61
- smooth.construct.ps.smooth.spec, 84
- smooth.construct.re.smooth.spec, 94
- smooth.construct.si.smooth.spec (si), 108
- smoothCon, 34
- softplus, 38, 111
- special_fit (special_terms), 112
- special_fit.cf (cf), 11
- special_fit.ct (ct), 12
- special_fit.fk (fk), 31
- special_fit.im (im), 50
- special_fit.lo (lo), 64
- special_fit.ms (ms), 77
- special_fit.n (n), 79
- special_fit.tree (tree), 122
- special_predict (special_terms), 112
- special_predict.cf.fitted (cf), 11
- special_predict.ct.fitted (ct), 12
- special_predict.fk.fitted (fk), 31
- special_predict.im.fitted (im), 50
- special_predict.lo.fitted (lo), 64
- special_predict.ms.fitted (ms), 77
- special_predict.n.fitted (n), 79
- special_predict.tree.fitted (tree), 122
- special_terms, 12, 13, 32, 51, 64, 78, 80, 109, 112, 123
- specials, 3, 20, 33, 47, 58, 62, 94
- specials (special_terms), 112
- SpirometryUS, 118
- sRS (select_gamlss2), 105
- step_gamlss2, 81, 82
- step_gamlss2 (stepwise), 120
- stepwise, 120
- summary, 69
- support.GAMLSS2 (GAMLSS2), 36
- support.gamlss2 (GAMLSS2), 36
- terms.formula, 22
- transform_family (family_extensions), 26
- tree, 114, 122
- truncate_family (family_extensions), 26
- try, 30
- user, 114, 123
- variance.GAMLSS2 (GAMLSS2), 36
- variance.gamlss2 (GAMLSS2), 36
- vcov.gamlss2, 54, 88, 125