Package: topmodels (via r-universe)

May 29, 2026

Title Infrastructure for Forecasting and Assessment of Probabilistic Models

Version 0.3-0

Date 2025-09-20

Depends R (>= 3.6.0)

Imports graphics, grDevices, grid, stats, utils, colorspace, distributions3 (>= 0.2.1), ggplot2

Suggests bamlss (>= 1.2-1), countreg, crch (>= 1.1-1), digest, gamlss (>= 5.4-20), gamlss.dist (>= 6.1-1), MASS, PoissonBinomial, qqconf, scoringRules, tibble, tinytest, numDeriv

Description Unified infrastructure for probabilistic models and distributional regressions: Probabilistic forecasting of in-sample and out-of-sample of probabilities, densities, quantiles, and moments. Probabilistic residuals and scoring via log-score (or log-likelihood), (continuous) ranked probability score, etc. Diagnostic graphics like rootograms, PIT histograms, (randomized) quantile residual Q-Q plots, and reliagrams (reliability diagrams).

License GPL-2 | GPL-3

Encoding UTF-8

URL https://topmodels.R-Forge.R-project.org/topmodels/

Additional_repositories https://zeileis.R-universe.dev

RoxygenNote 7.3.3

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

Date/Publication 2025-09-26 18:37:17 UTC

RemoteUrl https://github.com/r-forge/topmodels

RemoteRef HEAD

RemoteSha 0325f4bb389eba1a8187fd791ea0f039c6116468

RemoteSubdir pkg/topmodels

Contents

`crps.distribution` *Method for Numerically Evaluating the CRPS of Probability Distributions*

Description

Method to the `crps` generic function from the **scoringRules** package for numerically evaluating the (continuous) ranked probability score (CRPS) of any probability **distributions3** object.

Usage

```
## S3 method for class 'distribution'
crps(
  y,
  x,
  drop = TRUE,
  elementwise = NULL,
  gridsize = 500L,
  batchsize = 10000L,
```

```
  applyfun = NULL,
  cores = NULL,
  method = NULL,
  ...
)

## S3 method for class 'Beta'
crps(y, x, drop = TRUE, elementwise = NULL, ...)

## S3 method for class 'Bernoulli'
crps(y, x, drop = TRUE, elementwise = NULL, ...)

## S3 method for class 'Binomial'
crps(y, x, drop = TRUE, elementwise = NULL, ...)

## S3 method for class 'Erlang'
crps(y, x, drop = TRUE, elementwise = NULL, ...)

## S3 method for class 'Exponential'
crps(y, x, drop = TRUE, elementwise = NULL, ...)

## S3 method for class 'Gamma'
crps(y, x, drop = TRUE, elementwise = NULL, ...)

## S3 method for class 'GEV'
crps(y, x, drop = TRUE, elementwise = NULL, ...)

## S3 method for class 'Geometric'
crps(y, x, drop = TRUE, elementwise = NULL, ...)

## S3 method for class 'Gumbel'
crps(y, x, drop = TRUE, elementwise = NULL, ...)

## S3 method for class 'HyperGeometric'
crps(y, x, drop = TRUE, elementwise = NULL, ...)

## S3 method for class 'Logistic'
crps(y, x, drop = TRUE, elementwise = NULL, ...)

## S3 method for class 'LogNormal'
crps(y, x, drop = TRUE, elementwise = NULL, ...)

## S3 method for class 'NegativeBinomial'
crps(y, x, drop = TRUE, elementwise = NULL, ...)

## S3 method for class 'Normal'
crps(y, x, drop = TRUE, elementwise = NULL, ...)
```

```
## S3 method for class 'Poisson'
crps(y, x, drop = TRUE, elementwise = NULL, ...)

## S3 method for class 'StudentsT'
crps(y, x, drop = TRUE, elementwise = NULL, ...)

## S3 method for class 'Uniform'
crps(y, x, drop = TRUE, elementwise = NULL, ...)

## S3 method for class 'XBetaX'
crps(y, x, drop = TRUE, elementwise = NULL, method = "cdf", ...)

## S3 method for class 'GAMLSS'
crps(y, x, drop = TRUE, elementwise = NULL, ...)

## S3 method for class 'BAMLSS'
crps(y, x, drop = TRUE, elementwise = NULL, ...)
```

Arguments

y	A distribution object, e.g., as created by Normal or Binomial.
x	A vector of elements whose CRPS should be determined given the distribution y.
drop	logical. Should the result be simplified to a vector if possible?
elementwise	logical. Should each distribution in y be evaluated at all elements of x (elementwise = FALSE, yielding a matrix)? Or, if y and x have the same length, should the evaluation be done element by element (elementwise = TRUE, yielding a vector)? The default of NULL means that elementwise = TRUE is used if the lengths match and otherwise elementwise = FALSE is used.
gridsize	positive size of the grid used to approximate the CDF for the numerical calculation of the CRPS.
batchsize	maximum batch size. Used to split the input into batches. Lower values reduce required memory but may increase computation time.
applyfun	an optional lapply-style function with arguments function(X, FUN, ...). It is used to compute the CRPS for each element of y. The default is to use the basic lapply function unless the cores argument is specified (see below).
cores	numeric. If set to an integer the applyfun is set to mclapply with the desired number of cores, except on Windows where parLapply with makeCluster(cores) is used.
method	character. Should the grid be set up on the observation scale and method = "cdf" be used to compute the corresponding probabilities? Or should the grid be set up on the probability scale and method = "quantile" be used to compute the corresponding observations? By default, "cdf" is used for discrete observations whose range is smaller than the gridsize and "quantile" otherwise.
...	currently not used.

Details

The (continuous) ranked probability score (CRPS) for (univariate) probability distributions can be computed based on the the object-oriented infrastructure provided by the **distributions3** package. The general `crps.distribution` method does so by using numeric integration based on the `cdf` and/or `quantile` methods (for more details see below). Additionally, if dedicated closed-form CRPS computations are provided by the **scoringRules** package for the specified distribution, then these are used because they are both computationally faster and numerically more precise. For example, the `crps` method for `Normal` objects leverages `crps_norm` rather than relying on numeric integration.

The general method for any `distribution` object uses the following strategy for numerical CRPS computation. By default (if the `method` argument is `NULL`), it distinguishes distributions whose entire support is continuous, or whose entire support is discrete, or mixed discrete-continuous distribution using `is_continuous` and `is_discrete`, respectively.

For continuous and mixed distributions, an equidistant grid of `gridsize + 5` probabilities is drawn for which the corresponding `quantiles` for each distribution `y` are calculated (including the observation `x`). The calculation of the CRPS then uses a trapezoidal approximation for the numeric integration. For discrete distributions, `gridsize` equidistant quantiles (in steps of 1) are drawn and the corresponding probabilities from the `cdf` are calculated for each distribution `y` (including the observation `x`) and the CRPS calculated using numeric integration. If the `gridsize` in steps of 1 is not sufficient to cover the required range, the method falls back to the procedure used for continuous and mixed distributions to approximate the CRPS.

If the `method` argument is set to either `"cdf"` or `"quantile"`, then the specific strategy for setting up the grid of observations and corresponding probabilities can be enforced. This can be useful if for a certain distribution class, only a `cdf` or only a `quantile` method is available or only one of them is numerically stable or computationally efficient etc.

The numeric approximation requires to set up a matrix of dimension `length(y) * (gridsize + 5)` (or `length(y) * (gridsize + 1)`) which may be very memory intensive if `length(y)` and/or `gridsize` are large. Thus, the data is split batches of (approximately) equal size, not larger than `batchsize`. Thus, the memory requirement is reduced to `batchsize * (gridsize + 5)` in each step. Hence, a smaller value of `batchsize` will reduce memory footprint but will slightly increase computation time.

The error (deviation between numerical approximation and analytic solution) has been shown to be in the order of `1e-2` for a series of distributions tested. Accuracy can be increased by increasing `gridsize` and will be lower for a smaller `gridsize`.

For parallelization of the numeric computations, a suitable `applyfun` can be provided that carries out the integration for each element of `y`. To facilitate setting up a suitable `applyfun` using the basic **parallel** package, the argument `cores` is provided for convenience. When used, `y` is split into `B` equidistant batches; at least `B = cores` batches or a multiple of `cores` with a maximum size of `batchsize`. On systems running Windows `parlapply` is used, else `mclapply`.

Value

In case of a single distribution object, either a numeric vector of `length(x)` (if `drop = TRUE`, default) or a matrix with `length(x)` columns (if `drop = FALSE`). In case of a vectorized distribution object, a matrix with `length(x)` columns containing all possible combinations.

Examples

```
set.seed(6020)

## three normal distributions X and observations x
library("distributions3")
X <- Normal(mu = c(0, 1, 2), sigma = c(2, 1, 1))
x <- c(0, 0, 1)

## evaluate crps
## using infrastructure from scoringRules (based on closed-form analytic equations)
library("scoringRules")
crps(X, x)

## using general distribution method explicitly (based on numeric integration)
crps.distribution(X, x)

## analogously for Poisson distribution
Y <- Poisson(c(0.5, 1, 2))
crps(Y, x)
crps.distribution(Y, x)
```

---

distribution_calculate_moments

Method for Numerically Evaluate Central Moments of Probability Distributions

---

Description

Method used to evaluate (approximate) the central moments (mean, variance, skewness, and kurtosis) for probability distributions for which only the cummulative distribution function (CDF) and - potentially - the quantile function is provided.

Usage

```
distribution_calculate_moments(
  x,
  what,
  gridsize = 500L,
  batchsize = 10000L,
  applyfun = NULL,
  cores = NULL,
  method = NULL,
  ...
)

## S3 method for class 'distribution'
random(x, n = 1L, drop = TRUE, ...)
```

```
## S3 method for class 'distribution'
mean(x, ...)

## S3 method for class 'distribution'
variance(x, ...)

## S3 method for class 'distribution'
skewness(x, ...)

## S3 method for class 'distribution'
kurtosis(x, ...)
```

Arguments

x	object of class c("NumericNormal", "distribution").
what	single integer, controls what the C code returns. 1L (mean) 2L (variance) 3L (skewness) 4L (kurtosis).
gridsize	integer, number of grid points used for approximation. Defaults to 500L.
batchsize	maximum batch size. Used to split the input into batches. Lower values reduce required memory but may increase computation time.
applyfun	an optional lapply-style function with arguments function(X, FUN, ...). It is used to compute the CRPS for each element of y. The default is to use the basic lapply function unless the cores argument is specified (see below).
cores	numeric. If set to an integer the applyfun is set to mclapply with the desired number of cores, except on Windows where parLapply with makeCluster(cores) is used.
method	character. Should the grid be set up on the observation scale and method = "cdf" be used to compute the corresponding probabilities? Or should the grid be set up on the probability scale and method = "quantile" be used to compute the corresponding observations? By default, "cdf" is used for discrete observations whose range is smaller than the gridsize and "quantile" otherwise.
...	mean.distribution, variance.distribution, skewness.distribution and kurtosis.distribution forward the additional arguments (...) to distribution_calculate_moment all other functions/methods ignore additional arguments.
n	Integer. Number of observations to be drawn.
drop	Logical. Should the result be simplified to a vector if possible?

Details

For discrete distributions spanning a range less than gridsize the PDF is calculated at $i = \{0, 1, 2, 3, \dots\}$ by differenciating the CDF provided which is then used to calculate the central moments.

For continuous distributions as well as discrete distributions spanning a wide range of values (larger than gridsize) a grid with gridsize intervals is created. Given the distribution provides a quantile function, this grid is specified on a (mostly) uniform grid on the quantile scale. If no quantile

function is provided, the `0.01` and `99.99` percentile are calculated approximated via the CDF, between which a uniform grid is spanned. For each interval the density is approximated using numeric forward differences

$$f(x_j) = (F(x_{i+1}) - F(x_i))/(x_{i+1} - x_i)$$

at each $x_j = (x_{i+1} + x_i) * 0.5$ with interval width of $x_{i+1} - x_i$. The densities $f(x_j)$ and interval mids x_j are used to calculate the weighted moments.

Value

A (potentially named) numeric vector of length `length(x)` with the requested central moment.

Examples

```
library("distributions3")

## ------------- custom Normal distribution (MyNormal) ----------------

## Constructor function for new 'MyNormal' distribution
MyNormal <- function(mu, sigma) {
    d <- data.frame(mu = mu, sigma = sigma)
    class(d) <- c("MyNormal", "distribution")
    return(d)
}

## Additional S3 methods required
cdf.MyNormal <- getS3method("cdf", class = "Normal")
is_discrete.MyNormal   <- getS3method("is_discrete", class = "Normal")
support.MyNormal       <- getS3method("support", class = "Normal")

## ------------- custom Poisson distribution (MyPoisson) --------------

## Custom constructor function for the 'MyPoisson' distribution
MyPoisson <- function(lambda) {
    d <- data.frame(lambda = lambda)
    class(d) <- c("MyPoisson", "distribution")
    return(d)
}

## Additional S3 methods required
cdf.MyPoisson <- getS3method("cdf", class = "Poisson")
is_discrete.MyPoisson   <- getS3method("is_discrete", class = "Poisson")
support.MyPoisson       <- getS3method("support", class = "Poisson")
```

Empirical *Create an Empirical Distribution*

Description

An empirical distribution consists of a series of N observations out of a typically unknown distribution, i.e., a random sample 'X'.

Draws n random values from the empirical ensemble with replacement.

Please see the documentation of [Empirical()] for some properties of the empircal ensemble distribution, as well as extensive examples showing to how calculate p-values and confidence intervals.

Please see the documentation of [Empirical()] for some properties of the Empirical distribution, as well as extensive examples showing to how calculate p-values and confidence intervals. 'quantile()'

TODO(RETO): Check description

Usage

```
Empirical(x)

pempirical(q, y, lower.tail = TRUE, log.p = FALSE, na.rm = TRUE)

dempirical(x, y, log = FALSE, method = "hist", ...)

qempirical(p, y, lower.tail = TRUE, log.p = FALSE, na.rm = TRUE, ...)

rempirical(n, y, na.rm = TRUE)

## S3 method for class 'Empirical'
mean(x, ...)

## S3 method for class 'Empirical'
variance(x, ...)

## S3 method for class 'Empirical'
skewness(x, type = 1L, ...)

## S3 method for class 'Empirical'
kurtosis(x, type = 3L, ...)

## S3 method for class 'Empirical'
random(x, n = 1L, drop = TRUE, ...)

## S3 method for class 'Empirical'
pdf(d, x, drop = TRUE, elementwise = NULL, ...)

## S3 method for class 'Empirical'
```

```
log_pdf(d, x, drop = TRUE, elementwise = NULL, ...)

## S3 method for class 'Empirical'
cdf(d, x, drop = TRUE, elementwise = NULL, ...)

## S3 method for class 'Empirical'
quantile(x, probs, drop = TRUE, elementwise = NULL, ...)

## S3 method for class 'Empirical'
support(d, drop = TRUE, ...)
```

Arguments

x	A vector of elements whose cumulative probabilities you would like to determine given the distribution ‘d‘.
q	vector of quantiles.
y	vector of observations of the empirical distribution with two or more non-missing finite values.
lower.tail	logical; if TRUE (default), probabilities are P[X <= x] otherwise, P[X > x]. or "density".
na.rm	logical evaluating to TRUE or FALSE indicating whether NA values should be stripped before the computation proceeds.
log, log.p	logical; if TRUE, probabilities p are given as log(p).
method	character; the method to calculate the empirical density. Either "hist" (default)
...	Currently not used.
p	vector of probabilities.
n	The number of samples to draw. Defaults to ‘1L‘.
type	integer between 1L and 3L (default) selecting one of three algorithms. See Details for more information.
drop	logical. Should the result be simplified to a vector if possible?
d	An ‘Empirical‘ object created by a call to [Empirical()].
elementwise	logical. Should each distribution in x be evaluated at all elements of probs (elementwise = FALSE, yielding a matrix)? Or, if x and probs have the same length, should the evaluation be done element by element (elementwise = TRUE, yielding a vector)? The default of NULL means that elementwise = TRUE is used if the lengths match and otherwise elementwise = FALSE is used.
probs	A vector of probabilities.

Details

The creation function [Empirical()] allows for a variety of different objects as main input x.

* Vector: Assumes that the vector contains a series of observations from one empirical distribution.

* List (named or unnamed) of vectors: Each element in the list describes one empirical distribution defined by the numeric values in each of the vectors.

* Matrix or data.frame: Each row corresponds to one empirical distribution, whilst the columns contain the individual observations.

Missing values are allowed, however, each distribution requires at least two finite observations (`-Inf`/`Inf` is replaced by `NA`).

Support: R, the set of all real numbers

Mean:

$$\bar{x} = \frac{1}{N}\sum_{i=1}^{N} x_i$$

Variance:

$$\frac{1}{N-1}\sum_{i=1}^{N}(x_i - \bar{x})$$

Skewness:
$S_1 = \sqrt{N}\frac{\sum_{i=1}^{N}(x_i-\bar{x})^3}{\sqrt{\left(\sum_{i=1}^{N}(x_i-\bar{x})^2\right)^3}}$

$S_2 = \frac{\sqrt{N*(N-1)}}{(N-2)}S_1$ (only defined for $N > 2$)

$S_3 = \sqrt{(1-\frac{1}{N})^3} * S_1$ (default)

For more details about the different types of sample skewness see Joanes and Gill (1998).

Kurtosis:
$K_1 = N * \frac{\sum_{i=1}^{N}(x_i-\bar{x})^4}{\left(\sum_{i=1}^{N}(x_i-\bar{x})^2\right)^2} - 3$

$K_2 = \frac{(N+1)*K_1+6)*(N-1)}{(N-2)*(N-3)}$ (only defined for $N > 2$)

$K_3 = \left(1-\frac{1}{N}\right)^2 * (K_1+3) - 3$ (default)

For more details about the different types of sample kurtosis see Joanes and Gill (1998).

TODO(RETO): Add empirical distribution function information (step-function 1/N)

Probability density function (p.d.f):

This function returns the same values that you get from a Z-table. Note ‘quantile()‘ is the inverse of ‘cdf()‘. Please see the documentation of [Empirical()] for some properties of the Empirical distribution, as well as extensive examples showing to how calculate p-values and confidence intervals.

Value

An ‘Empirical‘ object.

In case of a single distribution object or ‘n = 1‘, either a numeric vector of length ‘n‘ (if ‘drop = TRUE‘, default) or a ‘matrix‘ with ‘n‘ columns (if ‘drop = FALSE‘).

In case of a single distribution object, either a numeric vector of length ‘probs‘ (if ‘drop = TRUE‘, default) or a ‘matrix‘ with ‘length(x)‘ columns (if ‘drop = FALSE‘). In case of a vectorized distribution object, a matrix with ‘length(x)‘ columns containing all possible combinations.

In case of a single distribution object, either a numeric vector of length ‘probs‘ (if ‘drop = TRUE‘, default) or a ‘matrix‘ with ‘length(x)‘ columns (if ‘drop = FALSE‘). In case of a vectorized distribution object, a matrix with ‘length(x)‘ columns containing all possible combinations.

In case of a single distribution object, either a numeric vector of length ‘probs‘ (if ‘drop = TRUE‘, default) or a ‘matrix‘ with ‘length(probs)‘ columns (if ‘drop = FALSE‘). In case of a vectorized distribution object, a matrix with ‘length(probs)‘ columns containing all possible combinations.

In case of a single distribution object, a numeric vector of length 2 with the minimum and maximum value of the support (if ‘drop = TRUE‘, default) or a ‘matrix‘ with 2 columns. In case of a vectorized distribution object, a matrix with 2 columns containing all minima and maxima.

References

Joanes DN and Gill CA (1998). “Comparing Measures of Sample Skewness and Kurtosis.” *Journal of the Royal Statistical Society D*, **47**(1), 183–189. doi:10.1111/14679884.00122

Examples

```
require("distributions3")
set.seed(28)

X <- Empirical(rnorm(50))
X

mean(X)
variance(X)
skewness(X)
kurtosis(X)

random(X, 10)

pdf(X, 2)
log_pdf(X, 2)

cdf(X, 4)
quantile(X, 0.7)

### example: allowed types/classes of input arguments

## Single vector (will be coerced to numeric)
Y1  <- rnorm(3, mean = -10)
d1 <- Empirical(Y1)
d1
mean(d1)

## Unnamed list of vectors
Y2 <- list(as.character(rnorm(3, mean = -10)),
           runif(6),
           rpois(4, lambda = 15))
d2 <- Empirical(Y2)
d2
mean(d2)

## Named list of vectors
Y3 <- list("Normal"  = as.character(rnorm(3, mean = -10)),
           "Uniform" = runif(6),
```

```
           "Poisson" = rpois(4, lambda = 15))
d3 <- Empirical(Y3)
d3
mean(d3)

## Matrix or data.frame
Y4 <- matrix(rnorm(20), ncol = 5, dimnames = list(sprintf("D_%d", 1:4), sprintf("obs_%d", 1:5)))
d4 <- Empirical(Y4)
d4
d5 <- Empirical(as.data.frame(Y4))
d5
```

`geom_qqrplot` *`geom_*` and `stat_*` for Producing Quantile Residual Q-Q Plots with 'ggplot2'*

Description

Various `geom_*` and `stat_*` used within `autoplot` for producing quantile residual Q-Q plots.

Usage

```
geom_qqrplot(
  mapping = NULL,
  data = NULL,
  stat = "identity",
  position = "identity",
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE,
  ...
)

stat_qqrplot_simint(
  mapping = NULL,
  data = NULL,
  geom = "qqrplot_simint",
  position = "identity",
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE,
  ...
)

geom_qqrplot_simint(
  mapping = NULL,
  data = NULL,
```

```
  stat = "qqrplot_simint",
  position = "identity",
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE,
  ...
)

stat_qqrplot_ref(
  mapping = NULL,
  data = NULL,
  geom = "qqrplot_ref",
  position = "identity",
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE,
  detrend = FALSE,
  identity = TRUE,
  probs = c(0.25, 0.75),
  scale = c("normal", "uniform"),
  ...
)

geom_qqrplot_ref(
  mapping = NULL,
  data = NULL,
  stat = "qqrplot_ref",
  position = "identity",
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE,
  detrend = FALSE,
  identity = TRUE,
  probs = c(0.25, 0.75),
  scale = c("normal", "uniform"),
  ...
)

geom_qqrplot_confint(
  mapping = NULL,
  data = NULL,
  stat = "qqrplot_confint",
  position = "identity",
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE,
  detrend = FALSE,
  type = c("pointwise", "simultaneous", "beta", "normal", "ks", "ell"),
```

```
  level = 0.95,
  identity = TRUE,
  probs = c(0.25, 0.75),
  scale = c("normal", "uniform"),
  style = c("polygon", "line"),
  ...
)

GeomQqrplotConfint
```

Arguments

mapping	Set of aesthetic mappings created by `aes()`. If specified and `inherit.aes = TRUE` (the default), it is combined with the default mapping at the top level of the plot. You must supply `mapping` if there is no plot mapping.
data	The data to be displayed in this layer. There are three options: If `NULL`, the default, the data is inherited from the plot data as specified in the call to `ggplot()`. A `data.frame`, or other object, will override the plot data. All objects will be fortified to produce a data frame. See `fortify()` for which variables will be created. A `function` will be called with a single argument, the plot data. The return value must be a `data.frame`, and will be used as the layer data. A `function` can be created from a `formula` (e.g. `~ head(.x, 10)`).
stat	The statistical transformation to use on the data for this layer. When using a `geom_*()` function to construct a layer, the `stat` argument can be used to override the default coupling between geoms and stats. The `stat` argument accepts the following: • A `Stat` ggproto subclass, for example `StatCount`. • A string naming the stat. To give the stat as a string, strip the function name of the `stat_` prefix. For example, to use `stat_count()`, give the stat as `"count"`. • For more information and other ways to specify the stat, see the layer stat documentation.
position	A position adjustment to use on the data for this layer. This can be used in various ways, including to prevent overplotting and improving the display. The `position` argument accepts the following: • The result of calling a position function, such as `position_jitter()`. This method allows for passing extra arguments to the position. • A string naming the position adjustment. To give the position as a string, strip the function name of the `position_` prefix. For example, to use `position_jitter()`, give the position as `"jitter"`. • For more information and other ways to specify the position, see the layer position documentation.
na.rm	If `FALSE`, the default, missing values are removed with a warning. If `TRUE`, missing values are silently removed.

show.legend — logical. Should this layer be included in the legends? NA, the default, includes if any aesthetics are mapped. FALSE never includes, and TRUE always includes. It can also be a named logical vector to finely select the aesthetics to display. To include legend keys for all levels, even when no data exists, use TRUE. If NA, all levels are shown in legend, but unobserved levels are omitted.

inherit.aes — If FALSE, overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn't inherit behaviour from the default plot specification, e.g. annotation_borders().

... — Other arguments passed on to layer()'s params argument. These arguments broadly fall into one of 4 categories below. Notably, further arguments to the position argument, or aesthetics that are required can *not* be passed through .... Unknown arguments that are not part of the 4 categories below are ignored.

- Static aesthetics that are not mapped to a scale, but are at a fixed value and apply to the layer as a whole. For example, colour = "red" or linewidth = 3. The geom's documentation has an **Aesthetics** section that lists the available options. The 'required' aesthetics cannot be passed on to the params. Please note that while passing unmapped aesthetics as vectors is technically possible, the order and required length is not guaranteed to be parallel to the input data.
- When constructing a layer using a stat_*() function, the ... argument can be used to pass on parameters to the geom part of the layer. An example of this is stat_density(geom = "area", outline.type = "both"). The geom's documentation lists which parameters it can accept.
- Inversely, when constructing a layer using a geom_*() function, the ... argument can be used to pass on parameters to the stat part of the layer. An example of this is geom_area(stat = "density", adjust = 0.5). The stat's documentation lists which parameters it can accept.
- The key_glyph argument of layer() may also be passed on through .... This can be one of the functions described as key glyphs, to change the display of the layer in the legend.

geom — The geometric object to use to display the data for this layer. When using a stat_*() function to construct a layer, the geom argument can be used to override the default coupling between stats and geoms. The geom argument accepts the following:

- A Geom ggproto subclass, for example GeomPoint.
- A string naming the geom. To give the geom as a string, strip the function name of the geom_ prefix. For example, to use geom_point(), give the geom as "point".
- For more information and other ways to specify the geom, see the layer geom documentation.

detrend — logical, default FALSE. If set to TRUE the qqrplot is detrended, i.e, plotted as a wormplot.

identity — logical. Should the identity line be plotted or a theoretical line which passes through probs quantiles on the "uniform" or "normal" scale.

probs — numeric vector of length two, representing probabilities of reference line used.

scale	character. Scale on which the quantile residuals will be shown: "uniform" (default) for uniform scale or "normal" for normal scale. Used for the reference line which goes through the first and third quartile of theoretical distributions.
type	character. Method for creating the confidence intervals. There are two methods for pointwise confidence intervals: Based on the ‘"beta"‘ or ‘"normal"‘ distribution, yielding very similar results. And there are two methods for simultaneous confidence intervals: Based on the Kolmogorov-Smirnov test (‘"ks"‘) or equal local levels (‘"ell"‘), where the latter has much better properties but requires the **qqconf** package to be installed ([qqconf::get_qq_band()]). Finally, the methods ‘"pointwise"‘ and ‘"simultaneous"‘ are simply aliases for the preferred corresponding methods ‘"beta"‘ and ‘"ell"‘, respectively.
level	numeric. The confidence level required, defaults to 0.95.
style	character. Style for plotting confidence intervals. Either "polygon" (default) or "line").

Format

An object of class GeomQqrplotConfint (inherits from Geom, ggproto, gg) of length 6.

Examples

```
if (require("ggplot2")) {
  ## Fit model
  data("CrabSatellites", package = "countreg")
  m1_pois <- glm(satellites ~ width + color, data = CrabSatellites, family = poisson)
  m2_pois <- glm(satellites ~ color, data = CrabSatellites, family = poisson)

  ## Compute qqrplot
  q1 <- qqrplot(m1_pois, plot = FALSE)
  q2 <- qqrplot(m2_pois, plot = FALSE)

  d <- c(q1, q2)

  ## Get label names
  xlab <- unique(attr(d, "xlab"))
  ylab <- unique(attr(d, "ylab"))
  main <- attr(d, "main")
  main <- make.names(main, unique = TRUE)
  d$group <- factor(d$group, labels = main)

  ## Polygon CI around identity line used as reference
  gg1 <- ggplot(data = d, aes(x = expected, y = observed, na.rm = TRUE)) +
    geom_qqrplot_ref() +
    geom_qqrplot_confint(fill = "red") +
    geom_qqrplot() +
    geom_qqrplot_simint(
      aes(
        x = simint_expected,
        ymin = simint_observed_lwr,
        ymax = simint_observed_upr,
        group = group
```

```
    )
  ) +
  xlab(xlab) + ylab(ylab)

gg1
gg1 + facet_wrap(~group)

## Polygon CI around robust reference line
gg2 <- ggplot(data = d, aes(x = expected, y = observed, na.rm = TRUE)) +
  geom_qqrplot_ref(identity = FALSE, scale = attr(d, "scale")) +
  geom_qqrplot_confint(identity = FALSE, scale = attr(d, "scale"), style = "line") +
  geom_qqrplot() +
  geom_qqrplot_simint(
    aes(
      x = simint_expected,
      ymin = simint_observed_lwr,
      ymax = simint_observed_upr,
      group = group
    )
  ) +
  xlab(xlab) + ylab(ylab)

gg2
gg2 + facet_wrap(~group)

## Use different `scale`s with confidence intervals
q1 <- qqrplot(m1_pois, scale = "uniform", plot = FALSE)
q2 <- qqrplot(m2_pois, plot = FALSE)

gg3 <- ggplot(data = q1, aes(x = expected, y = observed, na.rm = TRUE)) +
  geom_qqrplot_ref() +
  geom_qqrplot_confint(fill = "red", scale = "uniform") +
  geom_qqrplot()
gg3

gg4 <- ggplot(data = q2, aes(x = expected, y = observed, na.rm = TRUE)) +
  geom_qqrplot_ref() +
  geom_qqrplot_confint(fill = "red", scale = "uniform") +
  geom_qqrplot()
gg4
}
```

`newresponse` *Extract Observed Responses from New Data*

Description

Generic function and methods for extracting response variables from new data based on fitted model objects.

Usage

```
newresponse(object, ...)

## Default S3 method:
newresponse(object, newdata, na.action = na.pass, ...)

## S3 method for class 'glm'
newresponse(object, newdata, na.action = na.pass, initialize = NULL, ...)

## S3 method for class 'distribution'
newresponse(object, newdata, ...)
```

Arguments

object	a fitted model object. For the default method this needs to needs to be formula-based so that model.frame can be used to extract the response from the original data the model was fitted to or terms can be used to set up the response on newdata.
...	further arguments passed to methods.
newdata	optionally, a data frame in which to look for variables with which to predict. If omitted, the original observations are used.
na.action	function determining how to handle missing values in newdata, by default these are preserved.
initialize	logical. Should the response variable from glm objects be initialized using the corresponding expression from the family? If NULL (the default), the initialization is only used for binomial and quasibinomial families.

Details

newresponse is a convenience function that supports functions like proscore or proresiduals which assess discrepancies between predictions/forecasts on new data and the corresponding observed response variables.

The default method takes an approach that is similar to many predict methods which rebuild the model.frame after dropping the response from the terms of a model object. However, here only the response variable is preserved and all explanatory variables are dropped. Missing values values are typically preserved (i.e., using na.pass).

If the new model.frame contains a variable "(weights)", it is preserved along with the response variable(s).

A method for distribution objects is provided which expects that newdata is essentially already the corresponding new response. Thus, it needs to be a vector (or data frame) of the same length as distribution. If it is not a data frame, yet, it is transformed to one but no further modifications are made.

Value

A data.frame (model.frame) containing the response variable (and optionally a variable with "(weights)").

See Also

terms, model.frame

Examples

```
## linear regression model
m <- lm(dist ~ speed, data = cars)

## extract response variable on data used for model fitting
newresponse(m)

## extract response variable on "new" data
newresponse(m, newdata = cars[1:3, ])
```

---

pdf.distribution *Methods for Numerically Approximating PDF and Quantile Functions*

Description

Methods to the generic pdf and quantile functions from the **distributions3** package for numerically approximating the probability density function (PDF) or the quantile function (inverse CDF) if only the cummulative distribution function (CDF) is given.

Usage

```
## S3 method for class 'distribution'
pdf(
  d,
  x,
  drop = TRUE,
  elementwise = NULL,
  log = FALSE,
  applyfun = NULL,
  cores = NULL,
  ...
)

## S3 method for class 'distribution'
quantile(
  x,
  probs,
  drop = TRUE,
  elementwise = NULL,
  lower = -1/sqrt(.Machine$double.eps),
  upper = +1/sqrt(.Machine$double.eps),
  tol = .Machine$double.eps^0.5,
```

```
  maxit = 1000,
  ...
)

## S3 method for class 'distribution'
cdf(d, x, drop = TRUE, elementwise = NULL, lower.tail = TRUE, ...)
```

Arguments

d	An object of class "distribution".
x	Either a numeric vector of probabilities to be evaluated (if pdf() is called), or an object of class "distributions" (as d) when calling the quantile() function.
drop	Logical. Should the result be simplified to a vector if possible?
elementwise	Logical. Should each distribution (in d/x) be evaluated at all elements in x (when pdf() is called) or probs (if quantile() is called)? The default NULL means that elementwise = TRUE is used if the lengths match, else elementwise is set FALSE.
log	Logical. If TRUE, probabilities are given as log(p).
applyfun	an optional lapply-style function with arguments function(X, FUN, ...). It is used to compute the CRPS for each element of y. The default is to use the basic lapply function unless the cores argument is specified (see below).
cores	numeric. If set to an integer the applyfun is set to mclapply with the desired number of cores, except on Windows where parLapply with makeCluster(cores) is used.
...	currently ignored.
probs	Numeric vector of probabilities with values in [0,1].
lower, upper	numeric. Lower and upper end points for the interval to be searched, forwarded to [stats::uniroot()].
tol	numeric. Desired accuracy for [stats::uniroot()].
maxit	Integer (default 1000L). Maximum number of iterations used when iteratively evaluating quantiles based on a pdf (discrete distributions only). If maxit is reached before the quantile has been found, an error will be thrown.
lower.tail	Logical. If TRUE (default), probabilities are $P[X \leq x]$, else $P[X \geq x]$.

Examples

```
library("distributions3")
library("ggplot2")

## ------------- custom Normal distribution (MyNormal) ----------------

## Constructor function for new 'MyNormal' distribution
MyNormal <- function(mu, sigma) {
    d <- data.frame(mu = mu, sigma = sigma)
    class(d) <- c("MyNormal", "distribution")
    return(d)
```

```
}

## Additional S3 methods required
cdf.MyNormal <- getS3method("cdf", class = "Normal")
is_discrete.MyNormal   <- getS3method("is_discrete", class = "Normal")
support.MyNormal       <- getS3method("support", class = "Normal")

## Constructing objects; three normal distributions with
## mean c(1, 2, 3) and standard deviation c(1, 2.5, 5).
## n3: Based on MyNormal where only cdf, id_discrete, and support are defined.
## N3: Analytic solution (distributions3::Normal()) with analytic
##     functions for all distribution functions (pdf, cdf, quantile)
##     as well as for the first four central moments
##     (mean, variance, skewness, kurtosis)
n3 <- MyNormal(mu = 1:3, sigma = c(1, 2.5, 5))
N3 <- Normal(mu = 1:3, sigma = c(1, 2.5, 5))

## Class 'MyNormal' knows the analytic cdf:
cdf(n3, x = 2)
identical(cdf(n3, x = 2), cdf(N3, x = 2))

## Calculating probability at x = 2
pdf(n3, x = 2) ## Numeric approximation
pdf(N3, x = 2) ## Analytic solution
pdf(n3, x = 2) - pdf(N3, x = 2) ## Pairwise differences/precision

## Calculating quantiles
probs <- c(0.0, 0.01, 0.25, 0.5, 0.75, 0.99, 1.0)
quantile(n3, probs = probs) ## Numeric approximation
quantile(N3, probs = probs) ## Analytic solution

probs2 <- seq(0.01, 0.99, by = 0.01)
qn3 <- quantile(n3, probs = probs2) ## Numeric approximation
qN3 <- quantile(N3, probs = probs2) ## Analytic solution
range(qn3 - qN3) ## Range of pairwise differences/precision

## Visual comparison
x <- seq(-3, 5, by = 0.1)

d <- data.frame(x = rep(x, times = 2),
                y = c(pdf(n3[1], x = x), pdf(N3[1], x = x)),
           solution = rep(c("approximated MyNormal", "analytic Normal"), each = length(x)))
ggplot(data = d) + geom_line(aes(x = x, y = y, col = solution, lty = solution), lwd = 1) +
    scale_color_manual(values = 1:2) +
    labs(title = "Density function (mean = 1, sigma = 1)")

probs <- seq(0.01, 0.99, by = 0.01)
d <- data.frame(x = c(quantile(n3[1], probs = probs), quantile(N3[1], probs = probs)),
                y = rep(probs, times = 2),
           solution = rep(c("approximated MyNormal", "analytic Normal"), each = length(probs)))
ggplot(data = d) + geom_line(aes(x = x, y = y, col = solution, lty = solution), lwd = 1) +
    scale_color_manual(values = 1:2) +
    labs(title = "Quantile function (mean = 3, sigma = 5)")
```

```
## Drawing random numbers
set.seed(6020)
d <- data.frame(x = c(random(n3[3], 500L), random(N3[3], 500L)),
           distribution = rep(c("approximated MyNormal", "analytic Normal"), each = 500L))
ggplot(data = d) + geom_density(aes(x = x, col = distribution)) +
    scale_color_manual(values = 1:2) +
    labs(title = "Density of random numbers (mean = 3, sigma = 5)")

## ------------- custom Poisson distribution (MyPoisson) --------------

## Custom constructor function for the 'MyPoisson' distribution
MyPoisson <- function(lambda) {
    d <- data.frame(lambda = lambda)
    class(d) <- c("MyPoisson", "distribution")
    return(d)
}

## Additional S3 methods required
cdf.MyPoisson <- getS3method("cdf", class = "Poisson")
is_discrete.MyPoisson   <- getS3method("is_discrete", class = "Poisson")
support.MyPoisson       <- getS3method("support", class = "Poisson")

## Constructing objects; three normal distributions with
## parameter lambda = c(1, 2.5, 5).
## p3: Based on MyPoisson where only cdf, id_discrete, and support are defined.
## P3: Analytic solution (distributions3::Poisson()) with analytic
##     functions for all distribution functions (pdf, cdf, quantile)
##     as well as for the first four central moments
##     (mean, variance, skewness, kurtosis)
p3 <- MyPoisson(lambda = c(1, 2.5, 5))
P3 <- Poisson(lambda = c(1, 2.5, 5))

## Class 'MyPoisson' knows the analytic cdf:
cdf(p3, x = 2)
identical(cdf(p3, x = 2), cdf(P3, x = 2))

## Calculating probability at x = 2
pdf(p3, x = 2) ## Numeric approximation
pdf(P3, x = 2) ## Analytic solution
pdf(p3, x = 2) - pdf(P3, x = 2) ## Pairwise differences/precision

## Calculating quantiles
probs <- c(0.0, 0.01, 0.25, 0.5, 0.75, 0.99, 1.0)
quantile(p3, probs = probs) ## Numeric approximation
quantile(P3, probs = probs) ## Analytic solution

probs2 <- seq(0.01, 0.99, by = 0.01)
qp3 <- quantile(p3, probs = probs2) ## Numeric approximation
qP3 <- quantile(P3, probs = probs2) ## Analytic solution
range(qp3 - qP3) ## Range of pairwise differences/precision

## Visual comparison
```

```
x <- seq(-1, 11, by = 1)
d <- data.frame(x = rep(x, times = 2),
                y = c(pdf(p3[2], x = x), pdf(P3[2], x = x)),
           solution = rep(c("approximated MyPoisson", "analytic Poisson"), each = length(x)))
ggplot(data = d, aes(x = x, y = y, fill = solution)) +
    geom_bar(stat = "identity", position = "dodge") +
    scale_fill_manual(values = c("tomato", "black")) +
    labs(title = "Density (lambda = 5.0)")

probs <- seq(0.01, 0.99, by = 0.01)
d <- data.frame(x = c(quantile(p3[2], probs = probs), quantile(P3[2], probs = probs)),
                y = rep(probs, times = 2),
           solution = rep(c("approximated MyPoisson", "analytic Poisson"), each = length(probs)))
ggplot(data = d, aes(x = x, y = y, col = solution, lty = solution)) +
    geom_line(lwd = 1) +
    scale_color_manual(values = c("tomato", "black")) +
    labs(title = "Quantile function")

## Drawing random numbers
set.seed(6020)
d <- data.frame(x = c(random(p3[2], 500L), random(p3[2], 500L)),
           distribution = rep(c("approximated MyPoisson", "analytic Poisson"), each = 500L))
aggregate(x ~ distribution, data = d, FUN = function(x) c(mean = mean(x), var = var(x)))

freq <- as.data.frame(with(d, table(x, distribution))) # Calculating frequency
ggplot(data = freq, aes(x = x, y = Freq, fill = distribution)) +
    geom_bar(stat = "identity", position = "dodge") +
    scale_fill_manual(values = 1:2) +
    labs(title = "Density of random numbers (lambda = 2.5)")
```

`pithist` *PIT Histograms for Assessing Goodness of Fit of Probability Models*

Description

Probability integral transform (PIT) histograms graphically compare empirical probabilities from fitted models with a uniform distribution. If `plot = TRUE`, the resulting object of class `"pithist"` is plotted by `plot.pithist` or `autoplot.pithist` depending on whether the package `ggplot2` is loaded, before the `"pithist"` object is returned.

Usage

```
pithist(object, ...)

## Default S3 method:
pithist(
  object,
  newdata = NULL,
```

```
  plot = TRUE,
  class = NULL,
  scale = c("uniform", "normal"),
  breaks = NULL,
  type = c("expected", "random"),
  nsim = 1L,
  delta = NULL,
  simint = NULL,
  simint_level = 0.95,
  simint_nrep = 250,
  style = c("bar", "line"),
  freq = FALSE,
  expected = TRUE,
  confint = TRUE,
  xlab = "PIT",
  ylab = if (freq) "Frequency" else "Density",
  main = NULL,
  ...
)
```

Arguments

object	an object from which probability integral transforms can be extracted using the generic function procast.
...	further graphical parameters forwarded to the plotting functions.
newdata	an optional data frame in which to look for variables with which to predict. If omitted, the original observations are used.
plot	logical or character. Should the plot or autoplot method be called to draw the computed extended reliability diagram? Logical FALSE will suppress plotting, TRUE (default) will choose the type of plot conditional if the package ggplot2 is loaded. Alternatively "base" or "ggplot2" can be specified to explicitly choose the type of plot.
class	should the invisible return value be either a data.frame or a tbl_df. Can be set to "data.frame" or "tibble" to explicitly specify the return class, or to NULL (default) in which case the return class is conditional on whether the package "tibble" is loaded.
scale	controls the scale on which the PIT residuals are computed: on the probability scale ("uniform"; default) or on the normal scale ("normal").
breaks	NULL (default) or numeric to manually specify the breaks for the rootogram intervals. A single numeric (larger 0) specifies the number of breaks to be automatically chosen, multiple numeric values are interpreted as manually specified breaks.
type	character. In case of discrete distributions, should an expected (non-normal) PIT histogram be computed according to Czado et al. (2009) ("expected"; default) or should the PIT be drawn randomly from the corresponding interval ("random")?

nsim	positive integer, defaults to 1L. Only used when type = "random"; how many simulated PITs should be drawn?
delta	NULL or numeric. The minimal difference to compute the range of probabilities corresponding to each observation to get (randomized) quantile residuals. For NULL (default), the minimal observed difference in the response divided by 5e-6 is used.
simint	NULL (default) or logical. In case of discrete distributions, should the simulation (confidence) interval due to the randomization be visualized?
simint_level	numeric, defaults to 0.95. The confidence level required for calculating the simulation (confidence) interval due to the randomization.
simint_nrep	numeric, defaults to 250. The repetition number of simulated quantiles for calculating the simulation (confidence) interval due to the randomization.
style	character specifying plotting style. For style = "bar" (default) a traditional PIT histogram is drawn, style = "line" solely plots the upper border of the bars. If single_graph = TRUE is used (see plot.pithist), line-style PIT histograms will be enforced.
freq	logical. If TRUE, the PIT histogram is represented by frequencies, the counts component of the result; if FALSE, probability densities, component density, are plotted (so that the histogram has a total area of one).
expected	logical. Should the expected values be plotted as reference?
confint	logical. Should confident intervals be drawn?
xlab, ylab, main	graphical parameters passed to plot.pithist or autoplot.pithist.

Details

PIT histograms graphically evaluate the probability integral transform (PIT), i.e., the value that the predictive CDF attains at the observation, with a uniform distribution. For a well calibrated model fit, the PIT will have a standard uniform distribution. For computation, pithist leverages the function proresiduals employing the procast generic and then essentially draws a hist.

In addition to the plot and autoplot method for pithist objects, it is also possible to combine two (or more) PIT histograms by c/rbind, which creates a set of PIT histograms that can then be plotted in one go.

Value

An object of class "pithist" inheriting from data.frame or tbl_df conditional on the argument class including the following variables:

x	histogram interval midpoints on the x-axis,
y	bottom coordinate of the histogram bars,
width	widths of the histogram bars,
confint_lwr	lower bound of the confidence interval,
confint_upr	upper bound of the confidence interval,
expected	y-coordinate of the expected curve.

Additionally, freq, xlab, ylab, main, and confint_level are stored as attributes.

References

Agresti A, Coull AB (1998). "Approximate is Better than "Exact" for Interval Estimation of Binomial Proportions." *The American Statistician*, **52**(2), 119–126. doi:10.1080/00031305.1998.10480550

Czado C, Gneiting T, Held L (2009). "Predictive Model Assessment for Count Data." *Biometrics*, **65**(4), 1254–1261. doi:10.1111/j.15410420.2009.01191.x

Dawid AP (1984). "Present Position and Potential Developments: Some Personal Views: Statistical Theory: The Prequential Approach", *Journal of the Royal Statistical Society: Series A (General)*, **147**(2), 278–292. doi:10.2307/2981683

Diebold FX, Gunther TA, Tay AS (1998). "Evaluating Density Forecasts with Applications to Financial Risk Management". *International Economic Review*, **39**(4), 863–883. doi:10.2307/2527342

Gneiting T, Balabdaoui F, Raftery AE (2007). "Probabilistic Forecasts, Calibration and Sharpness". *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*. **69**(2), 243–268. doi:10.1111/j.14679868.2007.00587.x

See Also

`plot.pithist`, `proresiduals`, `procast`

Examples

```
## speed and stopping distances of cars
m1_lm <- lm(dist ~ speed, data = cars)

## compute and plot pithist
pithist(m1_lm)

#------------------------------------------------------------------------------
## determinants for male satellites to nesting horseshoe crabs
data("CrabSatellites", package = "countreg")

## linear poisson model
m1_pois <- glm(satellites ~ width + color, data = CrabSatellites, family = poisson)
m2_pois <- glm(satellites ~ color, data = CrabSatellites, family = poisson)

## compute and plot pithist as base graphic
p1 <- pithist(m1_pois, plot = FALSE)
p2 <- pithist(m2_pois, plot = FALSE)

## plot combined pithist as "ggplot2" graphic
ggplot2::autoplot(c(p1, p2), single_graph = TRUE, style = "line", col = c(1, 2))
```

`plot.pithist` *S3 Methods for Plotting PIT Histograms*

Description

Generic plotting functions for probability integral transform (PIT) histograms of the class `"pithist"` computed by `link{pithist}`.

Usage

```
## S3 method for class 'pithist'
plot(
  x,
  single_graph = FALSE,
  style = NULL,
  freq = NULL,
  expected = TRUE,
  confint = NULL,
  confint_level = 0.95,
  confint_type = c("exact", "approximation"),
  simint = NULL,
  xlim = c(NA, NA),
  ylim = c(0, NA),
  xlab = NULL,
  ylab = NULL,
  main = NULL,
  axes = TRUE,
  box = TRUE,
  col = "black",
  border = "black",
  lwd = NULL,
  lty = 1,
  alpha_min = 0.2,
  expected_col = NULL,
  expected_lty = NULL,
  expected_lwd = 1.75,
  confint_col = NULL,
  confint_lty = 2,
  confint_lwd = 1.75,
  confint_alpha = NULL,
  simint_col = "black",
  simint_lty = 1,
  simint_lwd = 1.75,
  ...
)

## S3 method for class 'pithist'
lines(
  x,
  freq = NULL,
  expected = FALSE,
  confint = FALSE,
  confint_level = 0.95,
  confint_type = c("exact", "approximation"),
  simint = FALSE,
  col = "black",
  lwd = 2,
```

```
  lty = 1,
  expected_col = "black",
  expected_lty = 2,
  expected_lwd = 1.75,
  confint_col = "black",
  confint_lty = 1,
  confint_lwd = 1.75,
  confint_alpha = 1,
  simint_col = "black",
  simint_lty = 1,
  simint_lwd = 1.75,
  ...
)

## S3 method for class 'pithist'
autoplot(
  object,
  single_graph = FALSE,
  style = NULL,
  freq = NULL,
  expected = NULL,
  confint = NULL,
  confint_level = 0.95,
  confint_type = c("exact", "approximation"),
  simint = NULL,
  xlim = c(NA, NA),
  ylim = c(0, NA),
  xlab = NULL,
  ylab = NULL,
  main = NULL,
  legend = FALSE,
  theme = NULL,
  colour = NULL,
  fill = NULL,
  linewidth = NULL,
  linetype = NULL,
  alpha = NULL,
  expected_colour = NULL,
  expected_linewidth = 0.75,
  expected_linetype = NULL,
  expected_alpha = NA,
  confint_colour = NULL,
  confint_fill = NULL,
  confint_linewidth = 0.75,
  confint_linetype = NULL,
  confint_alpha = NULL,
  simint_colour = "black",
  simint_linewidth = 0.5,
```

```
  simint_linetype = 1,
  simint_alpha = NA,
  ...
)
```

Arguments

single_graph	logical. Should all computed extended reliability diagrams be plotted in a single graph? If yes, style must be set to "line".
style	NULL or character specifying the style of pithist. For style = "bar" a traditional PIT hisogram is drawn, for style = "line" solely the upper border line is plotted. single_graph = TRUE always results in a combined line-style PIT histogram.
freq	NULL or logical. TRUE will enforce the PIT to be represented by frequencies (counts) while FALSE will enforce densities.
expected	logical. Should the expected values be plotted as reference?
confint	NULL or logical. Should confident intervals be drawn? Either logical or as
confint_level	numeric in [0, 1]. The confidence level to be shown.
confint_type	character. Which type of confidence interval should be plotted: ‘"exact"‘ or ‘"approximation"‘. According to Agresti and Coull (1998), for interval estimation of binomial proportions an approximation can be better than exact.
simint	NULL or logical. In case of discrete distributions, should the simulation (confidence) interval due to the randomization be visualized? character string defining one of ‘"polygon"‘, ‘"line"‘ or ‘"none"‘. If freq = NULL it is taken from the object.
xlim, ylim, xlab, ylab, main, axes, box	graphical parameters.
col, border, lwd, lty, alpha_min	graphical parameters for the main part of the base plot.
simint_col, simint_lty, simint_lwd, confint_col, confint_lty, confint_lwd, confint_alpha, expected_col, expected_lty, expected_lwd	Further graphical parameters for the ‘confint‘ and ‘simint‘ line/polygon in the base plot.
...	further graphical parameters passed to the plotting function.
object, x	an object of class pithist.
legend	logical. Should a legend be added in the ggplot2 style graphic?
theme	Which ‘ggplot2‘ theme should be used. If not set, theme_bw is employed.
colour, fill, linewidth, linetype, alpha	graphical parameters for the histogram style part in the autoplot.
simint_colour, simint_linewidth, simint_linetype, simint_alpha, confint_colour, confint_fill, confint_linewidth, confint_linetype, expected_colour, expected_linewidth, expected_linetype, expected_alpha	Further graphical parameters for the ‘confint‘ and ‘simint‘ line/polygon using autoplot.

Details

PIT histograms graphically evaluate the probability integral transform (PIT), i.e., the value that the predictive CDF attains at the observation, with a uniform distribution. For a well calibrated model fit, the observation will be drawn from the predictive distribution and the PIT will have a standard uniform distribution.

PIT histograms can be rendered as ggplot2 or base R graphics by using the generics autoplot or plot. For a single base R graphically panel, lines adds an additional PIT histogram.

References

Agresti A, Coull AB (1998). "Approximate is Better than "Exact" for Interval Estimation of Binomial Proportions." *The American Statistician*, **52**(2), 119–126. doi:10.1080/00031305.1998.10480550

Czado C, Gneiting T, Held L (2009). "Predictive Model Assessment for Count Data." *Biometrics*, **65**(4), 1254–1261. doi:10.2307/2981683

Dawid AP (1984). "Present Position and Potential Developments: Some Personal Views: Statistical Theory: The Prequential Approach", *Journal of the Royal Statistical Society: Series A (General)*, **147**(2), 278–292. doi:10.2307/2981683

Diebold FX, Gunther TA, Tay AS (1998). "Evaluating Density Forecasts with Applications to Financial Risk Management". *International Economic Review*, **39**(4), 863–883. doi:10.2307/2527342

Gneiting T, Balabdaoui F, Raftery AE (2007). "Probabilistic Forecasts, Calibration and Sharpness". *Journal of the Royal Statistical Society: Series B (Methodological)*. **69**(2), 243–268. doi:10.1111/j.14679868.2007.00587.x

See Also

pithist, procast, hist

Examples

```
## speed and stopping distances of cars
m1_lm <- lm(dist ~ speed, data = cars)

## compute and plot pithist
pithist(m1_lm)

## customize colors and style
pithist(m1_lm, expected_col = "blue", lty = 2, pch = 20, style = "line")

## add separate model
if (require("crch", quietly = TRUE)) {
  m1_crch <- crch(dist ~ speed | speed, data = cars)
  #lines(pithist(m1_crch, plot = FALSE), col = 2, lty = 2, confint_col = 2) #FIXME
}

#------------------------------------------------------------------------------
if (require("crch")) {

  ## precipitation observations and forecasts for Innsbruck
  data("RainIbk", package = "crch")
```

```
  RainIbk <- sqrt(RainIbk)
  RainIbk$ensmean <- apply(RainIbk[, grep("^rainfc", names(RainIbk))], 1, mean)
  RainIbk$enssd <- apply(RainIbk[, grep("^rainfc", names(RainIbk))], 1, sd)
  RainIbk <- subset(RainIbk, enssd > 0)

  ## linear model w/ constant variance estimation
  m2_lm <- lm(rain ~ ensmean, data = RainIbk)

  ## logistic censored model
  m2_crch <- crch(rain ~ ensmean | log(enssd), data = RainIbk, left = 0, dist = "logistic")

  ## compute pithists
  pit2_lm <- pithist(m2_lm, plot = FALSE)
  pit2_crch <- pithist(m2_crch, plot = FALSE)

  ## plot in single graph with style "line"
  plot(c(pit2_lm, pit2_crch),
    col = c(1, 2), confint_col = c(1, 2), expected_col = 3,
    style = "line", single_graph = TRUE
  )
}

#------------------------------------------------------------------------------
## determinants for male satellites to nesting horseshoe crabs
data("CrabSatellites", package = "countreg")

## linear poisson model
m3_pois <- glm(satellites ~ width + color, data = CrabSatellites, family = poisson)

## compute and plot pithist as "ggplot2" graphic
pithist(m3_pois, plot = "ggplot2")
```

`plot.qqrplot` *S3 Methods for Plotting Q-Q Residuals Plots*

Description

Generic plotting functions for Q-Q residual plots for objects of class `"qqrplot"` returned by `link{qqrplot}`.

Usage

```
## S3 method for class 'qqrplot'
plot(
  x,
  single_graph = FALSE,
  detrend = NULL,
  simint = NULL,
  confint = NULL,
  confint_type = c("pointwise", "simultaneous", "beta", "normal", "ks", "ell"),
```

```
  confint_level = 0.95,
  ref = NULL,
  ref_type = NULL,
  xlim = c(NA, NA),
  ylim = c(NA, NA),
  xlab = NULL,
  ylab = NULL,
  main = NULL,
  axes = TRUE,
  box = TRUE,
  col = "black",
  pch = 19,
  simint_col = "black",
  simint_alpha = 0.2,
  confint_col = "black",
  confint_lty = 2,
  confint_lwd = 1.25,
  confint_alpha = NULL,
  ref_col = "black",
  ref_lty = 2,
  ref_lwd = 1.25,
  ...
)

## S3 method for class 'qqrplot'
points(
  x,
  detrend = NULL,
  simint = FALSE,
  col = "black",
  pch = 19,
  simint_col = "black",
  simint_alpha = 0.2,
  ...
)

## S3 method for class 'qqrplot'
autoplot(
  object,
  single_graph = FALSE,
  detrend = NULL,
  simint = NULL,
  confint = NULL,
  confint_type = c("pointwise", "simultaneous", "beta", "normal", "ks", "ell"),
  confint_level = 0.95,
  ref = NULL,
  ref_type = NULL,
  xlim = c(NA, NA),
```

```
  ylim = c(NA, NA),
  xlab = NULL,
  ylab = NULL,
  main = NULL,
  legend = FALSE,
  theme = NULL,
  alpha = NA,
  colour = "black",
  fill = NA,
  shape = 19,
  size = 2,
  stroke = 0.5,
  simint_fill = "black",
  simint_alpha = 0.2,
  confint_colour = NULL,
  confint_fill = NULL,
  confint_linewidth = NULL,
  confint_linetype = NULL,
  confint_alpha = NULL,
  ref_colour = "black",
  ref_linewidth = 0.5,
  ref_linetype = 2,
  ...
)
```

Arguments

x, object	an object of class qqrplot as returned by qqrplot.
single_graph	logical, defaults to FALSE. In case of multiple Q-Q residual plots: should all be drawn in a single graph?
detrend	logical. Should the qqrplot be detrended, i.e, plotted as a 'wormplot()'? If NULL (default) this is extracted from x/object.
simint	logical or quantile specification. Should the simint of quantiles of the randomized quantile residuals be visualized?
confint	logical or character string describing the style for plotting 'c("polygon", "line")'.
confint_type	character. Method for creating the confidence intervals. There are two methods for pointwise confidence intervals: Based on the '"beta"' or '"normal"' distribution, yielding very similar results. And there are two methods for simultaneous confidence intervals: Based on the Kolmogorov-Smirnov test ('"ks"') or equal local levels ('"ell"'), where the latter has much better properties but requires the **qqconf** package to be installed ([qqconf::get_qq_band()]). Finally, the methods '"pointwise"' and '"simultaneous"' are simply aliases for the preferred corresponding methods '"beta"' and '"ell"', respectively.
confint_level	numeric. The confidence level required, defaults to 0.95.
ref	logical. Should a reference line be plotted?
ref_type	character specifying that the '"identity"' line should be used as a reference or, alternatively, a line through the '"quartiles"' of the quantile residuals. Moreover,

also a numeric vector of length two can be used to define the probabilities of the quantiles to be used for defining the reference line. Note, that the reference is also used for detrending the quantile residuals. For uniform scales, the identity line must be used for reference (‘ref_type = "identity"‘).

`xlim`, `ylim`, `axes`, `box`
: additional graphical parameters for base plots, whereby `x` is a object of class `qqrplot`.

`xlab`, `ylab`, `main`, `...`
: graphical plotting parameters passed to `plot` or `points`, respectively.

`col`, `pch`
: graphical parameters for the main part of the base plot.

`simint_col`, `simint_alpha`, `confint_col`, `confint_lty`, `confint_lwd`, `ref_col`, `ref_lty`, `ref_lwd`
: Further graphical parameters for the ‘confint‘ and ‘simint‘ line/polygon in the base plot.

`legend`
: logical. Should a legend be added in the `ggplot2` style graphic?

`theme`
: name of the ‘ggplot2‘ theme to be used. If `theme = NULL`, the `theme_bw` is employed.

`colour`, `fill`, `alpha`, `shape`, `size`, `stroke`
: graphical parameters passed to `ggplot2` style plots.

`simint_fill`, `confint_colour`, `confint_fill`, `confint_linewidth`, `confint_linetype`, `confint_alpha`, `ref_colour`, `ref_linewidth`, `ref_linetype`
: Further graphical parameters for the ‘confint‘ and ‘simint‘ line/polygon using `autoplot`.

Details

Q-Q residuals plots draw quantile residuals (by default on the standard normal scale) against theoretical quantiles from the same distribution. Alternatively, quantile residuals can also be compared on the uniform scale (`scale = "uniform"`) using no transformation.

Q-Q residuals plots can be rendered as `ggplot2` or base R graphics by using the generics `autoplot` or `plot`. `points` (`points.qqrplot`) can be used to add Q-Q residuals to an existing base R graphics panel.

References

Dunn KP, Smyth GK (1996). “Randomized Quantile Residuals.” *Journal of Computational and Graphical Statistics*, **5**(3), 236–244. doi:10.2307/1390802

See Also

`qqrplot`, `wormplot`, `proresiduals`, `qqnorm`

Examples

```
## speed and stopping distances of cars
m1_lm <- lm(dist ~ speed, data = cars)
```

```
## compute and plot qqrplot
qqrplot(m1_lm)

## customize colors
qqrplot(m1_lm, plot = "base", ref_col = "blue", lty = 2, pch = 20)

## add separate model
if (require("crch", quietly = TRUE)) {
  m1_crch <- crch(dist ~ speed | speed, data = cars)
  points(qqrplot(m1_crch, plot = FALSE), col = 2, lty = 2, simint = 2)
}

#------------------------------------------------------------------------------
if (require("crch")) {

  ## precipitation observations and forecasts for Innsbruck
  data("RainIbk", package = "crch")
  RainIbk <- sqrt(RainIbk)
  RainIbk$ensmean <- apply(RainIbk[,grep('^rainfc',names(RainIbk))], 1, mean)
  RainIbk$enssd <- apply(RainIbk[,grep('^rainfc',names(RainIbk))], 1, sd)
  RainIbk <- subset(RainIbk, enssd > 0)

  ## linear model w/ constant variance estimation
  m2_lm <- lm(rain ~ ensmean, data = RainIbk)

  ## logistic censored model
  m2_crch <- crch(rain ~ ensmean | log(enssd), data = RainIbk, left = 0, dist = "logistic")

  ## compute qqrplots
  qq2_lm <- qqrplot(m2_lm, plot = FALSE)
  qq2_crch <- qqrplot(m2_crch, plot = FALSE)

  ## plot in single graph
  plot(c(qq2_lm, qq2_crch), col = c(1, 2), simint_col = c(1, 2), single_graph = TRUE)
}

#------------------------------------------------------------------------------
## determinants for male satellites to nesting horseshoe crabs
data("CrabSatellites", package = "countreg")

## linear poisson model
m3_pois  <- glm(satellites ~ width + color, data = CrabSatellites, family = poisson)

## compute and plot qqrplot as "ggplot2" graphic
qqrplot(m3_pois, plot = "ggplot2")
```

---

plot.reliagram *S3 Methods for a Reliagram (Extended Reliability Diagram)*

Description

Generic plotting functions for reliability diagrams of the class "reliagram" computed by link{reliagram}.

Usage

```
## S3 method for class 'reliagram'
plot(
  x,
  single_graph = FALSE,
  minimum = 0,
  confint = TRUE,
  ref = TRUE,
  xlim = c(0, 1),
  ylim = c(0, 1),
  xlab = NULL,
  ylab = NULL,
  main = NULL,
  col = "black",
  fill = adjustcolor("black", alpha.f = 0.2),
  alpha_min = 0.2,
  lwd = 2,
  pch = 19,
  lty = 1,
  type = NULL,
  add_hist = TRUE,
  add_info = TRUE,
  add_rug = TRUE,
  add_min = TRUE,
  axes = TRUE,
  box = TRUE,
  ...
)

## S3 method for class 'reliagram'
lines(
  x,
  minimum = 0,
  confint = FALSE,
  ref = FALSE,
  col = "black",
  fill = adjustcolor("black", alpha.f = 0.2),
  alpha_min = 0.2,
  lwd = 2,
  pch = 19,
  lty = 1,
  type = "b",
  ...
)
```

```
## S3 method for class 'reliagram'
autoplot(
  object,
  single_graph = FALSE,
  minimum = 0,
  confint = TRUE,
  ref = TRUE,
  xlim = c(0, 1),
  ylim = c(0, 1),
  xlab = NULL,
  ylab = NULL,
  main = NULL,
  colour = "black",
  fill = adjustcolor("black", alpha.f = 0.2),
  alpha_min = 0.2,
  size = 1,
  linewidth = 1,
  shape = 19,
  linetype = 1,
  type = NULL,
  add_hist = TRUE,
  add_info = TRUE,
  add_rug = TRUE,
  add_min = TRUE,
  legend = FALSE,
  ...
)
```

Arguments

single_graph	logical. Should all computed extended reliability diagrams be plotted in a single graph?
minimum, ref, xlim, ylim, col, fill, alpha_min, lwd, pch, lty, type, add_hist, add_info, add_rug, add_min, axes, box	additional graphical parameters for base plots, whereby x is a object of class reliagram.
confint	logical. Should confident intervals be calculated and drawn?
xlab, ylab, main	graphical parameters.
...	further graphical parameters.
object, x	an object of class reliagram.
colour, size, shape, linewidth, linetype, legend	graphical parameters passed for ggplot2 style plots, whereby object is a object of class reliagram.

Details

Reliagrams evaluate if a probability model is calibrated (reliable) by first partitioning the forecast probability for a binary event into a certain number of bins and then plotting (within each bin) the averaged forecast probability against the observered/empirical relative frequency.

For continous probability forecasts, reliability diagrams can be plotted either for a pre-specified threshold or for a specific quantile probability of the response values.

Reliagrams can be rendered as ggplot2 or base R graphics by using the generics `autoplot` or `plot`. For a single base R graphically panel, `points` adds an additional reliagram.

References

Wilks DS (2011) *Statistical Methods in the Atmospheric Sciences*, 3rd ed., Academic Press, 704 pp.

See Also

link{reliagram}, `procast`

Examples

```
## speed and stopping distances of cars
m1_lm <- lm(dist ~ speed, data = cars)

## compute and plot reliagram
reliagram(m1_lm)

## customize colors
reliagram(m1_lm, ref = "blue", lty = 2, pch = 20)

## add separate model
if (require("crch", quietly = TRUE)) {
  m1_crch <- crch(dist ~ speed | speed, data = cars)
  lines(reliagram(m1_crch, plot = FALSE), col = 2, lty = 2, confint = 2)
}

#------------------------------------------------------------------------------
if (require("crch")) {

  ## precipitation observations and forecasts for Innsbruck
  data("RainIbk", package = "crch")
  RainIbk <- sqrt(RainIbk)
  RainIbk$ensmean <- apply(RainIbk[,grep('^rainfc',names(RainIbk))], 1, mean)
  RainIbk$enssd <- apply(RainIbk[,grep('^rainfc',names(RainIbk))], 1, sd)
  RainIbk <- subset(RainIbk, enssd > 0)

  ## linear model w/ constant variance estimation
  m2_lm <- lm(rain ~ ensmean, data = RainIbk)

  ## logistic censored model
  m2_crch <- crch(rain ~ ensmean | log(enssd), data = RainIbk, left = 0, dist = "logistic")

  ## compute reliagrams
```

```
  rel2_lm <- reliagram(m2_lm, plot = FALSE)
  rel2_crch <- reliagram(m2_crch, plot = FALSE)

  ## plot in single graph
 plot(c(rel2_lm, rel2_crch), col = c(1, 2), confint = c(1, 2), ref = 3, single_graph = TRUE)
}

#-------------------------------------------------------------------------------
## determinants for male satellites to nesting horseshoe crabs
data("CrabSatellites", package = "countreg")

## linear poisson model
m3_pois  <- glm(satellites ~ width + color, data = CrabSatellites, family = poisson)

## compute and plot reliagram as "ggplot2" graphic
reliagram(m3_pois, plot = "ggplot2")
```

---

plot.rootogram *S3 Methods for Plotting Rootograms*

Description

Generic plotting functions for rootograms of the class "rootogram" computed by link{rootogram}.

Usage

```
## S3 method for class 'rootogram'
plot(
  x,
  style = NULL,
  scale = NULL,
  expected = NULL,
  ref = NULL,
  confint = NULL,
  confint_level = 0.95,
  confint_type = c("tukey", "pointwise", "simultaneous"),
  confint_nrep = 1000,
  xlim = c(NA, NA),
  ylim = c(NA, NA),
  xlab = NULL,
  ylab = NULL,
  main = NULL,
  axes = TRUE,
  box = FALSE,
  col = "darkgray",
  border = "black",
  lwd = 1,
```

```
  lty = 1,
  alpha_min = 0.8,
  expected_col = 2,
  expected_pch = 19,
  expected_lty = 1,
  expected_lwd = 2,
  confint_col = "black",
  confint_lty = 2,
  confint_lwd = 1.75,
  ref_col = "black",
  ref_lty = 1,
  ref_lwd = 1.25,
  ...
)

## S3 method for class 'rootogram'
autoplot(
  object,
  style = NULL,
  scale = NULL,
  expected = NULL,
  ref = NULL,
  confint = NULL,
  confint_level = 0.95,
  confint_type = c("tukey", "pointwise", "simultaneous"),
  confint_nrep = 1000,
  xlim = c(NA, NA),
  ylim = c(NA, NA),
  xlab = NULL,
  ylab = NULL,
  main = NULL,
  legend = FALSE,
  theme = NULL,
  colour = "black",
  fill = "darkgray",
  linewidth = 0.5,
  linetype = 1,
  alpha = NA,
  expected_colour = 2,
  expected_linewidth = 1,
  expected_linetype = 1,
  expected_size = expected_linewidth * 2,
  expected_alpha = 1,
  expected_fill = NA,
  expected_stroke = 0.5,
  expected_shape = 19,
  confint_colour = "black",
  confint_linewidth = 0.5,
```

```
  confint_linetype = 2,
  confint_alpha = NA,
  ref_colour = "black",
  ref_linewidth = 0.5,
  ref_linetype = 1,
  ref_alpha = NA,
  ...
)
```

Arguments

x, object	an object of class rootogram.
style	character specifying the syle of rootogram.
scale	character specifying whether raw frequencies or their square roots (default) should be drawn.
expected	Should the expected (fitted) frequencies be plotted?
ref	logical. Should a reference line be plotted?
confint	logical. Should confident intervals be drawn?
confint_level	numeric. The confidence level required.
confint_type	character. Should "tukey", "pointwise", or "simultaneous" confidence intervals be visualized?
confint_nrep	numeric. The repetition number of simulation for computing the confidence intervals.
xlim, ylim, xlab, ylab, main, axes, box	graphical parameters.
col, border, lwd, lty, alpha_min	graphical parameters for the histogram style part of the base plot.
expected_col, expected_pch, expected_lty, expected_lwd, ref_col, ref_lty, ref_lwd, expected_colour, expected_linewidth, expected_linetype, expected_size, expected_alpha, expected_fill, expected_stroke, expected_shape, ref_colour, ref_linewidth, ref_linetype, ref_alpha, confint_col, confint_lty, confint_lwd, confint_colour, confint_linewidth, confint_linetype, confint_alpha	Further graphical parameters for the 'expected' and 'ref' line using either autoplot or plot.
...	further graphical parameters passed to the plotting function.
legend	logical. Should a legend be added in the ggplot2 style graphic?
theme	Which 'ggplot2' theme should be used. If not set, theme_bw is employed.
colour, fill, linewidth, linetype, alpha	graphical parameters for the histogram style part in the autoplot.

Details

Rootograms graphically compare (square roots) of empirical frequencies with expected (fitted) frequencies from a probability model. For the observed distribution the histogram is drawn on a square

root scale (hence the name) and superimposed with a line for the expected frequencies. The histogram can be `"standing"` on the x-axis (as usual), or `"hanging"` from the expected (fitted) curve, or a `"suspended"` histogram of deviations can be drawn.

Rootograms are associated with the work of John W. Tukey (see Tukey 1977) and were originally proposed for assessing the goodness of fit of univariate distributions and extended by Kleiber and Zeileis (2016) to regression setups.

As the expected distribution is typically a sum of different conditional distributions in regression models, the `"pointwise"` confidence intervals for each bin can be computed from mid-quantiles of a Poisson-Binomial distribution (Wilson and Einbeck 2021). Corresponding `"simultaneous"` confidence intervals for all bins can be obtained via simulation from the Poisson-Binomial distributions. As the pointwise confidence intervals are typically not substantially different from the warning limits of Tukey (1972, p. 61), set at +/- 1, these `"tukey"` intervals are used by default.

Note that for computing the exact `"pointwise"` intervals from the Poisson-Binomial distribution, the **PoissonBinomial** needs to be installed. Otherwise, a warning is issueed and a normal approximation is used.

References

Kleiber C, Zeileis A (2016). "Visualizing Count Data Regressions Using Rootograms." *The American Statistician*, **70**(3), 296–303. doi:10.1080/00031305.2016.1173590

Tukey JW (1972), "Some Graphic and Semigraphic Displays," in *Statistical Papers in Honor of George W. Snedecor,* pp.293–316. Bancroft TA (Ed.). Iowa State University Press, Ames. Reprinted in William S. Cleveland (Ed.) (1988). *The Collected Works of John W. Tukey, Volume V. Graphics: 1965–1985,* Wadsworth & Brooks/Cole, Pacific Grove.

Tukey JW (1977). *Exploratory Data Analysis*. Addison-Wesley, Reading.

Wilson P, Einbeck J (2021). "A Graphical Tool for Assessing the Suitability of a Count Regression Model", *Austrian Journal of Statistics*, **50**(1), 1–23. doi:10.17713/ajs.v50i1.921

See Also

`rootogram`, `procast`

Examples

```
## speed and stopping distances of cars
m1_lm <- lm(dist ~ speed, data = cars)

## compute and plot rootogram
rootogram(m1_lm)

## customize colors
rootogram(m1_lm, ref_col = "blue", lty = 2, pch = 20)

#--------------------------------------------------------------------------------
if (require("crch")) {

  ## precipitation observations and forecasts for Innsbruck
  data("RainIbk", package = "crch")
```

```
  RainIbk <- sqrt(RainIbk)
  RainIbk$ensmean <- apply(RainIbk[, grep("^rainfc", names(RainIbk))], 1, mean)
  RainIbk$enssd <- apply(RainIbk[, grep("^rainfc", names(RainIbk))], 1, sd)
  RainIbk <- subset(RainIbk, enssd > 0)

  ## linear model w/ constant variance estimation
  m2_lm <- lm(rain ~ ensmean, data = RainIbk)

  ## logistic censored model
  m2_crch <- crch(rain ~ ensmean | log(enssd), data = RainIbk, left = 0, dist = "logistic")

  ### compute rootograms FIXME
  #r2_lm <- rootogram(m2_lm, plot = FALSE)
  #r2_crch <- rootogram(m2_crch, plot = FALSE)

  ### plot in single graph
  #plot(c(r2_lm, r2_crch), col = c(1, 2))
}

#------------------------------------------------------------------------------
## determinants for male satellites to nesting horseshoe crabs
data("CrabSatellites", package = "countreg")

## linear poisson model
m3_pois <- glm(satellites ~ width + color, data = CrabSatellites, family = poisson)

## compute and plot rootogram as "ggplot2" graphic
rootogram(m3_pois, plot = "ggplot2")

#------------------------------------------------------------------------------
## artificial data from negative binomial (mu = 3, theta = 2)
## and Poisson (mu = 3) distribution
set.seed(1090)
y <- rnbinom(100, mu = 3, size = 2)
x <- rpois(100, lambda = 3)

## glm method: fitted values via glm()
m4_pois <- glm(y ~ x, family = poisson)

## correctly specified Poisson model fit
par(mfrow = c(1, 3))
r4a_pois <- rootogram(m4_pois, style = "standing", ylim = c(-2.2, 4.8), main = "Standing")
r4b_pois <- rootogram(m4_pois, style = "hanging", ylim = c(-2.2, 4.8), main = "Hanging")
r4c_pois <- rootogram(m4_pois, style = "suspended", ylim = c(-2.2, 4.8), main = "Suspended")
par(mfrow = c(1, 1))
```

---

procast *Procast: Probabilistic Forecasting*

Description

Generic function and methods for computing various kinds of probabilistic forecasts from (regression) models.

Usage

```
procast(
  object,
  newdata = NULL,
  na.action = na.pass,
  type = "distribution",
  at = 0.5,
  drop = FALSE,
  ...
)

## Default S3 method:
procast(
  object,
  newdata = NULL,
  na.action = na.pass,
 type = c("distribution", "mean", "variance", "quantile", "probability", "density",
    "loglikelihood", "parameters", "kurtosis", "skewness"),
  at = 0.5,
  drop = FALSE,
  ...
)

## S3 method for class 'lm'
procast(
  object,
  newdata = NULL,
  na.action = na.pass,
  type = "distribution",
  at = 0.5,
  drop = FALSE,
  ...,
  sigma = "ML"
)

## S3 method for class 'glm'
procast(
  object,
  newdata = NULL,
  na.action = na.pass,
  type = "distribution",
  at = 0.5,
  drop = FALSE,
```

```
  ...,
  dispersion = NULL
)

## S3 method for class 'bamlss'
procast(
  object,
  newdata = NULL,
  na.action = na.pass,
  type = "distribution",
  at = 0.5,
  drop = FALSE,
  ...,
  distributions3 = FALSE
)

## S3 method for class 'disttree'
procast(
  object,
  newdata = NULL,
  na.action = na.pass,
  type = "distribution",
  at = 0.5,
  drop = FALSE,
  ...,
  distributions3 = FALSE
)
```

Arguments

object	a fitted model object. For the default method this needs to have a prodist method (or object can inherit from distribution directly).
newdata	optionally, a data frame in which to look for variables with which to predict. If omitted, the original observations are used.
na.action	function determining what should be done with missing values in newdata. The default is to employ NA.
type	character specifying the type of probabilistic forecast to compute. The default is to return a "distribution" object (using the infrastructure from **distributions3**). Alternatively, just the "parameters" of the distribution can be computed or the corresponding moments: "mean", "variance", "skewness", "kurtosis". Finally, standard functions for the distribution can be evaluated (at argument at, see below), namely the "density" (or equivalently "pdf" or "pmf"), the "log_likelihood" (or equivalently "log_pdf"), the "quantile" function, or the cumulative "probability" (or equivalently "cdf").
at	numeric vector at which the forecasts should be evaluated if type specifies a function that takes an additional argument.
drop	logical. Should forecasts be returned in a data frame (default) or (if possible) dropped to a vector, see return value description below.

`...` further parameters passed to methods. In particular, this includes the logical argument `elementwise = NULL`. Should each element of distribution only be evaluated at the corresponding element of `at` (`elementwise = TRUE`) or at all elements in `at` (`elementwise = FALSE`). Elementwise evaluation is only possible if the number of observations is the same as the length of `at` and in that case a vector of the same length is returned. Otherwise a matrix is returned. The default is to use `elementwise = TRUE` if possible, and otherwise `elementwise = FALSE`.

`sigma` character or numeric or `NULL`. Specification of the standard deviation `sigma` to be used for the `Normal` distribution in the `lm` method. The default `"ML"` (or equivalently `"MLE"` or `NULL`) uses the maximum likelihood estimate based on the residual sum of squares divided by the number of observations, n. Alternatively, `sigma = "OLS"` uses the least-squares estimate (divided by the residual degrees of freedom, n - k). Finally, a concrete numeric value can also be specified in `sigma`.

`dispersion` character or numeric or `NULL`. Specification of the dispersion parameter in the `glm` method. The default `NULL` (or equivalently `"deviance"`) is to use the `deviance` divided by the number of observations, n. Alternatively, `dispersion = "Chisquared"` uses the Chi-squared statistic divided by the residual degrees of freedom, n - k. Finally, a concrete numeric value can also be specified in `dispersion`.

`distributions3` logical. If a dedicated **distributions3** object is available (e.g., such as `Normal`) and uses the same parameterization, should this be used instead of the general `disttree` distribution?

Details

The function `procast` provides a unified framework for probabilistic forcasting (or procasting, for short) based on probabilistic (regression) models, also known as distributional regression approaches. Typical types of predictions include quantiles, probabilities, (conditional) expectations, variances, and (log-)densities. Internally, `procast` methods typically compute the predicted parameters for each observation and then compute the desired outcome for the distributions with the respective parameters.

Some quantities, e.g., the moments of the distribution (like mean or variance), can be computed directly from the predicted parameters of the distribution while others require an additional argument `at` which the distribution is evaluated (e.g., the probability of a quantile or an observation of the response).

The default `procast` method leverages the S3 classes and methods for probability distributions from the **distributions3** package. In a first step the predicted probability distribution object is obtained and, by default (`type = "distribution"`), returned in order to reflect the distributional nature of the forecast. For all other `type`s (e.g., `"mean"`, `"quantile"`, or `"density"`), the corresponding extractor methods (e.g., `mean`, `quantile`, or `pdf`) are used to compute the desired quantity from the distribution objects. The examples provide some worked illustrations.

Package authors or users, who want to enable `procast` for new types of model objects, only need to provide a suitable `prodist` extractor for the predicted probability distribution. Then the default `procast` works out of the box. However, if the **distributions3** package does not support the necessary probability distribution, then it may also be necessary to implement a new distribution objects, see `apply_dpqr`.

Value

Either a data.frame of predictions with the same number of rows as the newdata (or the original observations if that is NULL). If drop = TRUE predictions with just a single column are simplified to a vector and predictions with multiple columns to a matrix.

Examples

```
## load packages
library("topmodels")
library("distributions3")

## Poisson regression model for FIFA 2018 data:
## number of goals scored by each team in each game, explained by
## predicted ability difference of the competing teams
data("FIFA2018", package = "distributions3")
m <- glm(goals ~ difference, data = FIFA2018, family = poisson)

## predicted probability distributions for all matches (in sample)
head(procast(m))
head(procast(m, drop = TRUE))

## procasts for new data
## much lower, equal, and much higher ability than opponent
nd <- data.frame(difference = c(-1, 0, 1))

## predicted goal distribution object
goals <- procast(m, newdata = nd, drop = TRUE)
goals

## predicted densities/probabilities for scoring 0, 1, ..., 5 goals
procast(m, newdata = nd, type = "density", at = 0:5)
## by hand
pdf(goals, 0:5)

## means and medians
procast(m, newdata = nd, type = "mean")
procast(m, newdata = nd, type = "quantile", at = 0.5)
## by hand
mean(goals)
quantile(goals, 0.5)

## evaluate procast elementwise or for all possible combinations
## of distributions from 'nd' and observations in 'at'
procast(m, newdata = nd, type = "probability", at = 1:3, elementwise = TRUE)
procast(m, newdata = nd, type = "probability", at = 1:3, elementwise = FALSE)

## compute in-sample log-likelihood sum via procast
sum(procast(m, type = "density", at = FIFA2018$goals, log = TRUE))
logLik(m)
```

promodel *Predictions and Residuals Dispatch for Probabilistic Models*

Description

The function promodel is a wrapper for dispatching the base predict and residuals methods to the procast and proresiduals functions for probabilistic forecasts and probabilistic residuals, respectively.

Usage

```
promodel(object)

## S3 method for class 'promodel'
residuals(object, ...)

## S3 method for class 'promodel'
predict(object, ...)
```

Arguments

object	a fitted model object for which procast and/or proresiduals work.
...	further arguments passed on to procast or proresiduals, respectively.

Details

The default methods for procast and proresiduals in this package make a wide range of different probabilistic forecasts and probabilistic residuals available for many fitted model object classes. However, it may sometimes be useful to call these flexible methods via the base predict and residuals methods. For example, this may be useful in combination with other packages that rely on the base functions such as **marginaleffects**.

Therefore, the promodel wrapper function simply adds an additional class "promodel" (probabilistic model) to the original class of an object. Then the methods for predict and residuals then strip off this class again before calling procast and proresiduals, respectively.

Examples

```
## Poisson regression model for FIFA 2018 data:
## number of goals scored by each team in each game, explained by
## predicted ability difference of the competing teams
data("FIFA2018", package = "distributions3")
m <- glm(goals ~ difference, data = FIFA2018, family = poisson)

## prediction using a new data set (final of the tournament)
final <- tail(FIFA2018, 2)

## base predict method computes linear predictor on link scale (here in logs)
predict(m, newdata = final)
```

```
## procast-based method computes distribution object by default
pm <- promodel(m)
predict(pm, newdata = final)

## all other procast types are available as well
predict(pm, newdata = final, type = "density", at = 0:4)
predict(pm, newdata = final, type = "cdf", at = 0:4)

## the base residuals method defaults to deviance residuals
## but the proresiduals-based method defaults to quantile residuals
head(residuals(m))
head(residuals(pm))
```

`proresiduals` *Residuals for Probabilistic Regression Models*

Description

Generic function and default method for (randomized) quantile residuals, PIT, Pearson, and raw response residuals based on **distributions3** support.

Usage

```
proresiduals(object, ...)

## Default S3 method:
proresiduals(
  object,
  newdata = NULL,
  type = c("quantile", "pit", "pearson", "response"),
  nsim = NULL,
  prob = NULL,
  delta = NULL,
  ...
)
```

Arguments

`object`	an object for which a `newresponse` and a `prodist` method is available.
`...`	further parameters passed to methods.
`newdata`	optionally, a data frame in which to look for variables with which to predict. If omitted, the original observations are used.
`type`	character indicating whether quantile (default), PIT, Pearson, or raw response residuals should be computed.
`nsim`	integer. The number of randomly simulated residuals of `type = "quantile"` or `"pit"`. By default one simulation is returned.

prob — numeric. Instead of simulating the probabilities (between 0 and 1) for type = "quantile" or "pit", a vector of probabilities can be specified, e.g., prob = 0.5 corresponding to mid-quantile residuals.

delta — numeric. The minimal difference to compute the range of proabilities corresponding to each observation according to get (randomized) "quantile" or "pit" residuals. For NULL, the minimal observed difference in the resonse divided by 5e-6 is used. Ignored for continuous distributions.

Details

The new generic function proresiduals comes with a powerful default method that is based on the following idea: newresponse and prodist can be used to extract the observed response and expected distribution for it, respectively. For all model classes that have methods for these two generic functions, proresiduals can compute a range of different types of residuals.

The simplest definition of residuals are the so-called "response" residuals which simply compute the difference between the observations and the expected means. The "pearson" residuals additionally standardize these residuals by the square root of the expected variance. Thus, these residuals are based only on the first and on the first two moments, respectively.

To assess the entire distribution and not just the first moments, there are also residuals based on the probability integral transform (PIT). For regression models with a continuous response distribution, "pit" residuals (see Warton 2007) are simply the expected cumulative distribution (CDF) evaluated at the observations (Dawid, 1984). For discrete distributions, a uniform random value is drawn from the range of probabilities between the CDF at the observation and the supremum of the CDF to the left of it. If the model fits well the PIT residuals should be uniformly distributed.

In order to obtain normally distributed residuals for well-fitting models (like often desired in linear regression models), "quantile" residuals, proposed by Dunn and Smyth (1996), additionally transform the PIT residuals by the standard normal quantile function.

As quantile residuals and PIT residuals are subject to randomness for discrete distributions (and also for mixed discrete-continuous distributions), it is sometimes useful to explore the extent of the random variation. This can be done either by obtaining multiple replications (via nsim) or by computing fixed quantiles of each probability interval such as prob = 0.5 (corresponding to mid-quantile residuals, see Feng et al. 2020). Another common setting is prob = c(0, 1) yielding the range of possible residuals.

Value

A vector or matrix of residuals. A matrix of residuals is returned if more than one replication of quantile or PIT residuals is computed, i.e., if either random > 1 or random = FALSE and length(prob) > 1.

References

Dawid AP (1984). "Present Position and Potential Developments: Some Personal Views: Statistical Theory: The Prequential Approach." *Journal of the Royal Statistical Society A*, **147**(2), 278–292. doi:10.2307/2981683.

Dunn KP, Smyth GK (1996). "Randomized Quantile Residuals." *Journal of Computational and Graphical Statistics*, **5**(3), 236–244. doi:10.2307/1390802

Feng C, Li L, Sadeghpour A (2020). "A Comparison of Residual Diagnosis Tools for Diagnosing Regression Models for Count Data" *BMC Medical Research Methodology*, **20**(175), 1–21. doi:10.1186/s12874020010552

Warton DI, Thibaut L, Wang YA (2017) "The PIT-Trap – A 'Model-Free' Bootstrap Procedure for Inference about Regression Models with Discrete, Multivariate Responses". *PLOS ONE*, **12**(7), 1–18. doi:10.1371/journal.pone.0181790.

See Also

qnorm, qqrplot

Examples

```
## Poisson GLM for FIFA 2018 data
data("FIFA2018", package = "distributions3")
m <- glm(goals ~ difference, data = FIFA2018, family = poisson)

## random quantile residuals (on original data)
proresiduals(m)

## various flavors of residuals on small new data
nd <- data.frame(goals = c(1, 1, 1), difference = c(-1, 0, 1))

## quantile residuals: random (1 sample), random (5 samples), mid-quantile (non-random)
proresiduals(m, newdata = nd, type = "quantile")
proresiduals(m, newdata = nd, type = "quantile", nsim = 5)
proresiduals(m, newdata = nd, type = "quantile", prob = 0.5)

## PIT residuals (without transformation to normal): random vs. minimum/maximum quantile
proresiduals(m, newdata = nd, type = "pit", nsim = 5)
proresiduals(m, newdata = nd, type = "pit", prob = c(0, 1))

## raw response residuals (observation - expected mean)
proresiduals(m, newdata = nd, type = "response")

## standardized Pearson residuals (response residuals divided by standard deviation)
proresiduals(m, newdata = nd, type = "pearson")

## compute residuals by manually obtaining distribution and response
## proresiduals(procast(m, newdata = nd, drop = TRUE), nd$goals)
```

---

proscore *Scoring Probabilistic Forecasts*

Description

Generic function and default method for computing various kinds of scores for fitted or predicted probability distributions from (regression) models.

Usage

```
proscore(object, newdata = NULL, ...)

## Default S3 method:
proscore(
  object,
  newdata = NULL,
  na.action = na.pass,
  type = c("logs", "crps"),
  aggregate = TRUE,
  drop = FALSE,
  ...
)
```

Arguments

object	a fitted model object. For the default method this needs to have a prodist and a newresponse method.
newdata	optionally, a data frame in which to look for variables with which to predict and from which to obtain the response variable. If omitted, the original observations are used.
...	further parameters passed to the aggregate function (if any).
na.action	function determining what should be done with missing values in newdata. The default is to employ NA.
type	character specifying the type of score to compute. Avaible types: "logs" (or equivalently "log-score"), "loglikelihood" (or equivalently "log_pdf"), "CRPS" (or equivalently "RPS"), "MAE", "MSE", "DSS" (or equivalently "Dawid-Sebastiani"). Upper or lower case spellings can be used interchangably, hyphens or underscores can be included or omitted. Setting type = NULL yields all available scores.
aggregate	logical or function to be used for aggregating scores across observations. Setting aggregate = TRUE (the default) corresponds to using mean.
drop	logical. Should scores be returned in a data frame (default) or (if possible) dropped to a vector?

Details

The function proscore provides a unified framework for scoring probabilistic forecasts (in-sample or out-of-sample). The following scores are currently available, using the following notation: Y is the predicted random variable with cumulative distribution function $F(\cdot)$ and probability density function $f(\cdot)$. The corresponding expectation and variance are denoted by $E(Y)$ and $V(Y)$. The actual observation is y.

Log-score: Also known as logarithmic score. This is the negative log-likelihood where the negative sign has the effect that smaller values indicate a better fit.

$$-\log f(y)$$

Log-likelihood: Also known as log-density. Clearly, this is equivalent to the log-score above but using the conventional sign where bigger values indicate a better fit.

$$\log f(y)$$

Continuous ranked probability score (CRPS):

$$\int_{-\infty}^{\infty} \left(F(x) - 1(x \geq y)\right)^2 \mathrm{d}x$$

where $1(\cdot)$ denotes the indicator function.

In case of a discrete rather than a continuous distribution, the ranked probability score (RPS) is defined analogously using the sum rather than the integral. In other words it is then the sum of the squared deviations between the predicted cumulative probabilities $F(x)$ and the ideal step function for the actual observation y.

Mean absolute error (MAE):

$$|y - E(Y)|$$

Mean squared error (MSE):

$$\left(y - E(Y)\right)^2$$

Dawid-Sebastiani score (DSS):

$$\frac{\left(y - E(Y)\right)^2}{V(Y)} + \log(V(Y))$$

Internally, the default `proscore` method first computes the fitted/predicted probability distribution object using `prodist` (corresponding to Y above) and then obtains the corresponding observation y using `newresponse`. Subsequently, the scores are evaluated using either the `log_pdf` method, `crps` method, or simply the `mean`. Finally, the resulting individual scores per observation can be returned as a full data frame, or aggregated (e.g., by using `mean`, `sum`, or `summary`, etc.).

Value

Either a `data.frame` of scores (if `drop = FALSE`, default) or a named numeric vector (if `drop = TRUE` and the scores are not a matrix). The names are the `type` specified by the user (i.e., are not canonicalized by partial matching etc.).

Examples

```
## Poisson regression model for FIFA 2018 data:
## number of goals scored by each team in each game, explained by
## predicted ability difference of the competing teams
data("FIFA2018", package = "distributions3")
m <- glm(goals ~ difference, data = FIFA2018, family = poisson)

## default: in-sample mean log-score and CRPS
```

```
proscore(m)

## element-wise score using a new data set (final of the tournament)
final <- tail(FIFA2018, 2)
proscore(m, newdata = final, aggregate = FALSE)

## replicate in-sample log-likelihood
proscore(m, type = "loglik", aggregate = sum)
logLik(m)

## compute mean of all available scores
proscore(m, type = NULL)

## upper vs. lower case spelling is matched internally but preserved in output
proscore(m, type = c("logs", "crps"))
proscore(m, type = c("Log-score", "CRPS"))

## least-squares regression for speed and breaking distance of cars
data("cars", package = "datasets")
m <- lm(dist ~ speed, data = cars)

## replicate in-sample log-likelihood and residual sum of squares
## (aka deviance) by taking the sum (rather than the mean) of the
## log-density and squared errors, respectively
proscore(m, type = c("loglik", "MSE"), aggregate = sum)
logLik(m)
deviance(m)
```

---

qqrplot *Q-Q Plots for Quantile Residuals*

Description

Visualize goodness of fit of regression models by Quantile-Quantile (Q-Q) plots using quantile residuals. If `plot = TRUE`, the resulting object of class `"qqrplot"` is plotted by `plot.qqrplot` or `autoplot.qqrplot` before it is returned, depending on whether the package `ggplot2` is loaded.

Usage

```
qqrplot(object, ...)

## Default S3 method:
qqrplot(
  object,
  newdata = NULL,
  plot = TRUE,
  class = NULL,
  detrend = FALSE,
  ref_type = "identity",
```

```
  scale = c("normal", "uniform"),
  nsim = 1L,
  delta = NULL,
  simint = TRUE,
  simint_level = 0.95,
  simint_nrep = 250,
  confint = TRUE,
  ref = TRUE,
  xlab = "Theoretical quantiles",
  ylab = if (!detrend) "Quantile residuals" else "Deviation",
  main = NULL,
  ...
)
```

Arguments

object	an object from which probability integral transforms can be extracted using the generic function procast.
newdata	an optional data frame in which to look for variables with which to predict. If omitted, the original observations are used.
plot	logical or character. Should the plot or autoplot method be called to draw the computed Q-Q plot? Logical FALSE will suppress plotting, TRUE (default) will choose the type of plot conditional if the package ggplot2 is loaded. Alternatively "base" or "ggplot2" can be specified to explicitly choose the type of plot.
class	should the invisible return value be either a data.frame or a tibble. Either set class expicitly to "data.frame" vs. "tibble", or for NULL it's chosen automatically conditional if the package tibble is loaded.
detrend	logical, defaults to FALSE. Should the qqrplot be detrended, i.e, plotted as a wormplot?
ref_type	character specifying that the "identity" line should be used as as a reference or the "quartiles" of the quantile residuals should be used for defining the reference line. Alternatively, also a numeric vector of length two can be used to define the probabilities to be used for defining the reference line. Note, that the reference is also used for detrending the quantile residuals.
scale	character. On which scale should the quantile residuals be shown: on the probability scale ("uniform") or on the normal scale ("normal").
nsim, delta	arguments passed to proresiduals.
simint	logical. In case of discrete distributions, should the simulation (confidence) interval due to the randomization be visualized?
simint_level	numeric. The confidence level required for calculating the simulation (confidence) interval due to the randomization.
simint_nrep	numeric (positive; default 250). The number of repetitions of simulated quantiles for calculating the simulation (confidence) interval due to the randomization.

`confint` logical or character describing the style for plotting confidence intervals. `TRUE` (default) and `"line"` will add point-wise confidence intervals of the (randomized) quantile residuals as lines, `"polygon"` will draw a polygon instead, and `FALSE` suppresses the drawing.

`ref` logical, defaults to `TRUE`. Should a reference line be plotted?

`xlab, ylab, main, ...` graphical parameters passed to `plot.qqrplot` or `autoplot.qqrplot`.

Details

Q-Q residuals plots draw quantile residuals (by default on the standard normal scale) against theoretical quantiles from the same distribution. Alternatively, quantile residuals can also be compared on the uniform scale (`scale = "uniform"`) using no transformation. For computation, `qqrplot` leverages the function `proresiduals` employing the `procast` generic.

Additional options are offered for models with discrete responses where randomization of quantiles is needed.

In addition to the `plot` and `autoplot` method for qqrplot objects, it is also possible to combine two (or more) Q-Q residuals plots by `c`/`rbind`, which creates a set of Q-Q residuals plots that can then be plotted in one go.

Value

An object of class `"qqrplot"` inheriting from `"data.frame"` or `"tibble"` conditional on the argument `class` with the following variables:

`observed` deviations between theoretical and empirical quantiles,

`expected` theoretical quantiles,

`simint_observed_lwr` lower bound of the simulated confidence interval,

`simint_observed_upr` upper bound of the simulated confidence interval,

`simint_expected` TODO: (ML) Description missing.

In case of `nsim > 1`, a set of `nsim` pairs of observed and expected quantiles are returned (`observed_1`, `expected_1`, ... `observed_nsim`, `observed_nsim`) is returned.

The `"qqrplot"` also contains additional attributes `xlab`, `ylab`, `main`, `simint_level`, `scale`, and `detrended` used to create the plot.

References

Dunn KP, Smyth GK (1996). "Randomized Quantile Residuals." *Journal of Computational and Graphical Statistics*, **5**(3), 236–244. doi:10.2307/1390802

See Also

`plot.qqrplot`, `wormplot`, `proresiduals`, `qqnorm`

Examples

```
## speed and stopping distances of cars
m1_lm <- lm(dist ~ speed, data = cars)

## compute and plot qqrplot
qqrplot(m1_lm)

#--------------------------------------------------------------------------------
## determinants for male satellites to nesting horseshoe crabs
data("CrabSatellites", package = "countreg")

## linear poisson model
m1_pois <- glm(satellites ~ width + color, data = CrabSatellites, family = poisson)
m2_pois <- glm(satellites ~ color, data = CrabSatellites, family = poisson)

## compute and plot qqrplot as base graphic
q1 <- qqrplot(m1_pois, plot = FALSE)
q2 <- qqrplot(m2_pois, plot = FALSE)

## plot combined qqrplot as "ggplot2" graphic
ggplot2::autoplot(c(q1, q2), single_graph = TRUE, col = c(1, 2), fill = c(1, 2))

## Use different `scale`s with confidence intervals
qqrplot(m1_pois, scale = "uniform")
qqrplot(m1_pois, scale = "normal")
qqrplot(m1_pois, detrend = TRUE, scale = "uniform", confint = "line")
qqrplot(m1_pois, detrend = TRUE, scale = "normal", confint = "line")
```

`reliagram` *Reliagram (Extended Reliability Diagram)*

Description

Reliagram (extended reliability diagram) assess the reliability of a fitted probabilistic distributional forecast for a binary event. If `plot = TRUE`, the resulting object of class `"reliagram"` is plotted by `plot.reliagram` or `autoplot.reliagram` before it is returned, depending on whether the package `ggplot2` is loaded.

Usage

```
reliagram(object, ...)

## Default S3 method:
reliagram(
  object,
  newdata = NULL,
  plot = TRUE,
  class = NULL,
```

```
  breaks = seq(0, 1, by = 0.1),
  quantiles = 0.5,
  thresholds = NULL,
  confint = TRUE,
  confint_level = 0.95,
  confint_nboot = 250,
  confint_seed = 1,
  single_graph = FALSE,
  xlab = "Forecast probability",
  ylab = "Observed relative frequency",
  main = NULL,
  ...
)
```

Arguments

object	an object from which an extended reliability diagram can be extracted with procast.
...	further graphical parameters.
newdata	optionally, a data frame in which to look for variables with which to predict. If omitted, the original observations are used.
plot	Should the plot or autoplot method be called to draw the computed extended reliability diagram? Either set plot expicitly to "base" vs. "ggplot2" to choose the type of plot, or for a logical plot argument it's chosen conditional if the package ggplot2 is loaded.
class	Should the invisible return value be either a data.frame or a tibble. Either set class expicitly to "data.frame" vs. "tibble", or for NULL it's chosen automatically conditional if the package tibble is loaded.
breaks	numeric vector passed on to cut in order to bin the observations and the predicted probabilities or a function applied to the predicted probabilities to calculate a numeric value for cut. Typically quantiles to ensure equal number of predictions per bin, e.g., by breaks = function(x) quantile(x).
quantiles	numeric vector of quantile probabilities with values in [0,1] to calculate single or several thresholds. Only used if thresholds is not specified. For binary responses typically the 50%-quantile is used.
thresholds	numeric vector specifying both where to cut the observations into binary values and at which values the predicted probabilities should be calculated (procast).
confint	logical. Should confident intervals be calculated and drawn?
confint_level	numeric. The confidence level required.
confint_nboot	numeric. The number of bootstrap steps.
confint_seed	numeric. The seed to be set for the bootstrapping.
single_graph	logical. Should all computed extended reliability diagrams be plotted in a single graph?
xlab, ylab, main	graphical parameters.

Details

Reliagrams evaluate if a probability model is calibrated (reliable) by first partitioning the predicted probability for a binary event into a certain number of bins and then plotting (within each bin) the averaged forecast probability against the observered/empirical relative frequency. For computation, `reliagram` leverages the `procast` generic to forecast the respective predictive probabilities.

For continous probability forecasts, reliability diagrams can be computed either for a pre-specified threshold or for a specific quantile probability of the response values. Per default, reliagrams are computed for the 50%-quantile of the reponse.

In addition to the `plot` and `autoplot` method for reliagram objects, it is also possible to combine two (or more) reliability diagrams by `c`/`rbind`, which creates a set of reliability diagrams that can then be plotted in one go.

Value

An object of class `"reliagram"` inheriting from `"data.frame"` or `"tibble"` conditional on the argument `class` with the following variables:

`x`	forecast probabilities,
`y`	observered/empirical relative frequencies,
`bin_lwr`, `bin_upr`	lower and upper bound of the binned forecast probabilities,
`n_pred`	number of predictions within the binned forecasts probabilites,
`ci_lwr`, `ci_upr`	lower and upper confidence interval bound.

Additionally, `xlab`, `ylab`, `main`, and `treshold`, `confint_level`, as well as the total and the decomposed Brier Score (`bs`, `rel`, `res`, `unc`) are stored as attributes.

Note

Note that there is also a `reliability.plot` function in the **verification** package. However, it only works for numeric forecast probabilities and numeric observed relative frequencies, hence a function has been created here.

References

Wilks DS (2011) *Statistical Methods in the Atmospheric Sciences*, 3rd ed., Academic Press, 704 pp.

See Also

`link{plot.reliagram}`, `procast`

Examples

```
## speed and stopping distances of cars
m1_lm <- lm(dist ~ speed, data = cars)

## compute and plot reliagram
reliagram(m1_lm)
```

```
#-------------------------------------------------------------------------------
## determinants for male satellites to nesting horseshoe crabs
data("CrabSatellites", package = "countreg")

## linear poisson model
m1_pois <- glm(satellites ~ width + color, data = CrabSatellites, family = poisson)
m2_pois <- glm(satellites ~ color, data = CrabSatellites, family = poisson)

## compute and plot reliagram as base graphic
r1 <- reliagram(m1_pois, plot = FALSE)
r2 <- reliagram(m2_pois, plot = FALSE)

## plot combined reliagram as "ggplot2" graphic
ggplot2::autoplot(c(r1, r2), single_graph = TRUE, col = c(1, 2), fill = c(1, 2))
```

---

rootogram *Rootograms for Assessing Goodness of Fit of Probability Models*

Description

Rootograms graphically compare (square roots) of empirical frequencies with expected (fitted) frequencies from a probabilistic model. If `plot = TRUE`, the resulting object of class `"rootogram"` is plotted by `plot.rootogram` or `autoplot.rootogram` before it is returned, depending on whether the package `ggplot2` is loaded.

Usage

```
rootogram(object, ...)

## Default S3 method:
rootogram(
  object,
  newdata = NULL,
  plot = TRUE,
  class = NULL,
  breaks = NULL,
  width = NULL,
  style = c("hanging", "standing", "suspended"),
  scale = c("sqrt", "raw"),
  expected = TRUE,
  confint = TRUE,
  ref = TRUE,
  xlab = NULL,
  ylab = NULL,
  main = NULL,
  ...
)
```

Arguments

object	an object from which an rootogram can be extracted with procast.
...	further graphical parameters passed to the plotting function.
newdata	an optional data frame in which to look for variables with which to predict. If omitted, the original observations are used.
plot	logical or character. Should the plot or autoplot method be called to draw the computed extended reliability diagram? Logical FALSE will suppress plotting, TRUE (default) will choose the type of plot conditional if the package ggplot2 is loaded. Alternatively "base" or "ggplot2" can be specified to explicitly choose the type of plot.
class	should the invisible return value be either a data.frame or a tbl_df. Can be set to "data.frame" or "tibble" to explicitly specify the return class, or to NULL (default) in which case the return class is conditional on whether the package "tibble" is loaded.
breaks	NULL (default) or numeric vector to specifying the breaks for the rootogram intervals. A single numeric (larger than 0) specifies the number of breaks to be chosen via pretty (except for discrete distributions).
width	NULL (default) or single positive numeric. Width of the histogram bars. Will be ignored for non-discrete distributions.
style	character specifying the syle of rootogram (see 'Details').
scale	character specifying whether "raw" frequencies or their square roots ("sqrt"; default) should be drawn.
expected	logical or character. Should the expected (fitted) frequencies be plotted? Can be set to "both" (same as TRUE; default), "line", "point", or FALSE which will suppress plotting.
confint	logical, defaults to TRUE. Should confident intervals be drawn?
ref	logical, defaults to TRUE. Should a reference line be plotted?
xlab, ylab, main	graphical parameters forwarded to plot.rootogram or autoplot.rootogram.

Details

Rootograms graphically compare frequencies of empirical distributions and expected (fitted) probability models. For the observed distribution the histogram is drawn on a square root scale (hence the name) and superimposed with a line for the expected frequencies. The histogram can be "hanging" from the expected curve (default), "standing" on the (like bars in barplot), or drawn as a "suspended" histogram of deviations.

Rootograms are associated with the work of John W. Tukey (see Tukey 1977) and were originally proposed for assessing the goodness of fit of univariate distributions. See Friendly (2000) for a software implementation, in particular geared towards count data models. Kleiber and Zeileis (2016) extend it to regression models for count data, essentially by replacing the expected frequencies of a univariate distribution by the sum of the expected frequencies from the different conditional distributions for all observations.

The function rootogram leverages the procast generic in order to compute all necessary coordinates based on observed and expected (fitted) frequencies. It is thus not only applicable to count data regressions but to all (regression) models that are supported by procast.

In addition to the `plot` and `autoplot` method for rootogram objects, it is also possible to combine two (or more) rootograms by `c`/`rbind`, which creates a set of rootograms that can then be plotted in one go.

Value

An object of class `"rootogram"` inheriting from `"data.frame"` or `"tibble"` conditional on the argument `class` with the following variables:

`observed`	observed frequencies,
`expected`	expected (fitted) frequencies,
`mid`	histogram interval midpoints on the x-axis,
`width`	widths of the histogram bars,
`confint_lwr`, `confint_upr`	lower and upper confidence interval bound.

Additionally, `style`, `scale`, `expected`, `confint`, `ref`, `xlab`, `ylab`, amd `main` are stored as attributes.

Note

Note that there is also a `rootogram` function in the **vcd** package that is similar to the `numeric` method provided here. However, it is much more limited in scope, hence a function has been created here.

References

Friendly M (2000), *Visualizing Categorical Data*. SAS Institute, Cary.

Kleiber C, Zeileis A (2016). "Visualizing Count Data Regressions Using Rootograms." *The American Statistician*, **70**(3), 296–303. doi:10.1080/00031305.2016.1173590

Tukey JW (1977). *Exploratory Data Analysis*. Addison-Wesley, Reading.

See Also

`plot.rootogram`, `procast`

Examples

```
## plots and output

## number of deaths by horsekicks in Prussian army (Von Bortkiewicz 1898)
deaths <- rep(0:4, c(109, 65, 22, 3, 1))

## fit glm model
m1_pois <- glm(deaths ~ 1, family = poisson)
rootogram(m1_pois)

## inspect output (without plotting)
r1 <- rootogram(m1_pois, plot = FALSE)
```

```
r1

## combine plots
plot(c(r1, r1), col = c(1, 2), expected_col = c(1, 2))


#-------------------------------------------------------------------------------
## different styles

## artificial data from negative binomial (mu = 3, theta = 2)
## and Poisson (mu = 3) distribution
set.seed(1090)
y <- rnbinom(100, mu = 3, size = 2)
x <- rpois(100, lambda = 3)

## glm method: fitted values via glm()
m2_pois <- glm(y ~ x, family = poisson)

## correctly specified Poisson model fit
par(mfrow = c(1, 3))
r1 <- rootogram(m2_pois, style = "standing", ylim = c(-2.2, 4.8), main = "Standing")
r2 <- rootogram(m2_pois, style = "hanging", ylim = c(-2.2, 4.8), main = "Hanging")
r3 <- rootogram(m2_pois, style = "suspended", ylim = c(-2.2, 4.8), main = "Suspended")
par(mfrow = c(1, 1))

#-------------------------------------------------------------------------------
## linear regression with normal/Gaussian response: anorexia data

data("anorexia", package = "MASS")

m3_gauss <- glm(Postwt ~ Prewt + Treat + offset(Prewt), family = gaussian, data = anorexia)

## plot rootogram as "ggplot2" graphic
rootogram(m3_gauss, plot = "ggplot2")
```

SerumPotassium *Serum Potassium Levels*

Description

Sample of 152 serum potassium levels.

Usage

```
data("SerumPotassium", package = "topmodels")
```

Format

A numeric vector of 152 serum potassium levels.

Details

The data are taken from Rice (2007) who obtained the data from Martin, Gudzinowicz and Fanger (1975) and reports them rounded to one digit.

Source

Page 350 in Rice (2007).

References

Rice JA (2007). *Mathematical Statistics and Data Analysis*, 3rd ed. Duxbury, Belmont, CA.

Martin HF, Gudzinowicz BJ, Fanger H (1975). *Normal Values in Clinical Chemistry: A Guide to Statistical Analysis of Laboratory Data*. Marcel Dekker, New York.

Examples

```
library("topmodels")
data("SerumPotassium", package = "topmodels")

## Figure 9.3a-c from Rice (2007), and actual hanging rootogram
## (note that Rice erroneously refers to suspended rootograms as hanging)
sp <- lm(SerumPotassium ~ 1)
br <- 32:54/10 - 0.05
rootogram(sp, scale = "raw", style = "standing",
  breaks = br, col = "transparent")
rootogram(sp, scale = "raw", style = "suspended",
  breaks = br, col = "transparent", ylim = c(2.8, -4))
rootogram(sp, scale = "sqrt", style = "suspended",
  breaks = br, col = "transparent", ylim = c(1, -1.5))
rootogram(sp, breaks = br)
```

---

stat_pithist `geom_*` *and* `stat_*` *for Producing PIT Histograms with 'ggplot2'*

Description

Various `geom_*` and `stat_*` used within `autoplot` for producing PIT histograms.

Usage

```
stat_pithist(
  mapping = NULL,
  data = NULL,
  geom = "pithist",
  position = "identity",
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE,
```

```
  freq = FALSE,
  style = c("bar", "line"),
  ...
)

geom_pithist(
  mapping = NULL,
  data = NULL,
  stat = "pithist",
  position = "identity",
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE,
  freq = FALSE,
  style = c("bar", "line"),
  ...
)

stat_pithist_expected(
  mapping = NULL,
  data = NULL,
  geom = "pithist_expected",
  position = "identity",
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE,
  scale = c("uniform", "normal"),
  freq = FALSE,
  ...
)

geom_pithist_expected(
  mapping = NULL,
  data = NULL,
  stat = "pithist_expected",
  position = "identity",
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE,
  scale = c("uniform", "normal"),
  freq = FALSE,
  ...
)

stat_pithist_confint(
  mapping = NULL,
  data = NULL,
  geom = "pithist_confint",
```

```
  position = "identity",
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE,
  scale = c("uniform", "normal"),
  level = 0.95,
  type = "approximation",
  freq = FALSE,
  style = c("polygon", "line"),
  ...
)

geom_pithist_confint(
  mapping = NULL,
  data = NULL,
  stat = "pithist_confint",
  position = "identity",
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE,
  scale = c("uniform", "normal"),
  level = 0.95,
  type = "approximation",
  freq = FALSE,
  style = c("polygon", "line"),
  ...
)

stat_pithist_simint(
  mapping = NULL,
  data = NULL,
  geom = "pithist_simint",
  position = "identity",
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE,
  freq = FALSE,
  ...
)

geom_pithist_simint(
  mapping = NULL,
  data = NULL,
  stat = "pithist_simint",
  position = "identity",
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE,
```

```
  freq = FALSE,
  ...
)
```

Arguments

mapping	Set of aesthetic mappings created by aes(). If specified and inherit.aes = TRUE (the default), it is combined with the default mapping at the top level of the plot. You must supply mapping if there is no plot mapping.
data	The data to be displayed in this layer. There are three options: If NULL, the default, the data is inherited from the plot data as specified in the call to ggplot(). A data.frame, or other object, will override the plot data. All objects will be fortified to produce a data frame. See fortify() for which variables will be created. A function will be called with a single argument, the plot data. The return value must be a data.frame, and will be used as the layer data. A function can be created from a formula (e.g. ~ head(.x, 10)).
geom	The geometric object to use to display the data for this layer. When using a stat_*() function to construct a layer, the geom argument can be used to override the default coupling between stats and geoms. The geom argument accepts the following: • A Geom ggproto subclass, for example GeomPoint. • A string naming the geom. To give the geom as a string, strip the function name of the geom_ prefix. For example, to use geom_point(), give the geom as "point". • For more information and other ways to specify the geom, see the layer geom documentation.
position	A position adjustment to use on the data for this layer. This can be used in various ways, including to prevent overplotting and improving the display. The position argument accepts the following: • The result of calling a position function, such as position_jitter(). This method allows for passing extra arguments to the position. • A string naming the position adjustment. To give the position as a string, strip the function name of the position_ prefix. For example, to use position_jitter(), give the position as "jitter". • For more information and other ways to specify the position, see the layer position documentation.
na.rm	If FALSE, the default, missing values are removed with a warning. If TRUE, missing values are silently removed.
show.legend	logical. Should this layer be included in the legends? NA, the default, includes if any aesthetics are mapped. FALSE never includes, and TRUE always includes. It can also be a named logical vector to finely select the aesthetics to display. To include legend keys for all levels, even when no data exists, use TRUE. If NA, all levels are shown in legend, but unobserved levels are omitted.

inherit.aes — If FALSE, overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn't inherit behaviour from the default plot specification, e.g. annotation_borders().

freq — logical. If TRUE, the PIT histogram is represented by frequencies, the counts component of the result; if FALSE, probability densities, component density, are plotted (so that the histogram has a total area of one).

style — character specifying the style of pithist. For style = "bar" a traditional PIT hisogram is drawn, for style = "line" solely the upper border line is plotted.

... — Other arguments passed on to layer()'s params argument. These arguments broadly fall into one of 4 categories below. Notably, further arguments to the position argument, or aesthetics that are required can *not* be passed through .... Unknown arguments that are not part of the 4 categories below are ignored.

- Static aesthetics that are not mapped to a scale, but are at a fixed value and apply to the layer as a whole. For example, colour = "red" or linewidth = 3. The geom's documentation has an **Aesthetics** section that lists the available options. The 'required' aesthetics cannot be passed on to the params. Please note that while passing unmapped aesthetics as vectors is technically possible, the order and required length is not guaranteed to be parallel to the input data.
- When constructing a layer using a stat_*() function, the ... argument can be used to pass on parameters to the geom part of the layer. An example of this is stat_density(geom = "area", outline.type = "both"). The geom's documentation lists which parameters it can accept.
- Inversely, when constructing a layer using a geom_*() function, the ... argument can be used to pass on parameters to the stat part of the layer. An example of this is geom_area(stat = "density", adjust = 0.5). The stat's documentation lists which parameters it can accept.
- The key_glyph argument of layer() may also be passed on through .... This can be one of the functions described as key glyphs, to change the display of the layer in the legend.

stat — The statistical transformation to use on the data for this layer. When using a geom_*() function to construct a layer, the stat argument can be used to override the default coupling between geoms and stats. The stat argument accepts the following:

- A Stat ggproto subclass, for example StatCount.
- A string naming the stat. To give the stat as a string, strip the function name of the stat_ prefix. For example, to use stat_count(), give the stat as "count".
- For more information and other ways to specify the stat, see the layer stat documentation.

scale — On which scale should the PIT residuals be computed: on the probability scale ("uniform") or on the normal scale ("normal").

level — numeric. The confidence level required.

type — character. Which type of confidence interval should be plotted: ‘"exact"‘ or ‘"approximation"‘. According to Agresti and Coull (1998), for interval estimation of binomial proportions an approximation can be better than exact.

Examples

```
if (require("ggplot2")) {
  ## Fit model
  data("CrabSatellites", package = "countreg")
  m1_pois <- glm(satellites ~ width + color, data = CrabSatellites, family = poisson)
  m2_pois <- glm(satellites ~ color, data = CrabSatellites, family = poisson)

  ## Compute pithist
  p1 <- pithist(m1_pois, type = "random", plot = FALSE)
  p2 <- pithist(m2_pois, type = "random", plot = FALSE)

  d <- c(p1, p2)

  ## Create factor
  main <- attr(d, "main")
  main <- make.names(main, unique = TRUE)
  d$group <- factor(d$group, labels = main)

  ## Plot bar style PIT histogram
  gg1 <- ggplot(data = d) +
    geom_pithist(aes(x = mid, y = observed, width = width, group = group), freq = TRUE) +
    geom_pithist_simint(aes(x = mid, ymin = simint_lwr, ymax = simint_upr), freq = TRUE) +
   geom_pithist_confint(aes(x = mid, y = observed, width = width), style = "line", freq = TRUE) +
    geom_pithist_expected(aes(x = mid, y = observed, width = width), freq = TRUE) +
    facet_grid(group ~ .) +
    xlab("PIT") +
    ylab("Frequency")
  gg1

  gg2 <- ggplot(data = d) +
    geom_pithist(aes(x = mid, y = observed, width = width, group = group), freq = FALSE) +
    geom_pithist_simint(aes(
      x = mid, ymin = simint_lwr, ymax = simint_upr, y = observed,
      width = width
    ), freq = FALSE) +
   geom_pithist_confint(aes(x = mid, y = observed, width = width), style = "line", freq = FALSE) +
    geom_pithist_expected(aes(x = mid, y = observed, width = width), freq = FALSE) +
    facet_grid(group ~ .) +
    xlab("PIT") +
    ylab("Density")
  gg2

  ## Plot line style PIT histogram
  gg3 <- ggplot(data = d) +
   geom_pithist(aes(x = mid, y = observed, width = width, group = group), style = "line") +
    geom_pithist_confint(aes(x = mid, y = observed, width = width), style = "polygon") +
    facet_grid(group ~ .) +
    xlab("PIT") +
    ylab("Density")
  gg3
}
```

stat_rootogram geom_* *and* stat_* *for Producing Rootograms with 'ggplot2'*

Description

Various geom_* and stat_* used within autoplot for producing rootograms.

Usage

```
stat_rootogram(
  mapping = NULL,
  data = NULL,
  geom = "rootogram",
  position = "identity",
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE,
  scale = c("sqrt", "raw"),
  style = c("hanging", "standing", "suspended"),
  ...
)

geom_rootogram(
  mapping = NULL,
  data = NULL,
  stat = "rootogram",
  position = "identity",
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE,
  scale = c("sqrt", "raw"),
  style = c("hanging", "standing", "suspended"),
  ...
)

stat_rootogram_expected(
  mapping = NULL,
  data = NULL,
  geom = "rootogram_expected",
  position = "identity",
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE,
  scale = c("sqrt", "raw"),
  ...
)
```

```
geom_rootogram_expected(
  mapping = NULL,
  data = NULL,
  stat = "rootogram_expected",
  position = "identity",
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE,
  scale = c("sqrt", "raw"),
  linestyle = c("both", "line", "point"),
  ...
)

GeomRootogramExpected

geom_rootogram_ref(
  mapping = NULL,
  data = NULL,
  stat = "identity",
  position = "identity",
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE,
  ...
)

stat_rootogram_confint(
  mapping = NULL,
  data = NULL,
  geom = "rootogram_confint",
  position = "identity",
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE,
  level = 0.95,
  nrep = 1000,
  type = c("tukey", "pointwise", "simultaneous"),
  scale = c("sqrt", "raw"),
  rootogram_style = c("hanging", "standing", "suspended"),
  ...
)

geom_rootogram_confint(
  mapping = NULL,
  data = NULL,
  stat = "rootogram_confint",
  position = "identity",
  na.rm = FALSE,
```

```
  show.legend = NA,
  inherit.aes = TRUE,
  level = 0.95,
  nrep = 1000,
  type = c("tukey", "pointwise", "simultaneous"),
  scale = c("sqrt", "raw"),
  rootogram_style = c("hanging", "standing", "suspended"),
  ...
)
```

Arguments

mapping	Set of aesthetic mappings created by aes(). If specified and inherit.aes = TRUE (the default), it is combined with the default mapping at the top level of the plot. You must supply mapping if there is no plot mapping.
data	The data to be displayed in this layer. There are three options: If NULL, the default, the data is inherited from the plot data as specified in the call to ggplot(). A data.frame, or other object, will override the plot data. All objects will be fortified to produce a data frame. See fortify() for which variables will be created. A function will be called with a single argument, the plot data. The return value must be a data.frame, and will be used as the layer data. A function can be created from a formula (e.g. ~ head(.x, 10)).
geom	The geometric object to use to display the data for this layer. When using a stat_*() function to construct a layer, the geom argument can be used to override the default coupling between stats and geoms. The geom argument accepts the following: • A Geom ggproto subclass, for example GeomPoint. • A string naming the geom. To give the geom as a string, strip the function name of the geom_ prefix. For example, to use geom_point(), give the geom as "point". • For more information and other ways to specify the geom, see the layer geom documentation.
position	A position adjustment to use on the data for this layer. This can be used in various ways, including to prevent overplotting and improving the display. The position argument accepts the following: • The result of calling a position function, such as position_jitter(). This method allows for passing extra arguments to the position. • A string naming the position adjustment. To give the position as a string, strip the function name of the position_ prefix. For example, to use position_jitter(), give the position as "jitter". • For more information and other ways to specify the position, see the layer position documentation.
na.rm	If FALSE, the default, missing values are removed with a warning. If TRUE, missing values are silently removed.

show.legend — logical. Should this layer be included in the legends? NA, the default, includes if any aesthetics are mapped. FALSE never includes, and TRUE always includes. It can also be a named logical vector to finely select the aesthetics to display. To include legend keys for all levels, even when no data exists, use TRUE. If NA, all levels are shown in legend, but unobserved levels are omitted.

inherit.aes — If FALSE, overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn't inherit behaviour from the default plot specification, e.g. annotation_borders().

scale — character specifying whether values should be transformed to the square root scale (not checking for original scale, so maybe applied again).

style — character specifying the syle of rootogram (see below).

... — Other arguments passed on to layer()'s params argument. These arguments broadly fall into one of 4 categories below. Notably, further arguments to the position argument, or aesthetics that are required can *not* be passed through .... Unknown arguments that are not part of the 4 categories below are ignored.

- Static aesthetics that are not mapped to a scale, but are at a fixed value and apply to the layer as a whole. For example, colour = "red" or linewidth = 3. The geom's documentation has an **Aesthetics** section that lists the available options. The 'required' aesthetics cannot be passed on to the params. Please note that while passing unmapped aesthetics as vectors is technically possible, the order and required length is not guaranteed to be parallel to the input data.
- When constructing a layer using a stat_*() function, the ... argument can be used to pass on parameters to the geom part of the layer. An example of this is stat_density(geom = "area", outline.type = "both"). The geom's documentation lists which parameters it can accept.
- Inversely, when constructing a layer using a geom_*() function, the ... argument can be used to pass on parameters to the stat part of the layer. An example of this is geom_area(stat = "density", adjust = 0.5). The stat's documentation lists which parameters it can accept.
- The key_glyph argument of layer() may also be passed on through .... This can be one of the functions described as key glyphs, to change the display of the layer in the legend.

stat — The statistical transformation to use on the data for this layer. When using a geom_*() function to construct a layer, the stat argument can be used to override the default coupling between geoms and stats. The stat argument accepts the following:

- A Stat ggproto subclass, for example StatCount.
- A string naming the stat. To give the stat as a string, strip the function name of the stat_ prefix. For example, to use stat_count(), give the stat as "count".
- For more information and other ways to specify the stat, see the layer stat documentation.

linestyle — Character string defining one of '"both"', '"line"' or '"point"'.

level — numeric. The confidence level required.

nrep — numeric. The repetition number of simulation for computing the confidence intervals.

type — character. Should "tukey", "pointwise", or "simultaneous" confidence intervals be visualized?

rootogram_style — character specifying the syle of rootogram.

Format

An object of class GeomRootogramExpected (inherits from GeomPath, Geom, ggproto, gg) of length 3.

Examples

```
if (require("ggplot2")) {
  ## Fit model
  data("CrabSatellites", package = "countreg")
  m1_pois <- glm(satellites ~ width + color, data = CrabSatellites, family = poisson)
  m2_pois <- glm(satellites ~ color, data = CrabSatellites, family = poisson)

  ## Compute rootogram (on raw scale)
  p1 <- rootogram(m1_pois, scale = "raw", plot = FALSE)
  p2 <- rootogram(m2_pois, scale = "raw", plot = FALSE)

  d <- c(p1, p2)

  ## Get label names
  main <- attr(d, "main")
  main <- make.names(main, unique = TRUE)
  d$group <- factor(d$group, labels = main)

  ## Plot rootograms w/ on default "sqrt" scale
  gg1 <- ggplot(data = d) +
    geom_rootogram(aes(
      observed = observed, expected = expected, mid = mid,
      width = width, group = group
    )) +
    geom_rootogram_expected(aes(expected = expected, mid = mid)) +
    geom_rootogram_ref() +
    facet_grid(group ~ .) +
    xlab("satellites") +
    ylab("sqrt(Frequency)")
  gg1
}
```

---

topmodels *Plotting Graphical Evaluation Tools for Probabilistic Models*

Description

A quick overview plot with panels for all graphical evaluation methods provided for probabilistic (regression) model objects. If `plot = TRUE`, the resulting objects are plotted by `plot` or `autoplot` before they are returned within a single list, depending on whether the package `ggplot2` is loaded.

Usage

```
topmodels(
  object,
  plot = TRUE,
  class = NULL,
  newdata = NULL,
  na.action = na.pass,
  which = NULL,
  ask = dev.interactive(),
  spar = TRUE,
  single_page = NULL,
  envir = parent.frame(),
  ...
)
```

Arguments

`object`	An object supported by `"procast"`.
`plot`	Should the `plot` or `autoplot` method be called to draw all chosen plots? Either set `plot` expicitly to `"base"` vs. `"ggplot2"` to choose the type of plot, or for a logical `plot` argument it's chosen conditional if the package `ggplot2` is loaded.
`class`	Should the invisible return value be either a `data.frame` or a `tibble`. Either set `class` expicitly to `"data.frame"` vs. `"tibble"`, or for `NULL` it's chosen automatically conditional if the package `tibble` is loaded.
`newdata`	optionally, a data frame in which to look for variables with which to predict. If omitted, the original observations are used.
`na.action`	function determining what should be done with missing values in `newdata`. The default is to employ `NA`.
`which`	Character or integer, selects the type of plot: `"rootogram"` graphically compares (square roots) of empirical frequencies with fitted frequencies from a probability model, `"pithist"` compares empirical probabilities from fitted models with a uniform distribution, `"reliagram"` shows a reliability diagram for assessing the reliability of a fitted probabilistic distributional forecast, `"qqrplot"` shows a quantile-quantile plot of quantile residuals, and `"wormplot"` shows a worm plot using quantile resiudals.
`ask`	For multiple plots, the user is asked to show the next plot. Argument is ignored for `ggplot2` style graphics.
`spar`	Should graphical parameters be set? Will be ignored for `ggplot2` style graphics.
`single_page`	Logical. Should all plots be shown on a single page? Only choice for `ggplot2` style graphics.

envir	environment, default is parent.frame()
...	Arguments to be passed to rootogram, pithist, reliagram, qqrplot, and wormplot.

Details

Render the diagnostic graphics rootogram, pithist, reliagram qqrplot, and wormplot.

Value

A list containing the objects plotted conditional on the arguemnt which.

See Also

rootogram, pithist, reliagram qqrplot, wormplot

Examples

```
data("CrabSatellites", package = "countreg")
CrabSatellites2 <- CrabSatellites[CrabSatellites$satellites <= 1, ]

m1 <- glm(satellites ~ width + color, data = CrabSatellites, family = poisson)
m2 <- glm(satellites ~ width + color, data = CrabSatellites2, family = binomial)

## ggplot2 graphics
topmodels(m1, single_page = TRUE, nsim = 30, plot = "ggplot2")
topmodels(m2, single_page = TRUE, nsim = 30, plot = "ggplot2")
```

VolcanoHeights *Tukey's Volcano Heights*

Description

Heights of 218 volcanos taken from Tukey (1972).

Usage

```
data("VolcanoHeights", package = "topmodels")
```

Format

A numeric vector of 218 volcano heights (in 1000 feet).

Details

The data are taken from Tukey (1972) who obtained them from *The World Almanac, 1966* (New York: The New York World-Telegram and The Sun, 1966), pp. 282–283.

Source

Figure 1 in Tukey (1972).

References

Tukey JW (1972). "Some Graphic and Semigraphic Displays." In Bancroft TA (ed.), *Statistical Papers in Honor of George W. Snedecor*, pp. 293–316. Iowa State University Press, Ames, IA. Reprinted in Cleveland WS (ed.): *The Collected Works of John W. Tukey, Volume V. Graphics: 1965–1985*, Wadsworth & Brooks/Cole, Pacific Grove, CA, 1988.

Examples

```
## Rootograms from Tukey (1972)
## (some 'breaks' don't match exactly)
library("topmodels")
data("VolcanoHeights", package = "topmodels")

## Figure 16
rootogram(lm(VolcanoHeights ~ 1), style = "standing",
  breaks = 0:20 - 0.01, expected = FALSE, confint = FALSE)

## Figure 17
rootogram(lm(sqrt(1000 * VolcanoHeights) ~ 1), style = "standing",
  breaks = 0:17 * 10 - 1.1, expected = FALSE, confint = FALSE)

## Figure 18
rootogram(lm(sqrt(1000 * VolcanoHeights) ~ 1), style = "hanging",
  breaks = -2:18 * 10 - 1.1, confint = FALSE)

## Figure 19
rootogram(lm(sqrt(1000 * VolcanoHeights) ~ 1), style = "suspended",
  breaks = -2:18 * 10 - 1.1, ylim = c(6, -2), confint = FALSE)
abline(h = c(-1.5, -1, 1, 1.5), lty = c(2, 3, 3, 2))
```

`wormplot` *Worm Plots for Quantile Residuals*

Description

Visualize goodness of fit of regression models by worm plots using quantile residuals. If `plot = TRUE`, the resulting object of class `"wormplot"` is plotted by `plot.qqrplot` or `autoplot.qqrplot` before it is returned, depending on whether the package `ggplot2` is loaded.

Usage

```
wormplot(object, ...)

## Default S3 method:
wormplot(
```

```
  object,
  newdata = NULL,
  plot = TRUE,
  class = NULL,
  detrend = TRUE,
  scale = c("normal", "uniform"),
  nsim = 1L,
  delta = NULL,
  confint = TRUE,
  simint = TRUE,
  simint_level = 0.95,
  simint_nrep = 250,
  single_graph = FALSE,
  xlab = "Theoretical quantiles",
  ylab = "Deviation",
  main = NULL,
  ...
)
```

Arguments

object	an object from which probability integral transforms can be extracted using the generic function procast.
newdata	optionally, a data frame in which to look for variables with which to predict. If omitted, the original observations are used.
plot	Should the plot or autoplot method be called to draw the computed Q-Q plot? Either set plot expicitly to "base" vs. "ggplot2" to choose the type of plot, or for a logical plot argument it's chosen conditional if the package ggplot2 is loaded.
class	Should the invisible return value be either a data.frame or a tibble. Either set class expicitly to "data.frame" vs. "tibble", or for NULL it's chosen automatically conditional if the package tibble is loaded.
detrend	logical. Should the qqrplot be detrended, i.e, plotted as a 'wormplot()'?
scale	On which scale should the quantile residuals be shown: on the probability scale ("uniform") or on the normal scale ("normal").
nsim, delta	arguments passed to proresiduals.
confint	logical or character string describing the type for plotting 'c("polygon", "line")'. If not set to 'FALSE', the pointwise confidence interval of the (randomized) quantile residuals are visualized.
simint	logical. In case of discrete distributions, should the simulation (confidence) interval due to the randomization be visualized?
simint_level	numeric. The confidence level required for calculating the simulation (confidence) interval due to the randomization.
simint_nrep	numeric. The repetition number of simulated quantiles for calculating the simulation (confidence) interval due to the randomization.

`single_graph` logical. Should all computed extended reliability diagrams be plotted in a single graph?

`xlab`, `ylab`, `main`, `...` graphical parameters passed to `plot.qqrplot` or `autoplot.qqrplot`.

Details

Worm plots (de-trended Q-Q plots) draw deviations of quantile residuals (by default: transformed to standard normal scale) and theoretical quantiles from the same distribution against the same theoretical quantiles. For computation, `wormplot` leverages the function `proresiduals` employing the `procast`.

Additional options are offered for models with discrete responses where randomization of quantiles is needed.

In addition to the `plot` and `autoplot` method for wormplot objects, it is also possible to combine two (or more) worm plots by `c`/`rbind`, which creates a set of worm plots that can then be plotted in one go.

Value

An object of class `"qqrplot"` inheriting from `"data.frame"` or `"tibble"` conditional on the argument `class` with the following variables:

`x` theoretical quantiles,

`y` deviations between theoretical and empirical quantiles.

In case of randomized residuals, `nsim` different `x` and `y` values, and lower and upper confidence interval bounds (`x_rg_lwr`, `y_rg_lwr`, `x_rg_upr`, `y_rg_upr`) can optionally be returned. Additionally, `xlab`, `ylab`, `main`, and `simint_level`, as well as the the (`scale`) and wether a `detrended` Q-Q residuals plot was computed are stored as attributes.

References

van Buuren S and Fredriks M (2001). "Worm plot: simple diagnostic device for modelling growth reference curves". *Statistics in Medicine*, **20**, 1259–1277. doi:10.1002/sim.746

See Also

`qqrplot`, `plot.qqrplot`, `qqrplot`, `proresiduals`, `qqnorm`

Examples

```
## speed and stopping distances of cars
m1_lm <- lm(dist ~ speed, data = cars)

## compute and plot wormplot
wormplot(m1_lm)

#--------------------------------------------------------------------------------
## determinants for male satellites to nesting horseshoe crabs
data("CrabSatellites", package = "countreg")
```

```
## linear poisson model
m1_pois <- glm(satellites ~ width + color, data = CrabSatellites, family = poisson)
m2_pois <- glm(satellites ~ color, data = CrabSatellites, family = poisson)

## compute and plot wormplot as base graphic
w1 <- wormplot(m1_pois, plot = FALSE)
w2 <- wormplot(m2_pois, plot = FALSE)

## plot combined wormplot as "ggplot2" graphic
ggplot2::autoplot(c(w1, w2), single_graph = TRUE, col = c(1, 2), fill = c(1, 2))
```

Index