Package 'distributions3'

August 20, 2026

Title Probability Distributions as S3 Objects

Version 0.3.0

Description Tools to create and manipulate probability distributions using S3. Generics pdf(), cdf(), quantile(), and random() provide replacements for base R's d/p/q/r style functions. Functions and arguments have been named carefully to minimize confusion for students in intro stats courses. The documentation for each distribution contains detailed mathematical notes.

License MIT + file LICENSE

URL https://github.com/zeileis/distributions3, https://zeileis.github.io/distributions3/

BugReports https://github.com/zeileis/distributions3/issues

Depends R (>= 3.0.2)

Imports rlang

Suggests cowplot, crch (>= 1.1-1), ggplot2 (>= 3.4.0), knitr, numDeriv, PoissonBinomial, revdbayes (>= 1.3.5), rmarkdown, scoringRules, testthat (>= 3.0.0), tibble, vctrs

VignetteBuilder knitr

Encoding UTF-8

Config/testthat/edition 3

Config/testthat/parallel true

Config/roxygen2/version 8.0.0

NeedsCompilation yes

Author Alex Hayes [aut] (ORCID: <https://orcid.org/0000-0002-4985-5160>), Ralph Moller-Trane [aut] (ORCID: <https://orcid.org/0000-0001-7824-1785>), Daniel Jordan [aut], Paul Northrop [aut] (ORCID: <https://orcid.org/0000-0002-1992-4882>), Moritz N. Lang [aut] (ORCID: <https://orcid.org/0000-0002-2533-9903>), Reto Stauffer [aut] (ORCID: <https://orcid.org/0000-0002-3798-5507>),

Achim Zeileis [aut, cre] (ORCID:
<https://orcid.org/0000-0003-0918-3766>),
Emil Hvitfeldt [ctb] (ORCID: <https://orcid.org/0000-0002-0679-1945>),
Bruna Wundervald [ctb] (ORCID: <https://orcid.org/0000-0001-8163-220X>),
Alessandro Gasparini [ctb] (ORCID:
<https://orcid.org/0000-0002-8319-7624>)

Maintainer Achim Zeileis <Achim.Zeileis@R-project.org>

Repository CRAN

Date/Publication 2026-08-20 05:21:11 UTC

Contents

apply_dpqr . . . 7
Bernoulli . . . 10
Beta . . . 12
Binomial . . . 13
Categorical . . . 15
Cauchy . . . 16
cdf . . . 17
cdf.Bernoulli . . . 18
cdf.Beta . . . 19
cdf.Binomial . . . 20
cdf.Categorical . . . 21
cdf.Cauchy . . . 23
cdf.ChiSquare . . . 24
cdf.Empirical . . . 25
cdf.Erlang . . . 27
cdf.Exponential . . . 28
cdf.FisherF . . . 30
cdf.Frechet . . . 31
cdf.Gamma . . . 32
cdf.Geometric . . . 33
cdf.GEV . . . 34
cdf.GP . . . 35
cdf.Gumbel . . . 36
cdf.HurdleNegativeBinomial . . . 37
cdf.HurdlePoisson . . . 38
cdf.HyperGeometric . . . 39
cdf.Logistic . . . 40
cdf.LogNormal . . . 42
cdf.NegativeBinomial . . . 43
cdf.Normal . . . 44
cdf.Poisson . . . 46
cdf.PoissonBinomial . . . 48
cdf.RevWeibull . . . 49
cdf.SinhArcsinh . . . 50
cdf.StudentsT . . . 52

cdf.Tukey . . . 54
cdf.Uniform . . . 55
cdf.Weibull . . . 56
cdf.ZINegativeBinomial . . . 57
cdf.ZIPoisson . . . 58
cdf.ZTNegativeBinomial . . . 59
cdf.ZTPoisson . . . 60
ChiSquare . . . 62
crps.distribution . . . 63
dempirical . . . 67
dhnbinom . . . 69
dhpois . . . 70
dsinharcsinh . . . 71
dzinbinom . . . 73
dzipois . . . 74
dztnbinom . . . 75
dztpois . . . 76
Empirical . . . 78
Erlang . . . 81
Exponential . . . 82
FIFA2018 . . . 84
FisherF . . . 86
fit_mle . . . 87
fit_mle.Bernoulli . . . 88
fit_mle.Binomial . . . 88
fit_mle.Exponential . . . 89
fit_mle.Gamma . . . 89
fit_mle.Geometric . . . 90
fit_mle.LogNormal . . . 90
fit_mle.Normal . . . 91
fit_mle.Poisson . . . 91
Frechet . . . 92
Gamma . . . 93
Geometric . . . 95
GEV . . . 96
GP . . . 98
Gumbel . . . 100
HurdleNegativeBinomial . . . 101
HurdlePoisson . . . 103
HyperGeometric . . . 104
is_discrete . . . 106
is_distribution . . . 107
Logistic . . . 107
LogNormal . . . 109
log_likelihood . . . 110
mean.distribution . . . 111
Multinomial . . . 113
NegativeBinomial . . . 115

Normal . . . 116
pdf . . . 119
pdf.Bernoulli . . . 120
pdf.Beta . . . 122
pdf.Binomial . . . 123
pdf.Categorical . . . 124
pdf.Cauchy . . . 125
pdf.ChiSquare . . . 126
pdf.distribution . . . 128
pdf.Empirical . . . 135
pdf.Erlang . . . 137
pdf.Exponential . . . 138
pdf.FisherF . . . 139
pdf.Frechet . . . 140
pdf.Gamma . . . 142
pdf.Geometric . . . 143
pdf.GEV . . . 144
pdf.GP . . . 145
pdf.Gumbel . . . 146
pdf.HurdleNegativeBinomial . . . 147
pdf.HurdlePoisson . . . 148
pdf.HyperGeometric . . . 150
pdf.Logistic . . . 151
pdf.LogNormal . . . 152
pdf.Multinomial . . . 153
pdf.NegativeBinomial . . . 154
pdf.Normal . . . 156
pdf.Poisson . . . 158
pdf.PoissonBinomial . . . 159
pdf.RevWeibull . . . 161
pdf.SinhArcsinh . . . 162
pdf.StudentsT . . . 164
pdf.Uniform . . . 166
pdf.Weibull . . . 167
pdf.ZINegativeBinomial . . . 168
pdf.ZIPoisson . . . 169
pdf.ZTNegativeBinomial . . . 170
pdf.ZTPoisson . . . 172
plot.distribution . . . 173
plot_cdf . . . 175
plot_pdf . . . 176
Poisson . . . 176
PoissonBinomial . . . 178
prodist . . . 180
quantile.Bernoulli . . . 183
quantile.Beta . . . 184
quantile.Binomial . . . 185
quantile.Categorical . . . 186

quantile.Cauchy . . . 187
quantile.ChiSquare . . . 188
quantile.Empirical . . . 189
quantile.Erlang . . . 192
quantile.Exponential . . . 193
quantile.FisherF . . . 194
quantile.Frechet . . . 195
quantile.Gamma . . . 196
quantile.Geometric . . . 197
quantile.GEV . . . 198
quantile.GP . . . 199
quantile.Gumbel . . . 200
quantile.HurdleNegativeBinomial . . . 201
quantile.HurdlePoisson . . . 202
quantile.HyperGeometric . . . 203
quantile.Logistic . . . 204
quantile.LogNormal . . . 205
quantile.NegativeBinomial . . . 206
quantile.Normal . . . 208
quantile.Poisson . . . 210
quantile.PoissonBinomial . . . 211
quantile.RevWeibull . . . 213
quantile.SinhArcsinh . . . 214
quantile.StudentsT . . . 216
quantile.Tukey . . . 218
quantile.Uniform . . . 219
quantile.Weibull . . . 220
quantile.ZINegativeBinomial . . . 221
quantile.ZIPoisson . . . 222
quantile.ZTNegativeBinomial . . . 223
quantile.ZTPoisson . . . 224
random . . . 225
random.Bernoulli . . . 227
random.Beta . . . 228
random.Binomial . . . 229
random.Categorical . . . 230
random.Cauchy . . . 231
random.ChiSquare . . . 232
random.distribution . . . 233
random.Empirical . . . 234
random.Erlang . . . 236
random.Exponential . . . 237
random.FisherF . . . 238
random.Frechet . . . 239
random.Gamma . . . 240
random.Geometric . . . 241
random.GEV . . . 242
random.GP . . . 243

random.Gumbel . . . 244
random.HurdleNegativeBinomial . . . 245
random.HurdlePoisson . . . 246
random.HyperGeometric . . . 247
random.Logistic . . . 248
random.LogNormal . . . 249
random.Multinomial . . . 250
random.NegativeBinomial . . . 251
random.Normal . . . 252
random.Poisson . . . 254
random.PoissonBinomial . . . 255
random.RevWeibull . . . 256
random.SinhArcsinh . . . 257
random.StudentsT . . . 258
random.Tukey . . . 260
random.Uniform . . . 261
random.Weibull . . . 262
random.ZINegativeBinomial . . . 263
random.ZIPoisson . . . 264
random.ZTNegativeBinomial . . . 265
random.ZTPoisson . . . 266
RevWeibull . . . 267
score-hessian . . . 268
simulate.default . . . 271
SinhArcsinh . . . 272
stat_auc . . . 274
StudentsT . . . 277
suff_stat . . . 279
suff_stat.Bernoulli . . . 280
suff_stat.Binomial . . . 280
suff_stat.Exponential . . . 281
suff_stat.Gamma . . . 282
suff_stat.Geometric . . . 282
suff_stat.LogNormal . . . 283
suff_stat.Normal . . . 284
suff_stat.Poisson . . . 284
support . . . 285
support.Bernoulli . . . 286
support.Beta . . . 286
support.Binomial . . . 287
support.Cauchy . . . 287
support.ChiSquare . . . 288
support.Empirical . . . 288
support.Erlang . . . 289
support.Exponential . . . 289
support.FisherF . . . 290
support.Frechet . . . 290
support.Gamma . . . 291

support.Geometric . . . 292
support.GEV . . . 292
support.GP . . . 293
support.Gumbel . . . 293
support.HurdleNegativeBinomial . . . 294
support.HurdlePoisson . . . 294
support.HyperGeometric . . . 295
support.Logistic . . . 295
support.LogNormal . . . 296
support.NegativeBinomial . . . 296
support.Normal . . . 297
support.Poisson . . . 297
support.PoissonBinomial . . . 298
support.RevWeibull . . . 298
support.SinhArcsinh . . . 299
support.StudentsT . . . 299
support.Tukey . . . 300
support.Uniform . . . 300
support.Weibull . . . 301
support.ZINegativeBinomial . . . 301
support.ZIPoisson . . . 302
support.ZTNegativeBinomial . . . 302
support.ZTPoisson . . . 303
Tukey . . . 303
Uniform . . . 304
variance . . . 305
Weibull . . . 306
ZINegativeBinomial . . . 307
ZIPoisson . . . 309
ZTNegativeBinomial . . . 311
ZTPoisson . . . 312

Index **315**

`apply_dpqr` *Utilities for* `distributions3` *objects*

Description

Various utility functions to implement methods for distributions with a unified workflow, in particular to facilitate working with vectorized `distributions3` objects. These are particularly useful in the computation of densities, probabilities, quantiles, and random samples when classical d/p/q/r functions are readily available for the distribution of interest.

Usage

```
apply_dpqr(d, FUN, at, elementwise = NULL, drop = TRUE, type = NULL, ...)

make_support(min, max, d, drop = TRUE)

make_positive_integer(n)
```

Arguments

d	A distributions3 object.
FUN	Function to be computed. Function should be of type FUN(at, d), where at is the argument at which the function should be evaluated (e.g., a quantile, probability, or sample size) and d is a distributions3 object.
at	Specification of values at which FUN should be evaluated, typically a numeric vector (e.g., of quantiles, probabilities, etc.) but possibly also a matrix or data frame.
elementwise	logical. Should each element of d 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 length of d and at is the same 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.
drop	logical. Should the result be simplified to a vector if possible (by dropping the dimension attribute)? If FALSE a matrix is always returned.
type	Character string used for naming, typically one of "density", "logLik", "probability", "quantile", and "random". Note that the "random" case is processed differently internally in order to vectorize the random number generation more efficiently.
...	Arguments to be passed to FUN.
min, max	Numeric vectors. Minima and maxima of the supports of a distributions3 object.
n	numeric. Number of observations for computing random draws. If length(n) > 1, the length is taken to be the number required (consistent with base R as, e.g., for rnorm()).

Examples

```
## Implementing a new distribution based on the provided utility functions
## Illustration: Gaussian distribution
## Note: Gaussian() is really just a copy of Normal() with a different class/distribution name


## Generator function for the distribution object.
Gaussian <- function(mu = 0, sigma = 1) {
  stopifnot(
    "parameter lengths do not match (only scalars are allowed to be recycled)" =
      length(mu) == length(sigma) | length(mu) == 1 | length(sigma) == 1
```

```
  )
  d <- data.frame(mu = mu, sigma = sigma)
  class(d) <- c("Gaussian", "distribution")
  d
}

## Set up a vector Y containing four Gaussian distributions:
Y <- Gaussian(mu = 1:4, sigma = c(1, 1, 2, 2))
Y

## Extract the underlying parameters:
as.matrix(Y)


## Extractor functions for moments of the distribution include
## mean(), variance(), skewness(), kurtosis().
## These can be typically be defined as functions of the list of parameters.
mean.Gaussian <- function(x, ...) {
  rlang::check_dots_used()
  setNames(x$mu, names(x))
}
## Analogously for other moments, see distributions3:::variance.Normal etc.

mean(Y)


## The support() method should return a matrix of "min" and "max" for the
## distribution. The make_support() function helps to set the right names and
## dimension.
support.Gaussian <- function(d, drop = TRUE, ...) {
  min <- rep(-Inf, length(d))
  max <- rep(Inf, length(d))
  make_support(min, max, d, drop = drop)
}

support(Y)


## Evaluating certain functions associated with the distribution, e.g.,
## pdf(), log_pdf(), cdf() quantile(), random(), etc. The apply_dpqr()
## function helps to call the typical d/p/q/r functions (like dnorm,
## pnorm, etc.) and set suitable names and dimension.
pdf.Gaussian <- function(d, x, elementwise = NULL, drop = TRUE, ...) {
  FUN <- function(at, d) dnorm(x = at, mean = d$mu, sd = d$sigma, ...)
 apply_dpqr(d = d, FUN = FUN, at = x, type = "density", elementwise = elementwise, drop = drop)
}

## Evaluate all densities at the same argument (returns vector):
pdf(Y, 0)

## Evaluate all densities at several arguments (returns matrix):
pdf(Y, c(0, 5))
```

```
## Evaluate each density at a different argument (returns vector):
pdf(Y, 4:1)

## Force evaluation of each density at a different argument (returns vector)
## or at all arguments (returns matrix):
pdf(Y, 4:1, elementwise = TRUE)
pdf(Y, 4:1, elementwise = FALSE)

## Drawing random() samples also uses apply_dpqr() with the argument
## n assured to be a positive integer.
random.Gaussian <- function(x, n = 1L, drop = TRUE, ...) {
  n <- make_positive_integer(n)
  if (n == 0L) {
    return(numeric(0L))
  }
  FUN <- function(at, d) rnorm(n = at, mean = d$mu, sd = d$sigma)
  apply_dpqr(d = x, FUN = FUN, at = n, type = "random", drop = drop)
}

## One random sample for each distribution (returns vector):
random(Y, 1)

## Several random samples for each distribution (returns matrix):
random(Y, 3)


## For further analogous methods see the "Normal" distribution provided
## in distributions3.
methods(class = "Normal")
```

`Bernoulli` *Create a Bernoulli distribution*

Description

Bernoulli distributions are used to represent events like coin flips when there is single trial that is either successful or unsuccessful. The Bernoulli distribution is a special case of the `Binomial()` distribution with `n = 1`.

Usage

```
Bernoulli(p = 0.5)
```

Arguments

`p`	The success probability for the distribution. `p` can be any value in `[0, 1]`, and defaults to `0.5`.

Details

We recommend reading this documentation on https://zeileis.github.io/distributions3/, where the math will render with additional detail.

In the following, let X be a Bernoulli random variable with parameter $p = p$. Some textbooks also define $q = 1 - p$, or use π instead of p.

The Bernoulli probability distribution is widely used to model binary variables, such as 'failure' and 'success'. The most typical example is the flip of a coin, when p is thought as the probability of flipping a head, and $q = 1 - p$ is the probability of flipping a tail.

Support: $\{0, 1\}$

Mean: p

Variance: $p \cdot (1 - p) = p \cdot q$

Probability mass function (p.m.f):

$$P(X = x) = p^x (1 - p)^{1-x} = p^x q^{1-x}$$

Cumulative distribution function (c.d.f):

$$P(X \leq x) = \begin{cases} 0 & x < 0 \\ 1 - p & 0 \leq x < 1 \\ 1 & x \geq 1 \end{cases}$$

Moment generating function (m.g.f):

$$E(e^{tX}) = (1 - p) + pe^t$$

Value

A `Bernoulli` object.

See Also

Other discrete distributions: `Binomial()`, `Categorical()`, `Geometric()`, `HurdleNegativeBinomial()`, `HurdlePoisson()`, `HyperGeometric()`, `Multinomial()`, `NegativeBinomial()`, `Poisson()`, `PoissonBinomial()`, `ZINegativeBinomial()`, `ZIPoisson()`, `ZTNegativeBinomial()`, `ZTPoisson()`

Examples

```
set.seed(27)

X <- Bernoulli(0.7)
X

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

```
random(X, 10)
pdf(X, 1)
log_pdf(X, 1)
cdf(X, 0)
quantile(X, 0.7)

cdf(X, quantile(X, 0.7))
quantile(X, cdf(X, 0.7))
```

`Beta` *Create a Beta distribution*

Description

Create a Beta distribution

Usage

```
Beta(alpha = 1, beta = 1)
```

Arguments

`alpha`	The alpha parameter. `alpha` can be any value strictly greater than zero. Defaults to 1.
`beta`	The beta parameter. `beta` can be any value strictly greater than zero. Defaults to 1.

Value

A `beta` object.

See Also

Other continuous distributions: `Cauchy()`, `ChiSquare()`, `Erlang()`, `Exponential()`, `FisherF()`, `Frechet()`, `GEV()`, `GP()`, `Gamma()`, `Gumbel()`, `LogNormal()`, `Logistic()`, `Normal()`, `RevWeibull()`, `SinhArcsinh()`, `StudentsT()`, `Tukey()`, `Uniform()`, `Weibull()`

Examples

```
set.seed(27)

X <- Beta(1, 2)
X

random(X, 10)

pdf(X, 0.7)
log_pdf(X, 0.7)
```

```
cdf(X, 0.7)
quantile(X, 0.7)

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

cdf(X, quantile(X, 0.7))
quantile(X, cdf(X, 0.7))
```

`Binomial` *Create a Binomial distribution*

Description

Binomial distributions are used to represent situations can that can be thought as the result of n Bernoulli experiments (here the n is defined as the `size` of the experiment). The classical example is n independent coin flips, where each coin flip has probability `p` of success. In this case, the individual probability of flipping heads or tails is given by the Bernoulli(p) distribution, and the probability of having x equal results (x heads, for example), in n trials is given by the Binomial(n, p) distribution. The equation of the Binomial distribution is directly derived from the equation of the Bernoulli distribution.

Usage

```
Binomial(size = numeric(), p = 0.5)
```

Arguments

`size`	The number of trials. Must be an integer greater than or equal to one. When `size` = `1L`, the Binomial distribution reduces to the bernoulli distribution. Often called `n` in textbooks.
`p`	The success probability for a given trial. `p` can be any value in `[0, 1]`, and defaults to `0.5`.

Details

The Binomial distribution comes up when you are interested in the portion of people who do a thing. The Binomial distribution also comes up in the sign test, sometimes called the Binomial test (see `stats::binom.test()`), where you may need the Binomial C.D.F. to compute p-values.

We recommend reading this documentation on https://zeileis.github.io/distributions3/, where the math will render with additional detail.

In the following, let X be a Binomial random variable with parameter `size` = n and `p` = p. Some textbooks define $q = 1 - p$, or called π instead of p.

Support: $\{0, 1, 2, ..., n\}$

Mean: np

Variance: $np \cdot (1-p) = np \cdot q$

Probability mass function (p.m.f):

$$P(X = k) = \binom{n}{k} p^k (1-p)^{n-k}$$

Cumulative distribution function (c.d.f):

$$P(X \leq k) = \sum_{i=0}^{\lfloor k \rfloor} \binom{n}{i} p^i (1-p)^{n-i}$$

Moment generating function (m.g.f):

$$E(e^{tX}) = (1 - p + pe^t)^n$$

Value

A `Binomial` object.

See Also

Other discrete distributions: `Bernoulli()`, `Categorical()`, `Geometric()`, `HurdleNegativeBinomial()`, `HurdlePoisson()`, `HyperGeometric()`, `Multinomial()`, `NegativeBinomial()`, `Poisson()`, `PoissonBinomial()`, `ZINegativeBinomial()`, `ZIPoisson()`, `ZTNegativeBinomial()`, `ZTPoisson()`

Examples

```
set.seed(27)

X <- Binomial(10, 0.2)
X

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

random(X, 10)

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

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

cdf(X, quantile(X, 0.7))
quantile(X, cdf(X, 7))
```

`Categorical` *Create a Categorical distribution*

Description

Create a Categorical distribution

Usage

```
Categorical(outcomes = numeric(), p = NULL)
```

Arguments

`outcomes`	A vector specifying the elements in the sample space. Can be numeric, factor, character, or logical.
`p`	A vector of success probabilities for each outcome. Each element of `p` can be any positive value – the vector gets normalized internally. Defaults to `NULL`, in which case the distribution is assumed to be uniform.

Value

A `Categorical` object.

See Also

Other discrete distributions: `Bernoulli()`, `Binomial()`, `Geometric()`, `HurdleNegativeBinomial()`, `HurdlePoisson()`, `HyperGeometric()`, `Multinomial()`, `NegativeBinomial()`, `Poisson()`, `PoissonBinomial()`, `ZINegativeBinomial()`, `ZIPoisson()`, `ZTNegativeBinomial()`, `ZTPoisson()`

Examples

```
set.seed(27)

X <- Categorical(1:3, p = c(0.4, 0.1, 0.5))
X

Y <- Categorical(LETTERS[1:4])
Y

random(X, 10)
random(Y, 10)

pdf(X, 1)
log_pdf(X, 1)

cdf(X, 1)
quantile(X, 0.5)

# cdfs are only defined for numeric sample spaces. this errors!
```

```
# cdf(Y, "a")

# same for quantiles. this also errors!
# quantile(Y, 0.7)
```

`Cauchy` *Create a Cauchy distribution*

Description

Note that the Cauchy distribution is the student's t distribution with one degree of freedom. The Cauchy distribution does not have a well defined mean or variance. Cauchy distributions often appear as priors in Bayesian contexts due to their heavy tails.

Usage

```
Cauchy(location = 0, scale = 1)
```

Arguments

`location`	The location parameter. Can be any real number. Defaults to `0`.
`scale`	The scale parameter. Must be greater than zero (?). Defaults to `1`.

Details

We recommend reading this documentation on https://zeileis.github.io/distributions3/, where the math will render with additional detail and much greater clarity.

In the following, let X be a Cauchy variable with mean `location` = x_0 and `scale` = γ.

Support: R, the set of all real numbers

Mean: Undefined.

Variance: Undefined.

Probability density function (p.d.f):

$$f(x) = \frac{1}{\pi\gamma\left[1 + \left(\frac{x-x_0}{\gamma}\right)^2\right]}$$

Cumulative distribution function (c.d.f):

$$F(t) = \frac{1}{\pi}\arctan\left(\frac{t-x_0}{\gamma}\right) + \frac{1}{2}$$

Moment generating function (m.g.f):

Does not exist.

Value

A Cauchy object.

See Also

Other continuous distributions: Beta(), ChiSquare(), Erlang(), Exponential(), FisherF(), Frechet(), GEV(), GP(), Gamma(), Gumbel(), LogNormal(), Logistic(), Normal(), RevWeibull(), SinhArcsinh(), StudentsT(), Tukey(), Uniform(), Weibull()

Examples

```
set.seed(27)

X <- Cauchy(10, 0.2)
X

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

random(X, 10)

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

cdf(X, 2)
quantile(X, 0.7)

cdf(X, quantile(X, 0.7))
quantile(X, cdf(X, 7))
```

---

cdf *Evaluate the cumulative distribution function of a probability distribution*

Description

Generic function for computing probabilities from distribution objects based on the cumulative distribution function (CDF).

Usage

```
cdf(d, x, drop = TRUE, ...)
```

Arguments

d	An object. The package provides methods for distribution objects such as those from Normal() or Binomial() etc.
x	A vector of elements whose cumulative probabilities you would like to determine given the distribution d.
drop	logical. Should the result be simplified to a vector if possible?
...	Arguments passed to methods. Unevaluated arguments will generate a warning to catch mispellings or other possible errors.

Value

Probabilities corresponding to the vector x.

Examples

```
## distribution object
X <- Normal()
## probabilities from CDF
cdf(X, c(1, 2, 3, 4, 5))
```

---

cdf.Bernoulli *Evaluate the cumulative distribution function of a Bernoulli distribution*

Description

Evaluate the cumulative distribution function of a Bernoulli distribution

Usage

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

Arguments

d	A Bernoulli object created by a call to Bernoulli().
x	A vector of elements whose cumulative probabilities you would like to determine given the distribution d.
drop	logical. Should the result be simplified to a vector if possible?
elementwise	logical. Should each distribution in d be evaluated at all elements of x (elementwise = FALSE, yielding a matrix)? Or, if d 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.
...	Arguments to be passed to pbinom. Unevaluated arguments will generate a warning to catch mispellings or other possible errors.

Value

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.

Examples

```
set.seed(27)

X <- Bernoulli(0.7)
X

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

random(X, 10)
pdf(X, 1)
log_pdf(X, 1)
cdf(X, 0)
quantile(X, 0.7)

cdf(X, quantile(X, 0.7))
quantile(X, cdf(X, 0.7))
```

---

cdf.Beta *Evaluate the cumulative distribution function of a Beta distribution*

Description

Evaluate the cumulative distribution function of a Beta distribution

Usage

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

Arguments

d	A Beta object created by a call to Beta().
x	A vector of elements whose cumulative probabilities you would like to determine given the distribution d.
drop	logical. Should the result be simplified to a vector if possible?
elementwise	logical. Should each distribution in d be evaluated at all elements of x (elementwise = FALSE, yielding a matrix)? Or, if d 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.

...	Arguments to be passed to pbeta. Unevaluated arguments will generate a warning to catch mispellings or other possible errors.

Value

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.

Examples

```
set.seed(27)

X <- Beta(1, 2)
X

random(X, 10)

pdf(X, 0.7)
log_pdf(X, 0.7)

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

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

cdf(X, quantile(X, 0.7))
quantile(X, cdf(X, 0.7))
```

---

cdf.Binomial *Evaluate the cumulative distribution function of a Binomial distribution*

Description

Evaluate the cumulative distribution function of a Binomial distribution

Usage

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

Arguments

d	A Binomial object created by a call to Binomial().
x	A vector of elements whose cumulative probabilities you would like to determine given the distribution d.
drop	logical. Should the result be simplified to a vector if possible?
elementwise	logical. Should each distribution in d be evaluated at all elements of x (elementwise = FALSE, yielding a matrix)? Or, if d 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.
...	Arguments to be passed to pbinom. Unevaluated arguments will generate a warning to catch mispellings or other possible errors.

Value

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.

Examples

```
set.seed(27)

X <- Binomial(10, 0.2)
X

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

random(X, 10)

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

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

cdf(X, quantile(X, 0.7))
quantile(X, cdf(X, 7))
```

---

cdf.Categorical *Evaluate the cumulative distribution function of a Categorical distribution*

Description

Evaluate the cumulative distribution function of a Categorical distribution

Usage

```
## S3 method for class 'Categorical'
cdf(d, x, ...)
```

Arguments

d	A Categorical object created by a call to Categorical().
x	A vector of elements whose cumulative probabilities you would like to determine given the distribution d.
...	Unused. Unevaluated arguments will generate a warning to catch mispellings or other possible errors.

Value

A vector of probabilities, one for each element of x.

Examples

```
set.seed(27)

X <- Categorical(1:3, p = c(0.4, 0.1, 0.5))
X

Y <- Categorical(LETTERS[1:4])
Y

random(X, 10)
random(Y, 10)

pdf(X, 1)
log_pdf(X, 1)

cdf(X, 1)
quantile(X, 0.5)

# cdfs are only defined for numeric sample spaces. this errors!
# cdf(Y, "a")

# same for quantiles. this also errors!
# quantile(Y, 0.7)
```

cdf.Cauchy *Evaluate the cumulative distribution function of a Cauchy distribution*

Description

Evaluate the cumulative distribution function of a Cauchy distribution

Usage

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

Arguments

d	A Cauchy object created by a call to Cauchy().
x	A vector of elements whose cumulative probabilities you would like to determine given the distribution d.
drop	logical. Should the result be simplified to a vector if possible?
elementwise	logical. Should each distribution in d be evaluated at all elements of x (elementwise = FALSE, yielding a matrix)? Or, if d 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.
...	Arguments to be passed to pcauchy. Unevaluated arguments will generate a warning to catch mispellings or other possible errors.

Value

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.

Examples

```
set.seed(27)

X <- Cauchy(10, 0.2)
X

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

random(X, 10)

pdf(X, 2)
```

```
log_pdf(X, 2)

cdf(X, 2)
quantile(X, 0.7)

cdf(X, quantile(X, 0.7))
quantile(X, cdf(X, 7))
```

`cdf.ChiSquare` *Evaluate the cumulative distribution function of a chi square distribution*

Description

Evaluate the cumulative distribution function of a chi square distribution

Usage

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

Arguments

`d`	A `ChiSquare` object created by a call to `ChiSquare()`.
`x`	A vector of elements whose cumulative probabilities you would like to determine given the distribution `d`.
`drop`	logical. Should the result be simplified to a vector if possible?
`elementwise`	logical. Should each distribution in `d` be evaluated at all elements of `x` (`elementwise = FALSE`, yielding a matrix)? Or, if `d` 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.
`...`	Arguments to be passed to `pchisq`. Unevaluated arguments will generate a warning to catch mispellings or other possible errors.

Value

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.

Examples

```
set.seed(27)

X <- ChiSquare(5)
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)

cdf(X, quantile(X, 0.7))
quantile(X, cdf(X, 7))
```

`cdf.Empirical` *Evaluate the cumulative distribution function of an Empirical distribution*

Description

Please see the documentation of `Empirical()` for some properties of the Empirical distribution.

Usage

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

Arguments

`d`	An `Empirical` object created by a call to `Empirical()`.
`x`	A vector of elements whose cumulative probabilities you would like to determine given the distribution `d`.
`drop`	logical. Should the result be simplified to a vector if possible?
`elementwise`	logical. Should each distribution in `d` be evaluated at all elements of `x` (`elementwise = FALSE`, yielding a matrix)? Or, if `d` 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.
`...`	arguments to be passed to `pempirical()`.

Value

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.

See Also

Other Empirical distribution: `Empirical()`, `dempirical()`, `pdf.Empirical()`, `quantile.Empirical()`, `random.Empirical()`, `support.Empirical()`

Examples

```
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
Y4 <- matrix(rnorm(20), ncol = 5,
             dimnames = list(paste0("D_", 1:4), paste0("obs_", 1:5)))
d4 <- Empirical(Y4)
d4

## Data frame
d5 <- Empirical(as.data.frame(Y4))
d5

identical(d4, d5)

mean(d5)
variance(d5)
skewness(d5)
kurtosis(d5)

pdf(d5, c(-0.5, 0, 0.5, 1)) # Defaults to elementwise = TRUE
pdf(d5, c(-0.5, 0, 0.5, 1), elementwise = FALSE)

cdf(d5, c(-0.5, 0, 0.5, 1)) # Defaults to elementwise = TRUE
cdf(d5, c(-0.5, 0, 0.5, 1), elementwise = FALSE)

quantile(d5, c(0.2, 0.4, 0.6, 0.8)) # Defaults to elementwise = TRUE
quantile(d5, c(0.2, 0.4, 0.6, 0.8), elementwise = FALSE)

## The quantile function is the inverse of the distribution
## function (cdf) if x in Y
set.seed(6020)
Y <- round(rlnorm(20, log(3), log(2)), 1)
d <- Empirical(Y)

cdf(d, 4.0)
quantile(d, cdf(d, 4.0))
```

`cdf.Erlang` *Evaluate the cumulative distribution function of an Erlang distribution*

Description

Evaluate the cumulative distribution function of an Erlang distribution

Usage

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

Arguments

d	An Erlang object created by a call to Erlang().
x	A vector of elements whose cumulative probabilities you would like to determine given the distribution d.
drop	logical. Should the result be simplified to a vector if possible?
elementwise	logical. Should each distribution in d be evaluated at all elements of x (elementwise = FALSE, yielding a matrix)? Or, if d 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.
...	Arguments to be passed to pgamma. Unevaluated arguments will generate a warning to catch mispellings or other possible errors.

Value

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.

Examples

```
set.seed(27)

X <- Erlang(5, 2)
X

random(X, 10)

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

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

cdf(X, quantile(X, 0.7))
quantile(X, cdf(X, 7))
```

---

cdf.Exponential *Evaluate the cumulative distribution function of an Exponential distribution*

Description

Evaluate the cumulative distribution function of an Exponential distribution

Usage

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

Arguments

d	An Exponential object created by a call to Exponential().
x	A vector of elements whose cumulative probabilities you would like to determine given the distribution d.
drop	logical. Should the result be simplified to a vector if possible?
elementwise	logical. Should each distribution in d be evaluated at all elements of x (elementwise = FALSE, yielding a matrix)? Or, if d 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.
...	Arguments to be passed to pexp. Unevaluated arguments will generate a warning to catch mispellings or other possible errors.

Value

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.

Examples

```
set.seed(27)

X <- Exponential(5)
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)

cdf(X, quantile(X, 0.7))
quantile(X, cdf(X, 7))
```

`cdf.FisherF` *Evaluate the cumulative distribution function of an F distribution*

Description

Evaluate the cumulative distribution function of an F distribution

Usage

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

Arguments

`d`	A `FisherF` object created by a call to `FisherF()`.
`x`	A vector of elements whose cumulative probabilities you would like to determine given the distribution `d`.
`drop`	logical. Should the result be simplified to a vector if possible?
`elementwise`	logical. Should each distribution in `d` be evaluated at all elements of `x` (`elementwise = FALSE`, yielding a matrix)? Or, if `d` 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.
`...`	Arguments to be passed to `pf`. Unevaluated arguments will generate a warning to catch mispellings or other possible errors.

Value

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.

Examples

```
set.seed(27)

X <- FisherF(5, 10, 0.2)
X

random(X, 10)

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

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

```
cdf(X, quantile(X, 0.7))
quantile(X, cdf(X, 7))
```

`cdf.Frechet` *Evaluate the cumulative distribution function of a Frechet distribution*

Description

Evaluate the cumulative distribution function of a Frechet distribution

Usage

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

Arguments

`d`	A `Frechet` object created by a call to `Frechet()`.
`x`	A vector of elements whose cumulative probabilities you would like to determine given the distribution `d`.
`drop`	logical. Should the result be simplified to a vector if possible?
`elementwise`	logical. Should each distribution in `d` be evaluated at all elements of `x` (`elementwise = FALSE`, yielding a matrix)? Or, if `d` 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.
`...`	Arguments to be passed to `pgev`. Unevaluated arguments will generate a warning to catch mispellings or other possible errors.

Value

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.

Examples

```
set.seed(27)

X <- Frechet(0, 2)
X

random(X, 10)

pdf(X, 0.7)
log_pdf(X, 0.7)
```

```
cdf(X, 0.7)
quantile(X, 0.7)

cdf(X, quantile(X, 0.7))
quantile(X, cdf(X, 0.7))
```

`cdf.Gamma` *Evaluate the cumulative distribution function of a Gamma distribution*

Description

Evaluate the cumulative distribution function of a Gamma distribution

Usage

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

Arguments

`d`	A `Gamma` object created by a call to `Gamma()`.
`x`	A vector of elements whose cumulative probabilities you would like to determine given the distribution `d`.
`drop`	logical. Should the result be simplified to a vector if possible?
`elementwise`	logical. Should each distribution in `d` be evaluated at all elements of `x` (`elementwise = FALSE`, yielding a matrix)? Or, if `d` 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.
`...`	Arguments to be passed to `pgamma`. Unevaluated arguments will generate a warning to catch mispellings or other possible errors.

Value

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.

Examples

```
set.seed(27)

X <- Gamma(5, 2)
X

random(X, 10)
```

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

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

cdf(X, quantile(X, 0.7))
quantile(X, cdf(X, 7))
```

`cdf.Geometric` *Evaluate the cumulative distribution function of a Geometric distribution*

Description

Evaluate the cumulative distribution function of a Geometric distribution

Usage

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

Arguments

`d`	A `Geometric` object created by a call to `Geometric()`.
`x`	A vector of elements whose cumulative probabilities you would like to determine given the distribution `d`.
`drop`	logical. Should the result be simplified to a vector if possible?
`elementwise`	logical. Should each distribution in `d` be evaluated at all elements of `x` (`elementwise = FALSE`, yielding a matrix)? Or, if `d` 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.
`...`	Arguments to be passed to `pgeom`. Unevaluated arguments will generate a warning to catch mispellings or other possible errors.

Value

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.

See Also

Other Geometric distribution: `pdf.Geometric()`, `quantile.Geometric()`, `random.Geometric()`

Examples

```
set.seed(27)

X <- Geometric(0.3)
X

random(X, 10)

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

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

---

cdf.GEV *Evaluate the cumulative distribution function of a GEV distribution*

Description

Evaluate the cumulative distribution function of a GEV distribution

Usage

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

Arguments

d	A GEV object created by a call to GEV().
x	A vector of elements whose cumulative probabilities you would like to determine given the distribution d.
drop	logical. Should the result be simplified to a vector if possible?
elementwise	logical. Should each distribution in d be evaluated at all elements of x (elementwise = FALSE, yielding a matrix)? Or, if d 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.
...	Arguments to be passed to pgev. Unevaluated arguments will generate a warning to catch mispellings or other possible errors.

Value

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.

Examples

```
set.seed(27)

X <- GEV(1, 2, 0.1)
X

random(X, 10)

pdf(X, 0.7)
log_pdf(X, 0.7)

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

cdf(X, quantile(X, 0.7))
quantile(X, cdf(X, 0.7))
```

`cdf.GP` *Evaluate the cumulative distribution function of a GP distribution*

Description

Evaluate the cumulative distribution function of a GP distribution

Usage

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

Arguments

d	A GP object created by a call to GP().
x	A vector of elements whose cumulative probabilities you would like to determine given the distribution d.
drop	logical. Should the result be simplified to a vector if possible?
elementwise	logical. Should each distribution in d be evaluated at all elements of x (elementwise = FALSE, yielding a matrix)? Or, if d 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.
...	Arguments to be passed to pgp. Unevaluated arguments will generate a warning to catch mispellings or other possible errors.

Value

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.

Examples

```
set.seed(27)

X <- GP(0, 2, 0.1)
X

random(X, 10)

pdf(X, 0.7)
log_pdf(X, 0.7)

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

cdf(X, quantile(X, 0.7))
quantile(X, cdf(X, 0.7))
```

`cdf.Gumbel` *Evaluate the cumulative distribution function of a Gumbel distribution*

Description

Evaluate the cumulative distribution function of a Gumbel distribution

Usage

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

Arguments

d	A Gumbel object created by a call to Gumbel().
x	A vector of elements whose cumulative probabilities you would like to determine given the distribution d.
drop	logical. Should the result be simplified to a vector if possible?
elementwise	logical. Should each distribution in d be evaluated at all elements of x (elementwise = FALSE, yielding a matrix)? Or, if d 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.
...	Arguments to be passed to pgev. Unevaluated arguments will generate a warning to catch mispellings or other possible errors.

Value

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.

Examples

```
set.seed(27)

X <- Gumbel(1, 2)
X

random(X, 10)

pdf(X, 0.7)
log_pdf(X, 0.7)

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

cdf(X, quantile(X, 0.7))
quantile(X, cdf(X, 0.7))
```

`cdf.HurdleNegativeBinomial` *Evaluate the cumulative distribution function of a hurdle negative binomial distribution*

Description

Evaluate the cumulative distribution function of a hurdle negative binomial distribution

Usage

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

Arguments

d	A HurdleNegativeBinomial object created by a call to HurdleNegativeBinomial().
x	A vector of elements whose cumulative probabilities you would like to determine given the distribution d.
drop	logical. Should the result be simplified to a vector if possible?
elementwise	logical. Should each distribution in d be evaluated at all elements of x (elementwise = FALSE, yielding a matrix)? Or, if d 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.
...	Arguments to be passed to phnbinom. Unevaluated arguments will generate a warning to catch mispellings or other possible errors.

Value

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.

Examples

```
## set up a hurdle negative binomial distribution
X <- HurdleNegativeBinomial(mu = 2.5, theta = 1, pi = 0.75)
X

## standard functions
pdf(X, 0:8)
cdf(X, 0:8)
quantile(X, seq(0, 1, by = 0.25))

## cdf() and quantile() are inverses for each other
quantile(X, cdf(X, 3))

## density visualization
plot(0:8, pdf(X, 0:8), type = "h", lwd = 2)

## corresponding sample with histogram of empirical frequencies
set.seed(0)
x <- random(X, 500)
hist(x, breaks = -1:max(x) + 0.5)
```

---

cdf.HurdlePoisson *Evaluate the cumulative distribution function of a hurdle Poisson distribution*

Description

Evaluate the cumulative distribution function of a hurdle Poisson distribution

Usage

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

Arguments

d	A HurdlePoisson object created by a call to HurdlePoisson().
x	A vector of elements whose cumulative probabilities you would like to determine given the distribution d.
drop	logical. Should the result be simplified to a vector if possible?

elementwise	logical. Should each distribution in d be evaluated at all elements of x (elementwise = FALSE, yielding a matrix)? Or, if d 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.
...	Arguments to be passed to phpois. Unevaluated arguments will generate a warning to catch mispellings or other possible errors.

Value

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.

Examples

```
## set up a hurdle Poisson distribution
X <- HurdlePoisson(lambda = 2.5, pi = 0.75)
X

## standard functions
pdf(X, 0:8)
cdf(X, 0:8)
quantile(X, seq(0, 1, by = 0.25))

## cdf() and quantile() are inverses for each other
quantile(X, cdf(X, 3))

## density visualization
plot(0:8, pdf(X, 0:8), type = "h", lwd = 2)

## corresponding sample with histogram of empirical frequencies
set.seed(0)
x <- random(X, 500)
hist(x, breaks = -1:max(x) + 0.5)
```

---

cdf.HyperGeometric *Evaluate the cumulative distribution function of a HyperGeometric distribution*

Description

Evaluate the cumulative distribution function of a HyperGeometric distribution

Usage

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

Arguments

d	A HyperGeometric object created by a call to HyperGeometric().
x	A vector of elements whose cumulative probabilities you would like to determine given the distribution d.
drop	logical. Should the result be simplified to a vector if possible?
elementwise	logical. Should each distribution in d be evaluated at all elements of x (elementwise = FALSE, yielding a matrix)? Or, if d 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.
...	Arguments to be passed to phyper. Unevaluated arguments will generate a warning to catch mispellings or other possible errors.

Value

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.

See Also

Other HyperGeometric distribution: pdf.HyperGeometric(), quantile.HyperGeometric(), random.HyperGeometric()

Examples

```
set.seed(27)

X <- HyperGeometric(4, 5, 8)
X

random(X, 10)

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

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

---

cdf.Logistic *Evaluate the cumulative distribution function of a Logistic distribution*

Description

Evaluate the cumulative distribution function of a Logistic distribution

Usage

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

Arguments

d	A Logistic object created by a call to Logistic().
x	A vector of elements whose cumulative probabilities you would like to determine given the distribution d.
drop	logical. Should the result be simplified to a vector if possible?
elementwise	logical. Should each distribution in d be evaluated at all elements of x (elementwise = FALSE, yielding a matrix)? Or, if d 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.
...	Arguments to be passed to plogis. Unevaluated arguments will generate a warning to catch mispellings or other possible errors.

Value

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.

See Also

Other Logistic distribution: pdf.Logistic(), quantile.Logistic(), random.Logistic()

Examples

```
set.seed(27)

X <- Logistic(2, 4)
X

random(X, 10)

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

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

`cdf.LogNormal` *Evaluate the cumulative distribution function of a LogNormal distribution*

Description

Evaluate the cumulative distribution function of a LogNormal distribution

Usage

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

Arguments

`d`	A `LogNormal` object created by a call to `LogNormal()`.
`x`	A vector of elements whose cumulative probabilities you would like to determine given the distribution `d`.
`drop`	logical. Should the result be simplified to a vector if possible?
`elementwise`	logical. Should each distribution in `d` be evaluated at all elements of `x` (`elementwise = FALSE`, yielding a matrix)? Or, if `d` 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.
`...`	Arguments to be passed to `plnorm`. Unevaluated arguments will generate a warning to catch mispellings or other possible errors.

Value

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.

See Also

Other LogNormal distribution: `fit_mle.LogNormal()`, `pdf.LogNormal()`, `quantile.LogNormal()`, `random.LogNormal()`

Examples

```
set.seed(27)

X <- LogNormal(0.3, 2)
X

random(X, 10)
```

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

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

`cdf.NegativeBinomial` *Evaluate the cumulative distribution function of a negative binomial distribution*

Description

Evaluate the cumulative distribution function of a negative binomial distribution

Usage

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

Arguments

`d`	A `NegativeBinomial` object created by a call to `NegativeBinomial()`.
`x`	A vector of elements whose cumulative probabilities you would like to determine given the distribution `d`.
`drop`	logical. Should the result be simplified to a vector if possible?
`elementwise`	logical. Should each distribution in `d` be evaluated at all elements of `x` (`elementwise = FALSE`, yielding a matrix)? Or, if `d` 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.
`...`	Arguments to be passed to `pnbinom`. Unevaluated arguments will generate a warning to catch mispellings or other possible errors.

Value

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.

See Also

Other NegativeBinomial distribution: `pdf.NegativeBinomial()`, `quantile.NegativeBinomial()`, `random.NegativeBinomial()`

Examples

```
set.seed(27)

X <- NegativeBinomial(size = 5, p = 0.1)
X

random(X, 10)

pdf(X, 50)
log_pdf(X, 50)

cdf(X, 50)
quantile(X, 0.7)

## alternative parameterization of X
Y <- NegativeBinomial(mu = 45, size = 5)
Y
cdf(Y, 50)
quantile(Y, 0.7)
```

`cdf.Normal` *Evaluate the cumulative distribution function of a Normal distribution*

Description

Evaluate the cumulative distribution function of a Normal distribution

Usage

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

Arguments

d	A Normal object created by a call to Normal().
x	A vector of elements whose cumulative probabilities you would like to determine given the distribution d.
drop	logical. Should the result be simplified to a vector if possible?
elementwise	logical. Should each distribution in d be evaluated at all elements of x (elementwise = FALSE, yielding a matrix)? Or, if d 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.
...	Arguments to be passed to pnorm. Unevaluated arguments will generate a warning to catch mispellings or other possible errors.

Value

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.

See Also

Other Normal distribution: `fit_mle.Normal()`, `pdf.Normal()`, `quantile.Normal()`

Examples

```
set.seed(27)

X <- Normal(5, 2)
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: calculating p-values for two-sided Z-test

# here the null hypothesis is H_0: mu = 3
# and we assume sigma = 2

# exactly the same as: Z <- Normal(0, 1)
Z <- Normal()

# data to test
x <- c(3, 7, 11, 0, 7, 0, 4, 5, 6, 2)
nx <- length(x)

# calculate the z-statistic
z_stat <- (mean(x) - 3) / (2 / sqrt(nx))
z_stat

# calculate the two-sided p-value
1 - cdf(Z, abs(z_stat)) + cdf(Z, -abs(z_stat))

# exactly equivalent to the above
2 * cdf(Z, -abs(z_stat))

# p-value for one-sided test
```

```
# H_0: mu <= 3   vs   H_A: mu > 3
1 - cdf(Z, z_stat)

# p-value for one-sided test
# H_0: mu >= 3   vs   H_A: mu < 3
cdf(Z, z_stat)

### example: calculating a 88 percent Z CI for a mean

# same `x` as before, still assume `sigma = 2`

# lower-bound
mean(x) - quantile(Z, 1 - 0.12 / 2) * 2 / sqrt(nx)

# upper-bound
mean(x) + quantile(Z, 1 - 0.12 / 2) * 2 / sqrt(nx)

# equivalent to
mean(x) + c(-1, 1) * quantile(Z, 1 - 0.12 / 2) * 2 / sqrt(nx)

# also equivalent to
mean(x) + quantile(Z, 0.12 / 2) * 2 / sqrt(nx)
mean(x) + quantile(Z, 1 - 0.12 / 2) * 2 / sqrt(nx)

### generating random samples and plugging in ks.test()

set.seed(27)

# generate a random sample
ns <- random(Normal(3, 7), 26)

# test if sample is Normal(3, 7)
ks.test(ns, pnorm, mean = 3, sd = 7)

# test if sample is gamma(8, 3) using base R pgamma()
ks.test(ns, pgamma, shape = 8, rate = 3)

### MISC

# note that the cdf() and quantile() functions are inverses
cdf(X, quantile(X, 0.7))
quantile(X, cdf(X, 7))
```

`cdf.Poisson` *Evaluate the cumulative distribution function of a Poisson distribution*

Description

Evaluate the cumulative distribution function of a Poisson distribution

Usage

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

Arguments

d	A Poisson object created by a call to Poisson().
x	A vector of elements whose cumulative probabilities you would like to determine given the distribution d.
drop	logical. Should the result be simplified to a vector if possible?
elementwise	logical. Should each distribution in d be evaluated at all elements of x (elementwise = FALSE, yielding a matrix)? Or, if d 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.
...	Arguments to be passed to ppois. Unevaluated arguments will generate a warning to catch mispellings or other possible errors.

Value

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.

Examples

```
set.seed(27)

X <- Poisson(2)
X

random(X, 10)

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

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

cdf(X, quantile(X, 0.7))
quantile(X, cdf(X, 7))
```

`cdf.PoissonBinomial` *Evaluate the cumulative distribution function of a PoissonBinomial distribution*

Description

Evaluate the cumulative distribution function of a PoissonBinomial distribution

Usage

```
## S3 method for class 'PoissonBinomial'
cdf(
  d,
  x,
  drop = TRUE,
  elementwise = NULL,
  lower.tail = TRUE,
  log.p = FALSE,
  verbose = TRUE,
  ...
)
```

Arguments

`d`	A `PoissonBinomial` object created by a call to `PoissonBinomial()`.
`x`	A vector of elements whose cumulative probabilities you would like to determine given the distribution `d`.
`drop`	logical. Should the result be simplified to a vector if possible?
`elementwise`	logical. Should each distribution in `d` be evaluated at all elements of `x` (`elementwise = FALSE`, yielding a matrix)? Or, if `d` 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.
`lower.tail, log.p, ...`	Arguments to be passed to `ppbinom` or `pnorm`, respectively.
`verbose`	logical. Should a warning be issued if the normal approximation is applied because the **PoissonBinomial** package is not installed?

Value

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.

Examples

```
set.seed(27)

X <- PoissonBinomial(0.5, 0.3, 0.8)
X

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

random(X, 10)

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

cdf(X, 2)
quantile(X, 0.8)

cdf(X, quantile(X, 0.8))
quantile(X, cdf(X, 2))

## equivalent definitions of four Poisson binomial distributions
## each summing up three Bernoulli probabilities
p <- cbind(
  p1 = c(0.1, 0.2, 0.1, 0.2),
  p2 = c(0.5, 0.5, 0.5, 0.5),
  p3 = c(0.8, 0.7, 0.9, 0.8))
PoissonBinomial(p)
PoissonBinomial(p[, 1], p[, 2], p[, 3])
PoissonBinomial(p[, 1:2], p[, 3])
```

---

cdf.RevWeibull *Evaluate the cumulative distribution function of an RevWeibull distribution*

Description

Evaluate the cumulative distribution function of an RevWeibull distribution

Usage

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

Arguments

d	A RevWeibull object created by a call to RevWeibull().

x	A vector of elements whose cumulative probabilities you would like to determine given the distribution d.
drop	logical. Should the result be simplified to a vector if possible?
elementwise	logical. Should each distribution in d be evaluated at all elements of x (elementwise = FALSE, yielding a matrix)? Or, if d 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.
...	Arguments to be passed to pgev. Unevaluated arguments will generate a warning to catch mispellings or other possible errors.

Value

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.

Examples

```
set.seed(27)

X <- RevWeibull(1, 2)
X

random(X, 10)

pdf(X, 0.7)
log_pdf(X, 0.7)

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

cdf(X, quantile(X, 0.7))
quantile(X, cdf(X, 0.7))
```

---

cdf.SinhArcsinh *Evaluate the cumulative distribution function of a Sinh-Arcsinh distribution*

Description

Evaluate the cumulative distribution function of a Sinh-Arcsinh distribution

Usage

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

Arguments

d	A SinhArcsinh object created by a call to SinhArcsinh().
x	A vector of elements whose cumulative probabilities you would like to determine given the distribution d.
drop	logical. Should the result be simplified to a vector if possible?
elementwise	logical. Should each distribution in d be evaluated at all elements of x (elementwise = FALSE, yielding a matrix)? Or, if d 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.
cores	NULL or integer. Number of cores/threads to use when calling psinharcsinh().
...	Arguments to be passed to pnorm. Unevaluated arguments will generate a warning to catch mispellings or other possible errors.

Value

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.

See Also

Other SinhArcsinh distribution: pdf.SinhArcsinh(), quantile.SinhArcsinh()

Examples

```
## SinhArcsinh() by default uses nu = 1, tau = 1 which
## results in the standard normal distribution
set.seed(6020)
X <- SinhArcsinh() # Uses mu = 1, sigma = 0, nu = 1, tau = 1)
x <- random(X, 300)
qqnorm(x); qqline(x, col = 2, lwd = 2)
curve(pdf(X, x), xlim = c(-5, 5), main = paste(X, "density"))

## Calculation of central moments is based on numeric integration,
## thus not being identical to the standard normal distribution
c(mean = mean(x), sd = sd(x))

## Skewed Sinh-Arcsinh distribution
X <- SinhArcsinh(mu = 7, sigma = 2, nu = c(0.7, 1, 0.7), tau = c(1, 0.7, 0.7))
as.matrix(X)

## Visualization of density functions using different parameters for nu/tau
curve(pdf(X[1], x), xlim = c(0, 20), ylim = c(0, 0.2), main = "Density function")
curve(pdf(X[2], x), xlim = c(0, 20), col = 2, add = TRUE)
curve(pdf(X[3], x), xlim = c(0, 20), col = 4, add = TRUE)

## Visualization of distribution function using different parameters for nu/tau
curve(cdf(X[1], x), xlim = c(0, 20), ylim = 0:1, main = "Distribution function")
```

```
curve(cdf(X[2], x), xlim = c(0, 20), col = 2, add = TRUE)
curve(cdf(X[3], x), xlim = c(0, 20), col = 4, add = TRUE)

## Central moments
mean(X)
variance(X)
skewness(X)
kurtosis(X)

## Drawing random values
random(X, 10)

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

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

# note that the cdf() and quantile() functions are inverses
X <- SinhArcsinh(mu = 3, sigma = 2, nu = 0.9, tau = 1.2)
cdf(X, quantile(X, 0.7))
quantile(X, cdf(X, 7))
```

`cdf.StudentsT` *Evaluate the cumulative distribution function of a StudentsT distribution*

Description

Evaluate the cumulative distribution function of a StudentsT distribution

Usage

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

Arguments

d	A StudentsT object created by a call to StudentsT().
x	A vector of elements whose cumulative probabilities you would like to determine given the distribution d.
drop	logical. Should the result be simplified to a vector if possible?
elementwise	logical. Should each distribution in d be evaluated at all elements of x (elementwise = FALSE, yielding a matrix)? Or, if d 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.
...	Arguments to be passed to pt. Unevaluated arguments will generate a warning to catch mispellings or other possible errors.

Value

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.

See Also

Other StudentsT distribution: `pdf.StudentsT()`, `quantile.StudentsT()`, `random.StudentsT()`

Examples

```
set.seed(27)

X <- StudentsT(3)
X

random(X, 10)

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

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

### example: calculating p-values for two-sided T-test

# here the null hypothesis is H_0: mu = 3

# data to test
x <- c(3, 7, 11, 0, 7, 0, 4, 5, 6, 2)
nx <- length(x)

# calculate the T-statistic
t_stat <- (mean(x) - 3) / (sd(x) / sqrt(nx))
t_stat

# null distribution of statistic depends on sample size!
T <- StudentsT(df = nx - 1)

# calculate the two-sided p-value
1 - cdf(T, abs(t_stat)) + cdf(T, -abs(t_stat))

# exactly equivalent to the above
2 * cdf(T, -abs(t_stat))

# p-value for one-sided test
# H_0: mu <= 3   vs   H_A: mu > 3
1 - cdf(T, t_stat)

# p-value for one-sided test
# H_0: mu >= 3   vs   H_A: mu < 3
cdf(T, t_stat)
```

```
### example: calculating a 88 percent T CI for a mean

# lower-bound
mean(x) - quantile(T, 1 - 0.12 / 2) * sd(x) / sqrt(nx)

# upper-bound
mean(x) + quantile(T, 1 - 0.12 / 2) * sd(x) / sqrt(nx)

# equivalent to
mean(x) + c(-1, 1) * quantile(T, 1 - 0.12 / 2) * sd(x) / sqrt(nx)

# also equivalent to
mean(x) + quantile(T, 0.12 / 2) * sd(x) / sqrt(nx)
mean(x) + quantile(T, 1 - 0.12 / 2) * sd(x) / sqrt(nx)
```

`cdf.Tukey` *Evaluate the cumulative distribution function of a Tukey distribution*

Description

Evaluate the cumulative distribution function of a Tukey distribution

Usage

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

Arguments

`d`	A `Tukey` distribution created by a call to `Tukey()`.
`x`	A vector of elements whose cumulative probabilities you would like to determine given the distribution `d`.
`drop`	logical. Should the result be simplified to a vector if possible?
`elementwise`	logical. Should each distribution in `d` be evaluated at all elements of `x` (`elementwise = FALSE`, yielding a matrix)? Or, if `d` 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.
`...`	Arguments to be passed to `ptukey`. Unevaluated arguments will generate a warning to catch mispellings or other possible errors.

Value

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.

See Also

Other Tukey distribution: quantile.Tukey()

Examples

```
set.seed(27)

X <- Tukey(4L, 16L, 2L)
X

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

---

cdf.Uniform *Evaluate the cumulative distribution function of a continuous Uniform distribution*

Description

Evaluate the cumulative distribution function of a continuous Uniform distribution

Usage

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

Arguments

d	A Uniform object created by a call to Uniform().
x	A vector of elements whose cumulative probabilities you would like to determine given the distribution d.
drop	logical. Should the result be simplified to a vector if possible?
elementwise	logical. Should each distribution in d be evaluated at all elements of x (elementwise = FALSE, yielding a matrix)? Or, if d 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.
...	Arguments to be passed to punif. Unevaluated arguments will generate a warning to catch mispellings or other possible errors.

Value

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.

Examples

```
set.seed(27)

X <- Uniform(1, 2)
X

random(X, 10)

pdf(X, 0.7)
log_pdf(X, 0.7)

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

cdf(X, quantile(X, 0.7))
quantile(X, cdf(X, 0.7))
```

`cdf.Weibull` *Evaluate the cumulative distribution function of a Weibull distribution*

Description

Evaluate the cumulative distribution function of a Weibull distribution

Usage

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

Arguments

d	A `Weibull` object created by a call to `Weibull()`.
x	A vector of elements whose cumulative probabilities you would like to determine given the distribution `d`.
drop	logical. Should the result be simplified to a vector if possible?
elementwise	logical. Should each distribution in `d` be evaluated at all elements of `x` (`elementwise = FALSE`, yielding a matrix)? Or, if `d` 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.
...	Arguments to be passed to `pweibull`. Unevaluated arguments will generate a warning to catch mispellings or other possible errors.

Value

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.

See Also

Other Weibull distribution: pdf.Weibull(), quantile.Weibull(), random.Weibull()

Examples

```
set.seed(27)

X <- Weibull(0.3, 2)
X

random(X, 10)

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

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

---

cdf.ZINegativeBinomial

Evaluate the cumulative distribution function of a zero-inflated negative binomial distribution

---

Description

Evaluate the cumulative distribution function of a zero-inflated negative binomial distribution

Usage

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

Arguments

d	A ZINegativeBinomial object created by a call to ZINegativeBinomial().
x	A vector of elements whose cumulative probabilities you would like to determine given the distribution d.
drop	logical. Should the result be simplified to a vector if possible?
elementwise	logical. Should each distribution in d be evaluated at all elements of x (elementwise = FALSE, yielding a matrix)? Or, if d 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.
...	Arguments to be passed to pzinbinom. Unevaluated arguments will generate a warning to catch mispellings or other possible errors.

Value

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.

Examples

```
## set up a zero-inflated negative binomial distribution
X <- ZINegativeBinomial(mu = 2.5, theta = 1, pi = 0.25)
X

## standard functions
pdf(X, 0:8)
cdf(X, 0:8)
quantile(X, seq(0, 1, by = 0.25))

## cdf() and quantile() are inverses for each other
quantile(X, cdf(X, 3))

## density visualization
plot(0:8, pdf(X, 0:8), type = "h", lwd = 2)

## corresponding sample with histogram of empirical frequencies
set.seed(0)
x <- random(X, 500)
hist(x, breaks = -1:max(x) + 0.5)
```

---

cdf.ZIPoisson *Evaluate the cumulative distribution function of a zero-inflated Poisson distribution*

Description

Evaluate the cumulative distribution function of a zero-inflated Poisson distribution

Usage

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

Arguments

d	A ZIPoisson object created by a call to ZIPoisson().
x	A vector of elements whose cumulative probabilities you would like to determine given the distribution d.
drop	logical. Should the result be simplified to a vector if possible?

elementwise	logical. Should each distribution in d be evaluated at all elements of x (elementwise = FALSE, yielding a matrix)? Or, if d 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.
...	Arguments to be passed to pzipois. Unevaluated arguments will generate a warning to catch mispellings or other possible errors.

Value

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.

Examples

```
## set up a zero-inflated Poisson distribution
X <- ZIPoisson(lambda = 2.5, pi = 0.25)
X

## standard functions
pdf(X, 0:8)
cdf(X, 0:8)
quantile(X, seq(0, 1, by = 0.25))

## cdf() and quantile() are inverses for each other
quantile(X, cdf(X, 3))

## density visualization
plot(0:8, pdf(X, 0:8), type = "h", lwd = 2)

## corresponding sample with histogram of empirical frequencies
set.seed(0)
x <- random(X, 500)
hist(x, breaks = -1:max(x) + 0.5)
```

---

cdf.ZTNegativeBinomial

Evaluate the cumulative distribution function of a zero-truncated negative binomial distribution

---

Description

Evaluate the cumulative distribution function of a zero-truncated negative binomial distribution

Usage

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

Arguments

d	A ZTNegativeBinomial object created by a call to ZTNegativeBinomial().
x	A vector of elements whose cumulative probabilities you would like to determine given the distribution d.
drop	logical. Should the result be simplified to a vector if possible?
elementwise	logical. Should each distribution in d be evaluated at all elements of x (elementwise = FALSE, yielding a matrix)? Or, if d 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.
...	Arguments to be passed to pztnbinom. Unevaluated arguments will generate a warning to catch mispellings or other possible errors.

Value

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.

Examples

```
## set up a zero-truncated negative binomial distribution
X <- ZTNegativeBinomial(mu = 2.5, theta = 1)
X

## standard functions
pdf(X, 0:8)
cdf(X, 0:8)
quantile(X, seq(0, 1, by = 0.25))

## cdf() and quantile() are inverses for each other
quantile(X, cdf(X, 3))

## density visualization
plot(0:8, pdf(X, 0:8), type = "h", lwd = 2)

## corresponding sample with histogram of empirical frequencies
set.seed(0)
x <- random(X, 500)
hist(x, breaks = -1:max(x) + 0.5)
```

---

cdf.ZTPoisson *Evaluate the cumulative distribution function of a zero-truncated Poisson distribution*

Description

Evaluate the cumulative distribution function of a zero-truncated Poisson distribution

Usage

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

Arguments

d	A ZTPoisson object created by a call to ZTPoisson().
x	A vector of elements whose cumulative probabilities you would like to determine given the distribution d.
drop	logical. Should the result be simplified to a vector if possible?
elementwise	logical. Should each distribution in d be evaluated at all elements of x (elementwise = FALSE, yielding a matrix)? Or, if d 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.
...	Arguments to be passed to pztpois. Unevaluated arguments will generate a warning to catch mispellings or other possible errors.

Value

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.

Examples

```
## set up a zero-truncated Poisson distribution
X <- ZTPoisson(lambda = 2.5)
X

## standard functions
pdf(X, 0:8)
cdf(X, 0:8)
quantile(X, seq(0, 1, by = 0.25))

## cdf() and quantile() are inverses for each other
quantile(X, cdf(X, 3))

## density visualization
plot(0:8, pdf(X, 0:8), type = "h", lwd = 2)

## corresponding sample with histogram of empirical frequencies
set.seed(0)
x <- random(X, 500)
hist(x, breaks = -1:max(x) + 0.5)
```

`ChiSquare` *Create a Chi-Square distribution*

Description

Chi-square distributions show up often in frequentist settings as the sampling distribution of test statistics, especially in maximum likelihood estimation settings.

Usage

```
ChiSquare(df = numeric())
```

Arguments

`df`	Degrees of freedom. Must be positive.

Details

We recommend reading this documentation on https://zeileis.github.io/distributions3/, where the math will render with additional detail and much greater clarity.

In the following, let X be a χ^2 random variable with `df` = k.

Support: R^+, the set of positive real numbers

Mean: k

Variance: $2k$

Probability density function (p.d.f):

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-(x-\mu)^2/2\sigma^2}$$

Cumulative distribution function (c.d.f):

The cumulative distribution function has the form

$$F(t) = \int_{-\infty}^{t} \frac{1}{\sqrt{2\pi\sigma^2}} e^{-(x-\mu)^2/2\sigma^2} dx$$

but this integral does not have a closed form solution and must be approximated numerically. The c.d.f. of a standard normal is sometimes called the "error function". The notation $\Phi(t)$ also stands for the c.d.f. of a standard normal evaluated at t. Z-tables list the value of $\Phi(t)$ for various t.

Moment generating function (m.g.f):

$$E(e^{tX}) = e^{\mu t + \sigma^2 t^2/2}$$

Value

A `ChiSquare` object.

Transformations

A squared standard `Normal()` distribution is equivalent to a χ^2_1 distribution with one degree of freedom. The χ^2 distribution is a special case of the `Gamma()` distribution with shape (TODO: check this) parameter equal to a half. Sums of χ^2 distributions are also distributed as χ^2 distributions, where the degrees of freedom of the contributing distributions get summed. The ratio of two χ^2 distributions is a `FisherF()` distribution. The ratio of a `Normal()` and the square root of a scaled `ChiSquare()` is a `StudentsT()` distribution.

See Also

Other continuous distributions: `Beta()`, `Cauchy()`, `Erlang()`, `Exponential()`, `FisherF()`, `Frechet()`, `GEV()`, `GP()`, `Gamma()`, `Gumbel()`, `LogNormal()`, `Logistic()`, `Normal()`, `RevWeibull()`, `SinhArcsinh()`, `StudentsT()`, `Tukey()`, `Uniform()`, `Weibull()`

Examples

```
set.seed(27)

X <- ChiSquare(5)
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)

cdf(X, quantile(X, 0.7))
quantile(X, cdf(X, 7))
```

---

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

```
crps.distribution(
  y,
  x,
  drop = TRUE,
  elementwise = NULL,
  gridsize = 500L,
  batchsize = 10000L,
  applyfun = NULL,
  cores = NULL,
  method = NULL,
  ...
)

crps.Beta(y, x, drop = TRUE, elementwise = NULL, ...)

crps.Bernoulli(y, x, drop = TRUE, elementwise = NULL, ...)

crps.Binomial(y, x, drop = TRUE, elementwise = NULL, ...)

crps.Erlang(y, x, drop = TRUE, elementwise = NULL, ...)

crps.Exponential(y, x, drop = TRUE, elementwise = NULL, ...)

crps.Gamma(y, x, drop = TRUE, elementwise = NULL, ...)

crps.GEV(y, x, drop = TRUE, elementwise = NULL, ...)

crps.Geometric(y, x, drop = TRUE, elementwise = NULL, ...)

crps.Gumbel(y, x, drop = TRUE, elementwise = NULL, ...)

crps.HyperGeometric(y, x, drop = TRUE, elementwise = NULL, ...)

crps.Logistic(y, x, drop = TRUE, elementwise = NULL, ...)

crps.LogNormal(y, x, drop = TRUE, elementwise = NULL, ...)

crps.NegativeBinomial(y, x, drop = TRUE, elementwise = NULL, ...)

crps.Normal(y, x, drop = TRUE, elementwise = NULL, ...)

crps.Poisson(y, x, drop = TRUE, elementwise = NULL, ...)

crps.StudentsT(y, x, drop = TRUE, elementwise = NULL, ...)

crps.Uniform(y, x, drop = TRUE, elementwise = NULL, ...)
```

```
crps.XBetaX(y, x, drop = TRUE, elementwise = NULL, method = "cdf", ...)

crps.GAMLSS(y, x, drop = TRUE, elementwise = NULL, ...)

crps.BAMLSS(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 integer. Size of the grid used to approximate the CDF for the numerical calculation of the CRPS.
batchsize	positive integer. 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	NULL (default) or positive integer. If integer, the applyfun is set to mclapply with the desired number of cores, except on Windows where parLapply with makeCluster(cores) is used.
method	NULL (default) or character. Specifies how the grid for the calculation is set up. If NULL it is set to method = "cdf" if the distribution (y) is discrete and gridsize is large enough to span the required range for CRPS calculation. Else (including continuous distributions) method = "quantile" is used.
...	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). If dedicated closed-form CRPS computations are provided (either by the **scoringRules** package or **distributions3**) these are used as 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.

Value

If length(y) equals one or length(y) = length(x) a (possibly named) numeric vector is returned if drop = TRUE (default). Else a matrix of dimension length(y) times length(x) is returned containing the CRPS.

Numerical approximation strategy

The general method for any `distribution` object uses the following strategy. 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 + 5L` 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 `1L`) are drawn and the corresponding probabilities from the `cdf` are calculated (including the observation `x`) used to numerically integrate for calculating the CRPS. If `gridsize` 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 + 5L)` (or `length(y) * (gridsize + 1L)`) which may be very memory intensive if `length(y)` and/or `gridsize` are large. Thus, the data is split in 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`.

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)
```

`dempirical` *The Empirical distribution*

Description

Density (point mass), distribution, quantile function, as well as a random generation for the Empirical distribution.

Usage

```
dempirical(x, y, log = FALSE, na.rm = TRUE, method = NULL, ...)

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

qempirical(
  p,
  y,
  lower.tail = TRUE,
  log.p = FALSE,
  na.rm = TRUE,
  type = 1L,
  ...
)

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

Arguments

`x`	Vector of finite quantiles.
`y`	Vector of observations of the empirical distribution with two or more non-missing finite values.
`log, log.p`	logical. Indicates whether probabilities p are given as log(p) (both default to `FALSE`).
`na.rm`	logical indicating whether missing values (`NA`) are stripped before computation, defaults to `TRUE`.
`method`	`NULL` or one of `"hist"` or `"density"`. If `NULL`, `y` is considered a random variable from a discrete empirical distribution. Method `"hist"` and `"density"` approximate a 'continuous' distribution based on the empirical sample `y`.

...	Allows to forward arguments to hist, density() and apply/sapply when calling dempirical() or sample quantile function when calling qempirical(). Else currently unused.
q	vector of quantiles.
lower.tail	logical indicating whether probabilities are $P[X \leq x]$ (lower tail) or $P[X > x]$ (upper tail).
p	numeric vector of probabilities ([0, 1]).
type	integer, forwarded to quantile. Defaults to type = 1L.
n	number of observations. If length(n) > 1, the length is taken to be the number required.

Details

All functions follow the usual conventions of d/p/q/r functions in base R. In particular, all four functions for the Empirical distribution call the corresponding *empirical functions.

See Also

Other Empirical distribution: Empirical(), cdf.Empirical(), pdf.Empirical(), quantile.Empirical(), random.Empirical(), support.Empirical()

Examples

```
## Drawing two random empirical sample Y from the LogNormal distribution
## rounded to closest 0.5 (discrete)
set.seed(6020)
Y <- rlnorm(500L, meanlog = 1.5, sdlog = log(1.5))
Y <- round(Y * 2) / 2

bk <- seq(-0.25, 18.25, by = 1L)
hist(Y, freq = FALSE, breaks = bk, main = "Sample histogram")

x <- seq(0, 15, by = 0.5) # Quantiles
density <- dempirical(x, Y)
plot(density ~ x, type = "h", main = "Empirical density")

probability <- pempirical(x, Y)
plot(probability ~ x, type = "s", main = "Empirical distribution")

probs <- seq(0.01, 0.99, by = 0.01)
quantiles <- qempirical(probs, Y)
plot(probs ~ quantiles, type = "S", col = 2,
     main = "Empirical quantile function")

## Drawing random numbers (sampling with replacement)
set.seed(6020)
r <- rempirical(500L, Y)

hist(Y, freq = FALSE, breaks = bk, main = "Sample histogram")
hist(r, freq = FALSE, breaks = bk, main = "Random sample histogram")
```

dhnbinom *The hurdle negative binomial distribution*

Description

Density, distribution function, quantile function, and random generation for the zero-hurdle negative binomial distribution with parameters `mu`, `theta` (or `size`), and `pi`.

Usage

```
dhnbinom(x, mu, theta, size, pi, log = FALSE)

phnbinom(q, mu, theta, size, pi, lower.tail = TRUE, log.p = FALSE)

qhnbinom(p, mu, theta, size, pi, lower.tail = TRUE, log.p = FALSE)

rhnbinom(n, mu, theta, size, pi)
```

Arguments

`x`	vector of (non-negative integer) quantiles.
`mu`	vector of (non-negative) negative binomial location parameters.
`theta`, `size`	vector of (non-negative) negative binomial overdispersion parameters. Only `theta` or, equivalently, `size` may be specified.
`pi`	vector of zero-hurdle probabilities in the unit interval.
`log`, `log.p`	logical indicating whether probabilities p are given as log(p).
`q`	vector of quantiles.
`lower.tail`	logical indicating whether probabilities are $P[X \leq x]$ (lower tail) or $P[X > x]$ (upper tail).
`p`	vector of probabilities.
`n`	number of random values to return.

Details

All functions follow the usual conventions of d/p/q/r functions in base R. In particular, all four `hnbinom` functions for the hurdle negative binomial distribution call the corresponding `nbinom` functions for the negative binomial distribution from base R internally.

Note, however, that the precision of `qhnbinom` for very large probabilities (close to 1) is limited because the probabilities are internally handled in levels and not in logs (even if `log.p = TRUE`).

See Also

`HurdleNegativeBinomial`, `dnbinom`

Examples

```
## theoretical probabilities for a hurdle negative binomial distribution
x <- 0:8
p <- dhnbinom(x, mu = 2.5, theta = 1, pi = 0.75)
plot(x, p, type = "h", lwd = 2)

## corresponding empirical frequencies from a simulated sample
set.seed(0)
y <- rhnbinom(500, mu = 2.5, theta = 1, pi = 0.75)
hist(y, breaks = -1:max(y) + 0.5)
```

---

dhpois *The hurdle Poisson distribution*

Description

Density, distribution function, quantile function, and random generation for the zero-hurdle Poisson distribution with parameters `lambda` and `pi`.

Usage

```
dhpois(x, lambda, pi, log = FALSE)

phpois(q, lambda, pi, lower.tail = TRUE, log.p = FALSE)

qhpois(p, lambda, pi, lower.tail = TRUE, log.p = FALSE)

rhpois(n, lambda, pi)
```

Arguments

`x`	vector of (non-negative integer) quantiles.
`lambda`	vector of (non-negative) Poisson parameters.
`pi`	vector of zero-hurdle probabilities in the unit interval.
`log`, `log.p`	logical indicating whether probabilities p are given as log(p).
`q`	vector of quantiles.
`lower.tail`	logical indicating whether probabilities are $P[X \leq x]$ (lower tail) or $P[X > x]$ (upper tail).
`p`	vector of probabilities.
`n`	number of random values to return.

Details

All functions follow the usual conventions of d/p/q/r functions in base R. In particular, all four hpois functions for the hurdle Poisson distribution call the corresponding pois functions for the Poisson distribution from base R internally.

Note, however, that the precision of qhpois for very large probabilities (close to 1) is limited because the probabilities are internally handled in levels and not in logs (even if log.p = TRUE).

See Also

HurdlePoisson, dpois

Examples

```
## theoretical probabilities for a hurdle Poisson distribution
x <- 0:8
p <- dhpois(x, lambda = 2.5, pi = 0.75)
plot(x, p, type = "h", lwd = 2)

## corresponding empirical frequencies from a simulated sample
set.seed(0)
y <- rhpois(500, lambda = 2.5, pi = 0.75)
hist(y, breaks = -1:max(y) + 0.5)
```

---

dsinharcsinh *The Sinh-Arcsinh (SHASH) distribution*

Description

Density, distribution function, quantile function, and random generation for the sinh-arcsinh distribution with four parameters mu, sigma, nu, and tau.

Usage

```
dsinharcsinh(x, mu = 0, sigma = 1, nu = 1, tau = 1, log = FALSE, cores = NULL)

psinharcsinh(
  q,
  mu = 0,
  sigma = 1,
  nu = 1,
  tau = 1,
  lower.tail = TRUE,
  log.p = FALSE,
  cores = NULL
)
```

```
qsinharcsinh(
  p,
  mu = 0,
  sigma = 1,
  nu = 1,
  tau = 1,
  lower.tail = TRUE,
  log.p = FALSE,
  cores = NULL
)

rsinharcsinh(n, mu = 0, sigma = 1, nu = 1, tau = 1, cores = NULL)
```

Arguments

x	vector of (non-negative integer) quantiles.
mu, sigma, nu, tau	vector of (non-negative) parameters.
log, log.p	logical indicating whether probabilities p are given as log(p).
cores	integer. Number of cores/threads to be used (requires OMP support).
q	vector of quantiles.
lower.tail	logical indicating whether probabilities are $P[X \leq x]$ (lower tail) or $P[X > x]$ (upper tail).
p	vector of probabilities.
n	number of random values to return.

Details

The Sinh-Arcsinh generalizes the Normal distribution by separately controlling location, scale, skewness, and tail-heaviness and can thus produce a wide range of shapes. Using nu = 1 and tau = 1 results in a normal distribution.

All functions follow the usual conventions of d/p/q/r functions in base R.

Examples

```
## theoretical probabilities for a Sinh-Arcsinh distribution
## with mu = 0, sigma = 1, nu = 1, tau = 1 (default) the Sinh-Arcsinh distribution
## corresponds to the standard normal distribution
x <- seq(-5, 5, by = 0.1)
p <- dsinharcsinh(x)
plot(x, p, type = "l", lwd = 2)
lines(x, dnorm(x), col = 2, lty = 2, lwd = 2)

## corresponding empirical frequencies from a simulated sample
## with mu = 5, sigma = 3, nu = 0.7, tau = 0.7
set.seed(0)
y <- rsinharcsinh(500, mu = 5, sigma = 3, nu = 1.1, tau = 0.7)
hist(y)
```

```
## the quantile function is the inverse of the distribution function
psinharcsinh(qsinharcsinh(0.7))
qsinharcsinh(psinharcsinh(3))

## inversion using custom parameters mu = 5, sigma = 2, nu = 0.7, tau = 1.3
psinharcsinh(qsinharcsinh(0.7, 5, 2, 0.7, 1.3), 5, 2, 0.7, 1.3)
qsinharcsinh(psinharcsinh(3,  5, 2, 0.7, 1.3),  5, 2, 0.7, 1.3)
```

`dzinbinom` *The zero-inflated negative binomial distribution*

Description

Density, distribution function, quantile function, and random generation for the zero-inflated negative binomial distribution with parameters `mu`, `theta` (or `size`), and `pi`.

Usage

```
dzinbinom(x, mu, theta, size, pi, log = FALSE)

pzinbinom(q, mu, theta, size, pi, lower.tail = TRUE, log.p = FALSE)

qzinbinom(p, mu, theta, size, pi, lower.tail = TRUE, log.p = FALSE)

rzinbinom(n, mu, theta, size, pi)
```

Arguments

`x`	vector of (non-negative integer) quantiles.
`mu`	vector of (non-negative) negative binomial location parameters.
`theta`, `size`	vector of (non-negative) negative binomial overdispersion parameters. Only `theta` or, equivalently, `size` may be specified.
`pi`	vector of zero-inflation probabilities in the unit interval.
`log`, `log.p`	logical indicating whether probabilities p are given as log(p).
`q`	vector of quantiles.
`lower.tail`	logical indicating whether probabilities are $P[X \leq x]$ (lower tail) or $P[X > x]$ (upper tail).
`p`	vector of probabilities.
`n`	number of random values to return.

Details

All functions follow the usual conventions of d/p/q/r functions in base R. In particular, all four `zinbinom` functions for the zero-inflated negative binomial distribution call the corresponding `nbinom` functions for the negative binomial distribution from base R internally.

Note, however, that the precision of `qzinbinom` for very large probabilities (close to 1) is limited because the probabilities are internally handled in levels and not in logs (even if `log.p = TRUE`).

See Also

`ZINegativeBinomial`, `dnbinom`

Examples

```
## theoretical probabilities for a zero-inflated negative binomial distribution
x <- 0:8
p <- dzinbinom(x, mu = 2.5, theta = 1, pi = 0.25)
plot(x, p, type = "h", lwd = 2)

## corresponding empirical frequencies from a simulated sample
set.seed(0)
y <- rzinbinom(500, mu = 2.5, theta = 1, pi = 0.25)
hist(y, breaks = -1:max(y) + 0.5)
```

---

dzipois *The zero-inflated Poisson distribution*

Description

Density, distribution function, quantile function, and random generation for the zero-inflated Poisson distribution with parameters `lambda` and `pi`.

Usage

```
dzipois(x, lambda, pi, log = FALSE)

pzipois(q, lambda, pi, lower.tail = TRUE, log.p = FALSE)

qzipois(p, lambda, pi, lower.tail = TRUE, log.p = FALSE)

rzipois(n, lambda, pi)
```

Arguments

`x`	vector of (non-negative integer) quantiles.
`lambda`	vector of (non-negative) Poisson parameters.
`pi`	vector of zero-inflation probabilities in the unit interval.

log, log.p	logical indicating whether probabilities p are given as log(p).
q	vector of quantiles.
lower.tail	logical indicating whether probabilities are $P[X \leq x]$ (lower tail) or $P[X > x]$ (upper tail).
p	vector of probabilities.
n	number of random values to return.

Details

All functions follow the usual conventions of d/p/q/r functions in base R. In particular, all four zipois functions for the zero-inflated Poisson distribution call the corresponding pois functions for the Poisson distribution from base R internally.

Note, however, that the precision of qzipois for very large probabilities (close to 1) is limited because the probabilities are internally handled in levels and not in logs (even if log.p = TRUE).

See Also

ZIPoisson, dpois

Examples

```
## theoretical probabilities for a zero-inflated Poisson distribution
x <- 0:8
p <- dzipois(x, lambda = 2.5, pi = 0.25)
plot(x, p, type = "h", lwd = 2)

## corresponding empirical frequencies from a simulated sample
set.seed(0)
y <- rzipois(500, lambda = 2.5, pi = 0.25)
hist(y, breaks = -1:max(y) + 0.5)
```

---

dztnbinom *The zero-truncated negative binomial distribution*

Description

Density, distribution function, quantile function, and random generation for the zero-truncated negative binomial distribution with parameters mu and theta (or size).

Usage

```
dztnbinom(x, mu, theta, size, log = FALSE)

pztnbinom(q, mu, theta, size, lower.tail = TRUE, log.p = FALSE)

qztnbinom(p, mu, theta, size, lower.tail = TRUE, log.p = FALSE)

rztnbinom(n, mu, theta, size)
```

Arguments

x	vector of (non-negative integer) quantiles.
mu	vector of (non-negative) negative binomial location parameters.
theta, size	vector of (non-negative) negative binomial overdispersion parameters. Only theta or, equivalently, size may be specified.
log, log.p	logical indicating whether probabilities p are given as log(p).
q	vector of quantiles.
lower.tail	logical indicating whether probabilities are $P[X \leq x]$ (lower tail) or $P[X > x]$ (upper tail).
p	vector of probabilities.
n	number of random values to return.

Details

The negative binomial distribution left-truncated at zero (or zero-truncated negative binomial for short) is the distribution obtained, when considering a negative binomial variable Y conditional on Y being greater than zero.

All functions follow the usual conventions of d/p/q/r functions in base R. In particular, all four ztnbinom functions for the zero-truncated negative binomial distribution call the corresponding nbinom functions for the negative binomial distribution from base R internally.

See Also

ZTNegativeBinomial, dnbinom

Examples

```
## theoretical probabilities for a zero-truncated negative binomial distribution
x <- 0:8
p <- dztnbinom(x, mu = 2.5, theta = 1)
plot(x, p, type = "h", lwd = 2)

## corresponding empirical frequencies from a simulated sample
set.seed(0)
y <- rztnbinom(500, mu = 2.5, theta = 1)
hist(y, breaks = -1:max(y) + 0.5)
```

---

dztpois *The zero-truncated Poisson distribution*

Description

Density, distribution function, quantile function, and random generation for the zero-truncated Poisson distribution with parameter lambda.

Usage

```
dztpois(x, lambda, log = FALSE)

pztpois(q, lambda, lower.tail = TRUE, log.p = FALSE)

qztpois(p, lambda, lower.tail = TRUE, log.p = FALSE)

rztpois(n, lambda)
```

Arguments

x	vector of (non-negative integer) quantiles.
lambda	vector of (non-negative) Poisson parameters.
log, log.p	logical indicating whether probabilities p are given as log(p).
q	vector of quantiles.
lower.tail	logical indicating whether probabilities are $P[X \leq x]$ (lower tail) or $P[X > x]$ (upper tail).
p	vector of probabilities.
n	number of random values to return.

Details

The Poisson distribution left-truncated at zero (or zero-truncated Poisson for short) is the distribution obtained, when considering a Poisson variable Y conditional on Y being greater than zero.

All functions follow the usual conventions of d/p/q/r functions in base R. In particular, all four `ztpois` functions for the zero-truncated Poisson distribution call the corresponding `pois` functions for the Poisson distribution from base R internally.

See Also

ZTPoisson, dpois

Examples

```
## theoretical probabilities for a zero-truncated Poisson distribution
x <- 0:8
p <- dztpois(x, lambda = 2.5)
plot(x, p, type = "h", lwd = 2)

## corresponding empirical frequencies from a simulated sample
set.seed(0)
y <- rztpois(500, lambda = 2.5)
hist(y, breaks = -1:max(y) + 0.5)
```

Empirical *Create an Empirical distribution*

Description

An empirical distribution based on a random `sample`.

Usage

```
Empirical(sample = numeric())

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

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

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

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

Arguments

`sample`	A numeric vector, list of numeric vectors, matrix, or data.frame (see section 'Details' for more information).
`x`	an object of class `Empirical` (see `Empirical()`).
`...`	currently unused.
`type`	integer between `1L` and `3L` (default) selecting one of three algorithms used for calculating sample skewness/kurtosis. See section Details for more information.

Details

The constructor function `Empirical()` allows for a variety of different objects as main input `sample`.

- 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/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`). Certain types of moments (see `skewness.Empirical()`, `kurtosis.Empirical()`) require at least three or four finite observations.

Support: Set of unique observations in the `sample`, denoted y below.

Probability mass function (p.m.f.):

$$f(x) = \frac{1}{n}\sum_{i=1}^{N}(y_i = x)$$

Cummulative distribution function (c.d.f.):

$$F(x) = \frac{1}{N}\sum_{i=1}^{N}\mathbf{I}(y_i \leq x)$$

Moment generating functions:

- Mean/expectation:

$$\bar{y} = \frac{1}{N}\sum_{i=1}^{N} y_i$$

- Variance:

$$\frac{1}{N-1}\sum_{i=1}^{N}(y_i - \bar{y})$$

Third and fourth central moments are also available via `skewness()` and `kurtosis()`. For both different types are available as defined below. For details see Joanes and Gill (1998).

- Skewness:
  - Type 1:

$$S_1 = \sqrt{N}\frac{\sum_{i=1}^{N}(y_i - \bar{y})^3}{\sqrt{\left(\sum_{i=1}^{N}(y_i - \bar{y})^2\right)^3}}$$

  - Type 2 (only defined for three or more finite values):

$$S_2 = \frac{\sqrt{N \cdot (N-1)}}{(N-2)}S_1$$

  - Type 3 (default):

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

- Kurtosis:
  - Type 1:

$$K_1 = N \cdot \frac{\sum_{i=1}^{N}(y_i - \bar{y})^4}{\left(\sum_{i=1}^{N}(y_i - \bar{y})^2\right)^2} - 3$$

  - Type 2 (only defined for four or more finite values):

$$K_2 = \frac{((N+1) \cdot K_1 + 6) \cdot (N-1)}{(N-2) \cdot (N-3)}$$

  - Type 3 (default):

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

Value

An `Empirical` object.

References

Joanes DN, 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

See Also

Other Empirical distribution: `cdf.Empirical()`, `dempirical()`, `pdf.Empirical()`, `quantile.Empirical()`, `random.Empirical()`, `support.Empirical()`

Examples

```
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
Y4 <- matrix(rnorm(20), ncol = 5,
             dimnames = list(paste0("D_", 1:4), paste0("obs_", 1:5)))
d4 <- Empirical(Y4)
d4

## Data frame
d5 <- Empirical(as.data.frame(Y4))
d5

identical(d4, d5)

mean(d5)
variance(d5)
skewness(d5)
kurtosis(d5)

pdf(d5, c(-0.5, 0, 0.5, 1)) # Defaults to elementwise = TRUE
pdf(d5, c(-0.5, 0, 0.5, 1), elementwise = FALSE)

cdf(d5, c(-0.5, 0, 0.5, 1)) # Defaults to elementwise = TRUE
cdf(d5, c(-0.5, 0, 0.5, 1), elementwise = FALSE)

quantile(d5, c(0.2, 0.4, 0.6, 0.8)) # Defaults to elementwise = TRUE
quantile(d5, c(0.2, 0.4, 0.6, 0.8), elementwise = FALSE)

## The quantile function is the inverse of the distribution
## function (cdf) if x in Y
set.seed(6020)
Y <- round(rlnorm(20, log(3), log(2)), 1)
d <- Empirical(Y)

cdf(d, 4.0)
quantile(d, cdf(d, 4.0))
```

`Erlang` *Create an Erlang distribution*

Description

The Erlang distribution is a two-parameter family of continuous probability distributions with support $x \in [0, \infty)$. The two parameters are a positive integer shape parameter k and a positive real rate parameter λ. The Erlang distribution with shape parameter $k = 1$ simplifies to the exponential distribution, and it is a special case of the gamma distribution. It corresponds to a sum of k independent exponential variables with mean $1/\lambda$ each.

Usage

```
Erlang(k = numeric(), lambda = numeric())
```

Arguments

k	The shape parameter. Can be any positive integer number.
lambda	The rate parameter. Can be any positive number.

Value

An Erlang object.

See Also

Other continuous distributions: Beta(), Cauchy(), ChiSquare(), Exponential(), FisherF(), Frechet(), GEV(), GP(), Gamma(), Gumbel(), LogNormal(), Logistic(), Normal(), RevWeibull(), SinhArcsinh(), StudentsT(), Tukey(), Uniform(), Weibull()

Examples

```
set.seed(27)

X <- Erlang(5, 2)
X

random(X, 10)

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

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

cdf(X, quantile(X, 0.7))
quantile(X, cdf(X, 7))
```

Exponential *Create an Exponential distribution*

Description

Exponential distributions are frequently used for modeling the amount of time that passes until a specific event occurs. For example, exponential distributions could be used to model the time between two earthquakes, the amount of delay between internet packets, or the amount of time a piece of machinery can run before needing repair.

Usage

```
Exponential(rate = 1)
```

Arguments

`rate`	The rate parameter, written λ in textbooks. Can be any positive number. Defaults to 1.

Details

We recommend reading this documentation on https://zeileis.github.io/distributions3/, where the math will render with additional detail and much greater clarity.

In the following, let X be an Exponential random variable with rate parameter `rate` = λ.

Support: $x \in (0, \infty)$

Mean: $\frac{1}{\lambda}$

Variance: $\frac{1}{\lambda^2}$

Probability density function (p.d.f):

$$f(x) = \lambda e^{-\lambda x}$$

Cumulative distribution function (c.d.f):

$$F(x) = 1 - e^{-\lambda x}$$

Moment generating function (m.g.f):

$$\frac{\lambda}{\lambda - t}, for t < \lambda$$

Value

An `Exponential` object.

See Also

Other continuous distributions: `Beta()`, `Cauchy()`, `ChiSquare()`, `Erlang()`, `FisherF()`, `Frechet()`, `GEV()`, `GP()`, `Gamma()`, `Gumbel()`, `LogNormal()`, `Logistic()`, `Normal()`, `RevWeibull()`, `SinhArcsinh()`, `StudentsT()`, `Tukey()`, `Uniform()`, `Weibull()`

Examples

```
set.seed(27)

X <- Exponential(5)
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)

cdf(X, quantile(X, 0.7))
quantile(X, cdf(X, 7))
```

FIFA2018 *Goals scored in all 2018 FIFA World Cup matches*

Description

Data from all 64 matches in the 2018 FIFA World Cup along with predicted ability differences based on bookmakers odds.

Usage

```
data("FIFA2018", package = "distributions3")
```

Format

A data frame with 128 rows and 7 columns.

goals integer. Number of goals scored in normal time (90 minutes), \ i.e., excluding potential extra time or penalties in knockout matches.

team character. 3-letter FIFA code for the team.

match integer. Match ID ranging from 1 (opening match) to 64 (final).

type factor. Type of match for groups A to H, round of 16 (R16), quarter final, semi-final, match for 3rd place, and final.

stage factor. Group vs. knockout tournament stage.

logability numeric. Estimated log-ability for each team based on bookmaker consensus model.

difference numeric. Difference in estimated log-abilities between a team and its opponent in each match.

Details

To investigate the number of goals scored per match in the 2018 FIFA World Cup, `FIFA2018` provides two rows, one for each team, for each of the matches during the tournament. In addition some basic meta-information for the matches (an ID, team name abbreviations, type of match, group vs. knockout stage), information on the estimated log-ability for each team is provided. These have been estimated by Zeileis et al. (2018) prior to the start of the tournament (2018-05-20) based on quoted odds from 26 online bookmakers using the bookmaker consensus model of Leitner et al. (2010). The difference in log-ability between a team and its opponent is a useful predictor for the number of goals scored.

To model the data a basic Poisson regression model provides a good fit. This treats the number of goals by the two teams as independent given the ability difference which is a reasonable assumption in this data set.

Source

The goals for each match have been obtained from Wikipedia (https://en.wikipedia.org/wiki/2018_FIFA_World_Cup) and the log-abilities from Zeileis et al. (2018) based on quoted odds from Oddschecker.com and Bwin.com.

References

Leitner C, Zeileis A, Hornik K (2010). Forecasting Sports Tournaments by Ratings of (Prob)abilities: A Comparison for the EURO 2008. *International Journal of Forecasting*, **26**(3), 471-481. doi:10.1016/j.ijforecast.2009.10.001

Zeileis A, Leitner C, Hornik K (2018). Probabilistic Forecasts for the 2018 FIFA World Cup Based on the Bookmaker Consensus Model. Working Paper 2018-09, Working Papers in Economics and Statistics, Research Platform Empirical and Experimental Economics, University of Innsbruck. https://econpapers.repec.org/RePEc:inn:wpaper:2018-09

Examples

```
## load data
data("FIFA2018", package = "distributions3")

## observed relative frequencies of goals in all matches
obsrvd <- prop.table(table(FIFA2018$goals))

## expected probabilities assuming a simple Poisson model,
## using the average number of goals across all teams/matches
## as the point estimate for the mean (lambda) of the distribution
p_const <- Poisson(lambda = mean(FIFA2018$goals))
p_const
expctd <- pdf(p_const, 0:6)

## comparison: observed vs. expected frequencies
## frequencies for 3 and 4 goals are slightly overfitted
## while 5 and 6 goals are slightly underfitted
cbind("observed" = obsrvd, "expected" = expctd)

## instead of fitting the same average Poisson model to all
## teams/matches, take ability differences into account
m <- glm(goals ~ difference, data = FIFA2018, family = poisson)
summary(m)
## when the ratio of abilities increases by 1 percent, the
## expected number of goals increases by around 0.4 percent

## this yields a different predicted Poisson distribution for
## each team/match
p_reg <- Poisson(lambda = fitted(m))
head(p_reg)
```

```
## as an illustration, the following goal distributions
## were expected for the final (that France won 4-2 against Croatia)
p_final <- tail(p_reg, 2)
p_final
pdf(p_final, 0:6)
## clearly France was expected to score more goals than Croatia
## but both teams scored more goals than expected, albeit not unlikely many

## assuming independence of the number of goals scored, obtain
## table of possible match results (after normal time), along with
## overall probabilities of win/draw/lose
res <- outer(pdf(p_final[1], 0:6), pdf(p_final[2], 0:6))
sum(res[lower.tri(res)]) ## France wins
sum(diag(res))           ## draw
sum(res[upper.tri(res)]) ## France loses

## update expected frequencies table based on regression model
expctd <- pdf(p_reg, 0:6)
head(expctd)
expctd <- colMeans(expctd)
cbind("observed" = obsrvd, "expected" = expctd)
```

`FisherF` *Create an F distribution*

Description

Create an F distribution

Usage

```
FisherF(df1 = numeric(), df2 = numeric(), lambda = 0)
```

Arguments

`df1`	Numerator degrees of freedom. Can be any positive number.
`df2`	Denominator degrees of freedom. Can be any positive number.
`lambda`	Non-centrality parameter. Can be any positive number. Defaults to `0`.

Details

We recommend reading this documentation on https://zeileis.github.io/distributions3/, where the math will render with additional detail.

TODO

Value

A `FisherF` object.

See Also

Other continuous distributions: Beta(), Cauchy(), ChiSquare(), Erlang(), Exponential(), Frechet(), GEV(), GP(), Gamma(), Gumbel(), LogNormal(), Logistic(), Normal(), RevWeibull(), SinhArcsinh(), StudentsT(), Tukey(), Uniform(), Weibull()

Examples

```
set.seed(27)

X <- FisherF(5, 10, 0.2)
X

random(X, 10)

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

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

cdf(X, quantile(X, 0.7))
quantile(X, cdf(X, 7))
```

---

fit_mle *Fit a distribution to data*

Description

Generic function for fitting maximum-likelihood estimates (MLEs) of a distribution based on empirical data.

Usage

```
fit_mle(d, x, ...)
```

Arguments

d	An object. The package provides methods for distribution objects such as those from Normal() or Binomial() etc.
x	A vector of data to compute the likelihood.
...	Arguments passed to methods. Unevaluated arguments will generate a warning to catch mispellings or other possible errors.

Value

A distribution (the same kind as d) where the parameters are the MLE estimates based on x.

Examples

```
X <- Normal()
fit_mle(X, c(-1, 0, 0, 0, 3))
```

---

fit_mle.Bernoulli *Fit a Bernoulli distribution to data*

Description

Fit a Bernoulli distribution to data

Usage

```
## S3 method for class 'Bernoulli'
fit_mle(d, x, ...)
```

Arguments

d	A Bernoulli object.
x	A vector of zeroes and ones.
...	Unused.

Value

a Bernoulli object

---

fit_mle.Binomial *Fit a Binomial distribution to data*

Description

The fit distribution will inherit the same size parameter as the Binomial object passed.

Usage

```
## S3 method for class 'Binomial'
fit_mle(d, x, ...)
```

Arguments

d	A Binomial object.
x	A vector of zeroes and ones.
...	Unused.

Value

a Binomial object

fit_mle.Exponential *Fit an Exponential distribution to data*

Description

Fit an Exponential distribution to data

Usage

```
## S3 method for class 'Exponential'
fit_mle(d, x, ...)
```

Arguments

d	An Exponential object created by a call to Exponential().
x	A vector of data.
...	Unused.

Value

An Exponential object.

fit_mle.Gamma *Fit a Gamma distribution to data*

Description

Fit a Gamma distribution to data

Usage

```
## S3 method for class 'Gamma'
fit_mle(d, x, ...)
```

Arguments

d	A Gamma object created by a call to Gamma().
x	A vector to fit the Gamma distribution to.
...	Unused.

Value

a Gamma object

fit_mle.Geometric *Fit a Geometric distribution to data*

Description

Fit a Geometric distribution to data

Usage

```
## S3 method for class 'Geometric'
fit_mle(d, x, ...)
```

Arguments

d	A Geometric object.
x	A vector of zeroes and ones.
...	Unused.

Value

a Geometric object

fit_mle.LogNormal *Fit a Log Normal distribution to data*

Description

Fit a Log Normal distribution to data

Usage

```
## S3 method for class 'LogNormal'
fit_mle(d, x, ...)
```

Arguments

d	A LogNormal object created by a call to LogNormal().
x	A vector of data.
...	Unused.

Value

A LogNormal object.

See Also

Other LogNormal distribution: `cdf.LogNormal()`, `pdf.LogNormal()`, `quantile.LogNormal()`, `random.LogNormal()`

`fit_mle.Normal` *Fit a Normal distribution to data*

Description

Fit a Normal distribution to data

Usage

```
## S3 method for class 'Normal'
fit_mle(d, x, ...)
```

Arguments

`d`	A `Normal` object created by a call to `Normal()`.
`x`	A vector of data.
`...`	Unused.

Value

A `Normal` object.

See Also

Other Normal distribution: `cdf.Normal()`, `pdf.Normal()`, `quantile.Normal()`

`fit_mle.Poisson` *Fit an Poisson distribution to data*

Description

Fit an Poisson distribution to data

Usage

```
## S3 method for class 'Poisson'
fit_mle(d, x, ...)
```

Arguments

d	An Poisson object created by a call to Poisson().
x	A vector of data.
...	Unused.

Value

An Poisson object.

Frechet *Create a Frechet distribution*

Description

The Frechet distribution is a special case of the \link{GEV} distribution, obtained when the GEV shape parameter ξ is positive. It may be referred to as a type II extreme value distribution.

Usage

```
Frechet(location = 0, scale = 1, shape = 1)
```

Arguments

location	The location (minimum) parameter m. location can be any real number. Defaults to 0.
scale	The scale parameter s. scale can be any positive number. Defaults to 1.
shape	The shape parameter α. shape can be any positive number. Defaults to 1.

Details

We recommend reading this documentation on https://zeileis.github.io/distributions3/, where the math will render with additional detail and much greater clarity.

In the following, let X be a Frechet random variable with location parameter location = m, scale parameter scale = s, and shape parameter shape = α. A Frechet(m, s, α) distribution is equivalent to a \link{GEV}($m + s, s/\alpha, 1/\alpha$) distribution.

Support: (m, ∞).

Mean: $m + s\Gamma(1 - 1/\alpha)$, for $\alpha > 1$; undefined otherwise.

Median: $m + s(\ln 2)^{-1/\alpha}$.

Variance: $s^2[\Gamma(1 - 2/\alpha) - \Gamma(1 - 1/\alpha)^2]$ for $\alpha > 2$; undefined otherwise.

Probability density function (p.d.f):

$$f(x) = \alpha s^{-1}[(x - m)/s]^{-(1+\alpha)} \exp\{-[(x - m)/s]^{-\alpha}\}$$

for $x > m$. The p.d.f. is 0 for $x \leq m$.

Cumulative distribution function (c.d.f):

$$F(x) = \exp\{-[(x-m)/s]^{-\alpha}\}$$

for $x > m$. The c.d.f. is 0 for $x \leq m$.

Value

A `Frechet` object.

See Also

Other continuous distributions: `Beta()`, `Cauchy()`, `ChiSquare()`, `Erlang()`, `Exponential()`, `FisherF()`, `GEV()`, `GP()`, `Gamma()`, `Gumbel()`, `LogNormal()`, `Logistic()`, `Normal()`, `RevWeibull()`, `SinhArcsinh()`, `StudentsT()`, `Tukey()`, `Uniform()`, `Weibull()`

Examples

```
set.seed(27)

X <- Frechet(0, 2)
X

random(X, 10)

pdf(X, 0.7)
log_pdf(X, 0.7)

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

cdf(X, quantile(X, 0.7))
quantile(X, cdf(X, 0.7))
```

`Gamma` *Create a Gamma distribution*

Description

Several important distributions are special cases of the Gamma distribution. When the shape parameter is 1, the Gamma is an exponential distribution with parameter $1/\beta$. When the $shape = n/2$ and $rate = 1/2$, the Gamma is a equivalent to a chi squared distribution with n degrees of freedom. Moreover, if we have X_1 is $Gamma(\alpha_1, \beta)$ and X_2 is $Gamma(\alpha_2, \beta)$, a function of these two variables of the form $\frac{X_1}{X_1+X_2}$ $Beta(\alpha_1, \alpha_2)$. This last property frequently appears in another distributions, and it has extensively been used in multivariate methods. More about the Gamma distribution will be added soon.

Usage

```
Gamma(shape = numeric(), rate = 1)
```

Arguments

shape	The shape parameter. Can be any positive number.
rate	The rate parameter. Can be any positive number. Defaults to 1.

Details

We recommend reading this documentation on https://zeileis.github.io/distributions3/, where the math will render with additional detail.

In the following, let X be a Gamma random variable with parameters shape = α and rate = β.

Support: $x \in (0, \infty)$

Mean: $\frac{\alpha}{\beta}$

Variance: $\frac{\alpha}{\beta^2}$

Probability density function (p.m.f):

$$f(x) = \frac{\beta^\alpha}{\Gamma(\alpha)} x^{\alpha-1} e^{-\beta x}$$

Cumulative distribution function (c.d.f):

$$f(x) = \frac{\Gamma(\alpha, \beta x)}{\Gamma \alpha}$$

Moment generating function (m.g.f):

$$E(e^{tX}) = \left(\frac{\beta}{\beta - t}\right)^\alpha, \, t < \beta$$

Value

A Gamma object.

See Also

Other continuous distributions: Beta(), Cauchy(), ChiSquare(), Erlang(), Exponential(), FisherF(), Frechet(), GEV(), GP(), Gumbel(), LogNormal(), Logistic(), Normal(), RevWeibull(), SinhArcsinh(), StudentsT(), Tukey(), Uniform(), Weibull()

Examples

```
set.seed(27)

X <- Gamma(5, 2)
X

random(X, 10)

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

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

cdf(X, quantile(X, 0.7))
quantile(X, cdf(X, 7))
```

`Geometric` *Create a Geometric distribution*

Description

The Geometric distribution can be thought of as a generalization of the `Bernoulli()` distribution where we ask: "if I keep flipping a coin with probability `p` of heads, what is the probability I need k flips before I get my first heads?" The Geometric distribution is a special case of Negative Binomial distribution.

Usage

```
Geometric(p = 0.5)
```

Arguments

`p`	The success probability for the distribution. `p` can be any value in `[0, 1]`, and defaults to `0.5`.

Details

We recommend reading this documentation on https://zeileis.github.io/distributions3/, where the math will render with additional detail and much greater clarity.

In the following, let X be a Geometric random variable with success probability `p` = p. Note that there are multiple parameterizations of the Geometric distribution.

Support: $0 <$ p < 1, $x = 0, 1, \ldots$

Mean: $\frac{1-p}{p}$

Variance: $\frac{1-p}{p^2}$

Probability mass function (p.m.f):

$$P(X = x) = p(1-p)^x,$$

Cumulative distribution function (c.d.f):

$$P(X \leq x) = 1 - (1-p)^{x+1}$$

Moment generating function (m.g.f):

$$E(e^{tX}) = \frac{pe^t}{1-(1-p)e^t}$$

Value

A Geometric object.

See Also

Other discrete distributions: Bernoulli(), Binomial(), Categorical(), HurdleNegativeBinomial(), HurdlePoisson(), HyperGeometric(), Multinomial(), NegativeBinomial(), Poisson(), PoissonBinomial(), ZINegativeBinomial(), ZIPoisson(), ZTNegativeBinomial(), ZTPoisson()

Examples

```
set.seed(27)

X <- Geometric(0.3)
X

random(X, 10)

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

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

GEV *Create a Generalised Extreme Value (GEV) distribution*

Description

The GEV distribution arises from the Extremal Types Theorem, which is rather like the Central Limit Theorem (see Normal()) but it relates to the *maximum* of n i.i.d. random variables rather than to the sum. If, after a suitable linear rescaling, the distribution of this maximum tends to a non-degenerate limit as n tends to infinity then this limit must be a GEV distribution. The requirement that the variables are independent can be relaxed substantially. Therefore, the GEV distribution is often used to model the maximum of a large number of random variables.

Usage

```
GEV(mu = 0, sigma = 1, xi = 0)
```

Arguments

mu	The location parameter, written μ in textbooks. mu can be any real number. Defaults to 0.
sigma	The scale parameter, written σ in textbooks. sigma can be any positive number. Defaults to 1.
xi	The shape parameter, written ξ in textbooks. xi can be any real number. Defaults to 0, which corresponds to a Gumbel distribution.

Details

We recommend reading this documentation on https://zeileis.github.io/distributions3/, where the math will render with additional detail and much greater clarity.

In the following, let X be a GEV random variable with location parameter `mu` = μ, scale parameter `sigma` = σ and shape parameter `xi` = ξ.

Support: $(-\infty, \mu - \sigma/\xi)$ for $\xi < 0$; $(\mu - \sigma/\xi, \infty)$ for $\xi > 0$; and R, the set of all real numbers, for $\xi = 0$.

Mean: $\mu + \sigma[\Gamma(1-\xi) - 1]/\xi$ for $\xi < 1, \xi \neq 0$; $\mu + \sigma\gamma$ for $\xi = 0$, where γ is Euler's constant, approximately equal to 0.57722; undefined otherwise.

Median: $\mu + \sigma[(\ln 2)^{-\xi} - 1]/\xi$ for $\xi \neq 0$; $\mu - \sigma \ln(\ln 2)$ for $\xi = 0$.

Variance: $\sigma^2[\Gamma(1-2\xi) - \Gamma(1-\xi)^2]/\xi^2$ for $\xi < 1/2, \xi \neq 0$; $\sigma^2\pi^2/6$ for $\xi = 0$; undefined otherwise.

Probability density function (p.d.f):

If $\xi \neq 0$ then

$$f(x) = \sigma^{-1}[1 + \xi(x-\mu)/\sigma]^{-(1+1/\xi)} \exp\{-[1 + \xi(x-\mu)/\sigma]^{-1/\xi}\}$$

for $1 + \xi(x-\mu)/\sigma > 0$. The p.d.f. is 0 outside the support.

In the $\xi = 0$ (Gumbel) special case

$$f(x) = \sigma^{-1} \exp[-(x-\mu)/\sigma] \exp\{-\exp[-(x-\mu)/\sigma]\}$$

for x in R, the set of all real numbers.

Cumulative distribution function (c.d.f):

If $\xi \neq 0$ then

$$F(x) = \exp\{-[1 + \xi(x-\mu)/\sigma]^{-1/\xi}\}$$

for $1 + \xi(x-\mu)/\sigma > 0$. The c.d.f. is 0 below the support and 1 above the support.

In the $\xi = 0$ (Gumbel) special case

$$F(x) = \exp\{-\exp[-(x-\mu)/\sigma]\}$$

for x in R, the set of all real numbers.

Value

A `GEV` object.

See Also

Other continuous distributions: `Beta()`, `Cauchy()`, `ChiSquare()`, `Erlang()`, `Exponential()`, `FisherF()`, `Frechet()`, `GP()`, `Gamma()`, `Gumbel()`, `LogNormal()`, `Logistic()`, `Normal()`, `RevWeibull()`, `SinhArcsinh()`, `StudentsT()`, `Tukey()`, `Uniform()`, `Weibull()`

Examples

```
set.seed(27)

X <- GEV(1, 2, 0.1)
X

random(X, 10)

pdf(X, 0.7)
log_pdf(X, 0.7)

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

cdf(X, quantile(X, 0.7))
quantile(X, cdf(X, 0.7))
```

GP *Create a Generalised Pareto (GP) distribution*

Description

The GP distribution has a link to the `\link{GEV}` distribution. Suppose that the maximum of n i.i.d. random variables has approximately a GEV distribution. For a sufficiently large threshold u, the conditional distribution of the amount (the threshold excess) by which a variable exceeds u given that it exceeds u has approximately a GP distribution. Therefore, the GP distribution is often used to model the threshold excesses of a high threshold u. The requirement that the variables are independent can be relaxed substantially, but then exceedances of u may cluster.

Usage

```
GP(mu = 0, sigma = 1, xi = 0)
```

Arguments

`mu`	The location parameter, written μ in textbooks. `mu` can be any real number. Defaults to `0`.
`sigma`	The scale parameter, written σ in textbooks. `sigma` can be any positive number. Defaults to `1`.
`xi`	The shape parameter, written ξ in textbooks. `xi` can be any real number. Defaults to `0`, which corresponds to a Gumbel distribution.

Details

We recommend reading this documentation on https://zeileis.github.io/distributions3/, where the math will render with additional detail and much greater clarity.

In the following, let X be a GP random variable with location parameter `mu` = μ, scale parameter `sigma` = σ and shape parameter `xi` = ξ.

Support: $[\mu, \mu - \sigma/\xi]$ for $\xi < 0$; $[\mu, \infty)$ for $\xi \geq 0$.

Mean: $\mu + \sigma/(1 - \xi)$ for $\xi < 1$; undefined otherwise.

Median: $\mu + \sigma[2^{\xi} - 1]/\xi$ for $\xi \neq 0$; $\mu + \sigma \ln 2$ for $\xi = 0$.

Variance: $\sigma^2/(1 - \xi)^2(1 - 2\xi)$ for $\xi < 1/2$; undefined otherwise.

Probability density function (p.d.f):

If $\xi \neq 0$ then

$$f(x) = \sigma^{-1}[1 + \xi(x - \mu)/\sigma]^{-(1+1/\xi)}$$

for $1 + \xi(x - \mu)/\sigma > 0$. The p.d.f. is 0 outside the support.

In the $\xi = 0$ special case

$$f(x) = \sigma^{-1} \exp[-(x - \mu)/\sigma]$$

for x in $[\mu, \infty)$. The p.d.f. is 0 outside the support.

Cumulative distribution function (c.d.f):

If $\xi \neq 0$ then

$$F(x) = 1 - \exp\{-[1 + \xi(x - \mu)/\sigma]^{-1/\xi}\}$$

for $1 + \xi(x - \mu)/\sigma > 0$. The c.d.f. is 0 below the support and 1 above the support.

In the $\xi = 0$ special case

$$F(x) = 1 - \exp[-(x - \mu)/\sigma]\}$$

for x in R, the set of all real numbers.

Value

A `GP` object.

See Also

Other continuous distributions: `Beta()`, `Cauchy()`, `ChiSquare()`, `Erlang()`, `Exponential()`, `FisherF()`, `Frechet()`, `GEV()`, `Gamma()`, `Gumbel()`, `LogNormal()`, `Logistic()`, `Normal()`, `RevWeibull()`, `SinhArcsinh()`, `StudentsT()`, `Tukey()`, `Uniform()`, `Weibull()`

Examples

```
set.seed(27)

X <- GP(0, 2, 0.1)
X

random(X, 10)

pdf(X, 0.7)
log_pdf(X, 0.7)

cdf(X, 0.7)
quantile(X, 0.7)
```

```
cdf(X, quantile(X, 0.7))
quantile(X, cdf(X, 0.7))
```

`Gumbel` *Create a Gumbel distribution*

Description

The Gumbel distribution is a special case of the `\link{GEV}` distribution, obtained when the GEV shape parameter ξ is equal to 0. It may be referred to as a type I extreme value distribution.

Usage

```
Gumbel(mu = 0, sigma = 1)
```

Arguments

`mu`	The location parameter, written μ in textbooks. `mu` can be any real number. Defaults to `0`.
`sigma`	The scale parameter, written σ in textbooks. `sigma` can be any positive number. Defaults to `1`.

Details

We recommend reading this documentation on https://zeileis.github.io/distributions3/, where the math will render with additional detail and much greater clarity.

In the following, let X be a Gumbel random variable with location parameter `mu` = μ, scale parameter `sigma` = σ.

Support: R, the set of all real numbers.

Mean: $\mu + \sigma\gamma$, where γ is Euler's constant, approximately equal to 0.57722.

Median: $\mu - \sigma \ln(\ln 2)$.

Variance: $\sigma^2\pi^2/6$.

Probability density function (p.d.f):

$$f(x) = \sigma^{-1} \exp[-(x-\mu)/\sigma] \exp\{-\exp[-(x-\mu)/\sigma]\}$$

for x in R, the set of all real numbers.

Cumulative distribution function (c.d.f):

In the $\xi = 0$ (Gumbel) special case

$$F(x) = \exp\{-\exp[-(x-\mu)/\sigma]\}$$

for x in R, the set of all real numbers.

Value

A Gumbel object.

See Also

Other continuous distributions: Beta(), Cauchy(), ChiSquare(), Erlang(), Exponential(), FisherF(), Frechet(), GEV(), GP(), Gamma(), LogNormal(), Logistic(), Normal(), RevWeibull(), SinhArcsinh(), StudentsT(), Tukey(), Uniform(), Weibull()

Examples

```
set.seed(27)

X <- Gumbel(1, 2)
X

random(X, 10)

pdf(X, 0.7)
log_pdf(X, 0.7)

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

cdf(X, quantile(X, 0.7))
quantile(X, cdf(X, 0.7))
```

HurdleNegativeBinomial

Create a hurdle negative binomial distribution

Description

Hurdle negative binomial distributions are frequently used to model counts with overdispersion and many zero observations.

Usage

```
HurdleNegativeBinomial(mu = numeric(), theta = numeric(), pi = numeric())
```

Arguments

mu	Location parameter of the negative binomial component of the distribution. Can be any positive number.
theta	Overdispersion parameter of the negative binomial component of the distribution. Can be any positive number.
pi	Zero-hurdle probability, can be any value in [0, 1].

Details

We recommend reading this documentation on https://zeileis.github.io/distributions3/, where the math will render with additional detail.

In the following, let X be a hurdle negative binomial random variable with parameters `mu` = μ and `theta` = θ.

Support: $\{0, 1, 2, 3, ...\}$

Mean:

$$\mu \cdot \frac{\pi}{1 - F(0; \mu, \theta)}$$

where $F(k; \mu)$ is the c.d.f. of the `NegativeBinomial` distribution.

Variance:

$$m \cdot \left(1 + \frac{\mu}{\theta} + \mu - m\right)$$

where m is the mean above.

Probability mass function (p.m.f.): $P(X = 0) = 1 - \pi$ and for $k > 0$

$$P(X = k) = \pi \cdot \frac{f(k; \mu, \theta)}{1 - F(0; \mu, \theta)}$$

where $f(k; \mu, \theta)$ is the p.m.f. of the `NegativeBinomial` distribution.

Cumulative distribution function (c.d.f.): $P(X \leq 0) = 1 - \pi$ and for $k > 0$

$$P(X \leq k) = 1 - \pi + \pi \cdot \frac{F(k; \mu, \theta) - F(0; \mu, \theta)}{1 - F(0; \mu, \theta)}$$

Moment generating function (m.g.f.):

Omitted for now.

Value

A `HurdleNegativeBinomial` object.

See Also

Other discrete distributions: `Bernoulli()`, `Binomial()`, `Categorical()`, `Geometric()`, `HurdlePoisson()`, `HyperGeometric()`, `Multinomial()`, `NegativeBinomial()`, `Poisson()`, `PoissonBinomial()`, `ZINegativeBinomial()`, `ZIPoisson()`, `ZTNegativeBinomial()`, `ZTPoisson()`

Examples

```
## set up a hurdle negative binomial distribution
X <- HurdleNegativeBinomial(mu = 2.5, theta = 1, pi = 0.75)
X

## standard functions
pdf(X, 0:8)
cdf(X, 0:8)
```

```
quantile(X, seq(0, 1, by = 0.25))

## cdf() and quantile() are inverses for each other
quantile(X, cdf(X, 3))

## density visualization
plot(0:8, pdf(X, 0:8), type = "h", lwd = 2)

## corresponding sample with histogram of empirical frequencies
set.seed(0)
x <- random(X, 500)
hist(x, breaks = -1:max(x) + 0.5)
```

`HurdlePoisson` *Create a hurdle Poisson distribution*

Description

Hurdle Poisson distributions are frequently used to model counts with many zero observations.

Usage

```
HurdlePoisson(lambda = numeric(), pi = numeric())
```

Arguments

`lambda`	Parameter of the Poisson component of the distribution. Can be any positive number.
`pi`	Zero-hurdle probability, can be any value in `[0, 1]`.

Details

We recommend reading this documentation on https://zeileis.github.io/distributions3/, where the math will render with additional detail.

In the following, let X be a hurdle Poisson random variable with parameter `lambda` = λ.

Support: $\{0, 1, 2, 3, ...\}$

Mean:

$$\lambda \cdot \frac{\pi}{1 - e^{-\lambda}}$$

Variance: $m \cdot (\lambda + 1 - m)$, where m is the mean above.

Probability mass function (p.m.f.): $P(X = 0) = 1 - \pi$ and for $k > 0$

$$P(X = k) = \pi \cdot \frac{f(k; \lambda)}{1 - f(0; \lambda)}$$

where $f(k; \lambda)$ is the p.m.f. of the Poisson distribution.

Cumulative distribution function (c.d.f.): $P(X \leq 0) = 1 - \pi$ and for $k > 0$

$$P(X \leq k) = 1 - \pi + \pi \cdot \frac{F(k; \lambda) - F(0; \lambda)}{1 - F(0; \lambda)}$$

where $F(k; \lambda)$ is the c.d.f. of the Poisson distribution.

Moment generating function (m.g.f.):

Omitted for now.

Value

A HurdlePoisson object.

See Also

Other discrete distributions: Bernoulli(), Binomial(), Categorical(), Geometric(), HurdleNegativeBinomial(), HyperGeometric(), Multinomial(), NegativeBinomial(), Poisson(), PoissonBinomial(), ZINegativeBinomial(), ZIPoisson(), ZTNegativeBinomial(), ZTPoisson()

Examples

```
## set up a hurdle Poisson distribution
X <- HurdlePoisson(lambda = 2.5, pi = 0.75)
X

## standard functions
pdf(X, 0:8)
cdf(X, 0:8)
quantile(X, seq(0, 1, by = 0.25))

## cdf() and quantile() are inverses for each other
quantile(X, cdf(X, 3))

## density visualization
plot(0:8, pdf(X, 0:8), type = "h", lwd = 2)

## corresponding sample with histogram of empirical frequencies
set.seed(0)
x <- random(X, 500)
hist(x, breaks = -1:max(x) + 0.5)
```

HyperGeometric *Create a HyperGeometric distribution*

Description

To understand the HyperGeometric distribution, consider a set of r objects, of which m are of the type I and n are of the type II. A sample with size k ($k < r$) with no replacement is randomly chosen. The number of observed type I elements observed in this sample is set to be our random variable X. For example, consider that in a set of 20 car parts, there are 4 that are defective (type I). If we take a sample of size 5 from those car parts, the probability of finding 2 that are defective will be given by the HyperGeometric distribution (needs double checking).

Usage

```
HyperGeometric(m = numeric(), n = numeric(), k = numeric())
```

Arguments

m	The number of type I elements available.
n	The number of type II elements available.
k	The size of the sample taken.

Details

We recommend reading this documentation on https://zeileis.github.io/distributions3/, where the math will render with additional detail and much greater clarity.

In the following, let X be a HyperGeometric random variable with success probability p = $p = m/(m+n)$.

Support: $x \in \{\max(0, k-n), \ldots, \min(k, m)\}$

Mean: $\frac{km}{n+m} = kp$

Variance: $\frac{km(n)(n+m-k)}{(n+m)^2(n+m-1)} = kp(1-p)(1-\frac{k-1}{m+n-1})$

Probability mass function (p.m.f):

$$P(X = x) = \frac{\binom{m}{x}\binom{n}{k-x}}{\binom{m+n}{k}}$$

Cumulative distribution function (c.d.f):

$$P(X \leq k) \approx \Phi\Big(\frac{x - kp}{\sqrt{kp(1-p)}}\Big)$$

Moment generating function (m.g.f):

Not useful.

Value

A HyperGeometric object.

See Also

Other discrete distributions: Bernoulli(), Binomial(), Categorical(), Geometric(), HurdleNegativeBinomial(), HurdlePoisson(), Multinomial(), NegativeBinomial(), Poisson(), PoissonBinomial(), ZINegativeBinomial(), ZIPoisson(), ZTNegativeBinomial(), ZTPoisson()

Examples

```
set.seed(27)

X <- HyperGeometric(4, 5, 8)
X

random(X, 10)

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

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

`is_discrete` *Determine whether a distribution is discrete or continuous*

Description

Generic functions for determining whether a certain probability distribution is discrete or continuous, respectively.

Usage

```
is_discrete(d, ...)

is_continuous(d, ...)
```

Arguments

`d`	An object. The package provides methods for distribution objects such as those from `Normal()` or `Binomial()` etc.
`...`	Arguments passed to methods. Unevaluated arguments will generate a warning to catch mispellings or other possible errors.

Details

The generic function `is_discrete` is intended to return `TRUE` for every distribution whose entire support is discrete and `FALSE` otherwise. Analogously, `is_continuous` is intended to return `TRUE` for every distribution whose entire support is continuous and `FALSE` otherwise. For mixed discrete-continuous distributions both methods should return `FALSE`.

Methods for both generics are provided for all `distribution` classes set up in this package.

Value

A logical vector indicating whether the distribution(s) in `d` is/are discrete or continuous, respectively.

Examples

```
X <- Normal()
is_discrete(X)
is_continuous(X)
Y <- Binomial(size = 10, p = c(0.2, 0.5, 0.8))
is_discrete(Y)
is_continuous(Y)
```

---

is_distribution *Is an object a distribution?*

Description

`is_distribution` tests if `x` inherits from `"distribution"`.

Usage

```
is_distribution(x)
```

Arguments

`x`	An object to test.

Examples

```
Z <- Normal()

is_distribution(Z)
is_distribution(1L)
```

Logistic *Create a Logistic distribution*

Description

A continuous distribution on the real line. For binary outcomes the model given by $P(Y = 1|X) = F(X\beta)$ where F is the Logistic `cdf()` is called *logistic regression*.

Usage

```
Logistic(location = 0, scale = 1)
```

Arguments

`location`	The location parameter for the distribution. For Logistic distributions, the location parameter is the mean, median and also mode. Defaults to zero.
`scale`	The scale parameter for the distribution. Defaults to one.

Details

We recommend reading this documentation on https://zeileis.github.io/distributions3/, where the math will render with additional detail and much greater clarity.

In the following, let X be a Logistic random variable with `location` = μ and `scale` = s.

Support: R, the set of all real numbers

Mean: μ

Variance: $s^2\pi^2/3$

Probability density function (p.d.f):

$$f(x) = \frac{e^{-(\frac{x-\mu}{s})}}{s[1 + \exp(-(\frac{x-\mu}{s}))]^2}$$

Cumulative distribution function (c.d.f):

$$F(t) = \frac{1}{1 + e^{-(\frac{t-\mu}{s})}}$$

Moment generating function (m.g.f):

$$E(e^{tX}) = e^{\mu t}\beta(1 - st, 1 + st)$$

where $\beta(x, y)$ is the Beta function.

Value

A `Logistic` object.

See Also

Other continuous distributions: `Beta()`, `Cauchy()`, `ChiSquare()`, `Erlang()`, `Exponential()`, `FisherF()`, `Frechet()`, `GEV()`, `GP()`, `Gamma()`, `Gumbel()`, `LogNormal()`, `Normal()`, `RevWeibull()`, `SinhArcsinh()`, `StudentsT()`, `Tukey()`, `Uniform()`, `Weibull()`

Examples

```
set.seed(27)

X <- Logistic(2, 4)
X

random(X, 10)

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

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

`LogNormal` *Create a LogNormal distribution*

Description

A random variable created by exponentiating a `Normal()` distribution. Taking the log of LogNormal data returns in `Normal()` data.

Usage

```
LogNormal(log_mu = 0, log_sigma = 1)
```

Arguments

`log_mu`	The location parameter, written μ in textbooks. Can be any real number. Defaults to `0`.
`log_sigma`	The scale parameter, written σ in textbooks. Can be any positive real number. Defaults to `1`.

Details

We recommend reading this documentation on https://zeileis.github.io/distributions3/, where the math will render with additional detail and much greater clarity.

In the following, let X be a LogNormal random variable with success probability `p` = p.

Support: R^+

Mean: $\exp(\mu + \sigma^2/2)$

Variance: $[\exp(\sigma^2) - 1]\exp(2\mu + \sigma^2)$

Probability density function (p.d.f):

$$f(x) = \frac{1}{x\sigma\sqrt{2\pi}} \exp\left(-\frac{(\log x - \mu)^2}{2\sigma^2}\right)$$

Cumulative distribution function (c.d.f):

$$F(x) = \frac{1}{2} + \frac{1}{2\sqrt{pi}} \int_{-x}^{x} e^{-t^2} dt$$

Moment generating function (m.g.f): Undefined.

Value

A `LogNormal` object.

See Also

Other continuous distributions: Beta(), Cauchy(), ChiSquare(), Erlang(), Exponential(), FisherF(), Frechet(), GEV(), GP(), Gamma(), Gumbel(), Logistic(), Normal(), RevWeibull(), SinhArcsinh(), StudentsT(), Tukey(), Uniform(), Weibull()

Examples

```
set.seed(27)

X <- LogNormal(0.3, 2)
X

random(X, 10)

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

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

---

log_likelihood *Compute the (log-)likelihood of a probability distribution given data*

Description

Functions for computing the (log-)likelihood based on a distribution object and observed data. The log-likelihood is computed as the sum of log-density contributions and the likelihood by taking the exponential thereof.

Usage

```
log_likelihood(d, x, ...)

likelihood(d, x, ...)
```

Arguments

d	An object. The package provides methods for distribution objects such as those from Normal() or Binomial() etc.
x	A vector of data to compute the likelihood.
...	Arguments passed to methods. Unevaluated arguments will generate a warning to catch mispellings or other possible errors.

Value

Numeric value of the (log-)likelihood.

Examples

```
## distribution object
X <- Normal()
## sum of log_pdf() contributions
log_likelihood(X, c(-1, 0, 0, 0, 3))
## exp of log_likelihood()
likelihood(X, c(-1, 0, 0, 0, 3))
```

`mean.distribution` *Methods for numerically calculating central moments of probability distributions*

Description

Several fallback S3 methods for numerically calculating/approximating central moments (mean, variance, skewness, kurtosis) if no dedicated S3 method for a certain distribution exists.

Usage

```
## 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`	An object of class `distribution`.
`...`	Forwarded to internal function `distribution_calculate_moments()` when calculating moments (i.e., mean, variance, skewness, kurtosis), else ignored (see `distribution_calculate_moments()` for available options/arguments).

Value

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

Examples

```
## For demonstration using the Normal distribution, comparing the
## numerically calculated central moments to their analytic solution.

## Mockup class
MyNormal <- function(mu, sigma) {
    structure(data.frame(mu = mu, sigma = sigma),
              class = c("MyNormal", "distribution"))
}

## Adding minimal required methods (borrowed from class Normal)
registerS3method("cdf", "MyNormal",
                 getS3method("cdf", "Normal"))
registerS3method("is_discrete", "MyNormal",
                 getS3method("is_discrete", "Normal"))
registerS3method("support", "MyNormal",
                 getS3method("support", "Normal"))

## Drawing parameters
set.seed(6020)
N     <- 20L
mu    <- runif(N, -30, 30)
sigma <- runif(N, 0.5, 20)^2 / 100 + 0.2

## Setting up distributions objects
n20 <- MyNormal(mu = mu,  sigma = sigma)
N20 <- Normal(mu = mu, sigma = sigma)

## Calculating moments for first distribution (n20[1], N20[1])
do.call(cbind,
    list(numerically  = list(mean = mean(n20[1]),
                             variance = variance(n20[1L]),
                             skewness = skewness(n20[1L]),
                             kurtosis = kurtosis(n20[1L])),
         analytically = list(mean = mean(N20[1]),
                             variance = variance(N20[1L]),
                             skewness = skewness(N20[1L]),
                             kurtosis = kurtosis(N20[1L])))
)

## Visual comparison
par(ask = TRUE)

plot(mean(n20), mean(N20),
     main = "Mean: Normal vs. MyNormal",
     xlab = "numerically approximated mean",
     ylab = "analytic mean")
abline(0, 1, col = 2, lty = 2)

plot(mean(n20, gridsize = 10L), mean(N20),
     main = "Mean: Normal vs. MyNormal\nminimal gridsize (less precise)",
     xlab = "numerically approximated mean",
```

```
     ylab = "analytic mean")
abline(0, 1, col = 2, lty = 2)

plot(variance(n20), variance(N20),
     main = "Variance: Normal vs. MyNormal",
     xlab = "numerically approximated variance",
     ylab = "analytic variance")
abline(0, 1, col = 2, lty = 2)

plot(skewness(n20), skewness(N20),
     main = "Skewness: Normal vs. MyNormal",
     xlab = "numerically approximated skewness",
     ylab = "analytic skewness")
abline(0, 1, col = 2, lty = 2)

plot(kurtosis(n20), kurtosis(N20),
     main = "Kurtosis: Normal vs. MyNormal",
     xlab = "numerically approximated kurtosis",
     ylab = "analytic kurtosis")
abline(h = 0, col = 2, lty = 2)

plot(kurtosis(n20, gridsize = 150L), kurtosis(N20),
     main = "Kurtosis: Normal vs. MyNormal\nreduced gridsize (less precise)",
     xlab = "numerically approximated kurtosis",
     ylab = "analytic kurtosis")
abline(h = 0, col = 2, lty = 2)
```

`Multinomial` *Create a Multinomial distribution*

Description

The multinomial distribution is a generalization of the binomial distribution to multiple categories. It is perhaps easiest to think that we first extend a `Bernoulli()` distribution to include more than two categories, resulting in a `Categorical()` distribution. We then extend repeat the Categorical experiment several (n) times.

Usage

```
Multinomial(size = numeric(), p = numeric())
```

Arguments

`size`	The number of trials. Must be an integer greater than or equal to one. When `size = 1L`, the Multinomial distribution reduces to the categorical distribution (also called the discrete uniform). Often called `n` in textbooks.
`p`	A vector of success probabilities for each trial. `p` can take on any positive value, and the vector is normalized internally.

Details

We recommend reading this documentation on https://zeileis.github.io/distributions3/, where the math will render with additional detail and much greater clarity.

In the following, let $X = (X_1, ..., X_k)$ be a Multinomial random variable with success probability `p` = p. Note that p is vector with k elements that sum to one. Assume that we repeat the Categorical experiment `size` = n times.

Support: Each X_i is in $0, 1, 2, ..., n$.

Mean: The mean of X_i is np_i.

Variance: The variance of X_i is $np_i(1 - p_i)$. For $i \neq j$, the covariance of X_i and X_j is $-np_ip_j$.

Probability mass function (p.m.f):

$$P(X_1 = x_1, ..., X_k = x_k) = \frac{n!}{x_1!x_2!...x_k!} p_1^{x_1} \cdot p_2^{x_2} \cdot ... \cdot p_k^{x_k}$$

Cumulative distribution function (c.d.f):

Omitted for multivariate random variables for the time being.

Moment generating function (m.g.f):

$$E(e^{tX}) = \left(\sum_{i=1}^{k} p_i e^{t_i} \right)^n$$

Value

A `Multinomial` object.

See Also

Other discrete distributions: `Bernoulli()`, `Binomial()`, `Categorical()`, `Geometric()`, `HurdleNegativeBinomial()`, `HurdlePoisson()`, `HyperGeometric()`, `NegativeBinomial()`, `Poisson()`, `PoissonBinomial()`, `ZINegativeBinomial()`, `ZIPoisson()`, `ZTNegativeBinomial()`, `ZTPoisson()`

Examples

```
set.seed(27)

X <- Multinomial(size = 5, p = c(0.3, 0.4, 0.2, 0.1))
X

random(X, 10)

# pdf(X, 2)
# log_pdf(X, 2)
```

`NegativeBinomial` *Create a negative binomial distribution*

Description

A generalization of the geometric distribution. It is the number of failures in a sequence of i.i.d. Bernoulli trials before a specified target number (r) of successes occurs.

Usage

```
NegativeBinomial(size = numeric(), p = 0.5, mu = size)
```

Arguments

`size`	The target number of successes (greater than 0) until the experiment is stopped. Denoted r below.
`p`	The success probability for a given trial. `p` can be any value in `[0, 1]`, and defaults to `0.5`.
`mu`	Alternative parameterization via the non-negative mean of the distribution (instead of the probability `p`), defaults to `size`.

Details

We recommend reading this documentation on https://zeileis.github.io/distributions3/, where the math will render with additional detail and much greater clarity.

In the following, let X be a negative binomial random variable with success probability `p` = p.

Support: $\{0, 1, 2, 3, ...\}$

Mean: $\frac{(1-p)r}{p} = \mu$

Variance: $\frac{(1-p)r}{p^2}$

Probability mass function (p.m.f.):

$$f(k) = \binom{k+r-1}{k} \cdot p^r (1-p)^k$$

Cumulative distribution function (c.d.f.):

Omitted for now.

Moment generating function (m.g.f.):

$$\left(\frac{p}{1-(1-p)e^t}\right)^r, t < -\log(1-p)$$

Alternative parameterization: Sometimes, especially when used in regression models, the negative binomial distribution is parameterized by its mean μ (as listed above) plus the size parameter r. This implies a success probability of $p = r/(r+\mu)$. This can also be seen as a generalization

of the Poisson distribution where the assumption of equidispersion (i.e., variance equal to mean) is relaxed. The negative binomial distribution is overdispersed (i.e., variance greater than mean) and its variance can also be written as $\mu + 1/r\mu^2$. The Poisson distribution is then obtained as r goes to infinity. Note that in this view it is natural to also allow for non-integer r parameters. The factorials in the equations above are then expressed in terms of the gamma function.

Value

A `NegativeBinomial` object.

See Also

Other discrete distributions: `Bernoulli()`, `Binomial()`, `Categorical()`, `Geometric()`, `HurdleNegativeBinomial()`, `HurdlePoisson()`, `HyperGeometric()`, `Multinomial()`, `Poisson()`, `PoissonBinomial()`, `ZINegativeBinomial()`, `ZIPoisson()`, `ZTNegativeBinomial()`, `ZTPoisson()`

Examples

```
set.seed(27)

X <- NegativeBinomial(size = 5, p = 0.1)
X

random(X, 10)

pdf(X, 50)
log_pdf(X, 50)

cdf(X, 50)
quantile(X, 0.7)

## alternative parameterization of X
Y <- NegativeBinomial(mu = 45, size = 5)
Y
cdf(Y, 50)
quantile(Y, 0.7)
```

`Normal` *Create a Normal distribution*

Description

The Normal distribution is ubiquitous in statistics, partially because of the central limit theorem, which states that sums of i.i.d. random variables eventually become Normal. Linear transformations of Normal random variables result in new random variables that are also Normal. If you are taking an intro stats course, you'll likely use the Normal distribution for Z-tests and in simple linear regression. Under regularity conditions, maximum likelihood estimators are asymptotically Normal. The Normal distribution is also called the gaussian distribution.

Usage

```
Normal(mu = 0, sigma = 1)
```

Arguments

`mu`	The location parameter, written μ in textbooks, which is also the mean of the distribution. Can be any real number. Defaults to `0`.
`sigma`	The scale parameter, written σ in textbooks, which is also the **standard deviation** of the distribution. Can be any positive number. Defaults to `1`. If you would like a Normal distribution with **variance** σ^2, be sure to take the square root, as this is a common source of errors.

Details

We recommend reading this documentation on https://zeileis.github.io/distributions3/, where the math will render with additional detail and much greater clarity.

In the following, let X be a Normal random variable with mean `mu` = μ and standard deviation `sigma` = σ.

Support: R, the set of all real numbers

Mean: μ

Variance: σ^2

Probability density function (p.d.f):

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-(x-\mu)^2/2\sigma^2}$$

Cumulative distribution function (c.d.f):

The cumulative distribution function has the form

$$F(t) = \int_{-\infty}^{t} \frac{1}{\sqrt{2\pi\sigma^2}} e^{-(x-\mu)^2/2\sigma^2} dx$$

but this integral does not have a closed form solution and must be approximated numerically. The c.d.f. of a standard Normal is sometimes called the "error function". The notation $\Phi(t)$ also stands for the c.d.f. of a standard Normal evaluated at t. Z-tables list the value of $\Phi(t)$ for various t.

Moment generating function (m.g.f):

$$E(e^{tX}) = e^{\mu t + \sigma^2 t^2/2}$$

Value

A `Normal` object.

See Also

Other continuous distributions: Beta(), Cauchy(), ChiSquare(), Erlang(), Exponential(), FisherF(), Frechet(), GEV(), GP(), Gamma(), Gumbel(), LogNormal(), Logistic(), RevWeibull(), SinhArcsinh(), StudentsT(), Tukey(), Uniform(), Weibull()

Examples

```
set.seed(27)

X <- Normal(5, 2)
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: calculating p-values for two-sided Z-test

# here the null hypothesis is H_0: mu = 3
# and we assume sigma = 2

# exactly the same as: Z <- Normal(0, 1)
Z <- Normal()

# data to test
x <- c(3, 7, 11, 0, 7, 0, 4, 5, 6, 2)
nx <- length(x)

# calculate the z-statistic
z_stat <- (mean(x) - 3) / (2 / sqrt(nx))
z_stat

# calculate the two-sided p-value
1 - cdf(Z, abs(z_stat)) + cdf(Z, -abs(z_stat))

# exactly equivalent to the above
2 * cdf(Z, -abs(z_stat))

# p-value for one-sided test
# H_0: mu <= 3   vs   H_A: mu > 3
1 - cdf(Z, z_stat)

# p-value for one-sided test
```

```
# H_0: mu >= 3   vs   H_A: mu < 3
cdf(Z, z_stat)

### example: calculating a 88 percent Z CI for a mean

# same `x` as before, still assume `sigma = 2`

# lower-bound
mean(x) - quantile(Z, 1 - 0.12 / 2) * 2 / sqrt(nx)

# upper-bound
mean(x) + quantile(Z, 1 - 0.12 / 2) * 2 / sqrt(nx)

# equivalent to
mean(x) + c(-1, 1) * quantile(Z, 1 - 0.12 / 2) * 2 / sqrt(nx)

# also equivalent to
mean(x) + quantile(Z, 0.12 / 2) * 2 / sqrt(nx)
mean(x) + quantile(Z, 1 - 0.12 / 2) * 2 / sqrt(nx)

### generating random samples and plugging in ks.test()

set.seed(27)

# generate a random sample
ns <- random(Normal(3, 7), 26)

# test if sample is Normal(3, 7)
ks.test(ns, pnorm, mean = 3, sd = 7)

# test if sample is gamma(8, 3) using base R pgamma()
ks.test(ns, pgamma, shape = 8, rate = 3)

### MISC

# note that the cdf() and quantile() functions are inverses
cdf(X, quantile(X, 0.7))
quantile(X, cdf(X, 7))
```

---

pdf *Evaluate the probability density of a probability distribution*

Description

Generic function for computing probability density function (PDF) contributions based on a distribution object and observed data.

Usage

```
pdf(d, x, drop = TRUE, ...)
```

```
log_pdf(d, x, ...)

pmf(d, x, ...)
```

Arguments

d	An object. The package provides methods for distribution objects such as those from Normal() or Binomial() etc.
x	A vector of elements whose probabilities you would like to determine given the distribution d.
drop	logical. Should the result be simplified to a vector if possible?
...	Arguments passed to methods. Unevaluated arguments will generate a warning to catch mispellings or other possible errors.

Details

The generic function pdf() computes the probability density, both for continuous and discrete distributions. pmf() (for the probability mass function) is an alias that just calls pdf() internally. For computing log-density contributions (e.g., to a log-likelihood) either pdf(..., log = TRUE) can be used or the generic function log_pdf().

Value

Probabilities corresponding to the vector x.

Examples

```
## distribution object
X <- Normal()
## probability density
pdf(X, c(1, 2, 3, 4, 5))
pmf(X, c(1, 2, 3, 4, 5))
## log-density
pdf(X, c(1, 2, 3, 4, 5), log = TRUE)
log_pdf(X, c(1, 2, 3, 4, 5))
```

---

pdf.Bernoulli *Evaluate the probability mass function of a Bernoulli distribution*

Description

Evaluate the probability mass function of a Bernoulli distribution

Usage

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

## S3 method for class 'Bernoulli'
log_pdf(d, x, drop = TRUE, elementwise = NULL, ...)
```

Arguments

d	A Bernoulli object created by a call to Bernoulli().
x	A vector of elements whose probabilities you would like to determine given the distribution d.
drop	logical. Should the result be simplified to a vector if possible?
elementwise	logical. Should each distribution in d be evaluated at all elements of x (elementwise = FALSE, yielding a matrix)? Or, if d 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.
...	Arguments to be passed to dbinom. Unevaluated arguments will generate a warning to catch mispellings or other possible errors.

Value

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.

Examples

```
set.seed(27)

X <- Bernoulli(0.7)
X

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

random(X, 10)
pdf(X, 1)
log_pdf(X, 1)
cdf(X, 0)
quantile(X, 0.7)

cdf(X, quantile(X, 0.7))
quantile(X, cdf(X, 0.7))
```

pdf.Beta *Evaluate the probability mass function of a Beta distribution*

Description

Evaluate the probability mass function of a Beta distribution

Usage

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

## S3 method for class 'Beta'
log_pdf(d, x, drop = TRUE, elementwise = NULL, ...)
```

Arguments

d	A Beta object created by a call to Beta().
x	A vector of elements whose probabilities you would like to determine given the distribution d.
drop	logical. Should the result be simplified to a vector if possible?
elementwise	logical. Should each distribution in d be evaluated at all elements of x (elementwise = FALSE, yielding a matrix)? Or, if d 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.
...	Arguments to be passed to dbeta. Unevaluated arguments will generate a warning to catch mispellings or other possible errors.

Value

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.

Examples

```
set.seed(27)

X <- Beta(1, 2)
X

random(X, 10)

pdf(X, 0.7)
log_pdf(X, 0.7)
```

```
cdf(X, 0.7)
quantile(X, 0.7)

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

cdf(X, quantile(X, 0.7))
quantile(X, cdf(X, 0.7))
```

`pdf.Binomial` *Evaluate the probability mass function of a Binomial distribution*

Description

Evaluate the probability mass function of a Binomial distribution

Usage

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

## S3 method for class 'Binomial'
log_pdf(d, x, drop = TRUE, elementwise = NULL, ...)
```

Arguments

`d`	A `Binomial` object created by a call to `Binomial()`.
`x`	A vector of elements whose probabilities you would like to determine given the distribution `d`.
`drop`	logical. Should the result be simplified to a vector if possible?
`elementwise`	logical. Should each distribution in `d` be evaluated at all elements of `x` (`elementwise = FALSE`, yielding a matrix)? Or, if `d` 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.
`...`	Arguments to be passed to `dbinom`. Unevaluated arguments will generate a warning to catch mispellings or other possible errors.

Value

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.

Examples

```
set.seed(27)

X <- Binomial(10, 0.2)
X

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

random(X, 10)

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

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

cdf(X, quantile(X, 0.7))
quantile(X, cdf(X, 7))
```

---

pdf.Categorical *Evaluate the probability mass function of a Categorical discrete distribution*

Description

Evaluate the probability mass function of a Categorical discrete distribution

Usage

```
## S3 method for class 'Categorical'
pdf(d, x, ...)

## S3 method for class 'Categorical'
log_pdf(d, x, ...)
```

Arguments

d	A Categorical object created by a call to Categorical().
x	A vector of elements whose probabilities you would like to determine given the distribution d.
...	Unused. Unevaluated arguments will generate a warning to catch mispellings or other possible errors.

Value

A vector of probabilities, one for each element of x.

Examples

```
set.seed(27)

X <- Categorical(1:3, p = c(0.4, 0.1, 0.5))
X

Y <- Categorical(LETTERS[1:4])
Y

random(X, 10)
random(Y, 10)

pdf(X, 1)
log_pdf(X, 1)

cdf(X, 1)
quantile(X, 0.5)

# cdfs are only defined for numeric sample spaces. this errors!
# cdf(Y, "a")

# same for quantiles. this also errors!
# quantile(Y, 0.7)
```

---

pdf.Cauchy *Evaluate the probability mass function of a Cauchy distribution*

Description

Evaluate the probability mass function of a Cauchy distribution

Usage

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

## S3 method for class 'Cauchy'
log_pdf(d, x, drop = TRUE, elementwise = NULL, ...)
```

Arguments

d	A Cauchy object created by a call to Cauchy().
x	A vector of elements whose probabilities you would like to determine given the distribution d.
drop	logical. Should the result be simplified to a vector if possible?

elementwise	logical. Should each distribution in d be evaluated at all elements of x (elementwise = FALSE, yielding a matrix)? Or, if d 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.
...	Arguments to be passed to dcauchy. Unevaluated arguments will generate a warning to catch mispellings or other possible errors.

Value

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.

Examples

```
set.seed(27)

X <- Cauchy(10, 0.2)
X

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

random(X, 10)

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

cdf(X, 2)
quantile(X, 0.7)

cdf(X, quantile(X, 0.7))
quantile(X, cdf(X, 7))
```

---

pdf.ChiSquare *Evaluate the probability mass function of a chi square distribution*

Description

Evaluate the probability mass function of a chi square distribution

Usage

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

```
## S3 method for class 'ChiSquare'
log_pdf(d, x, drop = TRUE, elementwise = NULL, ...)
```

Arguments

d	A ChiSquare object created by a call to ChiSquare().
x	A vector of elements whose probabilities you would like to determine given the distribution d.
drop	logical. Should the result be simplified to a vector if possible?
elementwise	logical. Should each distribution in d be evaluated at all elements of x (elementwise = FALSE, yielding a matrix)? Or, if d 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.
...	Arguments to be passed to dchisq. Unevaluated arguments will generate a warning to catch mispellings or other possible errors.

Value

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.

Examples

```
set.seed(27)

X <- ChiSquare(5)
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)

cdf(X, quantile(X, 0.7))
quantile(X, cdf(X, 7))
```

pdf.distribution *Methods for numerically approximating distribution functions*

Description

S3 methods for distribution objects used if certain (analytic) distribution functions are not available. Allows to use all **distributions3** methods even if there are no dedicated methods for a certain distribution. See section 'Details' for more information.

Usage

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

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

## 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 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)? By default (if elementwise = NULL) it is set to elementwise = TRUE 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	NULL or positive integer. 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 uniroot.
tol	numeric. Desired accuracy for uniroot.
maxit	integer. 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 $P[X \geq x]$.

Details

For distribution classes that do not (or not yet) provide all S3 methods for the usual generic functions (pdf, cdf, quantile, random), a fallback method for the general distribution class is provided. This fallback method numerically approximates the probability density function (PDF), cumulative distribution function (CDF), quantile function, or random number generation, provided that at least some of the methods are available (see below).

S3 methods for is_discrete and support are mandatory. Besides these two (fairly simple) methods, the following methods are required.

Continuous distributions:

- provide a cdf.* method,
- or a pdf.* and quantile.* method.

Discrete distributions:

- provide a pdf.* method,
- or a cdf.* method.

Any additional S3 method provided on top of the minimal requirements will always be leveraged if available (i.e., the more dedicated/analytical methods provided the better).

Examples

```
## ------------- custom Normal distribution (MyNormal1) ---------------
## -------------------------- using cdf -------------------------------

## Constructor function for new 'MyNormal1' distribution
MyNormal1 <- function(mu, sigma) {
    d <- data.frame(mu = mu, sigma = sigma)
    class(d) <- c("MyNormal1", "distribution")
    return(d)
}

## Additionally required S3 methods (borrowed from class Normal)
registerS3method("cdf", "MyNormal1",
                 getS3method("cdf", class = "Normal"))
registerS3method("is_discrete", "MyNormal1",
                 getS3method("is_discrete", class = "Normal"))
registerS3method("support", "MyNormal1",
                 getS3method("support", class = "Normal"))

## Constructing objects; three normal distributions with
## mean c(1, 2, 3) and standard deviation c(1, 2.5, 5).
## mn3: Based on MyNormal1 where only cdf, id_discrete, and support
##      are defined.
## n3:  Analytic solution (Normal()) with analytic functions
##      for all distribution functions (pdf, cdf, quantile) as well as for
##      the first four central moments (mean, variance, skewness, kurtosis)
mn3 <- MyNormal1(mu = 1:3, sigma = c(1, 2.5, 5))
n3  <- Normal(mu = 1:3, sigma = c(1, 2.5, 5))

## Class 'MyNormal1' knows the analytic cdf:
cdf(mn3, x = 2)
identical(cdf(mn3, x = 2), cdf(n3, x = 2))

## Calculating probability at x = 2
pdf(mn3, x = 2) ## Numeric approximation
pdf(n3,  x = 2) ## Analytic solution
pdf(mn3, 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(mn3, probs = probs) ## Numeric approximation
quantile(n3,  probs = probs) ## Analytic solution

probs2 <- seq(0.01, 0.99, by = 0.01)
qmn3   <- quantile(mn3, probs = probs2) ## Numeric approximation
qn3    <- quantile(n3,  probs = probs2) ## Analytic solution
range(qmn3 - qn3) ## Range of pairwise differences/precision

## Central moments
cbind(mean     = mean(mn3),     variance = variance(mn3),
      skewness = skewness(mn3), kurtosis = kurtosis(mn3))
```

```
## Visual comparison: density
x    <- seq(-3, 5, by = 0.1)
pmn3 <- pdf(mn3[1], x = x)
pn3  <- pdf(n3[1],  x = x)
dpdf <- data.frame(x          = x,
                   analytical = pdf(n3[1],  x = x),
                   numerical  = pdf(mn3[1], x = x))

matplot(dpdf[, 1], dpdf[, -1],  type = "l", col = 1:2,
        xlab = NA, ylab = "density", lty = 1, lwd = 2:1,
        main = "Density function (mean = 1, sigma = 1)")
legend("bottom", legend = names(dpdf)[-1], bty = "n",
       col = 1:2, lty = 1, lwd = 2:1)

## Visual comparison: quantiles
probs     <- c(0.001, seq(0.01, 0.99, by = 0.01), 0.999)
dquantile <- data.frame(probs      = probs,
                        analytical = quantile(n3[1],  probs = probs),
                        numerical  = quantile(mn3[1], probs = probs))

matplot(dquantile[, -1], dquantile[, 1], type = "l", col = 1:2,
        xlab = NA, ylab = "probability", lty = 1, lwd = 2:1,
        main = "Quantile function (mean = 1, sigma = 1)")
legend("bottom", legend = names(dquantile)[-1], bty = "n",
       col = 1:2, lwd = 2:1)

## Visual comparison: quantile-quantile plot
plot(dquantile[, "analytical"], dquantile[, "analytical"],
     xlab = "numerically approximated quantiles",
     ylab = "theoretical quantiles",
     main = "QQ-plot Normal vs. MyNormal1")
abline(0, 1, col = 2, lty = 2)

## Drawing random numbers (500 on third distribution)
set.seed(6020); rmn <- random(mn3[3], 500L)
set.seed(6020); rn  <- random(n3[3],  500L)

hmn <- hist(rmn, breaks = 15L, plot = FALSE)
hn  <- hist(rn,  breaks = 15L, plot = FALSE)

plot(hmn$mids, hmn$density, type = "l",
     xlab = NA, ylab = "density",
     main = "Density of random numbers: Normal vs. MyNormal1")
lines(hn$mids, hn$density, col = 2)

## ------------ custom Normal distribution (MyNormal2) ----------------
## ------------------- using pdf and quantile -------------------------

## Constructor function for new 'MyNormal2' distribution
MyNormal2 <- function(mu, sigma) {
    d <- data.frame(mu = mu, sigma = sigma)
    class(d) <- c("MyNormal2", "distribution")
    return(d)
```

```
}

## Additionally required S3 methods (borrowed from class Normal)
registerS3method("pdf", "MyNormal2",
                 getS3method("pdf", class = "Normal"))
registerS3method("quantile", "MyNormal2",
                 getS3method("quantile", class = "Normal"))
registerS3method("is_discrete", "MyNormal2",
                 getS3method("is_discrete", class = "Normal"))
registerS3method("support", "MyNormal2",
                 getS3method("support", class = "Normal"))

## Constructing objects; creating three (named) 'MyNormal2' distributions
## with mean c(1, 2, 3) and standard deviation c(1, 2.5, 5) for which only
## pdf, quantile, is_discrete, and support are defined.
mn3 <- MyNormal2(mu = 1:3, sigma = c(1, 2.5, 5))
mn3 <- setNames(mn3, LETTERS[1:3])

random(mn3, n = 3L)
cdf(mn3, x = 2)
pdf(mn3, x = 2)
quantile(mn3, 0.5)
cbind(mean     = mean(mn3),     variance = variance(mn3),
      skewness = skewness(mn3), kurtosis = kurtosis(mn3))



## ------------ custom Poisson distribution (MyPoisson1) --------------
## -------------------------- using pdf -------------------------------

## Custom constructor function for the 'MyPoisson1' distribution
MyPoisson1 <- function(lambda) {
    d <- data.frame(lambda = lambda)
    class(d) <- c("MyPoisson1", "distribution")
    return(d)
}

## Additionally required S3 methods (borrowed from class Poisson)
registerS3method("pdf", "MyPoisson1",
                 getS3method("pdf", class = "Poisson"))
registerS3method("is_discrete", "MyPoisson1",
                 getS3method("is_discrete", class = "Poisson"))
registerS3method("support", "MyPoisson1",
                 getS3method("support", class = "Poisson"))

## Constructing objects; three normal distributions with
## parameter lambda = c(1, 2.5, 5).
## mp3: Based on MyPoisson1 where only pdf, id_discrete, and support
##      are defined.
## p3:  Analytic solution (Poisson()) with analytic
##      functions for all distribution functions (pdf, cdf, quantile)
##      as well as for the first four central moments (mean,
##      variance, skewness, kurtosis)
```

```
mp3 <- MyPoisson1(lambda = c(1, 2.5, 5))
p3  <- Poisson(lambda = c(1, 2.5, 5))

## Class 'MyPoisson1' knows the analytic cdf:
pdf(mp3, x = 2)
identical(pdf(mp3, x = 2), pdf(p3, x = 2))

## Calculating distribution at x = 2
cdf(mp3, x = 2) ## Numeric approximation
cdf(mp3, x = 2) ## Analytic solution
cdf(mp3, x = 2) - cdf(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(mp3, probs = probs) ## Numeric approximation
quantile(p3,  probs = probs) ## Analytic solution

probs2 <- seq(0.01, 0.99, by = 0.01)
qmp3   <- quantile(mp3, probs = probs2) ## Numeric approximation
qp3    <- quantile(p3,  probs = probs2) ## Analytic solution
range(qmp3 - qp3) ## Range of pairwise differences/precision

## Central moments
cbind(mean     = mean(mp3),     variance = variance(mp3),
      skewness = skewness(mp3), kurtosis = kurtosis(mp3))

## Visual comparison: distribution function
x    <- seq(-5, 20, by = 1)
dcdf <- data.frame(x          = x,
                   analytical = cdf(p3[2L], x = x),
                   numerical  = cdf(mp3[2L], x = x))

matplot(dcdf[, 1], dcdf[, -1], type = "s", col = 1:2,
        xlab = NA, ylab = "probability",
        lty = 1, lwd = 2:1, main = "Distribution function (lambda = 2.5)")
legend("bottomright", legend = names(dcdf)[-1], lty = 1,
       col = 1:2, lwd = 2:1, bty = "n")


## Visual comparison: quantile function
probs     <- seq(0.01, 0.99, by = 0.01)
dquantile <- data.frame(probs = probs,
                        analytical = quantile(p3[2L],  probs = probs),
                        numerical  = quantile(mp3[2L], probs = probs))

matplot(dquantile[, -1], dquantile[, 1], type = "s", col = 1:2,
        xlab = NA, ylab = "probability",
        lty = 1, lwd = 2:1, main = "Quantile function (lambda = 2.5)")
legend("bottomright", legend = names(dquantile)[-1], lty = 1,
       col = 1:2, lwd = 2:1, bty = "n")

## Quantile-Quantile plot
probs <- seq(0.01, 0.99, by = 0.01)
```

```
plot(quantile(p3[2], probs), quantile(mp3[2], probs),
     main = "QQ-plot Poisson vs. MyPoisson1",
     xlab = "approximated quantiles", ylab = "theoretical quantiles")
abline(0, 1, col = 2, lty = 2)

## Drawing random numbers (200 on third distribution)
set.seed(6020); rmp <- random(mp3[3], 100L)
set.seed(6020); rp  <- random(p3[3],  100L)

hmp <- hist(rmp, breaks = 0:15, plot = FALSE)
hp  <- hist(rp,  breaks = 0:15, plot = FALSE)

plot(hmp$breaks[-1], hmp$density, type = "s",
     xlab = NA, ylab = "density",
     main = "Density of random numbers: Poisson vs. MyPoisson1")
lines(hp$breaks[-1], hp$density, col = 2, type = "s")


## ------------ custom Poisson distribution (MyPoisson2) --------------
## -------------------------- using cdf -------------------------------

## Custom constructor function for the 'MyPoisson2' distribution
MyPoisson2 <- function(lambda) {
    d <- data.frame(lambda = lambda)
    class(d) <- c("MyPoisson2", "distribution")
    return(d)
}

## Additionally required S3 methods (borrowed from class Poisson)
registerS3method("cdf", "MyPoisson2",
                 getS3method("cdf", class = "Poisson"))
registerS3method("is_discrete", "MyPoisson2",
                 getS3method("is_discrete", class = "Poisson"))
registerS3method("support", "MyPoisson2",
                 getS3method("support", class = "Poisson"))


## Constructing objects; creating three (named) 'MyPoisson2' distributions
## with lambda c(1, 2.5, 5) for which only cdf, is_discrete,
## and support are defined.
mp3 <- MyPoisson2(lambda = c(1, 2.5, 5))
mp3 <- setNames(mp3, LETTERS[4:6])

random(mp3, n = 3L)
cdf(p3, x = 2)
pdf(p3, x = 2)
quantile(p3, 0.5)
cbind(mean     = mean(p3),     variance = variance(p3),
      skewness = skewness(p3), kurtosis = kurtosis(p3))
```

`pdf.Empirical` *Evaluate the probability mass function of an Empirical distribution*

Description

Please see the documentation of `Empirical()` for some properties of the Empirical distribution.

Usage

```
## 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, ...)
```

Arguments

`d`	An `Empirical` object created by a call to `Empirical()`.
`x`	A vector of elements whose probabilities you would like to determine given the distribution `d`.
`drop`	logical. Should the result be simplified to a vector if possible?
`elementwise`	logical. Should each distribution in `d` be evaluated at all elements of `x` (`elementwise = FALSE`, yielding a matrix)? Or, if `d` 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.
`...`	arguments to be passed to `dempirical()`.

Value

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.

See Also

Other Empirical distribution: `Empirical()`, `cdf.Empirical()`, `dempirical()`, `quantile.Empirical()`, `random.Empirical()`, `support.Empirical()`

Examples

```
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
Y4 <- matrix(rnorm(20), ncol = 5,
             dimnames = list(paste0("D_", 1:4), paste0("obs_", 1:5)))
d4 <- Empirical(Y4)
d4

## Data frame
d5 <- Empirical(as.data.frame(Y4))
d5

identical(d4, d5)

mean(d5)
variance(d5)
skewness(d5)
kurtosis(d5)
```

```
pdf(d5, c(-0.5, 0, 0.5, 1)) # Defaults to elementwise = TRUE
pdf(d5, c(-0.5, 0, 0.5, 1), elementwise = FALSE)

cdf(d5, c(-0.5, 0, 0.5, 1)) # Defaults to elementwise = TRUE
cdf(d5, c(-0.5, 0, 0.5, 1), elementwise = FALSE)

quantile(d5, c(0.2, 0.4, 0.6, 0.8)) # Defaults to elementwise = TRUE
quantile(d5, c(0.2, 0.4, 0.6, 0.8), elementwise = FALSE)

## The quantile function is the inverse of the distribution
## function (cdf) if x in Y
set.seed(6020)
Y <- round(rlnorm(20, log(3), log(2)), 1)
d <- Empirical(Y)

cdf(d, 4.0)
quantile(d, cdf(d, 4.0))
```

`pdf.Erlang` *Evaluate the probability mass function of an Erlang distribution*

Description

Evaluate the probability mass function of an Erlang distribution

Usage

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

## S3 method for class 'Erlang'
log_pdf(d, x, drop = TRUE, elementwise = NULL, ...)
```

Arguments

`d`	An `Erlang` object created by a call to `Erlang()`.
`x`	A vector of elements whose probabilities you would like to determine given the distribution `d`.
`drop`	logical. Should the result be simplified to a vector if possible?
`elementwise`	logical. Should each distribution in `d` be evaluated at all elements of `x` (`elementwise = FALSE`, yielding a matrix)? Or, if `d` 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.
`...`	Arguments to be passed to `dgamma`. Unevaluated arguments will generate a warning to catch mispellings or other possible errors.

Value

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.

Examples

```
set.seed(27)

X <- Erlang(5, 2)
X

random(X, 10)

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

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

cdf(X, quantile(X, 0.7))
quantile(X, cdf(X, 7))
```

---

pdf.Exponential *Evaluate the probability density function of an Exponential distribution*

Description

Evaluate the probability density function of an Exponential distribution

Usage

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

## S3 method for class 'Exponential'
log_pdf(d, x, drop = TRUE, elementwise = NULL, ...)
```

Arguments

d	An `Exponential` object created by a call to `Exponential()`.
x	A vector of elements whose probabilities you would like to determine given the distribution `d`.
drop	logical. Should the result be simplified to a vector if possible?

elementwise	logical. Should each distribution in d be evaluated at all elements of x (elementwise = FALSE, yielding a matrix)? Or, if d 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.
...	Arguments to be passed to dexp. Unevaluated arguments will generate a warning to catch mispellings or other possible errors.

Value

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.

Examples

```
set.seed(27)

X <- Exponential(5)
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)

cdf(X, quantile(X, 0.7))
quantile(X, cdf(X, 7))
```

---

pdf.FisherF *Evaluate the probability mass function of an F distribution*

Description

Evaluate the probability mass function of an F distribution

Usage

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

```
## S3 method for class 'FisherF'
log_pdf(d, x, drop = TRUE, elementwise = NULL, ...)
```

Arguments

d	A FisherF object created by a call to FisherF().
x	A vector of elements whose probabilities you would like to determine given the distribution d.
drop	logical. Should the result be simplified to a vector if possible?
elementwise	logical. Should each distribution in d be evaluated at all elements of x (elementwise = FALSE, yielding a matrix)? Or, if d 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.
...	Arguments to be passed to df. Unevaluated arguments will generate a warning to catch mispellings or other possible errors.

Value

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.

Examples

```
set.seed(27)

X <- FisherF(5, 10, 0.2)
X

random(X, 10)

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

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

cdf(X, quantile(X, 0.7))
quantile(X, cdf(X, 7))
```

---

pdf.Frechet *Evaluate the probability mass function of a Frechet distribution*

Description

Evaluate the probability mass function of a Frechet distribution

Usage

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

## S3 method for class 'Frechet'
log_pdf(d, x, drop = TRUE, elementwise = NULL, ...)
```

Arguments

d	A Frechet object created by a call to Frechet().
x	A vector of elements whose probabilities you would like to determine given the distribution d.
drop	logical. Should the result be simplified to a vector if possible?
elementwise	logical. Should each distribution in d be evaluated at all elements of x (elementwise = FALSE, yielding a matrix)? Or, if d 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.
...	Arguments to be passed to dgev. Unevaluated arguments will generate a warning to catch mispellings or other possible errors.

Value

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.

Examples

```
set.seed(27)

X <- Frechet(0, 2)
X

random(X, 10)

pdf(X, 0.7)
log_pdf(X, 0.7)

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

cdf(X, quantile(X, 0.7))
quantile(X, cdf(X, 0.7))
```

pdf.Gamma *Evaluate the probability mass function of a Gamma distribution*

Description

Evaluate the probability mass function of a Gamma distribution

Usage

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

## S3 method for class 'Gamma'
log_pdf(d, x, drop = TRUE, elementwise = NULL, ...)
```

Arguments

d	A Gamma object created by a call to Gamma().
x	A vector of elements whose probabilities you would like to determine given the distribution d.
drop	logical. Should the result be simplified to a vector if possible?
elementwise	logical. Should each distribution in d be evaluated at all elements of x (elementwise = FALSE, yielding a matrix)? Or, if d 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.
...	Arguments to be passed to dgamma. Unevaluated arguments will generate a warning to catch mispellings or other possible errors.

Value

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.

Examples

```
set.seed(27)

X <- Gamma(5, 2)
X

random(X, 10)

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

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

cdf(X, quantile(X, 0.7))
quantile(X, cdf(X, 7))
```

`pdf.Geometric` *Evaluate the probability mass function of a Geometric distribution*

Description

Please see the documentation of `Geometric()` for some properties of the Geometric distribution, as well as extensive examples showing to how calculate p-values and confidence intervals.

Usage

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

## S3 method for class 'Geometric'
log_pdf(d, x, drop = TRUE, elementwise = NULL, ...)
```

Arguments

`d`	A `Geometric` object created by a call to `Geometric()`.
`x`	A vector of elements whose probabilities you would like to determine given the distribution `d`.
`drop`	logical. Should the result be simplified to a vector if possible?
`elementwise`	logical. Should each distribution in `d` be evaluated at all elements of `x` (`elementwise = FALSE`, yielding a matrix)? Or, if `d` 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.
`...`	Arguments to be passed to `dgeom`. Unevaluated arguments will generate a warning to catch mispellings or other possible errors.

Value

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.

See Also

Other Geometric distribution: `cdf.Geometric()`, `quantile.Geometric()`, `random.Geometric()`

Examples

```
set.seed(27)

X <- Geometric(0.3)
X

random(X, 10)

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

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

---

pdf.GEV *Evaluate the probability mass function of a GEV distribution*

Description

Evaluate the probability mass function of a GEV distribution

Usage

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

## S3 method for class 'GEV'
log_pdf(d, x, drop = TRUE, elementwise = NULL, ...)
```

Arguments

d	A GEV object created by a call to GEV().
x	A vector of elements whose probabilities you would like to determine given the distribution d.
drop	logical. Should the result be simplified to a vector if possible?
elementwise	logical. Should each distribution in d be evaluated at all elements of x (elementwise = FALSE, yielding a matrix)? Or, if d 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.
...	Arguments to be passed to dgev. Unevaluated arguments will generate a warning to catch mispellings or other possible errors.

Value

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.

Examples

```
set.seed(27)

X <- GEV(1, 2, 0.1)
X

random(X, 10)

pdf(X, 0.7)
log_pdf(X, 0.7)

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

cdf(X, quantile(X, 0.7))
quantile(X, cdf(X, 0.7))
```

`pdf.GP` *Evaluate the probability mass function of a GP distribution*

Description

Evaluate the probability mass function of a GP distribution

Usage

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

## S3 method for class 'GP'
log_pdf(d, x, drop = TRUE, elementwise = NULL, ...)
```

Arguments

`d`	A `GP` object created by a call to `GP()`.
`x`	A vector of elements whose probabilities you would like to determine given the distribution `d`.
`drop`	logical. Should the result be simplified to a vector if possible?
`elementwise`	logical. Should each distribution in `d` be evaluated at all elements of `x` (`elementwise = FALSE`, yielding a matrix)? Or, if `d` 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.
`...`	Arguments to be passed to `dgp`. Unevaluated arguments will generate a warning to catch mispellings or other possible errors.

Value

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.

Examples

```
set.seed(27)

X <- GP(0, 2, 0.1)
X

random(X, 10)

pdf(X, 0.7)
log_pdf(X, 0.7)

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

cdf(X, quantile(X, 0.7))
quantile(X, cdf(X, 0.7))
```

---

pdf.Gumbel *Evaluate the probability mass function of a Gumbel distribution*

Description

Evaluate the probability mass function of a Gumbel distribution

Usage

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

## S3 method for class 'Gumbel'
log_pdf(d, x, drop = TRUE, elementwise = NULL, ...)
```

Arguments

`d`	A `Gumbel` object created by a call to `Gumbel()`.
`x`	A vector of elements whose probabilities you would like to determine given the distribution `d`.
`drop`	logical. Should the result be simplified to a vector if possible?

elementwise	logical. Should each distribution in d be evaluated at all elements of x (elementwise = FALSE, yielding a matrix)? Or, if d 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.
...	Arguments to be passed to dgev. Unevaluated arguments will generate a warning to catch mispellings or other possible errors.

Value

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.

Examples

```
set.seed(27)

X <- Gumbel(1, 2)
X

random(X, 10)

pdf(X, 0.7)
log_pdf(X, 0.7)

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

cdf(X, quantile(X, 0.7))
quantile(X, cdf(X, 0.7))
```

---

pdf.HurdleNegativeBinomial

Evaluate the probability mass function of a hurdle negative binomial distribution

---

Description

Evaluate the probability mass function of a hurdle negative binomial distribution

Usage

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

## S3 method for class 'HurdleNegativeBinomial'
log_pdf(d, x, drop = TRUE, elementwise = NULL, ...)
```

Arguments

d	A HurdleNegativeBinomial object created by a call to HurdleNegativeBinomial().
x	A vector of elements whose probabilities you would like to determine given the distribution d.
drop	logical. Should the result be simplified to a vector if possible?
elementwise	logical. Should each distribution in d be evaluated at all elements of x (elementwise = FALSE, yielding a matrix)? Or, if d 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.
...	Arguments to be passed to dhnbinom. Unevaluated arguments will generate a warning to catch mispellings or other possible errors.

Value

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.

Examples

```
## set up a hurdle negative binomial distribution
X <- HurdleNegativeBinomial(mu = 2.5, theta = 1, pi = 0.75)
X

## standard functions
pdf(X, 0:8)
cdf(X, 0:8)
quantile(X, seq(0, 1, by = 0.25))

## cdf() and quantile() are inverses for each other
quantile(X, cdf(X, 3))

## density visualization
plot(0:8, pdf(X, 0:8), type = "h", lwd = 2)

## corresponding sample with histogram of empirical frequencies
set.seed(0)
x <- random(X, 500)
hist(x, breaks = -1:max(x) + 0.5)
```

---

pdf.HurdlePoisson *Evaluate the probability mass function of a hurdle Poisson distribution*

Description

Evaluate the probability mass function of a hurdle Poisson distribution

Usage

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

## S3 method for class 'HurdlePoisson'
log_pdf(d, x, drop = TRUE, elementwise = NULL, ...)
```

Arguments

d	A HurdlePoisson object created by a call to HurdlePoisson().
x	A vector of elements whose probabilities you would like to determine given the distribution d.
drop	logical. Should the result be simplified to a vector if possible?
elementwise	logical. Should each distribution in d be evaluated at all elements of x (elementwise = FALSE, yielding a matrix)? Or, if d 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.
...	Arguments to be passed to dhpois. Unevaluated arguments will generate a warning to catch mispellings or other possible errors.

Value

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.

Examples

```
## set up a hurdle Poisson distribution
X <- HurdlePoisson(lambda = 2.5, pi = 0.75)
X

## standard functions
pdf(X, 0:8)
cdf(X, 0:8)
quantile(X, seq(0, 1, by = 0.25))

## cdf() and quantile() are inverses for each other
quantile(X, cdf(X, 3))

## density visualization
plot(0:8, pdf(X, 0:8), type = "h", lwd = 2)

## corresponding sample with histogram of empirical frequencies
set.seed(0)
x <- random(X, 500)
hist(x, breaks = -1:max(x) + 0.5)
```

`pdf.HyperGeometric` *Evaluate the probability mass function of a HyperGeometric distribution*

Description

Please see the documentation of `HyperGeometric()` for some properties of the HyperGeometric distribution, as well as extensive examples showing to how calculate p-values and confidence intervals.

Usage

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

## S3 method for class 'HyperGeometric'
log_pdf(d, x, drop = TRUE, elementwise = NULL, ...)
```

Arguments

`d`	A `HyperGeometric` object created by a call to `HyperGeometric()`.
`x`	A vector of elements whose probabilities you would like to determine given the distribution `d`.
`drop`	logical. Should the result be simplified to a vector if possible?
`elementwise`	logical. Should each distribution in `d` be evaluated at all elements of `x` (`elementwise = FALSE`, yielding a matrix)? Or, if `d` 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.
`...`	Arguments to be passed to `dhyper`. Unevaluated arguments will generate a warning to catch mispellings or other possible errors.

Value

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.

See Also

Other HyperGeometric distribution: `cdf.HyperGeometric()`, `quantile.HyperGeometric()`, `random.HyperGeometric()`

Examples

```
set.seed(27)

X <- HyperGeometric(4, 5, 8)
X

random(X, 10)

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

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

`pdf.Logistic` *Evaluate the probability mass function of a Logistic distribution*

Description

Please see the documentation of `Logistic()` for some properties of the Logistic distribution, as well as extensive examples showing to how calculate p-values and confidence intervals.

Usage

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

## S3 method for class 'Logistic'
log_pdf(d, x, drop = TRUE, elementwise = NULL, ...)
```

Arguments

`d`	A `Logistic` object created by a call to `Logistic()`.
`x`	A vector of elements whose probabilities you would like to determine given the distribution `d`.
`drop`	logical. Should the result be simplified to a vector if possible?
`elementwise`	logical. Should each distribution in `d` be evaluated at all elements of `x` (`elementwise = FALSE`, yielding a matrix)? Or, if `d` 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.
`...`	Arguments to be passed to `dlogis`. Unevaluated arguments will generate a warning to catch mispellings or other possible errors.

Value

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.

See Also

Other Logistic distribution: `cdf.Logistic()`, `quantile.Logistic()`, `random.Logistic()`

Examples

```
set.seed(27)

X <- Logistic(2, 4)
X

random(X, 10)

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

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

`pdf.LogNormal` *Evaluate the probability mass function of a LogNormal distribution*

Description

Please see the documentation of `LogNormal()` for some properties of the LogNormal distribution, as well as extensive examples showing to how calculate p-values and confidence intervals.

Usage

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

## S3 method for class 'LogNormal'
log_pdf(d, x, drop = TRUE, elementwise = NULL, ...)
```

Arguments

`d`	A `LogNormal` object created by a call to `LogNormal()`.
`x`	A vector of elements whose probabilities you would like to determine given the distribution `d`.
`drop`	logical. Should the result be simplified to a vector if possible?

elementwise	logical. Should each distribution in d be evaluated at all elements of x (elementwise = FALSE, yielding a matrix)? Or, if d 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.
...	Arguments to be passed to dlnorm. Unevaluated arguments will generate a warning to catch mispellings or other possible errors.

Value

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.

See Also

Other LogNormal distribution: cdf.LogNormal(), fit_mle.LogNormal(), quantile.LogNormal(), random.LogNormal()

Examples

```
set.seed(27)

X <- LogNormal(0.3, 2)
X

random(X, 10)

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

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

---

pdf.Multinomial *Evaluate the probability mass function of a Multinomial distribution*

Description

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

Usage

```
## S3 method for class 'Multinomial'
pdf(d, x, ...)

## S3 method for class 'Multinomial'
log_pdf(d, x, ...)
```

Arguments

d	A Multinomial object created by a call to Multinomial().
x	A vector of elements whose probabilities you would like to determine given the distribution d.
...	Unused. Unevaluated arguments will generate a warning to catch mispellings or other possible errors.

Value

A vector of probabilities, one for each element of x.

See Also

Other Multinomial distribution: random.Multinomial()

Examples

```
set.seed(27)

X <- Multinomial(size = 5, p = c(0.3, 0.4, 0.2, 0.1))
X

random(X, 10)

# pdf(X, 2)
# log_pdf(X, 2)
```

---

pdf.NegativeBinomial *Evaluate the probability mass function of a NegativeBinomial distribution*

Description

Evaluate the probability mass function of a NegativeBinomial distribution

Usage

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

## S3 method for class 'NegativeBinomial'
log_pdf(d, x, drop = TRUE, elementwise = NULL, ...)
```

Arguments

d	A NegativeBinomial object created by a call to NegativeBinomial().
x	A vector of elements whose probabilities you would like to determine given the distribution d.
drop	logical. Should the result be simplified to a vector if possible?
elementwise	logical. Should each distribution in d be evaluated at all elements of x (elementwise = FALSE, yielding a matrix)? Or, if d 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.
...	Arguments to be passed to dnbinom. Unevaluated arguments will generate a warning to catch mispellings or other possible errors.

Value

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.

See Also

Other NegativeBinomial distribution: cdf.NegativeBinomial(), quantile.NegativeBinomial(), random.NegativeBinomial()

Examples

```
set.seed(27)

X <- NegativeBinomial(size = 5, p = 0.1)
X

random(X, 10)

pdf(X, 50)
log_pdf(X, 50)

cdf(X, 50)
quantile(X, 0.7)

## alternative parameterization of X
Y <- NegativeBinomial(mu = 45, size = 5)
Y
cdf(Y, 50)
quantile(Y, 0.7)
```

pdf.Normal *Evaluate the probability mass function of a Normal distribution*

Description

Please see the documentation of `Normal()` for some properties of the Normal distribution, as well as extensive examples showing to how calculate p-values and confidence intervals.

Usage

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

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

Arguments

`d`	A `Normal` object created by a call to `Normal()`.
`x`	A vector of elements whose probabilities you would like to determine given the distribution `d`.
`drop`	logical. Should the result be simplified to a vector if possible?
`elementwise`	logical. Should each distribution in `d` be evaluated at all elements of `x` (`elementwise = FALSE`, yielding a matrix)? Or, if `d` 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.
`...`	Arguments to be passed to `dnorm`. Unevaluated arguments will generate a warning to catch mispellings or other possible errors.

Value

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.

See Also

Other Normal distribution: `cdf.Normal()`, `fit_mle.Normal()`, `quantile.Normal()`

Examples

```
set.seed(27)

X <- Normal(5, 2)
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: calculating p-values for two-sided Z-test

# here the null hypothesis is H_0: mu = 3
# and we assume sigma = 2

# exactly the same as: Z <- Normal(0, 1)
Z <- Normal()

# data to test
x <- c(3, 7, 11, 0, 7, 0, 4, 5, 6, 2)
nx <- length(x)

# calculate the z-statistic
z_stat <- (mean(x) - 3) / (2 / sqrt(nx))
z_stat

# calculate the two-sided p-value
1 - cdf(Z, abs(z_stat)) + cdf(Z, -abs(z_stat))

# exactly equivalent to the above
2 * cdf(Z, -abs(z_stat))

# p-value for one-sided test
# H_0: mu <= 3   vs   H_A: mu > 3
1 - cdf(Z, z_stat)

# p-value for one-sided test
# H_0: mu >= 3   vs   H_A: mu < 3
cdf(Z, z_stat)

### example: calculating a 88 percent Z CI for a mean

# same `x` as before, still assume `sigma = 2`

# lower-bound
mean(x) - quantile(Z, 1 - 0.12 / 2) * 2 / sqrt(nx)

# upper-bound
mean(x) + quantile(Z, 1 - 0.12 / 2) * 2 / sqrt(nx)
```

```
# equivalent to
mean(x) + c(-1, 1) * quantile(Z, 1 - 0.12 / 2) * 2 / sqrt(nx)

# also equivalent to
mean(x) + quantile(Z, 0.12 / 2) * 2 / sqrt(nx)
mean(x) + quantile(Z, 1 - 0.12 / 2) * 2 / sqrt(nx)

### generating random samples and plugging in ks.test()

set.seed(27)

# generate a random sample
ns <- random(Normal(3, 7), 26)

# test if sample is Normal(3, 7)
ks.test(ns, pnorm, mean = 3, sd = 7)

# test if sample is gamma(8, 3) using base R pgamma()
ks.test(ns, pgamma, shape = 8, rate = 3)

### MISC

# note that the cdf() and quantile() functions are inverses
cdf(X, quantile(X, 0.7))
quantile(X, cdf(X, 7))
```

`pdf.Poisson` *Evaluate the probability mass function of a Poisson distribution*

Description

Evaluate the probability mass function of a Poisson distribution

Usage

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

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

Arguments

d	A Poisson object created by a call to Poisson().
x	A vector of elements whose probabilities you would like to determine given the distribution d.
drop	logical. Should the result be simplified to a vector if possible?

elementwise	logical. Should each distribution in d be evaluated at all elements of x (elementwise = FALSE, yielding a matrix)? Or, if d 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.
...	Arguments to be passed to dpois. Unevaluated arguments will generate a warning to catch mispellings or other possible errors.

Value

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.

Examples

```
set.seed(27)

X <- Poisson(2)
X

random(X, 10)

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

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

cdf(X, quantile(X, 0.7))
quantile(X, cdf(X, 7))
```

---

pdf.PoissonBinomial *Evaluate the probability mass function of a PoissonBinomial distribution*

Description

Evaluate the probability mass function of a PoissonBinomial distribution

Usage

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

## S3 method for class 'PoissonBinomial'
log_pdf(d, x, drop = TRUE, elementwise = NULL, ...)
```

Arguments

d	A PoissonBinomial object created by a call to PoissonBinomial().
x	A vector of elements whose probabilities you would like to determine given the distribution d.
drop	logical. Should the result be simplified to a vector if possible?
elementwise	logical. Should each distribution in d be evaluated at all elements of x (elementwise = FALSE, yielding a matrix)? Or, if d 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.
log, ...	Arguments to be passed to dpbinom or pnorm, respectively.
verbose	logical. Should a warning be issued if the normal approximation is applied because the **PoissonBinomial** package is not installed?

Value

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.

Examples

```
set.seed(27)

X <- PoissonBinomial(0.5, 0.3, 0.8)
X

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

random(X, 10)

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

cdf(X, 2)
quantile(X, 0.8)

cdf(X, quantile(X, 0.8))
quantile(X, cdf(X, 2))

## equivalent definitions of four Poisson binomial distributions
## each summing up three Bernoulli probabilities
p <- cbind(
  p1 = c(0.1, 0.2, 0.1, 0.2),
  p2 = c(0.5, 0.5, 0.5, 0.5),
  p3 = c(0.8, 0.7, 0.9, 0.8))
```

```
PoissonBinomial(p)
PoissonBinomial(p[, 1], p[, 2], p[, 3])
PoissonBinomial(p[, 1:2], p[, 3])
```

`pdf.RevWeibull` *Evaluate the probability mass function of an RevWeibull distribution*

Description

Evaluate the probability mass function of an RevWeibull distribution

Usage

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

## S3 method for class 'RevWeibull'
log_pdf(d, x, drop = TRUE, elementwise = NULL, ...)
```

Arguments

`d`	A `RevWeibull` object created by a call to `RevWeibull()`.
`x`	A vector of elements whose probabilities you would like to determine given the distribution `d`.
`drop`	logical. Should the result be simplified to a vector if possible?
`elementwise`	logical. Should each distribution in `d` be evaluated at all elements of `x` (`elementwise = FALSE`, yielding a matrix)? Or, if `d` 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.
`...`	Arguments to be passed to `dgev`. Unevaluated arguments will generate a warning to catch mispellings or other possible errors.

Value

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.

Examples

```
set.seed(27)

X <- RevWeibull(1, 2)
X

random(X, 10)
```

```
pdf(X, 0.7)
log_pdf(X, 0.7)

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

cdf(X, quantile(X, 0.7))
quantile(X, cdf(X, 0.7))
```

`pdf.SinhArcsinh` *Evaluate the probability mass function of a Sinh-Arcsinh distribution*

Description

Please see the documentation of `SinhArcsinh()` for some properties of the Sinh-Arcsinh distribution, as well as extensive examples showing to how calculate p-values and confidence intervals.

Usage

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

## S3 method for class 'SinhArcsinh'
log_pdf(d, x, drop = TRUE, elementwise = NULL, cores = NULL, ...)
```

Arguments

`d`	A `SinhArcsinh` object created by a call to `SinhArcsinh()`.
`x`	A vector of elements whose probabilities you would like to determine given the distribution `d`.
`drop`	logical. Should the result be simplified to a vector if possible?
`elementwise`	logical. Should each distribution in `d` be evaluated at all elements of `x` (`elementwise = FALSE`, yielding a matrix)? Or, if `d` 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.
`cores`	`NULL` or positive integer. TODO(R): Just a development option. If not `NULL` we use the C code with `cores` threads.
`...`	Arguments to be passed to `dnorm`. Unevaluated arguments will generate a warning to catch mispellings or other possible errors.

Value

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.

See Also

Other SinhArcsinh distribution: cdf.SinhArcsinh(), quantile.SinhArcsinh()

Examples

```
## SinhArcsinh() by default uses nu = 1, tau = 1 which
## results in the standard normal distribution
set.seed(6020)
X <- SinhArcsinh() # Uses mu = 1, sigma = 0, nu = 1, tau = 1)
x <- random(X, 300)
qqnorm(x); qqline(x, col = 2, lwd = 2)
curve(pdf(X, x), xlim = c(-5, 5), main = paste(X, "density"))

## Calculation of central moments is based on numeric integration,
## thus not being identical to the standard normal distribution
c(mean = mean(x), sd = sd(x))

## Skewed Sinh-Arcsinh distribution
X <- SinhArcsinh(mu = 7, sigma = 2, nu = c(0.7, 1, 0.7), tau = c(1, 0.7, 0.7))
as.matrix(X)

## Visualization of density functions using different parameters for nu/tau
curve(pdf(X[1], x), xlim = c(0, 20), ylim = c(0, 0.2), main = "Density function")
curve(pdf(X[2], x), xlim = c(0, 20), col = 2, add = TRUE)
curve(pdf(X[3], x), xlim = c(0, 20), col = 4, add = TRUE)

## Visualization of distribution function using different parameters for nu/tau
curve(cdf(X[1], x), xlim = c(0, 20), ylim = 0:1, main = "Distribution function")
curve(cdf(X[2], x), xlim = c(0, 20), col = 2, add = TRUE)
curve(cdf(X[3], x), xlim = c(0, 20), col = 4, add = TRUE)

## Central moments
mean(X)
variance(X)
skewness(X)
kurtosis(X)

## Drawing random values
random(X, 10)

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

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

# note that the cdf() and quantile() functions are inverses
X <- SinhArcsinh(mu = 3, sigma = 2, nu = 0.9, tau = 1.2)
cdf(X, quantile(X, 0.7))
quantile(X, cdf(X, 7))
```

`pdf.StudentsT` *Evaluate the probability mass function of a StudentsT distribution*

Description

Please see the documentation of `StudentsT()` for some properties of the StudentsT distribution, as well as extensive examples showing to how calculate p-values and confidence intervals.

Usage

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

## S3 method for class 'StudentsT'
log_pdf(d, x, drop = TRUE, elementwise = NULL, ...)
```

Arguments

`d`	A `StudentsT` object created by a call to `StudentsT()`.
`x`	A vector of elements whose probabilities you would like to determine given the distribution `d`.
`drop`	logical. Should the result be simplified to a vector if possible?
`elementwise`	logical. Should each distribution in `d` be evaluated at all elements of `x` (`elementwise = FALSE`, yielding a matrix)? Or, if `d` 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.
`...`	Arguments to be passed to `dt`. Unevaluated arguments will generate a warning to catch mispellings or other possible errors.

Value

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.

See Also

Other StudentsT distribution: `cdf.StudentsT()`, `quantile.StudentsT()`, `random.StudentsT()`

Examples

```
set.seed(27)

X <- StudentsT(3)
X
```

```
random(X, 10)

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

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

### example: calculating p-values for two-sided T-test

# here the null hypothesis is H_0: mu = 3

# data to test
x <- c(3, 7, 11, 0, 7, 0, 4, 5, 6, 2)
nx <- length(x)

# calculate the T-statistic
t_stat <- (mean(x) - 3) / (sd(x) / sqrt(nx))
t_stat

# null distribution of statistic depends on sample size!
T <- StudentsT(df = nx - 1)

# calculate the two-sided p-value
1 - cdf(T, abs(t_stat)) + cdf(T, -abs(t_stat))

# exactly equivalent to the above
2 * cdf(T, -abs(t_stat))

# p-value for one-sided test
# H_0: mu <= 3   vs   H_A: mu > 3
1 - cdf(T, t_stat)

# p-value for one-sided test
# H_0: mu >= 3   vs   H_A: mu < 3
cdf(T, t_stat)

### example: calculating a 88 percent T CI for a mean

# lower-bound
mean(x) - quantile(T, 1 - 0.12 / 2) * sd(x) / sqrt(nx)

# upper-bound
mean(x) + quantile(T, 1 - 0.12 / 2) * sd(x) / sqrt(nx)

# equivalent to
mean(x) + c(-1, 1) * quantile(T, 1 - 0.12 / 2) * sd(x) / sqrt(nx)

# also equivalent to
mean(x) + quantile(T, 0.12 / 2) * sd(x) / sqrt(nx)
mean(x) + quantile(T, 1 - 0.12 / 2) * sd(x) / sqrt(nx)
```

`pdf.Uniform` *Evaluate the probability mass function of a continuous Uniform distribution*

Description

Evaluate the probability mass function of a continuous Uniform distribution

Usage

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

## S3 method for class 'Uniform'
log_pdf(d, x, drop = TRUE, elementwise = NULL, ...)
```

Arguments

`d`	A `Uniform` object created by a call to `Uniform()`.
`x`	A vector of elements whose probabilities you would like to determine given the distribution `d`.
`drop`	logical. Should the result be simplified to a vector if possible?
`elementwise`	logical. Should each distribution in `d` be evaluated at all elements of `x` (`elementwise = FALSE`, yielding a matrix)? Or, if `d` 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.
`...`	Arguments to be passed to `dunif`. Unevaluated arguments will generate a warning to catch mispellings or other possible errors.

Value

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.

Examples

```
set.seed(27)

X <- Uniform(1, 2)
X

random(X, 10)

pdf(X, 0.7)
log_pdf(X, 0.7)
```

```
cdf(X, 0.7)
quantile(X, 0.7)

cdf(X, quantile(X, 0.7))
quantile(X, cdf(X, 0.7))
```

`pdf.Weibull` *Evaluate the probability mass function of a Weibull distribution*

Description

Please see the documentation of `Weibull()` for some properties of the Weibull distribution, as well as extensive examples showing to how calculate p-values and confidence intervals.

Usage

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

## S3 method for class 'Weibull'
log_pdf(d, x, drop = TRUE, elementwise = NULL, ...)
```

Arguments

`d`	A `Weibull` object created by a call to `Weibull()`.
`x`	A vector of elements whose probabilities you would like to determine given the distribution `d`.
`drop`	logical. Should the result be simplified to a vector if possible?
`elementwise`	logical. Should each distribution in `d` be evaluated at all elements of `x` (`elementwise = FALSE`, yielding a matrix)? Or, if `d` 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.
`...`	Arguments to be passed to `dweibull`. Unevaluated arguments will generate a warning to catch mispellings or other possible errors.

Value

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.

See Also

Other Weibull distribution: `cdf.Weibull()`, `quantile.Weibull()`, `random.Weibull()`

Examples

```
set.seed(27)

X <- Weibull(0.3, 2)
X

random(X, 10)

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

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

`pdf.ZINegativeBinomial` *Evaluate the probability mass function of a zero-inflated negative binomial distribution*

Description

Evaluate the probability mass function of a zero-inflated negative binomial distribution

Usage

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

## S3 method for class 'ZINegativeBinomial'
log_pdf(d, x, drop = TRUE, elementwise = NULL, ...)
```

Arguments

`d`	A `ZINegativeBinomial` object created by a call to `ZINegativeBinomial()`.
`x`	A vector of elements whose probabilities you would like to determine given the distribution `d`.
`drop`	logical. Should the result be simplified to a vector if possible?
`elementwise`	logical. Should each distribution in `d` be evaluated at all elements of `x` (`elementwise = FALSE`, yielding a matrix)? Or, if `d` 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.
`...`	Arguments to be passed to `dzinbinom`. Unevaluated arguments will generate a warning to catch mispellings or other possible errors.

Value

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.

Examples

```
## set up a zero-inflated negative binomial distribution
X <- ZINegativeBinomial(mu = 2.5, theta = 1, pi = 0.25)
X

## standard functions
pdf(X, 0:8)
cdf(X, 0:8)
quantile(X, seq(0, 1, by = 0.25))

## cdf() and quantile() are inverses for each other
quantile(X, cdf(X, 3))

## density visualization
plot(0:8, pdf(X, 0:8), type = "h", lwd = 2)

## corresponding sample with histogram of empirical frequencies
set.seed(0)
x <- random(X, 500)
hist(x, breaks = -1:max(x) + 0.5)
```

---

pdf.ZIPoisson *Evaluate the probability mass function of a zero-inflated Poisson distribution*

Description

Evaluate the probability mass function of a zero-inflated Poisson distribution

Usage

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

## S3 method for class 'ZIPoisson'
log_pdf(d, x, drop = TRUE, elementwise = NULL, ...)
```

Arguments

d	A ZIPoisson object created by a call to ZIPoisson().
x	A vector of elements whose probabilities you would like to determine given the distribution d.

drop	logical. Should the result be simplified to a vector if possible?
elementwise	logical. Should each distribution in d be evaluated at all elements of x (elementwise = FALSE, yielding a matrix)? Or, if d 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.
...	Arguments to be passed to dzipois. Unevaluated arguments will generate a warning to catch mispellings or other possible errors.

Value

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.

Examples

```
## set up a zero-inflated Poisson distribution
X <- ZIPoisson(lambda = 2.5, pi = 0.25)
X

## standard functions
pdf(X, 0:8)
cdf(X, 0:8)
quantile(X, seq(0, 1, by = 0.25))

## cdf() and quantile() are inverses for each other
quantile(X, cdf(X, 3))

## density visualization
plot(0:8, pdf(X, 0:8), type = "h", lwd = 2)

## corresponding sample with histogram of empirical frequencies
set.seed(0)
x <- random(X, 500)
hist(x, breaks = -1:max(x) + 0.5)
```

---

pdf.ZTNegativeBinomial

Evaluate the probability mass function of a zero-truncated negative binomial distribution

---

Description

Evaluate the probability mass function of a zero-truncated negative binomial distribution

Usage

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

## S3 method for class 'ZTNegativeBinomial'
log_pdf(d, x, drop = TRUE, elementwise = NULL, ...)
```

Arguments

d	A ZTNegativeBinomial object created by a call to ZTNegativeBinomial().
x	A vector of elements whose probabilities you would like to determine given the distribution d.
drop	logical. Should the result be simplified to a vector if possible?
elementwise	logical. Should each distribution in d be evaluated at all elements of x (elementwise = FALSE, yielding a matrix)? Or, if d 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.
...	Arguments to be passed to dztnbinom. Unevaluated arguments will generate a warning to catch mispellings or other possible errors.

Value

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.

Examples

```
## set up a zero-truncated negative binomial distribution
X <- ZTNegativeBinomial(mu = 2.5, theta = 1)
X

## standard functions
pdf(X, 0:8)
cdf(X, 0:8)
quantile(X, seq(0, 1, by = 0.25))

## cdf() and quantile() are inverses for each other
quantile(X, cdf(X, 3))

## density visualization
plot(0:8, pdf(X, 0:8), type = "h", lwd = 2)

## corresponding sample with histogram of empirical frequencies
set.seed(0)
x <- random(X, 500)
hist(x, breaks = -1:max(x) + 0.5)
```

pdf.ZTPoisson *Evaluate the probability mass function of a zero-truncated Poisson distribution*

Description

Evaluate the probability mass function of a zero-truncated Poisson distribution

Usage

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

## S3 method for class 'ZTPoisson'
log_pdf(d, x, drop = TRUE, elementwise = NULL, ...)
```

Arguments

d	A ZTPoisson object created by a call to ZTPoisson().
x	A vector of elements whose probabilities you would like to determine given the distribution d.
drop	logical. Should the result be simplified to a vector if possible?
elementwise	logical. Should each distribution in d be evaluated at all elements of x (elementwise = FALSE, yielding a matrix)? Or, if d 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.
...	Arguments to be passed to dztpois. Unevaluated arguments will generate a warning to catch mispellings or other possible errors.

Value

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.

Examples

```
## set up a zero-truncated Poisson distribution
X <- ZTPoisson(lambda = 2.5)
X

## standard functions
pdf(X, 0:8)
cdf(X, 0:8)
quantile(X, seq(0, 1, by = 0.25))
```

```
## cdf() and quantile() are inverses for each other
quantile(X, cdf(X, 3))

## density visualization
plot(0:8, pdf(X, 0:8), type = "h", lwd = 2)

## corresponding sample with histogram of empirical frequencies
set.seed(0)
x <- random(X, 500)
hist(x, breaks = -1:max(x) + 0.5)
```

`plot.distribution` *Plot the p.m.f, p.d.f or c.d.f. of a univariate distribution*

Description

Plot method for an object inheriting from class `"distribution"`. By default the probability density function (p.d.f.), for a continuous variable, or the probability mass function (p.m.f.), for a discrete variable, is plotted. The cumulative distribution function (c.d.f.) will be plotted if `cdf = TRUE`. Multiple functions are included in the plot if any of the parameter vectors in `x` has length greater than 1. See the argument `all`.

Usage

```
## S3 method for class 'distribution'
plot(
  x,
  cdf = FALSE,
  p = c(0.1, 99.9),
  len = 1000,
  all = FALSE,
  legend_args = list(),
  ...
)
```

Arguments

`x`	an object of class `c("name", "distribution")`, where `"name"` is the name of the distribution.
`cdf`	A logical scalar. If `cdf = TRUE` then the cumulative distribution function (c.d.f.) is plotted. Otherwise, the probability density function (p.d.f.), for a continuous variable, or the probability mass function (p.m.f.), for a discrete variable, is plotted.
`p`	A numeric vector. If `xlim` is not passed in `...` then `p` is the fallback option for setting the range of values over which the p.m.f, p.d.f. or c.d.f is plotted. See **Details**.

len	An integer scalar. If x is a continuous distribution object then len is the number of values at which the p.d.f or c.d.f. is evaluated to produce the plot. The larger len is the smoother is the curve.
all	A logical scalar. If all = TRUE then a separate distribution is plotted for all the combinations of parameter values present in the parameter vectors present in x. These combinations are generated using expand.grid. If all = FALSE then the number of distributions plotted is equal to the maximum of the lengths of these parameter vectors, with shorter vectors recycled to this length if necessary using rep_len.
legend_args	A list of arguments to be passed to legend. In particular, the argument x (perhaps in conjunction with legend_args$y) can be used to set the position of the legend. If legend_args$x is not supplied then "bottomright" is used if cdf = TRUE and "topright" if cdf = FALSE.
...	Further arguments to be passed to plot, plot.ecdf and lines, such as xlim, ylim, xlab, ylab, main, lwd, lty, col, pch.

Details

If xlim is passed in ... then this determines the range of values of the variable to be plotted on the horizontal axis. If x is a discrete distribution object then the values for which the p.m.f. or c.d.f. is plotted is the smallest set of consecutive integers that contains both components of xlim. Otherwise, xlim is used directly.

If xlim is not passed in ... then the range of values spans the support of the distribution, with the following proviso: if the lower (upper) endpoint of the distribution is -Inf (Inf) then the lower (upper) limit of the plotting range is set to the p[1]\

If the name of x is a single upper case letter then that name is used to labels the axes of the plot. Otherwise, x and P(X = x) or f(x) are used.

A legend is included only if at least one of the parameter vectors in x has length greater than 1.

Plots of c.d.f.s are produced using calls to approxfun and plot.ecdf.

Value

An object with the same class as x, in which the parameter vectors have been expanded to contain a parameter combination for each function plotted.

Examples

```
B <- Binomial(20, 0.7)
plot(B)
plot(B, cdf = TRUE)

B2 <- Binomial(20, c(0.1, 0.5, 0.9))
plot(B2, legend_args = list(x = "top"))
x <- plot(B2, cdf = TRUE)
x$size
x$p

X <- Poisson(2)
```

```
plot(X)
plot(X, cdf = TRUE)

G <- Gamma(c(1, 3), 1:2)
plot(G)
plot(G, all = TRUE)
plot(G, cdf = TRUE)

C <- Cauchy()
plot(C, p = c(1, 99), len = 10000)
plot(C, cdf = TRUE, p = c(1, 99))
```

`plot_cdf` *Plot the CDF of a distribution*

Description

A function to easily plot the CDF of a distribution using `ggplot2`. Requires `ggplot2` to be loaded.

Usage

```
plot_cdf(d, limits = NULL, p = 0.001, plot_theme = NULL)
```

Arguments

`d`	A `distribution` object
`limits`	either `NULL` (default) or a vector of length 2 that specifies the range of the x-axis
`p`	If `limits` is `NULL`, the range of the x-axis will be the support of `d` if this is a bounded interval, or `quantile(d, p)` and `quantile(d, 1 - p)` if lower and/or upper limits of the support is `-Inf/Inf`. Defaults to 0.001.
`plot_theme`	specify theme of resulting plot using `ggplot2`. Default is `theme_minimal`

Examples

```
N1 <- Normal()
plot_cdf(N1)

N2 <- Normal(0, c(1, 2))
plot_cdf(N2)

B1 <- Binomial(10, 0.2)
plot_cdf(B1)

B2 <- Binomial(10, c(0.2, 0.5))
plot_cdf(B2)
```

plot_pdf *Plot the PDF of a distribution*

Description

A function to easily plot the PDF of a distribution using ggplot2. Requires ggplot2 to be loaded.

Usage

```
plot_pdf(d, limits = NULL, p = 0.001, plot_theme = NULL)
```

Arguments

d	A distribution object
limits	either NULL (default) or a vector of length 2 that specifies the range of the x-axis
p	If limits is NULL, the range of the x-axis will be the support of d if this is a bounded interval, or quantile(d, p) and quantile(d, 1 - p) if lower and/or upper limits of the support is -Inf/Inf. Defaults to 0.001.
plot_theme	specify theme of resulting plot using ggplot2. Default is theme_minimal

Examples

```
N1 <- Normal()
plot_pdf(N1)

N2 <- Normal(0, c(1, 2))
plot_pdf(N2)

B1 <- Binomial(10, 0.2)
plot_pdf(B1)

B2 <- Binomial(10, c(0.2, 0.5))
plot_pdf(B2)
```

Poisson *Create a Poisson distribution*

Description

Poisson distributions are frequently used to model counts.

Usage

```
Poisson(lambda = numeric())
```

Arguments

lambda	The shape parameter, which is also the mean and the variance of the distribution. Can be any positive number.

Details

We recommend reading this documentation on https://zeileis.github.io/distributions3/, where the math will render with additional detail.

In the following, let X be a Poisson random variable with parameter lambda = λ.

Support: $\{0, 1, 2, 3, ...\}$

Mean: λ

Variance: λ

Probability mass function (p.m.f):

$$P(X = k) = \frac{\lambda^k e^{-\lambda}}{k!}$$

Cumulative distribution function (c.d.f):

$$P(X \leq k) = e^{-\lambda} \sum_{i=0}^{\lfloor k \rfloor} \frac{\lambda^i}{i!}$$

Moment generating function (m.g.f):

$$E(e^{tX}) = e^{\lambda(e^t - 1)}$$

Value

A Poisson object.

See Also

Other discrete distributions: Bernoulli(), Binomial(), Categorical(), Geometric(), HurdleNegativeBinomial(), HurdlePoisson(), HyperGeometric(), Multinomial(), NegativeBinomial(), PoissonBinomial(), ZINegativeBinomial(), ZIPoisson(), ZTNegativeBinomial(), ZTPoisson()

Examples

```
set.seed(27)

X <- Poisson(2)
X

random(X, 10)

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

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

cdf(X, quantile(X, 0.7))
quantile(X, cdf(X, 7))
```

`PoissonBinomial` *Create a Poisson binomial distribution*

Description

The Poisson binomial distribution is a generalization of the `Binomial` distribution. It is also a sum of n independent Bernoulli experiments. However, the success probabilities can vary between the experiments so that they are not identically distributed.

Usage

```
PoissonBinomial(...)
```

Arguments

`...`	An arbitrary number of numeric vectors or matrices of success probabilities in `[0, 1]` (with matching number of rows).

Details

The Poisson binomial distribution comes up when you consider the number of successes in independent binomial experiments (coin flips) with potentially varying success probabilities.

The `PoissonBinomial` distribution class in **distributions3** is mostly based on the **PoissonBinomial** package, providing fast **Rcpp** implementations of efficient algorithms. Hence, it is recommended to install the **PoissonBinomial** package when working with this distribution. However, as a fallback for when the **PoissonBinomial** package is not installed the methods for the `PoissonBinomial` distribution employ a normal approximation.

We recommend reading the following documentation on https://zeileis.github.io/distributions3/, where the math will render with additional detail.

In the following, let X be a Poisson binomial random variable with success probabilities p_1 to p_n.

Support: $\{0, 1, 2, ..., n\}$

Mean: $p_1 + \cdots + p_n$

Variance: $p_1 \cdot (1 - p_1) + \cdots + p_1 \cdot (1 - p_1)$

Probability mass function (p.m.f):

$$P(X = k) = \sum_{A} \prod_{i \in A} p_i \prod_{j \in A^C} (1 - p_j)$$

where the sum is taken over all sets A with k elements from $\{0, 1, 2, ..., n\}$. A^C is the complement of A.

Cumulative distribution function (c.d.f):

$$P(X \leq k) = \sum_{i=0}^{\lfloor k \rfloor} P(X = i)$$

Moment generating function (m.g.f):

$$E(e^{tX}) = \prod_{i=1}^{n} (1 - p_i + p_i e^t)$$

Value

A `PoissonBinomial` object.

See Also

Other discrete distributions: `Bernoulli()`, `Binomial()`, `Categorical()`, `Geometric()`, `HurdleNegativeBinomial()`, `HurdlePoisson()`, `HyperGeometric()`, `Multinomial()`, `NegativeBinomial()`, `Poisson()`, `ZINegativeBinomial()`, `ZIPoisson()`, `ZTNegativeBinomial()`, `ZTPoisson()`

Examples

```
set.seed(27)

X <- PoissonBinomial(0.5, 0.3, 0.8)
X

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

random(X, 10)

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

cdf(X, 2)
quantile(X, 0.8)

cdf(X, quantile(X, 0.8))
quantile(X, cdf(X, 2))

## equivalent definitions of four Poisson binomial distributions
## each summing up three Bernoulli probabilities
p <- cbind(
  p1 = c(0.1, 0.2, 0.1, 0.2),
  p2 = c(0.5, 0.5, 0.5, 0.5),
```

```
  p3 = c(0.8, 0.7, 0.9, 0.8))
PoissonBinomial(p)
PoissonBinomial(p[, 1], p[, 2], p[, 3])
PoissonBinomial(p[, 1:2], p[, 3])
```

`prodist` *Extracting fitted or predicted probability distributions from models*

Description

Generic function with methods for various model classes for extracting fitted (in-sample) or predicted (out-of-sample) probability `distributions3` objects.

Usage

```
prodist(object, ...)

## S3 method for class 'lm'
prodist(object, ..., sigma = "ML")

## S3 method for class 'glm'
prodist(object, ..., dispersion = NULL)

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

Arguments

`object`	A model object.
`...`	Arguments passed on to methods, typically for calling the underlying `predict` methods, e.g., `newdata` for `lm` or `glm` objects or `n.ahead` for `arima` objects.
`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`.

Details

To facilitate making probabilistic forecasts based on regression and time series model objects, the function `prodist` extracts fitted or predicted probability `distribution` objects. Currently, methods are provided for objects fitted by `lm`, `glm`, and `arima` in base R as well as `glm.nb` from the **MASS** package and `hurdle`/`zeroinfl`/`zerotrunc` from the **pscl** or **countreg** packages.

All methods essentially proceed in two steps: First, the standard `predict` method for these model objects is used to compute fitted (in-sample, default) or predicted (out-of-sample) distribution parameters. Typically, this includes the mean plus further parameters describing scale, dispersion, shape, etc.). Second, the `distributions` objects are set up using the generator functions from **distributions3**.

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

For both linear regression models and generalized linear models, estimated by `lm` and `glm` respectively, there is some ambiguity as to which estimate for the dispersion parameter of the model is to be used. While the `logLik` methods use the maximum-likelihood (ML) estimate implicitly, the `summary` methods report an estimate that is standardized with the residual degrees of freedom, n - k (rather than the number of observations, n). The `prodist` methods for these objects follow the `logLik` method by default but the `summary` behavior can be mimicked by setting the `sigma` or `dispersion` arguments accordingly.

Value

An object inheriting from `distribution`.

See Also

`predict`, `lm`, `glm`, `arima`

Examples

```
## Model: Linear regression
## Fit: lm
## Data: 1920s cars data
data("cars", package = "datasets")

## Stopping distance (ft) explained by speed (mph)
reg <- lm(dist ~ speed, data = cars)

## Extract fitted normal distributions (in-sample, with constant variance)
pd <- prodist(reg)
head(pd)

## Extract log-likelihood from model object
logLik(reg)

## Replicate log-likelihood via distributions object
sum(log_pdf(pd, cars$dist))
log_likelihood(pd, cars$dist)
```

```
## Compute corresponding medians and 90% interval
qd <- quantile(pd, c(0.05, 0.5, 0.95))
head(qd)

## Visualize observations with predicted quantiles
plot(dist ~ speed, data = cars)
matplot(cars$speed, qd, add = TRUE, type = "l", col = 2, lty = 1)

## Sigma estimated by maximum-likelihood estimate (default, used in logLik)
## vs. least-squares estimate (used in summary)
nd <- data.frame(speed = 50)
prodist(reg, newdata = nd, sigma = "ML")
prodist(reg, newdata = nd, sigma = "OLS")
summary(reg)$sigma


## Model: Poisson generalized linear model
## Fit: glm
## Data: FIFA 2018 World Cup data
data("FIFA2018", package = "distributions3")

## Number of goals per team explained by ability differences
poisreg <- glm(goals ~ difference, data = FIFA2018, family = poisson)
summary(poisreg)
## Interpretation: When the ratio of abilities increases by 1 percent,
## the expected number of goals increases by around 0.4 percent

## Predict fitted Poisson distributions for teams with equal ability (out-of-sample)
nd <- data.frame(difference = 0)
prodist(poisreg, newdata = nd)

## Extract fitted Poisson distributions (in-sample)
pd <- prodist(poisreg)
head(pd)

## Extract log-likelihood from model object
logLik(poisreg)

## Replicate log-likelihood via distributions object
sum(log_pdf(pd, FIFA2018$goals))
log_likelihood(pd, FIFA2018$goals)


## Model: Autoregressive integrated moving average model
## Fit: arima
## Data: Quarterly approval ratings of U.S. presidents (1945-1974)
data("presidents", package = "datasets")

## ARMA(2,1) model
arma21 <- arima(presidents, order = c(2, 0, 1))

## Extract predicted normal distributions for next two years
```

```
p <- prodist(arma21, n.ahead = 8)
p

## Compute median (= mean) forecast along with 80% and 95% interval
quantile(p, c(0.5, 0.1, 0.9, 0.025, 0.975))
```

---

quantile.Bernoulli *Determine quantiles of a Bernoulli distribution*

Description

quantile() is the inverse of cdf().

Usage

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

Arguments

x	A Bernoulli object created by a call to Bernoulli().
probs	A vector of probabilities.
drop	logical. Should the result be simplified to a vector if possible?
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.
...	Arguments to be passed to qbinom. Unevaluated arguments will generate a warning to catch mispellings or other possible errors.

Value

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.

Examples

```
set.seed(27)

X <- Bernoulli(0.7)
X

mean(X)
variance(X)
```

```
skewness(X)
kurtosis(X)

random(X, 10)
pdf(X, 1)
log_pdf(X, 1)
cdf(X, 0)
quantile(X, 0.7)

cdf(X, quantile(X, 0.7))
quantile(X, cdf(X, 0.7))
```

`quantile.Beta` *Determine quantiles of a Beta distribution*

Description

`quantile()` is the inverse of `cdf()`.

Usage

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

Arguments

`x`	A `Beta` object created by a call to `Beta()`.
`probs`	A vector of probabilities.
`drop`	logical. Should the result be simplified to a vector if possible?
`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.
`...`	Arguments to be passed to `qbeta`. Unevaluated arguments will generate a warning to catch mispellings or other possible errors.

Value

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.

Examples

```
set.seed(27)

X <- Beta(1, 2)
X

random(X, 10)

pdf(X, 0.7)
log_pdf(X, 0.7)

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

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

cdf(X, quantile(X, 0.7))
quantile(X, cdf(X, 0.7))
```

---

quantile.Binomial *Determine quantiles of a Binomial distribution*

Description

`quantile()` is the inverse of `cdf()`.

Usage

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

Arguments

`x`	A `Binomial` object created by a call to `Binomial()`.
`probs`	A vector of probabilities.
`drop`	logical. Shoul the result be simplified to a vector if possible?
`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.
`...`	Arguments to be passed to `qbinom`. Unevaluated arguments will generate a warning to catch mispellings or other possible errors.

Value

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.

Examples

```
set.seed(27)

X <- Binomial(10, 0.2)
X

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

random(X, 10)

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

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

cdf(X, quantile(X, 0.7))
quantile(X, cdf(X, 7))
```

---

quantile.Categorical *Determine quantiles of a Categorical discrete distribution*

Description

`quantile()` is the inverse of `cdf()`.

Usage

```
## S3 method for class 'Categorical'
quantile(x, probs, ...)
```

Arguments

`x`	A `Categorical` object created by a call to `Categorical()`.
`probs`	A vector of probabilities.
`...`	Unused. Unevaluated arguments will generate a warning to catch mispellings or other possible errors.

Value

A vector of quantiles, one for each element of probs.

Examples

```
set.seed(27)

X <- Categorical(1:3, p = c(0.4, 0.1, 0.5))
X

Y <- Categorical(LETTERS[1:4])
Y

random(X, 10)
random(Y, 10)

pdf(X, 1)
log_pdf(X, 1)

cdf(X, 1)
quantile(X, 0.5)

# cdfs are only defined for numeric sample spaces. this errors!
# cdf(Y, "a")

# same for quantiles. this also errors!
# quantile(Y, 0.7)
```

---

quantile.Cauchy *Determine quantiles of a Cauchy distribution*

Description

quantile() is the inverse of cdf().

Usage

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

Arguments

x	A Cauchy object created by a call to Cauchy().
probs	A vector of probabilities.
drop	logical. Should the result be simplified to a vector if possible?

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.
...	Arguments to be passed to qcauchy. Unevaluated arguments will generate a warning to catch mispellings or other possible errors.

Value

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.

Examples

```
set.seed(27)

X <- Cauchy(10, 0.2)
X

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

random(X, 10)

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

cdf(X, 2)
quantile(X, 0.7)

cdf(X, quantile(X, 0.7))
quantile(X, cdf(X, 7))
```

---

quantile.ChiSquare *Determine quantiles of a chi square distribution*

Description

quantile() is the inverse of cdf().

Usage

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

Arguments

x	A ChiSquare object created by a call to ChiSquare().
probs	A vector of probabilities.
drop	logical. Should the result be simplified to a vector if possible?
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.
...	Arguments to be passed to qchisq. Unevaluated arguments will generate a warning to catch mispellings or other possible errors.

Value

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.

Examples

```
set.seed(27)

X <- ChiSquare(5)
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)

cdf(X, quantile(X, 0.7))
quantile(X, cdf(X, 7))
```

---

quantile.Empirical *Determine quantiles of an Empirical distribution*

Description

Please see the documentation of Empirical() for some properties of the Empirical distribution.

Usage

```
## S3 method for class 'Empirical'
quantile(x, probs, drop = TRUE, elementwise = NULL, type = 1L, ...)
```

Arguments

x	an object of class Empirical.
probs	A vector of probabilities.
drop	logical. Should the result be simplified to a vector if possible?
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.
type	integer, forwarded to quantile. Defaults to type = 1L.
...	arguments to be passed to qempirical().

Value

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.

See Also

Other Empirical distribution: Empirical(), cdf.Empirical(), dempirical(), pdf.Empirical(), random.Empirical(), support.Empirical()

Examples

```
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
Y4 <- matrix(rnorm(20), ncol = 5,
             dimnames = list(paste0("D_", 1:4), paste0("obs_", 1:5)))
d4 <- Empirical(Y4)
d4

## Data frame
d5 <- Empirical(as.data.frame(Y4))
d5

identical(d4, d5)

mean(d5)
variance(d5)
skewness(d5)
kurtosis(d5)

pdf(d5, c(-0.5, 0, 0.5, 1)) # Defaults to elementwise = TRUE
pdf(d5, c(-0.5, 0, 0.5, 1), elementwise = FALSE)

cdf(d5, c(-0.5, 0, 0.5, 1)) # Defaults to elementwise = TRUE
cdf(d5, c(-0.5, 0, 0.5, 1), elementwise = FALSE)

quantile(d5, c(0.2, 0.4, 0.6, 0.8)) # Defaults to elementwise = TRUE
quantile(d5, c(0.2, 0.4, 0.6, 0.8), elementwise = FALSE)

## The quantile function is the inverse of the distribution
## function (cdf) if x in Y
set.seed(6020)
Y <- round(rlnorm(20, log(3), log(2)), 1)
```

```
d <- Empirical(Y)

cdf(d, 4.0)
quantile(d, cdf(d, 4.0))
```

`quantile.Erlang` *Determine quantiles of an Erlang distribution*

Description

`quantile()` is the inverse of `cdf()`.

Usage

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

Arguments

`x`	An `Erlang` object created by a call to `Erlang()`.
`probs`	A vector of probabilities.
`drop`	logical. Should the result be simplified to a vector if possible?
`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.
`...`	Arguments to be passed to `qgamma`. Unevaluated arguments will generate a warning to catch mispellings or other possible errors.

Value

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.

Examples

```
set.seed(27)

X <- Erlang(5, 2)
X

random(X, 10)

pdf(X, 2)
```

```
log_pdf(X, 2)

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

cdf(X, quantile(X, 0.7))
quantile(X, cdf(X, 7))
```

`quantile.Exponential` *Determine quantiles of an Exponential distribution*

Description

`quantile()` is the inverse of `cdf()`.

Usage

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

Arguments

`x`	An `Exponential` object created by a call to `Exponential()`.
`probs`	A vector of probabilities.
`drop`	logical. Should the result be simplified to a vector if possible?
`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.
`...`	Arguments to be passed to `qexp`. Unevaluated arguments will generate a warning to catch mispellings or other possible errors.

Value

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.

Examples

```
set.seed(27)

X <- Exponential(5)
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)

cdf(X, quantile(X, 0.7))
quantile(X, cdf(X, 7))
```

`quantile.FisherF` *Determine quantiles of an F distribution*

Description

`quantile()` is the inverse of `cdf()`.

Usage

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

Arguments

`x`	A `FisherF` object created by a call to `FisherF()`.
`probs`	A vector of probabilities.
`drop`	logical. Should the result be simplified to a vector if possible?
`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.
`...`	Arguments to be passed to `qf`. Unevaluated arguments will generate a warning to catch mispellings or other possible errors.

Value

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.

Examples

```
set.seed(27)

X <- FisherF(5, 10, 0.2)
X

random(X, 10)

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

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

cdf(X, quantile(X, 0.7))
quantile(X, cdf(X, 7))
```

`quantile.Frechet` *Determine quantiles of a Frechet distribution*

Description

`quantile()` is the inverse of `cdf()`.

Usage

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

Arguments

`x`	A `Frechet` object created by a call to `Frechet()`.
`probs`	A vector of probabilities.
`drop`	logical. Should the result be simplified to a vector if possible?
`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.
`...`	Arguments to be passed to `qgev`. Unevaluated arguments will generate a warning to catch mispellings or other possible errors.

Value

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.

Examples

```
set.seed(27)

X <- Frechet(0, 2)
X

random(X, 10)

pdf(X, 0.7)
log_pdf(X, 0.7)

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

cdf(X, quantile(X, 0.7))
quantile(X, cdf(X, 0.7))
```

`quantile.Gamma` *Determine quantiles of a Gamma distribution*

Description

`quantile()` is the inverse of `cdf()`.

Usage

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

Arguments

`x`	A `Gamma` object created by a call to `Gamma()`.
`probs`	A vector of probabilities.
`drop`	logical. Should the result be simplified to a vector if possible?
`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.
`...`	Arguments to be passed to `qgamma`. Unevaluated arguments will generate a warning to catch mispellings or other possible errors.

Value

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.

Examples

```
set.seed(27)

X <- Gamma(5, 2)
X

random(X, 10)

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

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

cdf(X, quantile(X, 0.7))
quantile(X, cdf(X, 7))
```

`quantile.Geometric` *Determine quantiles of a Geometric distribution*

Description

Determine quantiles of a Geometric distribution

Usage

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

Arguments

`x`	A `Geometric` object created by a call to `Geometric()`.
`probs`	A vector of probabilities.
`drop`	logical. Should the result be simplified to a vector if possible?
`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.
`...`	Arguments to be passed to `qgeom`. Unevaluated arguments will generate a warning to catch mispellings or other possible errors.

Value

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.

See Also

Other Geometric distribution: cdf.Geometric(), pdf.Geometric(), random.Geometric()

Examples

```
set.seed(27)

X <- Geometric(0.3)
X

random(X, 10)

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

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

---

quantile.GEV *Determine quantiles of a GEV distribution*

Description

quantile() is the inverse of cdf().

Usage

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

Arguments

x	A GEV object created by a call to GEV().
probs	A vector of probabilities.
drop	logical. Should the result be simplified to a vector if possible?
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.
...	Arguments to be passed to qgev. Unevaluated arguments will generate a warning to catch mispellings or other possible errors.

Value

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.

Examples

```
set.seed(27)

X <- GEV(1, 2, 0.1)
X

random(X, 10)

pdf(X, 0.7)
log_pdf(X, 0.7)

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

cdf(X, quantile(X, 0.7))
quantile(X, cdf(X, 0.7))
```

`quantile.GP` *Determine quantiles of a GP distribution*

Description

`quantile()` is the inverse of `cdf()`.

Usage

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

Arguments

`x`	A `GP` object created by a call to `GP()`.
`probs`	A vector of probabilities.
`drop`	logical. Should the result be simplified to a vector if possible?
`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.
`...`	Arguments to be passed to `qgp`. Unevaluated arguments will generate a warning to catch mispellings or other possible errors.

Value

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.

Examples

```
set.seed(27)

X <- GP(0, 2, 0.1)
X

random(X, 10)

pdf(X, 0.7)
log_pdf(X, 0.7)

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

cdf(X, quantile(X, 0.7))
quantile(X, cdf(X, 0.7))
```

`quantile.Gumbel` *Determine quantiles of a Gumbel distribution*

Description

`quantile()` is the inverse of `cdf()`.

Usage

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

Arguments

`x`	A `Gumbel` object created by a call to `Gumbel()`.
`probs`	A vector of probabilities.
`drop`	logical. Should the result be simplified to a vector if possible?
`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.
`...`	Arguments to be passed to `qgev`. Unevaluated arguments will generate a warning to catch mispellings or other possible errors.

Value

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.

Examples

```
set.seed(27)

X <- Gumbel(1, 2)
X

random(X, 10)

pdf(X, 0.7)
log_pdf(X, 0.7)

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

cdf(X, quantile(X, 0.7))
quantile(X, cdf(X, 0.7))
```

---

quantile.HurdleNegativeBinomial

Determine quantiles of a hurdle negative binomial distribution

---

Description

`quantile()` is the inverse of `cdf()`.

Usage

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

Arguments

`x`	A `HurdleNegativeBinomial` object created by a call to `HurdleNegativeBinomial()`.
`probs`	A vector of probabilities.
`drop`	logical. Should the result be simplified to a vector if possible?
`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.
`...`	Arguments to be passed to `qhnbinom`. Unevaluated arguments will generate a warning to catch mispellings or other possible errors.

Value

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.

Examples

```
## set up a hurdle negative binomial distribution
X <- HurdleNegativeBinomial(mu = 2.5, theta = 1, pi = 0.75)
X

## standard functions
pdf(X, 0:8)
cdf(X, 0:8)
quantile(X, seq(0, 1, by = 0.25))

## cdf() and quantile() are inverses for each other
quantile(X, cdf(X, 3))

## density visualization
plot(0:8, pdf(X, 0:8), type = "h", lwd = 2)

## corresponding sample with histogram of empirical frequencies
set.seed(0)
x <- random(X, 500)
hist(x, breaks = -1:max(x) + 0.5)
```

`quantile.HurdlePoisson`

Determine quantiles of a hurdle Poisson distribution

Description

`quantile()` is the inverse of `cdf()`.

Usage

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

Arguments

`x`	A `HurdlePoisson` object created by a call to `HurdlePoisson()`.
`probs`	A vector of probabilities.
`drop`	logical. Should the result be simplified to a vector if possible?
`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.
`...`	Arguments to be passed to `qhpois`. Unevaluated arguments will generate a warning to catch mispellings or other possible errors.

Value

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.

Examples

```
## set up a hurdle Poisson distribution
X <- HurdlePoisson(lambda = 2.5, pi = 0.75)
X

## standard functions
pdf(X, 0:8)
cdf(X, 0:8)
quantile(X, seq(0, 1, by = 0.25))

## cdf() and quantile() are inverses for each other
quantile(X, cdf(X, 3))

## density visualization
plot(0:8, pdf(X, 0:8), type = "h", lwd = 2)

## corresponding sample with histogram of empirical frequencies
set.seed(0)
x <- random(X, 500)
hist(x, breaks = -1:max(x) + 0.5)
```

---

quantile.HyperGeometric

Determine quantiles of a HyperGeometric distribution

Description

Determine quantiles of a HyperGeometric distribution

Usage

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

Arguments

`x`	A `HyperGeometric` object created by a call to `HyperGeometric()`.
`probs`	A vector of probabilities.
`drop`	logical. Should the result be simplified to a vector if possible?

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.
...	Arguments to be passed to qhyper. Unevaluated arguments will generate a warning to catch mispellings or other possible errors.

Value

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.

See Also

Other HyperGeometric distribution: cdf.HyperGeometric(), pdf.HyperGeometric(), random.HyperGeometric()

Examples

```
set.seed(27)

X <- HyperGeometric(4, 5, 8)
X

random(X, 10)

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

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

---

quantile.Logistic *Determine quantiles of a Logistic distribution*

Description

Determine quantiles of a Logistic distribution

Usage

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

Arguments

x	A Logistic object created by a call to Logistic().
probs	A vector of probabilities.
drop	logical. Should the result be simplified to a vector if possible?
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.
...	Arguments to be passed to qlogis. Unevaluated arguments will generate a warning to catch mispellings or other possible errors.

Value

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.

See Also

Other Logistic distribution: cdf.Logistic(), pdf.Logistic(), random.Logistic()

Examples

```
set.seed(27)

X <- Logistic(2, 4)
X

random(X, 10)

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

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

---

quantile.LogNormal *Determine quantiles of a LogNormal distribution*

Description

Determine quantiles of a LogNormal distribution

Usage

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

Arguments

x	A LogNormal object created by a call to LogNormal().
probs	A vector of probabilities.
drop	logical. Should the result be simplified to a vector if possible?
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.
...	Arguments to be passed to qlnorm. Unevaluated arguments will generate a warning to catch mispellings or other possible errors.

Value

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.

See Also

Other LogNormal distribution: cdf.LogNormal(), fit_mle.LogNormal(), pdf.LogNormal(), random.LogNormal()

Examples

```
set.seed(27)

X <- LogNormal(0.3, 2)
X

random(X, 10)

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

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

---

quantile.NegativeBinomial

Determine quantiles of a NegativeBinomial distribution

Description

Determine quantiles of a NegativeBinomial distribution

Usage

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

Arguments

x	A NegativeBinomial object created by a call to NegativeBinomial().
probs	A vector of probabilities.
drop	logical. Should the result be simplified to a vector if possible?
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.
...	Arguments to be passed to qnbinom. Unevaluated arguments will generate a warning to catch mispellings or other possible errors.

Value

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.

See Also

Other NegativeBinomial distribution: cdf.NegativeBinomial(), pdf.NegativeBinomial(), random.NegativeBinomial

Examples

```
set.seed(27)

X <- NegativeBinomial(size = 5, p = 0.1)
X

random(X, 10)

pdf(X, 50)
log_pdf(X, 50)

cdf(X, 50)
quantile(X, 0.7)

## alternative parameterization of X
Y <- NegativeBinomial(mu = 45, size = 5)
Y
cdf(Y, 50)
quantile(Y, 0.7)
```

quantile.Normal *Determine quantiles of a Normal distribution*

Description

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

Usage

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

Arguments

`x`	A `Normal` object created by a call to `Normal()`.
`probs`	A vector of probabilities.
`drop`	logical. Should the result be simplified to a vector if possible?
`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.
`...`	Arguments to be passed to `qnorm`. Unevaluated arguments will generate a warning to catch mispellings or other possible errors.

Details

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 `Normal()` for some properties of the Normal distribution, as well as extensive examples showing to how calculate p-values and confidence intervals.

Value

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.

See Also

Other Normal distribution: `cdf.Normal()`, `fit_mle.Normal()`, `pdf.Normal()`

Examples

```
set.seed(27)

X <- Normal(5, 2)
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: calculating p-values for two-sided Z-test

# here the null hypothesis is H_0: mu = 3
# and we assume sigma = 2

# exactly the same as: Z <- Normal(0, 1)
Z <- Normal()

# data to test
x <- c(3, 7, 11, 0, 7, 0, 4, 5, 6, 2)
nx <- length(x)

# calculate the z-statistic
z_stat <- (mean(x) - 3) / (2 / sqrt(nx))
z_stat

# calculate the two-sided p-value
1 - cdf(Z, abs(z_stat)) + cdf(Z, -abs(z_stat))

# exactly equivalent to the above
2 * cdf(Z, -abs(z_stat))

# p-value for one-sided test
# H_0: mu <= 3   vs   H_A: mu > 3
1 - cdf(Z, z_stat)

# p-value for one-sided test
# H_0: mu >= 3   vs   H_A: mu < 3
cdf(Z, z_stat)

### example: calculating a 88 percent Z CI for a mean

# same `x` as before, still assume `sigma = 2`
```

```
# lower-bound
mean(x) - quantile(Z, 1 - 0.12 / 2) * 2 / sqrt(nx)

# upper-bound
mean(x) + quantile(Z, 1 - 0.12 / 2) * 2 / sqrt(nx)

# equivalent to
mean(x) + c(-1, 1) * quantile(Z, 1 - 0.12 / 2) * 2 / sqrt(nx)

# also equivalent to
mean(x) + quantile(Z, 0.12 / 2) * 2 / sqrt(nx)
mean(x) + quantile(Z, 1 - 0.12 / 2) * 2 / sqrt(nx)

### generating random samples and plugging in ks.test()

set.seed(27)

# generate a random sample
ns <- random(Normal(3, 7), 26)

# test if sample is Normal(3, 7)
ks.test(ns, pnorm, mean = 3, sd = 7)

# test if sample is gamma(8, 3) using base R pgamma()
ks.test(ns, pgamma, shape = 8, rate = 3)

### MISC

# note that the cdf() and quantile() functions are inverses
cdf(X, quantile(X, 0.7))
quantile(X, cdf(X, 7))
```

`quantile.Poisson` *Determine quantiles of a Poisson distribution*

Description

`quantile()` is the inverse of `cdf()`.

Usage

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

Arguments

`x`	A `Poisson` object created by a call to `Poisson()`.
`probs`	A vector of probabilities.

drop	logical. Should the result be simplified to a vector if possible?
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.
...	Arguments to be passed to qpois. Unevaluated arguments will generate a warning to catch mispellings or other possible errors.

Value

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.

Examples

```
set.seed(27)

X <- Poisson(2)
X

random(X, 10)

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

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

cdf(X, quantile(X, 0.7))
quantile(X, cdf(X, 7))
```

---

quantile.PoissonBinomial

Determine quantiles of a PoissonBinomial distribution

Description

quantile() is the inverse of cdf().

Usage

```
## S3 method for class 'PoissonBinomial'
quantile(
  x,
  probs,
  drop = TRUE,
```

```
  elementwise = NULL,
  lower.tail = TRUE,
  log.p = FALSE,
  verbose = TRUE,
  ...
)
```

Arguments

x	A PoissonBinomial object created by a call to PoissonBinomial().
probs	A vector of probabilities.
drop	logical. Shoul the result be simplified to a vector if possible?
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.
lower.tail, log.p, ...	Arguments to be passed to qpbinom or qnorm, respectively.
verbose	logical. Should a warning be issued if the normal approximation is applied because the **PoissonBinomial** package is not installed?

Value

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.

Examples

```
set.seed(27)

X <- PoissonBinomial(0.5, 0.3, 0.8)
X

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

random(X, 10)

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

cdf(X, 2)
quantile(X, 0.8)

cdf(X, quantile(X, 0.8))
```

```
quantile(X, cdf(X, 2))

## equivalent definitions of four Poisson binomial distributions
## each summing up three Bernoulli probabilities
p <- cbind(
  p1 = c(0.1, 0.2, 0.1, 0.2),
  p2 = c(0.5, 0.5, 0.5, 0.5),
  p3 = c(0.8, 0.7, 0.9, 0.8))
PoissonBinomial(p)
PoissonBinomial(p[, 1], p[, 2], p[, 3])
PoissonBinomial(p[, 1:2], p[, 3])
```

`quantile.RevWeibull` *Determine quantiles of a RevWeibull distribution*

Description

`quantile()` is the inverse of `cdf()`.

Usage

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

Arguments

`x`	A `RevWeibull` object created by a call to `RevWeibull()`.
`probs`	A vector of probabilities.
`drop`	logical. Should the result be simplified to a vector if possible?
`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.
`...`	Arguments to be passed to `qgev`. Unevaluated arguments will generate a warning to catch mispellings or other possible errors.

Value

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.

Examples

```
set.seed(27)

X <- RevWeibull(1, 2)
X

random(X, 10)

pdf(X, 0.7)
log_pdf(X, 0.7)

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

cdf(X, quantile(X, 0.7))
quantile(X, cdf(X, 0.7))
```

---

quantile.SinhArcsinh *Determine quantiles of a Sinh-Arcsinh distribution*

Description

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

Usage

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

Arguments

x	A SinhArcsinh object created by a call to SinhArcsinh().
probs	A vector of probabilities.
drop	logical. Should the result be simplified to a vector if possible?
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.
cores	NULL or positive integer. TODO(R): Just a development option. If not NULL we use the C code with cores threads.
...	Arguments to be passed to qnorm. Unevaluated arguments will generate a warning to catch mispellings or other possible errors.

Details

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 `SinhArcsinh()` for some properties of the Sinh-Arcsinh distribution, as well as extensive examples showing to how calculate p-values and confidence intervals.

Value

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.

See Also

Other SinhArcsinh distribution: `cdf.SinhArcsinh()`, `pdf.SinhArcsinh()`

Examples

```
## SinhArcsinh() by default uses nu = 1, tau = 1 which
## results in the standard normal distribution
set.seed(6020)
X <- SinhArcsinh() # Uses mu = 1, sigma = 0, nu = 1, tau = 1)
x <- random(X, 300)
qqnorm(x); qqline(x, col = 2, lwd = 2)
curve(pdf(X, x), xlim = c(-5, 5), main = paste(X, "density"))

## Calculation of central moments is based on numeric integration,
## thus not being identical to the standard normal distribution
c(mean = mean(x), sd = sd(x))

## Skewed Sinh-Arcsinh distribution
X <- SinhArcsinh(mu = 7, sigma = 2, nu = c(0.7, 1, 0.7), tau = c(1, 0.7, 0.7))
as.matrix(X)

## Visualization of density functions using different parameters for nu/tau
curve(pdf(X[1], x), xlim = c(0, 20), ylim = c(0, 0.2), main = "Density function")
curve(pdf(X[2], x), xlim = c(0, 20), col = 2, add = TRUE)
curve(pdf(X[3], x), xlim = c(0, 20), col = 4, add = TRUE)

## Visualization of distribution function using different parameters for nu/tau
curve(cdf(X[1], x), xlim = c(0, 20), ylim = 0:1, main = "Distribution function")
curve(cdf(X[2], x), xlim = c(0, 20), col = 2, add = TRUE)
curve(cdf(X[3], x), xlim = c(0, 20), col = 4, add = TRUE)

## Central moments
mean(X)
variance(X)
skewness(X)
kurtosis(X)

## Drawing random values
```

```
random(X, 10)

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

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

# note that the cdf() and quantile() functions are inverses
X <- SinhArcsinh(mu = 3, sigma = 2, nu = 0.9, tau = 1.2)
cdf(X, quantile(X, 0.7))
quantile(X, cdf(X, 7))
```

`quantile.StudentsT` *Determine quantiles of a StudentsT distribution*

Description

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

Usage

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

Arguments

`x`	A `StudentsT` object created by a call to `StudentsT()`.
`probs`	A vector of probabilities.
`drop`	logical. Should the result be simplified to a vector if possible?
`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.
`...`	Arguments to be passed to `qt`. Unevaluated arguments will generate a warning to catch mispellings or other possible errors.

Details

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 `StudentsT()` for some properties of the StudentsT distribution, as well as extensive examples showing to how calculate p-values and confidence intervals.

Value

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.

See Also

Other StudentsT distribution: cdf.StudentsT(), pdf.StudentsT(), random.StudentsT()

Examples

```
set.seed(27)

X <- StudentsT(3)
X

random(X, 10)

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

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

### example: calculating p-values for two-sided T-test

# here the null hypothesis is H_0: mu = 3

# data to test
x <- c(3, 7, 11, 0, 7, 0, 4, 5, 6, 2)
nx <- length(x)

# calculate the T-statistic
t_stat <- (mean(x) - 3) / (sd(x) / sqrt(nx))
t_stat

# null distribution of statistic depends on sample size!
T <- StudentsT(df = nx - 1)

# calculate the two-sided p-value
1 - cdf(T, abs(t_stat)) + cdf(T, -abs(t_stat))

# exactly equivalent to the above
2 * cdf(T, -abs(t_stat))

# p-value for one-sided test
# H_0: mu <= 3   vs   H_A: mu > 3
1 - cdf(T, t_stat)

# p-value for one-sided test
# H_0: mu >= 3   vs   H_A: mu < 3
cdf(T, t_stat)
```

```
### example: calculating a 88 percent T CI for a mean

# lower-bound
mean(x) - quantile(T, 1 - 0.12 / 2) * sd(x) / sqrt(nx)

# upper-bound
mean(x) + quantile(T, 1 - 0.12 / 2) * sd(x) / sqrt(nx)

# equivalent to
mean(x) + c(-1, 1) * quantile(T, 1 - 0.12 / 2) * sd(x) / sqrt(nx)

# also equivalent to
mean(x) + quantile(T, 0.12 / 2) * sd(x) / sqrt(nx)
mean(x) + quantile(T, 1 - 0.12 / 2) * sd(x) / sqrt(nx)
```

`quantile.Tukey` *Determine quantiles of a Tukey distribution*

Description

Determine quantiles of a Tukey distribution

Usage

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

Arguments

`x`	A vector of elements whose cumulative probabilities you would like to determine given the distribution `d`.
`probs`	A vector of probabilities.
`drop`	logical. Should the result be simplified to a vector if possible?
`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.
`...`	Arguments to be passed to `qtukey`. Unevaluated arguments will generate a warning to catch mispellings or other possible errors.

Value

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.

See Also

Other Tukey distribution: cdf.Tukey()

Examples

```
set.seed(27)

X <- Tukey(4L, 16L, 2L)
X

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

---

quantile.Uniform *Determine quantiles of a continuous Uniform distribution*

Description

quantile() is the inverse of cdf().

Usage

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

Arguments

x	A Uniform object created by a call to Uniform().
probs	A vector of probabilities.
drop	logical. Should the result be simplified to a vector if possible?
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.
...	Arguments to be passed to qunif. Unevaluated arguments will generate a warning to catch mispellings or other possible errors.

Value

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.

Examples

```
set.seed(27)

X <- Uniform(1, 2)
X

random(X, 10)

pdf(X, 0.7)
log_pdf(X, 0.7)

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

cdf(X, quantile(X, 0.7))
quantile(X, cdf(X, 0.7))
```

`quantile.Weibull` *Determine quantiles of a Weibull distribution*

Description

Determine quantiles of a Weibull distribution

Usage

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

Arguments

`x`	A `Weibull` object created by a call to `Weibull()`.
`probs`	A vector of probabilities.
`drop`	logical. Should the result be simplified to a vector if possible?
`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.
`...`	Arguments to be passed to `qweibull`. Unevaluated arguments will generate a warning to catch mispellings or other possible errors.

Value

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.

See Also

Other Weibull distribution: cdf.Weibull(), pdf.Weibull(), random.Weibull()

Examples

```
set.seed(27)

X <- Weibull(0.3, 2)
X

random(X, 10)

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

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

---

quantile.ZINegativeBinomial

Determine quantiles of a zero-inflated negative binomial distribution

---

Description

quantile() is the inverse of cdf().

Usage

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

Arguments

x	A ZINegativeBinomial object created by a call to ZINegativeBinomial().
probs	A vector of probabilities.
drop	logical. Should the result be simplified to a vector if possible?
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.
...	Arguments to be passed to qzinbinom. Unevaluated arguments will generate a warning to catch mispellings or other possible errors.

Value

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.

Examples

```
## set up a zero-inflated negative binomial distribution
X <- ZINegativeBinomial(mu = 2.5, theta = 1, pi = 0.25)
X

## standard functions
pdf(X, 0:8)
cdf(X, 0:8)
quantile(X, seq(0, 1, by = 0.25))

## cdf() and quantile() are inverses for each other
quantile(X, cdf(X, 3))

## density visualization
plot(0:8, pdf(X, 0:8), type = "h", lwd = 2)

## corresponding sample with histogram of empirical frequencies
set.seed(0)
x <- random(X, 500)
hist(x, breaks = -1:max(x) + 0.5)
```

---

quantile.ZIPoisson *Determine quantiles of a zero-inflated Poisson distribution*

Description

quantile() is the inverse of cdf().

Usage

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

Arguments

x	A ZIPoisson object created by a call to ZIPoisson().
probs	A vector of probabilities.
drop	logical. Should the result be simplified to a vector if possible?
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.

... Arguments to be passed to qzipois. Unevaluated arguments will generate a warning to catch mispellings or other possible errors.

Value

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.

Examples

```
## set up a zero-inflated Poisson distribution
X <- ZIPoisson(lambda = 2.5, pi = 0.25)
X

## standard functions
pdf(X, 0:8)
cdf(X, 0:8)
quantile(X, seq(0, 1, by = 0.25))

## cdf() and quantile() are inverses for each other
quantile(X, cdf(X, 3))

## density visualization
plot(0:8, pdf(X, 0:8), type = "h", lwd = 2)

## corresponding sample with histogram of empirical frequencies
set.seed(0)
x <- random(X, 500)
hist(x, breaks = -1:max(x) + 0.5)
```

---

quantile.ZTNegativeBinomial

Determine quantiles of a zero-truncated negative binomial distribution

---

Description

quantile() is the inverse of cdf().

Usage

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

Arguments

x	A ZTNegativeBinomial object created by a call to ZTNegativeBinomial().
probs	A vector of probabilities.
drop	logical. Should the result be simplified to a vector if possible?

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.

... — Arguments to be passed to qztnbinom. Unevaluated arguments will generate a warning to catch mispellings or other possible errors.

Value

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.

Examples

```
## set up a zero-truncated negative binomial distribution
X <- ZTNegativeBinomial(mu = 2.5, theta = 1)
X

## standard functions
pdf(X, 0:8)
cdf(X, 0:8)
quantile(X, seq(0, 1, by = 0.25))

## cdf() and quantile() are inverses for each other
quantile(X, cdf(X, 3))

## density visualization
plot(0:8, pdf(X, 0:8), type = "h", lwd = 2)

## corresponding sample with histogram of empirical frequencies
set.seed(0)
x <- random(X, 500)
hist(x, breaks = -1:max(x) + 0.5)
```

---

quantile.ZTPoisson *Determine quantiles of a zero-truncated Poisson distribution*

Description

quantile() is the inverse of cdf().

Usage

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

Arguments

x	A ZTPoisson object created by a call to ZTPoisson().
probs	A vector of probabilities.
drop	logical. Should the result be simplified to a vector if possible?
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.
...	Arguments to be passed to qztpois. Unevaluated arguments will generate a warning to catch mispellings or other possible errors.

Value

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.

Examples

```
## set up a zero-truncated Poisson distribution
X <- ZTPoisson(lambda = 2.5)
X

## standard functions
pdf(X, 0:8)
cdf(X, 0:8)
quantile(X, seq(0, 1, by = 0.25))

## cdf() and quantile() are inverses for each other
quantile(X, cdf(X, 3))

## density visualization
plot(0:8, pdf(X, 0:8), type = "h", lwd = 2)

## corresponding sample with histogram of empirical frequencies
set.seed(0)
x <- random(X, 500)
hist(x, breaks = -1:max(x) + 0.5)
```

---

random *Draw a random sample from a probability distribution*

Description

Generic function for drawing random samples from distribution objects.

Usage

```
random(x, n = 1L, drop = TRUE, ...)

## S3 method for class 'distribution'
simulate(object, nsim = 1L, seed = NULL, ...)
```

Arguments

x, object	An object. The package provides methods for distribution objects such as those from Normal() or Binomial() etc.
n, nsim	The number of samples to draw. Should be a positive integer. Defaults to 1L.
drop	logical. Should the result be simplified to a vector if possible?
...	Arguments passed to methods. Unevaluated arguments will generate a warning to catch mispellings or other possible errors.
seed	An optional random seed that is to be set using set.seed prior to drawing the random sample. The previous random seed from the global environment (if any) is restored afterwards.

Details

random is a new generic for drawing random samples from the S3 distribution objects provided in this package, such as Normal or Binomial etc. The respective methods typically call the "r" function for the corresponding distribution functions provided in base R such as rnorm, rbinom etc.

In addition to the new random generic there is also a simulate method for distribution objects which simply calls the random method internally.

Value

Random samples drawn from the distriubtion x. The random methods typically return either a matrix or, if possible, a vector. The simulate method always returns a data frame (with an attribute "seed" containing the .Random.seed from before the simulation).

Examples

```
## distribution object
X <- Normal()
## 10 random samples
random(X, 10)
```

`random.Bernoulli` *Draw a random sample from a Bernoulli distribution*

Description

Draw a random sample from a Bernoulli distribution

Usage

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

Arguments

`x`	A `Bernoulli` object created by a call to `Bernoulli()`.
`n`	The number of samples to draw. Defaults to `1L`.
`drop`	logical. Should the result be simplified to a vector if possible?
`...`	Unused. Unevaluated arguments will generate a warning to catch mispellings or other possible errors.

Value

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`).

Examples

```
set.seed(27)

X <- Bernoulli(0.7)
X

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

random(X, 10)
pdf(X, 1)
log_pdf(X, 1)
cdf(X, 0)
quantile(X, 0.7)

cdf(X, quantile(X, 0.7))
quantile(X, cdf(X, 0.7))
```

`random.Beta` *Draw a random sample from a Beta distribution*

Description

Draw a random sample from a Beta distribution

Usage

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

Arguments

`x`	A `Beta` object created by a call to `Beta()`.
`n`	The number of samples to draw. Defaults to `1L`.
`drop`	logical. Should the result be simplified to a vector if possible?
`...`	Unused. Unevaluated arguments will generate a warning to catch mispellings or other possible errors.

Value

Values in `[0, 1]`. 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`).

Examples

```
set.seed(27)

X <- Beta(1, 2)
X

random(X, 10)

pdf(X, 0.7)
log_pdf(X, 0.7)

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

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

cdf(X, quantile(X, 0.7))
quantile(X, cdf(X, 0.7))
```

`random.Binomial` *Draw a random sample from a Binomial distribution*

Description

Draw a random sample from a Binomial distribution

Usage

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

Arguments

`x`	A `Binomial` object created by a call to `Binomial()`.
`n`	The number of samples to draw. Defaults to `1L`.
`drop`	logical. Should the result be simplified to a vector if possible?
`...`	Unused. Unevaluated arguments will generate a warning to catch mispellings or other possible errors.

Value

Integers containing values between `0` and `x$size`. 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`).

Examples

```
set.seed(27)

X <- Binomial(10, 0.2)
X

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

random(X, 10)

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

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

cdf(X, quantile(X, 0.7))
quantile(X, cdf(X, 7))
```

`random.Categorical` *Draw a random sample from a Categorical distribution*

Description

Draw a random sample from a Categorical distribution

Usage

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

Arguments

`x`	A `Categorical` object created by a call to `Categorical()`.
`n`	The number of samples to draw. Defaults to `1L`.
`...`	Unused. Unevaluated arguments will generate a warning to catch mispellings or other possible errors.

Value

A vector containing values from `outcomes` of length `n`.

Examples

```
set.seed(27)

X <- Categorical(1:3, p = c(0.4, 0.1, 0.5))
X

Y <- Categorical(LETTERS[1:4])
Y

random(X, 10)
random(Y, 10)

pdf(X, 1)
log_pdf(X, 1)

cdf(X, 1)
quantile(X, 0.5)

# cdfs are only defined for numeric sample spaces. this errors!
# cdf(Y, "a")

# same for quantiles. this also errors!
# quantile(Y, 0.7)
```

`random.Cauchy` *Draw a random sample from a Cauchy distribution*

Description

Draw a random sample from a Cauchy distribution

Usage

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

Arguments

`x`	A `Cauchy` object created by a call to `Cauchy()`.
`n`	The number of samples to draw. Defaults to `1L`.
`drop`	logical. Should the result be simplified to a vector if possible?
`...`	Unused. Unevaluated arguments will generate a warning to catch mispellings or other possible errors.

Value

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`).

Examples

```
set.seed(27)

X <- Cauchy(10, 0.2)
X

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

random(X, 10)

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

cdf(X, 2)
quantile(X, 0.7)

cdf(X, quantile(X, 0.7))
quantile(X, cdf(X, 7))
```

`random.ChiSquare` *Draw a random sample from a chi square distribution*

Description

Draw a random sample from a chi square distribution

Usage

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

Arguments

`x`	A `ChiSquare` object created by a call to `ChiSquare()`.
`n`	The number of samples to draw. Defaults to `1L`.
`drop`	logical. Should the result be simplified to a vector if possible?
`...`	Unused. Unevaluated arguments will generate a warning to catch mispellings or other possible errors.

Value

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`).

Examples

```
set.seed(27)

X <- ChiSquare(5)
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)

cdf(X, quantile(X, 0.7))
quantile(X, cdf(X, 7))
```

`random.distribution` *Inverse transform sampling for random number generation*

Description

Generates random numbers from a distribution object by mapping uniform random variates through the target quantile function.

Usage

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

Arguments

`x`	An object of class `distribution`.
`n`	Number of observations. If `length(n) > 1`, the length is taken to be the number required.
`drop`	logical. Should the result be simplified to a vector if possible?
`...`	currently unused.

Value

A numeric vector of random values if `length(d)` equals one or `n = 1L` and `drop = TRUE` (default), or a matrix where rows correspond to the distribution(s) `d` whilst the columns contain the random values.

Examples

```
## Drawing random numbers from a Poisson distribution
## using the inverse transform sampling method
random.distribution(Poisson(3), n = 6)

## Drawing random numbers from a series of Normal distributions
## using the inverse transform sampling method
random.distribution(Normal(1:3, 1:3 / 2), n = 6)
```

`random.Empirical` *Draw a random sample from an Empirical distribution*

Description

Draws n random values from the empirical distribution with replacement. Please see the documentation of `Empirical()` for some properties of the empircal ensemble distribution.

Usage

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

Arguments

`x`	A n `Empirical` object created by a call to `Empirical()`.
`n`	The number of samples to draw. Defaults to `1L`.
`drop`	logical. Should the result be simplified to a vector if possible?
`...`	currently unused.

Value

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`).

See Also

Other Empirical distribution: `Empirical()`, `cdf.Empirical()`, `dempirical()`, `pdf.Empirical()`, `quantile.Empirical()`, `support.Empirical()`

Examples

```
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
Y4 <- matrix(rnorm(20), ncol = 5,
             dimnames = list(paste0("D_", 1:4), paste0("obs_", 1:5)))
d4 <- Empirical(Y4)
d4

## Data frame
d5 <- Empirical(as.data.frame(Y4))
d5

identical(d4, d5)

mean(d5)
variance(d5)
skewness(d5)
kurtosis(d5)

pdf(d5, c(-0.5, 0, 0.5, 1)) # Defaults to elementwise = TRUE
pdf(d5, c(-0.5, 0, 0.5, 1), elementwise = FALSE)

cdf(d5, c(-0.5, 0, 0.5, 1)) # Defaults to elementwise = TRUE
cdf(d5, c(-0.5, 0, 0.5, 1), elementwise = FALSE)

quantile(d5, c(0.2, 0.4, 0.6, 0.8)) # Defaults to elementwise = TRUE
quantile(d5, c(0.2, 0.4, 0.6, 0.8), elementwise = FALSE)

## The quantile function is the inverse of the distribution
```

```
## function (cdf) if x in Y
set.seed(6020)
Y <- round(rlnorm(20, log(3), log(2)), 1)
d <- Empirical(Y)

cdf(d, 4.0)
quantile(d, cdf(d, 4.0))
```

`random.Erlang` *Draw a random sample from an Erlang distribution*

Description

Draw a random sample from an Erlang distribution

Usage

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

Arguments

`x`	An `Erlang` object created by a call to `Erlang()`.
`n`	The number of samples to draw. Defaults to `1L`.
`drop`	logical. Should the result be simplified to a vector if possible?
`...`	Unused. Unevaluated arguments will generate a warning to catch mispellings or other possible errors.

Value

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`).

Examples

```
set.seed(27)

X <- Erlang(5, 2)
X

random(X, 10)

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

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

```
cdf(X, quantile(X, 0.7))
quantile(X, cdf(X, 7))
```

`random.Exponential` *Draw a random sample from an Exponential distribution*

Description

Draw a random sample from an Exponential distribution

Usage

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

Arguments

`x`	An `Exponential` object created by a call to `Exponential()`.
`n`	The number of samples to draw. Defaults to `1L`.
`drop`	logical. Should the result be simplified to a vector if possible?
`...`	Unused. Unevaluated arguments will generate a warning to catch mispellings or other possible errors.

Value

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`).

Examples

```
set.seed(27)

X <- Exponential(5)
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)
```

```
cdf(X, quantile(X, 0.7))
quantile(X, cdf(X, 7))
```

`random.FisherF` *Draw a random sample from an F distribution*

Description

Draw a random sample from an F distribution

Usage

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

Arguments

`x`	A `FisherF` object created by a call to `FisherF()`.
`n`	The number of samples to draw. Defaults to `1L`.
`drop`	logical. Should the result be simplified to a vector if possible?
`...`	Unused. Unevaluated arguments will generate a warning to catch mispellings or other possible errors.

Value

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`).

Examples

```
set.seed(27)

X <- FisherF(5, 10, 0.2)
X

random(X, 10)

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

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

cdf(X, quantile(X, 0.7))
quantile(X, cdf(X, 7))
```

`random.Frechet` *Draw a random sample from a Frechet distribution*

Description

Draw a random sample from a Frechet distribution

Usage

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

Arguments

`x`	A `Frechet` object created by a call to `Frechet()`.
`n`	The number of samples to draw. Defaults to `1L`.
`drop`	logical. Should the result be simplified to a vector if possible?
`...`	Unused. Unevaluated arguments will generate a warning to catch mispellings or other possible errors.

Value

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`).

Examples

```
set.seed(27)

X <- Frechet(0, 2)
X

random(X, 10)

pdf(X, 0.7)
log_pdf(X, 0.7)

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

cdf(X, quantile(X, 0.7))
quantile(X, cdf(X, 0.7))
```

`random.Gamma` *Draw a random sample from a Gamma distribution*

Description

Draw a random sample from a Gamma distribution

Usage

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

Arguments

`x`	A `Gamma` object created by a call to `Gamma()`.
`n`	The number of samples to draw. Defaults to `1L`.
`drop`	logical. Should the result be simplified to a vector if possible?
`...`	Unused. Unevaluated arguments will generate a warning to catch mispellings or other possible errors.

Value

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`).

Examples

```
set.seed(27)

X <- Gamma(5, 2)
X

random(X, 10)

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

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

cdf(X, quantile(X, 0.7))
quantile(X, cdf(X, 7))
```

random.Geometric *Draw a random sample from a Geometric distribution*

Description

Please see the documentation of `Geometric()` for some properties of the Geometric distribution, as well as extensive examples showing to how calculate p-values and confidence intervals.

Usage

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

Arguments

x	A `Geometric` object created by a call to `Geometric()`.
n	The number of samples to draw. Defaults to `1L`.
drop	logical. Should the result be simplified to a vector if possible?
...	Unused. Unevaluated arguments will generate a warning to catch mispellings or other possible errors.

Value

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`).

See Also

Other Geometric distribution: `cdf.Geometric()`, `pdf.Geometric()`, `quantile.Geometric()`

Examples

```
set.seed(27)

X <- Geometric(0.3)
X

random(X, 10)

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

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

`random.GEV` *Draw a random sample from a GEV distribution*

Description

Draw a random sample from a GEV distribution

Usage

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

Arguments

`x`	A `GEV` object created by a call to `GEV()`.
`n`	The number of samples to draw. Defaults to `1L`.
`drop`	logical. Should the result be simplified to a vector if possible?
`...`	Unused. Unevaluated arguments will generate a warning to catch mispellings or other possible errors.

Value

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`).

Examples

```
set.seed(27)

X <- GEV(1, 2, 0.1)
X

random(X, 10)

pdf(X, 0.7)
log_pdf(X, 0.7)

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

cdf(X, quantile(X, 0.7))
quantile(X, cdf(X, 0.7))
```

`random.GP` *Draw a random sample from a GP distribution*

Description

Draw a random sample from a GP distribution

Usage

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

Arguments

`x`	A `GP` object created by a call to `GP()`.
`n`	The number of samples to draw. Defaults to `1L`.
`drop`	logical. Should the result be simplified to a vector if possible?
`...`	Unused. Unevaluated arguments will generate a warning to catch mispellings or other possible errors.

Value

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`).

Examples

```
set.seed(27)

X <- GP(0, 2, 0.1)
X

random(X, 10)

pdf(X, 0.7)
log_pdf(X, 0.7)

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

cdf(X, quantile(X, 0.7))
quantile(X, cdf(X, 0.7))
```

`random.Gumbel` *Draw a random sample from a Gumbel distribution*

Description

Draw a random sample from a Gumbel distribution

Usage

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

Arguments

`x`	A `Gumbel` object created by a call to `Gumbel()`.
`n`	The number of samples to draw. Defaults to `1L`.
`drop`	logical. Should the result be simplified to a vector if possible?
`...`	Unused. Unevaluated arguments will generate a warning to catch mispellings or other possible errors.

Value

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`).

Examples

```
set.seed(27)

X <- Gumbel(1, 2)
X

random(X, 10)

pdf(X, 0.7)
log_pdf(X, 0.7)

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

cdf(X, quantile(X, 0.7))
quantile(X, cdf(X, 0.7))
```

`random.HurdleNegativeBinomial` *Draw a random sample from a hurdle negative binomial distribution*

Description

Draw a random sample from a hurdle negative binomial distribution

Usage

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

Arguments

`x`	A `HurdleNegativeBinomial` object created by a call to `HurdleNegativeBinomial()`.
`n`	The number of samples to draw. Defaults to `1L`.
`drop`	logical. Should the result be simplified to a vector if possible?
`...`	Unused. Unevaluated arguments will generate a warning to catch mispellings or other possible errors.

Value

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`).

Examples

```
## set up a hurdle negative binomial distribution
X <- HurdleNegativeBinomial(mu = 2.5, theta = 1, pi = 0.75)
X

## standard functions
pdf(X, 0:8)
cdf(X, 0:8)
quantile(X, seq(0, 1, by = 0.25))

## cdf() and quantile() are inverses for each other
quantile(X, cdf(X, 3))

## density visualization
plot(0:8, pdf(X, 0:8), type = "h", lwd = 2)

## corresponding sample with histogram of empirical frequencies
set.seed(0)
x <- random(X, 500)
hist(x, breaks = -1:max(x) + 0.5)
```

`random.HurdlePoisson` *Draw a random sample from a hurdle Poisson distribution*

Description

Draw a random sample from a hurdle Poisson distribution

Usage

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

Arguments

`x`	A `HurdlePoisson` object created by a call to `HurdlePoisson()`.
`n`	The number of samples to draw. Defaults to `1L`.
`drop`	logical. Should the result be simplified to a vector if possible?
`...`	Unused. Unevaluated arguments will generate a warning to catch mispellings or other possible errors.

Value

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`).

Examples

```
## set up a hurdle Poisson distribution
X <- HurdlePoisson(lambda = 2.5, pi = 0.75)
X

## standard functions
pdf(X, 0:8)
cdf(X, 0:8)
quantile(X, seq(0, 1, by = 0.25))

## cdf() and quantile() are inverses for each other
quantile(X, cdf(X, 3))

## density visualization
plot(0:8, pdf(X, 0:8), type = "h", lwd = 2)

## corresponding sample with histogram of empirical frequencies
set.seed(0)
x <- random(X, 500)
hist(x, breaks = -1:max(x) + 0.5)
```

`random.HyperGeometric` *Draw a random sample from a HyperGeometric distribution*

Description

Please see the documentation of `HyperGeometric()` for some properties of the HyperGeometric distribution, as well as extensive examples showing to how calculate p-values and confidence intervals.

Usage

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

Arguments

`x`	A `HyperGeometric` object created by a call to `HyperGeometric()`.
`n`	The number of samples to draw. Defaults to `1L`.
`drop`	logical. Should the result be simplified to a vector if possible?
`...`	Unused. Unevaluated arguments will generate a warning to catch mispellings or other possible errors.

Value

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`).

See Also

Other HyperGeometric distribution: `cdf.HyperGeometric()`, `pdf.HyperGeometric()`, `quantile.HyperGeometric()`

Examples

```
set.seed(27)

X <- HyperGeometric(4, 5, 8)
X

random(X, 10)

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

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

`random.Logistic` *Draw a random sample from a Logistic distribution*

Description

Draw a random sample from a Logistic distribution

Usage

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

Arguments

`x`	A `Logistic` object created by a call to `Logistic()`.
`n`	The number of samples to draw. Defaults to `1L`.
`drop`	logical. Should the result be simplified to a vector if possible?
`...`	Unused. Unevaluated arguments will generate a warning to catch mispellings or other possible errors.

Value

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`).

See Also

Other Logistic distribution: `cdf.Logistic()`, `pdf.Logistic()`, `quantile.Logistic()`

Examples

```
set.seed(27)

X <- Logistic(2, 4)
X

random(X, 10)

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

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

`random.LogNormal` *Draw a random sample from a LogNormal distribution*

Description

Draw a random sample from a LogNormal distribution

Usage

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

Arguments

`x`	A `LogNormal` object created by a call to `LogNormal()`.
`n`	The number of samples to draw. Defaults to `1L`.
`drop`	logical. Should the result be simplified to a vector if possible?
`...`	Unused. Unevaluated arguments will generate a warning to catch mispellings or other possible errors.

Value

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`).

See Also

Other LogNormal distribution: `cdf.LogNormal()`, `fit_mle.LogNormal()`, `pdf.LogNormal()`, `quantile.LogNormal()`

Examples

```
set.seed(27)

X <- LogNormal(0.3, 2)
X

random(X, 10)

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

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

`random.Multinomial` *Draw a random sample from a Multinomial distribution*

Description

Draw a random sample from a Multinomial distribution

Usage

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

Arguments

`x`	A `Multinomial` object created by a call to `Multinomial()`.
`n`	The number of samples to draw. Defaults to `1L`.
`...`	Unused. Unevaluated arguments will generate a warning to catch mispellings or other possible errors.

Value

An integer vector of length `n`.

See Also

Other Multinomial distribution: `pdf.Multinomial()`

Examples

```
set.seed(27)

X <- Multinomial(size = 5, p = c(0.3, 0.4, 0.2, 0.1))
X

random(X, 10)

# pdf(X, 2)
# log_pdf(X, 2)
```

`random.NegativeBinomial` *Draw a random sample from a negative binomial distribution*

Description

Draw a random sample from a negative binomial distribution

Usage

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

Arguments

`x`	A `NegativeBinomial` object created by a call to `NegativeBinomial()`.
`n`	The number of samples to draw. Defaults to `1L`.
`drop`	logical. Should the result be simplified to a vector if possible?
`...`	Unused. Unevaluated arguments will generate a warning to catch mispellings or other possible errors.

Value

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`).

See Also

Other NegativeBinomial distribution: `cdf.NegativeBinomial()`, `pdf.NegativeBinomial()`, `quantile.NegativeBinomi`

Examples

```
set.seed(27)

X <- NegativeBinomial(size = 5, p = 0.1)
X

random(X, 10)

pdf(X, 50)
log_pdf(X, 50)

cdf(X, 50)
quantile(X, 0.7)

## alternative parameterization of X
Y <- NegativeBinomial(mu = 45, size = 5)
Y
```

```
cdf(Y, 50)
quantile(Y, 0.7)
```

`random.Normal` *Draw a random sample from a Normal distribution*

Description

Please see the documentation of `Normal()` for some properties of the Normal distribution, as well as extensive examples showing to how calculate p-values and confidence intervals.

Usage

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

Arguments

`x`	A `Normal` object created by a call to `Normal()`.
`n`	The number of samples to draw. Defaults to `1L`.
`drop`	logical. Should the result be simplified to a vector if possible?
`...`	Unused. Unevaluated arguments will generate a warning to catch mispellings or other possible errors.

Value

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`).

Examples

```
set.seed(27)

X <- Normal(5, 2)
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: calculating p-values for two-sided Z-test

# here the null hypothesis is H_0: mu = 3
# and we assume sigma = 2

# exactly the same as: Z <- Normal(0, 1)
Z <- Normal()

# data to test
x <- c(3, 7, 11, 0, 7, 0, 4, 5, 6, 2)
nx <- length(x)

# calculate the z-statistic
z_stat <- (mean(x) - 3) / (2 / sqrt(nx))
z_stat

# calculate the two-sided p-value
1 - cdf(Z, abs(z_stat)) + cdf(Z, -abs(z_stat))

# exactly equivalent to the above
2 * cdf(Z, -abs(z_stat))

# p-value for one-sided test
# H_0: mu <= 3   vs   H_A: mu > 3
1 - cdf(Z, z_stat)

# p-value for one-sided test
# H_0: mu >= 3   vs   H_A: mu < 3
cdf(Z, z_stat)

### example: calculating a 88 percent Z CI for a mean

# same `x` as before, still assume `sigma = 2`

# lower-bound
mean(x) - quantile(Z, 1 - 0.12 / 2) * 2 / sqrt(nx)

# upper-bound
mean(x) + quantile(Z, 1 - 0.12 / 2) * 2 / sqrt(nx)

# equivalent to
mean(x) + c(-1, 1) * quantile(Z, 1 - 0.12 / 2) * 2 / sqrt(nx)

# also equivalent to
mean(x) + quantile(Z, 0.12 / 2) * 2 / sqrt(nx)
mean(x) + quantile(Z, 1 - 0.12 / 2) * 2 / sqrt(nx)

### generating random samples and plugging in ks.test()

set.seed(27)

# generate a random sample
```

```
ns <- random(Normal(3, 7), 26)

# test if sample is Normal(3, 7)
ks.test(ns, pnorm, mean = 3, sd = 7)

# test if sample is gamma(8, 3) using base R pgamma()
ks.test(ns, pgamma, shape = 8, rate = 3)

### MISC

# note that the cdf() and quantile() functions are inverses
cdf(X, quantile(X, 0.7))
quantile(X, cdf(X, 7))
```

`random.Poisson` *Draw a random sample from a Poisson distribution*

Description

Draw a random sample from a Poisson distribution

Usage

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

Arguments

`x`	A `Poisson` object created by a call to `Poisson()`.
`n`	The number of samples to draw. Defaults to `1L`.
`drop`	logical. Should the result be simplified to a vector if possible?
`...`	Unused. Unevaluated arguments will generate a warning to catch mispellings or other possible errors.

Value

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`).

Examples

```
set.seed(27)

X <- Poisson(2)
X

random(X, 10)
```

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

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

cdf(X, quantile(X, 0.7))
quantile(X, cdf(X, 7))
```

`random.PoissonBinomial` *Draw a random sample from a PoissonBinomial distribution*

Description

Draw a random sample from a PoissonBinomial distribution

Usage

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

Arguments

`x`	A `PoissonBinomial` object created by a call to `PoissonBinomial()`.
`n`	The number of samples to draw. Defaults to `1L`.
`drop`	logical. Should the result be simplified to a vector if possible?
`...`	Unused. Unevaluated arguments will generate a warning to catch mispellings or other possible errors.

Value

Integers containing values between `0` and `x$size`. 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`).

Examples

```
set.seed(27)

X <- PoissonBinomial(0.5, 0.3, 0.8)
X

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

```
random(X, 10)

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

cdf(X, 2)
quantile(X, 0.8)

cdf(X, quantile(X, 0.8))
quantile(X, cdf(X, 2))

## equivalent definitions of four Poisson binomial distributions
## each summing up three Bernoulli probabilities
p <- cbind(
  p1 = c(0.1, 0.2, 0.1, 0.2),
  p2 = c(0.5, 0.5, 0.5, 0.5),
  p3 = c(0.8, 0.7, 0.9, 0.8))
PoissonBinomial(p)
PoissonBinomial(p[, 1], p[, 2], p[, 3])
PoissonBinomial(p[, 1:2], p[, 3])
```

`random.RevWeibull` *Draw a random sample from an RevWeibull distribution*

Description

Draw a random sample from an RevWeibull distribution

Usage

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

Arguments

`x`	A `RevWeibull` object created by a call to `RevWeibull()`.
`n`	The number of samples to draw. Defaults to `1L`.
`drop`	logical. Should the result be simplified to a vector if possible?
`...`	Unused. Unevaluated arguments will generate a warning to catch mispellings or other possible errors.

Value

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`).

Examples

```
set.seed(27)

X <- RevWeibull(1, 2)
X

random(X, 10)

pdf(X, 0.7)
log_pdf(X, 0.7)

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

cdf(X, quantile(X, 0.7))
quantile(X, cdf(X, 0.7))
```

`random.SinhArcsinh` *Draw a random sample from a Sinh-Arcsinh distribution*

Description

Please see the documentation of `SinhArcsinh()` for some properties of the Sinh-Arcsinh distribution, as well as extensive examples showing to how calculate p-values and confidence intervals.

Usage

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

Arguments

`x`	A `SinhArcsinh` object created by a call to `SinhArcsinh()`.
`n`	The number of samples to draw. Defaults to `1L`.
`drop`	logical. Should the result be simplified to a vector if possible?
`cores`	`NULL` or positive integer. TODO(R): Just a development option. If not `NULL` we use the C code with `cores` threads.
`...`	Unused. Unevaluated arguments will generate a warning to catch mispellings or other possible errors.

Value

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`).

Examples

```
## SinhArcsinh() by default uses nu = 1, tau = 1 which
## results in the standard normal distribution
set.seed(6020)
X <- SinhArcsinh() # Uses mu = 1, sigma = 0, nu = 1, tau = 1)
x <- random(X, 300)
qqnorm(x); qqline(x, col = 2, lwd = 2)
curve(pdf(X, x), xlim = c(-5, 5), main = paste(X, "density"))

## Calculation of central moments is based on numeric integration,
## thus not being identical to the standard normal distribution
c(mean = mean(x), sd = sd(x))

## Skewed Sinh-Arcsinh distribution
X <- SinhArcsinh(mu = 7, sigma = 2, nu = c(0.7, 1, 0.7), tau = c(1, 0.7, 0.7))
as.matrix(X)

## Visualization of density functions using different parameters for nu/tau
curve(pdf(X[1], x), xlim = c(0, 20), ylim = c(0, 0.2), main = "Density function")
curve(pdf(X[2], x), xlim = c(0, 20), col = 2, add = TRUE)
curve(pdf(X[3], x), xlim = c(0, 20), col = 4, add = TRUE)

## Visualization of distribution function using different parameters for nu/tau
curve(cdf(X[1], x), xlim = c(0, 20), ylim = 0:1, main = "Distribution function")
curve(cdf(X[2], x), xlim = c(0, 20), col = 2, add = TRUE)
curve(cdf(X[3], x), xlim = c(0, 20), col = 4, add = TRUE)

## Central moments
mean(X)
variance(X)
skewness(X)
kurtosis(X)

## Drawing random values
random(X, 10)

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

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

# note that the cdf() and quantile() functions are inverses
X <- SinhArcsinh(mu = 3, sigma = 2, nu = 0.9, tau = 1.2)
cdf(X, quantile(X, 0.7))
quantile(X, cdf(X, 7))
```

`random.StudentsT` *Draw a random sample from a StudentsT distribution*

Description

Please see the documentation of `StudentsT()` for some properties of the T distribution, as well as extensive examples showing to how calculate p-values and confidence intervals.

Usage

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

Arguments

`x`	A `StudentsT` object created by a call to `StudentsT()`.
`n`	The number of samples to draw. Defaults to `1L`.
`drop`	logical. Should the result be simplified to a vector if possible?
`...`	Unused. Unevaluated arguments will generate a warning to catch mispellings or other possible errors.

Value

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`).

See Also

Other StudentsT distribution: `cdf.StudentsT()`, `pdf.StudentsT()`, `quantile.StudentsT()`

Examples

```
set.seed(27)

X <- StudentsT(3)
X

random(X, 10)

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

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

### example: calculating p-values for two-sided T-test

# here the null hypothesis is H_0: mu = 3

# data to test
x <- c(3, 7, 11, 0, 7, 0, 4, 5, 6, 2)
nx <- length(x)

# calculate the T-statistic
```

```
t_stat <- (mean(x) - 3) / (sd(x) / sqrt(nx))
t_stat

# null distribution of statistic depends on sample size!
T <- StudentsT(df = nx - 1)

# calculate the two-sided p-value
1 - cdf(T, abs(t_stat)) + cdf(T, -abs(t_stat))

# exactly equivalent to the above
2 * cdf(T, -abs(t_stat))

# p-value for one-sided test
# H_0: mu <= 3   vs   H_A: mu > 3
1 - cdf(T, t_stat)

# p-value for one-sided test
# H_0: mu >= 3   vs   H_A: mu < 3
cdf(T, t_stat)

### example: calculating a 88 percent T CI for a mean

# lower-bound
mean(x) - quantile(T, 1 - 0.12 / 2) * sd(x) / sqrt(nx)

# upper-bound
mean(x) + quantile(T, 1 - 0.12 / 2) * sd(x) / sqrt(nx)

# equivalent to
mean(x) + c(-1, 1) * quantile(T, 1 - 0.12 / 2) * sd(x) / sqrt(nx)

# also equivalent to
mean(x) + quantile(T, 0.12 / 2) * sd(x) / sqrt(nx)
mean(x) + quantile(T, 1 - 0.12 / 2) * sd(x) / sqrt(nx)
```

`random.Tukey` *Draw a random sample from a Tukey distribution*

Description

Draw a random sample from a Tukey distribution

Usage

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

Arguments

x	A Tukey object created by a call to Tukey().
n	The number of samples to draw. Defaults to 1L.
drop	logical. Should the result be simplified to a vector if possible?
...	Unused. Unevaluated arguments will generate a warning to catch mispellings or other possible errors.

Value

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).

Examples

```
set.seed(27)

X <- Tukey(4L, 16L, 2L)
X

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

---

random.Uniform *Draw a random sample from a continuous Uniform distribution*

Description

Draw a random sample from a continuous Uniform distribution

Usage

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

Arguments

x	A Uniform object created by a call to Uniform().
n	The number of samples to draw. Defaults to 1L.
drop	logical. Should the result be simplified to a vector if possible?
...	Unused. Unevaluated arguments will generate a warning to catch mispellings or other possible errors.

Value

Values in [a, b]. 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).

Examples

```
set.seed(27)

X <- Uniform(1, 2)
X

random(X, 10)

pdf(X, 0.7)
log_pdf(X, 0.7)

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

cdf(X, quantile(X, 0.7))
quantile(X, cdf(X, 0.7))
```

---

random.Weibull *Draw a random sample from a Weibull distribution*

Description

Draw a random sample from a Weibull distribution

Usage

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

Arguments

x	A Weibull object created by a call to Weibull().
n	The number of samples to draw. Defaults to 1L.
drop	logical. Should the result be simplified to a vector if possible?
...	Unused. Unevaluated arguments will generate a warning to catch mispellings or other possible errors.

Value

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).

See Also

Other Weibull distribution: cdf.Weibull(), pdf.Weibull(), quantile.Weibull()

Examples

```
set.seed(27)

X <- Weibull(0.3, 2)
X

random(X, 10)

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

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

`random.ZINegativeBinomial` *Draw a random sample from a zero-inflated negative binomial distribution*

Description

Draw a random sample from a zero-inflated negative binomial distribution

Usage

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

Arguments

`x`	A `ZINegativeBinomial` object created by a call to `ZINegativeBinomial()`.
`n`	The number of samples to draw. Defaults to `1L`.
`drop`	logical. Should the result be simplified to a vector if possible?
`...`	Unused. Unevaluated arguments will generate a warning to catch mispellings or other possible errors.

Value

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`).

Examples

```
## set up a zero-inflated negative binomial distribution
X <- ZINegativeBinomial(mu = 2.5, theta = 1, pi = 0.25)
X

## standard functions
```

```
pdf(X, 0:8)
cdf(X, 0:8)
quantile(X, seq(0, 1, by = 0.25))

## cdf() and quantile() are inverses for each other
quantile(X, cdf(X, 3))

## density visualization
plot(0:8, pdf(X, 0:8), type = "h", lwd = 2)

## corresponding sample with histogram of empirical frequencies
set.seed(0)
x <- random(X, 500)
hist(x, breaks = -1:max(x) + 0.5)
```

`random.ZIPoisson` *Draw a random sample from a zero-inflated Poisson distribution*

Description

Draw a random sample from a zero-inflated Poisson distribution

Usage

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

Arguments

`x`	A `ZIPoisson` object created by a call to `ZIPoisson()`.
`n`	The number of samples to draw. Defaults to `1L`.
`drop`	logical. Should the result be simplified to a vector if possible?
`...`	Unused. Unevaluated arguments will generate a warning to catch mispellings or other possible errors.

Value

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`).

Examples

```
## set up a zero-inflated Poisson distribution
X <- ZIPoisson(lambda = 2.5, pi = 0.25)
X

## standard functions
pdf(X, 0:8)
```

```
cdf(X, 0:8)
quantile(X, seq(0, 1, by = 0.25))

## cdf() and quantile() are inverses for each other
quantile(X, cdf(X, 3))

## density visualization
plot(0:8, pdf(X, 0:8), type = "h", lwd = 2)

## corresponding sample with histogram of empirical frequencies
set.seed(0)
x <- random(X, 500)
hist(x, breaks = -1:max(x) + 0.5)
```

`random.ZTNegativeBinomial` *Draw a random sample from a zero-truncated negative binomial distribution*

Description

Draw a random sample from a zero-truncated negative binomial distribution

Usage

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

Arguments

`x`	A `ZTNegativeBinomial` object created by a call to `ZTNegativeBinomial()`.
`n`	The number of samples to draw. Defaults to `1L`.
`drop`	logical. Should the result be simplified to a vector if possible?
`...`	Unused. Unevaluated arguments will generate a warning to catch mispellings or other possible errors.

Value

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`).

Examples

```
## set up a zero-truncated negative binomial distribution
X <- ZTNegativeBinomial(mu = 2.5, theta = 1)
X

## standard functions
```

```
pdf(X, 0:8)
cdf(X, 0:8)
quantile(X, seq(0, 1, by = 0.25))

## cdf() and quantile() are inverses for each other
quantile(X, cdf(X, 3))

## density visualization
plot(0:8, pdf(X, 0:8), type = "h", lwd = 2)

## corresponding sample with histogram of empirical frequencies
set.seed(0)
x <- random(X, 500)
hist(x, breaks = -1:max(x) + 0.5)
```

`random.ZTPoisson` *Draw a random sample from a zero-truncated Poisson distribution*

Description

Draw a random sample from a zero-truncated Poisson distribution

Usage

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

Arguments

`x`	A `ZTPoisson` object created by a call to `ZTPoisson()`.
`n`	The number of samples to draw. Defaults to `1L`.
`drop`	logical. Should the result be simplified to a vector if possible?
`...`	Unused. Unevaluated arguments will generate a warning to catch mispellings or other possible errors.

Value

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`).

Examples

```
## set up a zero-truncated Poisson distribution
X <- ZTPoisson(lambda = 2.5)
X

## standard functions
pdf(X, 0:8)
```

```
cdf(X, 0:8)
quantile(X, seq(0, 1, by = 0.25))

## cdf() and quantile() are inverses for each other
quantile(X, cdf(X, 3))

## density visualization
plot(0:8, pdf(X, 0:8), type = "h", lwd = 2)

## corresponding sample with histogram of empirical frequencies
set.seed(0)
x <- random(X, 500)
hist(x, breaks = -1:max(x) + 0.5)
```

`RevWeibull` *Create a reversed Weibull distribution*

Description

The reversed (or negated) Weibull distribution is a special case of the `\link{GEV}` distribution, obtained when the GEV shape parameter ξ is negative. It may be referred to as a type III extreme value distribution.

Usage

```
RevWeibull(location = 0, scale = 1, shape = 1)
```

Arguments

`location`	The location (maximum) parameter m. `location` can be any real number. Defaults to `0`.
`scale`	The scale parameter s. `scale` can be any positive number. Defaults to `1`.
`shape`	The scale parameter α. `shape` can be any positive number. Defaults to `1`.

Details

We recommend reading this documentation on https://zeileis.github.io/distributions3/, where the math will render with additional detail and much greater clarity.

In the following, let X be a reversed Weibull random variable with location parameter `location` = m, scale parameter `scale` = s, and shape parameter `shape` = α. An RevWeibull(m, s, α) distribution is equivalent to a `\link{GEV}`$(m - s, s/\alpha, -1/\alpha)$ distribution.

If X has an RevWeibull(m, λ, k) distribution then $m - X$ has a `\link{Weibull}`(k, λ) distribution, that is, a Weibull distribution with shape parameter k and scale parameter λ.

Support: $(-\infty, m)$.

Mean: $m + s\Gamma(1 + 1/\alpha)$.

Median: $m + s(\ln 2)^{1/\alpha}$.

Variance: $s^2[\Gamma(1+2/\alpha) - \Gamma(1+1/\alpha)^2]$.

Probability density function (p.d.f):

$$f(x) = \alpha s^{-1}[-(x-m)/s]^{\alpha-1} \exp\{-[-(x-m)/s]^{\alpha}\}$$

for $x < m$. The p.d.f. is 0 for $x \geq m$.

Cumulative distribution function (c.d.f):

$$F(x) = \exp\{-[-(x-m)/s]^{\alpha}\}$$

for $x < m$. The c.d.f. is 1 for $x \geq m$.

Value

A `RevWeibull` object.

See Also

Other continuous distributions: `Beta()`, `Cauchy()`, `ChiSquare()`, `Erlang()`, `Exponential()`, `FisherF()`, `Frechet()`, `GEV()`, `GP()`, `Gamma()`, `Gumbel()`, `LogNormal()`, `Logistic()`, `Normal()`, `SinhArcsinh()`, `StudentsT()`, `Tukey()`, `Uniform()`, `Weibull()`

Examples

```
set.seed(27)

X <- RevWeibull(1, 2)
X

random(X, 10)

pdf(X, 0.7)
log_pdf(X, 0.7)

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

cdf(X, quantile(X, 0.7))
quantile(X, cdf(X, 0.7))
```

---

score-hessian *Generic functions and methods for computing score and Hessian*

Description

The generic functions `score` and `hessian` along with the corresponding methods for distribution objects enable the computation of the score (first derivative of the log-likelihood with respect to the parameters) and Hessian (corresponding second derivative).

Usage

```
score(d, ...)

hessian(d, ...)

## S3 method for class 'distribution'
score(d, x, which = NULL, drop = TRUE, eps = .Machine$double.eps^(1/3), ...)

## S3 method for class 'distribution'
hessian(
  d,
  x,
  which = NULL,
  drop = TRUE,
  expected = FALSE,
  eps = .Machine$double.eps^(1/4),
  ...
)
```

Arguments

d	An object, typically a distribution object, e.g., as created by Normal() or Binomial().
x	A vector of elements whose score/Hessian should be determined given the distribution d. Either d and x need to have the same length or length 1.
which	character or NULL (default). Character labels for the derivatives to be included in the score or Hessian respectively. In score the possible values are (combinations of) the parameter names (e.g., "mu" and/or "sigma"). In hessian additionally the cross-derivatives (e.g., mu:sigma and sigma:mu) are available. By default (if which = NULL) all elements of the score or Hessian should be computed.
drop	logical. Should the result be simplified to a vector if possible?
...	Arguments passed to methods.
expected	logical. Should the expected Hessian be computed? If FALSE the observed Hessian is computed. Some methods might only support only one or the the other option and defaults might differ.
eps	numeric. Tolerance when obtaining the score or Hessian via numeric differentiation.

Details

In the methods for dedicated distributions analytical results for computing the score or Hessian should be used. All hessian methods should implemented the expected argument with an appropriate default. If a hessian method only supports the expected or the observed Hessian, then an error should be issued if needed.

In the fallback methods for general distribution objects a simple differencing approach is used. Either differences of log_pdf (for the score) or of score (for the hessian) with slightly modified

parameters are used. If the parameters in a distribution are on the boundary of the parameter space this might lead to errors in the computation.

Value

Either a numeric matrix with suitable column names is returned or a numeric vector (provided that `which` has length 1 and `drop = TRUE`).

Examples

```
X <- Normal(mu = c(0, 1, 2), sigma = c(2, 1, 1))
x <- c(0, 0, 1)
score(X, x)
hessian(X, x)
hessian(X, x, expected = TRUE)

h <- hessian(X[1], x[1], expected = TRUE)
matrix(h, ncol = 2, dimnames = list(c("mu", "sigma"), c("mu", "sigma")))

## Comparison of analytic and numeric score/Hessian (Normal(3, 2))
X <- Normal(mu = 3, sigma = 2)
x <- seq(0, 6, by = 0.01)

#' ## score:   derivative of log-likelihood by parameter sigma
s_analytic <- score(X, x, which = "sigma")
s_numeric  <- distributions3:::score.distribution(X, x, which = "sigma")
message("Sum of absolute differences (score): ", sum(abs(s_analytic - s_numeric)))

matplot(x, cbind(s_analytic, s_numeric), col = 1:2, type = "l", lty = 1:2,
        lwd = 3, xlab = "x", main = "score - analytic vs. numeric solution",
        ylab = expression(partialdiff * l(x) / partialdiff * sigma))
legend("topleft", legend = c("analytic score", "numeric score"),
       bty = "n", pch = NA, lty = 1:2, col = 1:2, lwd = 3)
abline(h = 0, v = 3, lty = 3)

#' ## Hessian: second derivative of log-likelihood by sigma^2
h_analytic <- hessian(X, x, which = "sigma")
h_numeric  <- distributions3:::hessian.distribution(X, x, which = "sigma")
message("Sum of absolute differences (Hessian): ", sum(abs(h_analytic - h_numeric)))

matplot(x, cbind(h_analytic, h_numeric), col = 1:2, type = "l", lty = 1:2,
        lwd = 3, xlab = "x", main = "Hessian - analytic vs. numeric solution",
        ylab = expression(partialdiff^2 * l(x) / partialdiff * sigma^2))
legend("topleft", legend = c("analytic score", "numeric score"),
       bty = "n", pch = NA, lty = 1:2, col = 1:2, lwd = 3)
abline(v = 3, lty = 3)
```

simulate.default *Simulate responses from fitted model objects*

Description

Default method for simulating new responses from any model object with a prodist method (for extracting a probability distribution object).

Usage

```
## Default S3 method:
simulate(object, nsim = 1, seed = NULL, ...)
```

Arguments

object	An object for which a prodist method is available.
nsim	The number of response vectors to simulate. Should be a positive integer. Defaults to 1.
seed	An optional random seed that is to be set using set.seed prior to drawing the random sample. The previous random seed from the global environment (if any) is restored afterwards.
...	Arguments passed to simulate.distribution.

Details

This default method simply combines two building blocks provided in this package: (1) prodist for extracting the probability distribution from a fitted model object, (2) simulate.distribution for simulating new observations from this distribution (internally calling random).

Thus, this enables simulation from any fitted model object that provides a prodist method. It waives the need to implement a dedicated simulate method for this model class.

Value

A data frame with an attribute "seed" containing the .Random.seed from before the simulation.

Examples

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

## simulate new goals via glm method
set.seed(0)
g_glm <- simulate(m, n = 3)

## alternatively use the new default method
set.seed(0)
```

```
g_default <- simulate.default(m, n = 3)

## same results
all.equal(g_glm, g_default, check.attributes = FALSE)
```

`SinhArcsinh` *Create a Sinh-Arcsinh (SHASH) distribution*

Description

The Sinh-Arcsinh (SHASH) distribution is a four-parameter distribution with support on the real space that generalizes the Normal distribution by separately controlling location, scale, skewness, and tail-heaviness. It can produce a wide range of shapes, including symmetric and asymmetric forms and both heavier and lighter tails, making it useful for modeling real-valued data with non-normal behavior. Using $\nu = 1$ and $\tau = 1$ results in a standard normal distribution.

Usage

```
SinhArcsinh(mu = 0, sigma = 1, nu = 1, tau = 1)
```

Arguments

`mu`	The location parameter, written μ in textbooks. Defaults to `0`.
`sigma`	The scale parameter, written σ in textbooks. Can be any positive number. Defaults to `1`.
`nu`	The skewness parameter ν, defaults to `1`.
`tau`	The kurtosis parameter τ, defaults to `1`.

Details

We recommend reading this documentation on https://zeileis.github.io/distributions3/, where the math will render with additional detail and much greater clarity.

In the following, let X be a Sinh-Arcsinh random variable with mean `mu` = μ, `sigma` = σ, `nu` = ν, and `tau` = τ.

Support: R, the set of all real numbers

Probability density function (p.d.f):

$$f(t) = \frac{1}{\sigma\sqrt{1+z^2}} \cdot \frac{1}{2}\left(\tau e^{\tau w} + \nu e^{-\nu w}\right) \cdot \phi\left(\frac{1}{2}\left(e^{\tau w} - e^{-\nu w}\right)\right)$$

$$\text{where } z = \frac{t-\mu}{\sigma},\ w = \operatorname{asinh}(z),\ \text{and } \phi() \text{ is the standard normal PDF.}$$

Cumulative distribution function (c.d.f):

$$F(t) = \Phi\left(H\left(\operatorname{asinh}\left(\frac{t-\mu}{\sigma}\right)\right)\right)$$

$$\text{where } H(w) = \frac{1}{2}\left(e^{\tau w} - e^{-\nu w}\right), \text{ and } \Phi() \text{ the standard normal CDF.}$$

Value

A SinhArcsinh object.

References

Jones MC, Pewsey A (2009). "Sinh-Arcsinh Distributions", *Journal of Statistical Software*, **96**(4), 761–780. doi:10.1093/biomet/asp053

See Also

Other continuous distributions: Beta(), Cauchy(), ChiSquare(), Erlang(), Exponential(), FisherF(), Frechet(), GEV(), GP(), Gamma(), Gumbel(), LogNormal(), Logistic(), Normal(), RevWeibull(), StudentsT(), Tukey(), Uniform(), Weibull()

Examples

```
## SinhArcsinh() by default uses nu = 1, tau = 1 which
## results in the standard normal distribution
set.seed(6020)
X <- SinhArcsinh() # Uses mu = 1, sigma = 0, nu = 1, tau = 1)
x <- random(X, 300)
qqnorm(x); qqline(x, col = 2, lwd = 2)
curve(pdf(X, x), xlim = c(-5, 5), main = paste(X, "density"))

## Calculation of central moments is based on numeric integration,
## thus not being identical to the standard normal distribution
c(mean = mean(x), sd = sd(x))

## Skewed Sinh-Arcsinh distribution
X <- SinhArcsinh(mu = 7, sigma = 2, nu = c(0.7, 1, 0.7), tau = c(1, 0.7, 0.7))
as.matrix(X)

## Visualization of density functions using different parameters for nu/tau
curve(pdf(X[1], x), xlim = c(0, 20), ylim = c(0, 0.2), main = "Density function")
curve(pdf(X[2], x), xlim = c(0, 20), col = 2, add = TRUE)
curve(pdf(X[3], x), xlim = c(0, 20), col = 4, add = TRUE)

## Visualization of distribution function using different parameters for nu/tau
curve(cdf(X[1], x), xlim = c(0, 20), ylim = 0:1, main = "Distribution function")
curve(cdf(X[2], x), xlim = c(0, 20), col = 2, add = TRUE)
curve(cdf(X[3], x), xlim = c(0, 20), col = 4, add = TRUE)

## Central moments
mean(X)
variance(X)
skewness(X)
kurtosis(X)
```

```
## Drawing random values
random(X, 10)

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

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

# note that the cdf() and quantile() functions are inverses
X <- SinhArcsinh(mu = 3, sigma = 2, nu = 0.9, tau = 1.2)
cdf(X, quantile(X, 0.7))
quantile(X, cdf(X, 7))
```

---

stat_auc *Fill out area under the curve for a plotted PDF*

Description

Fill out area under the curve for a plotted PDF

Usage

```
stat_auc(
  mapping = NULL,
  data = NULL,
  geom = "auc",
  position = "identity",
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE,
  from = -Inf,
  to = Inf,
  annotate = FALSE,
  digits = 3,
  ...
)

geom_auc(
  mapping = NULL,
  data = NULL,
  stat = "auc",
  position = "identity",
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE,
  from = -Inf,
```

```
  to = Inf,
  annotate = FALSE,
  digits = 3,
  ...
)
```

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().

from — Left end-point of interval

to — Right end-point of interval

annotate — Should P() be added in the upper left corner as an annotation? Works also with a colour character, e.g., "red".

digits — Number of digits shown in annotation

... — 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.

Examples

```
N1 <- Normal()
plot_pdf(N1) + geom_auc(to = -0.645)
plot_pdf(N1) + geom_auc(from = -0.645, to = 0.1, annotate = TRUE)
```

```
N2 <- Normal(0, c(1, 2))
plot_pdf(N2) + geom_auc(to = 0)
plot_pdf(N2) + geom_auc(from = -2, to = 2, annotate = TRUE)
```

`StudentsT` *Create a Student's T distribution*

Description

The Student's T distribution is closely related to the `Normal()` distribution, but has heavier tails. As ν increases to ∞, the Student's T converges to a Normal. The T distribution appears repeatedly throughout classic frequentist hypothesis testing when comparing group means.

Usage

```
StudentsT(df = numeric())
```

Arguments

`df`	Degrees of freedom. Can be any positive number. Often called ν in textbooks.

Details

We recommend reading this documentation on https://zeileis.github.io/distributions3/, where the math will render with additional detail and much greater clarity.

In the following, let X be a Students T random variable with `df` = ν.

Support: R, the set of all real numbers

Mean: Undefined unless $\nu \geq 2$, in which case the mean is zero.

Variance:

$$\frac{\nu}{\nu - 2}$$

Undefined if $\nu < 1$, infinite when $1 < \nu \leq 2$.

Probability density function (p.d.f):

$$f(x) = \frac{\Gamma(\frac{\nu+1}{2})}{\sqrt{\nu\pi}\Gamma(\frac{\nu}{2})}(1 + \frac{x^2}{\nu})^{-\frac{\nu+1}{2}}$$

Cumulative distribution function (c.d.f):

Nasty, omitted.

Moment generating function (m.g.f):

Undefined.

Value

A StudentsT object.

See Also

Other continuous distributions: Beta(), Cauchy(), ChiSquare(), Erlang(), Exponential(), FisherF(), Frechet(), GEV(), GP(), Gamma(), Gumbel(), LogNormal(), Logistic(), Normal(), RevWeibull(), SinhArcsinh(), Tukey(), Uniform(), Weibull()

Examples

```
set.seed(27)

X <- StudentsT(3)
X

random(X, 10)

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

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

### example: calculating p-values for two-sided T-test

# here the null hypothesis is H_0: mu = 3

# data to test
x <- c(3, 7, 11, 0, 7, 0, 4, 5, 6, 2)
nx <- length(x)

# calculate the T-statistic
t_stat <- (mean(x) - 3) / (sd(x) / sqrt(nx))
t_stat

# null distribution of statistic depends on sample size!
T <- StudentsT(df = nx - 1)

# calculate the two-sided p-value
1 - cdf(T, abs(t_stat)) + cdf(T, -abs(t_stat))

# exactly equivalent to the above
2 * cdf(T, -abs(t_stat))

# p-value for one-sided test
# H_0: mu <= 3   vs   H_A: mu > 3
1 - cdf(T, t_stat)

# p-value for one-sided test
# H_0: mu >= 3   vs   H_A: mu < 3
cdf(T, t_stat)
```

```
### example: calculating a 88 percent T CI for a mean

# lower-bound
mean(x) - quantile(T, 1 - 0.12 / 2) * sd(x) / sqrt(nx)

# upper-bound
mean(x) + quantile(T, 1 - 0.12 / 2) * sd(x) / sqrt(nx)

# equivalent to
mean(x) + c(-1, 1) * quantile(T, 1 - 0.12 / 2) * sd(x) / sqrt(nx)

# also equivalent to
mean(x) + quantile(T, 0.12 / 2) * sd(x) / sqrt(nx)
mean(x) + quantile(T, 1 - 0.12 / 2) * sd(x) / sqrt(nx)
```

`suff_stat` *Compute the sufficient statistics of a distribution from data*

Description

Generic function for computing the sufficient statistics of a distribution based on empirical data.

Usage

```
suff_stat(d, x, ...)
```

Arguments

`d`	An object. The package provides methods for distribution objects such as those from `Normal()` or `Binomial()` etc.
`x`	A vector of data to compute the likelihood.
`...`	Arguments passed to methods. Unevaluated arguments will generate a warning to catch mispellings or other possible errors.

Value

a named list of sufficient statistics

Examples

```
X <- Normal()
suff_stat(X, c(-1, 0, 0, 0, 3))
```

suff_stat.Bernoulli *Compute the sufficient statistics for a Bernoulli distribution from data*

Description

Compute the sufficient statistics for a Bernoulli distribution from data

Usage

```
## S3 method for class 'Bernoulli'
suff_stat(d, x, ...)
```

Arguments

d	A Bernoulli object.
x	A vector of zeroes and ones.
...	Unused.

Value

A named list of the sufficient statistics of the Bernoulli distribution:

- successes: The number of successful trials (sum(x == 1))
- failures: The number of failed trials (sum(x == 0)).

suff_stat.Binomial *Compute the sufficient statistics for the Binomial distribution from data*

Description

Compute the sufficient statistics for the Binomial distribution from data

Usage

```
## S3 method for class 'Binomial'
suff_stat(d, x, ...)
```

Arguments

d	A Binomial object.
x	A vector of zeroes and ones.
...	Unused.

Value

A named list of the sufficient statistics of the Binomial distribution:

- `successes`: The total number of successful trials.
- `experiments`: The number of experiments run.
- `trials`: The number of trials run per experiment.

`suff_stat.Exponential` *Compute the sufficient statistics of an Exponential distribution from data*

Description

Compute the sufficient statistics of an Exponential distribution from data

Usage

```
## S3 method for class 'Exponential'
suff_stat(d, x, ...)
```

Arguments

`d`	An `Exponential` object created by a call to `Exponential()`.
`x`	A vector of data.
`...`	Unused.

Value

A named list of the sufficient statistics of the exponential distribution:

- `sum`: The sum of the observations.
- `samples`: The number of observations.

suff_stat.Gamma *Compute the sufficient statistics for a Gamma distribution from data*

Description

- `sum`: The sum of the data.
- `log_sum`: The log of the sum of the data.
- `samples`: The number of samples in the data.

Usage

```
## S3 method for class 'Gamma'
suff_stat(d, x, ...)
```

Arguments

`d`	A `Gamma` object created by a call to `Gamma()`.
`x`	A vector to fit the Gamma distribution to.
`...`	Unused.

Value

a `Gamma` object

suff_stat.Geometric *Compute the sufficient statistics for the Geometric distribution from data*

Description

Compute the sufficient statistics for the Geometric distribution from data

Usage

```
## S3 method for class 'Geometric'
suff_stat(d, x, ...)
```

Arguments

`d`	A `Geometric` object.
`x`	A vector of zeroes and ones.
`...`	Unused.

Value

A named list of the sufficient statistics of the Geometric distribution:

- `trials`: The total number of trials ran until the first success.
- `experiments`: The number of experiments run.

`suff_stat.LogNormal` *Compute the sufficient statistics for a Log-normal distribution from data*

Description

Compute the sufficient statistics for a Log-normal distribution from data

Usage

```
## S3 method for class 'LogNormal'
suff_stat(d, x, ...)
```

Arguments

`d`	A `LogNormal` object created by a call to `LogNormal()`.
`x`	A vector of data.
`...`	Unused.

Value

A named list of the sufficient statistics of the normal distribution:

- `mu`: The sample mean of the log of the data.
- `sigma`: The sample standard deviation of the log of the data.
- `samples`: The number of samples in the data.

suff_stat.Normal *Compute the sufficient statistics for a Normal distribution from data*

Description

Compute the sufficient statistics for a Normal distribution from data

Usage

```
## S3 method for class 'Normal'
suff_stat(d, x, ...)
```

Arguments

d	A Normal object created by a call to Normal().
x	A vector of data.
...	Unused.

Value

A named list of the sufficient statistics of the normal distribution:

- mu: The sample mean of the data.
- sigma: The sample standard deviation of the data.
- samples: The number of samples in the data.

suff_stat.Poisson *Compute the sufficient statistics of an Poisson distribution from data*

Description

Compute the sufficient statistics of an Poisson distribution from data

Usage

```
## S3 method for class 'Poisson'
suff_stat(d, x, ...)
```

Arguments

d	An Poisson object created by a call to Poisson().
x	A vector of data.
...	Unused.

Value

A named list of the sufficient statistics of the Poisson distribution:

- `sum`: The sum of the data.
- `samples`: The number of samples in the data.

`support` *Return the support of a distribution*

Description

Generic function for computing the support interval (minimum and maximum) for a given probability distribution object.

Usage

```
support(d, drop = TRUE, ...)
```

Arguments

`d`	An object. The package provides methods for distribution objects such as those from `Normal()` or `Binomial()` etc.
`drop`	logical. Should the result be simplified to a vector if possible?
`...`	Arguments passed to methods. Unevaluated arguments will generate a warning to catch mispellings or other possible errors.

Value

A vector (or matrix) with two elements (or columns) indicating the range (minimum and maximum) of the support.

Examples

```
X <- Normal()
support(X)
Y <- Uniform(-1, 1:3)
support(Y)
```

support.Bernoulli *Return the support of the Bernoulli distribution*

Description

Return the support of the Bernoulli distribution

Usage

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

Arguments

d	An Bernoulli object created by a call to Bernoulli().
drop	logical. Should the result be simplified to a vector if possible?
...	Currently not used.

Value

A vector of length 2 with the minimum and maximum value of the support.

support.Beta *Return the support of the Beta distribution*

Description

Return the support of the Beta distribution

Usage

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

Arguments

d	An Beta object created by a call to Beta().
drop	logical. Should the result be simplified to a vector if possible?
...	Currently not used.

Value

A vector of length 2 with the minimum and maximum value of the support.

support.Binomial *Return the support of the Binomial distribution*

Description

Return the support of the Binomial distribution

Usage

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

Arguments

d	An Binomial object created by a call to Binomial().
drop	logical. Shoul the result be simplified to a vector if possible?
...	Currently not used.

Value

A vector of length 2 with the minimum and maximum value of the support.

support.Cauchy *Return the support of the Cauchy distribution*

Description

Return the support of the Cauchy distribution

Usage

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

Arguments

d	An Cauchy object created by a call to Cauchy().
drop	logical. Should the result be simplified to a vector if possible?
...	Currently not used.

Value

A vector of length 2 with the minimum and maximum value of the support.

support.ChiSquare *Return the support of the ChiSquare distribution*

Description

Return the support of the ChiSquare distribution

Usage

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

Arguments

d	An ChiSquare object created by a call to ChiSquare().
drop	logical. Should the result be simplified to a vector if possible?
...	Currently not used.

Value

A vector of length 2 with the minimum and maximum value of the support.

support.Empirical *Return the support of the Empirical distribution*

Description

Though the support of an empirical distribution is defined by its unique numeric values this function returns only the range, i.e., the lowest (minimum) and highest (maximum) observation as the outer bounds of the support.

Usage

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

Arguments

d	an Empirical object created by a call to Empirical().
drop	logical. Should the result be simplified to a vector if possible?
...	currently not used.

Value

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.

See Also

Other Empirical distribution: `Empirical()`, `cdf.Empirical()`, `dempirical()`, `pdf.Empirical()`, `quantile.Empirical()`, `random.Empirical()`

`support.Erlang` *Return the support of the Erlang distribution*

Description

Return the support of the Erlang distribution

Usage

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

Arguments

`d`	An `Erlang` object created by a call to `Erlang()`.
`drop`	logical. Should the result be simplified to a vector if possible?
`...`	Currently not used.

Value

A vector of length 2 with the minimum and maximum value of the support.

`support.Exponential` *Return the support of the Exponential distribution*

Description

Return the support of the Exponential distribution

Usage

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

Arguments

d	An Exponential object created by a call to Exponential().
drop	logical. Should the result be simplified to a vector if possible?
...	Currently not used.

Value

A vector of length 2 with the minimum and maximum value of the support.

---

support.FisherF *Return the support of the FisherF distribution*

Description

Return the support of the FisherF distribution

Usage

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

Arguments

d	An FisherF object created by a call to FisherF().
drop	logical. Should the result be simplified to a vector if possible?
...	Currently not used.

Value

A vector of length 2 with the minimum and maximum value of the support.

---

support.Frechet *Return the support of the Frechet distribution*

Description

Return the support of the Frechet distribution

Usage

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

Arguments

d	An Frechet object created by a call to Frechet().
drop	logical. Should the result be simplified to a vector if possible?
...	Currently not used.

Value

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.

---

support.Gamma *Return the support of the Gamma distribution*

Description

Return the support of the Gamma distribution

Usage

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

Arguments

d	An Gamma object created by a call to Gamma().
drop	logical. Should the result be simplified to a vector if possible?
...	Currently not used.

Value

A vector of length 2 with the minimum and maximum value of the support.

support.Geometric *Return the support of the Geometric distribution*

Description

Return the support of the Geometric distribution

Usage

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

Arguments

d	An Geometric object created by a call to Geometric().
drop	logical. Should the result be simplified to a vector if possible?
...	Currently not used.

Value

A vector of length 2 with the minimum and maximum value of the support.

support.GEV *Return the support of a GEV distribution*

Description

Return the support of a GEV distribution

Usage

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

Arguments

d	An GEV object created by a call to GEV().
drop	logical. Should the result be simplified to a vector if possible?
...	Currently not used.

Value

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.

support.GP *Return the support of the GP distribution*

Description

Return the support of the GP distribution

Usage

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

Arguments

d	An GP object created by a call to GP().
drop	logical. Should the result be simplified to a vector if possible?
...	Currently not used.

Value

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.

support.Gumbel *Return the support of the Gumbel distribution*

Description

Return the support of the Gumbel distribution

Usage

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

Arguments

d	An Gumbel object created by a call to Gumbel().
drop	logical. Should the result be simplified to a vector if possible?
...	Currently not used.

Value

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.

`support.HurdleNegativeBinomial` *Return the support of the hurdle negative binomial distribution*

Description

Return the support of the hurdle negative binomial distribution

Usage

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

Arguments

`d`	An `HurdleNegativeBinomial` object created by a call to `HurdleNegativeBinomial()`.
`drop`	logical. Should the result be simplified to a vector if possible?
`...`	Currently not used.

Value

A vector of length 2 with the minimum and maximum value of the support.

`support.HurdlePoisson` *Return the support of the hurdle Poisson distribution*

Description

Return the support of the hurdle Poisson distribution

Usage

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

Arguments

`d`	An `HurdlePoisson` object created by a call to `HurdlePoisson()`.
`drop`	logical. Should the result be simplified to a vector if possible?
`...`	Currently not used.

Value

A vector of length 2 with the minimum and maximum value of the support.

support.HyperGeometric *Return the support of the HyperGeometric distribution*

Description

Return the support of the HyperGeometric distribution

Usage

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

Arguments

d	An HyperGeometric object created by a call to HyperGeometric().
drop	logical. Should the result be simplified to a vector if possible?
...	Currently not used.

Value

A vector of length 2 with the minimum and maximum value of the support.

support.Logistic *Return the support of the Logistic distribution*

Description

Return the support of the Logistic distribution

Usage

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

Arguments

d	An Logistic object created by a call to Logistic().
drop	logical. Should the result be simplified to a vector if possible?
...	Currently not used.

Value

A vector of length 2 with the minimum and maximum value of the support.

`support.LogNormal` *Return the support of the LogNormal distribution*

Description

Return the support of the LogNormal distribution

Usage

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

Arguments

`d`	An `LogNormal` object created by a call to `LogNormal()`.
`drop`	logical. Should the result be simplified to a vector if possible?
`...`	Currently not used.

Value

A vector of length 2 with the minimum and maximum value of the support.

`support.NegativeBinomial`
Return the support of the NegativeBinomial distribution

Description

Return the support of the NegativeBinomial distribution

Usage

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

Arguments

`d`	An `NegativeBinomial` object created by a call to `NegativeBinomial()`.
`drop`	logical. Should the result be simplified to a vector if possible?
`...`	Currently not used.

Value

A vector of length 2 with the minimum and maximum value of the support.

support.Normal *Return the support of the Normal distribution*

Description

Return the support of the Normal distribution

Usage

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

Arguments

d	An Normal object created by a call to Normal().
drop	logical. Should the result be simplified to a vector if possible?
...	Currently not used.

Value

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.

support.Poisson *Return the support of the Poisson distribution*

Description

Return the support of the Poisson distribution

Usage

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

Arguments

d	An Poisson object created by a call to Poisson().
drop	logical. Should the result be simplified to a vector if possible?
...	Currently not used.

Value

A vector of length 2 with the minimum and maximum value of the support.

support.PoissonBinomial *Return the support of the PoissonBinomial distribution*

Description

Return the support of the PoissonBinomial distribution

Usage

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

Arguments

d	A PoissonBinomial object created by a call to PoissonBinomial().
drop	logical. Shoul the result be simplified to a vector if possible?
...	Currently not used.

Value

A vector of length 2 with the minimum and maximum value of the support.

support.RevWeibull *Return the support of the RevWeibull distribution*

Description

Return the support of the RevWeibull distribution

Usage

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

Arguments

d	An RevWeibull object created by a call to RevWeibull().
drop	logical. Should the result be simplified to a vector if possible?
...	Currently not used.

Value

A vector of length 2 with the minimum and maximum value of the support.

support.SinhArcsinh *Return the support of the Sinh-Arcsinh distribution*

Description

Return the support of the Sinh-Arcsinh distribution

Usage

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

Arguments

d	An SinhArcsinh object created by a call to SinhArcsinh().
drop	logical. Should the result be simplified to a vector if possible?
...	Currently not used.

Value

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.

support.StudentsT *Return the support of the StudentsT distribution*

Description

Return the support of the StudentsT distribution

Usage

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

Arguments

d	An StudentsT object created by a call to StudentsT().
drop	logical. Should the result be simplified to a vector if possible?
...	Currently not used.

Value

A vector of length 2 with the minimum and maximum value of the support.

support.Tukey *Return the support of the Tukey distribution*

Description

Return the support of the Tukey distribution

Usage

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

Arguments

d	An Tukey object created by a call to Tukey().
drop	logical. Should the result be simplified to a vector if possible?
...	Currently not used.

Value

A vector of length 2 with the minimum and maximum value of the support.

support.Uniform *Return the support of the Uniform distribution*

Description

Return the support of the Uniform distribution

Usage

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

Arguments

d	An Uniform object created by a call to Uniform().
drop	logical. Should the result be simplified to a vector if possible?
...	Currently not used.

Value

A vector of length 2 with the minimum and maximum value of the support.

support.Weibull *Return the support of the Weibull distribution*

Description

Return the support of the Weibull distribution

Usage

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

Arguments

d	An Weibull object created by a call to Weibull().
drop	logical. Should the result be simplified to a vector if possible?
...	Currently not used.

Value

A vector of length 2 with the minimum and maximum value of the support.

support.ZINegativeBinomial *Return the support of the zero-inflated negative binomial distribution*

Description

Return the support of the zero-inflated negative binomial distribution

Usage

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

Arguments

d	An ZINegativeBinomial object created by a call to ZINegativeBinomial().
drop	logical. Should the result be simplified to a vector if possible?
...	Currently not used.

Value

A vector of length 2 with the minimum and maximum value of the support.

support.ZIPoisson *Return the support of the zero-inflated Poisson distribution*

Description

Return the support of the zero-inflated Poisson distribution

Usage

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

Arguments

d	An ZIPoisson object created by a call to ZIPoisson().
drop	logical. Should the result be simplified to a vector if possible?
...	Currently not used.

Value

A vector of length 2 with the minimum and maximum value of the support.

support.ZTNegativeBinomial *Return the support of the zero-truncated negative binomial distribution*

Description

Return the support of the zero-truncated negative binomial distribution

Usage

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

Arguments

d	An ZTNegativeBinomial object created by a call to ZTNegativeBinomial().
drop	logical. Should the result be simplified to a vector if possible?
...	Currently not used.

Value

A vector of length 2 with the minimum and maximum value of the support.

support.ZTPoisson *Return the support of the zero-truncated Poisson distribution*

Description

Return the support of the zero-truncated Poisson distribution

Usage

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

Arguments

d	An ZTPoisson object created by a call to ZTPoisson().
drop	logical. Should the result be simplified to a vector if possible?
...	Currently not used.

Value

A vector of length 2 with the minimum and maximum value of the support.

Tukey *Create a Tukey distribution*

Description

Tukey's studentized range distribution, used for Tukey's honestly significant differences test in ANOVA.

Usage

```
Tukey(nmeans = numeric(), df = numeric(), nranges = numeric())
```

Arguments

nmeans	Sample size for each range.
df	Degrees of freedom.
nranges	Number of groups being compared.

Details

We recommend reading this documentation on https://zeileis.github.io/distributions3/, where the math will render with additional detail and much greater clarity.

Support: R^+, the set of positive real numbers.

Other properties of Tukey's Studentized Range Distribution are omitted, largely because the distribution is not fun to work with.

Value

A Tukey object.

See Also

Other continuous distributions: Beta(), Cauchy(), ChiSquare(), Erlang(), Exponential(), FisherF(), Frechet(), GEV(), GP(), Gamma(), Gumbel(), LogNormal(), Logistic(), Normal(), RevWeibull(), SinhArcsinh(), StudentsT(), Uniform(), Weibull()

Examples

```
set.seed(27)

X <- Tukey(4L, 16L, 2L)
X

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

Uniform *Create a Continuous Uniform distribution*

Description

A distribution with constant density on an interval. The continuous analogue to the Categorical() distribution.

Usage

```
Uniform(a = 0, b = 1)
```

Arguments

a	The a parameter. a can be any value in the set of real numbers. Defaults to 0.
b	The a parameter. b can be any value in the set of real numbers. It should be strictly bigger than a, but if is not, the order of the parameters is inverted. Defaults to 1.

Value

A Uniform object.

See Also

Other continuous distributions: Beta(), Cauchy(), ChiSquare(), Erlang(), Exponential(), FisherF(), Frechet(), GEV(), GP(), Gamma(), Gumbel(), LogNormal(), Logistic(), Normal(), RevWeibull(), SinhArcsinh(), StudentsT(), Tukey(), Weibull()

Examples

```
set.seed(27)

X <- Uniform(1, 2)
X

random(X, 10)

pdf(X, 0.7)
log_pdf(X, 0.7)

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

cdf(X, quantile(X, 0.7))
quantile(X, cdf(X, 0.7))
```

---

variance *Compute the moments of a probability distribution*

Description

Generic functions for computing moments (variance, skewness, excess kurtosis) from probability distributions.

Usage

```
variance(x, ...)

skewness(x, ...)

kurtosis(x, ...)
```

Arguments

x	An object. The package provides methods for distribution objects such as those from Normal() or Binomial() etc.
...	Arguments passed to methods. Unevaluated arguments will generate a warning to catch mispellings or other possible errors.

Details

The functions `variance`, `skewness`, and `kurtosis` are new generic functions for computing moments of probability distributions such as those provided in this package. Additionally, the probability distributions from **distributions3** all have methods for the `mean` generic. Moreover, quantiles can be computed with methods for `quantile`. For examples illustrating the usage with probability distribution objects, see the manual pages of the respective distributions, e.g., `Normal` or `Binomial` etc.

Value

Numeric vector with the values of the moments.

See Also

`mean`, `quantile`, `cdf`, `random`

`Weibull` *Create a Weibull distribution*

Description

Generalization of the gamma distribution. Often used in survival and time-to-event analyses.

Usage

```
Weibull(shape = numeric(), scale = numeric())
```

Arguments

`shape`	The shape parameter k. Can be any positive real number.
`scale`	The scale parameter λ. Can be any positive real number.

Details

We recommend reading this documentation on `https://zeileis.github.io/distributions3/`, where the math will render with additional detail and much greater clarity.

In the following, let X be a Weibull random variable with success probability `p` = p.

Support: R^+ and zero.

Mean: $\lambda\Gamma(1 + 1/k)$, where Γ is the gamma function.

Variance: $\lambda^2[\Gamma(1 + \frac{2}{k}) - (\Gamma(1 + \frac{1}{k}))^2]$

Probability density function (p.d.f):

$$f(x) = \frac{k}{\lambda}(\frac{x}{\lambda})^{k-1}e^{-(x/\lambda)^k}, x \geq 0$$

Cumulative distribution function (c.d.f):

$$F(x) = 1 - e^{-(x/\lambda)^k}, x \geq 0$$

Moment generating function (m.g.f):

$$\sum_{n=0}^{\infty} \frac{t^n \lambda^n}{n!} \Gamma(1 + n/k), k \geq 1$$

Value

A `Weibull` object.

See Also

Other continuous distributions: `Beta()`, `Cauchy()`, `ChiSquare()`, `Erlang()`, `Exponential()`, `FisherF()`, `Frechet()`, `GEV()`, `GP()`, `Gamma()`, `Gumbel()`, `LogNormal()`, `Logistic()`, `Normal()`, `RevWeibull()`, `SinhArcsinh()`, `StudentsT()`, `Tukey()`, `Uniform()`

Examples

```
set.seed(27)

X <- Weibull(0.3, 2)
X

random(X, 10)

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

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

`ZINegativeBinomial` *Create a zero-inflated negative binomial distribution*

Description

Zero-inflated negative binomial distributions are frequently used to model counts with overdispersion and many zero observations.

Usage

```
ZINegativeBinomial(mu, theta, pi)
```

Arguments

`mu`	Location parameter of the negative binomial component of the distribution. Can be any positive number.
`theta`	Overdispersion parameter of the negative binomial component of the distribution. Can be any positive number.
`pi`	Zero-inflation probability, can be any value in `[0, 1]`.

Details

We recommend reading this documentation on https://zeileis.github.io/distributions3/, where the math will render with additional detail.

In the following, let X be a zero-inflated negative binomial random variable with parameters `mu` = μ and `theta` = θ.

Support: $\{0, 1, 2, 3, ...\}$

Mean: $(1 - \pi) \cdot \mu$

Variance: $(1 - \pi) \cdot \mu \cdot (1 + (\pi + 1/\theta) \cdot \mu)$

Probability mass function (p.m.f.):

$$P(X = k) = \pi \cdot I_0(k) + (1 - \pi) \cdot f(k; \mu, \theta)$$

where $I_0(k)$ is the indicator function for zero and $f(k; \mu, \theta)$ is the p.m.f. of the `NegativeBinomial` distribution.

Cumulative distribution function (c.d.f.):

$$P(X \leq k) = \pi + (1 - \pi) \cdot F(k; \mu, \theta)$$

where $F(k; \mu, \theta)$ is the c.d.f. of the `NegativeBinomial` distribution.

Moment generating function (m.g.f.):

Omitted for now.

Value

A `ZINegativeBinomial` object.

See Also

Other discrete distributions: `Bernoulli()`, `Binomial()`, `Categorical()`, `Geometric()`, `HurdleNegativeBinomial()`, `HurdlePoisson()`, `HyperGeometric()`, `Multinomial()`, `NegativeBinomial()`, `Poisson()`, `PoissonBinomial()`, `ZIPoisson()`, `ZTNegativeBinomial()`, `ZTPoisson()`

Examples

```
## set up a zero-inflated negative binomial distribution
X <- ZINegativeBinomial(mu = 2.5, theta = 1, pi = 0.25)
X

## standard functions
pdf(X, 0:8)
cdf(X, 0:8)
quantile(X, seq(0, 1, by = 0.25))

## cdf() and quantile() are inverses for each other
quantile(X, cdf(X, 3))

## density visualization
plot(0:8, pdf(X, 0:8), type = "h", lwd = 2)

## corresponding sample with histogram of empirical frequencies
set.seed(0)
x <- random(X, 500)
hist(x, breaks = -1:max(x) + 0.5)
```

`ZIPoisson` *Create a zero-inflated Poisson distribution*

Description

Zero-inflated Poisson distributions are frequently used to model counts with many zero observations.

Usage

```
ZIPoisson(lambda, pi)
```

Arguments

`lambda`	Parameter of the Poisson component of the distribution. Can be any positive number.
`pi`	Zero-inflation probability, can be any value in `[0, 1]`.

Details

We recommend reading this documentation on https://zeileis.github.io/distributions3/, where the math will render with additional detail.

In the following, let X be a zero-inflated Poisson random variable with parameter `lambda` = λ.

Support: $\{0, 1, 2, 3, ...\}$

Mean: $(1 - \pi) \cdot \lambda$

Variance: $(1 - \pi) \cdot \lambda \cdot (1 + \pi \cdot \lambda)$

Probability mass function (p.m.f.):

$$P(X = k) = \pi \cdot I_0(k) + (1 - \pi) \cdot f(k; \lambda)$$

where $I_0(k)$ is the indicator function for zero and $f(k; \lambda)$ is the p.m.f. of the Poisson distribution.

Cumulative distribution function (c.d.f.):

$$P(X \leq k) = \pi + (1 - \pi) \cdot F(k; \lambda)$$

where $F(k; \lambda)$ is the c.d.f. of the Poisson distribution.

Moment generating function (m.g.f.):

$$E(e^{tX}) = \pi + (1 - \pi) \cdot e^{\lambda(e^t - 1)}$$

Value

A ZIPoisson object.

See Also

Other discrete distributions: Bernoulli(), Binomial(), Categorical(), Geometric(), HurdleNegativeBinomial(), HurdlePoisson(), HyperGeometric(), Multinomial(), NegativeBinomial(), Poisson(), PoissonBinomial(), ZINegativeBinomial(), ZTNegativeBinomial(), ZTPoisson()

Examples

```
## set up a zero-inflated Poisson distribution
X <- ZIPoisson(lambda = 2.5, pi = 0.25)
X

## standard functions
pdf(X, 0:8)
cdf(X, 0:8)
quantile(X, seq(0, 1, by = 0.25))

## cdf() and quantile() are inverses for each other
quantile(X, cdf(X, 3))

## density visualization
plot(0:8, pdf(X, 0:8), type = "h", lwd = 2)

## corresponding sample with histogram of empirical frequencies
set.seed(0)
x <- random(X, 500)
hist(x, breaks = -1:max(x) + 0.5)
```

`ZTNegativeBinomial` *Create a zero-truncated negative binomial distribution*

Description

Zero-truncated negative binomial distributions are frequently used to model counts where zero observations cannot occur or have been excluded.

Usage

```
ZTNegativeBinomial(mu = numeric(), theta = numeric())
```

Arguments

`mu`	Location parameter of the negative binomial component of the distribution. Can be any positive number.
`theta`	Overdispersion parameter of the negative binomial component of the distribution. Can be any positive number.

Details

We recommend reading this documentation on https://zeileis.github.io/distributions3/, where the math will render with additional detail.

In the following, let X be a zero-truncated negative binomial random variable with parameter `mu` = μ.

Support: $\{1, 2, 3, ...\}$

Mean:

$$\mu \cdot \frac{1}{1 - F(0; \mu, \theta)}$$

where $F(k; \mu, \theta)$ is the c.d.f. of the `NegativeBinomial` distribution.

Variance: $m \cdot (\mu + 1 - m)$, where m is the mean above.

Probability mass function (p.m.f.):

$$P(X = k) = \frac{f(k; \mu, \theta)}{1 - F(0; \mu, \theta)}$$

where $f(k; \mu, \theta)$ is the p.m.f. of the `NegativeBinomial` distribution.

Cumulative distribution function (c.d.f.):

$$P(X = k) = \frac{F(k; \mu, \theta)}{1 - F(0; \mu, \theta)}$$

Moment generating function (m.g.f.):

Omitted for now.

Value

A ZTNegativeBinomial object.

See Also

Other discrete distributions: Bernoulli(), Binomial(), Categorical(), Geometric(), HurdleNegativeBinomial(), HurdlePoisson(), HyperGeometric(), Multinomial(), NegativeBinomial(), Poisson(), PoissonBinomial(), ZINegativeBinomial(), ZIPoisson(), ZTPoisson()

Examples

```
## set up a zero-truncated negative binomial distribution
X <- ZTNegativeBinomial(mu = 2.5, theta = 1)
X

## standard functions
pdf(X, 0:8)
cdf(X, 0:8)
quantile(X, seq(0, 1, by = 0.25))

## cdf() and quantile() are inverses for each other
quantile(X, cdf(X, 3))

## density visualization
plot(0:8, pdf(X, 0:8), type = "h", lwd = 2)

## corresponding sample with histogram of empirical frequencies
set.seed(0)
x <- random(X, 500)
hist(x, breaks = -1:max(x) + 0.5)
```

ZTPoisson *Create a zero-truncated Poisson distribution*

Description

Zero-truncated Poisson distributions are frequently used to model counts where zero observations cannot occur or have been excluded.

Usage

```
ZTPoisson(lambda = numeric())
```

Arguments

lambda	Parameter of the underlying untruncated Poisson distribution. Can be any positive number.

Details

We recommend reading this documentation on https://zeileis.github.io/distributions3/, where the math will render with additional detail.

In the following, let X be a zero-truncated Poisson random variable with parameter `lambda` = λ.

Support: $\{1, 2, 3, ...\}$

Mean:

$$\lambda \cdot \frac{1}{1 - e^{-\lambda}}$$

Variance: $m \cdot (\lambda + 1 - m)$, where m is the mean above.

Probability mass function (p.m.f.):

$$P(X = k) = \frac{f(k; \lambda)}{1 - f(0; \lambda)}$$

where $f(k; \lambda)$ is the p.m.f. of the Poisson distribution.

Cumulative distribution function (c.d.f.):

$$P(X = k) = \frac{F(k; \lambda)}{1 - F(0; \lambda)}$$

where $F(k; \lambda)$ is the c.d.f. of the Poisson distribution.

Moment generating function (m.g.f.):

$$E(e^{tX}) = \frac{1}{1 - e^{-\lambda}} \cdot e^{\lambda(e^t - 1)}$$

Value

A `ZTPoisson` object.

See Also

Other discrete distributions: Bernoulli(), Binomial(), Categorical(), Geometric(), HurdleNegativeBinomial(), HurdlePoisson(), HyperGeometric(), Multinomial(), NegativeBinomial(), Poisson(), PoissonBinomial(), ZINegativeBinomial(), ZIPoisson(), ZTNegativeBinomial()

Examples

```
## set up a zero-truncated Poisson distribution
X <- ZTPoisson(lambda = 2.5)
X

## standard functions
pdf(X, 0:8)
cdf(X, 0:8)
quantile(X, seq(0, 1, by = 0.25))

## cdf() and quantile() are inverses for each other
```

```
quantile(X, cdf(X, 3))

## density visualization
plot(0:8, pdf(X, 0:8), type = "h", lwd = 2)

## corresponding sample with histogram of empirical frequencies
set.seed(0)
x <- random(X, 500)
hist(x, breaks = -1:max(x) + 0.5)
```

Index

∗ **Empirical distribution**
cdf.Empirical, 25
dempirical, 67
Empirical, 78
pdf.Empirical, 135
quantile.Empirical, 189
random.Empirical, 234
support.Empirical, 288
∗ **Exponential distribution**
fit_mle.Exponential, 89
∗ **Geometric distribution**
cdf.Geometric, 33
pdf.Geometric, 143
quantile.Geometric, 197
random.Geometric, 241
∗ **HyperGeometric distribution**
cdf.HyperGeometric, 39
pdf.HyperGeometric, 150
quantile.HyperGeometric, 203
random.HyperGeometric, 247
∗ **LogNormal distribution**
cdf.LogNormal, 42
fit_mle.LogNormal, 90
pdf.LogNormal, 152
quantile.LogNormal, 205
random.LogNormal, 249
∗ **Logistic distribution**
cdf.Logistic, 40
pdf.Logistic, 151
quantile.Logistic, 204
random.Logistic, 248
∗ **Multinomial distribution**
pdf.Multinomial, 153
random.Multinomial, 250
∗ **NegativeBinomial distribution**
cdf.NegativeBinomial, 43
pdf.NegativeBinomial, 154
quantile.NegativeBinomial, 206
random.NegativeBinomial, 251
∗ **Normal distribution**
cdf.Normal, 44
fit_mle.Normal, 91
pdf.Normal, 156
quantile.Normal, 208
∗ **Poisson distribution**
fit_mle.Poisson, 91
∗ **SinhArcsinh distribution**
cdf.SinhArcsinh, 50
pdf.SinhArcsinh, 162
quantile.SinhArcsinh, 214
∗ **SinhArcsinh**
dsinharcsinh, 71
∗ **StudentsT distribution**
cdf.StudentsT, 52
pdf.StudentsT, 164
quantile.StudentsT, 216
random.StudentsT, 258
∗ **Tukey distribution**
cdf.Tukey, 54
quantile.Tukey, 218
∗ **Weibull distribution**
cdf.Weibull, 56
pdf.Weibull, 167
quantile.Weibull, 220
random.Weibull, 262
∗ **continuous distributions**
Beta, 12
Cauchy, 16
ChiSquare, 62
Erlang, 81
Exponential, 82
FisherF, 86
Frechet, 92
Gamma, 93
GEV, 96
GP, 98
Gumbel, 100
Logistic, 107

LogNormal, 109
Normal, 116
RevWeibull, 267
SinhArcsinh, 272
StudentsT, 277
Tukey, 303
Uniform, 304
Weibull, 306
∗ **datasets**
FIFA2018, 84
∗ **discrete distributions**
Bernoulli, 10
Binomial, 13
Categorical, 15
Geometric, 95
HurdleNegativeBinomial, 101
HurdlePoisson, 103
HyperGeometric, 104
Multinomial, 113
NegativeBinomial, 115
Poisson, 176
PoissonBinomial, 178
ZINegativeBinomial, 307
ZIPoisson, 309
ZTNegativeBinomial, 311
ZTPoisson, 312
∗ **distribution**
dhnbinom, 69
dhpois, 70
dsinharcsinh, 71
dzinbinom, 73
dzipois, 74
dztnbinom, 75
dztpois, 76
prodist, 180
∗ **multivariate distributions**
Multinomial, 113

aes(), *275*
annotation_borders(), *276*
apply, *68*
apply_dpqr, 7
approxfun, *174*
arima, *180, 181*

Bernoulli, 10
Bernoulli(), *14, 15*, *18*, *95, 96*, *102*, *104, 105*, *113, 114*, *116*, *121*, *177*, *179*, *183*, *227*, *286*, *308*, *310*, *312, 313*
Beta, 12
Beta(), *17*, *19*, *63*, *82, 83*, *87*, *93, 94*, *97*, *99*, *101*, *108*, *110*, *118*, *122*, *184*, *228*, *268*, *273*, *278*, *286*, *304, 305*, *307*
Binomial, 13, *65*, *178*, *226*, *306*
Binomial(), *10, 11*, *15*, *18*, *21*, *87*, *96*, *102*, *104–106*, *110*, *114*, *116*, *120*, *123*, *177*, *179*, *185*, *226*, *229*, *269*, *279*, *285*, *287*, *305*, *308*, *310*, *312, 313*

Categorical, 15
Categorical(), *11*, *14*, *22*, *96*, *102*, *104, 105*, *113, 114*, *116*, *124*, *177*, *179*, *186*, *230*, *304*, *308*, *310*, *312, 313*
Cauchy, 16
Cauchy(), *12*, *23*, *63*, *82, 83*, *87*, *93, 94*, *97*, *99*, *101*, *108*, *110*, *118*, *125*, *187*, *231*, *268*, *273*, *278*, *287*, *304, 305*, *307*
cdf, 17, *306*
cdf(), *107*
cdf.Bernoulli, 18
cdf.Beta, 19
cdf.Binomial, 20
cdf.Categorical, 21
cdf.Cauchy, 23
cdf.ChiSquare, 24
cdf.distribution *(*pdf.distribution*)*, 128
cdf.Empirical, 25
cdf.Empirical(), *68*, *80*, *135*, *190*, *234*, *289*
cdf.Erlang, 27
cdf.Exponential, 28
cdf.FisherF, 30
cdf.Frechet, 31
cdf.Gamma, 32
cdf.Geometric, 33
cdf.Geometric(), *143*, *198*, *241*
cdf.GEV, 34
cdf.GP, 35
cdf.Gumbel, 36
cdf.HurdleNegativeBinomial, 37
cdf.HurdlePoisson, 38
cdf.HyperGeometric, 39
cdf.HyperGeometric(), *150*, *204*, *247*
cdf.Logistic, 40
cdf.Logistic(), *152*, *205*, *248*
cdf.LogNormal, 42
cdf.LogNormal(), *91*, *153*, *206*, *249*
cdf.NegativeBinomial, 43

cdf.NegativeBinomial(), *155*, *207*, *251*
cdf.Normal, 44
cdf.Normal(), *91*, *156*, *208*
cdf.Poisson, 46
cdf.PoissonBinomial, 48
cdf.RevWeibull, 49
cdf.SinhArcsinh, 50
cdf.SinhArcsinh(), *163*, *215*
cdf.StudentsT, 52
cdf.StudentsT(), *164*, *217*, *259*
cdf.Tukey, 54
cdf.Tukey(), *219*
cdf.Uniform, 55
cdf.Weibull, 56
cdf.Weibull(), *167*, *221*, *262*
cdf.ZINegativeBinomial, 57
cdf.ZIPoisson, 58
cdf.ZTNegativeBinomial, 59
cdf.ZTPoisson, 60
ChiSquare, 62
ChiSquare(), *12*, *17*, *24*, *63*, *82*, *83*, *87*, *93*, *94*, *97*, *99*, *101*, *108*, *110*, *118*, *127*, *189*, *232*, *268*, *273*, *278*, *288*, *304*, *305*, *307*
crps, *63*
crps.BAMLSS *(*crps.distribution*)*, 63
crps.Bernoulli *(*crps.distribution*)*, 63
crps.Beta *(*crps.distribution*)*, 63
crps.Binomial *(*crps.distribution*)*, 63
crps.distribution, 63
crps.Erlang *(*crps.distribution*)*, 63
crps.Exponential *(*crps.distribution*)*, 63
crps.GAMLSS *(*crps.distribution*)*, 63
crps.Gamma *(*crps.distribution*)*, 63
crps.Geometric *(*crps.distribution*)*, 63
crps.GEV *(*crps.distribution*)*, 63
crps.Gumbel *(*crps.distribution*)*, 63
crps.HyperGeometric *(*crps.distribution*)*, 63
crps.Logistic *(*crps.distribution*)*, 63
crps.LogNormal *(*crps.distribution*)*, 63
crps.NegativeBinomial *(*crps.distribution*)*, 63
crps.Normal *(*crps.distribution*)*, 63
crps.Poisson *(*crps.distribution*)*, 63
crps.StudentsT *(*crps.distribution*)*, 63
crps.Uniform *(*crps.distribution*)*, 63
crps.XBetaX *(*crps.distribution*)*, 63
crps_norm, *65*

dbeta, *122*
dbinom, *121*, *123*
dcauchy, *126*
dchisq, *127*
dempirical, 67
dempirical(), *26*, *68*, *80*, *135*, *190*, *234*, *289*
density(), *68*
deviance, *180*
dexp, *139*
df, *140*
dgamma, *137*, *142*
dgeom, *143*
dgev, *141*, *144*, *147*, *161*
dgp, *145*
dhnbinom, 69, *148*
dhpois, 70, *149*
dhyper, *150*
distribution_calculate_moments(), *111*
dlnorm, *153*
dlogis, *151*
dnbinom, *69*, *74*, *76*, *155*
dnorm, *156*, *162*
dpbinom, *160*
dpois, *71*, *75*, *77*, *159*
dsinharcsinh, 71
dt, *164*
dunif, *166*
dweibull, *167*
dzinbinom, 73, *168*
dzipois, 74, *170*
dztnbinom, 75, *171*
dztpois, 76, *172*

Empirical, 78
Empirical(), *25*, *26*, *68*, *78*, *135*, *189*, *190*, *234*, *288*, *289*
Erlang, 81
Erlang(), *12*, *17*, *28*, *63*, *83*, *87*, *93*, *94*, *97*, *99*, *101*, *108*, *110*, *118*, *137*, *192*, *236*, *268*, *273*, *278*, *289*, *304*, *305*, *307*
expand.grid, *174*
Exponential, 82
Exponential(), *12*, *17*, *29*, *63*, *82*, *87*, *89*, *93*, *94*, *97*, *99*, *101*, *108*, *110*, *118*, *138*, *193*, *237*, *268*, *273*, *278*, *281*, *290*, *304*, *305*, *307*

FIFA2018, 84
FisherF, 86
FisherF(), *12*, *17*, *30*, *63*, *82, 83*, *93, 94*, *97*, *99*, *101*, *108*, *110*, *118*, *140*, *194*, *238*, *268*, *273*, *278*, *290*, *304, 305*, *307*
fit_mle, 87
fit_mle.Bernoulli, 88
fit_mle.Binomial, 88
fit_mle.Exponential, 89
fit_mle.Gamma, 89
fit_mle.Geometric, 90
fit_mle.LogNormal, 90
fit_mle.LogNormal(), *42*, *153*, *206*, *249*
fit_mle.Normal, 91
fit_mle.Normal(), *45*, *156*, *208*
fit_mle.Poisson, 91
fortify(), *275*
Frechet, 92
Frechet(), *12*, *17*, *31*, *63*, *82, 83*, *87*, *94*, *97*, *99*, *101*, *108*, *110*, *118*, *141*, *195*, *239*, *268*, *273*, *278*, *291*, *304, 305*, *307*

Gamma, 93
Gamma(), *12*, *17*, *32*, *63*, *82, 83*, *87*, *89*, *93*, *97*, *99*, *101*, *108*, *110*, *118*, *142*, *196*, *240*, *268*, *273*, *278*, *282*, *291*, *304, 305*, *307*
geom_auc *(*stat_auc*)*, 274
GeomAuc *(*stat_auc*)*, 274
Geometric, 95
Geometric(), *11*, *14, 15*, *33*, *102*, *104, 105*, *114*, *116*, *143*, *177*, *179*, *197*, *241*, *292*, *308*, *310*, *312, 313*
GEV, 96
GEV(), *12*, *17*, *34*, *63*, *82, 83*, *87*, *93, 94*, *99*, *101*, *108*, *110*, *118*, *144*, *198*, *242*, *268*, *273*, *278*, *292*, *304, 305*, *307*
ggplot(), *275*
glm, *180, 181*
GP, 98
GP(), *12*, *17*, *35*, *63*, *82, 83*, *87*, *93, 94*, *97*, *101*, *108*, *110*, *118*, *145*, *199*, *243*, *268*, *273*, *278*, *293*, *304, 305*, *307*
Gumbel, 100
Gumbel(), *12*, *17*, *36*, *63*, *82, 83*, *87*, *93, 94*, *97*, *99*, *108*, *110*, *118*, *146*, *200*, *244*, *268*, *273*, *278*, *293*, *304, 305*, *307*

hessian *(*score-hessian*)*, 268
hist, *68*
HurdleNegativeBinomial, *69*, 101
HurdleNegativeBinomial(), *11*, *14, 15*, *37*, *96*, *104, 105*, *114*, *116*, *148*, *177*, *179*, *201*, *245*, *294*, *308*, *310*, *312, 313*
HurdlePoisson, *71*, 103
HurdlePoisson(), *11*, *14, 15*, *38*, *96*, *102*, *105*, *114*, *116*, *149*, *177*, *179*, *202*, *246*, *294*, *308*, *310*, *312, 313*
HyperGeometric, 104
HyperGeometric(), *11*, *14, 15*, *40*, *96*, *102*, *104*, *114*, *116*, *150*, *177*, *179*, *203*, *247*, *295*, *308*, *310*, *312, 313*

is_continuous, *66*
is_continuous *(*is_discrete*)*, 106
is_discrete, *66*, 106
is_distribution, 107

key glyphs, *276*
kurtosis *(*variance*)*, 305
kurtosis(), *79*
kurtosis.distribution *(*mean.distribution*)*, 111
kurtosis.Empirical *(*Empirical*)*, 78
kurtosis.Empirical(), *78*

lapply, *65*, *129*
layer geom, *275*
layer position, *275*
layer stat, *276*
layer(), *276*
legend, *174*
likelihood *(*log_likelihood*)*, 110
lines, *174*
lm, *180, 181*
log_likelihood, 110
log_pdf *(*pdf*)*, 119
log_pdf.Bernoulli *(*pdf.Bernoulli*)*, 120
log_pdf.Beta *(*pdf.Beta*)*, 122
log_pdf.Binomial *(*pdf.Binomial*)*, 123
log_pdf.Categorical *(*pdf.Categorical*)*, 124
log_pdf.Cauchy *(*pdf.Cauchy*)*, 125
log_pdf.ChiSquare *(*pdf.ChiSquare*)*, 126
log_pdf.distribution *(*pdf.distribution*)*, 128

log_pdf.Empirical (pdf.Empirical), 135
log_pdf.Erlang (pdf.Erlang), 137
log_pdf.Exponential (pdf.Exponential), 138
log_pdf.FisherF (pdf.FisherF), 139
log_pdf.Frechet (pdf.Frechet), 140
log_pdf.Gamma (pdf.Gamma), 142
log_pdf.Geometric (pdf.Geometric), 143
log_pdf.GEV (pdf.GEV), 144
log_pdf.GP (pdf.GP), 145
log_pdf.Gumbel (pdf.Gumbel), 146
log_pdf.HurdleNegativeBinomial (pdf.HurdleNegativeBinomial), 147
log_pdf.HurdlePoisson (pdf.HurdlePoisson), 148
log_pdf.HyperGeometric (pdf.HyperGeometric), 150
log_pdf.Logistic (pdf.Logistic), 151
log_pdf.LogNormal (pdf.LogNormal), 152
log_pdf.Multinomial (pdf.Multinomial), 153
log_pdf.NegativeBinomial (pdf.NegativeBinomial), 154
log_pdf.Normal (pdf.Normal), 156
log_pdf.Poisson (pdf.Poisson), 158
log_pdf.PoissonBinomial (pdf.PoissonBinomial), 159
log_pdf.RevWeibull (pdf.RevWeibull), 161
log_pdf.SinhArcsinh (pdf.SinhArcsinh), 162
log_pdf.StudentsT (pdf.StudentsT), 164
log_pdf.Uniform (pdf.Uniform), 166
log_pdf.Weibull (pdf.Weibull), 167
log_pdf.ZINegativeBinomial (pdf.ZINegativeBinomial), 168
log_pdf.ZIPoisson (pdf.ZIPoisson), 169
log_pdf.ZTNegativeBinomial (pdf.ZTNegativeBinomial), 170
log_pdf.ZTPoisson (pdf.ZTPoisson), 172
Logistic, 107
Logistic(), *12*, *17*, *41*, *63*, *82, 83*, *87*, *93, 94*, *97*, *99*, *101*, *110*, *118*, *151*, *205*, *248*, *268*, *273*, *278*, *295*, *304, 305*, *307*
logLik, *181*
LogNormal, 109
LogNormal(), *12*, *17*, *42*, *63*, *82, 83*, *87*, *90*, *93, 94*, *97*, *99*, *101*, *108*, *118*, *152*, *206*, *249*, *268*, *273*, *278*, *283*, *296*, *304, 305*, *307*

make_positive_integer (apply_dpqr), 7
make_support (apply_dpqr), 7
mclapply, *65*, *129*
mean, *306*
mean.distribution, 111
mean.Empirical (Empirical), 78
Multinomial, 113
Multinomial(), *11*, *14, 15*, *96*, *102*, *104, 105*, *116*, *153, 154*, *177*, *179*, *250*, *308*, *310*, *312, 313*

NegativeBinomial, *102*, 115, *308*, *311*
NegativeBinomial(), *11*, *14, 15*, *43*, *96*, *102*, *104, 105*, *114*, *155*, *177*, *179*, *207*, *251*, *296*, *308*, *310*, *312, 313*
Normal, *65*, 116, *180*, *226*, *306*
Normal(), *12*, *17, 18*, *44*, *63*, *82, 83*, *87*, *91*, *93, 94*, *96, 97*, *99*, *101*, *106*, *108–110*, *120*, *156*, *208*, *226*, *252*, *268, 269*, *273*, *277–279*, *284, 285*, *297*, *304, 305*, *307*

parLapply, *65*, *129*
pbeta, *20*
pbinom, *18*, *21*
pcauchy, *23*
pchisq, *24*
pdf, 119
pdf(), *128, 129*
pdf.Bernoulli, 120
pdf.Beta, 122
pdf.Binomial, 123
pdf.Categorical, 124
pdf.Cauchy, 125
pdf.ChiSquare, 126
pdf.distribution, 128
pdf.Empirical, 135
pdf.Empirical(), *26*, *68*, *80*, *190*, *234*, *289*
pdf.Erlang, 137
pdf.Exponential, 138
pdf.FisherF, 139
pdf.Frechet, 140
pdf.Gamma, 142
pdf.Geometric, 143
pdf.Geometric(), *33*, *198*, *241*
pdf.GEV, 144

pdf.GP, 145
pdf.Gumbel, 146
pdf.HurdleNegativeBinomial, 147
pdf.HurdlePoisson, 148
pdf.HyperGeometric, 150
pdf.HyperGeometric(), *40*, *204*, *247*
pdf.Logistic, 151
pdf.Logistic(), *41*, *205*, *248*
pdf.LogNormal, 152
pdf.LogNormal(), *42*, *91*, *206*, *249*
pdf.Multinomial, 153
pdf.Multinomial(), *250*
pdf.NegativeBinomial, 154
pdf.NegativeBinomial(), *43*, *207*, *251*
pdf.Normal, 156
pdf.Normal(), *45*, *91*, *208*
pdf.Poisson, 158
pdf.PoissonBinomial, 159
pdf.RevWeibull, 161
pdf.SinhArcsinh, 162
pdf.SinhArcsinh(), *51*, *215*
pdf.StudentsT, 164
pdf.StudentsT(), *53*, *217*, *259*
pdf.Uniform, 166
pdf.Weibull, 167
pdf.Weibull(), *57*, *221*, *262*
pdf.ZINegativeBinomial, 168
pdf.ZIPoisson, 169
pdf.ZTNegativeBinomial, 170
pdf.ZTPoisson, 172
pempirical *(*dempirical*)*, 67
pempirical(), *25*
pexp, *29*
pf, *30*
pgamma, *28*, *32*
pgeom, *33*
pgev, *31*, *34*, *36*, *50*
pgp, *35*
phnbinom, *37*
phnbinom *(*dhnbinom*)*, 69
phpois, *39*
phpois *(*dhpois*)*, 70
phyper, *40*
plnorm, *42*
plogis, *41*
plot, *174*
plot.distribution, 173
plot.ecdf, *174*
plot_cdf, 175
plot_pdf, 176
pmf *(*pdf*)*, 119
pnbinom, *43*
pnorm, *44*, *48*, *51*, *160*
Poisson, *103*, *104*, 176, *310*, *313*
Poisson(), *11*, *14*, *15*, *47*, *92*, *96*, *102*, *104*, *105*, *114*, *116*, *158*, *179*, *210*, *254*, *284*, *297*, *308*, *310*, *312*, *313*
PoissonBinomial, 178
PoissonBinomial(), *11*, *14*, *15*, *48*, *96*, *102*, *104*, *105*, *114*, *116*, *160*, *177*, *212*, *255*, *298*, *308*, *310*, *312*, *313*
ppbinom, *48*
ppois, *47*
predict, *180*, *181*
prodist, 180, *271*
psinharcsinh *(*dsinharcsinh*)*, 71
psinharcsinh(), *51*
pt, *52*
ptukey, *54*
punif, *55*
pweibull, *56*
pzinbinom, *57*
pzinbinom *(*dzinbinom*)*, 73
pzipois, *59*
pzipois *(*dzipois*)*, 74
pztnbinom, *60*
pztnbinom *(*dztnbinom*)*, 75
pztpois, *61*
pztpois *(*dztpois*)*, 76

qbeta, *184*
qbinom, *183*, *185*
qcauchy, *188*
qchisq, *189*
qempirical *(*dempirical*)*, 67
qempirical(), *68*, *190*
qexp, *193*
qf, *194*
qgamma, *192*, *196*
qgeom, *197*
qgev, *195*, *198*, *200*, *213*
qgp, *199*
qhnbinom, *201*
qhnbinom *(*dhnbinom*)*, 69
qhpois, *202*
qhpois *(*dhpois*)*, 70
qhyper, *204*

qlnorm, *206*
qlogis, *205*
qnbinom, *207*
qnorm, *208*, *212*, *214*
qpbinom, *212*
qpois, *211*
qsinharcsinh *(*dsinharcsinh*)*, 71
qt, *216*
qtukey, *218*
quantile, *68*, *190*, *306*
quantile(), *128, 129*
quantile.Bernoulli, 183
quantile.Beta, 184
quantile.Binomial, 185
quantile.Categorical, 186
quantile.Cauchy, 187
quantile.ChiSquare, 188
quantile.distribution *(*pdf.distribution*)*, 128
quantile.Empirical, 189
quantile.Empirical(), *26*, *68*, *80*, *135*, *234*, *289*
quantile.Erlang, 192
quantile.Exponential, 193
quantile.FisherF, 194
quantile.Frechet, 195
quantile.Gamma, 196
quantile.Geometric, 197
quantile.Geometric(), *33*, *143*, *241*
quantile.GEV, 198
quantile.GP, 199
quantile.Gumbel, 200
quantile.HurdleNegativeBinomial, 201
quantile.HurdlePoisson, 202
quantile.HyperGeometric, 203
quantile.HyperGeometric(), *40*, *150*, *247*
quantile.Logistic, 204
quantile.Logistic(), *41*, *152*, *248*
quantile.LogNormal, 205
quantile.LogNormal(), *42*, *91*, *153*, *249*
quantile.NegativeBinomial, 206
quantile.NegativeBinomial(), *43*, *155*, *251*
quantile.Normal, 208
quantile.Normal(), *45*, *91*, *156*
quantile.Poisson, 210
quantile.PoissonBinomial, 211
quantile.RevWeibull, 213
quantile.SinhArcsinh, 214
quantile.SinhArcsinh(), *51*, *163*
quantile.StudentsT, 216
quantile.StudentsT(), *53*, *164*, *259*
quantile.Tukey, 218
quantile.Tukey(), *55*
quantile.Uniform, 219
quantile.Weibull, 220
quantile.Weibull(), *57*, *167*, *262*
quantile.ZINegativeBinomial, 221
quantile.ZIPoisson, 222
quantile.ZTNegativeBinomial, 223
quantile.ZTPoisson, 224
qunif, *219*
qweibull, *220*
qzinbinom, *221*
qzinbinom *(*dzinbinom*)*, 73
qzipois, *223*
qzipois *(*dzipois*)*, 74
qztnbinom, *224*
qztnbinom *(*dztnbinom*)*, 75
qztpois, *225*
qztpois *(*dztpois*)*, 76

random, 225, *271*, *306*
random.Bernoulli, 227
random.Beta, 228
random.Binomial, 229
random.Categorical, 230
random.Cauchy, 231
random.ChiSquare, 232
random.distribution, 233
random.Empirical, 234
random.Empirical(), *26*, *68*, *80*, *135*, *190*, *289*
random.Erlang, 236
random.Exponential, 237
random.FisherF, 238
random.Frechet, 239
random.Gamma, 240
random.Geometric, 241
random.Geometric(), *33*, *143*, *198*
random.GEV, 242
random.GP, 243
random.Gumbel, 244
random.HurdleNegativeBinomial, 245
random.HurdlePoisson, 246
random.HyperGeometric, 247
random.HyperGeometric(), *40*, *150*, *204*

random.Logistic, 248
random.Logistic(), *41*, *152*, *205*
random.LogNormal, 249
random.LogNormal(), *42*, *91*, *153*, *206*
random.Multinomial, 250
random.Multinomial(), *154*
random.NegativeBinomial, 251
random.NegativeBinomial(), *43*, *155*, *207*
random.Normal, 252
random.Poisson, 254
random.PoissonBinomial, 255
random.RevWeibull, 256
random.SinhArcsinh, 257
random.StudentsT, 258
random.StudentsT(), *53*, *164*, *217*
random.Tukey, 260
random.Uniform, 261
random.Weibull, 262
random.Weibull(), *57*, *167*, *221*
random.ZINegativeBinomial, 263
random.ZIPoisson, 264
random.ZTNegativeBinomial, 265
random.ZTPoisson, 266
rempirical *(*dempirical*)*, 67
rep_len, *174*
RevWeibull, 267
RevWeibull(), *12*, *17*, *49*, *63*, *82, 83*, *87*, *93, 94*, *97*, *99*, *101*, *108*, *110*, *118*, *161*, *213*, *256*, *273*, *278*, *298*, *304, 305*, *307*
rhnbinom *(*dhnbinom*)*, 69
rhpois *(*dhpois*)*, 70
rsinharcsinh *(*dsinharcsinh*)*, 71
rzinbinom *(*dzinbinom*)*, 73
rzipois *(*dzipois*)*, 74
rztnbinom *(*dztnbinom*)*, 75
rztpois *(*dztpois*)*, 76

sapply, *68*
score *(*score-hessian*)*, 268
score-hessian, 268
score.Bernoulli *(*score-hessian*)*, 268
score.Binomial *(*score-hessian*)*, 268
score.distribution *(*score-hessian*)*, 268
score.Normal *(*score-hessian*)*, 268
score.Poisson *(*score-hessian*)*, 268
score.SinhArcsinh *(*score-hessian*)*, 268
score.Uniform *(*score-hessian*)*, 268
set.seed, *226*, *271*
simulate, *226*, *271*
simulate.default, 271
simulate.distribution, *271*
simulate.distribution *(*random*)*, 225
SinhArcsinh, 272
SinhArcsinh(), *12*, *17*, *51*, *63*, *82, 83*, *87*, *93, 94*, *97*, *99*, *101*, *108*, *110*, *118*, *162*, *214, 215*, *257*, *268*, *278*, *299*, *304, 305*, *307*
skewness *(*variance*)*, 305
skewness(), *79*
skewness.distribution *(*mean.distribution*)*, 111
skewness.Empirical *(*Empirical*)*, 78
skewness.Empirical(), *78*
stat_auc, 274
StatAuc *(*stat_auc*)*, 274
stats::binom.test(), *13*
StudentsT, 277
StudentsT(), *12*, *17*, *52*, *63*, *82, 83*, *87*, *93, 94*, *97*, *99*, *101*, *108*, *110*, *118*, *164*, *216*, *259*, *268*, *273*, *299*, *304, 305*, *307*
suff_stat, 279
suff_stat.Bernoulli, 280
suff_stat.Binomial, 280
suff_stat.Exponential, 281
suff_stat.Gamma, 282
suff_stat.Geometric, 282
suff_stat.LogNormal, 283
suff_stat.Normal, 284
suff_stat.Poisson, 284
support, 285
support.Bernoulli, 286
support.Beta, 286
support.Binomial, 287
support.Cauchy, 287
support.ChiSquare, 288
support.Empirical, 288
support.Empirical(), *26*, *68*, *80*, *135*, *190*, *234*
support.Erlang, 289
support.Exponential, 289
support.FisherF, 290
support.Frechet, 290
support.Gamma, 291
support.Geometric, 292
support.GEV, 292

support.GP, 293
support.Gumbel, 293
support.HurdleNegativeBinomial, 294
support.HurdlePoisson, 294
support.HyperGeometric, 295
support.Logistic, 295
support.LogNormal, 296
support.NegativeBinomial, 296
support.Normal, 297
support.Poisson, 297
support.PoissonBinomial, 298
support.RevWeibull, 298
support.SinhArcsinh, 299
support.StudentsT, 299
support.Tukey, 300
support.Uniform, 300
support.Weibull, 301
support.ZINegativeBinomial, 301
support.ZIPoisson, 302
support.ZTNegativeBinomial, 302
support.ZTPoisson, 303

Tukey, 303
Tukey(), *12*, *17*, *54*, *63*, *82, 83*, *87*, *93, 94*, *97*, *99*, *101*, *108*, *110*, *118*, *261*, *268*, *273*, *278*, *300*, *305*, *307*

Uniform, 304
Uniform(), *12*, *17*, *55*, *63*, *82, 83*, *87*, *93, 94*, *97*, *99*, *101*, *108*, *110*, *118*, *166*, *219*, *261*, *268*, *273*, *278*, *300*, *304*, *307*
uniroot, *129*

variance, 305
variance.distribution *(*mean.distribution*)*, 111
variance.Empirical *(*Empirical*)*, 78

Weibull, 306
Weibull(), *12*, *17*, *56*, *63*, *82, 83*, *87*, *93, 94*, *97*, *99*, *101*, *108*, *110*, *118*, *167*, *220*, *262*, *268*, *273*, *278*, *301*, *304, 305*

ZINegativeBinomial, *74*, 307
ZINegativeBinomial(), *11*, *14, 15*, *57*, *96*, *102*, *104, 105*, *114*, *116*, *168*, *177*, *179*, *221*, *263*, *301*, *310*, *312, 313*
ZIPoisson, *75*, 309
ZIPoisson(), *11*, *14, 15*, *58*, *96*, *102*, *104, 105*, *114*, *116*, *169*, *177*, *179*, *222*, *264*, *302*, *308*, *312, 313*
ZTNegativeBinomial, *76*, 311
ZTNegativeBinomial(), *11*, *14, 15*, *60*, *96*, *102*, *104, 105*, *114*, *116*, *171*, *177*, *179*, *223*, *265*, *302*, *308*, *310*, *313*
ZTPoisson, *77*, 312
ZTPoisson(), *11*, *14, 15*, *61*, *96*, *102*, *104, 105*, *114*, *116*, *172*, *177*, *179*, *225*, *266*, *303*, *308*, *310*, *312*