Package 'distributions3'

June 21, 2022

Title Probability Distributions as S3 Objects

Version 0.2.0

Description Tools to create and manipulate probability distributions using S3. Generics random(), pdf(), cdf() and quantile() provide replacements for base R's r/d/p/q 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/alexpghayes/distributions3, https://alexpghayes.github.io/distributions3/

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

Imports ellipsis, ggplot2, glue

Suggests covr, cowplot, knitr, revdbayes (>= 1.3.5), rmarkdown, testthat (>= 3.0.0)

VignetteBuilder knitr

Encoding UTF-8

RoxygenNote 7.1.2

Config/testthat/edition 3

Config/testthat/parallel true

NeedsCompilation no

Author Alex Hayes [aut, cre] (<https://orcid.org/0000-0002-4985-5160>), Ralph Moller-Trane [aut], Emil Hvitfeldt [ctb] (<https://orcid.org/0000-0002-0679-1945>), Daniel Jordan [aut], Paul Northrop [aut], Moritz N. Lang [aut] (<https://orcid.org/0000-0002-2533-9903>), Achim Zeileis [aut] (<https://orcid.org/0000-0003-0918-3766>), Bruna Wundervald [ctb], Alessandro Gasparini [ctb]

Maintainer Alex Hayes <alexpghayes@gmail.com>

Repository CRAN

Date/Publication 2022-06-21 20:50:02 UTC

R topics documented:

`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, 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.
`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, ...) {
  ellipsis::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) {
  stopifnot("d must be a supported distribution object" = is_distribution(d))
  stopifnot(is.logical(drop))

  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, 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", 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)


## 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://alexpghayes.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(), HurdlePoisson(), HyperGeometric(), Multinomial(), NegativeBinomial(), Poisson(), ZIPoisson()

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()`, `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, 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://alexpghayes.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()`, `HurdlePoisson()`, `HyperGeometric()`, `Multinomial()`, `NegativeBinomial()`, `Poisson()`, `ZIPoisson()`

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, 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()`, `HurdlePoisson()`, `HyperGeometric()`, `Multinomial()`, `NegativeBinomial()`, `Poisson()`, `ZIPoisson()`

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://alexpghayes.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(), 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 probability density of a probability distribution*

Description

For discrete distributions, the probability mass function.

Usage

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

Arguments

d	A probability distribution object such as those created by a call to `Bernoulli()`, `Beta()`, or `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?
...	Unused. Unevaluated arguments will generate a warning to catch mispellings or other possible errors.

Value

Probabilities corresponding to the vector x.

Examples

```
X <- Normal()

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, ...)
```

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?
...	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, ...)
```

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?
...	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, ...)
```

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?
...	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, ...)
```

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?
...	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, ...)
```

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?
`...`	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.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, ...)
```

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?
`...`	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, ...)
```

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?
`...`	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, ...)
```

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?
`...`	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, ...)
```

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?
...	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, ...)
```

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?
...	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, ...)
```

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?
`...`	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, ...)
```

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?
...	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, ...)
```

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?
...	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, ...)
```

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?
`...`	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.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, ...)
```

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?
`...`	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 = 0.75))

## cdf() and quantile() are inverses for each other
cdf(X, quantile(X, 0.3))
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, ...)
```

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?
`...`	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, ...)
```

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?
`...`	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, ...)
```

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?
`...`	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, ...)
```

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?
`...`	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, ...)
```

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?
`...`	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, ...)
```

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?
...	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.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, ...)
```

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?
`...`	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.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, ...)
```

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?
...	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, ...)
```

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?
`...`	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, ...)
```

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?
`...`	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, ...)
```

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?
`...`	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.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, ...)
```

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?
...	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
```

```
cdf(X, quantile(X, 0.3))
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)
```

Arguments

`df`	Degrees of freedom. Must be positive.

Details

We recommend reading this documentation on https://alexpghayes.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()`, `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))
```

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 frame 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)
```

---

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 frame 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)
```

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, lambda)
```

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(), 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://alexpghayes.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: $1 / \lambda$

Variance: $1 / \lambda^2$

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()`, `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, df2, 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://alexpghayes.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()`, `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

Approximates an empirical distribution with a theoretical one

Usage

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

Arguments

`d`	A probability distribution object such as those created by a call to `Bernoulli()`, `Beta()`, or `Binomial()`.
`x`	A vector of data to compute the likelihood.
`...`	Unused. 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://alexpghayes.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()`, `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, 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://alexpghayes.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(), 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://alexpghayes.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()`, `HurdlePoisson()`, `HyperGeometric()`, `Multinomial()`, `NegativeBinomial()`, `Poisson()`, `ZIPoisson()`

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 `\link{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://alexpghayes.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(), 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://alexpghayes.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(), 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://alexpghayes.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()`, `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))
```

`HurdlePoisson` *Create a hurdle Poisson distribution*

Description

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

Usage

```
HurdlePoisson(lambda, pi)
```

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://alexpghayes.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 $k > 0$

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

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

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

$$E(e^{tX}) = \frac{\pi}{1 - e^{-\lambda}} \cdot e^{\lambda(e^t - 1)}$$

Value

A HurdlePoisson object.

See Also

Other discrete distributions: Bernoulli(), Binomial(), Categorical(), Geometric(), HyperGeometric(), Multinomial(), NegativeBinomial(), Poisson(), ZIPoisson()

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 = 0.75))

## cdf() and quantile() are inverses for each other
cdf(X, quantile(X, 0.3))
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, n, k)
```

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://alexpghayes.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()`, `HurdlePoisson()`, `Multinomial()`, `NegativeBinomial()`, `Poisson()`, `ZIPoisson()`

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_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)
```

`likelihood` *Compute the likelihood of a probability distribution given data*

Description

Compute the likelihood of a probability distribution given data

Usage

```
likelihood(d, x, ...)
```

Arguments

`d`	A probability distribution object such as those created by a call to `Bernoulli()`, `Beta()`, or `Binomial()`.
`x`	A vector of data to compute the likelihood.
`...`	Unused. Unevaluated arguments will generate a warning to catch mispellings or other possible errors.

Value

the likelihood

Examples

```
X <- Normal()

likelihood(X, c(-1, 0, 0, 0, 3))
```

`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://alexpghayes.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()`, `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://alexpghayes.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 $\mathsf{p} = p$.

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()`, `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

Compute the log-likelihood of a probability distribution given data

Usage

```
log_likelihood(d, x, ...)
```

Arguments

d	A probability distribution object such as those created by a call to `Bernoulli()`, `Beta()`, or `Binomial()`.
x	A vector of data to compute the likelihood.
...	Unused. Unevaluated arguments will generate a warning to catch mispellings or other possible errors.

Value

the log-likelihood

Examples

```
X <- Normal()

log_likelihood(X, c(-1, 0, 0, 0, 3))
```

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, p)
```

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://alexpghayes.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()`, `HurdlePoisson()`, `HyperGeometric()`, `NegativeBinomial()`, `Poisson()`, `ZIPoisson()`

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, 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://alexpghayes.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()`, `HurdlePoisson()`, `HyperGeometric()`, `Multinomial()`, `Poisson()`, `ZIPoisson()`

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://alexpghayes.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()`, `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

For discrete distributions, the probability mass function. `pmf()` is an alias.

Usage

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

log_pdf(d, x, ...)

pmf(d, x, ...)
```

Arguments

d	A probability distribution object such as those created by a call to Bernoulli(), Beta(), or 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?
...	Unused. Unevaluated arguments will generate a warning to catch mispellings or other possible errors.

Value

Probabilities corresponding to the vector x.

Examples

```
X <- Normal()

pdf(X, c(1, 2, 3, 4, 5))
pmf(X, c(1, 2, 3, 4, 5))

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, ...)

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

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?
...	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, ...)

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

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?

... 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, ...)

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

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?
...	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, ...)

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

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?
...	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, ...)

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

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?
`...`	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.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, ...)

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

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?
`...`	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, ...)

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

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?
`...`	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, ...)

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

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?
...	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, ...)

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

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?
...	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, ...)

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

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?
...	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, ...)

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

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?
...	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, ...)

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

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?
...	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, ...)

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

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?
...	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, ...)

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

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?
...	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.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, ...)

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

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?
`...`	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 = 0.75))

## cdf() and quantile() are inverses for each other
cdf(X, quantile(X, 0.3))
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, ...)

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

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?
...	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, ...)

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

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?
...	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, ...)

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

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?
...	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, ...)

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

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?
...	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, ...)

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

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?
...	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, ...)

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

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?
`...`	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.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, ...)

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

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?
`...`	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.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, ...)

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

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?
...	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, ...)

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

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?
`...`	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, ...)

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

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?
...	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.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, ...)

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

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?
`...`	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
cdf(X, quantile(X, 0.3))
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)
```

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://alexpghayes.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()`, `HurdlePoisson()`, `HyperGeometric()`, `Multinomial()`, `NegativeBinomial()`, `ZIPoisson()`

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))
```

`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, ...)
```

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.

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. 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**.

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)

## 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)


## 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, ...)
```

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?
`...`	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, ...)
```

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?
`...`	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, ...)
```

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?
`...`	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, ...)
```

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?
`...`	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, ...)
```

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?
`...`	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.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, ...)
```

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?
`...`	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, ...)
```

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?
...	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, ...)
```

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?
`...`	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, ...)
```

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?
`...`	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, ...)
```

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?
`...`	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, ...)
```

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?
`...`	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, ...)
```

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?
`...`	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, ...)
```

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?
`...`	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, ...)
```

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?
`...`	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.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, ...)
```

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?
`...`	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 = 0.75))

## cdf() and quantile() are inverses for each other
cdf(X, quantile(X, 0.3))
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, ...)
```

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?
`...`	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, ...)
```

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?
`...`	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, ...)
```

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?
`...`	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, ...)
```

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?
`...`	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, ...)
```

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?
`...`	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, ...)
```

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?
...	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.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, ...)
```

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?
...	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.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, ...)
```

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?
...	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, ...)
```

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?
`...`	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, ...)
```

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?
...	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, ...)
```

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?
`...`	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.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, ...)
```

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?
`...`	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
cdf(X, quantile(X, 0.3))
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

Draw a random sample from a probability distribution

Usage

```
random(x, n = 1L, drop = TRUE, ...)
```

Arguments

`x`	A probability distribution object such as those created by a call to `Bernoulli()`, `Beta()`, or `Binomial()`.
`n`	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?
`...`	Unused. Unevaluated arguments will generate a warning to catch mispellings or other possible errors.

Value

Random samples drawn from the distriubtion `x`.

Examples

```
X <- Normal()

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.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.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 = 0.75))

## cdf() and quantile() are inverses for each other
cdf(X, quantile(X, 0.3))
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.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.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.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.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
cdf(X, quantile(X, 0.3))
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://alexpghayes.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(), 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))
```

---

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() or 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 display the data
position	Position adjustment, either as a string, or the result of a call to a position adjustment function.
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.
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. 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(). These are often aesthetics, used to set an aesthetic to a fixed value, like colour = "red" or size = 3. They may also be parameters to the paired geom/stat.
stat	The statistical transformation to use on the data for this layer, as a string.

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)
```

Arguments

df	Degrees of freedom. Can be any positive number. Often called ν in textbooks.

Details

We recommend reading this documentation on https://alexpghayes.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()`, `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

Compute the sufficient statistics of a distribution from data

Usage

```
suff_stat(d, x, ...)
```

Arguments

d	A probability distribution object such as those created by a call to `Bernoulli()`, `Beta()`, or `Binomial()`.
x	A vector of data to compute the likelihood.
...	Unused. Unevaluated arguments will generate a warning to catch mispellings or other possible errors.

Value

a named list of sufficient statistics

---

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

Return the support of a distribution

Usage

```
support(d, drop = TRUE)
```

Arguments

`d`	A probability distribution object such as those created by a call to `Bernoulli()`, `Beta()`, or `Binomial()`.
`drop`	logical. Should the result be simplified to a vector if possible?

Value

A vector with two elements indicating the range of the support.

---

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?

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?

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?

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?

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?

Value

A vector of length 2 with the minimum and maximum value of the support.

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?

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?

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?

Value

A vector of length 2 with the minimum and maximum value of the support.

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?

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?

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?

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?

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?

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?

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?

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?

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?

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?

Value

A vector of length 2 with the minimum and maximum value of the support.

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?

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?

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?

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?

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?

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, df, nranges)
```

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://alexpghayes.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()`, `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()`, `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

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.

Usage

```
variance(x, ...)

skewness(x, ...)

kurtosis(x, ...)
```

Arguments

`x`	An object. The package provides methods for probability distribution objects, e.g., those created by `Normal()` or `Beta()` etc.
`...`	Further arguments passed to or from other methods. Unevaluated arguments will generate a warning to catch mispellings or other possible errors.

Value

A numeric scalar

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, scale)
```

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://alexpghayes.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: $\lambda[\Gamma(1 + \frac{2}{k}) - (\Gamma(1 + \frac{1}{k}))^2]$

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(), 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)
```

`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://alexpghayes.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(), HurdlePoisson(), HyperGeometric(), Multinomial(), NegativeBinomial(), Poisson()

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
cdf(X, quantile(X, 0.3))
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