Package: distionary (via r-universe)

October 15, 2026

Title Create and Evaluate Probability Distributions

Version 0.2.0

Description Create and evaluate probability distribution objects from a variety of families or define custom distributions. Automatically compute distributional properties, even when they have not been specified. This package supports statistical modeling and simulations, and forms the core of the probaverse suite of R packages.

License MIT + file LICENSE

Suggests covr, knitr, rmarkdown, testthat (>= 3.0.0), tibble

Config/testthat/edition 3

Encoding UTF-8

LazyData true

Roxygen list (markdown = TRUE)

RoxygenNote 7.3.3

Imports checkmate, discretes (>= 0.1.1), lifecycle, rlang, stats, vctrs

VignetteBuilder knitr

URL https://distionary.probaverse.com/, https://github.com/probaverse/distionary

BugReports https://github.com/probaverse/distionary/issues

Depends R (>= 3.5)

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

Date/Publication 2026-09-15 07:34:43 UTC

RemoteUrl https://github.com/probaverse/distionary

RemoteRef main

RemoteSha 3c301c661786b92526c462dc334511fa267e8e84

Contents

distionary-package . . . 3
atoms . . . 5
distribution . . . 6
dst_bern . . . 8
dst_beta . . . 9
dst_binom . . . 10
dst_cauchy . . . 10
dst_chisq . . . 11
dst_degenerate . . . 11
dst_empirical . . . 12
dst_exp . . . 13
dst_f . . . 14
dst_finite . . . 15
dst_gamma . . . 15
dst_geom . . . 16
dst_gev . . . 16
dst_gp . . . 17
dst_gumbel . . . 18
dst_hyper . . . 19
dst_lnorm . . . 19
dst_lp3 . . . 20
dst_nbinom . . . 20
dst_norm . . . 21
dst_null . . . 22
dst_pearson3 . . . 23
dst_pois . . . 23
dst_t . . . 24
dst_unif . . . 24
dst_weibull . . . 25
empty_support . . . 25
eval_cdf . . . 26
eval_chf . . . 27
eval_density . . . 28
eval_hazard . . . 29
eval_odds . . . 30
eval_pmf . . . 31
eval_property . . . 32
eval_quantile . . . 33
eval_return . . . 35
eval_survival . . . 36
is_support . . . 37
kurtosis . . . 37
median.dst . . . 39
parameters . . . 39
pgev . . . 40
pgp . . . 41

plot.dst . . . 42
plp3 . . . 43
ppearson3 . . . 44
pretty_name . . . 45
prob_left . . . 45
range.dst . . . 46
realise . . . 47
scalar . . . 48
support . . . 49
support-construction . . . 50
support_add_atoms . . . 51
support_contains . . . 52
support_restrict . . . 53
support_transform . . . 54
support_union . . . 56
vtype . . . 57

Index **58**

`distionary-package` *distionary: Create and Evaluate Probability Distributions*

Description

Create and evaluate probability distribution objects from a variety of families or define custom distributions. Automatically compute distributional properties, even when they have not been specified. This package supports statistical modeling and simulations, and forms the core of the probaverse suite of R packages.

Overview

The **distionary** package provides a comprehensive framework for working with probability distributions in R. With **distionary**, you can:

1. Specify probability distributions from common families or create custom distributions.
2. Evaluate distributional properties and representations.
3. Access distributional calculations even when they're not directly specified.

The main purpose of **distionary** is to implement a distribution object that powers the wider probaverse ecosystem for making probability distributions that are representative of your data.

Creating Distributions

Use the `dst_*()` family of functions to create distributions from common families:

- `dst_norm()`, `dst_exp()`, `dst_unif()`, etc. are some continuous distributions.
- `dst_pois()`, `dst_binom()`, `dst_geom()`, etc. are some discrete distributions.
- `dst_empirical()` is useful for creating a non-parametric distribution from data.

You can also make your own distribution using the `distribution()` function, which allows you to specify any combination of distributional representations and properties. For this version of **distionary**, the CDF and density/PMF are required in order to access all functionality.

Evaluating Distributions

A distribution's *representations* are functions that fully describe the distribution. They can be accessed with the `eval_*()` functions. For example, `eval_cdf()` and `eval_quantile()` invoke the distribution's cumulative distribution function (CDF) and quantile function.

Other properties of the distribution can be calculated by functions of the property's name, such as `mean()` and `range()`.

Random Samples

Generate random samples from a distribution using `realise()`.

Getting Started

New users should start with the package vignettes:

- `vignette("specify", package = "distionary")` - Learn how to specify distributions.
- `vignette("evaluate", package = "distionary")` - Learn how to evaluate distributions.

Author(s)

Maintainer: Vincenzo Coia <vincenzo.coia@gmail.com> (ORCID) [copyright holder]

Other contributors:

- Amogh Joshi [contributor]
- Shuyi Tan [contributor]
- Zhipeng Zhu [contributor]
- olivroy (GitHub contributor) [contributor]

See Also

Useful links:

- https://distionary.probaverse.com/
- https://github.com/probaverse/distionary
- Report bugs at https://github.com/probaverse/distionary/issues

Examples

```
# Create a Poisson distribution.
poisson <- dst_pois(lambda = 1.5)
poisson

# Evaluate the probability mass function.
eval_pmf(poisson, at = 0:4)
```

```
plot(poisson)

# Get distribution properties.
mean(poisson)
variance(poisson)

# Create a continuous distribution (Normal).
normal <- dst_norm(mean = 0, sd = 1)

# Evaluate quantiles.
eval_quantile(normal, at = c(0.025, 0.5, 0.975))

# Create a custom distribution.
my_dist <- distribution(
  density = function(x) ifelse(x >= 0 & x <= 1, 2 * (1 - x), 0),
  cdf = function(x) ifelse(x >= 0 & x <= 1, 1 - (1 - x)^2, 0),
  .support = continuous(c(0, 1)),
  .name = "Linear"
)
plot(my_dist)
plot(my_dist, "cdf")

# Even without specifying all properties, they can still be computed.
mean(my_dist)
```

`atoms` *What a Support Is Made Of*

Description

Take a support apart: `atoms()` gives the points it places mass on, and `regions()` gives the intervals it spreads mass across. Each accepts a support object or a distribution.

Usage

```
atoms(x)

regions(x)
```

Arguments

`x`	A support object or a distribution.

Details

These are the inverses of the constructors. `discrete()` builds a support out of atoms and `atoms()` gives them back; `continuous()` builds one out of regions and `regions()` gives those back. So the discrete part of a support is `discrete(atoms(x))`, and its continuous part is `continuous(regions(x))`.

A support with no atoms gives an empty `discretes` object rather than nothing, and one with no regions gives a matrix of no rows, so neither has to be guarded against before being used. That is a definite answer: the support says there is no part of that kind.

`dst_null()` is different. It has no support at all, so there is nothing to take apart and nothing is known — both give `NULL`, as `support()` does for it, rather than claiming it has no atoms. This mirrors `range()`, which answers `c(NA, NA)` for it instead of refusing.

Value

For `atoms()`, a `discretes` object. For `regions()`, a two-column numeric matrix of intervals (`lower`, `upper`), one row each.

See Also

Other Support: `empty_support()`, `is_support()`, `support()`, `support-construction`

Examples

```
atoms(mixed(discrete = 0, continuous = c(0, Inf)))
regions(continuous(c(0, 1), c(3, 4)))

# Either part can be put back together into a support of its own.
s <- mixed(discrete = c(0, 5), continuous = c(0, 10))
discrete(atoms(s))
continuous(regions(s))
```

`distribution` *Build a Distribution Object*

Description

Make a distribution object by specifying properties (e.g., cdf, density, mean, etc.). Some properties, if not included, will be calculated based on other properties that are included (e.g., quantile from cdf; variance from standard deviation). A list of these representations can be found in the details.

Usage

```
distribution(
  ...,
  .support = NULL,
  .vtype = NULL,
  .name = NULL,
  .parameters = list()
)

is_distribution(object)

is.distribution(object)
```

Arguments

`...`	Name-value pairs for defining the distribution.
`.support`	**Required.** Where the distribution places probability, built with `discrete()`, `continuous()` or `mixed()`. A bare `discretes` object is also accepted, and treated as `discrete()`. See Details.
`.vtype`	**[Defunct]** Removed in favour of `.support`, and now an error. See Details.
`.name`	A name to give to the distribution. Can be any character vector of length 1.
`.parameters`	A named list with one entry per distribution parameter, each of which can be any data type. In this version of distionary, these parameters are only stored for the benefit of the user to know what distribution they are working with; the code never looks at these parameters to inform its calculations. This is anticipated to change in a future version of distionary.
`object`	Object to be tested

Details

The support:

Every distribution has to say where it places probability, and `.support` is how. It is the one thing distionary asks for rather than working out: a CDF does hold the answer, its jumps being the atoms and its flattening out marking where the distribution ends, but recovering that numerically means hunting for discontinuities in a function that can only be sampled. The estimate would be worst for small atoms and long tails, which are the cases where it matters most.

Declared instead, it is exact, and the difference shows: quantiles at probability 0 and 1 are read off rather than searched for in the numerical tail, atoms are located exactly, and moments can be decomposed. It is the same bargain as declaring atoms — a little more to say up front, in exchange for exact answers rather than approximate ones. The "The Support of a Distribution" vignette covers what a support is and how to build one.

The variable type (`vtype()`) and the `range()` follow from the support, so neither can be given here; see the property list below.

`.vtype` used to take a string such as `"continuous"` and is now defunct. A variable type cannot stand in for a support: `"discrete"` does not say *which* points carry mass, and `"continuous"` does not say over what region, so there is no translating one into the other, and a guess would be quietly wrong rather than an error. The argument is kept only so that old code gets a message saying what to do, rather than `unused argument`.

Properties:

Currently, the CDF (`cdf`) is required to be specified, along with the PMF (`pmf`) for discrete distributions and density (`density`) for continuous distributions. Otherwise, the full extent of distribution properties will not be accessible.

A distributional representation is a function that fully describes the distribution. Besides `cdf`, `density`, and `pmf`, other options understood by `distionary` include:

- `survival`: the survival function, or one minus the cdf.
- `hazard`: the hazard function, for continuous variables only.
- `chf`: the cumulative hazard function, for continuous variables only.
- `quantile`: the quantile function, or left-inverse of the cdf.

- `realise` or `realize`: a function that takes an integer and generates a vector of that many random draws from the distribution.
- `odds`: for discrete variables, the probability odds function (pmf / (1 - pmf))
- `return`: the quantiles associated with the provided return periods, where events are exceedances.

All functions should be vectorized.

Other properties that are understood by `distionary` include:

- `mean`, `stdev`, `variance`, `skewness`, `median` are self-explanatory.
- `kurtosis_exc` and `kurtosis` are the distribution's excess kurtosis and regular kurtosis.

`range` and `vtype` are properties too, and `eval_property()` reads them like any other, but they cannot be given here: the support determines both, and a stated one could disagree with it. A name distionary does not know is simply kept, retrievable with `eval_property()` and otherwise unused.

Value

A distribution object.

Examples

```
linear <- distribution(
  density = function(x) {
    d <- 2 * (1 - x)
    d[x < 0 | x > 1] <- 0
    d
  },
  cdf = function(x) {
    p <- 2 * x * (1 - x / 2)
    p[x < 0] <- 0
    p[x > 1] <- 1
    p
  },
  .support = continuous(c(0, 1)),
  .name = "My Linear",
  .parameters = list(could = "include", anything = data.frame(x = 1:10))
)

# Inspect
linear

# Plot
plot(linear)
```

`dst_bern` *Bernoulli Distribution*

Description

Makes a Bernoulli distribution, representing the outcome of a single trial with a given success probability.

Usage

```
dst_bern(prob)
```

Arguments

`prob`	Probability of success; single numeric between 0 and 1.

Value

A Bernoulli distribution.

Examples

```
dst_bern(0.3)
```

`dst_beta` *Beta Distribution*

Description

Makes a Beta distribution.

Usage

```
dst_beta(shape1, shape2)
```

Arguments

`shape1`, `shape2`	Shape parameters of the distribution; single positive numerics.

Value

A Beta distribution.

Examples

```
dst_beta(2, 3)
```

dst_binom *Binomial Distribution*

Description

Makes a Binomial distribution, representing the number of successes in a fixed number of independent trials.

Usage

```
dst_binom(size, prob)
```

Arguments

size	Number of trials; single positive integer.
prob	Success probability of each trial; single numeric between 0 and 1.

Value

A binomial distribution.

Examples

```
dst_binom(10, 0.6)
```

dst_cauchy *Cauchy Distribution*

Description

Makes a Cauchy distribution.

Usage

```
dst_cauchy(location, scale)
```

Arguments

location	Location parameter; single numeric.
scale	Scale parameter; single positive numeric.

Value

A Cauchy distribution.

Examples

```
d <- dst_cauchy(0, 1)

# Moments do not exist for the Cauchy distribution.
mean(d)
variance(d)
```

`dst_chisq` *Chi-Squared Distribution*

Description

Makes a Chi-Squared distribution.

Usage

```
dst_chisq(df)
```

Arguments

`df`	degrees of freedom parameter; single positive numeric.

Value

A Chi-Squared distribution

Examples

```
dst_chisq(3)
```

`dst_degenerate` *Degenerate Distribution*

Description

A degenerate distribution assigns a 100% probability to one outcome.

Usage

```
dst_degenerate(location)
```

Arguments

`location`	Outcome of the distribution; single positive numeric.

Value

A degenerate distribution

Examples

```
d <- dst_degenerate(5)
realise(d)
variance(d)
```

---

dst_empirical *Empirical Distribution*

Description

An empirical distribution is a non-parametric way to estimate a distribution using data. By default, it assigns equal probability to all observations (this can be overridden with the `weights` argument). Identical to `dst_finite()` with NA handling and with weights not needing to add to 1.

Usage

```
dst_empirical(
  y,
  weights = 1,
  data = NULL,
  na_action_y = c("null", "drop", "fail"),
  na_action_w = c("null", "drop", "fail")
)
```

Arguments

`y`	<data-masking> Numeric vector representing the potential outcomes of the distribution.
`weights`	<data-masking> Numeric vector of weights corresponding to to the outcomes `y`. These will be scaled so that they add up to 1.
`data`	Optionally, a data frame to compute `y` and `weights` from. `NULL` if data are not coming from a data frame (the default).
`na_action_y`, `na_action_w`	What should be done with `NA` entries in `y` and `weights`? Character vector of length 1: one of `"fail"`, `"null"` (default), or `"drop"`. See details.

Details

`y` and `weights` are recycled to have the same length, but only if one of them has length 1 (via `vctrs::vec_recycle_common()`).

`na_action_y` and `na_action_w` specify the NA action for `y` and `weights`. Options are, in order of precedence:

- "fail": Throw an error in the presence of NA.
- "null": Return a Null distribution (dst_null()) in the presence of NA.
- "drop" (the default for na_action_w): Remove outcome-weight pairs having an NA value in the specified vector.

Value

A finite distribution. If only one outcome, returns a degenerate distribution. Returns a Null distribution if NA values are present and "null" is specified as an NA action.

See Also

dst_finite()

Examples

```
t <- -2:7
dst_empirical(t)

# Using a data frame
df <- data.frame(time = c(NA, NA, t))
dst_empirical(time * 60, data = df)  # Null, since `NA` in `time`.

# Drop NA `time` values.
dst_empirical(time * 60, data = df, na_action_y = "drop")

# Weights explicit. Zero-weight outcomes ("-120") are gone.
df$w <- c(1, 1, 0:9)
dst_empirical(time * 60, w, data = df, na_action_y = "drop")

# "Null" takes precedence over "drop".
df$w <- c(NA, NA, 0:9)
df$time[1] <- -3
df$time[12] <- NA
dst_empirical(time, w, data = df, na_action_w = "null", na_action_y = "drop")
dst_empirical(time, w, data = df, na_action_w = "drop", na_action_y = "null")
dst_empirical(time, w, data = df, na_action_w = "drop", na_action_y = "drop")
```

---

dst_exp *Exponential Distribution*

Description

Makes an Exponential distribution.

Usage

```
dst_exp(rate)
```

Arguments

`rate`	Rate parameter; single positive numeric.

Value

An Exponential distribution.

Examples

```
dst_exp(1)
```

---

dst_f *F Distribution*

Description

Makes an F distribution.

Usage

```
dst_f(df1, df2)
```

Arguments

`df1`, `df2`	Degrees of freedom of the numerator and denominator, both single positive numerics.

Value

An F distribution.

Examples

```
dst_f(2, 3)
```

dst_finite *Finite Distribution*

Description

Makes a finite distribution, which is a distribution with a finite number of possible outcomes.

Usage

```
dst_finite(outcomes, probs)
```

Arguments

outcomes	Numeric vector representing the potential outcomes of the distribution.
probs	Numeric vector of probabilities corresponding to the outcomes in outcomes. Must not be negative and must sum to 1.

Value

A distribution with finite outcomes.

See Also

dst_empirical()

Examples

```
dst_finite(2:5, probs = 1:4 / 10)
```

dst_gamma *Gamma Distribution*

Description

Makes a Gamma distribution.

Usage

```
dst_gamma(shape, rate)
```

Arguments

shape	Shape parameter; single positive numeric.
rate	Rate parameter; single positive numeric.

Value

A Gamma distribution.

Examples

```
dst_gamma(2, 1)
```

---

dst_geom *Geometric Distribution*

Description

Makes a Geometric distribution, corresponding to the number of failures in a sequence of independent trials before observing a success.

Usage

```
dst_geom(prob)
```

Arguments

prob	Probability of success in each trial; single numeric between 0 and 1.

Value

A Geometric distribution.

Examples

```
d <- dst_geom(0.4)

# This version of the Geometric distribution does not count the success.
range(d)
```

---

dst_gev *Generalised Extreme Value Distribution*

Description

Makes a Generalised Extreme Value (GEV) distribution, which is the limiting distribution of the maximum.

Usage

```
dst_gev(location, scale, shape)
```

Arguments

`location`	Location parameter; single numeric.
`scale`	Scale parameter; single positive numeric.
`shape`	Shape parameter; single numeric. This is also the extreme value index, so that `shape > 0` is heavy tailed, and `shape < 0` is short-tailed.

Value

A GEV distribution.

Examples

```
# Short-tailed example
short <- dst_gev(0, 1, -1)
range(short)
mean(short)

# Heavy-tailed example
heavy <- dst_gev(0, 1, 1)
range(heavy)
mean(heavy)

# Light-tailed example (a Gumbel distribution)
light <- dst_gev(0, 1, 0)
range(light)
mean(light)
```

`dst_gp` *Generalised Pareto Distribution*

Description

Makes a Generalized Pareto (GP) distribution, corresponding to the limiting distribution of excesses over a threshold.

Usage

```
dst_gp(scale, shape)
```

Arguments

`scale`	Scale parameter; single positive numeric.
`shape`	Shape parameter; single positive numeric. This is also the extreme value index, so that `shape > 0` is heavy tailed, and `shape < 0` is short-tailed.

Value

A Generalised Pareto Distribution.

Examples

```
# Short-tailed example
short <- dst_gp(1, -1)
range(short)
mean(short)

# Heavy-tailed example
heavy <- dst_gp(1, 1)
range(heavy)
mean(heavy)

# Light-tailed example (a Gumbel distribution)
light <- dst_gp(1, 0)
range(light)
mean(light)
```

`dst_gumbel` *Gumbel Distribution*

Description

Makes a Gumbel distribution, which is a special case of the Generalised Extreme Value (GEV) distribution when the shape parameter is 0.

Usage

```
dst_gumbel(location, scale)
```

Arguments

`location`	Location parameter; single numeric.
`scale`	Scale parameter; single positive numeric.

Value

A Gumbel distribution.

Examples

```
dst_gumbel(0, 1)
```

dst_hyper *Hypergeometric Distribution*

Description

Creates a Hypergeometric distribution. The parameterization used here is the same as for `stats::phyper()`, where the outcome can be thought of as the number of red balls drawn from an urn of coloured balls, using a scoop that holds a fixed number of balls.

Usage

```
dst_hyper(m, n, k)
```

Arguments

`m`	The number of red balls in the urn; single positive integer.
`n`	The number of non-red balls in the urn; single positive integer.
`k`	the number of balls drawn from the urn (between 0 and `m + n`); single positive integer.

Value

A Hypergeometric distribution.

Examples

```
dst_hyper(15, 50, 10)
```

dst_lnorm *Log Normal Distribution*

Description

Makes a Log Normal distribution, which is the distribution of the exponential of a Normally distributed random variable.

Usage

```
dst_lnorm(meanlog, sdlog)
```

Arguments

`meanlog`	Mean of the log of the random variable; single numeric.
`sdlog`	Standard deviation of the log of the random variable; single positive numeric.

Value

A Log Normal distribution.

Examples

```
dst_lnorm(0, 1)
```

---

dst_lp3 *Log Pearson Type III distribution*

Description

Makes a Log Pearson Type III distribution, which is the distribution of the exponential of a random variable following a Pearson Type III distribution.

Usage

```
dst_lp3(meanlog, sdlog, skew)
```

Arguments

meanlog	Mean of the log of the random variable; single numeric.
sdlog	Standard deviation of the log of the random variable; single positive numeric.
skew	Skewness of the log of the random variable; single numeric.

Value

A Log Pearson Type III distribution.

Examples

```
dst_lp3(0, 1, 1)
```

---

dst_nbinom *Negative binomial Distribution*

Description

Makes a Negative Binomial distribution, corresponding to the number of failures in a sequence of independent trials until a given number of successes are observed.

Usage

```
dst_nbinom(size, prob)
```

Arguments

size	Number of successful trials; single positive numeric.
prob	Probability of a successful trial; single numeric between 0 and 1.

Value

A Negative Binomial distribution.

Examples

```
d <- dst_nbinom(10, 0.5)

# This version of the Negative Binomial distribution does not count
# the successes.
range(d)
```

---

dst_norm *Normal (Gaussian) Distribution*

Description

Makes a Normal (Gaussian) distribution.

Usage

```
dst_norm(mean, sd)
```

Arguments

mean	Mean of the distribution. Single numeric.
sd	Standard deviation of the distribution. Single positive numeric.

Value

A Normal distribution.

Examples

```
dst_norm(0, 1)
```

dst_null *Null Distribution*

Description

Sometimes it's convenient to work with a distribution object that is akin to a missing value. This is especially true when programmatically outputting distributions, such as when a distribution fails to fit to data. This function makes such a distribution object. It always evaluates to NA.

Usage

```
dst_null()
```

Details

The Null distribution is the missing value of the distribution world, and every query about it answers NA in whatever type that query returns: NA_real_ from mean() and the eval_*() functions, NA_character_ from vtype(), c(NA, NA) from range(), and no support at all — support() returns NULL, R's absent-object value. It is also the one distribution that is.na() finds; see length.dst().

Because of that it is assembled with the package's low-level constructor rather than through distribution(). A Null distribution cannot satisfy what distribution() asks of a real one, since it has nothing to declare; building it here keeps that bypass internal, so a distribution with no support cannot be made through the front door.

Value

A Null distribution.

Examples

```
x <- dst_null()
mean(x)
eval_pmf(x, at = 1:10)

# It is the distribution that `is.na()` finds.
is.na(x)
is.na(dst_norm(0, 1))

# Everything about it is missing, including its support.
vtype(x)
range(x)
support(x)
```

dst_pearson3 *Pearson Type III distribution*

Description

Makes a Pearson Type III distribution, which is a Gamma distribution, but shifted.

Usage

```
dst_pearson3(location, scale, shape)
```

Arguments

`location`	Location parameter, specifying the boundary of the distribution; single numeric. It is the left endpoint when `shape` is positive and the right endpoint when `shape` is negative.
`scale`	Scale parameter of the Gamma distribution; single positive numeric.
`shape`	Shape parameter of the Gamma distribution; single numeric. A negative value gives the distribution reflected about `location`: the Pearson Type III with negative skewness, upper-bounded at `location`.

Value

A Pearson Type III distribution.

Examples

```
dst_pearson3(1, 1, 1)
# A negative shape reflects the distribution about `location`:
dst_pearson3(1, 1, -1)
```

dst_pois *Poisson Distribution*

Description

Makes a Poisson distribution.

Usage

```
dst_pois(lambda)
```

Arguments

`lambda`	Mean of the Poisson distribution; single positive numeric.

Value

A Poisson distribution.

Examples

```
dst_pois(1)
```

---

dst_t *Student t Distribution*

Description

Makes a Student t distribution.

Usage

```
dst_t(df)
```

Arguments

df	Degrees of freedom; single positive numeric.

Value

A Student t distribution.

Examples

```
dst_t(3)
```

---

dst_unif *Uniform Distribution*

Description

Makes a Uniform distribution.

Usage

```
dst_unif(min, max)
```

Arguments

min, max	Minimum and maximum of the distribution. Single numerics.

Value

A Uniform distribution.

Examples

```
dst_unif(0, 1)
```

---

dst_weibull *Weibull Distribution*

Description

Makes a Weibull distribution.

Usage

```
dst_weibull(shape, scale)
```

Arguments

shape	Shape parameter; single positive numeric.
scale	Scale parameter; single positive numeric.

Value

A Weibull distribution.

Examples

```
dst_weibull(1, 1)
```

---

empty_support *The Empty Support*

Description

A support containing nothing: no atoms and no continuous part.

Usage

```
empty_support()
```

Details

No distribution has an empty support — probability has to go somewhere — and `distribution()` rejects one. It exists so that operations on supports are *closed*: restricting a support to a region it does not reach has to return something, and that something is the empty support. It is also the identity for taking unions.

Its variable type is `"empty"`, which is a different claim from `"unknown"`. Empty says there is nowhere to place probability; unknown says nobody specified where.

Value

A support object (class `"support"`) with both parts empty.

See Also

`is_empty_support()` to test for it, `discrete()`, `continuous()`, and `mixed()` for supports a distribution can actually have.

Other Support: `atoms()`, `is_support()`, `support()`, `support-construction`

Examples

```
empty_support()
is_empty_support(empty_support())

# It is also what any of the constructors gives when handed nothing.
continuous(numeric(0))
discrete(numeric(0))
mixed()
```

---

eval_cdf *Cumulative Distribution Function*

Description

Access a distribution's cumulative distribution function (cdf).

Usage

```
eval_cdf(distribution, at)

enframe_cdf(..., at, arg_name = ".arg", fn_prefix = "cdf", sep = "_")
```

Arguments

`distribution, ...`	A distribution, or possibly multiple distributions in the case of `...`.
`at`	Vector of values to evaluate the representation at.

arg_name	For enframe_, name of the column containing the function arguments. Length 1 character vector.
fn_prefix	For enframe_, name of the function to appear in the column(s). Length 1 character vector.
sep	When enframe'ing more than one distribution, the character that will be separating the fn_name and the distribution name. Length 1 character vector.

Value

The evaluated representation in vector form (for eval_) with length matching the length of at, and data frame or tibble form (for enframe_) with number of rows matching the length of at. The at input occupies the first column, named .arg by default, or the specification in arg_name; the evaluated representations for each distribution in ... go in the subsequent columns (one column per distribution). For a single distribution, this column is named according to the representation by default (cdf, survival, quantile, etc.), or the value in fn_prefix. For multiple distributions, unnamed distributions are auto-named, and columns are named <fn_prefix><sep><distribution_name> (e.g., cdf_distribution1).

See Also

Other distributional representations: eval_chf(), eval_density(), eval_hazard(), eval_odds(), eval_pmf(), eval_quantile(), eval_return(), eval_survival()

Examples

```
d1 <- dst_unif(0, 4)
d2 <- dst_pois(1.1)
eval_cdf(d1, at = 0:4)
enframe_cdf(d1, at = 0:4)
enframe_cdf(d1, d2, at = 0:4)
enframe_cdf(model1 = d1, model2 = d2, at = 0:4)
enframe_cdf(
  model1 = d1, model2 = d2, at = 0:4, arg_name = "value"
)
```

---

eval_chf *Cumulative Hazard Function*

Description

Access a distribution's cumulative hazard function (chf).

Usage

```
eval_chf(distribution, at)

enframe_chf(..., at, arg_name = ".arg", fn_prefix = "chf", sep = "_")
```

Arguments

distribution, ...	A distribution, or possibly multiple distributions in the case of ....
at	Vector of values to evaluate the representation at.
arg_name	For enframe_, name of the column containing the function arguments. Length 1 character vector.
fn_prefix	For enframe_, name of the function to appear in the column(s). Length 1 character vector.
sep	When enframe'ing more than one distribution, the character that will be separating the fn_name and the distribution name. Length 1 character vector.

Value

The evaluated representation in vector form (for eval_) with length matching the length of at, and data frame or tibble form (for enframe_) with number of rows matching the length of at. The at input occupies the first column, named .arg by default, or the specification in arg_name; the evaluated representations for each distribution in ... go in the subsequent columns (one column per distribution). For a single distribution, this column is named according to the representation by default (cdf, survival, quantile, etc.), or the value in fn_prefix. For multiple distributions, unnamed distributions are auto-named, and columns are named <fn_prefix><sep><distribution_name> (e.g., cdf_distribution1).

See Also

Other distributional representations: eval_cdf(), eval_density(), eval_hazard(), eval_odds(), eval_pmf(), eval_quantile(), eval_return(), eval_survival()

Examples

```
d <- dst_unif(0, 4)
eval_chf(d, at = 0:4)
enframe_chf(d, at = 0:4)
```

---

eval_density *Probability Density Function*

Description

Access a distribution's probability density function (pdf).

Usage

```
eval_density(distribution, at)

enframe_density(..., at, arg_name = ".arg", fn_prefix = "density", sep = "_")
```

Arguments

distribution, ...	A distribution, or possibly multiple distributions in the case of ... .
at	Vector of values to evaluate the representation at.
arg_name	For enframe_, name of the column containing the function arguments. Length 1 character vector.
fn_prefix	For enframe_, name of the function to appear in the column(s). Length 1 character vector.
sep	When enframe'ing more than one distribution, the character that will be separating the fn_name and the distribution name. Length 1 character vector.

Value

The evaluated representation in vector form (for eval_) with length matching the length of at, and data frame or tibble form (for enframe_) with number of rows matching the length of at. The at input occupies the first column, named .arg by default, or the specification in arg_name; the evaluated representations for each distribution in ... go in the subsequent columns (one column per distribution). For a single distribution, this column is named according to the representation by default (cdf, survival, quantile, etc.), or the value in fn_prefix. For multiple distributions, unnamed distributions are auto-named, and columns are named <fn_prefix><sep><distribution_name> (e.g., cdf_distribution1).

See Also

Other distributional representations: eval_cdf(), eval_chf(), eval_hazard(), eval_odds(), eval_pmf(), eval_quantile(), eval_return(), eval_survival()

Examples

```
d <- dst_unif(0, 4)
eval_density(d, at = 0:4)
enframe_density(d, at = 0:4)
```

---

eval_hazard *Hazard Function*

Description

Access a distribution's hazard function.

Usage

```
eval_hazard(distribution, at)

enframe_hazard(..., at, arg_name = ".arg", fn_prefix = "hazard", sep = "_")
```

Arguments

distribution, ...
: A distribution, or possibly multiple distributions in the case of ....

at
: Vector of values to evaluate the representation at.

arg_name
: For enframe_, name of the column containing the function arguments. Length 1 character vector.

fn_prefix
: For enframe_, name of the function to appear in the column(s). Length 1 character vector.

sep
: When enframe'ing more than one distribution, the character that will be separating the fn_name and the distribution name. Length 1 character vector.

Value

The evaluated representation in vector form (for eval_) with length matching the length of at, and data frame or tibble form (for enframe_) with number of rows matching the length of at. The at input occupies the first column, named .arg by default, or the specification in arg_name; the evaluated representations for each distribution in ... go in the subsequent columns (one column per distribution). For a single distribution, this column is named according to the representation by default (cdf, survival, quantile, etc.), or the value in fn_prefix. For multiple distributions, unnamed distributions are auto-named, and columns are named <fn_prefix><sep><distribution_name> (e.g., cdf_distribution1).

See Also

Other distributional representations: eval_cdf(), eval_chf(), eval_density(), eval_odds(), eval_pmf(), eval_quantile(), eval_return(), eval_survival()

Examples

```
d <- dst_unif(0, 4)
eval_hazard(d, at = 0:4)
enframe_hazard(d, at = 0:4)
```

---

eval_odds *Odds Function*

Description

Access a distribution's odds function. The odds of an event having probability p is p / (1 - p).

Usage

```
eval_odds(distribution, at)

enframe_odds(..., at, arg_name = ".arg", fn_prefix = "odds", sep = "_")
```

Arguments

distribution, ...	A distribution, or possibly multiple distributions in the case of ... .
at	Vector of values to evaluate the representation at.
arg_name	For enframe_, name of the column containing the function arguments. Length 1 character vector.
fn_prefix	For enframe_, name of the function to appear in the column(s). Length 1 character vector.
sep	When enframe'ing more than one distribution, the character that will be separating the fn_name and the distribution name. Length 1 character vector.

Value

The evaluated representation in vector form (for eval_) with length matching the length of at, and data frame or tibble form (for enframe_) with number of rows matching the length of at. The at input occupies the first column, named .arg by default, or the specification in arg_name; the evaluated representations for each distribution in ... go in the subsequent columns (one column per distribution). For a single distribution, this column is named according to the representation by default (cdf, survival, quantile, etc.), or the value in fn_prefix. For multiple distributions, unnamed distributions are auto-named, and columns are named <fn_prefix><sep><distribution_name> (e.g., cdf_distribution1).

See Also

Other distributional representations: eval_cdf(), eval_chf(), eval_density(), eval_hazard(), eval_pmf(), eval_quantile(), eval_return(), eval_survival()

Examples

```
d <- dst_pois(1)
eval_pmf(d, at = c(1, 2, 2.5))
eval_odds(d, at = c(1, 2, 2.5))
enframe_odds(d, at = 0:4)
```

---

eval_pmf *Probability Mass Function*

Description

Access a distribution's probability mass function (pmf).

Usage

```
eval_pmf(distribution, at)

enframe_pmf(..., at, arg_name = ".arg", fn_prefix = "pmf", sep = "_")
```

Arguments

distribution, ...	A distribution, or possibly multiple distributions in the case of ... .
at	Vector of values to evaluate the representation at.
arg_name	For enframe_, name of the column containing the function arguments. Length 1 character vector.
fn_prefix	For enframe_, name of the function to appear in the column(s). Length 1 character vector.
sep	When enframe'ing more than one distribution, the character that will be separating the fn_name and the distribution name. Length 1 character vector.

Value

The evaluated representation in vector form (for eval_) with length matching the length of at, and data frame or tibble form (for enframe_) with number of rows matching the length of at. The at input occupies the first column, named .arg by default, or the specification in arg_name; the evaluated representations for each distribution in ... go in the subsequent columns (one column per distribution). For a single distribution, this column is named according to the representation by default (cdf, survival, quantile, etc.), or the value in fn_prefix. For multiple distributions, unnamed distributions are auto-named, and columns are named <fn_prefix><sep><distribution_name> (e.g., cdf_distribution1).

See Also

Other distributional representations: eval_cdf(), eval_chf(), eval_density(), eval_hazard(), eval_odds(), eval_quantile(), eval_return(), eval_survival()

Examples

```
d <- dst_pois(5)
eval_pmf(d, at = c(1, 2, 2.5))
enframe_pmf(d, at = 0:4)
eval_pmf(dst_norm(0, 1), at = -3:3)
```

---

eval_property *Evaluate a distribution*

Description

Evaluate a distribution property. The distribution itself is first searched for the property, and if it can't be found, will attempt to calculate the property from other entries.

Usage

```
eval_property(distribution, entry, ...)
```

Arguments

distribution	Distribution object.
entry	Name of the property, such as "cdf" or "mean". Length 1 character vector.
...	If the property is a function, arguments to the function go here. Need not be named; inserted in the order they appear.

Value

The distribution's property, evaluated. If cannot be evaluated, returns NULL.

Examples

```
d <- distribution(
  cdf = function(x) {
    (x > 0) * pmin(x^2, 1)
  },
  g = 9.81,
  .support = continuous(c(0, 1))
)
eval_property(d, "g")
eval_property(d, "quantile", 1:9 / 10)
eval_property(d, "mean")
eval_property(d, "realise", 10)
eval_property(d, "foofy")
eval_property(d, "foofy", 1:10)
```

---

eval_quantile *Distribution Quantiles*

Description

Access a distribution's quantiles.

Usage

```
eval_quantile(distribution, at)

enframe_quantile(..., at, arg_name = ".arg", fn_prefix = "quantile", sep = "_")
```

Arguments

distribution, ...	A distribution, or possibly multiple distributions in the case of ... .
at	Vector of values to evaluate the representation at.
arg_name	For enframe_, name of the column containing the function arguments. Length 1 character vector.

fn_prefix	For enframe_, name of the function to appear in the column(s). Length 1 character vector.
sep	When enframe'ing more than one distribution, the character that will be separating the fn_name and the distribution name. Length 1 character vector.

Details

The 0- and 1-quantiles are the ends of the distribution's support: the 0-quantile is its lower end and the 1-quantile its upper end. They are read from the support (see support()) rather than computed, so an unbounded distribution gives -Inf and Inf instead of a large finite number found by searching into the tail.

When a quantile function does not exist, the remaining probabilities are found by inverting the CDF by bisection: an interval known to contain the solution is progressively cut in half, moving into whichever half still contains it. The whole vector is solved together — one vectorized CDF evaluation per step rather than one per probability — so evaluating many quantiles at once is considerably faster than one at a time. Because the support says where the atoms (discrete mass points) are, a probability landing inside an atom's jump in the CDF is returned as that atom exactly, rather than approximately. Tolerance is roughly 1e-9 in the quantile value, unless the maximum number of iterations (200) is reached.

Value

The evaluated representation in vector form (for eval_) with length matching the length of at, and data frame or tibble form (for enframe_) with number of rows matching the length of at. The at input occupies the first column, named .arg by default, or the specification in arg_name; the evaluated representations for each distribution in ... go in the subsequent columns (one column per distribution). For a single distribution, this column is named according to the representation by default (cdf, survival, quantile, etc.), or the value in fn_prefix. For multiple distributions, unnamed distributions are auto-named, and columns are named <fn_prefix><sep><distribution_name> (e.g., cdf_distribution1).

See Also

Other distributional representations: eval_cdf(), eval_chf(), eval_density(), eval_hazard(), eval_odds(), eval_pmf(), eval_return(), eval_survival()

Examples

```
d <- dst_unif(0, 4)
eval_quantile(d, at = 1:9 / 10)
enframe_quantile(d, at = 1:9 / 10)
```

eval_return — *Return Level Function*

Description

Compute return levels (quantiles) from a distribution by inputting return periods. The return periods correspond to events that are *exceedances* of a quantile, not non-exceedances.

Usage

```
eval_return(distribution, at)

enframe_return(..., at, arg_name = ".arg", fn_prefix = "return", sep = "_")
```

Arguments

`distribution, ...`	A distribution, or possibly multiple distributions in the case of `...`.
`at`	Vector of return periods >=1.
`arg_name`	For `enframe_`, name of the column containing the function arguments. Length 1 character vector.
`fn_prefix`	For `enframe_`, name of the function to appear in the column(s). Length 1 character vector.
`sep`	When `enframe`'ing more than one distribution, the character that will be separating the `fn_name` and the distribution name. Length 1 character vector.

Details

This function is simply the quantile function evaluated at `1 - 1 / at`.

Value

The evaluated representation in vector form (for `eval_`) with length matching the length of `at`, and data frame or tibble form (for `enframe_`) with number of rows matching the length of `at`. The `at` input occupies the first column, named `.arg` by default, or the specification in `arg_name`; the evaluated representations for each distribution in `...` go in the subsequent columns (one column per distribution). For a single distribution, this column is named according to the representation by default (cdf, survival, quantile, etc.), or the value in `fn_prefix`. For multiple distributions, unnamed distributions are auto-named, and columns are named `<fn_prefix><sep><distribution_name>` (e.g., `cdf_distribution1`).

See Also

Other distributional representations: `eval_cdf()`, `eval_chf()`, `eval_density()`, `eval_hazard()`, `eval_odds()`, `eval_pmf()`, `eval_quantile()`, `eval_survival()`

Examples

```
d <- dst_gp(24, 0.3)
eval_return(d, at = c(2, 25, 100, 200))
```

`eval_survival` *Survival Function*

Description

Access a distribution's survival function.

Usage

```
eval_survival(distribution, at)

enframe_survival(..., at, arg_name = ".arg", fn_prefix = "survival", sep = "_")
```

Arguments

`distribution, ...`	A distribution, or possibly multiple distributions in the case of `...`.
`at`	Vector of values to evaluate the representation at.
`arg_name`	For `enframe_`, name of the column containing the function arguments. Length 1 character vector.
`fn_prefix`	For `enframe_`, name of the function to appear in the column(s). Length 1 character vector.
`sep`	When `enframe`'ing more than one distribution, the character that will be separating the `fn_name` and the distribution name. Length 1 character vector.

Value

The evaluated representation in vector form (for `eval_`) with length matching the length of `at`, and data frame or tibble form (for `enframe_`) with number of rows matching the length of `at`. The `at` input occupies the first column, named `.arg` by default, or the specification in `arg_name`; the evaluated representations for each distribution in `...` go in the subsequent columns (one column per distribution). For a single distribution, this column is named according to the representation by default (cdf, survival, quantile, etc.), or the value in `fn_prefix`. For multiple distributions, unnamed distributions are auto-named, and columns are named `<fn_prefix><sep><distribution_name>` (e.g., `cdf_distribution1`).

See Also

Other distributional representations: `eval_cdf()`, `eval_chf()`, `eval_density()`, `eval_hazard()`, `eval_odds()`, `eval_pmf()`, `eval_quantile()`, `eval_return()`

Examples

```
d <- dst_unif(0, 4)
eval_survival(d, at = 0:4)
enframe_survival(d, at = 0:4)
```

`is_support` *Test for a Support Object*

Description

`is_empty_support()` tests whether a support is the empty one: no atoms and no continuous part. It is `FALSE` for anything that is not a support.

Usage

```
is_support(x)

is_empty_support(x)
```

Arguments

`x`	Object to test.

Value

`TRUE` if `x` is a support object, otherwise `FALSE`.

See Also

Other Support: `atoms()`, `empty_support()`, `support()`, `support-construction`

`kurtosis` *Moments of a Distribution*

Description

Get common moment-related quantities of a distribution: `mean`, `variance`, standard deviation (`stdev`), `skewness`, and `kurtosis` or excess kurtosis (`kurtosis_exc`). If these quantities are not supplied in the distribution's definition, a numerical algorithm may be used.

Usage

```
kurtosis(distribution)

kurtosis_exc(distribution)

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

skewness(distribution)

stdev(distribution)

variance(distribution)
```

Arguments

`x`, `distribution`	Distribution to evaluate.
`...`	When calculating the mean via integration of the quantile function, arguments passed to `stats::integrate()`.

Details

If a moment is not supplied in the distribution's definition, it is computed numerically over the distribution's support: a sum over the atoms (the discrete part) plus integration of the density over the continuous part. An infinite atomic support (such as a Poisson's) is summed by walking outward through its atoms until the tail contribution is negligible.

Value

A single numeric.

Note

When a moment is computed numerically and the underlying sum or integral does not converge — for example, a heavy-tailed distribution whose moment is not finite — the result is `NaN`.

Examples

```
a <- dst_gp(1, 0.5)
b <- dst_unif(0, 1)
c <- dst_norm(3, 4)
mean(a)
variance(b)
kurtosis(c)
kurtosis_exc(c)
```

median.dst *Median of a Distribution*

Description

Finds the median of a distribution.

Usage

```
## S3 method for class 'dst'
median(x, ...)
```

Arguments

x	Distribution to calculate median from.
...	Not used.

Details

Median is calculated as the 0.5-quantile when not found in the distribution. So, when the median is non-unique, the smallest of the possibilities is taken.

Value

Median of a distribution; single numeric.

Examples

```
d <- dst_gamma(3, 3)
median(d)
```

parameters *Parameters of a Distribution*

Description

Get or set the parameters of a distribution, if applicable. See details.

Usage

```
parameters(distribution)

parameters(distribution) <- value
```

Arguments

distribution	Distribution.
value	A list of named parameter values, or NULL.

Details

If a distribution is made by specifying parameter values (e.g., mean and variance for a Normal distribution; shape parameters for a Beta distribution), it is useful to keep track of what these parameters are. This is done by adding `parameters` to the list of objects defining the distribution; for instance, `distribution(parameters = c(shape1 = 1.4, shape2 = 3.4))`. Note that no checks are made to ensure the parameters are valid. It's important to note that, in this version of distionary, manually changing the parameters after the distribution has been created will not change the functionality of the distribution, because the parameters are never referred to when making calculations.

Value

A list of the distribution parameters. More specifically, returns the `"parameters"` entry of the list making up the probability distribution.

Examples

```
a <- dst_beta(1, 2)
parameters(a)

b <- distribution(mean = 5, .support = continuous())
parameters(b)
parameters(b) <- list(t = 7)
parameters(b)
```

---

pgev — *Representations of the Generalized Extreme Value Distribution*

Description

Representations of the Generalized Extreme Value Distribution

Usage

```
pgev(q, location, scale, shape)

qgev(p, location, scale, shape)

dgev(x, location, scale, shape)
```

Arguments

`location`	Location parameter; numeric vector.
`scale`	Scale parameter; positive numeric vector.
`shape`	Shape parameter; numeric vector. This is also the extreme value index, so that `shape > 0` is heavy tailed, and `shape < 0` is short-tailed.
`p`	Vector of probabilities.
`x, q`	Vector of quantiles.

Value

Vector of evaluated GEV distribution, with length equal to the recycled lengths of `q/x/p`, `location`, `scale`, and `shape`.

Examples

```
pgev(1:10, 0, 1, 1)
dgev(1:10, 1:10, 2, 0)
qgev(1:9 / 10, 2, 10, -2)
```

---

pgp — *Representations of the Generalized Pareto Distribution*

Description

Representations of the Generalized Pareto Distribution

Usage

```
pgp(q, scale, shape, lower.tail = TRUE)

qgp(p, scale, shape)

dgp(x, scale, shape)
```

Arguments

`scale`	Vector of scale parameters; positive numeric.
`shape`	Vector of shape parameters; positive numeric.
`lower.tail`	Single logical. If `TRUE`, cdf (default); if `FALSE`, survival function.
`p`	Vector of probabilities.
`x, q`	Vector of quantiles.

Value

Vector of evaluated GP distribution, with length equal to the recycled lengths of `q/x/p`, `scale`, and `shape`.

Examples

```
pgp(1:10, 1, 1)
dgp(1:10, 2, 0)
qgp(1:9 / 10, 10, -2)
```

`plot.dst` *Plot a Distribution*

Description

Plot a distribution's representation.

Usage

```
## S3 method for class 'dst'
plot(
  x,
  what = c("density", "pmf", "cdf", "survival", "quantile", "hazard", "chf"),
  ...
)
```

Arguments

`x`	Distribution object
`what`	Name of the representation to plot.
`...`	Other arguments to pass to the `graphics::curve` function, or `graphics::plot` in the case of the PMF.

Value

This function is run for its graphics byproduct, and therefore returns the original distribution, invisibly.

Examples

```
d <- dst_norm(0, 1)
plot(d, from = -4, to = 4)
plot(d, "cdf", n = 1000)
plot(d, "survival")
plot(d, "quantile")
plot(d, "hazard")
plot(d, "chf")

p <- dst_pois(4)
plot(p)
```

plp3 *Representations of the Log Pearson Type III Distribution*

Description

Representations of the Log Pearson Type III (LP3) Distribution

Usage

```
plp3(x, meanlog, sdlog, skew, lower.tail = TRUE)

dlp3(x, meanlog, sdlog, skew)

qlp3(p, meanlog, sdlog, skew)

rlp3(n, meanlog, sdlog, skew)
```

Arguments

x	Vector of quantiles.
meanlog	Parameter representing the mean of the random variable in log (base e) space; numeric. Vectors are allowed except for `rlp3()`.
sdlog	Parameter representing the standard deviation of the random variable in log (base e) space; positive numeric. Vectors are allowed except for `rlp3()`.
skew	Parameter representing the skewness of the random variable in log (base e) space; numeric. Vectors are allowed except for `rlp3()`.
lower.tail	Logical; if `TRUE` (default), probabilities are `P(X <= x)`, otherwise, `P(X > x)`.
p	Vector of probabilities.
n	Single positive whole number; number of observations to draw from the distribution.

Value

Vector of evaluated LP3 distribution, with length equal to the recycled lengths of `x`/`p`, `meanlog`, `sdlog`, and `skew`. For `rlp3()`, a vector of length `n`.

Examples

```
plp3(1:10, meanlog = 0, sdlog = 1, skew = 1)
dlp3(1:10, meanlog = 1:10, sdlog = 2, skew = 0)
qlp3(1:9 / 10, meanlog = 2, sdlog = 10, skew = 2)
set.seed(1)
rlp3(10, meanlog = 2, sdlog = 10, skew = 2)
```

ppearson3 *Representations of the Pearson Type III Distribution*

Description

Representations of the Pearson Type III Distribution

Usage

```
ppearson3(x, location, scale, shape, lower.tail = TRUE)

dpearson3(x, location, scale, shape, log = FALSE)

qpearson3(p, location, scale, shape)

rpearson3(n, location, scale, shape)
```

Arguments

x	Vector of quantiles.
location	Parameter representing the boundary (endpoint) of the distribution; numeric. This is the left endpoint when shape is positive and the right endpoint when shape is negative. Vectors are allowed except for rpearson3().
scale	Scale parameter; positive numeric. Vectors are allowed except for rpearson3().
shape	Shape parameter; numeric. A negative value gives the distribution reflected about location (an upper-bounded, left-skewed distribution). Vectors are allowed except for rpearson3().
lower.tail	Logical; if TRUE (default), probabilities are P(X <= x), otherwise, P(X > x).
log	Logical; if TRUE, probabilities are given as log-probabilities.
p	Vector of probabilities.
n	Single positive whole number; number of observations to draw from the distribution.

Value

Vector of evaluated Pearson Type III distribution, with length equal to the recycled lengths of x/p, location, scale, and shape. For rpearson3(), a vector of length n.

Examples

```
ppearson3(1:10, location = 0, scale = 1, shape = 1)
dpearson3(1:10, location = 1:10, scale = 2, shape = 0)
qpearson3(1:9 / 10, location = -2, scale = 10, shape = 2)
set.seed(1)
rpearson3(10, location = 2, scale = 10, shape = 2)
```

pretty_name — *Distribution name*

Description

Print the name of a distribution, possibly with parameters.

Usage

```
pretty_name(distribution, param_digits = 0)
```

Arguments

distribution	Distribution object.
param_digits	How many significant digits to include when displaying the parameters? `0` if you don't want to display parameters. Length 1 vector.

Value

A character containing the distribution's name, possibly followed by parameters in brackets.

Examples

```
d <- dst_norm(0.3552, 1.1453)
pretty_name(d)
pretty_name(d, 2)
```

prob_left — *Find the probability left or right of a number*

Description

Probability to the left or right of a number, inclusive or not. `prob_left()` is a more general cdf defined using either < or <=, and `prob_right()` is a more general survival function defined using either > or >=.

Usage

```
prob_left(distribution, of, inclusive)

prob_right(distribution, of, inclusive)
```

Arguments

distribution	Distribution to find probabilities of.
of	Find the probability to the left or right *of* this number. Could be a vector.
inclusive	Should `of` be included in the probability calculation? Logical.

Value

A vector of probabilities.

Examples

```
d <- dst_pois(5)
prob_left(d, of = 3, inclusive = TRUE)
prob_left(d, of = 3, inclusive = FALSE)
prob_right(d, of = 0:3, inclusive = TRUE)
```

---

range.dst *Range of Distribution*

Description

Range returns a vector of length two, with the minimum and maximum values of the (support of the) distribution.

The `support` method gives the smallest and largest values the support reaches — its two outermost points, taking the atoms and the continuous regions together. Gaps in between are not represented.

An empty support reaches nothing, and its range is `c(Inf, -Inf)` — what R gives for the range of nothing, and reversed on purpose, being the identity for combining ranges.

Usage

```
## S3 method for class 'dst'
range(distribution, ...)

## S3 method for class 'support'
range(support, ...)
```

Arguments

`distribution`	Distribution to compute range from.
`...`	Not used; vestige of the `base::range()` S3 generic.
`support`	A support object.

Details

The range is read from the distribution's support (see `support()`), which is where a distribution says what values it reaches. In this it behaves like `vtype()`: derived, not declared.

It is still a property, and `eval_property()` reaches it like any other, so code walking a list of property names need not know which are stored and which are worked out. What it cannot be is *stated*: `distribution()` refuses a `range` entry, since a stated one would be consulted ahead of the derived value and could disagree with it.

The Null distribution is a different case: it has no support at all, so neither end is *known*, and its range is `c(NA, NA)` — still a vector of length two, rather than a single `NA`. An empty support says there is nothing to reach; the Null distribution says nothing at all.

Value

Vector of length two, containing the minimum and maximum values of a distribution.

Examples

```
a <- dst_gp(1, 0.5)
b <- dst_unif(0, 1)
c <- dst_norm(3, 4)
range(a)
range(b)
range(c)
range(continuous(c(0, 1), c(3, 4)))
range(mixed(discrete = -1, continuous = c(0, Inf)))
range(empty_support())
```

---

realise — *Generate a Sample from a Distribution*

Description

Draw n independent observations from a distribution.

Usage

```
realise(distribution, n = 1)

realize(distribution, n = 1)
```

Arguments

distribution	Distribution object.
n	Number of observations to generate.

Value

Vector of independent values drawn from the inputted distribution.

Note

realise() and realize() are aliases and do the same thing.

Examples

```
d <- dst_pois(5)
set.seed(2)
realise(d, n = 10)
```

scalar *A Distribution has Length 1*

Description

A distribution object is one distribution, so `length()` gives 1 and `is.na()` gives a single logical. `as.list()` wraps the distribution in a list of one.

Usage

```
## S3 method for class 'dst'
length(x)

## S3 method for class 'dst'
is.na(x)

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

Arguments

`x`	A distribution object.
`...`	Not used.

Details

A distribution is built out of a list of its properties — a CDF, a density, a mean — and without these methods base R reports on that list rather than on the distribution. `length()` counted the properties and `is.na()` tested each one, so `dst_norm(0, 1)` answered with eleven `FALSE`s. Neither answer was about the distribution.

`is.na()` is `TRUE` for the Null distribution (`dst_null()`) and `FALSE` for every other. The Null distribution is the missing value of the distribution world, so it is the one that `is.na()` finds.

Note that the properties are still reachable, and are still what the object is made of: `x[["cdf"]]` and `names(x)` are unchanged, and `eval_property()` is the supported way to get at them. Only the questions asked of the distribution *as a whole* now answer about the whole.

To hold several distributions, put them in a list; in a data frame, that is a list-column. A distribution does not have length beyond one.

Value

For `length()`, the number 1. For `is.na()`, a single logical. For `as.list()`, a list containing the one distribution.

Examples

```
d <- dst_norm(0, 1)
length(d)
is.na(d)

# The Null distribution is the missing one.
is.na(dst_null())

# Several distributions go in a list.
ds <- list(dst_norm(0, 1), dst_null(), dst_pois(3))
vapply(ds, is.na, logical(1))
```

---

support *Retrieve the Support of a Distribution*

Description

Returns the structured support of a distribution: its atomic part and its continuous part. Every distribution has one, because `distribution()` requires it — the single exception being `dst_null()`, which has nothing to place anywhere and returns `NULL`.

Usage

```
support(distribution)
```

Arguments

`distribution`	Distribution object.

Value

A support object, or `NULL` if the distribution has no structured support.

See Also

`discrete()`, `continuous()`, `mixed()` to build supports; `vtype()` for the derived variable type.

Other Support: `atoms()`, `empty_support()`, `is_support()`, `support-construction`

Examples

```
support(distribution(.support = continuous(c(0, Inf))))
```

support-construction *Specify the Support of a Distribution*

Description

A support says where a distribution's probability lives, and in what form. Probability comes in two forms: *mass*, which sits on single points, and *density*, which is spread over regions. A support records both — the points carrying mass, its *atoms*, and the *regions* carrying density — and `discrete()`, `continuous()` and `mixed()` build one from those pieces.

Usage

```
discrete(atoms = numeric(0))

continuous(...)

mixed(discrete = numeric(0), continuous = numeric(0))
```

Arguments

`atoms` For `discrete()`, the points carrying mass: a `discretes` object (see the **discretes** package, e.g. `discretes::natural0()`), a numeric vector of finitely many atoms, or a purely discrete support. A bare numeric vector is unambiguous here because the argument names the intent (contrast with passing one to `.support`, which is rejected).

`...` For `continuous()`, one or more regions, each given as a length-2 numeric `c(lower, upper)`. With no arguments, `continuous()` defaults to the whole real line, `c(-Inf, Inf)`. Overlapping or touching regions are merged and sorted into a canonical form.

`discrete, continuous` For `mixed()`, the two halves. Each takes the same things its own constructor takes, or a support already built by it: `discrete` as for `atoms` above, `continuous` as for `...` below.

Details

The variable type (`vtype()`) is *derived* from the support: a support with only atoms is `"discrete"`, only a continuous part is `"continuous"`, both is `"mixed"`, and neither is `"empty"`.

Because the type is derived, none of the three insists on being handed something non-empty. Each builds whatever the parts describe, and describing nothing gives `empty_support()`. So `mixed(continuous = continuous())` is the whole real line, `discrete(numeric(0))` is empty, and `mixed()` is empty too. This is what makes them usable when the parts are computed rather than typed and may come out empty; `mixed()` is then the general constructor, with `discrete()` and `continuous()` the direct way to say one kind on its own.

A region is written as a closed interval, but its endpoints carry no probability either way, a single point having no width, so open against closed makes no difference there. An atom that happens to sit on a region's boundary is simply tracked as an atom.

Recording where the mass is and where the density is are two pieces of information, not one. Knowing which values are possible is not enough: `continuous(c(0, 1))` and `mixed(discrete = 0, continuous = c(0, 1))` cover the same values, but they are different supports and the distributions over them differ: one has `P(X = 0) = 0`, the other does not. This is why an atom lying inside a region is kept rather than absorbed into it.

The third kind:

Strictly, a measure on the real line splits into three parts, not two: mass on points, density over regions, and a third kind with neither — all of its probability on a set of zero total length, none of it sitting on any point. The Cantor distribution is the usual example. This is the Lebesgue decomposition, and the third part is called singular continuous. A support here has no way to describe one, so such distributions are out of reach.

Value

A support object (class `"support"`).

See Also

`support()` to retrieve a distribution's support, `vtype()` for the derived variable type.

Other Support: `atoms()`, `empty_support()`, `is_support()`, `support()`

Examples

```
discrete(discretes::natural0())   # e.g. the support of a Poisson
discrete(c(3.5, 1.2, 6.7))        # finitely many atoms
continuous(c(0, Inf))             # e.g. the support of a Gamma
continuous(c(0, 1), c(3, 4))      # a union of regions
mixed(discrete = 0, continuous = c(0, Inf))  # an atom, plus a tail
```

`support_add_atoms` *Add or Remove Atoms*

Description

Add atoms to a support, or take them away.

Usage

```
support_add_atoms(support, atoms)

support_drop_atoms(support, atoms)
```

Arguments

`support`	A support object, or a distribution.
`atoms`	Atoms to add or remove: a numeric vector, or a `discretes` object. Removing requires finitely many atoms, since they have to be enumerated; adding does not.

Details

Adding an atom that is already there changes nothing. Removing one that is not there changes nothing either. Removing an atom does not disturb the continuous part, so removing an atom sitting on an interval leaves the interval whole.

Value

A support object.

See Also

Other Support algebra: `support_contains()`, `support_restrict()`, `support_transform()`, `support_union()`

Examples

```
support_add_atoms(continuous(c(0, Inf)), 0)
support_drop_atoms(discrete(c(1, 2, 3)), 2)

# Removing an atom leaves the continuous part alone.
support_drop_atoms(mixed(discrete = 0, continuous = c(0, 1)), 0)
```

---

support_contains *Test Membership of a Support*

Description

Is a value in the support at all, or an atom of it specifically?

Usage

```
support_contains(support, at)

support_has_atom(support, at)
```

Arguments

`support`	A support object, or a distribution.
`at`	Values to test. Vectorised.

Details

`support_contains()` is `TRUE` for a value that is either an atom or inside one of the continuous intervals. `support_has_atom()` is `TRUE` only for the atoms, and so is the one to reach for when what matters is whether a point carries positive probability.

Continuous intervals count their endpoints as contained. Those endpoints carry no probability, so a value can be contained in a support without being a point of positive mass — which is exactly the distinction between these two functions.

Value

A logical vector the same length as `at`.

See Also

Other Support algebra: `support_add_atoms()`, `support_restrict()`, `support_transform()`, `support_union()`

Examples

```
s <- mixed(discrete = 0, continuous = c(2, 5))
support_contains(s, at = c(0, 1, 3, 9))
support_has_atom(s, at = c(0, 1, 3, 9))

support_has_atom(dst_pois(3), at = c(-1, 0, 2.5, 4))
```

---

support_restrict *Restrict a Support to an Interval*

Description

Cut a support down to the part of it lying within `[from, to]`.

Usage

```
support_restrict(
  support,
  ...,
  from = -Inf,
  to = Inf,
  include_from = TRUE,
  include_to = TRUE
)
```

Arguments

`support`	A support object, or a distribution.
`...`	Not used; must be empty. Present so that the arguments below are matched by name.
`from, to`	Endpoints of the interval to restrict to.
`include_from, include_to`	Whether the endpoints themselves are kept.

Details

The `include_*` flags apply to the atoms only. An endpoint of a continuous interval carries no probability either way, so including or excluding it makes no difference to the continuous part.

Restricting to a region the support does not reach gives `empty_support()`, which is the reason that object exists.

Value

A support object.

See Also

`support_union()` to combine supports instead.

Other Support algebra: `support_add_atoms()`, `support_contains()`, `support_transform()`, `support_union()`

Examples

```
support_restrict(continuous(c(0, 10)), from = 3, to = 6)
support_restrict(discrete(natural0()), to = 4)

# Excluding an endpoint drops the atom sitting on it.
support_restrict(discrete(natural0()), to = 4, include_to = FALSE)

# Restricting out of reach gives the empty support.
support_restrict(continuous(c(0, 1)), from = 5, to = 6)
```

---

support_transform *Transform a Support*

Description

Push a support through a strictly monotonic map, giving the support of the transformed variable.

Usage

```
support_transform(
  support,
  fun,
  inv,
  ...,
  increasing = TRUE,
  domain = c(-Inf, Inf),
  range = c(-Inf, Inf)
)

support_shift(support, by)
```

```
support_scale(support, by)

support_reciprocal(support)
```

Arguments

support	A support object, or a distribution.
fun, inv	The map and its inverse. Both must be vectorised, and fun must be strictly monotonic on the support.
...	Not used; must be empty. Present so that the arguments below are matched by name.
increasing	Whether fun is increasing. FALSE for a decreasing map, which reverses each interval's endpoints.
domain, range	The domain and range of fun, needed to transform an atomic part that is described rather than enumerated.
by	For support_shift() and support_scale(), the amount to shift or scale by.

Details

support_shift(), support_scale(), and support_reciprocal() are the common cases, and avoid having to supply an inverse, a domain, and a range by hand.

support_reciprocal() maps each side of zero separately, since 1 / x is monotonic on each side but not across the two. A support with an atom at zero has no reciprocal, and is an error. Zero lying inside a region is fine: a single point carries no probability there.

Scaling by zero sends every value to 0. Density that was spread over a region is compressed onto that single point, and density compressed onto a point is mass. So whatever the support was, the result has a mass at 0 and density nowhere: discrete(0). Only an empty support, having nothing to compress, stays empty.

It only works in that direction. A mass sits on one point and lands on one point, so mass stays mass.

A strictly monotonic map stretches and shifts regions but never squashes one down to a point, so density stays density and mass stays mass. That is why support_transform() asks for a monotonic map, and why scaling by zero — which is not one — is handled separately.

Value

A support object.

See Also

Other Support algebra: support_add_atoms(), support_contains(), support_restrict(), support_union()

Examples

```
support_shift(continuous(c(0, 1)), by = 5)
support_scale(discrete(natural0()), by = 2)

# A decreasing map reverses the region.
support_scale(continuous(c(1, 2)), by = -1)

# Scaling by zero collapses everything onto a single atom.
support_scale(continuous(c(1, 2)), by = 0)

# Reciprocal of a support spanning zero.
support_reciprocal(continuous(c(-2, 4)))

# The general form.
support_transform(
  continuous(c(0, Inf)),
  fun = exp, inv = log,
  domain = c(0, Inf), range = c(1, Inf)
)
```

`support_union` *Combine Supports*

Description

The union of two or more supports: everything covered by any of them.

Usage

```
support_union(...)
```

Arguments

`...`	Supports to combine, or a single list of them. Distributions are accepted in place of supports. With no arguments, the result is `empty_support()`, which is the identity for this operation. If any of them has no support — a `NULL`, or `dst_null()` — the result is `NULL`. A union cannot be known when one of the things being combined is not.

Details

The atomic parts are unioned as series, and the continuous parts are pooled and merged back into canonical form, so touching or overlapping intervals come out as one.

An atom that falls inside another support's continuous part stays an atom. The two carry different kinds of probability, and one does not absorb the other.

Value

A support object.

See Also

support_restrict() to cut a support down instead.

Other Support algebra: support_add_atoms(), support_contains(), support_restrict(), support_transform()

Examples

```
support_union(continuous(c(0, 1)), continuous(c(0.5, 3)))
support_union(discrete(c(1, 2)), continuous(c(5, 6)))

# The identity.
support_union()
```

---

vtype — *Variable Type of a Distribution*

Description

Retrieve the variable type of a distribution, such as "continuous" or "discrete".

Usage

```
vtype(distribution)
```

Arguments

distribution	Distribution object.

Value

Single character with the variable type.

Examples

```
vtype(dst_beta(1, 2))
vtype(dst_bern(0.4))
vtype(distribution(
  cdf = pnorm,
  density = dnorm,
  .support = continuous()
))
```

Index

∗ **Distribution Construction**
distribution, 6
∗ **Support algebra**
support_add_atoms, 51
support_contains, 52
support_restrict, 53
support_transform, 54
support_union, 56
∗ **Support**
atoms, 5
empty_support, 25
is_support, 37
support, 49
support-construction, 50
∗ **distributional representations**
eval_cdf, 26
eval_chf, 27
eval_density, 28
eval_hazard, 29
eval_odds, 30
eval_pmf, 31
eval_quantile, 33
eval_return, 35
eval_survival, 36

as.list.dst *(scalar)*, 48
atoms, 5, *26*, *37*, *49*, *51*

continuous *(support-construction)*, 50
continuous(), *7*, *26*, *49*

dgev *(pgev)*, 40
dgp *(pgp)*, 41
discrete *(support-construction)*, 50
discrete(), *7*, *26*, *49*
discretes::natural0(), *50*
distionary *(distionary-package)*, 3
distionary-package, 3
distribution, *4*, 6
distribution(), *22*, *26*, *46*, *49*
dlp3 *(plp3)*, 43
dpearson3 *(ppearson3)*, 44
dst_bern, 8
dst_beta, 9
dst_binom, *3*, 10
dst_cauchy, 10
dst_chisq, 11
dst_degenerate, 11
dst_empirical, *3*, 12
dst_empirical(), *15*
dst_exp, *3*, 13
dst_f, 14
dst_finite, 15
dst_finite(), *12*, *13*
dst_gamma, 15
dst_geom, *3*, 16
dst_gev, 16
dst_gp, 17
dst_gumbel, 18
dst_hyper, 19
dst_lnorm, 19
dst_lp3, 20
dst_nbinom, 20
dst_norm, *3*, 21
dst_null, 22
dst_null(), *6*, *48*, *49*, *56*
dst_pearson3, 23
dst_pois, *3*, 23
dst_t, 24
dst_unif, *3*, 24
dst_weibull, 25

empty_support, *6*, 25, *37*, *49*, *51*
empty_support(), *50*, *54*, *56*
enframe_cdf *(eval_cdf)*, 26
enframe_chf *(eval_chf)*, 27
enframe_density *(eval_density)*, 28
enframe_hazard *(eval_hazard)*, 29
enframe_odds *(eval_odds)*, 30
enframe_pmf *(eval_pmf)*, 31

enframe_quantile (eval_quantile), 33
enframe_return (eval_return), 35
enframe_survival (eval_survival), 36
eval_cdf, *4*, 26, *28–32*, *34–36*
eval_chf, 27, *27*, *29–32*, *34–36*
eval_density, *27*, 28, *28*, *30–32*, *34–36*
eval_hazard, *27, 28*, 29, *29*, *31, 32*, *34–36*
eval_odds, *27–29*, 30, *30*, *32*, *34–36*
eval_pmf, *27–30*, 31, *31*, *34–36*
eval_property, 32
eval_property(), *8*, *46*, *48*
eval_quantile, *4*, *27–32*, 33, *35, 36*
eval_return, *27–32*, *34*, 35, *36*
eval_survival, *27–32*, *34, 35*, 36

is.distribution (distribution), 6
is.na(), *22*
is.na.dst (scalar), 48
is_distribution (distribution), 6
is_empty_support (is_support), 37
is_empty_support(), *26*
is_support, *6*, *26*, 37, *49*, *51*

kurtosis, 37
kurtosis_exc (kurtosis), 37

length.dst (scalar), 48
length.dst(), *22*

mean, *4*
mean(), *22*
mean.dst (kurtosis), 37
median.dst, 39
mixed (support-construction), 50
mixed(), *7*, *26*, *49*

parameters, 39
parameters<- (parameters), 39
pgev, 40
pgp, 41
plot.dst, 42
plp3, 43
ppearson3, 44
pretty_name, 45
prob_left, 45
prob_right (prob_left), 45

qgev (pgev), 40
qgp (pgp), 41
qlp3 (plp3), 43
qpearson3 (ppearson3), 44

range, *4*
range(), *6, 7*, *22*
range.dst, 46
range.support (range.dst), 46
realise, *4*, 47
realize (realise), 47
regions (atoms), 5
rlp3 (plp3), 43
rpearson3 (ppearson3), 44

scalar, 48
skewness (kurtosis), 37
stdev (kurtosis), 37
support, *6*, *26*, *37*, 49, *51*
support(), *6*, *22*, *34*, *46*, *51*
support-construction, 50
support_add_atoms, 51, *53–55*, *57*
support_contains, 52, *52*, *54, 55*, *57*
support_drop_atoms (support_add_atoms), 51
support_has_atom (support_contains), 52
support_reciprocal (support_transform), 54
support_restrict, *52*, 53, *53*, *55*, *57*
support_restrict(), *57*
support_scale (support_transform), 54
support_shift (support_transform), 54
support_transform, *52, 53*, 54, *54*, *57*
support_union, *52–55*, 56
support_union(), *54*

variance (kurtosis), 37
vtype, 57
vtype(), *7*, *22*, *46*, *49–51*