Package 'distionary'

April 27, 2026

Title Create and Evaluate Probability Distributions

Version 0.1.1

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

RoxygenNote 7.3.3

Imports checkmate, rlang, stats, vctrs

VignetteBuilder knitr

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

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

Depends R (>= 2.10)

NeedsCompilation no

Author Vincenzo Coia [aut, cre, cph], Amogh Joshi [ctb], Shuyi Tan [ctb], Zhipeng Zhu [ctb], olivroy [ctb] (GitHub contributor)

Maintainer Vincenzo Coia <vincenzo.coia@gmail.com>

Repository CRAN

Date/Publication 2026-04-27 09:00:02 UTC

Contents

`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> [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),
  .vtype = "continuous",
  .name = "Linear"
)
plot(my_dist)
plot(my_dist, "cdf")

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

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

is_distribution(object)

is.distribution(object)
```

Arguments

`...`	Name-value pairs for defining the distribution.
`.vtype`	The variable type, typically "discrete" or "continuous". Can be any character vector of length 1, but is converted to lowercase with `tolower()` for compliance with known types.
`.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

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`: A vector of the minimum and maximum value of a distribution's support.

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
  },
  .vtype = "continuous",
  .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_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()
```

Value

A Null distribution.

Examples

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

`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 how to shift the Gamma distribution; single numeric.
`scale`	Scale parameter of the Gamma distribution; single positive numeric.
`shape`	Shape parameter of the Gamma distribution; single positive numeric.

Value

A Pearson Type III distribution.

Examples

```
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)
```

`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,
  .vtype = "continuous"
)
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

When a quantile function does not exist, an algorithm is deployed that calculates the left inverse of the CDF. This algorithm works by progressively cutting the specified range in half, moving into the left or right half depending on where the solution is. The algorithm is not currently fast and is subject to improvement, and is a simple idea that has been passed around on the internet here and there. Tolerance is less than 1e-9, unless the maximum number of iterations (200) is reached.

The algorithm is not new, and is rather simple. The algorithm works by progressively cutting an initially wide range in half, moving into the left or right half depending on where the solution is. I found the idea on Stack Overflow somewhere, but unfortunately cannot find the location anymore.

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)
```

---

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 there is no method associated with a subclass of x, then moments are calculated using stats::integrate() from the density function.

Value

A single numeric.

Note

Beware that if a quantity is being calculated numerically for a non-continuous (e.g., discrete) distribution, the calculation could be highly approximate. An upcoming version of distionary will resolve this issue.

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)
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)
```

---

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.

Usage

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

Arguments

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

Details

If there are no methods for the distribution's class, the range is calculated using eval_quantile() at 0 and at 1.

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)
```

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)
```

---

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())
```

Index