Package 'distplyr'

September 15, 2026

Type Package

Title Manipulate and Combine Probability Distributions

Version 0.3.0

Description Go beyond standard probability distributions such as the Normal or Exponential by combining, shifting, maximizing, and otherwise transforming distributions with simple, verb-based functions. Provides easy access to a broader space of distributions more representative of real-world systems such as river flows or insurance claims.
Part of the probaverse framework of packages to support advanced statistical modeling and simulations with an intuitive workflow.

License MIT + file LICENSE

Depends R (>= 3.5.0)

Imports rlang, vctrs, stats, checkmate, distionary (>= 0.2.0)

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

VignetteBuilder knitr

Config/testthat/edition 3

Encoding UTF-8

RoxygenNote 7.3.3

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

NeedsCompilation no

Author Vincenzo Coia [aut, cre, cph],
Amogh Joshi [ctb],
Shuyi Tan [ctb],
Zhipeng Zhu [ctb]

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

Repository CRAN

Date/Publication 2026-09-15 06:40:02 UTC

Contents

flip 2
graft_right 4
Math.dst 6
maximize 7
mix 9
Ops.dst 10
trim_left 12

Index **14**

`flip` *Linear and Reciprocal Transformations*

Description

Transform distributions using location shifts, scaling, negation, and reciprocals. If `X` is a random variable following a distribution, these functions return the distribution of the transformed variable.

Usage

```
flip(distribution)

invert(distribution)

multiply(distribution, constant)

shift(distribution, constant)
```

Arguments

`distribution`	A probability distribution.
`constant`	A numeric value for shifting or scaling.

Value

A transformed distribution.

Functions vs Operators

These transformations can be applied using named functions or arithmetic operators:

- `shift(d, a)` or `d + a` - Returns distribution of `X + a`
- `multiply(d, a)` or `d * a` - Returns distribution of `X * a`
- `flip(d)` or `-d` - Returns distribution of `-X`
- `invert(d)` or `1 / d` - Returns distribution of `1 / X`

For complete documentation of operator usage, see `Ops.dst()`.

Special Cases

Negation in multiplication: When `multiply()` receives a negative constant, it internally calls `flip()` on the result of multiplying by the absolute value.

Inversion constraint: `invert()` requires that the distribution has no mass at zero (i.e., `P(X = 0) = 0`). An error is returned if this condition is violated.

Simplifications

These functions apply automatic simplifications when possible. For example:

- Shifting a Normal distribution returns another Normal distribution
- Multiplying a Uniform distribution returns another Uniform distribution
- Flipping a symmetric distribution may preserve its form

More simplifications will be added in future versions of distplyr.

See Also

`Ops.dst()` for arithmetic operators including `+`, `-`, `*`, `/`.

Examples

```
d_pois <- distionary::dst_pois(1.1)
d_norm <- distionary::dst_norm(4, 1)
d_unif <- distionary::dst_unif(0, 1)

# Shifting
shift(d_pois, 1)
d_pois + 1           # Equivalent using operator

# Scaling
multiply(d_unif, 2)
d_unif * 2           # Equivalent using operator

# Negation
flip(d_norm)
-d_norm              # Equivalent using operator

# Inversion
d_positive <- distionary::dst_unif(1, 2)
invert(d_positive)
1 / d_positive       # Equivalent using operator

# Combine multiple operations
4 - 2 * d_pois
multiply(flip(multiply(d_pois, 2)), -1) + 4  # Equivalent
```

graft_right *Graft a tail onto a distribution*

Description

Replace one end of a distribution with a different model of that end. `graft_right()` keeps `body` below `of` — the knot — and hands everything above it to the tail; `graft_left()` does the same at the lower end. The tail's share of the total is the body's own probability of reaching past the knot.

Usage

```
graft_right(
  body,
  of,
  ...,
  tail_excess,
  tail_absolute,
  knot = c("body", "tail", "split")
)

graft_left(
  body,
  of,
  ...,
  tail_excess,
  tail_absolute,
  knot = c("body", "tail", "split")
)
```

Arguments

`body`	Distribution supplying the part of the range that is kept.
`of`	Value on the real line where the tail is attached: the knot.
`...`	Currently unused; must be empty.
`tail_excess`	Distribution of the tail measured from the knot, moved from zero to `of`. Name either this or `tail_absolute`, not both.
`tail_absolute`	Distribution of the tail on the body's own scale, left where it is and conditioned beyond the knot. Name either this or `tail_excess`, not both.
`knot`	Which side probability sitting exactly on the knot belongs to: the `"body"` (the default), the `"tail"`, or `"split"` between them.

Value

A graft: the body on one side of the knot and the tail on the other, which is a special type of mixture distribution.

Two ways to hand over the tail

Name exactly one of `tail_excess` and `tail_absolute`. Neither has a default, because nothing in a distribution says which scale it is on.

- `tail_excess` is measured from the knot: the distribution of `X - of`, whose zero is the knot. It is **moved from zero to the knot**, by adding `of` to it, so a generalised Pareto living on `[0, Inf)` becomes a tail living on `[of, Inf)`. All of its probability must lie on one side of zero: at or above for `graft_right()`, at or below for `graft_left()`.
- `tail_absolute` is on the body's scale already and stays where it is, conditioned on falling beyond the knot. Anything placed by hand goes here — `multiply(ratio, of)`, for a model of `X / of`, say.

A tail that is already in place but happens to sit above zero cannot be told apart from a model of excesses, so `tail_excess` accepts it. If it starts exactly at `of`, the likeliest case, you get a warning.

Where the knot goes

`knot` names the side that probability sitting exactly on the knot belongs to: `"body"` (the default), `"tail"`, or `"split"` for half each, the mid-p convention. Naming one side names the other, so the knot is counted once; what a side does not take passes into the other's share. None of this has any effect unless there is mass exactly at `of`, as there never is in a continuous distribution.

The default leaves the body alone up to and including the knot, and gives the tail `prob_right(body, of, inclusive = FALSE)`, the probability of exceeding it. That is the convention peaks-over-threshold is written in, where the excess `X - of` is conditioned on `X > of` strictly and an excess of exactly zero does not arise.

See Also

`trim_left()` and `trim_right()`, which discard an end rather than replacing it.

Examples

```
body <- distionary::dst_norm(0, 1)
u <- distionary::eval_quantile(body, at = 0.9)

# Excesses over `u`, living on [0, Inf): moved to start at `u`.
graft_right(body, of = u, tail_excess = distionary::dst_gp(1, 0.3))

# The same graft, placed by hand instead.
moved <- shift(distionary::dst_gp(1, 0.3), u)
graft_right(body, of = u, tail_absolute = moved)

# A model on the body's scale, conditioned above `u`.
graft_right(body, of = u, tail_absolute = distionary::dst_norm(1, 3))
```

Math.dst *Mathematical Transformations for Distributions*

Description

Apply mathematical functions like `log()` and `exp()` to probability distributions. If `X` is a random variable following a distribution, these functions return the distribution of the transformed variable.

Usage

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

Arguments

`x`	A probability distribution object.
`...`	Additional arguments passed to specific methods. For `log()`, you can specify `base` (defaults to `exp(1)` for natural log).

Details

These S3 methods extend base R functions to work with distributions.

Value

A transformed distribution object.

Supported Functions

`log(x, base = exp(1))` Returns the distribution of `log(X)`. The base can be specified (defaults to natural log). An error is returned if the distribution has non-positive values as possible outcomes.

`log10(x)` Returns the distribution of `log10(X)`. Equivalent to `log(x, base = 10)`.

`exp(x)` Returns the distribution of `exp(X)`.

`sqrt(x)` Returns the distribution of `sqrt(X)`. Equivalent to `x^0.5`. Requires all values to be non-negative.

Power Operator

The power operator `^` also works with distributions (see `Ops.dst()`). When raising a distribution to a numeric power (e.g., `dst^2`), it uses the relationship `X^a = exp(a * log(X))`, combining both exponential and logarithmic transformations.

See Also

- `Ops.dst()` for the `^` operator and other arithmetic operations
- `shift()`, `multiply()`, `flip()`, `invert()` for linear transformations

Examples

```
# Logarithmic transformations
d <- distionary::dst_unif(1, 10)
log(d)              # Natural log
log(d, base = 10)   # Log base 10
log10(d)            # Also log base 10
sqrt(d)             # Square root of uniform

# Exponential transformation
d2 <- distionary::dst_norm(0, 1)
d3 <- distionary::dst_beta(5, 4)
exp(d2)             # Log-normal distribution
exp(d3)             # No simplification

# These can be combined
log(exp(d2))        # Returns back to normal distribution
log(exp(d3))        # Still returns d3.
5^(log(d3, base = 5)) # Still returns d3.
```

`maximize` *Extremum of Several Distributions*

Description

For a collection of distributions, obtain the distributions of the maximum (`maximize()`) and minimum (`minimize()`) from independent draws of each component distribution.

Aliases `maximise()` and `minimise()` are also provided.

Usage

```
maximize(
  ...,
  draws = 1,
  na_action_dst = c("null", "drop", "fail"),
  na_action_draws = c("null", "drop", "fail")
)

minimize(
  ...,
  draws = 1,
  na_action_dst = c("null", "drop", "fail"),
  na_action_draws = c("null", "drop", "fail")
)
```

Arguments

`...`	Distribution objects, or list of distributions.

draws
: Number of draws from each distribution considered in the maximum (possibly not integer, but never negative). Either a single numeric applying to all distributions in `...`, or a vector matching the number of distributions in `...`.

na_action_dst, na_action_draws
: What should be done with Null distributions in `...` and `NA` in `draws`? Character vector of length 1: one of "fail", "null" (default), or "drop". See details.

Details

To give an example of what distribution is returned, if X1 and X2 are two random variables with distributions D1 and D2 respectively, then `maximize(D1, D2, draws = c(2, 3))` returns the distribution of `max(X1, X1, X2, X2, X2)`.

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

`na_action_dst` and `na_action_draws` specify the NA action for distributions and `draws`. "NA" here means either `NA` in the `draws` vector, or a Null distribution (`distionary::dst_null()`) in the distributions. Options are, in order of precedence:

- `"fail"`: Throw an error in the presence of NAs.
- `"null"`: Return a Null distribution in the presence of NAs.
- `"drop"`: Remove distribution-weight pairs having an NA value

Simplifications made in these functions include the following:

- If any distributions are entirely to the left (right) of others, then they are removed from consideration in `maximize()` (`minimize()`).
- If all Finite distributions are input, the result is also a Finite distribution.
- If the same distribution is input multiple times, their corresponding draws are summed.

Value

A distribution corresponding to the maximum or minimum.

Examples

```
library(distionary)
# One is always more extreme than the other in this case.
d1 <- dst_unif(-1, 2)
d2 <- dst_unif(5, 6)
maximize(d1, d2) # d2
minimize(d1, d2) # d1

# Visualizing the maximum and minimum
d3 <- dst_norm(4, 1)
d4 <- dst_exp(0.3)

dmax <- maximize(d3, d4, draws = 1:2)
dmin <- minimize(d3, d4, draws = 1:2)
```

```
# Maximum
plot(d3, col = "blue", lty = 2, from = 0, to = 14)
plot(d4, col = "red", lty = 2, add = TRUE)
plot(dmax, add = TRUE, n = 1000)
legend(
 "topright",
 legend = c("Maximum", "N(4,1)", "Exp(0.3)"),
 col = c("black", "blue", "red"),
 lty = c(1, 2, 2)
)

# Minimum
plot(d3, col = "blue", lty = 2, from = 0, to = 10)
plot(d4, col = "red", lty = 2, add = TRUE)
plot(dmin, add = TRUE, n = 1000)
legend(
  "topright",
  legend = c("Minimum", "N(4,1)", "Exp(0.3)"),
  col = c("black", "blue", "red"),
  lty = c(1, 2, 2)
)
```

`mix` *Mixture Distributions*

Description

Create a mixture distribution, which can be thought of as an average of multiple distributions (in terms of their CDF, density, PMF, or survival functions, for example). Data drawn from a mixture distribution involves two steps: first randomly selecting the distribution to draw from, followed by the random selection from that distribution.

Usage

```
mix(
  ...,
  weights = 1,
  na_action_dst = c("null", "drop", "fail"),
  na_action_w = c("null", "drop", "fail")
)
```

Arguments

`...`	Distribution objects, or list of distributions.
`weights`	Vector of weights corresponding to the distributions; or, single numeric for equal weights. When normalized, they correspond to the probabilities of selecting each distribution.

`na_action_dst`, `na_action_w`
: What should be done with null distributions in `...` and `NA` in `weights`? Character vector of length 1: one of "fail", "null" (default), or "drop". See details.

Details

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

`na_action_dst` and `na_action_w` specify the NA action for distributions and weights. "NA" here means either `NA` in the `weights` vector, or a Null distribution (`distionary::dst_null()`) in the distributions. Options are, in order of precedence:

- `"fail"`: Throw an error in the presence of NAs.
- `"null"`: Return a Null distribution in the presence of NAs.
- `"drop"`: Remove distribution-weight pairs having an NA value

Value

A mixture distribution.

Examples

```
library(distionary)
a <- dst_norm(0, 1)
b <- dst_norm(5, 2)
m1 <- mix(a, b, weights = c(1, 4))
plot(a, col = "red", lty = 2, from = -3, to = 11)
plot(b, add = TRUE, col = "blue", lty = 2)
plot(m1, add = TRUE)
legend(
  "topright",
   legend = c("Mixture", "N(0,1)", "N(5,2)"),
   col = c("black", "red", "blue"),
   lty = c(1, 2, 2)
)
```

Ops.dst *Arithmetic Operations for Distributions*

Description

Apply arithmetic operators to probability distributions. These operations transform distributions in intuitive ways, treating them similarly to numeric values.

Usage

```
## S3 method for class 'dst'
Ops(e1, e2)
```

Arguments

e1	A probability distribution or numeric value.
e2	A probability distribution or numeric value.

Details

These S3 methods extend arithmetic operators to work with distributions.

Value

A transformed distribution object.

Supported Operators

`d + a` **or** `a + d` Shifts the distribution by adding constant `a`. Equivalent to `shift()`.

`d - a` Shifts the distribution by subtracting constant `a`. Equivalent to `shift(d, -a)`.

`-d` Flips the distribution (negation). Equivalent to `flip()`.

`d * a` **or** `a * d` Scales the distribution by multiplying by constant `a`. Equivalent to `multiply()`.

`d / a` Scales the distribution by dividing by constant `a`. Equivalent to `multiply(d, 1/a)`.

`a / d` Returns the distribution of `a / X` (reciprocal scaling). For `a = 1`, equivalent to `invert()`.

`d ^ a` Raises the distribution to power `a`. For positive distributions only, computed as `exp(a * log(d))`.

`a ^ d` Returns the distribution of `a^X`. Requires positive base `a`.

Power Operator Details

The power operator `^` deserves special attention:

- When the **base is a distribution** (e.g., `dst_beta(1, 1)^2`), it computes the distribution of `X^a` using the transformation `exp(a * log(X))`. This requires all values in the distribution to be positive.
- When the **exponent is a distribution** (e.g., `2^dst_norm(0, 1)`), it computes the distribution of `a^X` using `exp(X * log(a))`. The base `a` must be positive.

These implementations internally use both `Math.dst()` methods `log()` and `exp()`.

See Also

- `shift()`, `multiply()`, `flip()`, `invert()` for the underlying transformation functions
- `Math.dst()` for `log()`, `exp()`, and `sqrt()` functions

Examples

```
d <- distionary::dst_beta(3, 2)

# Shifting and scaling
d + 10          # Shift right by 10
d * 2           # Scale by 2
3 * d - 5       # Scale then shift

# Power operations
exp(d)          # e^X: exponential of Beta
2^d             # 2^X: base 2 raised to Beta

# With positive distributions
d_pos <- distionary::dst_unif(1, 2)
d_pos^2         # X^2: uniform squared
d_pos^0.5       # sqrt(X): square root
sqrt(d_pos)     # Equivalent to d_pos^0.5
```

---

trim_left *Trim (condition) a distribution*

Description

Discard the probability lying to one side of a value, and scale up what remains so that it sums to 1 again. `trim_left()` discards the probability below `of`, giving the distribution of the variable conditioned on landing at or above it; `trim_right()` discards the probability above `of`, conditioning on landing at or below.

Usage

```
trim_left(distribution, of, ..., knot = c("keep", "discard", "split"))

trim_right(distribution, of, ..., knot = c("keep", "discard", "split"))
```

Arguments

`distribution`	Distribution to trim.
`of`	Value on the real line defining where to trim (single numeric).
`...`	Currently unused; must be empty.
`knot`	What to do with the probability sitting exactly on `of`: `"keep"` it (the default), `"discard"` it with the trimmed side, or `"split"` it evenly between the two sides. Only has an effect where `of` carries probability. See Details.

Value

The conditional distribution, renormalised to total probability 1; or the Null distribution, if the trim leaves nothing behind.

What `knot` does

`of` is the knot: the point the trim cuts at. `knot` says what becomes of the probability sitting exactly on it, and matters only when the knot carries mass of its own, as an atom does. Where there is no mass exactly at `of` — anywhere in a continuous distribution — all three actions give the same answer.

- `"keep"` (default) retains it, so `trim_left(d, of)` keeps outcomes greater than *or equal to* `of`. The knot is not on the side being trimmed away: `of` is not to the left of itself.
- `"discard"` throws the knot away with that side, so `trim_left(d, of)` keeps outcomes strictly greater than `of`. Under this setting the probability kept is exactly `distionary::prob_right(d, of, inclusive = FALSE)`, and a left and a right trim at the same knot share nothing.
- `"split"` retains half of it. This is the mid-p convention used in discrete inference, where a boundary atom is shared evenly between the two sides rather than assigned wholly to one.

Whatever is retained is renormalised along with the rest, so the result is always a distribution in its own right.

Values of `of` with nothing beside them

If `of` falls in a gap in the support, the trim takes effect where the support resumes. Trimming a distribution living on `[1, 2]` and `[4, 5]` to the left of 3 gives one living on `[4, 5]`.

If the trim leaves no probability at all — trimming a Uniform(0, 1) to the left of 2, say — the result is the Null distribution, `distionary::dst_null()`.

See Also

`graft_left()` and `graft_right()`, which replace a tail rather than discarding it.

Examples

```
d <- distionary::dst_norm(0, 1)
d <- trim_left(d, -2)
d <- trim_right(d, 2)
distionary::enframe_cdf(d, at = -3:3)

# A Poisson has an atom at 5, so the knot is visible there. By default
# the trim keeps it.
d <- distionary::dst_pois(3)
distionary::eval_pmf(trim_left(d, 5), at = 5)
distionary::eval_pmf(trim_left(d, 5, knot = "discard"), at = 5)
distionary::eval_pmf(trim_left(d, 5, knot = "split"), at = 5)
```

Index

flip, 2
flip(), *6*, *11*

graft_left *(*graft_right*)*, 4
graft_left(), *13*
graft_right, 4
graft_right(), *13*

invert *(*flip*)*, 2
invert(), *6*, *11*

Math.dst, 6
Math.dst(), *11*
maximise *(*maximize*)*, 7
maximize, 7
minimise *(*maximize*)*, 7
minimize *(*maximize*)*, 7
mix, 9
multiply *(*flip*)*, 2
multiply(), *6*, *11*

Ops.dst, 10
Ops.dst(), *2*, *3*, *6*

shift *(*flip*)*, 2
shift(), *6*, *11*

trim_left, 12
trim_left(), *5*
trim_right *(*trim_left*)*, 12
trim_right(), *5*