

# Package ‘RTMBdist’

January 13, 2026

**Type** Package

**Title** Distributions Compatible with Automatic Differentiation by  
‘RTMB’

**Version** 1.0.0

## Description

Extends the functionality of the ‘RTMB’ <<https://kaskr.r-universe.dev/RTMB>> package by providing a collection of non-standard probability distributions compatible with automatic differentiation (AD). While ‘RTMB’ enables flexible and efficient modelling, including random effects, its built-in support is limited to standard distributions. The package adds additional AD-compatible distributions, broadening the range of models that can be implemented and estimated using ‘RTMB’. Automatic differentiation and Laplace approximation are described in Kristensen et al. (2016) <[doi:10.18637/jss.v070.i05](https://doi.org/10.18637/jss.v070.i05)>.

**License** MIT + file LICENSE

**Encoding** UTF-8

**Imports** stats, gamlss.dist, circular, sn, statmod, movMF

**Depends** R (>= 3.5.0), RTMB (>= 1.7.0)

**RoxygenNote** 7.3.3

**Suggests** knitr, rmarkdown, testthat (>= 3.0.0), LaMa (>= 2.0.6),  
Matrix, TMB, gamlss.data, mvtnorm

**Config/testthat/edition** 3

**URL** <https://janoleko.github.io/RTMBdist/>

**VignetteBuilder** knitr

**NeedsCompilation** no

**Author** Jan-Ole Koslik [aut, cre] (ORCID:  
<<https://orcid.org/0009-0004-1556-9053>>)

**Maintainer** Jan-Ole Koslik <[jan-ole.koslik@uni-bielefeld.de](mailto:jan-ole.koslik@uni-bielefeld.de)>

**Repository** CRAN

**Date/Publication** 2026-01-13 11:50:17 UTC

## Contents

|                        |    |
|------------------------|----|
| abs_smooth . . . . .   | 3  |
| bccg . . . . .         | 4  |
| bcpe . . . . .         | 5  |
| bct . . . . .          | 6  |
| beta2 . . . . .        | 7  |
| betabinom . . . . .    | 8  |
| cclayton . . . . .     | 9  |
| cfrank . . . . .       | 10 |
| cgaussian . . . . .    | 11 |
| cgmrf . . . . .        | 12 |
| cgumbel . . . . .      | 13 |
| cmvgauss . . . . .     | 14 |
| dcopula . . . . .      | 15 |
| ddcopula . . . . .     | 16 |
| dirichlet . . . . .    | 17 |
| dirmult . . . . .      | 18 |
| dmvcopula . . . . .    | 19 |
| erf . . . . .          | 20 |
| exgauss . . . . .      | 21 |
| foldnorm . . . . .     | 22 |
| gamma2 . . . . .       | 23 |
| genpois . . . . .      | 24 |
| gumbel . . . . .       | 25 |
| invgauss . . . . .     | 26 |
| laplace . . . . .      | 27 |
| mvt . . . . .          | 28 |
| nbinom2 . . . . .      | 29 |
| oibeta . . . . .       | 30 |
| oibeta2 . . . . .      | 31 |
| pareto . . . . .       | 32 |
| pgweibull . . . . .    | 33 |
| powerexp . . . . .     | 34 |
| skewnorm . . . . .     | 36 |
| skewnorm2 . . . . .    | 37 |
| skewt . . . . .        | 38 |
| skewt2 . . . . .       | 39 |
| t2 . . . . .           | 41 |
| truncnorm . . . . .    | 42 |
| trunct . . . . .       | 43 |
| trunct2 . . . . .      | 44 |
| vm . . . . .           | 45 |
| vmf . . . . .          | 46 |
| vmf2 . . . . .         | 47 |
| wishart . . . . .      | 48 |
| wrpcauchy . . . . .    | 48 |
| zero_inflate . . . . . | 49 |

|                      |           |
|----------------------|-----------|
| zibeta . . . . .     | 50        |
| zibeta2 . . . . .    | 51        |
| zibinom . . . . .    | 52        |
| zigamma . . . . .    | 53        |
| zigamma2 . . . . .   | 54        |
| ziinvgauss . . . . . | 55        |
| zilnorm . . . . .    | 56        |
| zinbinom . . . . .   | 57        |
| zinbinom2 . . . . .  | 58        |
| zipois . . . . .     | 59        |
| zoibeta . . . . .    | 60        |
| zoibeta2 . . . . .   | 61        |
| ztbinom . . . . .    | 62        |
| ztnbinom . . . . .   | 63        |
| ztnbinom2 . . . . .  | 64        |
| ztpois . . . . .     | 65        |
| <b>Index</b>         | <b>67</b> |

|            |                                                            |
|------------|------------------------------------------------------------|
| abs_smooth | <i>Smooth approximation to the absolute value function</i> |
|------------|------------------------------------------------------------|

Description

Smooth approximation to the absolute value function

Usage

abs\_smooth(x, epsilon = 1e-06)

Arguments

- x                      vector of evaluation points
- epsilon                smoothing constant

Details

We approximate the absolute value here as

$$|x| \approx \sqrt{x^2 + \epsilon}$$

Value

Smooth absolute value of x.

Examples

abs(0)  
abs\_smooth(0, 1e-4)

bccg

*Box–Cox Cole and Green distribution (BCCG)***Description**

Density, distribution function, quantile function, and random generation for the Box–Cox Cole and Green distribution.

**Usage**

```
dbccg(x, mu = 1, sigma = 0.1, nu = 1, log = FALSE)

pbccg(q, mu = 1, sigma = 0.1, nu = 1, lower.tail = TRUE, log.p = FALSE)

qbccg(p, mu = 1, sigma = 0.1, nu = 1, lower.tail = TRUE, log.p = FALSE)

rbccg(n, mu = 1, sigma = 0.1, nu = 1)
```

**Arguments**

|            |                                                                                      |
|------------|--------------------------------------------------------------------------------------|
| x, q       | vector of quantiles                                                                  |
| mu         | location parameter, must be positive.                                                |
| sigma      | scale parameter, must be positive.                                                   |
| nu         | skewness parameter (real).                                                           |
| log, log.p | logical; if TRUE, probabilities/ densities $p$ are returned as $\log(p)$ .           |
| lower.tail | logical; if TRUE (default), probabilities are $P[X \leq x]$ , otherwise $P[X > x]$ . |
| p          | vector of probabilities                                                              |
| n          | number of random values to return                                                    |

**Details**

This implementation of dbccg and pbccg allows for automatic differentiation with RTMB while the other functions are imported from gamlss.dist package. See gamlss.dist: [BCCG](#) for more details.

**Value**

dbccg gives the density, pbccg gives the distribution function, qbccg gives the quantile function, and rbccg generates random deviates.

**References**

Rigby, R. A., Stasinopoulos, D. M., Heller, G. Z., and De Bastiani, F. (2019) Distributions for modeling location, scale, and shape: Using GAMLSS in R, Chapman and Hall/CRC, doi:10.1201/9780429298547. An older version can be found in <https://www.gamlss.com/>.

**Examples**

```
x <- rbccg(5, mu = 10, sigma = 0.2, nu = 0.5)
d <- dbccg(x, mu = 10, sigma = 0.2, nu = 0.5)
p <- pbccg(x, mu = 10, sigma = 0.2, nu = 0.5)
q <- qbccg(p, mu = 10, sigma = 0.2, nu = 0.5)
```

bcpe

*Box-Cox Power Exponential distribution (BCPE)***Description**

Density, distribution function, quantile function, and random generation for the Box-Cox Power Exponential distribution.

**Usage**

```
dbcpe(x, mu = 5, sigma = 0.1, nu = 1, tau = 2, log = FALSE)

pbcpe(q, mu = 5, sigma = 0.1, nu = 1, tau = 2, lower.tail = TRUE, log.p = FALSE)

qbcpe(p, mu = 5, sigma = 0.1, nu = 1, tau = 2, lower.tail = TRUE, log.p = FALSE)

rbcpe(n, mu = 5, sigma = 0.1, nu = 1, tau = 2)
```

**Arguments**

|            |                                                                                      |
|------------|--------------------------------------------------------------------------------------|
| x, q       | vector of quantiles                                                                  |
| mu         | location parameter, must be positive.                                                |
| sigma      | scale parameter, must be positive.                                                   |
| nu         | vector of nu parameter values.                                                       |
| tau        | vector of tau parameter values, must be positive.                                    |
| log, log.p | logical; if TRUE, probabilities/ densities $p$ are returned as $\log(p)$ .           |
| lower.tail | logical; if TRUE (default), probabilities are $P[X \leq x]$ , otherwise $P[X > x]$ . |
| p          | vector of probabilities                                                              |
| n          | number of random values to return                                                    |

**Details**

This implementation of dbcpe and pbcpe allows for automatic differentiation with RTMB while the other functions are imported from gamlss.dist package. See gamlss.dist::BCPE for more details.

**Value**

dbcpe gives the density, pbcpe gives the distribution function, qbcpe gives the quantile function, and rbcpe generates random deviates.

## References

Rigby, R. A., Stasinopoulos, D. M., Heller, G. Z., and De Bastiani, F. (2019) Distributions for modeling location, scale, and shape: Using GAMLSS in R, Chapman and Hall/CRC, doi:10.1201/9780429298547. An older version can be found in <https://www.gamlss.com/>.

## Examples

```
x <- rbcpe(1, mu = 5, sigma = 0.1, nu = 1, tau = 1)
d <- dbcpe(x, mu = 5, sigma = 0.1, nu = 1, tau = 1)
p <- pbcpe(x, mu = 5, sigma = 0.1, nu = 1, tau = 1)
q <- qbcpe(p, mu = 5, sigma = 0.1, nu = 1, tau = 1)
```

---

bct

*Box–Cox t distribution (BCT)*


---

## Description

Density, distribution function, quantile function, and random generation for the Box–Cox t distribution.

## Usage

```
dbct(x, mu = 5, sigma = 0.1, nu = 1, tau = 2, log = FALSE)

pbct(q, mu = 5, sigma = 0.1, nu = 1, tau = 2, lower.tail = TRUE, log.p = FALSE)

qbct(p, mu = 5, sigma = 0.1, nu = 1, tau = 2, lower.tail = TRUE, log.p = FALSE)

rbct(n, mu = 5, sigma = 0.1, nu = 1, tau = 2)
```

## Arguments

|            |                                                                                      |
|------------|--------------------------------------------------------------------------------------|
| x, q       | vector of quantiles                                                                  |
| mu         | location parameter, must be positive.                                                |
| sigma      | scale parameter, must be positive.                                                   |
| nu         | skewness parameter (real).                                                           |
| tau        | degrees of freedom, must be positive.                                                |
| log, log.p | logical; if TRUE, probabilities/ densities $p$ are returned as $\log(p)$ .           |
| lower.tail | logical; if TRUE (default), probabilities are $P[X \leq x]$ , otherwise $P[X > x]$ . |
| p          | vector of probabilities                                                              |
| n          | number of random values to return                                                    |

## Details

This implementation of dbct and pbct allows for automatic differentiation with RTMB while the other functions are imported from gamlss.dist package. See gamlss.dist:[:BCT](#) for more details.

**Value**

dbct gives the density, pbct gives the distribution function, qbct gives the quantile function, and rbct generates random deviates.

**References**

Rigby, R. A., Stasinopoulos, D. M., Heller, G. Z., and De Bastiani, F. (2019) Distributions for modeling location, scale, and shape: Using GAMLSS in R, Chapman and Hall/CRC, doi:10.1201/9780429298547. An older version can be found in <https://www.gamlss.com/>.

**Examples**

```
x <- rbct(1, mu = 10, sigma = 0.2, nu = 0.5, tau = 4)
d <- dbct(x, mu = 10, sigma = 0.2, nu = 0.5, tau = 4)
p <- pbct(x, mu = 10, sigma = 0.2, nu = 0.5, tau = 4)
q <- qbct(p, mu = 10, sigma = 0.2, nu = 0.5, tau = 4)
```

beta2

*Reparameterised beta distribution***Description**

Density, distribution function, quantile function, and random generation for the beta distribution reparameterised in terms of mean and concentration.

**Usage**

```
dbeta(x, shape1, shape2, log = FALSE, eps = 0)

dbeta2(x, mu, phi, log = FALSE, eps = 0)

pbeta2(q, mu, phi, lower.tail = TRUE, log.p = FALSE)

qbeta2(p, mu, phi, lower.tail = TRUE, log.p = FALSE)

rbeta2(n, mu, phi)
```

**Arguments**

|                |                                                                                      |
|----------------|--------------------------------------------------------------------------------------|
| x, q           | vector of quantiles                                                                  |
| shape1, shape2 | non-negative parameters                                                              |
| log, log.p     | logical; if TRUE, probabilities/ densities $p$ are returned as $\log(p)$ .           |
| eps            | for internal use only, don't change.                                                 |
| mu             | mean parameter, must be in the interval from 0 to 1.                                 |
| phi            | concentration parameter, must be positive.                                           |
| lower.tail     | logical; if TRUE (default), probabilities are $P[X \leq x]$ , otherwise $P[X > x]$ . |
| p              | vector of probabilities                                                              |
| n              | number of random values to return.                                                   |

**Details**

This implementation allows for automatic differentiation with RTMB.

Currently, dbeta masks RTMB::dbeta because the latter has a numerically unstable gradient.

**Value**

dbeta2 gives the density, pbeta2 gives the distribution function, qbeta2 gives the quantile function, and rbeta2 generates random deviates.

**Examples**

```
set.seed(123)
x <- rbeta2(1, 0.5, 1)
d <- dbeta2(x, 0.5, 1)
p <- pbeta2(x, 0.5, 1)
q <- qbeta2(p, 0.5, 1)
```

---

|           |                                   |
|-----------|-----------------------------------|
| betabinom | <i>Beta-binomial distribution</i> |
|-----------|-----------------------------------|

---

**Description**

Density and random generation for the beta-binomial distribution.

**Usage**

```
dbetabinom(x, size, shape1, shape2, log = FALSE)

rbetabinom(n, size, shape1, shape2)
```

**Arguments**

|        |                                                                   |
|--------|-------------------------------------------------------------------|
| x      | vector of non-negative counts.                                    |
| size   | vector of total counts (number of trials). Needs to be $\geq x$ . |
| shape1 | positive shape parameter 1 of the Beta prior.                     |
| shape2 | positive shape parameter 2 of the Beta prior.                     |
| log    | logical; if TRUE, densities are returned on the log scale.        |
| n      | number of random values to return (for rbetabinom).               |

**Details**

This implementation of dbetabinom allows for automatic differentiation with RTMB.

**Value**

dbetabinom gives the density and rbetabinom generates random samples.

**Examples**

```
set.seed(123)
x <- rbetabinom(1, 10, 2, 5)
d <- dbetabinom(x, 10, 2, 5)
```

---

cclayton

*Clayton copula constructors*


---

**Description**

Construct a function that computes the log density or CDF of the bivariate Clayton copula, intended to be used with [dcopula](#).

**Usage**

```
cclayton(theta)
```

```
Cclayton(theta)
```

**Arguments**

theta                      Positive dependence parameter ( $\theta > 0$ ).

**Details**

The Clayton copula density is

$$c(u, v; \theta) = (1 + \theta)(uv)^{-(1+\theta)} (u^{-\theta} + v^{-\theta} - 1)^{-(2\theta+1)/\theta}, \quad \theta > 0.$$

**Value**

A function of two arguments (u,v) returning log copula **density** (cclayton) or copula **CDF** (Cclayton).

**See Also**

[cgaussian\(\)](#), [cgumbel\(\)](#), [cfrank\(\)](#)

**Examples**

```
x <- c(0.5, 1); y <- c(0.2, 0.8)
d1 <- dnorm(x, 1, log = TRUE); d2 <- dbeta(y, 2, 1, log = TRUE)
p1 <- pnorm(x, 1); p2 <- pbeta(y, 2, 1)
dcopula(d1, d2, p1, p2, copula = cclayton(2), log = TRUE)

# CDF version (for discrete copulas)
Cclayton(1.5)(0.5, 0.4)
```

---

|        |                                 |
|--------|---------------------------------|
| cfrank | <i>Frank copula constructor</i> |
|--------|---------------------------------|

---

## Description

Returns a function computing the log density of the bivariate Frank copula, intended to be used with [dcopula](#).

## Usage

```
cfrank(theta)
```

```
Cfrank(theta)
```

## Arguments

theta                      Dependence parameter ( $\theta \neq 0$ ).

## Details

The Frank copula density is

$$c(u, v; \theta) = \frac{\theta(1 - e^{-\theta})e^{-\theta(u+v)}}{[(e^{-\theta u} - 1)(e^{-\theta v} - 1) + (1 - e^{-\theta})]^2}, \quad \theta \neq 0.$$

## Value

A function of two arguments (u, v) returning either the log copula density (cfrank) or the copula CDF (Cfrank).

## See Also

[cgaussian\(\)](#), [cclayton\(\)](#), [cgumbel\(\)](#)

## Examples

```
x <- c(0.5, 1); y <- c(1, 2)
d1 <- dnorm(x, 1, log = TRUE); d2 <- dexp(y, 2, log = TRUE)
p1 <- pnorm(x, 1); p2 <- pexp(y, 2)
dcopula(d1, d2, p1, p2, copula = cfrank(2), log = TRUE)
```

---

|           |                                    |
|-----------|------------------------------------|
| cgaussian | <i>Gaussian copula constructor</i> |
|-----------|------------------------------------|

---

## Description

Returns a function computing the log density of the bivariate Gaussian copula, intended to be used with [dcopula](#).

## Usage

```
cgaussian(rho = 0)
```

## Arguments

rho                      Correlation parameter ( $-1 < rho < 1$ ).

## Value

Function of two arguments (u,v) returning log copula density.

The Gaussian copula density is

$$c(u, v; \rho) = \frac{1}{\sqrt{1 - \rho^2}} \exp \left\{ -\frac{1}{2(1 - \rho^2)} (z_1^2 - 2\rho z_1 z_2 + z_2^2) + \frac{1}{2} (z_1^2 + z_2^2) \right\},$$

where  $z_1 = \Phi^{-1}(u)$ ,  $z_2 = \Phi^{-1}(v)$ , and  $-1 < \rho < 1$ .

## See Also

[cclayton\(\)](#), [cgumbel\(\)](#), [cfrank\(\)](#)

## Examples

```
x <- c(0.5, 1); y <- c(1, 2)
d1 <- dnorm(x, 1, log = TRUE); d2 <- dexp(y, 2, log = TRUE)
p1 <- pnorm(x, 1); p2 <- pexp(y, 2)
dcopula(d1, d2, p1, p2, copula = cgaussian(0.5), log = TRUE)
```

cgmrf

*Multivariate Gaussian copula constructor parameterised by inverse correlation matrix***Description**

Returns a function computing the log density of the multivariate Gaussian copula, parameterised by the inverse correlation matrix.

**Usage**

```
cgmrf(Q)
```

**Arguments**

**Q** Inverse of a positive definite correlation matrix with unit diagonal. Can either be sparse or dense matrix.

**Details**

**Caution:** Parameterising the inverse correlation directly is difficult, as inverting it needs to yield a positive definite matrix with **unit diagonal**. Hence we still advise parameterising the correlation matrix **R** and computing its inverse. This function is useful when you need access to the precision (i.e. inverse correlation) in your likelihood function.

**Value**

Function with matrix argument **U** returning log copula density.

**See Also**

[cmvgauss\(\)](#)

**Examples**

```
x <- c(0.5, 1); y <- c(1, 2); z <- c(0.2, 0.8)
d1 <- dnorm(x, 1, log = TRUE); d2 <- dexp(y, 2, log = TRUE); d3 <- dbeta(z, 2, 1, log = TRUE)
p1 <- pnorm(x, 1); p2 <- pexp(y, 2); p3 <- pbeta(z, 2, 1)
R <- matrix(c(1,0.5,0.3,0.5,1,0.4,0.3,0.4,1), nrow = 3)

## Based on correlation matrix
dmvcopula(cbind(d1, d2, d3), cbind(p1, p2, p3), copula = cmvgauss(R), log = TRUE)

## Based on precision matrix
Q <- solve(R)
dmvcopula(cbind(d1, d2, d3), cbind(p1, p2, p3), copula = cgmrf(Q), log = TRUE)

## Parameterisation inside a model
# using RTMB::unstructured to get a valid correlation matrix
```

```
library(RTMB)
d <- 5 # dimension
cor_func <- unstructured(d)
npar <- length(cor_func$parms())
R <- cor_func$corr(rep(0.1, npar))
```

cgumbel

*Gumbel copula constructors***Description**

Construct functions that compute either the log density or the CDF of the bivariate Gumbel copula, intended for use with [dcopula](#).

**Usage**

```
cgumbel(theta)
```

```
Cgumbel(theta)
```

**Arguments**

theta                      Dependence parameter ( $\theta \geq 1$ ).

**Details**

The Gumbel copula density

$$c(u, v; \theta) = \exp \left[ - \left( (-\log u)^\theta + (-\log v)^\theta \right)^{1/\theta} \right] \cdot h(u, v; \theta),$$

where  $h(u, v; \theta)$  contains the derivative terms ensuring the function is a density.

**Value**

A function of two arguments (u, v) returning either the log copula density (cgumbel) or the copula CDF (Cgumbel).

**See Also**

[cgaussian\(\)](#), [cclayton\(\)](#), [cfrank\(\)](#)

**Examples**

```
x <- c(0.5, 1); y <- c(0.2, 0.4)
d1 <- dnorm(x, 1, log = TRUE); d2 <- dbeta(y, 2, 1, log = TRUE)
p1 <- pnorm(x, 1); p2 <- pbeta(y, 2, 1)
dcopula(d1, d2, p1, p2, copula = cgumbel(1.5), log = TRUE)

# CDF version (for discrete copulas)
Cgumbel(1.5)(0.5, 0.4)
```

cmvgauss

*Multivariate Gaussian copula constructor***Description**

Returns a function computing the log density of the multivariate Gaussian copula, intended to be used with [dmvcopula](#).

**Usage**

```
cmvgauss(R)
```

**Arguments**

R                      Positive definite correlation matrix (unit diagonal)

**Value**

Function with matrix argument U returning log copula density.

**See Also**

[cgmrf\(\)](#)

**Examples**

```
x <- c(0.5, 1); y <- c(1, 2); z <- c(0.2, 0.8)
d1 <- dnorm(x, 1, log = TRUE); d2 <- dexp(y, 2, log = TRUE); d3 <- dbeta(z, 2, 1, log = TRUE)
p1 <- pnorm(x, 1); p2 <- pexp(y, 2); p3 <- pbeta(z, 2, 1)
R <- matrix(c(1,0.5,0.3,0.5,1,0.4,0.3,0.4,1), nrow = 3)

## Based on correlation matrix
dmvcopula(cbind(d1, d2, d3), cbind(p1, p2, p3), copula = cmvgauss(R), log = TRUE)

## Based on precision matrix
Q <- solve(R)
dmvcopula(cbind(d1, d2, d3), cbind(p1, p2, p3), copula = cgmrf(Q), log = TRUE)

## Parameterisation inside a model
# using RTMB::unstructured to get a valid correlation matrix
library(RTMB)
d <- 5 # dimension
cor_func <- unstructured(d)
npar <- length(cor_func$parms())
R <- cor_func$corr(rep(0.1, npar))
```

dcopula

*Joint density under a bivariate copula***Description**

Computes the joint density (or log-density) of a bivariate distribution constructed from two arbitrary margins combined with a specified copula.

**Usage**

```
dcopula(d1, d2, p1, p2, copula = cgaussian(), log = FALSE)
```

**Arguments**

|        |                                                                                                                                                                                    |
|--------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| d1, d2 | Marginal density values. If log = TRUE, supply the log-density. If log = FALSE, supply the raw density.                                                                            |
| p1, p2 | Marginal CDF values. Need not be supplied on log scale.                                                                                                                            |
| copula | A function of two arguments (u, v) returning the log copula density $\log c(u, v)$ . You can either construct this yourself or use the copula constructors available (see details) |
| log    | Logical; if TRUE, return the log joint density. In this case, d1 and d2 must be on the log scale.                                                                                  |

**Details**

The joint density is

$$f(x, y) = c(F_1(x), F_2(y)) f_1(x) f_2(y),$$

where  $F_i$  are the marginal CDFs,  $f_i$  are the marginal densities, and  $c$  is the copula density.

The marginal densities d1, d2 and CDFs p1, p2 must be differentiable for automatic differentiation (AD) to work.

Available copula constructors are:

- [cgaussian](#) (Gaussian copula)
- [cclayton](#) (Clayton copula)
- [cgumbel](#) (Gumbel copula)
- [cfrank](#) (Frank copula)

**Value**

Joint density (or log-density) under the bivariate copula.

**See Also**

[ddcopula\(\)](#), [dmvcopula\(\)](#)

## Examples

```
# Normal + Exponential margins with Gaussian copula
x <- c(0.5, 1); y <- c(1, 2)
d1 <- dnorm(x, 1, log = TRUE); d2 <- dexp(y, 2, log = TRUE)
p1 <- pnorm(x, 1); p2 <- pexp(y, 2)
ddcopula(d1, d2, p1, p2, copula = cgaussian(0.5), log = TRUE)

# Normal + Beta margins with Clayton copula
x <- c(0.5, 1); y <- c(0.2, 0.8)
d1 <- dnorm(x, 1, log = TRUE); d2 <- dbeta(y, 2, 1, log = TRUE)
p1 <- pnorm(x, 1); p2 <- pbeta(y, 2, 1)
ddcopula(d1, d2, p1, p2, copula = cclayton(2), log = TRUE)

# Normal + Beta margins with Gumbel copula
x <- c(0.5, 1); y <- c(0.2, 0.4)
d1 <- dnorm(x, 1, log = TRUE); d2 <- dbeta(y, 2, 1, log = TRUE)
p1 <- pnorm(x, 1); p2 <- pbeta(y, 2, 1)
ddcopula(d1, d2, p1, p2, copula = cgumbel(1.5), log = TRUE)

# Normal + Exponential margins with Frank copula
x <- c(0.5, 1); y <- c(1, 2)
d1 <- dnorm(x, 1, log = TRUE); d2 <- dexp(y, 2, log = TRUE)
p1 <- pnorm(x, 1); p2 <- pexp(y, 2)
ddcopula(d1, d2, p1, p2, copula = cfrank(2), log = TRUE)
```

---

ddcopula

*Joint probability under a discrete bivariate copula*


---

## Description

Computes the joint probability mass function of two **discrete** margins combined with a copula CDF.

## Usage

```
ddcopula(d1, d2, p1, p2, Copula, log = FALSE)
```

## Arguments

|        |                                                                                                   |
|--------|---------------------------------------------------------------------------------------------------|
| d1, d2 | Marginal p.m.f. values at the observed points, <b>not</b> on log-scale.                           |
| p1, p2 | Marginal CDF values at the observed points.                                                       |
| Copula | A function of two arguments returning the copula CDF.                                             |
| log    | Logical; if TRUE, return the log joint density. In this case, d1 and d2 must be on the log scale. |

**Details**

The joint probability mass function for two discrete margins is

$$\Pr(Y_1 = y_1, Y_2 = y_2) = C(F_1(y_1), F_2(y_2)) - C(F_1(y_1 - 1), F_2(y_2)) - C(F_1(y_1), F_2(y_2 - 1)) + C(F_1(y_1 - 1), F_2(y_2 - 1)),$$

where  $F_i$  are the marginal CDFs, and  $C$  is the copula CDF.

Available copula CDF constructors are:

- [Cclayton](#) (Clayton copula)
- [Cgumbel](#) (Gumbel copula)
- [Cfrank](#) (Frank copula)

**Value**

Joint probability (or log-probability) under chosen copula

**See Also**

[dcopula\(\)](#), [dmvcopula\(\)](#)

**Examples**

```
x <- c(3,5); y <- c(2,4)
d1 <- dpois(x, 4); d2 <- dpois(y, 3)
p1 <- ppois(x, 4); p2 <- ppois(y, 3)
ddcopula(d1, d2, p1, p2, Copula = Cclayton(2), log = FALSE)
```

---

dirichlet

*Dirichlet distribution*


---

**Description**

Density and random generation for the Dirichlet distribution.

**Usage**

```
ddirichlet(x, alpha, log = FALSE)
```

```
rdirichlet(n, alpha)
```

**Arguments**

|       |                                                                                                                        |
|-------|------------------------------------------------------------------------------------------------------------------------|
| x     | vector or matrix of quantiles. If x is a vector, it needs to sum to one. If x is a matrix, each row should sum to one. |
| alpha | vector or matrix of positive shape parameters                                                                          |
| log   | logical; if TRUE, densities $p$ are returned as $\log(p)$ .                                                            |
| n     | number of random values to return.                                                                                     |

**Details**

This implementation of `ddirichlet` allows for automatic differentiation with RTMB.

**Value**

`ddirichlet` gives the density, `rdirichlet` generates random deviates.

**Examples**

```
# single alpha
alpha <- c(1,2,3)
x <- rdirichlet(1, alpha)
d <- ddirichlet(x, alpha)
# vectorised over alpha
alpha <- rbind(alpha, 2*alpha)
x <- rdirichlet(2, alpha)
```

---

dirmult

*Dirichlet-multinomial distribution*


---

**Description**

Density and random generation for the Dirichlet-multinomial distribution.

**Usage**

```
ddirmult(x, size, alpha, log = FALSE)
```

```
rdirmult(n, size, alpha)
```

**Arguments**

|                    |                                                                                                  |
|--------------------|--------------------------------------------------------------------------------------------------|
| <code>x</code>     | vector or matrix of non-negative counts, where rows are observations and columns are categories. |
| <code>size</code>  | vector of total counts for each observation. Needs to match the row sums of <code>x</code> .     |
| <code>alpha</code> | vector or matrix of positive shape parameters                                                    |
| <code>log</code>   | logical; if TRUE, densities $p$ are returned as $\log(p)$ .                                      |
| <code>n</code>     | number of random values to return.                                                               |

**Details**

This implementation of `ddirmult` allows for automatic differentiation with RTMB.

**Value**

`ddirmult` gives the density and `rdirmult` generates random samples.

## Examples

```
# single alpha
alpha <- c(1,2,3)
size <- 10
x <- rdirmult(1, size, alpha)
d <- ddirmult(x, size, alpha)
# vectorised over alpha and size
alpha <- rbind(alpha, 2*alpha)
size <- c(size, 3*size)
x <- rdirmult(2, size, alpha)
```

dmvcopula

*Joint density under a multivariate copula*

## Description

Computes the joint density (or log-density) of a distribution constructed from any number of arbitrary margins combined with a specified copula.

## Usage

```
dmvcopula(D, P, copula = cmvgauss(diag(ncol(D))), log = FALSE)
```

## Arguments

|        |                                                                                                                                                                                                                                         |
|--------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| D      | Matrix of marginal density values of with rows corresponding to observations and columns corresponding to dimensions. If log = TRUE, supply the log-densities. If log = FALSE, supply the raw densities                                 |
| P      | Matrix of marginal CDF values of the same dimension as D. Need not be supplied on log scale.                                                                                                                                            |
| copula | A function of a matrix argument U returning the log copula density $\log c(u_1, \dots, u_d)$ . The columns of U correspond to dimensions. You can either construct this yourself or use the copula constructors available (see details) |
| log    | Logical; if TRUE, return the log joint density. In this case, D must be on the log scale.                                                                                                                                               |

## Details

The joint density is

$$f(x_1, \dots, x_d) = c(F_1(x_1), \dots, F_d(x_d)) f_1(x_1) \dots f_d(x_d),$$

where  $F_i$  are the marginal CDFs,  $f_i$  are the marginal densities, and  $c$  is the copula density.

The marginal densities  $d_1, \dots, d_d$  and CDFs  $p_1, \dots, p_d$  must be differentiable for automatic differentiation (AD) to work.

Available multivariate copula constructors are:

- [cmvgauss](#) (Multivariate Gaussian copula)
- [cgmrnf](#) (Multivariate Gaussian copula parameterised by precision (inverse correlation) matrix)

**Value**

Joint density (or log-density) under the chosen copula.

**See Also**

[dcopula\(\)](#), [ddcopula\(\)](#)

**Examples**

```
x <- c(0.5, 1); y <- c(1, 2); z <- c(0.2, 0.8)
d1 <- dnorm(x, 1, log = TRUE); d2 <- dexp(y, 2, log = TRUE); d3 <- dbeta(z, 2, 1, log = TRUE)
p1 <- pnorm(x, 1); p2 <- pexp(y, 2); p3 <- pbeta(z, 2, 1)
R <- matrix(c(1,0.5,0.3,0.5,1,0.4,0.3,0.4,1), nrow = 3)

### Multivariate Gaussian copula
## Based on correlation matrix
dmvcpula(cbind(d1, d2, d3), cbind(p1, p2, p3), copula = cmvgauss(R), log = TRUE)

## Based on precision matrix
Q <- solve(R)
dmvcpula(cbind(d1, d2, d3), cbind(p1, p2, p3), copula = cgmrf(Q), log = TRUE)

## Parameterisation inside a model
# using RTMB::unstructured to get a valid correlation matrix
library(RTMB)
d <- 5 # dimension
cor_func <- unstructured(d)
npar <- length(cor_func$parms())
R <- cor_func$corr(rep(0.1, npar))
```

---

erf

---

*AD-compatible error function and complementary error function*


---

**Description**

AD-compatible error function and complementary error function

**Usage**

```
erf(x)
```

```
erfc(x)
```

**Arguments**

x                      vector of evaluation points

**Value**

erf(x) returns the error function and erfc(x) returns the complementary error function.

**Examples**

```
erf(1)
erfc(1)
```

---

exgauss

*Exponentially modified Gaussian distribution*


---

**Description**

Density, distribution function, quantile function, and random generation for the exponentially modified Gaussian distribution.

**Usage**

```
dexgauss(x, mu = 0, sigma = 1, lambda = 1, log = FALSE)

pexgauss(q, mu = 0, sigma = 1, lambda = 1, lower.tail = TRUE, log.p = FALSE)

qexgauss(p, mu = 0, sigma = 1, lambda = 1, lower.tail = TRUE, log.p = FALSE)

rexgauss(n, mu = 0, sigma = 1, lambda = 1)
```

**Arguments**

|            |                                                                             |
|------------|-----------------------------------------------------------------------------|
| x, q       | vector of quantiles                                                         |
| mu         | mean parameter of the Gaussian part                                         |
| sigma      | standard deviation parameter of the Gaussian part, must be positive.        |
| lambda     | rate parameter of the exponential part, must be positive.                   |
| log, log.p | logical; if TRUE, probabilities/ densities $p$ are returned as $\log(p)$ .  |
| lower.tail | logical; if TRUE, probabilities are $P[X \leq x]$ , otherwise, $P[X > x]$ . |
| p          | vector of probabilities                                                     |
| n          | number of random values to return                                           |

**Details**

This implementation of dexgauss and pexgauss allows for automatic differentiation with RTMB. qexgauss and rexgauss are reparameterised imports from gamlss.dist::exGAUS.

If  $X \sim N(\mu, \sigma^2)$  and  $Y \sim \text{Exp}(\lambda)$ , then  $Z = X + Y$  follows the exponentially modified Gaussian distribution with parameters  $\mu$ ,  $\sigma$ , and  $\lambda$ .

**Value**

dexgauss gives the density, pexgauss gives the distribution function, qexgauss gives the quantile function, and rexgauss generates random deviates.

## References

Rigby, R. A., Stasinopoulos, D. M., Heller, G. Z., and De Bastiani, F. (2019) Distributions for modeling location, scale, and shape: Using GAMLSS in R, Chapman and Hall/CRC, doi:10.1201/9780429298547. An older version can be found in <https://www.gamlss.com/>.

## Examples

```
x <- rexgauss(1, 1, 2, 2)
d <- dexgauss(x, 1, 2, 2)
p <- pexgauss(x, 1, 2, 2)
q <- qexgauss(p, 1, 2, 2)
```

---

foldnorm

*Folded normal distribution*


---

## Description

Density, distribution function, and random generation for the folded normal distribution.

## Usage

```
dfoldnorm(x, mu = 0, sigma = 1, log = FALSE)

pfoldnorm(q, mu = 0, sigma = 1, lower.tail = TRUE, log.p = FALSE)

rfoldnorm(n, mu = 0, sigma = 1)
```

## Arguments

|            |                                                                             |
|------------|-----------------------------------------------------------------------------|
| x, q       | vector of quantiles                                                         |
| mu         | location parameter                                                          |
| sigma      | scale parameter, must be positive.                                          |
| log, log.p | logical; if TRUE, probabilities/ densities $p$ are returned as $\log(p)$ .  |
| lower.tail | logical; if TRUE, probabilities are $P[X \leq x]$ , otherwise, $P[X > x]$ . |
| n          | number of random values to return                                           |
| p          | vector of probabilities                                                     |

## Details

This implementation of `dfoldnorm` allows for automatic differentiation with RTMB.

## Value

`dfoldnorm` gives the density, `pfoldnorm` gives the distribution function, and `rfoldnorm` generates random deviates.

**Examples**

```
x <- rfoldnorm(1, 1, 2)
d <- dfoldnorm(x, 1, 2)
p <- pfoldnorm(x, 1, 2)
```

gamma2

*Reparameterised gamma distribution***Description**

Density, distribution function, quantile function, and random generation for the gamma distribution reparameterised in terms of mean and standard deviation.

**Usage**

```
dgamma2(x, mean = 1, sd = 1, log = FALSE)

pgamma2(q, mean = 1, sd = 1, lower.tail = TRUE, log.p = FALSE)

qgamma2(p, mean = 1, sd = 1, lower.tail = TRUE, log.p = FALSE)

rgamma2(n, mean = 1, sd = 1)
```

**Arguments**

|            |                                                                             |
|------------|-----------------------------------------------------------------------------|
| x, q       | vector of quantiles                                                         |
| mean       | mean parameter, must be positive.                                           |
| sd         | standard deviation parameter, must be positive.                             |
| log, log.p | logical; if TRUE, probabilities/ densities $p$ are returned as $\log(p)$ .  |
| lower.tail | logical; if TRUE, probabilities are $P[X \leq x]$ , otherwise, $P[X > x]$ . |
| p          | vector of probabilities                                                     |
| n          | number of random values to return.                                          |

**Details**

This implementation allows for automatic differentiation with RTMB.

**Value**

dgamma2 gives the density, pgamma2 gives the distribution function, qgamma2 gives the quantile function, and rgamma2 generates random deviates.

**Examples**

```
x <- rgamma2(1)
d <- dgamma2(x)
p <- pgamma2(x)
q <- qgamma2(p)
```

genpois

*Generalised Poisson distribution***Description**

Probability mass function, distribution function, and random generation for the generalised Poisson distribution.

**Usage**

```
dgenpois(x, lambda = 1, phi = 1, log = FALSE)

pgenpois(q, lambda = 1, phi = 1, lower.tail = TRUE, log.p = FALSE)

qgenpois(p, lambda = 1, phi = 1,
         lower.tail = TRUE, log.p = FALSE, max.value = 10000)

rgenpois(n, lambda = 1, phi = 1, max.value = 10000)
```

**Arguments**

|            |                                                                                            |
|------------|--------------------------------------------------------------------------------------------|
| x, q       | integer vector of counts                                                                   |
| lambda     | vector of positive means                                                                   |
| phi        | vector of non-negative dispersion parameters                                               |
| log, log.p | logical; return log-density if TRUE                                                        |
| lower.tail | logical; if TRUE, probabilities are $P[X \leq x]$ , otherwise, $P[X > x]$ .                |
| p          | vector of probabilities                                                                    |
| max.value  | a constant, set to the default value of 10000 for how far the algorithm should look for q. |
| n          | number of random values to return.                                                         |

**Details**

This implementation of `dgenpois` allows for automatic differentiation with RTMB. The other functions are imported from `gamlss.dist::GPO`.

The distribution has mean  $\lambda$  and variance  $\lambda(1 + \phi\lambda)^2$ . For  $\phi = 0$  it reduces to the Poisson distribution, however  $\phi$  must be strictly positive here.

**Value**

`dgenpois` gives the probability mass function, `pgenpois` gives the distribution function, `qgenpois` gives the quantile function, and `rgenpois` generates random deviates.

**Examples**

```

set.seed(123)
x <- rgenpois(1, 2, 3)
d <- dgenpois(x, 2, 3)
p <- pgenpois(x, 2, 3)
q <- qgenpois(p, 2, 3)

```

gumbel

*Gumbel distribution***Description**

Density, distribution function, quantile function, and random generation for the Gumbel distribution.

**Usage**

```

dgumbel(x, location = 0, scale = 1, log = FALSE)

pgumbel(q, location = 0, scale = 1, lower.tail = TRUE, log.p = FALSE)

qgumbel(p, location = 0, scale = 1, lower.tail = TRUE, log.p = FALSE)

rgumbel(n, location = 0, scale = 1)

```

**Arguments**

|            |                                                                             |
|------------|-----------------------------------------------------------------------------|
| x, q       | vector of quantiles                                                         |
| location   | location parameter                                                          |
| scale      | scale parameter, must be positive.                                          |
| log, log.p | logical; if TRUE, probabilities/ densities $p$ are returned as $\log(p)$ .  |
| lower.tail | logical; if TRUE, probabilities are $P[X \leq x]$ , otherwise, $P[X > x]$ . |
| p          | vector of probabilities                                                     |
| n          | number of random values to return                                           |

**Details**

This implementation of `dgumbel` allows for automatic differentiation with RTMB.

**Value**

`dgumbel` gives the density, `pgumbel` gives the distribution function, `qgumbel` gives the quantile function, and `rgumbel` generates random deviates.

**Examples**

```
x <- rgumbel(1, 0.5, 2)
d <- dgumbel(x, 0.5, 2)
p <- pgumbel(x, 0.5, 2)
q <- qgumbel(p, 0.5, 2)
```

---

invgauss

*Inverse Gaussian distribution*


---

**Description**

Density, distribution function, and random generation for the inverse Gaussian distribution.

**Usage**

```
dinvgauss(x, mean = 1, shape = 1, log = FALSE)

pinvgauss(q, mean = 1, shape = 1, lower.tail = TRUE, log.p = FALSE)

qinvgauss(p, mean = 1, shape = 1, lower.tail = TRUE, log.p = FALSE, ...)

rinvgauss(n, mean = 1, shape = 1)
```

**Arguments**

|            |                                                                                                                   |
|------------|-------------------------------------------------------------------------------------------------------------------|
| x, q       | vector of quantiles, must be positive.                                                                            |
| mean       | location parameter                                                                                                |
| shape      | shape parameter, must be positive.                                                                                |
| log, log.p | logical; if TRUE, probabilities/ densities $p$ are returned as $\log(p)$ .                                        |
| lower.tail | logical; if TRUE, probabilities are $P[X \leq x]$ , otherwise, $P[X > x]$ .                                       |
| p          | vector of probabilities                                                                                           |
| ...        | additional parameter passed to <code>statmod::qinvgauss</code> for numerical evaluation of the quantile function. |
| n          | number of random values to return                                                                                 |

**Details**

This implementation of `dinvgauss` allows for automatic differentiation with RTMB. `qinvgauss` and `rinvgauss` are imported from the `statmod` package.

**Value**

`dinvgauss` gives the density, `pinvgauss` gives the distribution function, `qinvgauss` gives the quantile function, and `rinvgauss` generates random deviates.

**Examples**

```
x <- rinvgauss(1, 1, 0.5)
d <- dinvgauss(x, 1, 0.5)
p <- pinvgauss(x, 1, 0.5)
q <- qinvgauss(p, 1, 0.5)
```

laplace

*Laplace distribution***Description**

Density, distribution function, quantile function, and random generation for the Laplace distribution.

**Usage**

```
dlaplace(x, mu = 0, b = 1, eps = NULL, log = FALSE)

plaplace(q, mu = 0, b = 1, lower.tail = TRUE, log.p = FALSE)

qlaplace(p, mu = 0, b = 1, lower.tail = TRUE, log.p = FALSE)

rlaplace(n, mu = 0, b = 1)
```

**Arguments**

|            |                                                                                                                                                                                                                      |
|------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| x, q       | vector of quantiles                                                                                                                                                                                                  |
| mu         | location parameter                                                                                                                                                                                                   |
| b          | scale parameter, must be positive.                                                                                                                                                                                   |
| eps        | optional smoothing parameter for dlaplace to smooth the absolute value function. See <a href="#">abs_smooth</a> for details. It is recommended to set this to a small constant like 1e-6 for numerical optimisation. |
| log, log.p | logical; if TRUE, probabilities/ densities $p$ are returned as $\log(p)$ .                                                                                                                                           |
| lower.tail | logical; if TRUE, probabilities are $P[X \leq x]$ , otherwise, $P[X > x]$ .                                                                                                                                          |
| p          | vector of probabilities                                                                                                                                                                                              |
| n          | number of random values to return                                                                                                                                                                                    |

**Details**

This implementation of dlaplace allows for automatic differentiation with RTMB.

**Value**

dlaplace gives the density, plaplace gives the distribution function, qlaplace gives the quantile function, and rlaplace generates random deviates.

**Examples**

```
x <- rlaplace(1, 1, 1)
d <- dlaplace(x, 1, 1)
p <- plaplace(x, 1, 1)
q <- qlaplace(p, 1, 1)
```

mvt

*Multivariate t distribution***Description**

Density and random generation for the multivariate t distribution

**Usage**

```
dmvt(x, mu, Sigma, df, log = FALSE)

rmvt(n, mu, Sigma, df)
```

**Arguments**

|       |                                                                                     |
|-------|-------------------------------------------------------------------------------------|
| x     | vector or matrix of quantiles                                                       |
| mu    | vector or matrix of location parameters (mean if $df > 1$ )                         |
| Sigma | positive definite scale matrix (proportional to the covariance matrix if $df > 2$ ) |
| df    | degrees of freedom; must be positive                                                |
| log   | logical; if TRUE, densities $p$ are returned as $\log(p)$ .                         |
| n     | number of random values to return.                                                  |

**Details**

This implementation of dmvt allows for automatic differentiation with RTMB.

Note: for  $df \leq 1$  the mean is undefined, and for  $df \leq 2$  the covariance is infinite. For  $df > 2$ , the covariance is  $df/(df-2) * \text{Sigma}$ .

**Value**

dmvt gives the density, rmvt generates random deviates.

**Examples**

```
# single mu
mu <- c(1,2,3)
Sigma <- diag(c(1,1,1))
df <- 5
x <- rmvt(2, mu, Sigma, df)
d <- dmvt(x, mu, Sigma, df)
```

```
# vectorised over mu
mu <- rbind(c(1,2,3), c(0, 0.5, 1))
x <- rmvt(2, mu, Sigma, df)
d <- dmvt(x, mu, Sigma, df)
```

nbinom2

*Reparameterised negative binomial distribution*

## Description

Probability mass function, distribution function, quantile function, and random generation for the negative binomial distribution reparameterised in terms of mean and size.

## Usage

```
dnbinom2(x, mu, size, log = FALSE)

pnbinom2(q, mu, size, lower.tail = TRUE, log.p = FALSE)

qnbinom2(p, mu, size, lower.tail = TRUE, log.p = FALSE)

rnbinom2(n, mu, size)

pnbinom(q, size, prob, lower.tail = TRUE, log.p = FALSE)
```

## Arguments

|            |                                                                             |
|------------|-----------------------------------------------------------------------------|
| x, q       | vector of quantiles                                                         |
| mu         | mean parameter, must be positive.                                           |
| size       | size parameter, must be positive.                                           |
| log, log.p | logical; if TRUE, probabilities/ densities $p$ are returned as $\log(p)$ .  |
| lower.tail | logical; if TRUE, probabilities are $P[X \leq x]$ , otherwise, $P[X > x]$ . |
| p          | vector of probabilities                                                     |
| n          | number of random values to return.                                          |
| prob       | probability of success in each trial. $0 < \text{prob} \leq 1$ .            |

## Details

This implementation allows for automatic differentiation with RTMB.

pnbinom is an AD-compatible implementation of the standard parameterisation of the CDF, missing from RTMB.

## Value

dnbinom2 gives the density, pnbinom2 gives the distribution function, qnbinom2 gives the quantile function, and rnbinom2 generates random deviates.

**Examples**

```

set.seed(123)
x <- rnbinom2(1, 1, 2)
d <- dnbinom2(x, 1, 2)
p <- pnbinom2(x, 1, 2)
q <- qnbinom2(p, 1, 2)

```

oibeta

*One-inflated beta distribution***Description**

Density, distribution function, and random generation for the one-inflated beta distribution.

**Usage**

```

doibeta(x, shape1, shape2, oneprob = 0, log = FALSE)

poibeta(q, shape1, shape2, oneprob = 0, lower.tail = TRUE, log.p = FALSE)

roibeta(n, shape1, shape2, oneprob = 0)

```

**Arguments**

|                |                                                                             |
|----------------|-----------------------------------------------------------------------------|
| x, q           | vector of quantiles                                                         |
| shape1, shape2 | non-negative shape parameters of the beta distribution                      |
| oneprob        | zero-inflation probability between 0 and 1.                                 |
| log, log.p     | logical; if TRUE, probabilities/ densities $p$ are returned as $\log(p)$ .  |
| lower.tail     | logical; if TRUE, probabilities are $P[X \leq x]$ , otherwise, $P[X > x]$ . |
| n              | number of random values to return.                                          |

**Details**

This implementation allows for automatic differentiation with RTMB.

**Value**

doibeta gives the density, poibeta gives the distribution function, and roibeta generates random deviates.

**Examples**

```

set.seed(123)
x <- roibeta(1, 2, 2, 0.5)
d <- doibeta(x, 2, 2, 0.5)
p <- poibeta(x, 2, 2, 0.5)

```

oibeta2

*Reparameterised one-inflated beta distribution***Description**

Density, distribution function, and random generation for the one-inflated beta distribution reparameterised in terms of mean and concentration.

**Usage**

```
doibeta2(x, mu, phi, oneprob = 0, log = FALSE)
poibeta2(q, mu, phi, oneprob = 0, lower.tail = TRUE, log.p = FALSE)
roibeta2(n, mu, phi, oneprob = 0)
```

**Arguments**

|            |                                                                             |
|------------|-----------------------------------------------------------------------------|
| x, q       | vector of quantiles                                                         |
| mu         | mean parameter, must be in the interval from 0 to 1.                        |
| phi        | concentration parameter, must be positive.                                  |
| oneprob    | zero-inflation probability between 0 and 1.                                 |
| log, log.p | logical; if TRUE, probabilities/ densities $p$ are returned as $\log(p)$ .  |
| lower.tail | logical; if TRUE, probabilities are $P[X \leq x]$ , otherwise, $P[X > x]$ . |
| n          | number of random values to return.                                          |

**Details**

This implementation allows for automatic differentiation with RTMB.

**Value**

doibeta2 gives the density, poibeta2 gives the distribution function, and roibeta2 generates random deviates.

**Examples**

```
set.seed(123)
x <- roibeta2(1, 0.6, 2, 0.5)
d <- doibeta2(x, 0.6, 2, 0.5)
p <- poibeta2(x, 0.6, 2, 0.5)
```

pareto

*Pareto distribution***Description**

Density, distribution function, quantile function, and random generation for the pareto distribution.

**Usage**

```
dpareto(x, mu = 1, log = FALSE)
ppareto(q, mu = 1, lower.tail = TRUE, log.p = FALSE)
qpareto(p, mu = 1, lower.tail = TRUE, log.p = FALSE)
rpareto(n, mu = 1)
```

**Arguments**

|            |                                                                                      |
|------------|--------------------------------------------------------------------------------------|
| x, q       | vector of quantiles                                                                  |
| mu         | location parameter, must be positive.                                                |
| log, log.p | logical; if TRUE, probabilities/ densities $p$ are returned as $\log(p)$ .           |
| lower.tail | logical; if TRUE (default), probabilities are $P[X \leq x]$ , otherwise $P[X > x]$ . |
| p          | vector of probabilities                                                              |
| n          | number of random values to return                                                    |

**Details**

This implementation of dpareto and ppareto allows for automatic differentiation with RTMB while the other functions are imported from gamlss.dist package. See gamlss.dist::[PARETO](#) for more details.

**Value**

dpareto gives the density, ppareto gives the distribution function, qpareto gives the quantile function, and rpareto generates random deviates.

**References**

Rigby, R. A., Stasinopoulos, D. M., Heller, G. Z., and De Bastiani, F. (2019) Distributions for modeling location, scale, and shape: Using GAMLSS in R, Chapman and Hall/CRC, doi:10.1201/9780429298547. An older version can be found in <https://www.gamlss.com/>.

**Examples**

```
set.seed(123)
x <- rpareto(1, mu = 5)
d <- dpareto(x, mu = 5)
p <- ppareto(x, mu = 5)
q <- qpareto(p, mu = 5)
```

pgweibull

*Power generalized Weibull distribution***Description**

Survival, hazard, cumulative distribution, density, quantile and sampling function for the power generalized Weibull (PgW) distribution with parameters scale, shape and powershape.

**Usage**

```
spgweibull(x, scale = 1, shape = 1, powershape = 1, log = FALSE)
hpgweibull(x, scale = 1, shape = 1, powershape = 1, log = FALSE)
ppgweibull(x, scale = 1, shape = 1, powershape = 1,
           lower.tail = TRUE, log.p = FALSE)
dpgweibull(x, scale = 1, shape = 1, powershape = 1, log = FALSE)
qpgweibull(p, scale = 1, shape = 1, powershape = 1)
rpgweibull(n, scale = 1, shape = 1, powershape = 1)
```

**Arguments**

|            |                                                                                      |
|------------|--------------------------------------------------------------------------------------|
| x          | vector of quantiles                                                                  |
| scale      | positive scale parameter                                                             |
| shape      | positive shape parameter                                                             |
| powershape | positive power shape parameter                                                       |
| log, log.p | logical; if TRUE, probabilities/ densities $p$ are returned as $\log(p)$ .           |
| lower.tail | logical; if TRUE (default), probabilities are $P[X \leq x]$ , otherwise $P[X > x]$ . |
| p          | vector of probabilities                                                              |
| n          | number of observations                                                               |

## Details

The survival function of the PgW distribution is:

$$S(x) = \exp \left\{ 1 - \left[ 1 + \left( \frac{x}{\theta} \right)^\nu \right]^{\frac{1}{\gamma}} \right\}.$$

The hazard function is

$$\frac{\nu}{\gamma \theta^\nu} \cdot x^{\nu-1} \cdot \left[ 1 + \left( \frac{x}{\theta} \right)^\nu \right]^{\frac{1}{\gamma-1}}$$

The cumulative distribution function is then  $F(x) = 1 - S(x)$  and the density function is  $S(x) \cdot h(x)$ .

If both shape parameters equal 1, the PgW distribution reduces to the exponential distribution (see [dexp](#)) with rate = 1/scale. If the power shape parameter equals 1, the PgW distribution simplifies to the Weibull distribution (see [dweibull](#)) with the same parametrization.

## Value

dpgweibull gives the density, ppgweibull gives the distribution function, qpgweibull gives the quantile function, and rpgweibull generates random deviates. spgweibull gives the survival function and hpgweibull gives the hazard function.

## Examples

```
x <- rpgweibull(1, 2, 2, 3)
d <- dpgweibull(x, 2, 2, 3)
p <- ppgweibull(x, 2, 2, 3)
q <- qpgweibull(p, 2, 2, 3)
s <- spgweibull(x, 2, 2, 3)
h <- hpgweibull(x, 2, 2, 3)
```

---

powerexp

*Power Exponential distribution (PE and PE2)*

---

## Description

Density, distribution function, quantile function, and random generation for the Power Exponential distribution (two versions).

## Usage

```
dpowerexp(x, mu = 0, sigma = 1, nu = 2, log = FALSE)

ppowerexp(q, mu = 0, sigma = 1, nu = 2, lower.tail = TRUE, log.p = FALSE)

qpowerexp(p, mu = 0, sigma = 1, nu = 2, lower.tail = TRUE, log.p = FALSE)

rpowerexp(n, mu = 0, sigma = 1, nu = 2)
```

```

dpowerexp2(x, mu = 0, sigma = 1, nu = 2, log = FALSE)

ppowerexp2(q, mu = 0, sigma = 1, nu = 2, lower.tail = TRUE, log.p = FALSE)

qpowerexp2(p, mu = 0, sigma = 1, nu = 2, lower.tail = TRUE, log.p = FALSE)

rpowerexp2(n, mu = 0, sigma = 1, nu = 2)

```

### Arguments

|            |                                                                                    |
|------------|------------------------------------------------------------------------------------|
| x, q       | vector of quantiles                                                                |
| mu         | location parameter                                                                 |
| sigma      | scale parameter, must be positive                                                  |
| nu         | shape parameter (real)                                                             |
| log, log.p | logical; if TRUE, probabilities/ densities $p$ are returned as $\log(p)$           |
| lower.tail | logical; if TRUE (default), probabilities are $P[X \leq x]$ , otherwise $P[X > x]$ |
| p          | vector of probabilities                                                            |
| n          | number of random values to return                                                  |

### Details

This implementation of the densities and distribution functions allow for automatic differentiation with RTMB while the other functions are imported from `gamlss.dist` package.

For `powerexp`, `mu` is the mean and `sigma` is the standard deviation while this does not hold for `powerexp2`.

See `gamlss.dist::PE` for more details.

### Value

`dpowerexp` gives the density, `ppowerexp` gives the distribution function, `qpowerexp` gives the quantile function, and `rpowerexp` generates random deviates.

### References

Rigby, R. A., Stasinopoulos, D. M., Heller, G. Z., and De Bastiani, F. (2019) Distributions for modeling location, scale, and shape: Using GAMLSS in R, Chapman and Hall/CRC, doi:10.1201/9780429298547. An older version can be found in <https://www.gamlss.com/>.

### Examples

```

# PE
x <- rpowerexp(1, mu = 0, sigma = 1, nu = 2)
d <- dpowerexp(x, mu = 0, sigma = 1, nu = 2)
p <- ppowerexp(x, mu = 0, sigma = 1, nu = 2)
q <- qpowerexp(p, mu = 0, sigma = 1, nu = 2)

# PE2

```

```
x <- rpowerexp2(1, mu = 0, sigma = 1, nu = 2)
d <- dpowerexp2(x, mu = 0, sigma = 1, nu = 2)
p <- ppowerexp2(x, mu = 0, sigma = 1, nu = 2)
q <- qpowerexp2(p, mu = 0, sigma = 1, nu = 2)
```

skewnorm

*Skew normal distribution***Description**

Density, distribution function, quantile function, and random generation for the skew normal distribution.

**Usage**

```
dskewnorm(x, xi = 0, omega = 1, alpha = 0, log = FALSE)

pskewnorm(q, xi = 0, omega = 1, alpha = 0, ...)

qskewnorm(p, xi = 0, omega = 1, alpha = 0, ...)

rskewnorm(n, xi = 0, omega = 1, alpha = 0)
```

**Arguments**

|       |                                                                                             |
|-------|---------------------------------------------------------------------------------------------|
| x, q  | vector of quantiles                                                                         |
| xi    | location parameter                                                                          |
| omega | scale parameter, must be positive.                                                          |
| alpha | skewness parameter, +/- Inf is allowed.                                                     |
| log   | logical; if TRUE, probabilities/ densities $p$ are returned as $\log(p)$ .                  |
| ...   | additional parameters to be passed to the sn package functions for pskewnorm and qskewnorm. |
| p     | vector of probabilities                                                                     |
| n     | number of random values to return                                                           |

**Details**

This implementation of dskewnorm allows for automatic differentiation with RTMB while the other functions are imported from the sn package. See [sn: dsn](#) for more details.

**Value**

dskewnorm gives the density, pskewnorm gives the distribution function, qskewnorm gives the quantile function, and rskewnorm generates random deviates.

**See Also**

[skewnorm2](#), [skewt](#), [skewt2](#)

**Examples**

```
# alpha is skew parameter
x <- rskewnorm(1, alpha = 1)
d <- dskewnorm(x, alpha = 1)
p <- pskewnorm(x, alpha = 1)
q <- qskewnorm(p, alpha = 1)
```

---

skewnorm2

*Reparameterised skew normal distribution*


---

**Description**

Density, distribution function, quantile function and random generation for the skew normal distribution reparameterised in terms of mean, standard deviation and skew magnitude

**Usage**

```
dskewnorm2(x, mean = 0, sd = 1, alpha = 0, log = FALSE)

pskewnorm2(q, mean = 0, sd = 1, alpha = 0, ...)

qskewnorm2(p, mean = 0, sd = 1, alpha = 0, ...)

rskewnorm2(n, mean = 0, sd = 1, alpha = 0)
```

**Arguments**

|       |                                                                                                                                     |
|-------|-------------------------------------------------------------------------------------------------------------------------------------|
| x, q  | vector of quantiles                                                                                                                 |
| mean  | mean parameter                                                                                                                      |
| sd    | standard deviation, must be positive.                                                                                               |
| alpha | skewness parameter, +/- Inf is allowed.                                                                                             |
| log   | logical; if TRUE, probabilities/ densities $p$ are returned as $\log(p)$ .                                                          |
| ...   | additional parameters to be passed to the <code>sn</code> package functions for <code>pskewnorm</code> and <code>qskewnorm</code> . |
| p     | vector of probabilities                                                                                                             |
| n     | number of random values to return                                                                                                   |

**Details**

This implementation of `dskewnorm2` allows for automatic differentiation with RTMB while the other functions are imported from the `sn` package.

**Value**

dskewnorm2 gives the density, pskewnorm2 gives the distribution function, qskewnorm2 gives the quantile function, and rskewnorm2 generates random deviates.

**See Also**

[skewnorm](#), [skewt](#), [skewt2](#)

**Examples**

```
# alpha is skew parameter
x <- rskewnorm2(1, alpha = 1)
d <- dskewnorm2(x, alpha = 1)
p <- pskewnorm2(x, alpha = 1)
q <- qskewnorm2(p, alpha = 1)
```

---

skewt

*Skewed students t distribution*


---

**Description**

Density, distribution function, quantile function, and random generation for the skew t distribution (type 2).

**Usage**

```
dskewt(x, mu = 0, sigma = 1, skew = 0, df = 100, log = FALSE)

pskewt(q, mu = 0, sigma = 1, skew = 0, df = 100,
       method = 0, lower.tail = TRUE, log.p = FALSE)

qskewt(p, mu = 0, sigma = 1, skew = 0, df = 100,
       tol = 1e-8, method = 0)

rskewt(n, mu = 0, sigma = 1, skew = 0, df = 100)
```

**Arguments**

|            |                                                                            |
|------------|----------------------------------------------------------------------------|
| x, q       | vector of quantiles                                                        |
| mu         | location parameter                                                         |
| sigma      | scale parameter, must be positive.                                         |
| skew       | skewness parameter, can be positive or negative.                           |
| df         | degrees of freedom, must be positive.                                      |
| log, log.p | logical; if TRUE, probabilities/ densities $p$ are returned as $\log(p)$ . |

|            |                                                                                                                                                                                                                                                                                                        |
|------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| method     | an integer value between 0 and 5 which selects the computing method; see ‘Details’ in the <a href="#">pst</a> documentation below for the meaning of these values. If method=0 (default value), an automatic choice is made among the four actual computing methods, depending on the other arguments. |
| lower.tail | logical; if TRUE, probabilities are $P[X \leq x]$ , otherwise, $P[X > x]$ .                                                                                                                                                                                                                            |
| p          | vector of probabilities                                                                                                                                                                                                                                                                                |
| tol        | a scalar value which regulates the accuracy of the result of qsn, measured on the probability scale.                                                                                                                                                                                                   |
| n          | number of random values to return.                                                                                                                                                                                                                                                                     |

### Details

This corresponds to the skew t type 2 distribution in GAMLSS ([ST2](#)), see pp. 411-412 of Rigby et al. (2019) and the version implemented in the sn package. This implementation of dskewt allows for automatic differentiation with RTMB while the other functions are imported from the sn package. See `sn:dst` for more details.

**Caution:** In a numerical optimisation, the skew parameter should NEVER be initialised with exactly zero. This will cause the initial and all subsequent derivatives to be exactly zero and hence the parameter will remain at its initial value.

### Value

dskewt gives the density, pskewt gives the distribution function, qskewt gives the quantile function, and rskewt generates random deviates.

### See Also

[skewt2](#), [skewnorm](#), [skewnorm2](#)

### Examples

```
x <- rskewt(1, 1, 2, 5, 2)
d <- dskewt(x, 1, 2, 5, 2)
p <- pskewt(x, 1, 2, 5, 2)
q <- qskewt(p, 1, 2, 5, 2)
```

---

skewt2

---

*Moment-parameterised skew t distribution*


---

### Description

Density, distribution function, quantile function, and random generation for the skew t distribution reparameterised so that mean and sd correspond to the *true* mean and standard deviation.

**Usage**

```

dskewt2(x, mean = 0, sd = 1, skew = 0, df = 100, log = FALSE)

pskewt2(q, mean = 0, sd = 1, skew = 0, df = 100,
        method = 0, lower.tail = TRUE, log.p = FALSE)

qskewt2(p, mean = 0, sd = 1, skew = 0, df = 100, tol = 1e-08, method = 0)

rskewt2(n, mean = 0, sd = 1, skew = 0, df = 100)

```

**Arguments**

|            |                                                                                                                                                                                                                                                                                                        |
|------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| x, q       | vector of quantiles                                                                                                                                                                                                                                                                                    |
| mean       | mean parameter                                                                                                                                                                                                                                                                                         |
| sd         | standard deviation parameter, must be positive.                                                                                                                                                                                                                                                        |
| skew       | skewness parameter, can be positive or negative.                                                                                                                                                                                                                                                       |
| df         | degrees of freedom, must be positive.                                                                                                                                                                                                                                                                  |
| log, log.p | logical; if TRUE, probabilities/ densities $p$ are returned as $\log(p)$ .                                                                                                                                                                                                                             |
| method     | an integer value between 0 and 5 which selects the computing method; see ‘Details’ in the <a href="#">pst</a> documentation below for the meaning of these values. If method=0 (default value), an automatic choice is made among the four actual computing methods, depending on the other arguments. |
| lower.tail | logical; if TRUE, probabilities are $P[X \leq x]$ , otherwise, $P[X > x]$ .                                                                                                                                                                                                                            |
| p          | vector of probabilities                                                                                                                                                                                                                                                                                |
| tol        | a scalar value which regulates the accuracy of the result of qsn, measured on the probability scale.                                                                                                                                                                                                   |
| n          | number of random values to return.                                                                                                                                                                                                                                                                     |

**Details**

This corresponds to the skew t type 2 distribution in GAMLSS ([ST2](#)), see pp. 411-412 of Rigby et al. (2019) and the version implemented in the `sn` package. However, it is reparameterised in terms of a standard deviation parameter `sd` rather than just a scale parameter `sigma`. Details of this reparameterisation are given below. This implementation of `dskewt` allows for automatic differentiation with RTMB while the other functions are imported from the `sn` package. See `sn::dst` for more details.

**Caution:** In a numerical optimisation, the skew parameter should NEVER be initialised with exactly zero. This will cause the initial and all subsequent derivatives to be exactly zero and hence the parameter will remain at its initial value.

For given skew =  $\alpha$  and df =  $\nu$ , define

$$\delta = \alpha / \sqrt{1 + \alpha^2}, \quad b_\nu = \sqrt{\nu/\pi} \Gamma((\nu - 1)/2) / \Gamma(\nu/2),$$

then

$$E(X) = \mu + \sigma \delta b_\nu, \quad Var(X) = \sigma^2 \left( \frac{\nu}{\nu - 2} - \delta^2 b_\nu^2 \right).$$

**Value**

dskewt2 gives the density, pskewt2 gives the distribution function, qskewt2 gives the quantile function, and rskewt2 generates random deviates.

**See Also**

[skewt](#), [skewnorm](#), [skewnorm2](#)

**Examples**

```
x <- rskewt2(1, 1, 2, 5, 5)
d <- dskewt2(x, 1, 2, 5, 5)
p <- pskewt2(x, 1, 2, 5, 5)
q <- qskewt2(p, 1, 2, 5, 5)
```

---

t2

---

*Student t distribution with location and scale*


---

**Description**

Density, distribution function, quantile function, and random generation for the t distribution with location and scale parameters.

**Usage**

```
dt2(x, mu, sigma, df, log = FALSE)

pt2(q, mu, sigma, df)

rt2(n, mu, sigma, df)

qt2(p, mu, sigma, df)

pt(q, df)
```

**Arguments**

|       |                                                                            |
|-------|----------------------------------------------------------------------------|
| x, q  | vector of quantiles                                                        |
| mu    | location parameter                                                         |
| sigma | scale parameter, must be positive.                                         |
| df    | degrees of freedom, must be positive.                                      |
| log   | logical; if TRUE, probabilities/ densities $p$ are returned as $\log(p)$ . |
| n     | number of random values to return.                                         |
| p     | vector of probabilities                                                    |

## Details

This implementation of `dt2` allows for automatic differentiation with RTMB.

## Value

`dt2` gives the density, `pt2` gives the distribution function, `qt2` gives the quantile function, and `rt2` generates random deviates.

## Examples

```
x <- rt2(1, 1, 2, 5)
d <- dt2(x, 1, 2, 5)
p <- pt2(x, 1, 2, 5)
q <- qt2(p, 1, 2, 5)
```

---

|           |                                      |
|-----------|--------------------------------------|
| truncnorm | <i>Truncated normal distribution</i> |
|-----------|--------------------------------------|

---

## Description

Density, distribution function, quantile function, and random generation for the truncated normal distribution.

## Usage

```
dtruncnorm(x, mean = 0, sd = 1, min = -Inf, max = Inf, log = FALSE)
```

```
ptruncnorm(q, mean = 0, sd = 1, min = -Inf, max = Inf,
  lower.tail = TRUE, log.p = FALSE)
```

```
qtruncnorm(p, mean = 0, sd = 1, min = -Inf, max = Inf,
  lower.tail = TRUE, log.p = FALSE)
```

```
rtruncnorm(n, mean = 0, sd = 1, min = -Inf, max = Inf)
```

## Arguments

|                         |                                                                             |
|-------------------------|-----------------------------------------------------------------------------|
| <code>x, q</code>       | vector of quantiles                                                         |
| <code>mean</code>       | mean parameter, must be positive.                                           |
| <code>sd</code>         | standard deviation parameter, must be positive.                             |
| <code>min, max</code>   | truncation bounds.                                                          |
| <code>log, log.p</code> | logical; if TRUE, probabilities/ densities $p$ are returned as $\log(p)$ .  |
| <code>lower.tail</code> | logical; if TRUE, probabilities are $P[X \leq x]$ , otherwise, $P[X > x]$ . |
| <code>p</code>          | vector of probabilities                                                     |
| <code>n</code>          | number of random values to return.                                          |

## Details

This implementation of `dtruncnorm` allows for automatic differentiation with RTMB.

## Value

`dtruncnorm` gives the density, `ptruncnorm` gives the distribution function, `qtruncnorm` gives the quantile function, and `rtruncnorm` generates random deviates.

## Examples

```
x <- rtruncnorm(1, mean = 2, sd = 2, min = -1, max = 5)
d <- dtruncnorm(x, mean = 2, sd = 2, min = -1, max = 5)
p <- ptruncnorm(x, mean = 2, sd = 2, min = -1, max = 5)
q <- qtruncnorm(p, mean = 2, sd = 2, min = -1, max = 5)
```

---

|        |                                 |
|--------|---------------------------------|
| trunct | <i>Truncated t distribution</i> |
|--------|---------------------------------|

---

## Description

Density, distribution function, quantile function, and random generation for the truncated t distribution.

## Usage

```
dtrunct(x, df, min = -Inf, max = Inf, log = FALSE)

ptrunct(q, df, min = -Inf, max = Inf, lower.tail = TRUE, log.p = FALSE)

qtrunct(p, df, min = -Inf, max = Inf, lower.tail = TRUE, log.p = FALSE)

rtrunct(n, df, min = -Inf, max = Inf)
```

## Arguments

|                         |                                                                            |
|-------------------------|----------------------------------------------------------------------------|
| <code>x, q</code>       | vector of quantiles                                                        |
| <code>df</code>         | degrees of freedom parameter, must be positive.                            |
| <code>min, max</code>   | truncation bounds.                                                         |
| <code>log, log.p</code> | logical; if TRUE, probabilities/densities $p$ are returned as $\log(p)$ .  |
| <code>lower.tail</code> | logical; if TRUE, probabilities are $P[X \leq x]$ , otherwise $P[X > x]$ . |
| <code>p</code>          | vector of probabilities                                                    |
| <code>n</code>          | number of random values to return.                                         |

## Details

This implementation of `dtrunct` allows for automatic differentiation with RTMB.

**Value**

dtrunc2 gives the density, ptrunc2 gives the distribution function, qtrunc2 gives the quantile function, and rtrunc2 generates random deviates.

**Examples**

```
x <- rtrunc2(1, df = 5, min = -1, max = 5)
d <- dtrunc2(x, df = 5, min = -1, max = 5)
p <- ptrunc2(x, df = 5, min = -1, max = 5)
q <- qtrunc2(p, df = 5, min = -1, max = 5)
```

trunc2

*Truncated t distribution with location and scale***Description**

Density, distribution function, quantile function, and random generation for the truncated t distribution with location  $\mu$  and scale  $\sigma$ .

**Usage**

```
dtrunc2(x, df, mu = 0, sigma = 1, min = -Inf, max = Inf, log = FALSE)

ptrunc2(q, df, mu = 0, sigma = 1, min = -Inf, max = Inf,
        lower.tail = TRUE, log.p = FALSE)

qtrunc2(p, df, mu = 0, sigma = 1, min = -Inf, max = Inf,
        lower.tail = TRUE, log.p = FALSE)

rtrunc2(n, df, mu = 0, sigma = 1, min = -Inf, max = Inf)
```

**Arguments**

|            |                                                                            |
|------------|----------------------------------------------------------------------------|
| x, q       | vector of quantiles                                                        |
| df         | degrees of freedom parameter, must be positive.                            |
| mu         | location parameter.                                                        |
| sigma      | scale parameter, must be positive.                                         |
| min, max   | truncation bounds.                                                         |
| log, log.p | logical; if TRUE, probabilities/densities $p$ are returned as $\log(p)$ .  |
| lower.tail | logical; if TRUE, probabilities are $P[X \leq x]$ , otherwise $P[X > x]$ . |
| p          | vector of probabilities                                                    |
| n          | number of random values to return.                                         |

**Details**

This implementation of dtrunc2 allows for automatic differentiation with RTMB.

**Value**

dtrunc2 gives the density, ptrunc2 gives the distribution function, qtrunc2 gives the quantile function, and rtrunc2 generates random deviates.

**Examples**

```
x <- rtrunc2(1, df = 5, mu = 2, sigma = 3, min = -1, max = 5)
d <- dtrunc2(x, df = 5, mu = 2, sigma = 3, min = -1, max = 5)
p <- ptrunc2(x, df = 5, mu = 2, sigma = 3, min = -1, max = 5)
q <- qtrunc2(p, df = 5, mu = 2, sigma = 3, min = -1, max = 5)
```

vm

*von Mises distribution***Description**

Density, distribution function, and random generation for the von Mises distribution.

**Usage**

```
dvm(x, mu = 0, kappa = 1, log = FALSE)

pvm(q, mu = 0, kappa = 1, from = NULL, tol = 1e-20)

rvm(n, mu = 0, kappa = 1, wrap = TRUE)
```

**Arguments**

|       |                                                                                       |
|-------|---------------------------------------------------------------------------------------|
| x, q  | vector of angles measured in radians at which to evaluate the density function.       |
| mu    | mean direction of the distribution measured in radians.                               |
| kappa | non-negative numeric value for the concentration parameter of the distribution.       |
| log   | logical; if TRUE, densities are returned on the log scale.                            |
| from  | value from which the integration for CDF starts. If NULL, is set to $\mu - \pi$ .     |
| tol   | the precision in evaluating the distribution function                                 |
| n     | number of random values to return.                                                    |
| wrap  | logical; if TRUE, generated angles are wrapped to the interval from $-\pi$ to $\pi$ . |

**Details**

This implementation of dvm allows for automatic differentiation with RTMB. rvm and pvm are simply wrappers of the corresponding functions from circular.

**Value**

dvm gives the density, pvm gives the distribution function, and rvm generates random deviates.

**Examples**

```
set.seed(1)
x <- rvm(10, 0, 1)
d <- dvm(x, 0, 1)
p <- pvm(x, 0, 1)
```

vmf

*von Mises-Fisher distribution***Description**

Density, distribution function, and random generation for the von Mises-Fisher distribution.

**Usage**

```
dvmf(x, mu, kappa, log = FALSE)

rvmf(n, mu, kappa)
```

**Arguments**

|       |                                                                                 |
|-------|---------------------------------------------------------------------------------|
| x     | unit vector or matrix (with each row being a unit vector) of evaluation points  |
| mu    | unit mean vector                                                                |
| kappa | non-negative numeric value for the concentration parameter of the distribution. |
| log   | logical; if TRUE, densities are returned on the log scale.                      |
| n     | number of random values to return.                                              |

**Details**

This implementation of dvmf allows for automatic differentiation with RTMB. rvmf is a reparameterised import from movMF: :rmovMF.

**Value**

dvmf gives the density and rvm generates random deviates.

**Examples**

```
set.seed(123)
# single parameter set
mu <- rep(1, 3) / sqrt(3)
kappa <- 4
x <- rvmf(1, mu, kappa)
d <- dvmf(x, mu, kappa)

# vectorised over parameters
mu <- matrix(mu, nrow = 1)
```

```
mu <- mu[rep(1,10), ]
kappa <- rep(kappa, 10)
x <- rvmf(10, mu, kappa)
d <- dvmf(x, mu, kappa)
```

vmf2

*Reparameterised von Mises-Fisher distribution***Description**

Density, distribution function, and random generation for the von Mises-Fisher distribution.

**Usage**

```
dvmf2(x, theta, log = FALSE)
```

```
rvmf2(n, theta)
```

**Arguments**

|       |                                                                                                                                                                               |
|-------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| x     | unit vector or matrix (with each row being a unit vector) of evaluation points                                                                                                |
| theta | direction and concentration vector. The direction of theta determines the mean direction on the sphere. The norm of theta is the concentration parameter of the distribution. |
| log   | logical; if TRUE, densities are returned on the log scale.                                                                                                                    |
| n     | number of random values to return.                                                                                                                                            |

**Details**

In this parameterisation,  $\theta = \kappa\mu$ , where  $\mu$  is a unit vector and  $\kappa$  is the concentration parameter. dvmf2 allows for automatic differentiation with RTMB. rvmf2 is imported from movMF: :rmovMF.

**Value**

dvmf gives the density and rvm generates random deviates.

**Examples**

```
set.seed(123)
# single parameter set
theta <- c(1,2,3)
x <- rvmf2(1, theta)
d <- dvmf2(x, theta)

# vectorised over parameters
theta <- matrix(theta, nrow = 1)
theta <- theta[rep(1,10), ]
x <- rvmf2(10, theta)
d <- dvmf2(x, theta)
```

---

|         |                             |
|---------|-----------------------------|
| wishart | <i>Wishart distribution</i> |
|---------|-----------------------------|

---

### Description

Density and random generation for the wishart distribution

### Usage

```
dwishart(x, nu, Sigma, log = FALSE)
```

```
rwishart(n, nu, Sigma)
```

### Arguments

|       |                                                                           |
|-------|---------------------------------------------------------------------------|
| x     | positive definite p x p matrix of evaluation points                       |
| nu    | degrees of freedom, needs to be greater than p - 1                        |
| Sigma | scale matrix, needs to be positive definite and match the dimension of x. |
| log   | logical; if TRUE, densities $p$ are returned as $\log(p)$ .               |
| n     | number of random deviates to return                                       |

### Value

dwishart gives the density, rwishart generates random deviates (matrix for n = 1, array for n > 1)

### Examples

```
x <- rwishart(1, nu = 5, Sigma = diag(3))
d <- dwishart(x, nu = 5, Sigma = diag(3))
```

---

|           |                                    |
|-----------|------------------------------------|
| wrpcauchy | <i>wrapped Cauchy distribution</i> |
|-----------|------------------------------------|

---

### Description

Density and random generation for the wrapped Cauchy distribution.

### Usage

```
dwrpcauchy(x, mu = 0, rho, log = FALSE)
```

```
rwrpcauchy(n, mu = 0, rho, wrap = TRUE)
```

**Arguments**

|      |                                                                                   |
|------|-----------------------------------------------------------------------------------|
| x    | vector of angles measured in radians at which to evaluate the density function.   |
| mu   | mean direction of the distribution measured in radians.                           |
| rho  | concentration parameter of the distribution, must be in the interval from 0 to 1. |
| log  | logical; if TRUE, densities are returned on the log scale.                        |
| n    | number of random values to return.                                                |
| wrap | logical; if TRUE, generated angles are wrapped to the interval from -pi to pi.    |

**Details**

This implementation of `dwrpcauchy` allows for automatic differentiation with RTMB. `rwrpcauchy` is simply a wrapper for `rwrappedcauchy` imported from `circular`.

**Value**

`dwrpcauchy` gives the density and `rwrpcauchy` generates random deviates.

**Examples**

```
set.seed(1)
x <- rwrpcauchy(10, 0, 0.5)
d <- dwrpcauchy(x, 0, 0.5)
```

---

|              |                                          |
|--------------|------------------------------------------|
| zero_inflate | <i>Zero-inflated density constructor</i> |
|--------------|------------------------------------------|

---

**Description**

Constructs a zero-inflated density function from a given probability density function

**Usage**

```
zero_inflate(dist, discrete = NULL)
```

**Arguments**

|          |                                                                                                                                                                                                                                                                                                                                 |
|----------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| dist     | either a probability density function or a probability mass function                                                                                                                                                                                                                                                            |
| discrete | logical; if TRUE, the density for $x = 0$ will be <code>zeroprob + (1-zeroprob) * dist(0, ...)</code> . Otherwise it will just be <code>zeroprob</code> . In standard cases, this will be determined automatically. For non-standard cases, set this to TRUE or FALSE depending on the type of <code>dist</code> . See details. |

### Details

The definition of zero-inflation is different for discrete and continuous distributions. For discrete distributions with p.m.f.  $f$  and zero-inflation probability  $p$ , we have

$$\Pr(X = 0) = p + (1 - p) \cdot f(0),$$

and

$$\Pr(X = x) = (1 - p) \cdot f(x), \quad x > 0.$$

For continuous distributions with p.d.f.  $f$ , we have

$$f_{\text{zinfl}}(x) = p \cdot \delta_0(x) + (1 - p) \cdot f(x),$$

where  $\delta_0$  is the Dirac delta function at zero.

### Value

zero-inflated density function with first argument  $x$ , second argument `zeroprob`, and additional arguments ... that will be passed to `dist`.

### Examples

```
# Zero-inflated normal distribution
dzinorm <- zero_inflate(dnorm)
dzinorm(c(NA, 0, 2), 0.5, mean = 1, sd = 1)

# Zero-inflated Poisson distribution
zipois <- zero_inflate(dpois)
zipois(c(NA, 0, 1), 0.5, 1)

# Non-standard case: Zero-inflated reparametrised beta distribution
dzibeta2 <- zero_inflate(dbeta2, discrete = FALSE)
```

---

zibeta

Zero-inflated beta distribution

---

### Description

Density, distribution function, and random generation for the zero-inflated beta distribution.

### Usage

```
dzibeta(x, shape1, shape2, zeroprob = 0, log = FALSE)

pzibeta(q, shape1, shape2, zeroprob = 0, lower.tail = TRUE, log.p = FALSE)

rzibeta(n, shape1, shape2, zeroprob = 0)
```

**Arguments**

|                             |                                                                             |
|-----------------------------|-----------------------------------------------------------------------------|
| <code>x, q</code>           | vector of quantiles                                                         |
| <code>shape1, shape2</code> | non-negative shape parameters of the beta distribution                      |
| <code>zeroprob</code>       | zero-inflation probability between 0 and 1.                                 |
| <code>log, log.p</code>     | logical; if TRUE, probabilities/ densities $p$ are returned as $\log(p)$ .  |
| <code>lower.tail</code>     | logical; if TRUE, probabilities are $P[X \leq x]$ , otherwise, $P[X > x]$ . |
| <code>n</code>              | number of random values to return.                                          |

**Details**

This implementation allows for automatic differentiation with RTMB.

**Value**

`dzibeta` gives the density, `pzibeta` gives the distribution function, and `rzibeta` generates random deviates.

**Examples**

```
set.seed(123)
x <- rzibeta(1, 2, 2, 0.5)
d <- dzibeta(x, 2, 2, 0.5)
p <- pzibeta(x, 2, 2, 0.5)
```

---

zibeta2

---

*Reparameterised zero-inflated beta distribution*


---

**Description**

Density, distribution function, and random generation for the zero-inflated beta distribution reparameterised in terms of mean and concentration.

**Usage**

```
dzibeta2(x, mu, phi, zeroprob = 0, log = FALSE)

pzibeta2(q, mu, phi, zeroprob = 0, lower.tail = TRUE, log.p = FALSE)

rzibeta2(n, mu, phi, zeroprob = 0)
```

**Arguments**

|                         |                                                                                      |
|-------------------------|--------------------------------------------------------------------------------------|
| <code>x, q</code>       | vector of quantiles                                                                  |
| <code>mu</code>         | mean parameter, must be in the interval from 0 to 1.                                 |
| <code>phi</code>        | concentration parameter, must be positive.                                           |
| <code>zeroprob</code>   | zero-inflation probability between 0 and 1.                                          |
| <code>log, log.p</code> | logical; if TRUE, probabilities/ densities $p$ are returned as $\log(p)$ .           |
| <code>lower.tail</code> | logical; if TRUE (default), probabilities are $P[X \leq x]$ , otherwise $P[X > x]$ . |
| <code>n</code>          | number of random values to return.                                                   |
| <code>p</code>          | vector of probabilities                                                              |

**Details**

This implementation allows for automatic differentiation with RTMB.

**Value**

`dzibeta2` gives the density, `pzibeta2` gives the distribution function, and `rzibeta2` generates random deviates.

**Examples**

```
set.seed(123)
x <- rzibeta2(1, 0.5, 1, 0.5)
d <- dzibeta2(x, 0.5, 1, 0.5)
p <- pzibeta2(x, 0.5, 1, 0.5)
```

---

zibinom

---

*Zero-inflated binomial distribution*


---

**Description**

Probability mass function, distribution function, and random generation for the zero-inflated binomial distribution.

**Usage**

```
dzibinom(x, size, prob, zeroprob = 0, log = FALSE)

pzibinom(q, size, prob, zeroprob = 0, lower.tail = TRUE, log.p = FALSE)

rzibinom(n, size, prob, zeroprob = 0)
```

**Arguments**

|                         |                                                                             |
|-------------------------|-----------------------------------------------------------------------------|
| <code>x, q</code>       | vector of quantiles                                                         |
| <code>size</code>       | number of trials (zero or more).                                            |
| <code>prob</code>       | probability of success on each trial.                                       |
| <code>zeroprob</code>   | zero-inflation probability between 0 and 1                                  |
| <code>log, log.p</code> | logical; return log-density if TRUE                                         |
| <code>lower.tail</code> | logical; if TRUE, probabilities are $P[X \leq x]$ , otherwise, $P[X > x]$ . |
| <code>n</code>          | number of random values to return.                                          |

**Details**

This implementation allows for automatic differentiation with RTMB.

**Value**

`dzibinom` gives the probability mass function, `pzibinom` gives the distribution function, and `rzibinom` generates random deviates.

**Examples**

```
set.seed(123)
x <- rzibinom(1, size = 10, prob = 0.5, zeroprob = 0.5)
d <- dzibinom(x, size = 10, prob = 0.5, zeroprob = 0.5)
p <- pzibinom(x, size = 10, prob = 0.5, zeroprob = 0.5)
```

---

zigamma

*Zero-inflated gamma distribution*


---

**Description**

Density, distribution function, and random generation for the zero-inflated gamma distribution.

**Usage**

```
dzigamma(x, shape, scale, zeroprob = 0, log = FALSE)

pzigamma(q, shape, scale, zeroprob = 0, lower.tail = TRUE, log.p = FALSE)

rzigamma(n, shape, scale, zeroprob = 0)
```

**Arguments**

|                         |                                                                             |
|-------------------------|-----------------------------------------------------------------------------|
| <code>x, q</code>       | vector of quantiles                                                         |
| <code>shape</code>      | positive shape parameter                                                    |
| <code>scale</code>      | positive scale parameter                                                    |
| <code>zeroprob</code>   | zero-inflation probability between 0 and 1.                                 |
| <code>log, log.p</code> | logical; if TRUE, probabilities/ densities $p$ are returned as $\log(p)$ .  |
| <code>lower.tail</code> | logical; if TRUE, probabilities are $P[X \leq x]$ , otherwise, $P[X > x]$ . |
| <code>n</code>          | number of random values to return                                           |

**Details**

This implementation allows for automatic differentiation with RTMB.

**Value**

`dzigamma` gives the density, `pzigamma` gives the distribution function, and `rzigamma` generates random deviates.

**Examples**

```
x <- rzigamma(1, 1, 1, 0.5)
d <- dzigamma(x, 1, 1, 0.5)
p <- pzigamma(x, 1, 1, 0.5)
```

---

zigamma2

---

*Zero-inflated and reparameterised gamma distribution*


---

**Description**

Density, distribution function, and random generation for the zero-inflated gamma distribution reparameterised in terms of mean and standard deviation.

**Usage**

```
dzigamma2(x, mean = 1, sd = 1, zeroprob = 0, log = FALSE)

pzigamma2(q, mean = 1, sd = 1, zeroprob = 0)

rzigamma2(n, mean = 1, sd = 1, zeroprob = 0)
```

**Arguments**

|          |                                                                            |
|----------|----------------------------------------------------------------------------|
| x, q     | vector of quantiles                                                        |
| mean     | mean parameter, must be positive.                                          |
| sd       | standard deviation parameter, must be positive.                            |
| zeroprob | zero-inflation probability between 0 and 1.                                |
| log      | logical; if TRUE, probabilities/ densities $p$ are returned as $\log(p)$ . |
| n        | number of random values to return                                          |

**Details**

This implementation allows for automatic differentiation with RTMB.

**Value**

dzigamma2 gives the density, pzigamma2 gives the distribution function, and rzigamma generates random deviates.

**Examples**

```
x <- rzigamma2(1, 2, 1, 0.5)
d <- dzigamma2(x, 2, 1, 0.5)
p <- pzigamma2(x, 2, 1, 0.5)
```

---

ziinvgauss

*Zero-inflated inverse Gaussian distribution*


---

**Description**

Density, distribution function, and random generation for the zero-inflated inverse Gaussian distribution.

**Usage**

```
dziinvgauss(x, mean = 1, shape = 1, zeroprob = 0, log = FALSE)

pziinvgauss(q, mean = 1, shape = 1, zeroprob = 0, lower.tail = TRUE, log.p = FALSE)

rziinvgauss(n, mean = 1, shape = 1, zeroprob = 0)
```

**Arguments**

|                         |                                                                             |
|-------------------------|-----------------------------------------------------------------------------|
| <code>x, q</code>       | vector of quantiles                                                         |
| <code>mean</code>       | location parameter                                                          |
| <code>shape</code>      | shape parameter, must be positive.                                          |
| <code>zeroprob</code>   | zero-probability, must be in $[0, 1]$ .                                     |
| <code>log, log.p</code> | logical; if TRUE, probabilities/ densities $p$ are returned as $\log(p)$ .  |
| <code>lower.tail</code> | logical; if TRUE, probabilities are $P[X \leq x]$ , otherwise, $P[X > x]$ . |
| <code>n</code>          | number of random values to return                                           |

**Details**

This implementation of `zidinvgauss` allows for automatic differentiation with RTMB.

**Value**

`dziinvgauss` gives the density, `pziinvgauss` gives the distribution function, and `rziinvgauss` generates random deviates.

**Examples**

```
x <- rziinvgauss(1, 1, 2, 0.5)
d <- dziinvgauss(x, 1, 2, 0.5)
p <- pziinvgauss(x, 1, 2, 0.5)
```

---

zilnorm

---

Zero-inflated log normal distribution

---

**Description**

Density, distribution function, and random generation for the zero-inflated log normal distribution.

**Usage**

```
dzilnorm(x, meanlog = 0, sdlog = 1, zeroprob = 0, log = FALSE)

pzilnorm(q, meanlog = 0, sdlog = 1, zeroprob = 0,
         lower.tail = TRUE, log.p = FALSE)

rzilnorm(n, meanlog = 0, sdlog = 1, zeroprob = 0)

plnorm(q, meanlog = 0, sdlog = 1, lower.tail = TRUE, log.p = FALSE)
```

**Arguments**

|                             |                                                                                                               |
|-----------------------------|---------------------------------------------------------------------------------------------------------------|
| <code>x, q</code>           | vector of quantiles                                                                                           |
| <code>meanlog, sdlog</code> | mean and standard deviation of the distribution on the log scale with default values of 0 and 1 respectively. |
| <code>zeroprob</code>       | zero-inflation probability between 0 and 1.                                                                   |
| <code>log, log.p</code>     | logical; if TRUE, probabilities/ densities $p$ are returned as $\log(p)$ .                                    |
| <code>lower.tail</code>     | logical; if TRUE, probabilities are $P[X \leq x]$ , otherwise, $P[X > x]$ .                                   |
| <code>n</code>              | number of random values to return                                                                             |

**Details**

This implementation allows for automatic differentiation with RTMB.

**Value**

`dzilnorm` gives the density, `pzilnorm` gives the distribution function, and `rzilnorm` generates random deviates.

**Examples**

```
x <- rzilnorm(1, 1, 1, 0.5)
d <- dzilnorm(x, 1, 1, 0.5)
p <- pzilnorm(x, 1, 1, 0.5)
```

---

zinbinom

---

*Zero-inflated negative binomial distribution*


---

**Description**

Probability mass function, distribution function, quantile function, and random generation for the zero-inflated negative binomial distribution.

**Usage**

```
dzinbinom(x, size, prob, zeroprob = 0, log = FALSE)

pzinbinom(q, size, prob, zeroprob = 0, lower.tail = TRUE, log.p = FALSE)

rzinbinom(n, size, prob, zeroprob = 0)
```

**Arguments**

|                         |                                                                             |
|-------------------------|-----------------------------------------------------------------------------|
| <code>x, q</code>       | vector of (non-negative integer) quantiles                                  |
| <code>size</code>       | size parameter, must be positive.                                           |
| <code>prob</code>       | mean parameter, must be positive.                                           |
| <code>zeroprob</code>   | zero-inflation probability between 0 and 1.                                 |
| <code>log, log.p</code> | logical; if TRUE, probabilities/ densities $p$ are returned as $\log(p)$ .  |
| <code>lower.tail</code> | logical; if TRUE, probabilities are $P[X \leq x]$ , otherwise, $P[X > x]$ . |
| <code>n</code>          | number of random values to return.                                          |
| <code>p</code>          | vector of probabilities                                                     |

**Details**

This implementation allows for automatic differentiation with RTMB.

**Value**

`dzinbinom` gives the density, `pzinbinom` gives the distribution function, and `rzinbinom` generates random deviates.

**Examples**

```
set.seed(123)
x <- rzinbinom(1, size = 2, prob = 0.5, zeroprob = 0.5)
d <- dzinbinom(x, size = 2, prob = 0.5, zeroprob = 0.5)
p <- pzinbinom(x, size = 2, prob = 0.5, zeroprob = 0.5)
```

---

zinbinom2

---

*Zero-inflated and reparameterised negative binomial distribution*


---

**Description**

Probability mass function, distribution function, quantile function and random generation for the zero-inflated negative binomial distribution reparameterised in terms of mean and size.

**Usage**

```
dzinbinom2(x, mu, size, zeroprob = 0, log = FALSE)

pzinbinom2(q, mu, size, zeroprob = 0, lower.tail = TRUE, log.p = FALSE)

rzinbinom2(n, mu, size, zeroprob = 0)
```

**Arguments**

|                         |                                                                             |
|-------------------------|-----------------------------------------------------------------------------|
| <code>x, q</code>       | vector of (non-negative integer) quantiles                                  |
| <code>mu</code>         | mean parameter, must be positive.                                           |
| <code>size</code>       | size parameter, must be positive.                                           |
| <code>zeroprob</code>   | zero-inflation probability between 0 and 1.                                 |
| <code>log, log.p</code> | logical; if TRUE, probabilities/ densities $p$ are returned as $\log(p)$ .  |
| <code>lower.tail</code> | logical; if TRUE, probabilities are $P[X \leq x]$ , otherwise, $P[X > x]$ . |
| <code>n</code>          | number of random values to return.                                          |
| <code>p</code>          | vector of probabilities                                                     |

**Details**

This implementation allows for automatic differentiation with RTMB.

**Value**

`dzinbinom2` gives the density, `pzinbinom2` gives the distribution function, and `rzinbinom2` generates random deviates.

**Examples**

```
set.seed(123)
x <- rzinbinom2(1, 2, 1, zeroprob = 0.5)
d <- dzinbinom2(x, 2, 1, zeroprob = 0.5)
p <- pzinbinom2(x, 2, 1, zeroprob = 0.5)
```

---

|        |                                           |
|--------|-------------------------------------------|
| zipois | <i>Zero-inflated Poisson distribution</i> |
|--------|-------------------------------------------|

---

**Description**

Probability mass function, distribution function, and random generation for the zero-inflated Poisson distribution.

**Usage**

```
dzipois(x, lambda, zeroprob = 0, log = FALSE)

pzipois(q, lambda, zeroprob = 0, lower.tail = TRUE, log.p = FALSE)

rzipois(n, lambda, zeroprob = 0)
```

**Arguments**

|                         |                                                                             |
|-------------------------|-----------------------------------------------------------------------------|
| <code>x, q</code>       | integer vector of counts                                                    |
| <code>lambda</code>     | vector of (non-negative) means                                              |
| <code>zeroprob</code>   | zero-inflation probability between 0 and 1                                  |
| <code>log, log.p</code> | logical; return log-density if TRUE                                         |
| <code>lower.tail</code> | logical; if TRUE, probabilities are $P[X \leq x]$ , otherwise, $P[X > x]$ . |
| <code>n</code>          | number of random values to return.                                          |

**Details**

This implementation allows for automatic differentiation with RTMB.

**Value**

`dzipois` gives the probability mass function, `pzipois` gives the distribution function, and `rzipois` generates random deviates.

**Examples**

```
set.seed(123)
x <- rzipois(1, 0.5, 1)
d <- dzipois(x, 0.5, 1)
p <- pzipois(x, 0.5, 1)
```

---

zoibeta

*Zero- and one-inflated beta distribution*


---

**Description**

Density, distribution function, and random generation for the zero-one-inflated beta distribution.

**Usage**

```
dzoibeta(x, shape1, shape2, zeroprob = 0, oneprob = 0, log = FALSE)

pzoibeta(q, shape1, shape2, zeroprob = 0, oneprob = 0,
         lower.tail = TRUE, log.p = FALSE)

rzoibeta(n, shape1, shape2, zeroprob = 0, oneprob = 0)
```

**Arguments**

|                             |                                                                             |
|-----------------------------|-----------------------------------------------------------------------------|
| <code>x, q</code>           | vector of quantiles                                                         |
| <code>shape1, shape2</code> | non-negative shape parameters of the beta distribution                      |
| <code>zeroprob</code>       | zero-inflation probability between 0 and 1.                                 |
| <code>oneprob</code>        | zero-inflation probability between 0 and 1.                                 |
| <code>log, log.p</code>     | logical; if TRUE, probabilities/ densities $p$ are returned as $\log(p)$ .  |
| <code>lower.tail</code>     | logical; if TRUE, probabilities are $P[X \leq x]$ , otherwise, $P[X > x]$ . |
| <code>n</code>              | number of random values to return.                                          |

**Details**

This implementation allows for automatic differentiation with RTMB.

**Value**

`dzoibeta` gives the density, `pzoibeta` gives the distribution function, and `rzoibeta` generates random deviates.

**Examples**

```
set.seed(123)
x <- rzoibeta(1, 2, 2, 0.2, 0.3)
d <- dzoibeta(x, 2, 2, 0.2, 0.3)
p <- pzoibeta(x, 2, 2, 0.2, 0.3)
```

---

zoibeta2

*Reparameterised zero- and one-inflated beta distribution*


---

**Description**

Density, distribution function, and random generation for the zero-one-inflated beta distribution reparameterised in terms of mean and concentration.

**Usage**

```
dzoibeta2(x, mu, phi, zeroprob = 0, oneprob = 0, log = FALSE)

pzoibeta2(q, mu, phi, zeroprob = 0, oneprob = 0,
  lower.tail = TRUE, log.p = FALSE)

rzoibeta2(n, mu, phi, zeroprob = 0, oneprob = 0)
```

**Arguments**

|                         |                                                                             |
|-------------------------|-----------------------------------------------------------------------------|
| <code>x, q</code>       | vector of quantiles                                                         |
| <code>mu</code>         | mean parameter, must be in the interval from 0 to 1.                        |
| <code>phi</code>        | concentration parameter, must be positive.                                  |
| <code>zeroprob</code>   | zero-inflation probability between 0 and 1.                                 |
| <code>oneprob</code>    | zero-inflation probability between 0 and 1.                                 |
| <code>log, log.p</code> | logical; if TRUE, probabilities/ densities $p$ are returned as $\log(p)$ .  |
| <code>lower.tail</code> | logical; if TRUE, probabilities are $P[X \leq x]$ , otherwise, $P[X > x]$ . |
| <code>n</code>          | number of random values to return.                                          |

**Details**

This implementation allows for automatic differentiation with RTMB.

**Value**

`dzoibeta2` gives the density, `pzoibeta2` gives the distribution function, and `rzoibeta2` generates random deviates.

**Examples**

```
set.seed(123)
x <- rzoibeta2(1, 0.6, 2, 0.2, 0.3)
d <- dzoibeta2(x, 0.6, 2, 0.2, 0.3)
p <- pzoibeta2(x, 0.6, 2, 0.2, 0.3)
```

---

ztbinom

---

*Zero-truncated Binomial distribution*


---

**Description**

Probability mass function, distribution function, and random generation for the zero-truncated Binomial distribution.

**Usage**

```
dzrtbinom(x, size, prob, log = FALSE)

pzrtbinom(q, size, prob, lower.tail = TRUE, log.p = FALSE)

rzrtbinom(n, size, prob)
```

**Arguments**

|                         |                                                                             |
|-------------------------|-----------------------------------------------------------------------------|
| <code>x, q</code>       | integer vector of counts                                                    |
| <code>size</code>       | number of trials                                                            |
| <code>prob</code>       | success probability in each trial                                           |
| <code>log, log.p</code> | logical; return log-density if TRUE                                         |
| <code>lower.tail</code> | logical; if TRUE, probabilities are $P[X \leq x]$ , otherwise, $P[X > x]$ . |
| <code>n</code>          | number of random values to return.                                          |

**Details**

This implementation allows for automatic differentiation with RTMB.

By definition, this distribution only has support on the positive integers (1, ..., size). Any zero-truncated distribution is defined as

$$P(X = x | X > 0) = P(X = x) / (1 - P(X = 0)),$$

where  $P(X = x)$  is the probability mass function of the corresponding untruncated distribution.

**Value**

`dztnbinom` gives the probability mass function, `pztnbinom` gives the distribution function, and `rztnbinom` generates random deviates.

**Examples**

```
set.seed(123)
x <- rztnbinom(1, size = 10, prob = 0.3)
d <- dztnbinom(x, size = 10, prob = 0.3)
p <- pztnbinom(x, size = 10, prob = 0.3)
```

---

ztnbinom

*Zero-truncated Negative Binomial distribution*


---

**Description**

Probability mass function, distribution function, and random generation for the zero-truncated Negative Binomial distribution.

**Usage**

```
dztnbinom(x, size, prob, log = FALSE)

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

rztnbinom(n, size, prob)
```

**Arguments**

|                         |                                                                                                                                                                         |
|-------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>x, q</code>       | integer vector of counts                                                                                                                                                |
| <code>size</code>       | target for number of successful trials, or dispersion parameter (the shape parameter of the gamma mixing distribution). Must be strictly positive, need not be integer. |
| <code>prob</code>       | probability of success in each trial. $0 < \text{prob} \leq 1$ .                                                                                                        |
| <code>log, log.p</code> | logical; return log-density if TRUE                                                                                                                                     |
| <code>lower.tail</code> | logical; if TRUE, probabilities are $P[X \leq x]$ , otherwise, $P[X > x]$ .                                                                                             |
| <code>n</code>          | number of random values to return.                                                                                                                                      |

**Details**

This implementation allows for automatic differentiation with RTMB.

By definition, this distribution only has support on the positive integers (1, 2, ...). Any zero-truncated distribution is defined as

$$P(X = x | X > 0) = P(X = x) / (1 - P(X = 0)),$$

where  $P(X = x)$  is the probability mass function of the corresponding untruncated distribution.

**Value**

`dztnbinom` gives the probability mass function, `pztnbinom` gives the distribution function, and `rztnbinom` generates random deviates.

**Examples**

```
set.seed(123)
x <- rztnbinom(1, size = 2, prob = 0.5)
d <- dztnbinom(x, size = 2, prob = 0.5)
p <- pztnbinom(x, size = 2, prob = 0.5)
```

---

ztnbinom2

*Reparameterised zero-truncated negative binomial distribution*


---

**Description**

Probability mass function, distribution function, quantile function, and random generation for the zero-truncated negative binomial distribution reparameterised in terms of mean and size.

**Usage**

```
dztnbinom2(x, mu, size, log = FALSE)

pztnbinom2(q, mu, size, lower.tail = TRUE, log.p = FALSE)

rztnbinom2(n, mu, size)
```

**Arguments**

|                         |                                                                             |
|-------------------------|-----------------------------------------------------------------------------|
| <code>x, q</code>       | integer vector of counts                                                    |
| <code>mu</code>         | mean parameter, must be positive                                            |
| <code>size</code>       | size/dispersion parameter, must be positive                                 |
| <code>log, log.p</code> | logical; return log-density if TRUE                                         |
| <code>lower.tail</code> | logical; if TRUE, probabilities are $P[X \leq x]$ , otherwise, $P[X > x]$ . |
| <code>n</code>          | number of random values to return.                                          |

**Details**

This implementation allows for automatic differentiation with RTMB.

By definition, this distribution only has support on the positive integers (1, 2, ...). Any zero-truncated distribution is defined as

$$P(X = x | X > 0) = P(X = x) / (1 - P(X = 0)),$$

where  $P(X = x)$  is the probability mass function of the corresponding untruncated distribution.

**Value**

`dztnbinom2` gives the probability mass function, `pztnbinom2` gives the distribution function, and `rztnbinom2` generates random deviates.

**Examples**

```
set.seed(123)
x <- rztnbinom2(1, mu = 2, size = 1)
d <- dztnbinom2(x, mu = 2, size = 1)
p <- pztnbinom2(x, mu = 2, size = 1)
```

---

ztpois

---

*Zero-truncated Poisson distribution*


---

**Description**

Probability mass function, distribution function, and random generation for the zero-truncated Poisson distribution.

**Usage**

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

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

rztpois(n, lambda)
```

**Arguments**

|                         |                                                                             |
|-------------------------|-----------------------------------------------------------------------------|
| <code>x, q</code>       | integer vector of counts                                                    |
| <code>lambda</code>     | vector of (non-negative) means                                              |
| <code>log, log.p</code> | logical; return log-density if TRUE                                         |
| <code>lower.tail</code> | logical; if TRUE, probabilities are $P[X \leq x]$ , otherwise, $P[X > x]$ . |
| <code>n</code>          | number of random values to return.                                          |

**Details**

This implementation allows for automatic differentiation with RTMB.

By definition, this distribution only has support on the positive integers (1, 2, ...). Any zero-truncated distribution is defined as

$$P(X = x | X > 0) = P(X = x) / (1 - P(X = 0)),$$

where  $P(X = x)$  is the probability mass function of the corresponding untruncated distribution.

**Value**

`dztpois` gives the probability mass function, `pztpois` gives the distribution function, and `rztpois` generates random deviates.

**Examples**

```
set.seed(123)
x <- rztpois(1, 0.5)
d <- dztpois(x, 0.5)
p <- pztpois(x, 0.5)
```

# Index

abs\_smooth, [3](#), [27](#)

BCCG, [4](#)  
bccg, [4](#)  
BCPE, [5](#)  
bcpe, [5](#)  
BCT, [6](#)  
bct, [6](#)  
beta2, [7](#)  
betabinom, [8](#)

Cclayton, [17](#)  
Cclayton (cclayton), [9](#)  
cclayton, [9](#), [15](#)  
cclayton(), [10](#), [11](#), [13](#)  
Cfrank, [17](#)  
Cfrank (cfrank), [10](#)  
cfrank, [10](#), [15](#)  
cfrank(), [9](#), [11](#), [13](#)  
cgaussian, [11](#), [15](#)  
cgaussian(), [9](#), [10](#), [13](#)  
cgmrfr, [12](#), [19](#)  
cgmrfr(), [14](#)  
Cgumbel, [17](#)  
Cgumbel (cgumbel), [13](#)  
cgumbel, [13](#), [15](#)  
cgumbel(), [9–11](#)  
cmvgauss, [14](#), [19](#)  
cmvgauss(), [12](#)

dbccg (bccg), [4](#)  
dbcpe (bcpe), [5](#)  
dbct (bct), [6](#)  
dbeta (beta2), [7](#)  
dbeta2 (beta2), [7](#)  
dbetabinom (betabinom), [8](#)  
dcopula, [9–11](#), [13](#), [15](#)  
dcopula(), [17](#), [20](#)  
ddcopula, [16](#)  
ddcopula(), [15](#), [20](#)

ddirichlet (dirichlet), [17](#)  
ddirmult (dirmult), [18](#)  
dexgauss (exgauss), [21](#)  
dexp, [34](#)  
dfoldnorm (foldnorm), [22](#)  
dgamma2 (gamma2), [23](#)  
dgenpois (genpois), [24](#)  
dgumbel (gumbel), [25](#)  
dinvgauss (invgauss), [26](#)  
dirichlet, [17](#)  
dirmult, [18](#)  
dlaplace (laplace), [27](#)  
dmvcpula, [14](#), [19](#)  
dmvcpula(), [15](#), [17](#)  
dmvt (mvt), [28](#)  
dnbinom2 (nbinom2), [29](#)  
doibeta (oibeta), [30](#)  
doibeta2 (oibeta2), [31](#)  
dpareto (pareto), [32](#)  
dpgweibull (pgweibull), [33](#)  
dpowerexp (powerexp), [34](#)  
dpowerexp2 (powerexp), [34](#)  
dskewnorm (skewnorm), [36](#)  
dskewnorm2 (skewnorm2), [37](#)  
dskewt (skewt), [38](#)  
dskewt2 (skewt2), [39](#)  
dsn, [36](#)  
dst, [39](#), [40](#)  
dt2 (t2), [41](#)  
dtruncnorm (truncnorm), [42](#)  
dtrunct (trunct), [43](#)  
dtrunct2 (trunct2), [44](#)  
dvm (vm), [45](#)  
dvmf (vmf), [46](#)  
dvmf2 (vmf2), [47](#)  
dweibull, [34](#)  
dwishart (wishart), [48](#)  
dwrpcauchy (wrpcauchy), [48](#)  
dzibeta (zibeta), [50](#)

- dzibeta2 (zibeta2), 51
- dzibinom (zibinom), 52
- dzigamma (zigamma), 53
- dzigamma2 (zigamma2), 54
- dziinvgauss (ziinvgauss), 55
- dzilnorm (zilnorm), 56
- dzinbinom (zinbinom), 57
- dzinbinom2 (zinbinom2), 58
- dzipois (zipois), 59
- dzoibeta (zoibeta), 60
- dzoibeta2 (zoibeta2), 61
- dztbinom (ztbinom), 62
- dztbinom2 (ztbinom2), 63
- dztbinom2 (ztbinom2), 64
- dztipois (ztipois), 65
- erf, 20
- erfc (erf), 20
- exGAUS, 21
- exgauss, 21
- foldnorm, 22
- gamma2, 23
- genpois, 24
- gumbel, 25
- hpgweibull (pgweibull), 33
- invgauss, 26
- laplace, 27
- mvt, 28
- nbinom2, 29
- oibeta, 30
- oibeta2, 31
- PARETO, 32
- pareto, 32
- pbccg (bccg), 4
- pbcpe (bcpe), 5
- pbct (bct), 6
- pbeta2 (beta2), 7
- PE, 35
- pexgauss (exgauss), 21
- pfoldnorm (foldnorm), 22
- pgamma2 (gamma2), 23
- pgenpois (genpois), 24
- pgumbel (gumbel), 25
- pgweibull, 33
- pinvgauss (invgauss), 26
- plaplace (laplace), 27
- plnorm (zilnorm), 56
- pnbinom (nbinom2), 29
- pnbinom2 (nbinom2), 29
- poibeta (oibeta), 30
- poibeta2 (oibeta2), 31
- powerexp, 34
- ppareto (pareto), 32
- ppgweibull (pgweibull), 33
- ppowerexp (powerexp), 34
- ppowerexp2 (powerexp), 34
- pskewnorm (skewnorm), 36
- pskewnorm2 (skewnorm2), 37
- pskewt (skewt), 38
- pskewt2 (skewt2), 39
- pst, 39, 40
- pt (t2), 41
- pt2 (t2), 41
- ptruncnorm (truncnorm), 42
- ptrunct (trunct), 43
- ptrunct2 (trunct2), 44
- pvm (vm), 45
- pzibeta (zibeta), 50
- pzibeta2 (zibeta2), 51
- pzibinom (zibinom), 52
- pzigamma (zigamma), 53
- pzigamma2 (zigamma2), 54
- pziinvgauss (ziinvgauss), 55
- pzilnorm (zilnorm), 56
- pzinbinom (zinbinom), 57
- pzinbinom2 (zinbinom2), 58
- pzipois (zipois), 59
- pzoibeta (zoibeta), 60
- pzoibeta2 (zoibeta2), 61
- pztbinom (ztbinom), 62
- pztbinom2 (ztbinom2), 63
- pztbinom2 (ztbinom2), 64
- pztipois (ztipois), 65
- qbccg (bccg), 4
- qbcpe (bcpe), 5
- qbct (bct), 6
- qbeta2 (beta2), 7
- qexgauss (exgauss), 21
- qgamma2 (gamma2), 23

- qgenpois (genpois), 24
- qgumbel (gumbel), 25
- qinvgauss (invgauss), 26
- qlaplace (laplace), 27
- qnbinom2 (nbinom2), 29
- qpareto (pareto), 32
- qpgweibull (pgweibull), 33
- qpowerexp (powerexp), 34
- qpowerexp2 (powerexp), 34
- qskewnorm (skewnorm), 36
- qskewnorm2 (skewnorm2), 37
- qskewt (skewt), 38
- qskewt2 (skewt2), 39
- qt2 (t2), 41
- qtruncnorm (truncnorm), 42
- qtrunct (trunct), 43
- qtrunct2 (trunct2), 44
  
- rbccg (bccg), 4
- rbcpe (bcpe), 5
- rbct (bct), 6
- rbeta2 (beta2), 7
- rbetabinom (betabinom), 8
- rdirichlet (dirichlet), 17
- rdirmult (dirmult), 18
- rexgauss (exgauss), 21
- rfoldnorm (foldnorm), 22
- rgamma2 (gamma2), 23
- rgenpois (genpois), 24
- rgumbel (gumbel), 25
- rinvgauss (invgauss), 26
- rlaplace (laplace), 27
- rmvt (mvt), 28
- rnbinom2 (nbinom2), 29
- roibeta (oibeta), 30
- roibeta2 (oibeta2), 31
- rpareto (pareto), 32
- rpgweibull (pgweibull), 33
- rpowerexp (powerexp), 34
- rpowerexp2 (powerexp), 34
- rskewnorm (skewnorm), 36
- rskewnorm2 (skewnorm2), 37
- rskewt (skewt), 38
- rskewt2 (skewt2), 39
- rt2 (t2), 41
- rtruncnorm (truncnorm), 42
- rtrunct (trunct), 43
- rtrunct2 (trunct2), 44
- rvm (vm), 45
- rvmf (vmf), 46
- rvmf2 (vmf2), 47
- rwishart (wishart), 48
- rwrpcauchy (wrpcauchy), 48
- rzibeta (zibeta), 50
- rzibeta2 (zibeta2), 51
- rzibinom (zibinom), 52
- zgamma (zigamma), 53
- zgamma2 (zigamma2), 54
- ziinvgauss (ziinvgauss), 55
- zlnorm (zilnorm), 56
- zlnbinom (zinbinom), 57
- zlnbinom2 (zinbinom2), 58
- zipois (zipois), 59
- zoibeta (zoibeta), 60
- zoibeta2 (zoibeta2), 61
- zrtbinom (ztbinom), 62
- zrtlnbinom (ztnbinom), 63
- zrtlnbinom2 (ztnbinom2), 64
- zrtpois (ztpois), 65
  
- skewnorm, 36, 38, 39, 41
- skewnorm2, 37, 37, 39, 41
- skewt, 37, 38, 38, 41
- skewt2, 37–39, 39
- spgweibull (pgweibull), 33
- ST2, 39, 40
  
- t2, 41
- truncnorm, 42
- trunct, 43
- trunct2, 44
  
- vm, 45
- vmf, 46
- vmf2, 47
  
- wishart, 48
- wrpcauchy, 48
  
- zero\_inflate, 49
- zibeta, 50
- zibeta2, 51
- zibinom, 52
- zgamma, 53
- zgamma2, 54
- ziinvgauss, 55
- zlnorm, 56
- zinbinom, 57

zinbinom2, [58](#)  
zipois, [59](#)  
zoibeta, [60](#)  
zoibeta2, [61](#)  
ztbinom, [62](#)  
ztnbinom, [63](#)  
ztnbinom2, [64](#)  
ztpois, [65](#)