Package 'RTMBdist'

October 7, 2025

Type Package

Title Distributions Compatible with Automatic Differentiation by 'RTMB'

Version 0.1.0

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>.

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 7.3.2

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>

Repository CRAN

Date/Publication 2025-10-07 18:30:15 UTC

Contents

`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` *Beta-binomial distribution*

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 >= `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 constructor*

Description

Returns a function that computes the log density of the bivariate Clayton copula, intended to be used with `dcopula`.

Usage

```
cclayton(theta)
```

Arguments

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

Details

The Clayton copula density is

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

Value

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

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)
```

---

cfrank *Frank copula constructor*

Description

Returns a function computing the log density of the bivariate Frank copula, intended to be used with dcopula.

Usage

```
cfrank(theta)
```

Arguments

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

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

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

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 *Gaussian copula constructor*

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)
```

`cgumbel` *Gumbel copula constructor*

Description

Returns a function that computes the log density of the bivariate Gumbel copula, intended to be used with `dcopula`.

Usage

```
cgumbel(theta)
```

Arguments

`theta`	Dependence parameter ($\theta >= 1$).

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 log copula density.

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)
```

`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(0), 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.

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)
dcopula(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)
dcopula(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)
dcopula(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)
dcopula(d1, d2, p1, p2, copula = cfrank(2), log = TRUE)
```

`dirichlet` *Dirichlet distribution*

Description

Density and 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 and random generation for the Dirichlet-multinomial distribution.

Usage

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

rdirmult(n, size, alpha)
```

Arguments

`x`	vector or matrix of non-negative counts, where rows are observations and columns are categories.
`size`	vector of total counts for each observation. Needs to match the row sums of `x`.
`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 `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)
```

---

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 $P[X \le x]$, 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 μ, σ, and λ.

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 $P[X \le x]$, 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 $P[X \le x]$, 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 λ and variance $\lambda(1 + \phi\lambda)^2$. For $\phi = 0$ it reduces to the Poisson distribution, however ϕ 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 $P[X \le x]$, 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 $P[X \le x]$, otherwise, $P[X > x]$.
`p`	vector of probabilities
`...`	additional parameter passed to `statmod::qinvgauss` 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, 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.
`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 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` ≤ 1 the mean is undefined, and for `df` ≤ 2 the covariance is infinite. For `df` > 2, the covariance is `df/(df-2) * 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 $P[X \le x]$, otherwise, $P[X > x]$.
`p`	vector of probabilities
`n`	number of random values to return.
`prob`	probability of success in each trial. 0 < prob <= 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 $P[X \le x]$, 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)
```

---

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.

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 `sn` package functions for `pskewnorm` and `qskewnorm`.
`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.

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 = 1000, log = FALSE)

pskewt(q, mu = 0, sigma = 1, skew = 0, df = 1000,
       method = 0, lower.tail = TRUE, log.p = FALSE)

qskewt(p, mu = 0, sigma = 1, skew = 0, df = 1000,
       tol = 1e-8, method = 0)

rskewt(n, mu = 0, sigma = 1, skew = 0, df = 1000)
```

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 `pst` 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 numerial 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.

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)
```

`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 *Truncated normal distribution*

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

`x, q`	vector of quantiles
`mean`	mean parameter, must be positive.
`sd`	standard deviation 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 `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 *Truncated t distribution*

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

`x, q`	vector of quantiles
`df`	degrees of freedom 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 $P[X \le x]$, otherwise $P[X > x]$.
`p`	vector of probabilities
`n`	number of random values to return.

Details

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

Value

`dtrunct` gives the density, `ptrunct` gives the distribution function, `qtrunct` gives the quantile function, and `rtrunct` generates random deviates.

Examples

```
x <- rtrunct(1, df = 5, min = -1, max = 5)
d <- dtrunct(x, df = 5, min = -1, max = 5)
p <- ptrunct(x, df = 5, min = -1, max = 5)
q <- qtrunct(p, df = 5, min = -1, max = 5)
```

trunct2 *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

```
dtrunct2(x, df, mu = 0, sigma = 1, min = -Inf, max = Inf, log = FALSE)

ptrunct2(q, df, mu = 0, sigma = 1, min = -Inf, max = Inf,
         lower.tail = TRUE, log.p = FALSE)

qtrunct2(p, df, mu = 0, sigma = 1, min = -Inf, max = Inf,
         lower.tail = TRUE, log.p = FALSE)

rtrunct2(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 `dtrunct2` allows for automatic differentiation with `RTMB`.

Value

`dtrunct2` gives the density, `ptrunct2` gives the distribution function, `qtrunct2` gives the quantile function, and `rtrunct2` generates random deviates.

Examples

```
x <- rtrunct2(1, df = 5, mu = 2, sigma = 3, min = -1, max = 5)
d <- dtrunct2(x, df = 5, mu = 2, sigma = 3, min = -1, max = 5)
p <- ptrunct2(x, df = 5, mu = 2, sigma = 3, min = -1, max = 5)
q <- qtrunct2(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 μ is a unit vector and κ 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)
```

wrpcauchy — *wrapped Cauchy distribution*

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 rwrappedcauchyimported 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` *Zero-inflated density constructer*

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 `zeroprob + (1-zeroprob) * dist(0, ...)`. Otherwise it will just be `zeroprob`. In standard cases, this will be determined automatically. For non-standard cases, set this to `TRUE` or `FALSE` depending on the type of `dist`. 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 δ_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

`x, q`	vector of quantiles
`shape1, shape2`	non-negative shape parameters of the beta distribution
`zeroprob`	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

`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

`x, q`	vector of quantiles
`mu`	mean parameter, must be in the interval from 0 to 1.
`phi`	concentration parameter, must be positive.
`zeroprob`	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` (default), probabilities are $P[X \le x]$, otherwise $P[X > x]$.
`n`	number of random values to return.
`p`	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

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

`x, q`	vector of quantiles
`shape`	positive shape parameter
`scale`	positive scale parameter
`zeroprob`	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 $P[X \le x]$, otherwise, $P[X > x]$.
`n`	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

`x, q`	vector of quantiles
`mean`	location parameter
`shape`	shape parameter, must be positive.
`zeroprob`	zero-probability, must be in $[0, 1]$.
`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 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

`x, q`	vector of quantiles
`meanlog, sdlog`	mean and standard deviation of the distribution on the log scale with default values of 0 and 1 respectively.
`zeroprob`	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

`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

`x, q`	vector of (non-negative integer) quantiles
`size`	size parameter, must be positive.
`prob`	mean parameter, must be positive.
`zeroprob`	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.
`p`	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

`x, q`	vector of (non-negative integer) quantiles
`mu`	mean parameter, must be positive.
`size`	size parameter, must be positive.
`zeroprob`	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 $P[X \le x]$, otherwise, $P[X > x]$.
`n`	number of random values to return.
`p`	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 *Zero-inflated Poisson distribution*

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

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

`x, q`	vector of quantiles
`shape1, shape2`	non-negative shape parameters of the beta distribution
`zeroprob`	zero-inflation probability between 0 and 1.
`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 $P[X \le x]$, otherwise, $P[X > x]$.
`n`	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

`x, q`	vector of quantiles
`mu`	mean parameter, must be in the interval from 0 to 1.
`phi`	concentration parameter, must be positive.
`zeroprob`	zero-inflation probability between 0 and 1.
`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 $P[X \le x]$, otherwise, $P[X > x]$.
`n`	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

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

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

rztbinom(n, size, prob)
```

Arguments

`x, q`	integer vector of counts
`size`	number of trials
`prob`	success probability in each trial
`log, log.p`	logical; return log-density if TRUE
`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`.

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

`dztbinom` gives the probability mass function, `pztbinom` gives the distribution function, and `rztbinom` generates random deviates.

Examples

```
set.seed(123)
x <- rztbinom(1, size = 10, prob = 0.3)
d <- dztbinom(x, size = 10, prob = 0.3)
p <- pztbinom(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

`x, q`	integer vector of counts
`size`	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.
`prob`	probability of success in each trial. 0 < prob <= 1.
`log, log.p`	logical; return log-density if TRUE
`lower.tail`	logical; if `TRUE`, probabilities are $P[X \le x]$, otherwise, $P[X > x]$.
`n`	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

`x, q`	integer vector of counts
`mu`	mean parameter, must be positive
`size`	size/dispersion parameter, must be positive
`log, log.p`	logical; return log-density if TRUE
`lower.tail`	logical; if `TRUE`, probabilities are $P[X \le x]$, otherwise, $P[X > x]$.
`n`	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

`x, q`	integer vector of counts
`lambda`	vector of (non-negative) means
`log, log.p`	logical; return log-density if TRUE
`lower.tail`	logical; if `TRUE`, probabilities are $P[X \le x]$, otherwise, $P[X > x]$.
`n`	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