Package 'RTMBdist'

September 6, 2026

Type Package

Title Distributions Compatible with Automatic Differentiation by 'RTMB'

Version 1.1.0

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 GPL-2 | GPL-3

Encoding UTF-8

Imports stats, Matrix, circular, sn, statmod, movMF

Depends R (>= 3.5.0), RTMB (>= 1.9.0)

RoxygenNote 7.3.3

Suggests knitr, rmarkdown, testthat (>= 3.0.0), TMB, gamlss.data, mvtnorm, LaMa (>= 2.1.2)

Config/testthat/edition 3

URL https://janolefi.github.io/RTMBdist/

VignetteBuilder knitr

NeedsCompilation no

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

Maintainer Jan-Ole Fischer <jan-ole.fischer@mailbox.org>

Repository CRAN

Date/Publication 2026-09-06 16:30:02 UTC

Contents

`abs_smooth` *Smooth approximation to the absolute value function*

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

dbccg and pbccg allow for automatic differentiation with RTMB. The parameterisation follows the BCCG family of the gamlss.dist package.

The density is

$$f(x;\mu,\sigma,\nu) = \frac{x^{\nu-1}}{\mu^{\nu}\sigma\sqrt{2\pi}} \exp\left(-\tfrac{z^2}{2}\right) \left[\Phi\left(\tfrac{1}{\sigma|\nu|}\right)\right]^{-1}, \quad x > 0,$$

where $z = [(x/\mu)^{\nu} - 1]/(\nu\sigma)$ for $\nu \neq 0$ and $z = \log(x/\mu)/\sigma$ for $\nu = 0$, and Φ is the standard normal CDF.

Value

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

References

Cole, T. J. and Green, P. J. (1992) Smoothing reference centile curves: the LMS method and penalized likelihood. Statistics in Medicine, 11, 1305-1319.

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/.

See Also

bct, bcpe, gengamma

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

dbcpe and pbcpe allow for automatic differentiation with RTMB. The parameterisation follows the BCPE family of the gamlss.dist package.

The density is

$$f(x;\mu,\sigma,\nu,\tau) = \frac{x^{\nu-1}}{\mu^{\nu}\sigma}\frac{f_T(z;\tau)}{F_T(1/(\sigma|\nu|);\tau)}, \quad x > 0,$$

where $z = [(x/\mu)^{\nu} - 1]/(\nu\sigma)$ for $\nu \neq 0$ and $z = \log(x/\mu)/\sigma$ for $\nu = 0$, and $f_T(\cdot;\tau)$ and $F_T(\cdot;\tau)$ are the PDF and CDF of the power exponential (PE) distribution with shape τ.

Value

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

References

Rigby, R. A. and Stasinopoulos, D. M. (2004) Smooth centile curves for skew and kurtotic data modelled using the Box-Cox Power Exponential distribution. Statistics in Medicine, 23, 3053-3076.

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/.

See Also

bccg, bct, powerexp

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

`dbct` and `pbct` allow for automatic differentiation with `RTMB`. The parameterisation follows the `BCT` family of the `gamlss.dist` package.

The density is

$$f(x;\mu,\sigma,\nu,\tau) = \frac{x^{\nu-1}}{\mu^{\nu}\sigma}\frac{f_t(z;\tau)}{F_t(1/(\sigma|\nu|);\tau)}, \quad x>0,$$

where $z = [(x/\mu)^{\nu}-1]/(\nu\sigma)$ for $\nu \neq 0$ and $z = \log(x/\mu)/\sigma$ for $\nu = 0$, and $f_t(\cdot;\tau)$ and $F_t(\cdot;\tau)$ are the PDF and CDF of Student's t distribution with τ degrees of freedom.

Value

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

References

Rigby, R. A. and Stasinopoulos, D. M. (2006) Using the Box-Cox t distribution in GAMLSS to model skewness and kurtosis. Statistical Modelling, 6(3), 209. doi:10.1191/1471082X06st122oa

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/.

See Also

bccg, bcpe, skewt

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)
```

---

bell *Bell distribution*

Description

Probability mass function, distribution function, quantile function, and random generation for the Bell distribution.

Usage

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

pbell(q, theta, lower.tail = TRUE, log.p = FALSE)

qbell(p, theta, lower.tail = TRUE, log.p = FALSE)

rbell(n, theta)
```

Arguments

x, q	integer vector of counts
theta	vector of positive Bell parameters
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 dbell and pbell allows for automatic differentiation with RTMB with respect to theta.

The Bell distribution (Castellares et al. 2018) is a one-parameter distribution for overdispersed counts with probability mass function

$$P(X = x;\, \theta) = \frac{e^{1-e^{\theta}}\, \theta^{x}\, B_x}{x!}, \quad x = 0, 1, 2, \ldots,$$

for $\theta > 0$, where B_x is the x-th Bell number, i.e. the number of ways a set of x elements can be partitioned into non-empty subsets.

Its mean and variance are

$$E(X) = \theta e^{\theta}, \qquad \mathrm{Var}(X) = \theta e^{\theta}(1 + \theta),$$

so the dispersion index is $\mathrm{Var}(X)/E(X) = 1 + \theta > 1$ and the distribution is always overdispersed. Note that, unlike the negative binomial or generalised Poisson distributions, the Bell distribution has no separate dispersion parameter: the amount of overdispersion is tied to the mean. As $\theta \to 0$ it approaches the Poisson distribution.

The distribution arises as a compound Poisson sum

$$X = \sum_{i=1}^{N} Y_i, \qquad N \sim \mathrm{Pois}(e^{\theta} - 1), \quad Y_i \sim \mathrm{ztPois}(\theta),$$

with the Y_i independent of N, and rbell uses exactly this representation. It is therefore infinitely divisible, and a member of the one-parameter exponential family with natural parameter $\log \theta$ and sufficient statistic x.

The Bell numbers grow faster than any exponential and overflow double precision at $x = 219$, so $\log B_x$ is used throughout rather than B_x. As `x` is data, $\log B_x$ is constant with respect to the parameters and is cached across calls.

Neither the distribution function nor the quantile function has a closed form; both are obtained by summing the probability mass function, with `qbell` choosing its summation range automatically from the mean and variance.

See `bell2` for the parameterisation by the mean.

Value

`dbell` gives the probability mass function, `pbell` gives the distribution function, `qbell` gives the quantile function, and `rbell` generates random deviates.

References

Castellares, F., Ferrari, S. L. P., and Lemonte, A. J. (2018). On the Bell distribution and its associated regression model for count data. *Applied Mathematical Modelling* 56, 172-185. doi:10.1016/j.apm.2017.12.014

Examples

```
set.seed(123)
x <- rbell(1, 1)
d <- dbell(x, 1)
p <- pbell(x, 1)
q <- qbell(p, 1)

# mean and variance
theta <- 0.8
xs <- 0:100
sum(xs * dbell(xs, theta)) # theta * exp(theta)
```

`bell2` *Reparameterised Bell distribution*

Description

Probability mass function, distribution function, quantile function, and random generation for the Bell distribution reparameterised in terms of its mean.

Usage

```
dbell2(x, mu, log = FALSE)

pbell2(q, mu, lower.tail = TRUE, log.p = FALSE)

qbell2(p, mu, lower.tail = TRUE, log.p = FALSE)

rbell2(n, mu)
```

Arguments

x, q	integer vector of counts
mu	vector of positive means
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 dbell2 and pbell2 allows for automatic differentiation with RTMB with respect to mu.

The Bell distribution has mean $\mu = \theta e^{\theta}$, which is a bijection from $\theta > 0$ to $\mu > 0$ and is inverted by the principal branch of the Lambert W function,

$$\theta = W(\mu).$$

Every positive mean therefore corresponds to exactly one θ. All four functions simply apply this transformation and hand over to their bell counterparts.

In this parameterisation the variance is $\mu(1 + W(\mu))$, so the distribution is always overdispersed relative to the Poisson distribution, but the degree of overdispersion is determined by the mean rather than by a free parameter.

lambertW is AD-compatible to arbitrary order, so mu may be a parameter of a model fitted by Laplace approximation.

Value

dbell2 gives the probability mass function, pbell2 gives the distribution function, qbell2 gives the quantile function, and rbell2 generates random deviates.

References

Castellares, F., Ferrari, S. L. P., and Lemonte, A. J. (2018). On the Bell distribution and its associated regression model for count data. *Applied Mathematical Modelling* 56, 172-185. doi:10.1016/j.apm.2017.12.014

See Also

bell for the natural parameterisation.

Examples

```
set.seed(123)
x <- rbell2(1, 3)
d <- dbell2(x, 3)
p <- pbell2(x, 3)
q <- qbell2(p, 3)

# the two parameterisations agree
all.equal(dbell2(0:5, 3), dbell(0:5, lambertW(3)))
```

`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.

`dbeta` and `dbeta2` compute the beta density. `dbeta2` reparameterises by mean μ and concentration ϕ:

$$f(x;\mu,\phi) = \frac{x^{\mu\phi-1}(1-x)^{(1-\mu)\phi-1}}{B(\mu\phi,\,(1-\mu)\phi)}, \quad x \in (0,1).$$

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.

$$P(X = k;\, n, a, b) = \binom{n}{k} \frac{B(k + a,\, n - k + b)}{B(a,\, b)}, \quad k = 0, 1, \ldots, n.$$

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)
```

`betaprime` *Beta prime distribution*

Description

Density, distribution function, quantile function, and random generation for the Beta prime distribution.

Usage

```
dbetaprime(x, shape1, shape2, log = FALSE)

pbetaprime(q, shape1, shape2, lower.tail = TRUE, log.p = FALSE)

qbetaprime(p, shape1, shape2, lower.tail = TRUE, log.p = FALSE)

rbetaprime(n, shape1, shape2)
```

Arguments

`x`, `q`	vector of quantiles
`shape1`, `shape2`	non-negative shape parameters of the corresponding Beta distribution
`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 allows for automatic differentiation with `RTMB`.

If $X \sim \text{Beta}(\alpha, \beta)$, then $\frac{X}{1-X} \sim \text{Betaprime}(\alpha, \beta)$

$$f(x;\, a, b) = \frac{x^{a-1}}{(1+x)^{a+b}\, B(a,b)}, \quad x > 0.$$

Value

`dbetaprime` gives the density, `pbetaprime` gives the distribution function, `qbetaprime` gives the quantile function, and `rbetaprime` generates random deviates.

Examples

```
x <- rbetaprime(1, 2, 1)
d <- dbetaprime(x, 2, 1)
p <- pbetaprime(x, 2, 1)
q <- qbetaprime(p, 2, 1)
```

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

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

Value

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

Cfrank(theta)
```

Arguments

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

Details

The Frank copula density is

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

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 *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)
```

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 correaltion 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 ($\theta >= 1$).

Details

The Gumbel copula density

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

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))
```

combinom *Conway-Maxwell-binomial distribution*

Description

Probability mass function, distribution function, quantile function, and random generation for the Conway-Maxwell-binomial (CMB) distribution.

Usage

```
dcombinom(x, size, prob, nu = 1, log = FALSE)

pcombinom(q, size, prob, nu = 1, lower.tail = TRUE, log.p = FALSE)

qcombinom(p, size, prob, nu = 1, lower.tail = TRUE, log.p = FALSE)

rcombinom(n, size, prob, nu = 1)
```

Arguments

x, q	integer vector of counts in $\{0, 1, \ldots,$size$\}$
size	vector of numbers of trials (non-negative integers)
prob	vector of success probabilities in $(0, 1)$
nu	vector of dispersion parameters; nu = 1 gives the binomial distribution, nu > 1 under-dispersion and nu < 1 over-dispersion. May be negative.
log, log.p	logical; if TRUE, probabilities/densities 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 dcombinom and pcombinom allows for automatic differentiation with RTMB, including differentiation with respect to x, such that one-step-ahead (OSA) residuals are supported.

The CMB distribution generalises the binomial distribution by an additional dispersion parameter ν in the same way that the Conway-Maxwell-Poisson distribution generalises the Poisson distribution. Its probability mass function is

$$P(X = x;\, n, p, \nu) = \frac{1}{Z(n, p, \nu)} \binom{n}{x}^{\nu} p^x (1-p)^{n-x}, \quad x = 0, 1, \ldots, n,$$

with normalising constant

$$Z(n, p, \nu) = \sum_{k=0}^{n} \binom{n}{k}^{\nu} p^k (1-p)^{n-k}.$$

For $\nu = 1$ this reduces to the binomial distribution. Values $\nu > 1$ give under-dispersion and $\nu < 1$ over-dispersion relative to a binomial distribution with the same mean. As the support is finite, Z converges for every real ν, so ν is not restricted to be positive; negative values yield strongly over-dispersed, U-shaped distributions.

The distribution arises as the sum of n exchangeable, possibly associated Bernoulli variables, where ν controls the association. Note that `prob` is *not* the mean divided by `size` unless $\nu = 1$; the mean has no closed form for general ν and must be obtained by summation over the support.

Value

`dcombinom` gives the probability mass function, `pcombinom` gives the distribution function, `qcombinom` gives the quantile function, and `rcombinom` generates random deviates.

References

Shmueli, G., Minka, T. P., Kadane, J. B., Borle, S., and Boatwright, P. (2005). A useful distribution for fitting discrete data: revival of the Conway-Maxwell-Poisson distribution. *Journal of the Royal Statistical Society: Series C* 54(1), 127-142.

Kadane, J. B. (2016). Sums of possibly associated Bernoulli variables: the Conway-Maxwell-binomial distribution. *Bayesian Analysis* 11(2), 403-420.

https://en.wikipedia.org/wiki/Conway-Maxwell-binomial_distribution

Examples

```
set.seed(123)
x <- rcombinom(1, size = 10, prob = 0.4, nu = 1.5)
d <- dcombinom(x, 10, 0.4, 1.5)
p <- pcombinom(x, 10, 0.4, 1.5)
q <- qcombinom(p, 10, 0.4, 1.5)

# nu = 1 recovers the binomial distribution
all.equal(dcombinom(0:10, 10, 0.3, 1), dbinom(0:10, 10, 0.3))
```

---

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.

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)
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)
```

`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, **not** 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 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.

$$f(\mathbf{x};\, \boldsymbol{\alpha}) = \frac{\Gamma(\sum_i \alpha_i)}{\prod_i \Gamma(\alpha_i)} \prod_i x_i^{\alpha_i - 1},$$

where $\mathbf{x}$ lies on the unit simplex and $\alpha_i > 0$.

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`.

$$P(\mathbf{x};\, \boldsymbol{\alpha}, n) = \frac{\Gamma(\alpha_0)\,\Gamma(n+1)}{\Gamma(n+\alpha_0)} \prod_i \frac{\Gamma(x_i+\alpha_i)}{\Gamma(\alpha_i)\,\Gamma(x_i+1)},$$

where $\alpha_0 = \sum_i \alpha_i$ and $n = \sum_i x_i$.

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 $\log c(u_1, ...u_d)$. 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

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

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

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

Available multivariate copula constructors are:

- cmvgauss (Multivariate Gaussian copula)
- cgmrf (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
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))
```

---

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` inverts `pexgauss` numerically, as the exponentially modified Gaussian has no closed-form quantile function. The parameterisation follows the `exGAUS` family of the `gamlss.dist` package, with the exponential rate $\lambda = 1/\nu$.

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.

The density is

$$f(x;\, \mu, \sigma, \lambda) = \lambda \exp\!\Big(\lambda\mu + \tfrac{\lambda^2\sigma^2}{2} - \lambda x\Big)\, \Phi\!\left(\frac{x - \mu - \lambda\sigma^2}{\sigma}\right),$$

where Φ is the standard normal CDF.

Value

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

References

Cousineau, D., Brown, S. and Heathcote, A. (2004) Fitting distributions using maximum likelihood: Methods and packages. Behavior Research Methods, Instruments, & Computers, 36, 742-756.

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/.

See Also

jsu, skewnorm

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`.

$$f(x;\,\mu,\sigma) = \frac{1}{\sigma\sqrt{2\pi}}\left[\exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right) + \exp\left(-\frac{(x+\mu)^2}{2\sigma^2}\right)\right], \quad x \geq 0.$$

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.

Reparameterises the gamma distribution via shape $= \mu^2/s^2$ and scale $= s^2/\mu$:

$$f(x;\, \mu, s) = \frac{x^{\mu^2/s^2-1}\exp(-xs^2/\mu^2)}{(s^2/\mu)^{\mu^2/s^2}\,\Gamma(\mu^2/s^2)}, \quad x > 0.$$

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)
```

---

gengamma *Generalised Gamma distribution (GG)*

Description

Density, distribution function, quantile function, and random generation for the generalised Gamma distribution.

Usage

```
dgengamma(x, mu = 1, sigma = 0.5, nu = 1, log = FALSE)

pgengamma(q, mu = 1, sigma = 0.5, nu = 1, lower.tail = TRUE, log.p = FALSE)

qgengamma(p, mu = 1, sigma = 0.5, nu = 1, lower.tail = TRUE, log.p = FALSE)

rgengamma(n, mu = 1, sigma = 0.5, 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 dgengamma, pgengamma, and qgengamma allows for automatic differentiation with RTMB.

For $\nu \neq 0$ the density is that of a generalised gamma: setting $z = (x/\mu)^\nu$,

$$f(x;\, \mu, \sigma, \nu) = \frac{|\nu|}{x\,(\sigma|\nu|)^{2/(\sigma\nu)^2}\,\Gamma(1/(\sigma\nu)^2)}\, z^{1/(\sigma\nu)^2}\, \exp\!\left(-\frac{z}{(\sigma|\nu|)^2}\right), \quad x > 0.$$

For $\nu = 0$ the distribution reduces to a log-normal with log-mean $\log \mu$ and log-standard-deviation σ.

Value

dgengamma gives the density, pgengamma gives the distribution function, qgengamma gives the quantile function, and rgengamma generates random deviates.

References

Lopatatzidis, A. and Green, P. J. (2000) Nonparametric quantile regression using the gamma distribution. Unpublished.

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/.

See Also

gamma2, bccg, invgauss

Examples

```
x <- rgengamma(5, mu = 4, sigma = 0.5, nu = 0.5)
d <- dgengamma(x, mu = 4, sigma = 0.5, nu = 0.5)
p <- pgengamma(x, mu = 4, sigma = 0.5, nu = 0.5)
q <- qgengamma(p, mu = 4, sigma = 0.5, nu = 0.5)
```

---

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 parameterisation follows the GPO family of the gamlss.dist package.

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.

$$P(X = x;\, \lambda, \phi) = \frac{\lambda\,(1+\phi x)^{x-1}\, e^{-\lambda(1+\phi x)/(1+\phi\lambda)}}{(1+\phi\lambda)^x\, x!}, \quad x = 0, 1, 2, \ldots$$

Value

dgenpois gives the probability mass function, pgenpois gives the distribution function, qgenpois gives the quantile function, and rgenpois 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/.

See Also

nbinom2, zipois, bell

Examples

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

`gompertz` *Gompertz distribution*

Description

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

Usage

```
dgompertz(x, eta = 1, b = 1, log = FALSE)

pgompertz(q, eta = 1, b = 1, lower.tail = TRUE, log.p = FALSE)

qgompertz(p, eta = 1, b = 1, lower.tail = TRUE, log.p = FALSE)

rgompertz(n, eta = 1, b = 1)
```

Arguments

`x`, `q`	vector of quantiles (non-negative)
`eta`	shape parameter, must be positive
`b`	rate parameter, must be positive
`log`, `log.p`	logical; if `TRUE`, probabilities/densities are returned as $\log(p)$.
`lower.tail`	logical; if `TRUE` (default), probabilities are $P[X \le x]$, otherwise $P[X > x]$.
`p`	vector of probabilities
`n`	number of random values to return

Details

The Gompertz distribution with shape $\eta > 0$ and rate $b > 0$ has density

$$f(x;\, \eta, b) = b\eta\, e^{bx} \exp\!\big(-\eta(e^{bx} - 1)\big), \quad x \ge 0,$$

with CDF $F(x) = 1 - \exp(-\eta(e^{bx} - 1))$ and quantile function $Q(p) = \log(1 - \log(1-p)/\eta) \,/\, b$.

Value

`dgompertz` gives the density, `pgompertz` gives the distribution function, `qgompertz` gives the quantile function, and `rgompertz` generates random deviates.

References

https://en.wikipedia.org/wiki/Gompertz_distribution

Examples

```
x <- rgompertz(1, eta = 1, b = 1)
d <- dgompertz(x, eta = 1, b = 1)
p <- pgompertz(x, eta = 1, b = 1)
q <- qgompertz(p, eta = 1, b = 1)
```

---

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.

$$f(x;\, \mu, \sigma) = \frac{1}{\sigma} \exp\bigl(-(z + e^{-z})\bigr), \quad z = \frac{x - \mu}{\sigma}.$$

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)
```

---

invchisq *Inverse Chi-squared distribution*

Description

Density, distribution function, quantile function, and random generation for the inverse Chi-squared distribution.

Usage

```
dinvchisq(x, df, scale = 1/df, log = FALSE)

pinvchisq(q, df, scale = 1/df, lower.tail = TRUE, log.p = FALSE)

qinvchisq(p, df, scale = 1/df, lower.tail = TRUE, log.p = FALSE)

rinvchisq(n, df, scale = 1/df)
```

Arguments

x, q	vector of quantiles, must be positive.
df	degrees of freedom ($\nu > 0$)
scale	optional positive scale parameter. Default value of 1/df corresponds to standard inverse Chi-squared
log, log.p	logical; if TRUE, probabilities/densities 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

If $X \sim$ Chisq(ν), then $1/X \sim$ invChisq(ν).

The inverse Chi-squared distribution with ν degrees of freedom has density

$$f(x) = \frac{(\nu s/2)^{\nu/2}}{\Gamma(\nu/2)} x^{-(\nu/2+1)} \exp\Big(-\frac{\nu s}{2x}\Big), \quad x > 0,$$

This implementation of `dinvchisq`, `pinvchisq`, and `qinvchisq` allows for automatic differentiation with `RTMB`.

Value

`dinvchisq` gives the density, `pinvchisq` gives the distribution function, `qinvchisq` gives the quantile function, and `rinvchisq` generates random deviates.

Examples

```
x <- rinvchisq(1, df = 5)
d <- dinvchisq(x, df = 5)
p <- pinvchisq(x, df = 5)
q <- qinvchisq(p, df = 5)
```

`invgamma` *Inverse Gamma distribution*

Description

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

Usage

```
dinvgamma(x, shape, rate, scale = 1/rate, log = FALSE)

pinvgamma(q, shape, rate, scale = 1/rate, lower.tail = TRUE, log.p = FALSE)

qinvgamma(p, shape, rate, scale = 1/rate, lower.tail = TRUE, log.p = FALSE)

rinvgamma(n, shape, rate, scale = 1/rate)
```

Arguments

`x`, `q`	vector of quantiles, must be positive.
`shape`, `rate`, `scale`	positive parameters of corresponding gamma distribution
`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 dinvgamma, pinvgamma, and qinvgamma allows for automatic differentiation with RTMB.

If $X \sim \Gamma(\alpha, \beta)$, then $1/X \sim$ InvGamma(α, β).

$$f(x;\, \alpha, s) = \frac{s^{\alpha}}{\Gamma(\alpha)}\, x^{-(\alpha+1)} \exp\Big(-\frac{s}{x}\Big), \quad x > 0,$$

where $s =$ scale $= 1/$rate.

Value

dinvgamma gives the density, pinvgamma gives the distribution function, qinvgamma gives the quantile function, and rinvgamma generates random deviates.

Examples

```
x <- rinvgamma(1, 1, 0.5)
d <- dinvgamma(x, 1, 0.5)
p <- pinvgamma(x, 1, 0.5)
q <- qinvgamma(p, 1, 0.5)
```

---

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 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.

$$f(x;\,\mu,\lambda) = \sqrt{\frac{\lambda}{2\pi x^3}}\exp\left(-\frac{\lambda(x-\mu)^2}{2\mu^2 x}\right), \quad x > 0.$$

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)
```

---

jsu *Johnson SU distribution (JSU)*

Description

Density, distribution function, quantile function, and random generation for the Johnson SU distribution, in the original and in the moment parameterisation.

Usage

```
djsu(x, mu = 0, sigma = 1, nu = 0, tau = 1, log = FALSE)

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

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

rjsu(n, mu = 0, sigma = 1, nu = 0, tau = 1)

djsu2(x, mu = 0, sigma = 1, nu = 0, tau = 1, log = FALSE)

pjsu2(q, mu = 0, sigma = 1, nu = 0, tau = 1, lower.tail = TRUE, log.p = FALSE)
```

```
qjsu2(p, mu = 0, sigma = 1, nu = 0, tau = 1, lower.tail = TRUE, log.p = FALSE)

rjsu2(n, mu = 0, sigma = 1, nu = 0, tau = 1)
```

Arguments

x, q	vector of quantiles
mu	location parameter for djsu; the mean for djsu2.
sigma	scale parameter for djsu; the standard deviation for djsu2. Must be positive.
nu	skewness parameter (real). Positive ν gives left skewness in djsu and right skewness in djsu2.
tau	kurtosis parameter, must be positive. Large τ approaches the normal distribution.
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

The Johnson SU distribution is a four-parameter continuous distribution on the whole real line, obtained by applying the transformation

$$Z = \nu + \tau \operatorname{asinh}\left(\frac{x-\mu}{\sigma}\right) \sim N(0,1)$$

to a standard normal variable. It covers a wide range of skewness and kurtosis combinations and is a common alternative to the Box-Cox families for data that are not restricted to be positive.

djsu uses the original parameterisation, in which μ and σ are a location and a scale parameter, ν controls skewness and τ controls kurtosis. The density is

$$f(x;\mu,\sigma,\nu,\tau) = \frac{\tau}{\sigma\sqrt{2\pi}} \frac{1}{\sqrt{z^2+1}} \exp\left(-\frac{r^2}{2}\right),$$

where $z = (x-\mu)/\sigma$ and $r = \nu + \tau \operatorname{asinh}(z)$. Here μ is *not* the mean and σ is *not* the standard deviation.

djsu2 uses the moment parameterisation, in which μ **is** the mean and σ **is** the standard deviation of the distribution, for any admissible ν and τ. This is usually the more convenient parameterisation for regression modelling, because the location and scale parameters keep their interpretation as ν and τ change. Writing $\omega = -\nu/\tau$ and $w = \exp(\tau^{-2})$, it is obtained from the original parameterisation by the reparameterisation

$$c = \left[\tfrac{1}{2}(w-1)\,(w\cosh(2\omega)+1)\right]^{-1/2}, \qquad \sigma^* = c\,\sigma, \qquad \mu^* = \mu + c\,\sigma\sqrt{w}\,\sinh(\omega),$$

so that djsu2(x, mu, sigma, nu, tau) equals djsu(x, μ^*, σ^*, -nu, tau).

These correspond to the JSUo and JSU families of the gamlss.dist package respectively; see Chapter 18 of Rigby et al. (2019). Note that the sign convention for ν differs between the two parameterisations, exactly as it does in gamlss.dist.

All four d and p functions are compatible with automatic differentiation by RTMB, so both simulation and one-step-ahead residuals are supported.

Value

djsu gives the density, pjsu gives the distribution function, qjsu gives the quantile function, and rjsu generates random deviates. djsu2, pjsu2, qjsu2 and rjsu2 are the corresponding functions for the moment parameterisation.

References

Johnson, N. L. (1954). Systems of frequency curves derived from the first law of Laplace. Trabajos de Estadistica, 5, 283-291.

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/.

See Also

bcpe, bct, skewt, skewnorm

Examples

```
set.seed(123)
# original parameterisation
x <- rjsu(5, mu = 0, sigma = 1, nu = -1, tau = 2)
d <- djsu(x, mu = 0, sigma = 1, nu = -1, tau = 2)
p <- pjsu(x, mu = 0, sigma = 1, nu = -1, tau = 2)
q <- qjsu(p, mu = 0, sigma = 1, nu = -1, tau = 2)

# moment parameterisation: mu is the mean, sigma the standard deviation
y <- rjsu2(1000, mu = 3, sigma = 2, nu = 1, tau = 3)
c(mean = mean(y), sd = sd(y))
```

---

kumar *Kumaraswamy distribution*

Description

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

Usage

```
dkumar(x, a, b, log = FALSE)

pkumar(q, a, b, lower.tail = TRUE, log.p = FALSE)

qkumar(p, a, b, lower.tail = TRUE, log.p = FALSE)

rkumar(n, a, b)
```

Arguments

x, q	vector of quantiles in $(0, 1)$
a, b	positive shape parameters
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

$$f(x;\, a, b) = ab\, x^{a-1}(1 - x^a)^{b-1}, \quad x \in (0, 1).$$

Value

dkumar gives the density, pkumar gives the distribution function, qkumar gives the quantile function, and rkumar generates random deviates.

Examples

```
x <- rkumar(1, a = 1, b = 2)
d <- dkumar(x, a = 1, b = 2)
p <- pkumar(x, a = 1, b = 2)
q <- qkumar(p, a = 1, b = 2)
```

---

lambertW *Lambert W function (principal branch)*

Description

Solves $W(x)e^{W(x)} = x$ for the principal branch W_0.

Usage

```
lambertW(x)
```

Arguments

x	vector of evaluation points, $x \geq -1/e$.

Details

This implementation allows for automatic differentiation with RTMB.

The value is obtained by Halley iteration, which converges to machine precision in a handful of steps over the whole domain. For AD, the function is registered as an atomic operation via ADjoint with the analytic derivative

$$W'(x) = \frac{1}{e^{W(x)}(1 + W(x))},$$

expressed through the returned value rather than through x. Written this way the derivative is itself an AD-able expression, so derivatives of every order are available; in particular the third-order derivatives that the gradient of a Laplace approximation requires.

The principal branch is defined for $x \geq -1/e$, with $W_0(-1/e) = -1$ and $W_0(x) \geq -1$ throughout. Values below $-1/e$ return NaN with a warning. The derivative is infinite at the branch point $x = -1/e$.

Value

The principal branch of the Lambert W function evaluated at x.

Examples

```
lambertW(exp(1)) # 1
lambertW(0) # 0
x <- c(0.5, 1, 10, 1000)
lambertW(x) * exp(lambertW(x)) - x # ~ 0
```

---

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 abs_smooth 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.

$$f(x;\, \mu, b) = \frac{1}{2b} \exp\left(-\frac{|x-\mu|}{b}\right).$$

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)
```

---

laplace_check *Assess the accuracy of the Laplace approximation*

Description

Diagnoses how well the Laplace approximation used by TMB/RTMB approximates the marginal likelihood, via the importance-sampling check of evaluating the integrand relative to the fitted Gaussian.

Usage

```
laplace_check(obj, nSamples = 1000)

## S3 method for class 'laplace_check'
print(x, digits = 4, ...)
```

Arguments

obj	A fitted TMB/RTMB object (as returned by MakeADFun) with random effects. The model is assumed to have been optimised, so that objenvlast.par.best holds the maximum-likelihood parameters and the corresponding mode of the latent variables.
nSamples	Number of Monte Carlo samples drawn from the Gaussian approximation. Default 1000.
x	An object of class "laplace_check".
digits	Number of decimal places for the diagnostic values.
...	Unused.

Details

For fixed parameters $\hat{\theta}$, the marginal likelihood is $Z = \int f(y \mid x) f(x)\, dx$. Writing the importance weight

$$\tilde{\ell}(x) = f(y \mid x) f(x) / N(x \mid \hat{x}, H^{-1}),$$

the Laplace approximation equals $\tilde{\ell}(\hat{x})$, while $E_{x \sim N(\hat{x}, H^{-1})}[\tilde{\ell}(x)] = Z$ exactly. The two coincide if and only if the joint negative log-likelihood is quadratic in the latent variables. The spread of the log-weights therefore measures the departure from this ideal: a standard deviation near zero indicates a near-exact approximation.

Value

An object of class "laplace_check": a list with the log-weights logw, their standard deviation sd_logw, the Laplace and importance-sampling estimates of the log marginal likelihood (lZ_laplace, lZ_is), their difference log_bias, and the relative effective sample size ess_ratio.

Examples

```
# Chicken weight example; taken from RTMB Introduction vignette
data(ChickWeight)

parameters <- list(
  mua=0,          ## Mean slope
  sda=1,          ## Std of slopes
  mub=0,          ## Mean intercept
  sdb=1,          ## Std of intercepts
  sdeps=1,        ## Residual Std
  a=rep(0, 50),   ## Random slope by chick
  b=rep(0, 50)    ## Random intercept by chick
)

jnll <- function(parms) {
  getAll(ChickWeight, parms, warn=FALSE)
  ## Optional (enables extra RTMB features)
  weight <- OBS(weight)
  ## Initialize joint negative log likelihood
  nll <- 0
  ## Random slopes
```

```
  nll <- nll - sum(dnorm(a, mean=mua, sd=sda, log=TRUE))
  ## Random intercepts
  nll <- nll - sum(dnorm(b, mean=mub, sd=sdb, log=TRUE))
  ## Data
  predWeight <- a[Chick] * Time + b[Chick]
  nll <- nll - sum(dnorm(weight, predWeight, sd=sdeps, log=TRUE))
  ## Get predicted weight uncertainties
  ADREPORT(predWeight)
  ## Return
  nll
}

obj <- MakeADFun(jnll, parameters, random=c("a", "b"), silent = TRUE)
opt <- nlminb(obj$par, obj$fn, obj$gr)

chk <- laplace_check(obj)
chk
# Laplace approximation exact here: linear Gaussian-Gaussian example
```

`llogis` *Log-logistic distribution*

Description

Density, distribution function, quantile function, and random generation for the log-logistic distribution.

Usage

```
dllogis(x, alpha = 1, beta = 1, log = FALSE)

pllogis(q, alpha = 1, beta = 1, lower.tail = TRUE, log.p = FALSE)

qllogis(p, alpha = 1, beta = 1, lower.tail = TRUE, log.p = FALSE)

rllogis(n, alpha = 1, beta = 1)
```

Arguments

`x`, `q`	vector of quantiles ($x > 0$).
`alpha`	scale parameter ($\alpha > 0$); equal to the median.
`beta`	shape parameter ($\beta > 0$); controls tail heaviness.
`log`	logical; if `TRUE`, densities are returned on the log scale.
`lower.tail`	logical; if `TRUE` (default), probabilities are $P[X \leq x]$, otherwise $P[X > x]$.
`log.p`	logical; if `TRUE`, probabilities are returned on the log scale.
`p`	vector of probabilities.
`n`	number of random values to return.

Details

The log-logistic distribution has density

$$f(x;\,\alpha,\beta) = \frac{(\beta/\alpha)\,(x/\alpha)^{\beta-1}}{\big(1+(x/\alpha)^{\beta}\big)^{2}}, \quad x > 0,$$

where $\alpha > 0$ is the scale parameter and $\beta > 0$ is the shape parameter. The scale parameter equals the median. Larger β gives lighter tails; the distribution has a finite mean only when $\beta > 1$ and a finite variance only when $\beta > 2$.

The log-logistic arises naturally as the distribution of $X = e^{Y}$ where $Y \sim \mathrm{Logistic}(\log\alpha,\, 1/\beta)$, which yields the numerically convenient log-density

$$\log f(x) = \log\beta - \log x + \log f_{\mathrm{logistic}}\big(\beta\log(x/\alpha)\big).$$

The CDF is

$$F(x;\,\alpha,\beta) = \frac{1}{1+(x/\alpha)^{-\beta}},$$

and the quantile function is

$$Q(p;\,\alpha,\beta) = \alpha\left(\frac{p}{1-p}\right)^{1/\beta}.$$

`dllogis` allows for automatic differentiation with `RTMB`.

Value

`dllogis` gives the density, `pllogis` gives the distribution function, `qllogis` gives the quantile function, and `rllogis` generates random deviates.

Examples

```
x <- rllogis(5, alpha = 1, beta = 2)
dllogis(x, alpha = 1, beta = 2)
pllogis(c(0.5, 1, 2), alpha = 1, beta = 2)
qllogis(c(0.25, 0.5, 0.75), alpha = 1, beta = 2)
```

`mcreport` *Sample parameters from approximate Gaussian posterior distribution*

Description

Efficient Monte Carlo sampling of parameters (and `REPORT`ed quantities) from the approximate posterior of an `(R)TMB` model. See `sdreport` for details on posterior variance-covariance in random effects models.

Usage

```
mcreport(
  obj,
  nSamples = 1000,
  include_random_pars = TRUE,
  report = FALSE,
  Q = NULL,
  ...
)
```

Arguments

obj	Optimised RTMB object generated by MakeADFun
nSamples	Number of samples to draw
include_random_pars	Logical; Should random parameters be included in the output?
report	Logical; Should REPORTed quantities be sampled as well? Defaults to FALSE because this may be slow depending on your model.
Q	Optional precalculated sparse precision matrix returned by sdreport(..., getJointPrecision = TRUE)$jointPrecision. If not provided, computed internally using sdreport. Only used for models with random effects.
...	For internal use only

Details

Caution: This has nothing to do with Bayesian posterior sampling. It is simple Monte Carlo sampling from a Gaussian distribution with mean at the MLE and covariance given by the inverse of the Hessian (for fixed effects models) or the joint precision matrix (for random effects models). This is a common approach to get an approximate idea of parameter uncertainty around the MLE, but it relies on large-sample asymptotics.

Sampling random effects from their posterior as compared to calculating marginal standard deviations like sdreport is particularly useful for multidimensional random effects (e.g. for locations x and y) where pointwise confidence intervals (e.g. along a path) based on standard deviations are not possible.

Value

A list structured like the original parameter list used in the MakeADFun call (potentially including additional REPORTed quantities). Each entry is a list with nSamples entries.

Examples

```
set.seed(123)

### Example 1: Without random effects ###

# simulate data
x <- rgumbel(100, location = 5, scale = 2)
```

```
# negative log-likelihood function
nll <- function(par) {
  getAll(par)
  x <- OBS(x) # mark x as the response
  scale <- exp(log_scale)
  ADREPORT(scale); REPORT(scale)
  -sum(dgumbel(x, loc, scale, log = TRUE))
}

# RTMB AD object
par <- list(loc = 5, log_scale = log(2))
obj <- MakeADFun(nll, par, silent = TRUE)

# model fitting using AD gradient
opt <- nlminb(obj$par, obj$fn, obj$gr)

# sample from asymptotic distribution of the MLE
samples <- mcreport(obj, report = TRUE)
# parameters
mean(unlist(samples$loc)); sd(unlist(samples$loc))
# reported transformed parameters
mean(unlist(samples$scale)); sd(unlist(samples$scale))

# compare to delta method sd from sdreport
sdr <- sdreport(obj)
summary(sdr)



### Example 2: With random effects ###

# simulate random effects
id <- rep(1:10, each = 100) # 10 groups with 100 observations each
true_gamma <- rnorm(10, 0, 2) # random coefficients
true_probs <- plogis(1 + true_gamma) # linear predictor

# simulate Bernoulli data conditional on random coefficients
x <- rbinom(1000, 1, true_probs[id])

# negative log-likelihood function
nll <- function(par) {
  getAll(par)
  x <- OBS(x) # mark x as the response

  # random effect likelihood
  sd_gamma <- exp(log_sd_gamma)
  ADREPORT(sd_gamma); REPORT(sd_gamma)
  nll <- -sum(dnorm(gamma, 0, sd_gamma, log = TRUE))

  # data likelihood
  probs <- plogis(beta0 + gamma) # linear predictor
  ADREPORT(probs); REPORT(probs)
```

```
  nll <- nll - sum(dbinom(x, 1, probs[id], log = TRUE))
  return(nll)
}

# RTMB Laplace approximation
par <- list(beta0 = 1, log_sd_gamma = log(2), gamma = rep(0,10))
obj <- MakeADFun(nll, par, random = "gamma", silent = TRUE)

# model fitting using AD gradient
opt <- nlminb(obj$par, obj$fn, obj$gr)

# draw samples
samples <- mcreport(obj, report = TRUE)

# run sdreport
sdr <- sdreport(obj)

# standard deviation using delta method
as.list(sdr, "Std", report = TRUE)$probs

# standard deviation from samples
apply(simplify2array(samples$probs), 1, sd)
```

`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`.

$$f(\mathbf{x};\, \boldsymbol{\mu}, \Sigma, \nu) = \frac{\Gamma((\nu + d)/2)}{\Gamma(\nu/2)\,(\nu\pi)^{d/2}\,|\Sigma|^{1/2}} \left(1 + \frac{(\mathbf{x} - \boldsymbol{\mu})^\top \Sigma^{-1} (\mathbf{x} - \boldsymbol{\mu})}{\nu}\right)^{-(\nu+d)/2},$$

where d is the dimension.

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)
```

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.

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.

Reparameterises the negative binomial by mean μ via $p = r/(r+\mu)$:

$$P(X = k;\, \mu, r) = \binom{k+r-1}{k} \left(\frac{r}{r+\mu}\right)^r \left(\frac{\mu}{r+\mu}\right)^k, \quad k = 0, 1, 2, \ldots$$

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	one-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.

$$f(x;\, a, b, p_1) = p_1\, \mathbf{1}[x = 1] + (1 - p_1)\, f_{\mathrm{Beta}}(x;\, a, b)\, \mathbf{1}[x \in (0, 1)],$$

where f_{Beta} is the beta density and p_1 is oneprob.

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	one-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.

Uses the same density as oibeta with $a = \mu\phi$ and $b = (1-\mu)\phi$:

$$f(x;\, \mu, \phi, p_1) = p_1\, \mathbf{1}[x = 1] + (1 - p_1)\, f_{\mathrm{Beta}}(x;\, \mu\phi,\, (1-\mu)\phi)\, \mathbf{1}[x \in (0, 1)].$$

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

dpareto and ppareto allow for automatic differentiation with RTMB. The parameterisation follows the PARETO family of the gamlss.dist package.

$$f(x;\, \mu) = \frac{\mu}{x^{\mu+1}}, \quad x > 1.$$

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/.

See Also

llogis, gengamma

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(q, scale = 1, shape = 1, powershape = 1, log = FALSE)

hpgweibull(x, scale = 1, shape = 1, powershape = 1, log = FALSE)

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

rpgweibull(n, scale = 1, shape = 1, powershape = 1)
```

Arguments

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 $\log(p)$.
x, q	vector of quantiles
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

The densities and distribution functions allow for automatic differentiation with RTMB.

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

The parameterisation follows the PE and PE2 families of the gamlss.dist package.

For powerexp (PE), σ is the standard deviation; the density is

$$f(x;\, \mu, \sigma, \nu) = \frac{\nu}{2c\,\sigma\,\Gamma(1/\nu)} \exp\left(-\tfrac{1}{2} \left| \frac{x-\mu}{c\sigma} \right|^{\nu} \right),$$

where $c = [2^{-2/\nu}\Gamma(1/\nu)/\Gamma(3/\nu)]^{1/2}$.

For powerexp2 (PE2), σ is a scale parameter; the density is

$$f(x;\, \mu, \sigma, \nu) = \frac{\nu}{2\sigma\,\Gamma(1/\nu)} \exp\left(- \left| \frac{x-\mu}{\sigma} \right|^{\nu} \right).$$

Value

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

References

Nelson, D. B. (1991) Conditional heteroskedasticity in asset returns: a new approach. Econometrica, 59, 347-370.

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/.

See Also

bcpe, jsu, laplace

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)
```

---

rgmrf *Sample from a multivariate Gaussian with a sparse precision matrix*

Description

Draw samples from a multivariate Gaussian distribution specified by a sparse precision matrix. This is numerically efficient for high-dimensional but sparse systems.

Usage

```
rgmrf(n, mean = 0, Q)
```

Arguments

n	Number of samples to draw.
mean	Mean vector (or scalar, which will be recycled to match the dimension of Q).
Q	Sparse precision matrix (Σ^{-1}).

Value

A matrix of samples with rows corresponding to samples and columns to dimensions.

Examples

```
rgmrf(3, mean = c(1, 2, 3), Q = Matrix::Diagonal(3))
```

skellam *Skellam distribution*

Description

Probability mass function, distribution function, and random generation for the Skellam distribution.

Usage

```
dskellam(x, mu1, mu2, log = FALSE)

rskellam(n, mu1, mu2)
```

Arguments

x	integer vector of counts
mu1, mu2	Poisson means
log	logical; return log-density if TRUE
n	number of random values to return.

Details

The Skellam distribution is the distribution of the difference of two Poisson random variables. Specifically, if $X_1 \sim \text{Pois}(\mu_1)$ and $X_2 \sim \text{Pois}(\mu_2)$, then $X_1 - X_2 \sim \text{Skellam}(\mu_1, \mu_2)$.

This implementation of dskellam allows for automatic differentiation with RTMB.

Value

dskellam gives the probability mass function and rskellam generates random deviates.

Examples

```
x <- rskellam(1, 2, 3)
d <- dskellam(x, 2, 3)
```

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, lower.tail = TRUE, log.p = FALSE)

qskewnorm(p, xi = 0, omega = 1, alpha = 0, lower.tail = TRUE, log.p = FALSE)

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, 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 dskewnorm allows for automatic differentiation with RTMB while the other functions are imported from the sn package. See sn::dsn for more details.

$$f(x;\, \xi, \omega, \alpha) = \frac{2}{\omega}\, \phi\left(\frac{x-\xi}{\omega}\right) \Phi\left(\alpha \frac{x-\xi}{\omega}\right),$$

where ϕ and Φ are the standard normal PDF and CDF.

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, lower.tail = TRUE, log.p = FALSE)

qskewnorm2(p, mean = 0, sd = 1, alpha = 0, lower.tail = TRUE, log.p = FALSE)

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, 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 dskewnorm2 allows for automatic differentiation with RTMB while the other functions are imported from the sn package.

Uses the same density as skewnorm with location ξ, scale ω, and shape α reparameterised from the mean m, standard deviation s, and skewness parameter α:

$$\delta = \frac{\alpha}{\sqrt{1+\alpha^2}}, \quad \omega = \frac{s}{\sqrt{1-2\delta^2/\pi}}, \quad \xi = m - \omega\delta\sqrt{2/\pi}.$$

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, lower.tail = TRUE, log.p = FALSE)

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 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.

$$f(x;\, \mu, \sigma, \lambda, \nu) = \frac{2}{\sigma}\, f_t\left(\frac{x-\mu}{\sigma};\, \nu\right) F_t\left(\lambda\sqrt{\frac{\nu+1}{\nu+z^2}}\cdot z;\, \nu+1\right),$$

where $z = (x-\mu)/\sigma$, $f_t(\cdot;\nu)$ is the Student-t PDF, and $F_t(\cdot;\nu+1)$ is its CDF.

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,
  lower.tail = TRUE,
  log.p = FALSE
)

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 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. 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 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.

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

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

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, lower.tail = TRUE, log.p = FALSE)

rt2(n, mu, sigma, df)

qt2(p, mu, sigma, df, lower.tail = TRUE, log.p = FALSE)

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, 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 $P[X > x]$.
n	number of random values to return.
p	vector of probabilities

Details

This implementation of dt2 allows for automatic differentiation with RTMB.

$$f(x;\, \mu, \sigma, \nu) = \frac{1}{\sigma}\, f_t\left(\frac{x-\mu}{\sigma};\, \nu\right),$$

where $f_t(\cdot;\nu)$ is the Student-t PDF with ν degrees of freedom.

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.

$$f(x;\, \mu, \sigma, a, b) = \frac{\phi((x-\mu)/\sigma)/\sigma}{\Phi((b-\mu)/\sigma) - \Phi((a-\mu)/\sigma)}, \quad x \in [a, b],$$

where ϕ and Φ are the standard normal PDF and CDF.

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 $P[X \leq x]$, 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`.

$$f(x;\, \nu, a, b) = \frac{f_t(x;\, \nu)}{F_t(b;\, \nu) - F_t(a;\, \nu)}, \quad x \in [a, b],$$

where $f_t(\cdot;\nu)$ and $F_t(\cdot;\nu)$ are the Student-t PDF and CDF.

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`.

$$f(x;\, \nu, \mu, \sigma, a, b) = \frac{f_t((x-\mu)/\sigma;\, \nu)/\sigma}{F_t((b-\mu)/\sigma;\, \nu) - F_t((a-\mu)/\sigma;\, \nu)}, \quad x \in [a, b].$$

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,
  lower.tail = TRUE,
  log.p = FALSE
)

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
lower.tail	logical; if TRUE (default), probabilities are $P[X \leq x]$, otherwise $P[X > x]$.
log.p	logical; if TRUE, probabilities are returned as $\log(p)$.
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.

$$f(x;\, \mu, \kappa) = \frac{\exp(\kappa \cos(x - \mu))}{2\pi\, I_0(\kappa)},$$

where I_0 is the modified Bessel function of the first kind of order 0.

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.

$$f(\mathbf{x};\, \boldsymbol{\mu}, \kappa) = C_p(\kappa)\, \exp(\kappa\, \boldsymbol{\mu}^\top \mathbf{x}),$$

where $C_p(\kappa) = \kappa^{p/2-1}/((2\pi)^{p/2} I_{p/2-1}(\kappa))$ is the normalising constant and I_ν is the modified Bessel function of the first kind of order ν.

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)
```

---

wishart *Wishart distribution*

Description

Density and random generation for the wishart distribution

Usage

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

rwishart(n, nu, Sigma)
```

Arguments

x	positive definite $p \times p$ matrix or array of such matrices of dimension $p \times p \times n$ (for n density evaluations)
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

Details

$$f(X;\, \nu, \Sigma) = \frac{|X|^{(\nu-p-1)/2} \exp\!\left(-\frac{1}{2}\operatorname{tr}(\Sigma^{-1}X)\right)}{2^{\nu p/2}\, |\Sigma|^{\nu/2}\, \Gamma_p(\nu/2)},$$

where p is the dimension and Γ_p is the multivariate gamma function.

Value

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

Examples

```
# single input: matrix-valued
x <- rwishart(1, nu = 5, Sigma = diag(3))
d <- dwishart(x, nu = 5, Sigma = diag(3))

# multiple inputs: array of matrices
x <- rwishart(4, nu = 5, Sigma = diag(3))
d <- dwishart(x, nu = 5, Sigma = diag(3))

# multiple inputs for x, nu and Sigma
nu <- c(7,5,8,9)
Sigma <- array(dim = c(3,3,4))
for(i in 1:4) Sigma[,,i] <- (i + 3) * diag(3)
x <- rwishart(4, nu, Sigma)
d <- dwishart(x, nu, Sigma)
```

---

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.

$$f(x;\,\mu,\rho) = \frac{1}{2\pi}\cdot\frac{1-\rho^2}{1+\rho^2-2\rho\cos(x-\mu)}.$$

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`.

$$f(x;\, a, b, p_0) = p_0\, \mathbf{1}[x = 0] + (1 - p_0)\, f_{\mathrm{Beta}}(x;\, a, b)\, \mathbf{1}[x \in (0, 1)],$$

where p_0 is `zeroprob`.

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 $P[X \leq x]$, otherwise $P[X > x]$.
n	number of random values to return.
p	vector of probabilities

Details

This implementation allows for automatic differentiation with RTMB.

Uses the same density as zibeta with $a = \mu\phi$ and $b = (1-\mu)\phi$:

$$f(x;\, \mu, \phi, p_0) = p_0\, \mathbf{1}[x = 0] + (1 - p_0)\, f_{\mathrm{Beta}}(x;\, \mu\phi,\, (1-\mu)\phi)\, \mathbf{1}[x \in (0, 1)].$$

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 $P[X \leq x]$, otherwise, $P[X > x]$.
`n`	number of random values to return.

Details

This implementation allows for automatic differentiation with `RTMB`.

$$P(X = k;\, n, p, p_0) = p_0\, \mathbf{1}[k = 0] + (1 - p_0)\, P_{\mathrm{Bin}}(k;\, n, p),$$

where p_0 is `zeroprob`.

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 $P[X \leq x]$, otherwise, $P[X > x]$.
n	number of random values to return

Details

This implementation allows for automatic differentiation with RTMB.

$$f(x;\, \alpha, s, p_0) = p_0\, \mathbf{1}[x = 0] + (1 - p_0)\, f_{\mathrm{Gamma}}(x;\, \alpha, s)\, \mathbf{1}[x > 0],$$

where p_0 is zeroprob.

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, lower.tail = TRUE, log.p = FALSE)

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, 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 $P[X > x]$.
n	number of random values to return

Details

This implementation allows for automatic differentiation with RTMB.

Uses the same density as zigamma with shape $= \mu^2/s^2$ and scale $= s^2/\mu$:

$$f(x;\, \mu, s, p_0) = p_0\, \mathbf{1}[x = 0] + (1 - p_0)\, f_{\mathrm{Gamma}}(x;\, \mu^2/s^2,\, s^2/\mu)\, \mathbf{1}[x > 0].$$

Value

dzigamma2 gives the density, pzigamma2 gives the distribution function, and rzigamma2 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`.

$$f(x;\, \mu, \lambda, p_0) = p_0\, \mathbf{1}[x = 0] + (1 - p_0)\, f_{\mathrm{IG}}(x;\, \mu, \lambda)\, \mathbf{1}[x > 0],$$

where p_0 is `zeroprob` and f_{IG} is the inverse Gaussian density.

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`.

$$f(x;\, \mu_\ell, \sigma_\ell, p_0) = p_0\, \mathbf{1}[x = 0] + (1 - p_0)\, f_{\mathrm{LN}}(x;\, \mu_\ell, \sigma_\ell)\, \mathbf{1}[x > 0],$$

where p_0 is `zeroprob`, μ_ℓ = `meanlog`, σ_ℓ = `sdlog`, and f_{LN} is the log-normal density.

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`.

$$P(X = k;\, r, p, p_0) = p_0\, \mathbf{1}[k = 0] + (1 - p_0)\, P_{\mathrm{NB}}(k;\, r, p),$$

where p_0 is `zeroprob`.

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 $P[X \leq x]$, otherwise, $P[X > x]$.
n	number of random values to return.
p	vector of probabilities

Details

This implementation allows for automatic differentiation with RTMB.

Uses the same density as zinbinom with $p = r/(r + \mu)$:

$$P(X = k;\, \mu, r, p_0) = p_0\, \mathbf{1}[k = 0] + (1 - p_0)\, P_{\mathrm{NB}}\left(k;\, r,\, \tfrac{r}{r+\mu}\right).$$

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 $P[X \leq x]$, otherwise, $P[X > x]$.
n	number of random values to return.

Details

This implementation allows for automatic differentiation with RTMB.

$$P(X = k;\, \lambda, p_0) = p_0\, \mathbf{1}[k = 0] + (1 - p_0)\, P_{\mathrm{Pois}}(k;\, \lambda),$$

where p_0 is zeroprob.

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)
```

`ziweibull` *Zero-inflated Weibull distribution*

Description

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

Usage

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

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

rziweibull(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 $P[X \leq x]$, otherwise, $P[X > x]$.
n	number of random values to return

Details

This implementation allows for automatic differentiation with RTMB.

$$f(x;\, k, \lambda, p_0) = p_0\, \mathbf{1}[x = 0] + (1 - p_0)\, f_{\text{Weibull}}(x;\, k, \lambda)\, \mathbf{1}[x > 0],$$

where p_0 is zeroprob, k = shape, λ = scale.

Value

dziweibull gives the density, pziweibull gives the distribution function, and rziweibull generates random deviates.

Examples

```
x <- rziweibull(1, 1, 1, 0.5)
d <- dziweibull(x, 1, 1, 0.5)
p <- pziweibull(x, 1, 1, 0.5)
```

`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`	one-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`.

$$f(x;\, a, b, p_0, p_1) = p_0\, \mathbf{1}[x = 0] + (1 - p_0 - p_1)\, f_{\mathrm{Beta}}(x;\, a, b)\, \mathbf{1}[x \in (0, 1)] + p_1\, \mathbf{1}[x = 1],$$

where p_0 = `zeroprob` and p_1 = `oneprob`.

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`	one-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`.

Uses the same density as `zoibeta` with $a = \mu\phi$ and $b = (1 - \mu)\phi$.

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 $P[X \leq x]$, 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, 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 $P[X \leq x]$, 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, 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 $P[X \leq x]$, 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, 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