Package 'RTMBdist'

November 6, 2026

Type Package

Title Distributions Compatible with Automatic Differentiation by 'RTMB'

Version 1.2.1

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, methods, Matrix, circular, sn, statmod, movMF

Depends R (>= 3.5.0), RTMB (>= 2.0)

RoxygenNote 7.3.3

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

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 2026-11-06 12:50:16 UTC

Contents

abs_smooth . . . 4
bccg . . . 5
bcpe . . . 6
bct . . . 7
bell . . . 9
bell2 . . . 11
beta2 . . . 12
betabinom . . . 13
betaprime . . . 14
bisa . . . 16
bnbinom . . . 17
bnbinom2 . . . 19
card . . . 20
cclayton . . . 22
cfold . . . 23
cfrank . . . 24
cgaussian . . . 25
cgmrf . . . 26
cgumbel . . . 27
cjw . . . 28
cmvgauss . . . 29
combinom . . . 30
dcopula . . . 32
ddcopula . . . 34
dirichlet . . . 35
dirmult . . . 36
dmvcopula . . . 37
erf . . . 38
exgauss . . . 39
foldnorm . . . 40
frechet . . . 41
gamma2 . . . 43
gb2 . . . 44
gengamma . . . 46
genpois . . . 47
geom.ad . . . 49
gev . . . 50
gompertz . . . 51
gpd . . . 52
gumbel . . . 54
halfcauchy . . . 55
halft . . . 56
hbetabinom . . . 58
hbinom . . . 59
hgeom . . . 61
hnbinom . . . 62

hnbinom2 . . . 63
hpois . . . 65
invchisq . . . 66
invgamma . . . 67
invgauss . . . 68
jsu . . . 69
kumar . . . 72
lambertW . . . 73
laplace . . . 74
laplace_check . . . 75
llogis . . . 76
mcreport . . . 78
mvt . . . 81
nbinom2 . . . 82
oibeta . . . 83
oibeta2 . . . 84
pareto . . . 85
pgweibull . . . 86
pig . . . 88
powerexp . . . 89
rgmrf . . . 91
sichel . . . 92
skellam . . . 94
skewnorm . . . 95
skewnorm2 . . . 96
skewt . . . 98
skewt2 . . . 99
t2 . . . 101
truncnorm . . . 102
trunct . . . 103
trunct2 . . . 104
vm . . . 105
vmf . . . 107
vmf2 . . . 108
waring . . . 109
wishart . . . 111
wrpcauchy . . . 112
wrpnorm . . . 114
yules . . . 115
zero_inflate . . . 116
zibeta . . . 117
zibeta2 . . . 118
zibetabinom . . . 119
zibinom . . . 121
zigamma . . . 122
zigamma2 . . . 123
zigeom . . . 124
ziinvgauss . . . 125

zilnorm 126
zinbinom 127
zinbinom2 128
zipois 129
ziweibull 130
zoibeta 131
zoibeta2 132
ztbetabinom 133
ztbinom 135
ztgeom 136
ztnbinom 137
ztnbinom2 138
ztpois 139

Index **141**

`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

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

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, distribution function and random generation for the beta-binomial distribution.

Usage

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

pbetabinom(q, size, shape1, shape2, lower.tail = TRUE, log.p = 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.
q	vector of quantiles.
lower.tail	logical; if TRUE (default), probabilities are $P[X \leq q]$, otherwise $P[X > q]$.
log.p	logical; if TRUE, probabilities 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.

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

The distribution function has no closed form and is computed by summing the probability mass function over $0, \ldots, q$. It is AD-compatible in the parameters, while q and size must be numeric data. This is also what one-step-ahead (OSA) residuals via RTMB::oneStepPredict need, so these are supported, e.g. with method = "cdf" and discrete = TRUE.

Value

dbetabinom gives the density, pbetabinom gives the distribution function, and rbetabinom generates random samples.

Examples

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

`bisa` *Birnbaum-Saunders distribution*

Description

Density, distribution function, quantile function, and random generation for the Birnbaum-Saunders (fatigue life) distribution.

Usage

```
dbisa(x, scale = 1, shape, log = FALSE)

pbisa(q, scale = 1, shape, lower.tail = TRUE, log.p = FALSE)

qbisa(p, scale = 1, shape, lower.tail = TRUE, log.p = FALSE)

rbisa(n, scale = 1, shape)
```

Arguments

`x`, `q`	vector of quantiles.
`scale`	scale parameter β, the median, must be positive.
`shape`	shape parameter α, 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 $P[X \le x]$, otherwise $P[X > x]$.
`p`	vector of probabilities.
`n`	number of random values to return.

Details

This implementation allows for automatic differentiation with `RTMB`.

The Birnbaum-Saunders distribution with scale β and shape α is the distribution of X for which

$$\xi(X) = \frac{1}{\alpha}\left(\sqrt{X/\beta} - \sqrt{\beta/X}\right) = \frac{2}{\alpha}\sinh\left(\tfrac{1}{2}\log(X/\beta)\right)$$

is standard normal, so that $F(x) = \Phi(\xi(x))$ and

$$f(x) = \phi(\xi(x))\,\frac{x+\beta}{2\alpha\sqrt{\beta}\,x^{3/2}}, \quad x > 0.$$

The scale β is the median, and $E(X) = \beta(1+\alpha^2/2)$. The quantile function is $\beta\exp(2\,\mathrm{asinh}(\alpha z/2))$ with $z = \Phi^{-1}(p)$.

Value

`dbisa` gives the density, `pbisa` gives the distribution function, `qbisa` gives the quantile function, and `rbisa` generates random deviates.

References

Birnbaum, Z. W. and Saunders, S. C. (1969) A new family of life distributions. Journal of Applied Probability, 6, 319-327.

See Also

invgauss, pgweibull, gompertz

Examples

```
set.seed(1)
x <- rbisa(5, scale = 2, shape = 0.5)
d <- dbisa(x, 2, 0.5)
p <- pbisa(x, 2, 0.5)
q <- qbisa(p, 2, 0.5)
```

---

bnbinom *Beta-negative binomial distribution*

Description

Probability mass function, distribution function and random generation for the beta-negative binomial distribution.

Usage

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

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

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

Arguments

x	vector of non-negative counts.
size	positive number of successes (need not be an integer).
shape1	positive shape parameter 1 of the beta prior.
shape2	positive shape parameter 2 of the beta prior.
log	logical; if TRUE, probabilities are returned on the log scale.
q	vector of quantiles.
lower.tail	logical; if TRUE (default), probabilities are $P[X \leq q]$, otherwise $P[X > q]$.
log.p	logical; if TRUE, probabilities are returned on the log scale.
n	number of random values to return (for rbnbinom).

Details

dbnbinom allows for automatic differentiation with RTMB.

The beta-negative binomial arises by giving the success probability of a negative binomial a beta prior, in the same way that the beta-binomial does for the binomial:

$$P(X = k;\, r, a, b) = \frac{\Gamma(k + r)}{k!\,\Gamma(r)} \frac{B(a + r,\, b + k)}{B(a,\, b)}, \quad k = 0, 1, 2, \ldots$$

The extra beta layer gives a much heavier tail than the negative binomial: the mean $rb/(a-1)$ exists only for $a > 1$ and the variance

$$\frac{rb(r + a - 1)(b + a - 1)}{(a - 2)(a - 1)^2}$$

only for $a > 2$. As $a \to \infty$ with $b/(a + b)$ held fixed the distribution collapses to the negative binomial.

Note that the three parameters are only weakly identified: the likelihood has a long ridge along which size and shape2 trade off against each other, so fitting all three at once needs either a lot of data or a restriction. The mean parameterisation is usually the more stable one to estimate in.

The distribution function has no closed form and is computed by summing the probability mass function over $0, \ldots, q$, so its cost grows with the largest q. It is AD-compatible in the parameters, while q must be numeric data. This is also what one-step-ahead (OSA) residuals via RTMB::oneStepPredict need, so these are supported, e.g. with method = "cdf" and discrete = TRUE.

Value

dbnbinom gives the probability mass function, pbnbinom gives the distribution function, and rbnbinom generates random deviates.

References

Johnson, N. L., Kemp, A. W. and Kotz, S. (2005) Univariate Discrete Distributions, 3rd edition, Wiley, doi:10.1002/0471715816.

See Also

bnbinom2, betabinom, nbinom2

Examples

```
set.seed(123)
x <- rbnbinom(5, size = 3, shape1 = 4, shape2 = 2)
d <- dbnbinom(x, size = 3, shape1 = 4, shape2 = 2)
p <- pbnbinom(x, size = 3, shape1 = 4, shape2 = 2)
```

bnbinom2 *Reparameterised beta-negative binomial distribution*

Description

Probability mass function, distribution function and random generation for the beta-negative binomial distribution reparameterised in terms of its mean.

Usage

```
dbnbinom2(x, mu, sigma, nu, log = FALSE)

pbnbinom2(q, mu, sigma, nu, lower.tail = TRUE, log.p = FALSE)

rbnbinom2(n, mu, sigma, nu)
```

Arguments

x	vector of non-negative counts.
mu	mean parameter, must be positive.
sigma	dispersion parameter, must be positive. The variance is finite only for sigma < 1.
nu	dispersion parameter, must be positive.
log	logical; if TRUE, probabilities are returned on the log scale.
q	vector of quantiles.
lower.tail	logical; if TRUE (default), probabilities are $P[X \leq q]$, otherwise $P[X > q]$.
log.p	logical; if TRUE, probabilities are returned on the log scale.
n	number of random values to return (for rbnbinom2).

Details

dbnbinom2 allows for automatic differentiation with RTMB. The parameterisation follows the BNB family of the gamlss.dist package, in which μ is exactly the mean.

Writing r, a and b for the arguments size, shape1 and shape2 of dbnbinom, the reparameterisation is

$$a = \frac{1}{\sigma} + 1, \qquad b = \frac{\mu\nu}{\sigma}, \qquad r = \frac{1}{\nu}.$$

This gives $E(X) = \mu$ for every admissible σ and ν, where the original parameterisation needs $a > 1$ for a mean to exist at all. The variance is

$$\mathrm{Var}(X) = \frac{\mu(\sigma + \nu)(\mu\nu + 1)}{\nu(1 - \sigma)},$$

which is finite for $\sigma < 1$ and increases without bound as $\sigma \to 1$. Both σ and ν add dispersion beyond the negative binomial, which is recovered as $\sigma \to 0$.

Because the mean is pinned to a single parameter, this parameterisation is the more stable of the two to estimate in; see bnbinom for the identifiability problem it avoids.

The distribution function has no closed form and is computed by summing the probability mass function over $0, \ldots, q$, so its cost grows with the largest q. It is AD-compatible in the parameters, while q must be numeric data. This is also what one-step-ahead (OSA) residuals via RTMB::oneStepPredict need, so these are supported, e.g. with method = "cdf" and discrete = TRUE.

Value

dbnbinom2 gives the probability mass function, pbnbinom2 gives the distribution function, and rbnbinom2 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

bnbinom, nbinom2, betabinom

Examples

```
set.seed(123)
x <- rbnbinom2(5, mu = 4, sigma = 0.4, nu = 0.5)
d <- dbnbinom2(x, mu = 4, sigma = 0.4, nu = 0.5)
p <- pbnbinom2(x, mu = 4, sigma = 0.4, nu = 0.5)
```

---

card *Cardioid distribution*

Description

Density, distribution function, and random generation for the cardioid distribution.

Usage

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

pcard(q, mu = 0, rho, from = NULL, lower.tail = TRUE, log.p = FALSE)

rcard(n, mu = 0, rho)
```

Arguments

x, q	vector of angles in radians.
mu	mean direction in radians.
rho	concentration parameter, must be in the interval from 0 to 1/2.
log, log.p	logical; if TRUE, probabilities/ densities p are returned as $\log(p)$.
from	origin, in radians, at which the circle is cut open for the distribution function. If NULL (default), it is set to mu - pi.
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.

The cardioid distribution has density

$$f(x;\, \mu, \rho) = \frac{1 + 2\rho\cos(x - \mu)}{2\pi}, \qquad 0 \leq \rho \leq \frac{1}{2},$$

and is the circular uniform for $\rho = 0$. With the circle cut open at $\mu - \pi$, its distribution function is

$$F(q) = \frac{q - \mu + \pi}{2\pi} + \frac{\rho}{\pi}\sin(q - \mu), \quad \mu - \pi \leq q < \mu + \pi.$$

The origin can be changed with from, exactly as for pwrpcauchy, where the choice of origin and the one-step-ahead (OSA) residuals are explained. As there, OSA residuals via RTMB::oneStepPredict with method = "cdf" cut the circle at the fixed origin $-\pi$.

Value

dcard gives the density, pcard gives the distribution function, and rcard generates random deviates in $[-\pi, \pi)$.

References

Jammalamadaka, S. R. and SenGupta, A. (2001) Topics in Circular Statistics. World Scientific.

See Also

wrpcauchy, wrpnorm, vm

Examples

```
set.seed(1)
x <- rcard(10, 0, 0.3)
d <- dcard(x, 0, 0.3)
p <- pcard(x, 0, 0.3)
```

`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)
```

`cfold` *Folded circular-linear copula constructor*

Description

Turns a copula for two linear variables into a circular-linear copula that is symmetric in the sign of the angle, intended to be used with `dcopula` to join a turning angle and a step length.

Usage

```
cfold(copula)
```

Arguments

`copula`	function of two arguments `(u, v)` returning a log copula density for two linear variables, e.g. `cgaussian(0.5)`.

Details

The copula density is

$$c(u, v) = c_0\big(1 - |2u - 1|,\, v\big),$$

where u is the distribution function of the circular margin, v that of the linear margin and c_0 is the density of the copula passed as `copula`, e.g. `cgaussian`, `cclayton`, `cgumbel` or `cfrank`. This is a copula for any c_0. It is the rectangular patchwork copula of Hodel and Fieberg (2022), with the linear copula in the rectangle $u \le 1/2$ and its mirror image in the rectangle $u > 1/2$.

The map $u \mapsto 1 - |2u - 1|$ folds the circle at $u = 1/2$. For a circular margin that is symmetric about its mean direction and whose distribution function is cut open at the antipode of that mean direction, as by default in `pwrpcauchy` and `pvm`, $u = 1/2$ is the mean direction. $1 - |2u - 1|$ is then the distribution function of the angular distance to the mean direction, counted from the antipode: it is 1 for angles at the mean direction and 0 for angles opposite to it. In words, c_0 links the straightness of a step to its length. A copula with positive dependence, such as `cgaussian(rho)` with `rho > 0`, makes long steps straight and lets short steps turn in either direction, the pattern most common in movement data. Unlike `cjw`, the resulting copula is symmetric, $c(u, v) = c(1 - u, v)$.

In `dcopula`, the circular margin must be the *first* one, i.e. `d1` and `p1` belong to the angle.

Value

Function of two arguments `(u, v)` returning the log copula density, with `u` for the circular and `v` for the linear margin.

References

Hodel, F. H. and Fieberg, J. R. (2022) Circular-linear copulae for animal movement data. Methods in Ecology and Evolution, 13, doi:10.1111/2041-210X.13821.

Durante, F., Saminger-Platz, S. and Sarkoci, P. (2009) Rectangular patchwork for bivariate copulas and tail dependence. Communications in Statistics - Theory and Methods, 38, 2515-2527, doi:10.1080/03610920802571203.

See Also

dcopula(), cjw(), vm, wrpcauchy

Examples

```
# turning angles with von Mises margin, step lengths with Weibull margin
angle <- c(-2, -0.3, 0.1, 1.5); step <- c(0.2, 1.1, 2.4, 0.6)
d1 <- dvm(angle, 0, 2, log = TRUE); p1 <- pvm(angle, 0, 2)
d2 <- dweibull(step, 2, 1, log = TRUE); p2 <- pweibull(step, 2, 1)
dcopula(d1, d2, p1, p2, copula = cfold(cgaussian(0.5)), log = TRUE)

# wrapped Cauchy margin, which also allows for automatic differentiation
d1 <- dwrpcauchy(angle, 0, 0.5, log = TRUE); p1 <- pwrpcauchy(angle, 0, 0.5)
dcopula(d1, d2, p1, p2, copula = cfold(cclayton(2)), log = TRUE)

# the copula is symmetric in the sign of the angle
cop <- cfold(cclayton(2))
cop(c(0.2, 0.8), 0.9)
```

---

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

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

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

$$c(u, v; \theta) = \frac{e^{-A}}{uv} (\tilde{u}\tilde{v})^{\theta-1} S^{1/\theta-2}(A + \theta - 1),$$

where $\tilde{u} = -\log u$, $\tilde{v} = -\log v$, $S = \tilde{u}^\theta + \tilde{v}^\theta$ and $A = S^{1/\theta}$, so that e^{-A} is the copula CDF.

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)
```

`cjw` *Johnson-Wehrly circular-linear copula constructor*

Description

Returns a function computing the log density of the circular-linear copula of Johnson and Wehrly (1978), intended to be used with `dcopula` to join a circular and a linear margin, such as the turning angles and step lengths of an animal track.

Usage

```
cjw(g, ..., q = 1)
```

Arguments

`g`	circular density function with first argument `x` and an argument `log`, such as `dvm` or `dwrpcauchy`.
`...`	parameters passed to `g`, such as `mu` and `kappa` for `dvm` or `mu` and `rho` for `dwrpcauchy`.
`q`	direction of the dependence, either `1` or `-1`.

Details

The copula density is

$$c(u, v) = 2\pi\, g\big(2\pi(u - qv)\big),$$

where u is the distribution function of the circular margin, v that of the linear margin, g is a density on the circle, the *binding density*, and $q = \pm 1$. The joint density of an angle θ and a step length s is then

$$f(\theta, s) = 2\pi\, g\big(2\pi(F_1(\theta) - qF_2(s))\big) f_1(\theta) f_2(s).$$

Because g is periodic, c is a copula for any circular density and any integer $q \neq 0$; only $q = \pm 1$ is allowed here. Any circular density of this package that allows for automatic differentiation can be used for g, in particular `dvm` and `dwrpcauchy`. Its concentration controls the strength of the dependence, and `q = -1` reverses its direction.

In `dcopula`, the circular margin must be the *first* one, i.e. `d1` and `p1` belong to the angle. Its distribution function can be `pwrpcauchy`, which allows for automatic differentiation. Where the circle is cut open for this distribution function only shifts u by a constant, which the location of g absorbs. The likelihood is therefore the same for any origin, but the meaning of the location of g is not.

The dependence of this copula is a helix: as v goes from 0 to 1, the angle it favours turns once around the circle. The copula is hence not symmetric in the sign of the angle, $c(u, v) \neq c(1 - u, v)$. With a turning angle margin centred at 0 and the location of g at 0, for example, medium steps are straight, short steps turn one way and long steps the other. It therefore cannot capture the pattern most common in movement data, where long steps are straight and short steps turn in either direction. For this, see `cfold`.

Random pairs from the copula are obtained by drawing v uniformly and z from g, and setting $u = (z/(2\pi) + qv) \bmod 1$; see the examples.

Value

Function of two arguments `(u, v)` returning the log copula density, with `u` for the circular and `v` for the linear margin.

References

Johnson, R. A. and Wehrly, T. E. (1978) Some angular-linear distributions and related regression models. Journal of the American Statistical Association, 73, 602-606, doi:10.1080/01621459.1978.10480062.

Hodel, F. H. and Fieberg, J. R. (2022) Circular-linear copulae for animal movement data. Methods in Ecology and Evolution, 13, doi:10.1111/2041-210X.13821.

See Also

`dcopula()`, `cfold()`, vm, wrpcauchy

Examples

```
# turning angles with wrapped Cauchy margin, step lengths with Weibull margin
angle <- c(-2, -0.3, 0.1, 1.5); step <- c(0.2, 1.1, 2.4, 0.6)
d1 <- dwrpcauchy(angle, 0, 0.5, log = TRUE); p1 <- pwrpcauchy(angle, 0, 0.5)
d2 <- dweibull(step, 2, 1, log = TRUE); p2 <- pweibull(step, 2, 1)

# von Mises binding density
dcopula(d1, d2, p1, p2, copula = cjw(dvm, mu = 0, kappa = 2), log = TRUE)

# wrapped Cauchy binding density, dependence in the other direction
dcopula(d1, d2, p1, p2, copula = cjw(dwrpcauchy, mu = 0, rho = 0.7, q = -1), log = TRUE)

# simulation from the joint distribution with von Mises binding density
n <- 1000
v <- runif(n)
z <- rvm(n, mu = 0, kappa = 2)
u <- (z / (2 * pi) + v) %% 1
angle <- qwrpcauchy(u, 0, 0.5)
step <- qweibull(v, 2, 1)
```

---

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)

Circular-linear copulas, e.g. for turning angles and step lengths, with the circular margin as the first one:

- `cjw` (Johnson-Wehrly copula with a circular binding density such as `dvm` or `dwrpcauchy`)
- `cfold` (folds any of the copulas above into one that is symmetric in the sign of the angle)

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`.

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

With $\mu = 0$ this is the half-normal distribution, the limiting case of the half-t as its degrees of freedom grow.

Value

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

See Also

halft, halfcauchy

Examples

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

---

frechet *Frechet distribution*

Description

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

Usage

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

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

qfrechet(p, mu = 0, sigma = 1, alpha = 1, lower.tail = TRUE, log.p = FALSE)

rfrechet(n, mu = 0, sigma = 1, alpha = 1)
```

Arguments

x, q	vector of quantiles
mu	location parameter, the lower end point of the support.
sigma	scale parameter, must be positive.
alpha	shape 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

dfrechet and pfrechet allow for automatic differentiation with RTMB.

With $y = (x - \mu)/\sigma$ the density is

$$f(x;\, \mu, \sigma, \alpha) = \frac{\alpha}{\sigma} y^{-\alpha-1} \exp(-y^{-\alpha}), \quad x > \mu,$$

and the distribution function is $F(x) = \exp(-y^{-\alpha})$.

The Frechet distribution is the heavy-tailed extreme value distribution: it is the generalised extreme value distribution with shape $\xi = 1/\alpha > 0$, reparameterised so that the shape enters as a tail index rather than as a reciprocal. All moments of order α and above are infinite, so the mean exists only for $\alpha > 1$ and the variance only for $\alpha > 2$.

Value

dfrechet gives the density, pfrechet gives the distribution function, qfrechet gives the quantile function, and rfrechet generates random deviates.

References

Frechet, M. (1927) Sur la loi de probabilite de l'ecart maximum. Annales de la Societe Polonaise de Mathematique, 6, 93-116.

Kotz, S. and Nadarajah, S. (2000) Extreme Value Distributions: Theory and Applications, Imperial College Press, doi:10.1142/p191.

See Also

gev, gumbel, pareto

Examples

```
set.seed(123)
x <- rfrechet(5, mu = 0, sigma = 1, alpha = 3)
d <- dfrechet(x, mu = 0, sigma = 1, alpha = 3)
p <- pfrechet(x, mu = 0, sigma = 1, alpha = 3)
q <- qfrechet(p, mu = 0, sigma = 1, alpha = 3)
```

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)
```

gb2 *Generalised beta distribution of the second kind and its special cases*

Description

Density, distribution function, quantile function, and random generation for the generalised beta distribution of the second kind (GB2) and three of its special cases: the Dagum, Singh-Maddala (Burr XII) and Lomax (Pareto II) distributions.

Usage

```
dgb2(x, scale = 1, shape1, shape2, shape3, log = FALSE)

pgb2(q, scale = 1, shape1, shape2, shape3, lower.tail = TRUE, log.p = FALSE)

qgb2(p, scale = 1, shape1, shape2, shape3, lower.tail = TRUE, log.p = FALSE)

rgb2(n, scale = 1, shape1, shape2, shape3)

ddagum(x, scale = 1, shape1, shape2, log = FALSE)

pdagum(q, scale = 1, shape1, shape2, lower.tail = TRUE, log.p = FALSE)

qdagum(p, scale = 1, shape1, shape2, lower.tail = TRUE, log.p = FALSE)

rdagum(n, scale = 1, shape1, shape2)

dsinmad(x, scale = 1, shape1, shape3, log = FALSE)

psinmad(q, scale = 1, shape1, shape3, lower.tail = TRUE, log.p = FALSE)

qsinmad(p, scale = 1, shape1, shape3, lower.tail = TRUE, log.p = FALSE)

rsinmad(n, scale = 1, shape1, shape3)

dlomax(x, scale = 1, shape3, log = FALSE)

plomax(q, scale = 1, shape3, lower.tail = TRUE, log.p = FALSE)

qlomax(p, scale = 1, shape3, lower.tail = TRUE, log.p = FALSE)

rlomax(n, scale = 1, shape3)
```

Arguments

x, q	vector of quantiles.
scale	scale parameter b, must be positive.

shape1	shape parameter a, must be positive.
shape2	shape parameter p, must be positive.
shape3	shape parameter q, 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 $P[X \le x]$, otherwise $P[X > x]$.
p	vector of probabilities.
n	number of random values to return.

Details

This implementation allows for automatic differentiation with RTMB.

The GB2 distribution with scale b and shape parameters a, p and q has density

$$f(x) = \frac{a\, x^{ap-1}}{b^{ap} B(p,q)\, (1 + (x/b)^a)^{p+q}}, \quad x > 0,$$

and its distribution function is a regularised incomplete beta function,

$$F(x) = I_z(p,q), \qquad z = \frac{(x/b)^a}{1 + (x/b)^a}.$$

The special cases fix some of the shapes, and their arguments keep the names of the shapes that remain:

- Dagum: $q = 1$, `ddagum(x, scale, shape1, shape2)`.
- Singh-Maddala, or Burr type XII: $p = 1$, `dsinmad(x, scale, shape1, shape3)`.
- Lomax, or Pareto type II: $a = p = 1$, `dlomax(x, scale, shape3)`.

The log-logistic distribution `dllogis` is the GB2 with $p = q = 1$, and the beta prime distribution `dbetaprime` the GB2 with $a = b = 1$. The k-th moment exists for $-ap < k < aq$.

The distribution functions are differentiable, so one-step-ahead (OSA) residuals via `RTMB::oneStepPredict` with `method = "cdf"` are supported.

Value

`d...` gives the density, `p...` gives the distribution function, `q...` gives the quantile function, and `r...` generates random deviates.

References

McDonald, J. B. (1984) Some generalized functions for the size distribution of income. Econometrica, 52, 647-663.

Kleiber, C. and Kotz, S. (2003) Statistical Size Distributions in Economics and Actuarial Sciences. Wiley.

See Also

llogis, betaprime, pareto

Examples

```
set.seed(1)
x <- rgb2(5, scale = 2, shape1 = 3, shape2 = 1.5, shape3 = 0.8)
d <- dgb2(x, 2, 3, 1.5, 0.8)
p <- pgb2(x, 2, 3, 1.5, 0.8)
q <- qgb2(p, 2, 3, 1.5, 0.8)

d <- ddagum(x, 2, 3, 1.5)
d <- dsinmad(x, 2, 3, 0.8)
d <- dlomax(x, 2, 0.8)
```

---

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)
```

---

geom.ad *AD-compatible geometric distribution*

Description

Density and distribution function for the geometric distribution, written so that they can be taped by RTMB.

Usage

```
dgeom.ad(x, prob, log = FALSE)

pgeom.ad(q, prob, lower.tail = TRUE, log.p = FALSE)
```

Arguments

x, q	integer vector of counts
prob	probability of success in each trial, in (0,1]
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 $P[X > x]$.

Details

stats already provides the geometric distribution, but its versions cannot be differentiated. These are AD-compatible replacements, reached automatically whenever an argument is an AD variable, so stats::dgeom and stats::pgeom are left untouched for ordinary use.

The parameterisation is the same as in stats: X is the number of failures before the first success, so

$$P(X = x;\, \pi) = \pi\,(1-\pi)^x, \quad x = 0, 1, 2, \ldots$$

The density is obtained from the negative binomial with size = 1, for which RTMB provides an AD method; the distribution function $1 - (1-\pi)^{x+1}$ is elementary.

Value

dgeom.ad gives the probability mass function and pgeom.ad gives the distribution function.

See Also

zigeom, ztgeom, hgeom

Examples

```
dgeom.ad(0:5, prob = 0.3)
pgeom.ad(0:5, prob = 0.3)
```

---

gev *Generalised extreme value distribution*

Description

Density, distribution function, quantile function, and random generation for the generalised extreme value (GEV) distribution.

Usage

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

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

qgev(p, mu = 0, sigma = 1, xi = 0, lower.tail = TRUE, log.p = FALSE)

rgev(n, mu = 0, sigma = 1, xi = 0)
```

Arguments

x, q	vector of quantiles
mu	location parameter
sigma	scale parameter, must be positive.
xi	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 $P[X \le x]$, otherwise $P[X > x]$.
p	vector of probabilities
n	number of random values to return

Details

dgev and pgev allow for automatic differentiation with RTMB.

With $z = (x - \mu)/\sigma$ the distribution function is

$$F(x;\, \mu, \sigma, \xi) = \exp\big(-t(x)\big), \qquad t(x) = \begin{cases} (1 + \xi z)^{-1/\xi} & \xi \neq 0, \\ e^{-z} & \xi = 0, \end{cases}$$

and the density is $f(x) = t(x)^{\xi+1} e^{-t(x)}/\sigma$.

The shape ξ determines the tail and with it the support: for $\xi > 0$ (Frechet case) the distribution is heavy-tailed on $x > \mu - \sigma/\xi$, for $\xi < 0$ (Weibull case) it is bounded above by $\mu - \sigma/\xi$, and $\xi = 0$ is the Gumbel distribution on the whole real line.

The three cases are covered by one expression, so no branch on the sign or the value of ξ is needed. In particular the derivative with respect to ξ is exact at $\xi = 0$, which is the usual starting value when the shape is estimated.

Value

`dgev` gives the density, `pgev` gives the distribution function, `qgev` gives the quantile function, and `rgev` generates random deviates.

References

Coles, S. (2001) An Introduction to Statistical Modeling of Extreme Values, Springer, doi:10.1007/978-1-4471-3675-0.

Jenkinson, A. F. (1955) The frequency distribution of the annual maximum (or minimum) values of meteorological elements. Quarterly Journal of the Royal Meteorological Society, 81, 158-171.

See Also

gumbel, gpd, frechet

Examples

```
set.seed(123)
x <- rgev(5, mu = 0, sigma = 1, xi = 0.2)
d <- dgev(x, mu = 0, sigma = 1, xi = 0.2)
p <- pgev(x, mu = 0, sigma = 1, xi = 0.2)
q <- qgev(p, mu = 0, sigma = 1, xi = 0.2)
```

---

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)
```

---

gpd *Generalised Pareto distribution*

Description

Density, distribution function, quantile function, and random generation for the generalised Pareto distribution (GPD).

Usage

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

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

qgpd(p, mu = 0, sigma = 1, xi = 0, lower.tail = TRUE, log.p = FALSE)

rgpd(n, mu = 0, sigma = 1, xi = 0)
```

Arguments

x, q	vector of quantiles
mu	location parameter, the threshold below which the density is zero.
sigma	scale parameter, must be positive.
xi	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 $P[X \le x]$, otherwise $P[X > x]$.
p	vector of probabilities
n	number of random values to return

Details

dgpd and pgpd allow for automatic differentiation with RTMB.

With $z = (x - \mu)/\sigma$ the survival function is

$$1 - F(x;\, \mu, \sigma, \xi) = \begin{cases} (1 + \xi z)^{-1/\xi} & \xi \neq 0, \\ e^{-z} & \xi = 0, \end{cases}$$

for $x \ge \mu$, and the density is

$$f(x;\, \mu, \sigma, \xi) = \frac{1}{\sigma}(1 + \xi z)^{-1/\xi - 1}.$$

This is the limiting distribution of exceedances over a high threshold, so μ is usually a fixed threshold rather than an estimated parameter. The support is $x \ge \mu$ for $\xi \ge 0$ and $\mu \le x \le \mu - \sigma/\xi$ for $\xi < 0$. At $\xi = 0$ the distribution is exponential with rate $1/\sigma$, and at $\xi > 0$ with $\mu = \sigma/\xi$ it is the Pareto distribution with $\mu = 1/\xi$.

The three cases are covered by one expression, so no branch on the sign or the value of ξ is needed. In particular the derivative with respect to ξ is exact at $\xi = 0$, which is the usual starting value when the shape is estimated.

The threshold itself belongs to the support: `dgpd(mu, mu, sigma, xi)` is $1/\sigma$, as `stats::dexp(0, rate)` is `rate`. The VGAM, evd and extraDistr implementations return zero there instead.

Value

dgpd gives the density, pgpd gives the distribution function, qgpd gives the quantile function, and rgpd generates random deviates.

References

Coles, S. (2001) An Introduction to Statistical Modeling of Extreme Values, Springer, doi:10.1007/978-1-4471-3675-0.

Pickands, J. (1975) Statistical inference using extreme order statistics. The Annals of Statistics, 3, 119-131.

See Also

gev, pareto, frechet

Examples

```
set.seed(123)
x <- rgpd(5, mu = 0, sigma = 1, xi = 0.3)
d <- dgpd(x, mu = 0, sigma = 1, xi = 0.3)
p <- pgpd(x, mu = 0, sigma = 1, xi = 0.3)
q <- qgpd(p, mu = 0, sigma = 1, xi = 0.3)
```

---

gumbel *Gumbel distribution*

Description

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

Usage

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

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

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

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

Arguments

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

Details

This implementation of dgumbel allows for automatic differentiation with RTMB.

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

Value

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

See Also

gev, frechet, gpd

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)
```

---

halfcauchy *Half-Cauchy distribution*

Description

Density, distribution function, quantile function, and random generation for the half-Cauchy distribution.

Usage

```
dhalfcauchy(x, sigma = 1, log = FALSE)

phalfcauchy(q, sigma = 1, lower.tail = TRUE, log.p = FALSE)

qhalfcauchy(p, sigma = 1, lower.tail = TRUE, log.p = FALSE)

rhalfcauchy(n, sigma = 1)
```

Arguments

x, q	vector of quantiles.
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 (default), probabilities are $P[X \le x]$, otherwise $P[X > x]$.
p	vector of probabilities.
n	number of random values to return.

Details

dhalfcauchy and phalfcauchy allow for automatic differentiation with RTMB.

The half-Cauchy is the distribution of $|Y|$ for Y Cauchy with scale σ:

$$f(x;\,\sigma) = \frac{2}{\pi\sigma\bigl(1 + (x/\sigma)^2\bigr)}, \quad x \geq 0,$$

with distribution function $F(x) = \frac{2}{\pi}\arctan(x/\sigma)$.

It is the standard weakly informative prior for the standard deviation of a hierarchical model, recommended by Gelman (2006) in place of the inverse-gamma: the density is flat and non-zero at the origin, so it does not force the variance component away from zero, while the tail is heavy enough to leave large values unpenalised. That makes it a natural companion to models fitted by the Laplace approximation, where the variance components are exactly the parameters at issue.

It has no moments of any order. It is the half-t with df = 1, and the folded normal with mu = 0 is the corresponding half-normal, which the half-t approaches as df grows.

Value

dhalfcauchy gives the density, phalfcauchy gives the distribution function, qhalfcauchy gives the quantile function, and rhalfcauchy generates random deviates.

References

Gelman, A. (2006) Prior distributions for variance parameters in hierarchical models. Bayesian Analysis, 1, 515-534, doi:10.1214/06-BA117A.

See Also

halft, foldnorm, trunct

Examples

```
set.seed(123)
x <- rhalfcauchy(5, sigma = 2)
d <- dhalfcauchy(x, sigma = 2)
p <- phalfcauchy(x, sigma = 2)
q <- qhalfcauchy(p, sigma = 2)
```

---

halft *Half-t distribution*

Description

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

Usage

```
dhalft(x, df, sigma = 1, log = FALSE)

phalft(q, df, sigma = 1, lower.tail = TRUE, log.p = FALSE)

qhalft(p, df, sigma = 1, lower.tail = TRUE, log.p = FALSE)

rhalft(n, df, sigma = 1)
```

Arguments

x, q	vector of quantiles.
df	degrees of freedom, must be positive.
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 (default), probabilities are $P[X \leq x]$, otherwise $P[X > x]$.
p	vector of probabilities.
n	number of random values to return.

Details

dhalft and phalft allow for automatic differentiation with RTMB, with respect to df as well as sigma.

The half-t is the distribution of $|Y|$ for $Y = \sigma T$ and T Student t on ν degrees of freedom:

$$f(x;\, \nu, \sigma) = \frac{2}{\sigma} f_T(x/\sigma;\, \nu), \quad x \geq 0,$$

with distribution function $F(x) = 2F_T(x/\sigma;\, \nu) - 1$.

Together with the half-Cauchy, which is the case df = 1, this is the standard weakly informative prior for the standard deviation of a hierarchical model, recommended by Gelman (2006) in place of the inverse-gamma. The degrees of freedom set how heavy the tail is: small df leaves large values essentially unpenalised, and as df grows the distribution approaches the half-normal, which is the folded normal with mu = 0.

The mean

$$E(X) = \frac{2\sigma\sqrt{\nu}\,\Gamma\left(\frac{\nu+1}{2}\right)}{\sqrt{\pi}\,(\nu-1)\,\Gamma\left(\frac{\nu}{2}\right)}$$

exists only for $\nu > 1$ and the variance $\sigma^2\nu/(\nu-2) - E(X)^2$ only for $\nu > 2$.

Value

dhalft gives the density, phalft gives the distribution function, qhalft gives the quantile function, and rhalft generates random deviates.

References

Gelman, A. (2006) Prior distributions for variance parameters in hierarchical models. Bayesian Analysis, 1, 515-534, doi:10.1214/06-BA117A.

See Also

halfcauchy, foldnorm, trunct, t2

Examples

```
set.seed(123)
x <- rhalft(5, df = 3, sigma = 2)
d <- dhalft(x, df = 3, sigma = 2)
p <- phalft(x, df = 3, sigma = 2)
q <- qhalft(p, df = 3, sigma = 2)
```

---

hbetabinom *Hurdle beta-binomial distribution*

Description

Probability mass function, distribution function and random generation for the hurdle (zero-altered) beta-binomial distribution.

Usage

```
dhbetabinom(x, size, shape1, shape2, zeroprob = 0.5, log = FALSE)

phbetabinom(
  q,
  size,
  shape1,
  shape2,
  zeroprob = 0.5,
  lower.tail = TRUE,
  log.p = FALSE
)

rhbetabinom(n, size, shape1, shape2, zeroprob = 0.5)
```

Arguments

x	integer vector of counts
size	number of trials (zero or more)
shape1, shape2	positive shape parameters of the mixing beta distribution
zeroprob	probability of a zero, between 0 and 1
log	logical; return log-density if TRUE
q	vector of quantiles.
lower.tail	logical; if TRUE (default), probabilities are $P[X \leq q]$, otherwise $P[X > q]$.
log.p	logical; if TRUE, probabilities are returned on the log scale.
n	number of random values to return.

Details

This implementation allows for automatic differentiation with RTMB.

A hurdle distribution models the zeros and the positive counts as two separate processes: the probability of a zero is a free parameter, and the positive counts follow the corresponding zero-truncated distribution. Writing p_0 for zeroprob,

$$P(X = 0) = p_0, \qquad P(X = x) = (1 - p_0)\,\frac{P_{\mathrm{BB}}(x;\, n, a, b)}{1 - \pi_0}, \quad x = 1, \ldots, n.$$

where $\pi_0 = P_{\mathrm{BB}}(0;\, n, a, b)$ is the probability of a zero under the ordinary beta-binomial.

Unlike zero-inflation, which can only add zeros, zeroprob here is exactly the probability of observing a zero and may be larger *or* smaller than the beta-binomial would give on its own.

The distribution function has no closed form and is computed by summing the probability mass function over $0, \ldots, q$. It is AD-compatible in the parameters, while q and size must be numeric data. This is also what one-step-ahead (OSA) residuals via RTMB::oneStepPredict need, so these are supported, e.g. with method = "cdf" and discrete = TRUE.

Value

dhbetabinom gives the probability mass function, phbetabinom gives the distribution function, and rhbetabinom generates random deviates.

See Also

betabinom, zibetabinom, ztbetabinom, hbinom

Examples

```
set.seed(123)
x <- rhbetabinom(5, size = 10, shape1 = 2, shape2 = 3, zeroprob = 0.4)
d <- dhbetabinom(x, size = 10, shape1 = 2, shape2 = 3, zeroprob = 0.4)
p <- phbetabinom(x, size = 10, shape1 = 2, shape2 = 3, zeroprob = 0.4)
```

---

hbinom *Hurdle binomial distribution*

Description

Probability mass function, distribution function, and random generation for the hurdle (zero-altered) binomial distribution.

Usage

```
dhbinom(x, size, prob, zeroprob = 0.5, log = FALSE)

phbinom(q, size, prob, zeroprob = 0.5, lower.tail = TRUE, log.p = FALSE)

rhbinom(n, size, prob, zeroprob = 0.5)
```

Arguments

x, q	integer vector of counts
size	number of trials (zero or more)
prob	probability of success on each trial
zeroprob	probability of a zero, 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.

A hurdle distribution models the zeros and the positive counts as two separate processes: the probability of a zero is a free parameter, and the positive counts follow the corresponding zero-truncated distribution. Writing p_0 for zeroprob,

$$P(X = 0) = p_0, \qquad P(X = x) = (1 - p_0)\,\frac{P_{\mathrm{Bin}}(x;\, n, \pi)}{1 - \pi_0}, \quad x = 1, \ldots, n.$$

where $\pi_0 = P_{\mathrm{Bin}}(0;\, n, \pi)$ is the probability of a zero under the ordinary binomial.

Unlike zero-inflation, which can only add zeros to those the binomial already produces, zeroprob here is exactly the probability of a zero and may be larger *or* smaller than π_0. The two coincide with the ordinary binomial when zeroprob equals π_0.

Value

dhbinom gives the probability mass function, phbinom gives the distribution function, and rhbinom generates random deviates.

References

Mullahy, J. (1986) Specification and testing of some modified count data models. Journal of Econometrics, 33, 341-365.

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

hpois, hnbinom, zibinom, ztbinom

Examples

```
set.seed(123)
x <- rhbinom(5, size = 10, prob = 0.3, zeroprob = 0.4)
d <- dhbinom(x, size = 10, prob = 0.3, zeroprob = 0.4)
p <- phbinom(x, size = 10, prob = 0.3, zeroprob = 0.4)
```

hgeom *Hurdle geometric distribution*

Description

Probability mass function, distribution function, and random generation for the hurdle (zero-altered) geometric distribution.

Usage

```
dhgeom(x, prob, zeroprob = 0.5, log = FALSE)

phgeom(q, prob, zeroprob = 0.5, lower.tail = TRUE, log.p = FALSE)

rhgeom(n, prob, zeroprob = 0.5)
```

Arguments

x, q	integer vector of counts
prob	probability of success in each trial, in (0,1)
zeroprob	probability of a zero, 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.

A hurdle distribution models the zeros and the positive counts as two separate processes: the probability of a zero is a free parameter, and the positive counts follow the corresponding zero-truncated distribution. Writing p_0 for zeroprob,

$$P(X = 0) = p_0, \qquad P(X = x) = (1 - p_0)\,\frac{P_{\mathrm{Geom}}(x;\,\pi)}{1 - \pi_0}, \quad x = 1, 2, \ldots$$

where $\pi_0 = P_{\mathrm{Geom}}(0;\,\pi) = \pi$ is the probability of a zero under the ordinary geometric.

Unlike zero-inflation, which can only add zeros to those the geometric already produces, zeroprob here is exactly the probability of a zero and may be larger *or* smaller than π_0. The two coincide with the ordinary geometric when zeroprob equals π_0.

Value

dhgeom gives the probability mass function, phgeom gives the distribution function, and rhgeom generates random deviates.

References

Mullahy, J. (1986) Specification and testing of some modified count data models. Journal of Econometrics, 33, 341-365.

See Also

zigeom, ztgeom, hnbinom, hpois

Examples

```
set.seed(123)
x <- rhgeom(5, prob = 0.3, zeroprob = 0.4)
d <- dhgeom(x, prob = 0.3, zeroprob = 0.4)
p <- phgeom(x, prob = 0.3, zeroprob = 0.4)
```

---

hnbinom *Hurdle negative binomial distribution*

Description

Probability mass function, distribution function, and random generation for the hurdle (zero-altered) negative binomial distribution.

Usage

```
dhnbinom(x, size, prob, zeroprob = 0.5, log = FALSE)

phnbinom(q, size, prob, zeroprob = 0.5, lower.tail = TRUE, log.p = FALSE)

rhnbinom(n, size, prob, zeroprob = 0.5)
```

Arguments

x, q	integer vector of counts
size	dispersion parameter, must be strictly positive
prob	probability of success in each trial, in (0,1)
zeroprob	probability of a zero, 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.

A hurdle distribution models the zeros and the positive counts as two separate processes: the probability of a zero is a free parameter, and the positive counts follow the corresponding zero-truncated distribution. Writing p_0 for zeroprob,

$$P(X = 0) = p_0, \qquad P(X = x) = (1 - p_0)\,\frac{P_{\mathrm{NB}}(x;\, r, \pi)}{1 - \pi_0}, \quad x = 1, 2, \ldots$$

where $\pi_0 = P_{\mathrm{NB}}(0;\, r, \pi)$ is the probability of a zero under the ordinary negative binomial.

Unlike zero-inflation, which can only add zeros to those the negative binomial already produces, zeroprob here is exactly the probability of a zero and may be larger *or* smaller than π_0. The two coincide with the ordinary negative binomial when zeroprob equals π_0.

Value

dhnbinom gives the probability mass function, phnbinom gives the distribution function, and rhnbinom generates random deviates.

References

Mullahy, J. (1986) Specification and testing of some modified count data models. Journal of Econometrics, 33, 341-365.

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

hnbinom2, hpois, hbinom, zinbinom, ztnbinom

Examples

```
set.seed(123)
x <- rhnbinom(5, size = 2, prob = 0.4, zeroprob = 0.3)
d <- dhnbinom(x, size = 2, prob = 0.4, zeroprob = 0.3)
p <- phnbinom(x, size = 2, prob = 0.4, zeroprob = 0.3)
```

---

hnbinom2 *Reparameterised hurdle negative binomial distribution*

Description

Probability mass function, distribution function, and random generation for the hurdle (zero-altered) negative binomial distribution, parameterised by the mean of the untruncated negative binomial.

Usage

```
dhnbinom2(x, mu, size, zeroprob = 0.5, log = FALSE)

phnbinom2(q, mu, size, zeroprob = 0.5, lower.tail = TRUE, log.p = FALSE)

rhnbinom2(n, mu, size, zeroprob = 0.5)
```

Arguments

x, q	integer vector of counts
mu	mean of the untruncated negative binomial, must be strictly positive
size	dispersion parameter, must be strictly positive
zeroprob	probability of a zero, 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.

This is hnbinom with the success probability replaced by

$$\pi = \frac{\text{size}}{\text{size} + \mu},$$

so that μ is the mean of the *untruncated* negative binomial, whose variance is $\mu + \mu^2/\text{size}$. Note that μ is not the mean of the hurdle distribution itself, which also depends on zeroprob.

As for all hurdle distributions, zeroprob is exactly the probability of observing a zero and may be larger *or* smaller than the negative binomial would give on its own.

Value

dhnbinom2 gives the probability mass function, phnbinom2 gives the distribution function, and rhnbinom2 generates random deviates.

References

Mullahy, J. (1986) Specification and testing of some modified count data models. Journal of Econometrics, 33, 341-365.

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

hnbinom, hpois, nbinom2, zinbinom2, ztnbinom2

Examples

```
set.seed(123)
x <- rhnbinom2(5, mu = 3, size = 2, zeroprob = 0.3)
d <- dhnbinom2(x, mu = 3, size = 2, zeroprob = 0.3)
p <- phnbinom2(x, mu = 3, size = 2, zeroprob = 0.3)
```

---

hpois *Hurdle Poisson distribution*

Description

Probability mass function, distribution function, and random generation for the hurdle (zero-altered) Poisson distribution.

Usage

```
dhpois(x, lambda, zeroprob = 0.5, log = FALSE)

phpois(q, lambda, zeroprob = 0.5, lower.tail = TRUE, log.p = FALSE)

rhpois(n, lambda, zeroprob = 0.5)
```

Arguments

x, q	integer vector of counts
lambda	vector of (non-negative) means of the underlying Poisson
zeroprob	probability of a zero, 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.

A hurdle distribution models the zeros and the positive counts as two separate processes: the probability of a zero is a free parameter, and the positive counts follow the corresponding zero-truncated distribution. Writing p_0 for zeroprob,

$$P(X = 0) = p_0, \qquad P(X = x) = (1 - p_0)\,\frac{P_{\text{Pois}}(x;\,\lambda)}{1 - \pi_0}, \quad x = 1, 2, \ldots$$

where $\pi_0 = P_{\text{Pois}}(0;\,\lambda)$ is the probability of a zero under the ordinary Poisson.

This differs from the zero-inflated Poisson, where the zeros are a mixture of structural zeros and zeros generated by the Poisson itself. In a hurdle model zeroprob is exactly the probability of observing a zero, so it can be larger *or* smaller than the Poisson would give on its own; zero-inflation can only ever add zeros. The two coincide with the ordinary Poisson when zeroprob equals π_0.

Value

dhpois gives the probability mass function, phpois gives the distribution function, and rhpois generates random deviates.

References

Mullahy, J. (1986) Specification and testing of some modified count data models. Journal of Econometrics, 33, 341-365.

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

hbinom, hnbinom, hnbinom2, zipois, ztpois

Examples

```
set.seed(123)
x <- rhpois(5, lambda = 2, zeroprob = 0.4)
d <- dhpois(x, lambda = 2, zeroprob = 0.4)
p <- phpois(x, lambda = 2, zeroprob = 0.4)
```

---

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 $X \sim \text{Chisq}(\nu)$, then $1/X \sim \text{invChisq}(\nu)$.

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. Give either rate or scale, not both.
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 $r = \nu + \tau\,\mathrm{asinh}(z)$. 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 `obj$env$last.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 $Y \sim \text{Logistic}(\log \alpha,\, 1/\beta)$, which yields the numerically convenient log-density

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

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 REPORTed 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,
  par = 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 precision matrix, dense or sparse. For models with random effects, this is typically the sparse matrix returned by sdreport(..., getJointPrecision = TRUE)$jointPrecision. If not provided, it is computed internally, using sdreport for models with random effects and the Hessian obj$he() otherwise. Q may have more rows than there are model parameters, see Details.
par	Optional mean of the samples. Defaults to objenvlast.par.best. Must have length nrow(Q) if Q is supplied.
...	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.

A user-supplied `Q` may be the joint precision of the model parameters and additional hyperparameters, such as log smoothing parameters, whose coordinates come after the model parameters. The mean `par` then needs to be given for all coordinates. Samples are drawn in the full space and the hyperparameters are dropped, so they are integrated out and the samples of the model parameters account for their uncertainty.

Value

A list structured like the original parameter list used in the `MakeADFun` call (potentially including additional `REPORT`ed 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

gpd, 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)
```

---

pig *Poisson-inverse Gaussian distribution*

Description

Probability mass function, distribution function and random generation for the Poisson-inverse Gaussian distribution.

Usage

```
dpig(x, mu, sigma, log = FALSE)

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

rpig(n, mu, sigma)
```

Arguments

x	vector of non-negative counts.
mu	mean parameter, must be positive.
sigma	dispersion parameter, must be positive.
log	logical; if TRUE, probabilities are returned on the log scale.
q	vector of quantiles.
lower.tail	logical; if TRUE (default), probabilities are $P[X \leq q]$, otherwise $P[X > q]$.
log.p	logical; if TRUE, probabilities are returned on the log scale.
n	number of random values to return (for rpig).

Details

This implementation allows for automatic differentiation with RTMB.

The Poisson-inverse Gaussian is a mixed Poisson distribution whose mean is drawn from an inverse Gaussian distribution with mean μ and variance $\sigma\mu^2$. Its probability mass function is

$$P(X = x) = \left(\frac{2\alpha}{\pi}\right)^{1/2} \frac{\mu^x e^{1/\sigma} K_{x-1/2}(\alpha)}{(\alpha\sigma)^x \, x!}, \qquad \alpha^2 = \frac{1}{\sigma^2} + \frac{2\mu}{\sigma}, \quad x = 0, 1, 2, \ldots$$

where K_ν is the modified Bessel function of the second kind. The parameterisation follows the `PIG` family of the `gamlss.dist` package, with

$$E(X) = \mu, \qquad \mathrm{Var}(X) = \mu + \sigma\mu^2.$$

This is the same mean-variance relationship as the negative binomial with `size = 1 / sigma`, but the Poisson-inverse Gaussian has a heavier right tail and puts more mass at zero for the same mean and variance. It approaches the Poisson as $\sigma \to 0$.

It is the Sichel distribution with $\nu = -1/2$, and is computed as such; see there for how the Bessel functions are evaluated. The distribution function is summed from the probability mass function, so one-step-ahead (OSA) residuals via `RTMB::oneStepPredict` are supported, e.g. with `method = "cdf"` and `discrete = TRUE`.

Value

`dpig` gives the probability mass function, `ppig` gives the distribution function, and `rpig` generates random deviates.

References

Dean, C., Lawless, J. F., and Willmot, G. E. (1989) A mixed Poisson-inverse-Gaussian regression model. Canadian Journal of Statistics, 17, 171-181.

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

sichel, nbinom2, invgauss

Examples

```
set.seed(123)
x <- rpig(5, mu = 4, sigma = 0.5)
d <- dpig(x, mu = 4, sigma = 0.5)
p <- ppig(x, mu = 4, sigma = 0.5)
```

---

powerexp *Power Exponential distribution (PE and PE2)*

Description

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

Usage

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

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

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

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

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

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

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

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

Arguments

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

Details

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))
```

`sichel` *Sichel distribution*

Description

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

Usage

```
dsichel(x, mu, sigma, nu, log = FALSE)

psichel(q, mu, sigma, nu, lower.tail = TRUE, log.p = FALSE)

rsichel(n, mu, sigma, nu)
```

Arguments

`x`	vector of non-negative counts.
`mu`	mean parameter, must be positive.
`sigma`	dispersion parameter, must be positive.
`nu`	shape parameter, any real number.
`log`	logical; if `TRUE`, probabilities are returned on the log scale.
`q`	vector of quantiles.
`lower.tail`	logical; if `TRUE` (default), probabilities are $P[X \leq q]$, otherwise $P[X > q]$.
`log.p`	logical; if `TRUE`, probabilities are returned on the log scale.
`n`	number of random values to return (for `rsichel`).

Details

This implementation allows for automatic differentiation with `RTMB`.

The Sichel distribution is a mixed Poisson distribution whose mean follows a generalised inverse Gaussian distribution. In the parameterisation of the `SICHEL` family of the `gamlss.dist` package, μ is the mean and the probability mass function is

$$P(X = x) = \frac{(\mu/c)^x \, K_{x+\nu}(\alpha)}{x! \, (\alpha\sigma)^{x+\nu} K_\nu(1/\sigma)}, \qquad \alpha^2 = \frac{1}{\sigma^2} + \frac{2\mu}{c\,\sigma}, \quad c = \frac{K_{\nu+1}(1/\sigma)}{K_\nu(1/\sigma)},$$

for $x = 0, 1, 2, \ldots$, where K_ν is the modified Bessel function of the second kind. The variance is

$$\mathrm{Var}(X) = \mu + \mu^2 \left(\frac{2\sigma(\nu+1)}{c} + \frac{1}{c^2} - 1 \right).$$

For $\nu = -1/2$ this is the Poisson-inverse Gaussian distribution, and as $\sigma \to \infty$ with $\nu > 0$ it approaches the negative binomial.

The Bessel functions are evaluated with `RTMB`'s `besselK` for orders up to 50. For larger orders, `besselK` overflows when its argument is small, so its uniform asymptotic expansion for large orders is used instead, which is accurate to about 10^{-10} on the log scale there. Both are differentiable in the order, so ν can be estimated.

The distribution function has no closed form and is computed by summing the probability mass function over $0, \ldots, q$, so its cost grows with the largest `q`. It is AD-compatible in the parameters, while `q` must be numeric data. This is also what one-step-ahead (OSA) residuals via `RTMB::oneStepPredict` need, so these are supported, e.g. with `method = "cdf"` and `discrete = TRUE`.

Value

`dsichel` gives the probability mass function, `psichel` gives the distribution function, and `rsichel` generates random deviates.

References

Sichel, H. S. (1982) Repeat-buying and the generalized inverse Gaussian-Poisson distribution. Journal of the Royal Statistical Society, Series C, 31, 193-204.

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/.

Olver, F. W. J. et al. (eds.) NIST Digital Library of Mathematical Functions, equation 10.41.4, https://dlmf.nist.gov/10.41.

See Also

pig, nbinom2, bnbinom2

Examples

```
set.seed(123)
x <- rsichel(5, mu = 4, sigma = 0.5, nu = 1)
d <- dsichel(x, mu = 4, sigma = 0.5, nu = 1)
p <- psichel(x, mu = 4, sigma = 0.5, nu = 1)
```

`skellam` *Skellam distribution*

Description

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

Usage

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

pskellam(q, mu1, mu2, lower.tail = TRUE, log.p = FALSE)

rskellam(n, mu1, mu2)
```

Arguments

`x`	integer vector of counts
`mu1`, `mu2`	Poisson means
`log`	logical; return log-density if TRUE
`q`	vector of quantiles
`lower.tail`	logical; if `TRUE` (default), probabilities are $P[X \leq q]$, otherwise $P[X > q]$.
`log.p`	logical; if `TRUE`, probabilities are returned on the log scale.
`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`.

The distribution function has no closed form, and as the support is unbounded in both directions, summing the probability mass function would need a range that depends on the parameters. Instead, `pskellam` uses the identity

$$P(X \leq q;\, \mu_1, \mu_2) = \int_{\mu_1}^{\infty} P(X = q;\, t, \mu_2)\, dt,$$

which follows from the relation between the Poisson and gamma distribution functions, and integrates it numerically with the AD-compatible `integrate` of `RTMB`. By symmetry, the upper tail is $P(X > q;\, \mu_1, \mu_2) = P(X \leq -q - 1;\, \mu_2, \mu_1)$, and whichever tail is smaller is integrated directly, which keeps small tail probabilities accurate. `pskellam` is AD-compatible in `mu1` and `mu2`, while `q` must be numeric data. This is also what one-step-ahead (OSA) residuals via `RTMB::oneStepPredict` need, so these are supported, e.g. with `method = "cdf"` and `discrete = TRUE`.

Value

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

Examples

```
x <- rskellam(1, 2, 3)
d <- dskellam(x, 2, 3)
p <- pskellam(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. When called during AD taping, pskewnorm instead integrates the density numerically with the AD-compatible integrate of RTMB, which makes it AD-compatible as well. 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. When called during AD taping, pskewnorm2 instead integrates the density numerically with the AD-compatible integrate of RTMB, which makes it AD-compatible as well.

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.

When called during AD taping, pskewt instead integrates the density numerically with the AD-compatible integrate of RTMB, which makes it AD-compatible as well. The method argument is then ignored. See sn::dst for more details.

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

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 dskewt2 allows for automatic differentiation with RTMB while the other functions are imported from the sn package. When called during AD taping, pskewt2 instead integrates the density numerically with the AD-compatible integrate of RTMB, which makes it AD-compatible as well. The method argument is then ignored. See sn::dst for more details.

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.ad(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.

`pt.ad` is an AD-compatible replacement for `stats::pt`. It is used internally and is reached automatically whenever an argument is an AD variable, so `stats::pt` is left untouched for ordinary use.

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, ignored in AD context.
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. When called during AD taping, pvm instead integrates the density numerically with the AD-compatible integrate of RTMB, which makes it AD-compatible in q, mu, kappa and from. The tol argument is then ignored.

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

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

A circular distribution has no smallest angle, so its distribution function depends on where the circle is cut open. By default, pvm cuts it at the antipode of the mean direction, $\mu - \pi$, so that $F(\mu) = 1/2$. A different origin is set with from. A fixed from is needed whenever the distribution function is averaged over different values of mu, for example over the states of a hidden Markov model or over a random effect: with the default, each value of mu cuts the circle at a different place, and the average of these distribution functions is not the distribution function of the mixture.

OSA residuals: dvm supports one-step-ahead (OSA) quantile residuals via RTMB::oneStepPredict. For the methods based on the distribution function, such as method = "cdf", the circle is cut at the fixed origin $-\pi$, i.e. the residuals are based on pvm(x, mu, kappa, from = -pi) rather than the default origin $\mu - \pi$. OSA residuals are computed from the predictive distribution function, which averages the distribution function over hidden states or random effects, and this is only valid with an origin that does not depend on mu (see above). Hence the residuals are valid for all models, but their interpretation depends on the data: for turning angles, with mu close to 0, the cut at $\pm\pi$ corresponds to a reversal and the residuals increase with the turning angle. For directions with mu far from 0,

angles close to $\pm\pi$ can give large residuals of either sign, even when they are close to the mean direction. For `method = "oneStepGeneric"`, set `range = c(-pi, pi)` in `oneStepPredict()`, so that the density is integrated from the same origin.

Value

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

See Also

wrpcauchy; `cjw()` and `cfold()` for circular-linear copulas joining turning angles and step lengths.

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)
```

---

waring *Waring distribution*

Description

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

Usage

```
dwaring(x, mu = 2, sigma = 2, log = FALSE)

pwaring(q, mu = 2, sigma = 2, lower.tail = TRUE, log.p = FALSE)

rwaring(n, mu = 2, sigma = 2)
```

Arguments

x, q	vector of non-negative counts.
mu	mean parameter, must be positive.
sigma	dispersion parameter, must be positive. The variance is finite only for sigma < 1.
log, log.p	logical; if TRUE, probabilities 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 (for rwaring).

Details

`dwaring` and `pwaring` allow for automatic differentiation with `RTMB`. The parameterisation follows the `WARING` family of the `gamlss.dist` package, in which μ is exactly the mean.

$$P(X = k;\, \mu, \sigma) = \frac{B\left(k + \frac{\mu}{\sigma},\ \frac{1}{\sigma} + 2\right)}{B\left(\frac{\mu}{\sigma},\ \frac{1}{\sigma} + 1\right)}, \quad k = 0, 1, 2, \ldots$$

The Waring is the beta-geometric: a geometric distribution whose success probability carries a beta prior. It is the two-parameter long-tailed count law behind accident proneness and repeat-buying models, and generalises the Yule-Simon distribution, which is the case $\sigma = \mu$ shifted to start at one. The variance

$$\mathrm{Var}(X) = \frac{\mu(\sigma + 1)(\mu + 1)}{1 - \sigma}$$

is finite only for $\sigma < 1$, and the tail is a power law throughout.

It is exactly the mean-parameterised beta-negative binomial with `nu = 1`. Fixing `size` at one is what gives it a closed-form distribution function, which the beta-negative binomial does not have in general, so one-step-ahead residuals are available here but not there.

Value

`dwaring` gives the probability mass function, `pwaring` gives the distribution function, and `rwaring` generates random deviates.

References

Irwin, J. O. (1963) The place of mathematics in medical and biological statistics. Journal of the Royal Statistical Society A, 126, 1-45, doi:10.2307/2982445.

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

yules, bnbinom2, bnbinom

Examples

```
set.seed(123)
x <- rwaring(5, mu = 2, sigma = 0.5)
d <- dwaring(x, mu = 2, sigma = 0.5)
p <- pwaring(x, mu = 2, sigma = 0.5)
```

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, distribution function, quantile function, and random generation for the wrapped Cauchy distribution.

Usage

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

pwrpcauchy(q, mu = 0, rho, from = NULL, lower.tail = TRUE, log.p = FALSE)

qwrpcauchy(p, mu = 0, rho, from = NULL, lower.tail = TRUE, log.p = FALSE)

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

Arguments

x, q	vector of angles measured in radians at which to evaluate the density or distribution 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, log.p	logical; if TRUE, probabilities/ densities p are returned as $\log(p)$.
from	origin, in radians, at which the circle is cut open for the distribution and quantile functions. If NULL (default), it is set to mu - pi.
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.
wrap	logical; if TRUE, generated angles are wrapped to the interval from -pi to pi, otherwise they lie in the interval from mu - pi to mu + pi.

Details

dwrpcauchy and pwrpcauchy allow for automatic differentiation with RTMB.

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

A circular distribution has no smallest angle, so its distribution function depends on where the circle is cut open. pwrpcauchy cuts it at the antipode of the mean direction, $\mu - \pi$, and is then available in closed form:

$$F(q) = P(\mu - \pi < X \le q) = \frac{1}{2} + \frac{1}{\pi} \arctan\left(\frac{1+\rho}{1-\rho} \tan\frac{q-\mu}{2}\right), \quad \mu - \pi < q \le \mu + \pi,$$

so that $F(\mu) = 1/2$. This is the default, from = NULL, and the same origin as the default of pvm.

Angles outside $(\mu-\pi, \mu+\pi]$ are wrapped onto this interval first, so pwrpcauchy is 2π-periodic in q. As a consequence, the default origin moves with mu: for angles on $[-\pi, \pi]$ and $\mu \neq 0$, pwrpcauchy is not monotone over that range but drops from 1 back to 0 at $\mu \pm \pi$.

A different origin is set with from, giving $P(\text{from} < X \le q) = (F(q) - F(\text{from})) \bmod 1$ for $\text{from} < q \le \text{from} + 2\pi$. A fixed from is needed whenever the distribution function is averaged over different values of mu, for example over the states of a hidden Markov model or over a random effect: with the default, each value of mu cuts the circle at a different place, and the average of these distribution functions is not the distribution function of the mixture.

OSA residuals: dwrpcauchy supports one-step-ahead (OSA) quantile residuals via RTMB::oneStepPredict. For the methods based on the distribution function, such as method = "cdf", the circle is cut at the fixed origin $-\pi$, i.e. the residuals are based on pwrpcauchy(x, mu, rho, from = -pi) rather than the default origin $\mu - \pi$. OSA residuals are computed from the predictive distribution function, which averages the distribution function over hidden states or random effects, and this is only valid with an origin that does not depend on mu (see above). Hence the residuals are valid for all models, but their interpretation depends on the data: for turning angles, with mu close to 0, the cut at $\pm\pi$ corresponds to a reversal and the residuals increase with the turning angle. For directions with mu far from 0, angles close to $\pm\pi$ can give large residuals of either sign, even when they are close to the mean direction. For method = "oneStepGeneric", set range = c(-pi, pi) in oneStepPredict(), so that the density is integrated from the same origin.

qwrpcauchy is the inverse of pwrpcauchy and returns angles in $[\text{from}, \text{from} + 2\pi]$, by default $[\mu - \pi, \mu + \pi]$, not wrapped to $[-\pi, \pi]$. The latter also holds for rwrpcauchy with wrap = FALSE.

Value

dwrpcauchy gives the density, pwrpcauchy gives the distribution function, qwrpcauchy gives the quantile function, and rwrpcauchy generates random deviates.

See Also

vm; cjw() and cfold() for circular-linear copulas joining turning angles and step lengths.

Examples

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

wrpnorm *Wrapped normal distribution*

Description

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

Usage

```
dwrpnorm(x, mu = 0, sigma, log = FALSE)

pwrpnorm(q, mu = 0, sigma, from = NULL, lower.tail = TRUE, log.p = FALSE)

rwrpnorm(n, mu = 0, sigma)
```

Arguments

x, q	vector of angles in radians.
mu	mean direction in radians.
sigma	standard deviation of the normal distribution before wrapping, must be positive.
log, log.p	logical; if TRUE, probabilities/ densities p are returned as $\log(p)$.
from	origin, in radians, at which the circle is cut open for the distribution function. If NULL (default), it is set to mu - pi.
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.

The wrapped normal is the distribution of $Y \bmod 2\pi$ for a normal Y with mean μ and standard deviation σ, with density

$$f(x;\,\mu,\sigma) = \sum_{k=-\infty}^{\infty} \frac{1}{\sigma}\,\phi\!\left(\frac{x-\mu+2\pi k}{\sigma}\right) = \frac{1}{2\pi}\left(1 + 2\sum_{k=1}^{\infty} e^{-k^2\sigma^2/2}\cos(k(x-\mu))\right).$$

For $\sigma < 2$ the first sum is used with $|k| \leq 3$, otherwise the second with $k \leq 4$; the terms left out are below 10^{-20}.

The distribution function cuts the circle open at $\mu - \pi$ by default, or at from, exactly as pwrpcauchy does, where the choice of origin and the one-step-ahead (OSA) residuals are explained. As there, OSA residuals via RTMB::oneStepPredict with method = "cdf" cut the circle at the fixed origin $-\pi$.

Value

dwrpnorm gives the density, pwrpnorm gives the distribution function, and rwrpnorm generates random deviates in $[-\pi, \pi)$.

See Also

wrpcauchy, vm, card; `cjw()` for circular-linear copulas.

Examples

```
set.seed(1)
x <- rwrpnorm(10, 0, 0.8)
d <- dwrpnorm(x, 0, 0.8)
p <- pwrpnorm(x, 0, 0.8)
```

---

yules *Yule-Simon distribution*

Description

Probability mass function, distribution function, and random generation for the Yule-Simon distribution.

Usage

```
dyules(x, shape = 1, log = FALSE)

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

ryules(n, shape = 1)
```

Arguments

`x`, `q`	vector of quantiles.
`shape`	positive shape parameter ρ.
`log`, `log.p`	logical; if `TRUE`, probabilities are returned as $\log(p)$.
`lower.tail`	logical; if `TRUE` (default), probabilities are $P[X \le x]$, otherwise $P[X > x]$.
`n`	number of random values to return (for `ryules`).

Details

`dyules` and `pyules` allow for automatic differentiation with `RTMB`.

$$P(X = k;\, \rho) = \rho\, B(k,\, \rho + 1), \quad k = 1, 2, \ldots$$

The Yule-Simon distribution is the classical long-tailed frequency law, used for word counts, city sizes, citation counts and species-per-genus data. Its tail is a power law, $P(X = k) \sim \rho\,\Gamma(\rho + 1)k^{-(\rho+1)}$, so the mean $\rho/(\rho-1)$ exists only for $\rho > 1$ and the variance $\rho^2/\{(\rho-1)^2(\rho-2)\}$ only for $\rho > 2$.

It is the beta-negative binomial with `size = 1` and `shape2 = 1`, shifted to start at one, and equivalently the Waring distribution with `sigma = mu`, shifted the same way. That special case is what gives it a closed-form distribution function, which the beta-negative binomial does not have in general.

The support here starts at one, as in `VGAM`. The `YULE` family of `gamlss.dist` instead starts at zero and is parameterised by its mean μ, which corresponds to $\rho = (\mu+1)/\mu$; use `dwaring(x, mu, mu)` for that version.

Value

`dyules` gives the probability mass function, `pyules` gives the distribution function, and `ryules` generates random deviates.

References

Simon, H. A. (1955) On a class of skew distribution functions. Biometrika, 42, 425-440, doi:10.1093/biomet/42.3-4.425.

See Also

waring, bnbinom, bnbinom2

Examples

```
set.seed(123)
x <- ryules(5, shape = 2)
d <- dyules(x, shape = 2)
p <- pyules(x, shape = 2)
```

---

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.

Because the continuous part places no mass at zero, zeroprob is exactly $P(X = 0)$, and zero-inflation coincides with a hurdle model here. The two constructions only differ for discrete distributions, where the base distribution can generate zeros of its own. GAMLSS therefore calls distributions of this type zero-adjusted rather than zero-inflated.

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 $b = (1-\mu)\phi$:

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

Because the continuous part places no mass at zero, zeroprob is exactly $P(X = 0)$, and zero-inflation coincides with a hurdle model here. The two constructions only differ for discrete distributions, where the base distribution can generate zeros of its own. GAMLSS therefore calls distributions of this type zero-adjusted rather than zero-inflated.

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)
```

---

zibetabinom *Zero-inflated beta-binomial distribution*

Description

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

Usage

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

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

rzibetabinom(n, size, shape1, shape2, zeroprob = 0)
```

Arguments

x	integer vector of counts
size	number of trials (zero or more)
shape1, shape2	positive shape parameters of the mixing beta distribution
zeroprob	zero-inflation probability between 0 and 1
log	logical; return log-density if TRUE
q	vector of quantiles.
lower.tail	logical; if TRUE (default), probabilities are $P[X \leq q]$, otherwise $P[X > q]$.
log.p	logical; if TRUE, probabilities are returned on the log scale.
n	number of random values to return.

Details

This implementation allows for automatic differentiation with RTMB.

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

where p_0 is zeroprob. The zeros are a mixture of structural zeros and zeros generated by the beta-binomial itself, so zeroprob is not the probability of observing a zero; see hbetabinom for the hurdle version, where it is.

The distribution function has no closed form and is computed by summing the probability mass function over $0, \ldots, q$. It is AD-compatible in the parameters, while q and size must be numeric data. This is also what one-step-ahead (OSA) residuals via RTMB::oneStepPredict need, so these are supported, e.g. with method = "cdf" and discrete = TRUE.

Value

dzibetabinom gives the probability mass function, pzibetabinom gives the distribution function, and rzibetabinom generates random deviates.

See Also

betabinom, ztbetabinom, hbetabinom, zibinom

Examples

```
set.seed(123)
x <- rzibetabinom(5, size = 10, shape1 = 2, shape2 = 3, zeroprob = 0.3)
d <- dzibetabinom(x, size = 10, shape1 = 2, shape2 = 3, zeroprob = 0.3)
p <- pzibetabinom(x, size = 10, shape1 = 2, shape2 = 3, zeroprob = 0.3)
```

---

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.

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

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`.

Because the continuous part places no mass at zero, `zeroprob` is exactly $P(X = 0)$, and zero-inflation coincides with a hurdle model here. The two constructions only differ for discrete distributions, where the base distribution can generate zeros of its own. GAMLSS therefore calls distributions of this type zero-adjusted rather than zero-inflated, as in its `ZAGA` family.

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 $= s^2/\mu$:

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

Because the continuous part places no mass at zero, zeroprob is exactly $P(X = 0)$, and zero-inflation coincides with a hurdle model here. The two constructions only differ for discrete distributions, where the base distribution can generate zeros of its own. GAMLSS therefore calls distributions of this type zero-adjusted rather than zero-inflated, as in its ZAGA family.

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)
```

---

zigeom *Zero-inflated geometric distribution*

Description

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

Usage

```
dzigeom(x, prob, zeroprob = 0, log = FALSE)

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

rzigeom(n, prob, zeroprob = 0)
```

Arguments

x, q	integer vector of counts
prob	probability of success in each trial, in (0,1]
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.

$$P(X = k;\, \pi, p_0) = p_0\, \mathbf{1}[k = 0] + (1 - p_0)\, P_{\text{Geom}}(k;\, \pi),$$

where p_0 is zeroprob. The zeros are a mixture of structural zeros and zeros generated by the geometric itself, so zeroprob is not the probability of observing a zero; see hgeom for the hurdle version, where it is.

Value

dzigeom gives the probability mass function, pzigeom gives the distribution function, and rzigeom generates random deviates.

See Also

ztgeom, hgeom, zinbinom, zipois

Examples

```
set.seed(123)
x <- rzigeom(5, prob = 0.3, zeroprob = 0.4)
d <- dzigeom(x, prob = 0.3, zeroprob = 0.4)
p <- pzigeom(x, prob = 0.3, zeroprob = 0.4)
```

---

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.

Because the continuous part places no mass at zero, `zeroprob` is exactly $P(X=0)$, and zero-inflation coincides with a hurdle model here. The two constructions only differ for discrete distributions, where the base distribution can generate zeros of its own. GAMLSS therefore calls distributions of this type zero-adjusted rather than zero-inflated, as in its `ZAIG` family.

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.ad(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.

Because the continuous part places no mass at zero, zeroprob is exactly $P(X = 0)$, and zero-inflation coincides with a hurdle model here. The two constructions only differ for discrete distributions, where the base distribution can generate zeros of its own. GAMLSS therefore calls distributions of this type zero-adjusted rather than zero-inflated.

Value

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

plnorm.ad is an AD-compatible replacement for stats::plnorm. It is used internally and is reached automatically whenever an argument is an AD variable, so stats::plnorm is left untouched for ordinary use.

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.

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

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.

Because the continuous part places no mass at zero, zeroprob is exactly $P(X = 0)$, and zero-inflation coincides with a hurdle model here. The two constructions only differ for discrete distributions, where the base distribution can generate zeros of its own. GAMLSS therefore calls distributions of this type zero-adjusted rather than zero-inflated.

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)
```

---

ztbetabinom *Zero-truncated beta-binomial distribution*

Description

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

Usage

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

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

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

Arguments

x	integer vector of counts
size	number of trials (zero or more)
shape1, shape2	positive shape parameters of the mixing beta distribution
log	logical; return log-density if TRUE
q	vector of quantiles.
lower.tail	logical; if TRUE (default), probabilities are $P[X \leq q]$, otherwise $P[X > q]$.
log.p	logical; if TRUE, probabilities are returned on the log scale.
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, ..., n). 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.

The distribution function has no closed form and is computed by summing the probability mass function over $0, \ldots, q$. It is AD-compatible in the parameters, while q and size must be numeric data. This is also what one-step-ahead (OSA) residuals via RTMB::oneStepPredict need, so these are supported, e.g. with method = "cdf" and discrete = TRUE.

Value

dztbetabinom gives the probability mass function, pztbetabinom gives the distribution function, and rztbetabinom generates random deviates.

See Also

betabinom, zibetabinom, hbetabinom, ztbinom

Examples

```
set.seed(123)
x <- rztbetabinom(5, size = 10, shape1 = 2, shape2 = 3)
d <- dztbetabinom(x, size = 10, shape1 = 2, shape2 = 3)
p <- pztbetabinom(x, size = 10, shape1 = 2, shape2 = 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)
```

ztgeom *Zero-truncated geometric distribution*

Description

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

Usage

```
dztgeom(x, prob, log = FALSE)

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

rztgeom(n, prob)
```

Arguments

x, q	integer vector of counts
prob	probability of success in each trial, in (0,1)
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. For the geometric with success probability π this gives

$$P(X = x|X > 0) = \pi\,(1 - \pi)^{x-1}, \quad x = 1, 2, \ldots$$

Value

dztgeom gives the probability mass function, pztgeom gives the distribution function, and rztgeom generates random deviates.

See Also

zigeom, hgeom, ztnbinom, ztpois

Examples

```
set.seed(123)
x <- rztgeom(5, prob = 0.3)
d <- dztgeom(x, prob = 0.3)
p <- pztgeom(x, 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

abs_smooth, 4, *74*
ADjoint, *73*

bccg, 5, *7*, *9*, *47*
bcpe, *6*, 6, *9*, *71*, *91*
bct, *6*, *7*, 7, *71*
bell, 9, *11*, *12*, *48*
bell2, *10*, 11
beta-binomial, *18*
beta-negative binomial, *116*
beta2, 12
betabinom, 13, *18*, *20*, *59*, *121*, *134*
betaprime, 14, *45*
bisa, 16
bnbinom, 17, *20*, *110*, *116*
bnbinom2, *18*, 19, *93*, *110*, *116*

card, 20, *115*
Cclayton, *34*
Cclayton *(*cclayton*)*, 22
cclayton, 22, *23*, *33*
cclayton(), *25*, *27*
cfold, 23, *28*, *33*
cfold(), *29*, *107*, *113*
Cfrank, *34*
Cfrank *(*cfrank*)*, 24
cfrank, *23*, 24, *33*
cfrank(), *22*, *25*, *27*
cgaussian, *23*, 25, *33*
cgaussian(), *22*, *25*, *27*
cgmrf, 26, *37*
cgmrf(), *30*
Cgumbel, *34*
Cgumbel *(*cgumbel*)*, 27
cgumbel, *23*, 27, *33*
cgumbel(), *22*, *25*
cjw, *23*, 28, *33*
cjw(), *24*, *107*, *113*, *115*
cmvgauss, 29, *37*
cmvgauss(), *26*

combinom, 30

dbccg *(*bccg*)*, 5
dbcpe *(*bcpe*)*, 6
dbct *(*bct*)*, 7
dbell *(*bell*)*, 9
dbell2 *(*bell2*)*, 11
dbeta *(*beta2*)*, 12
dbeta2 *(*beta2*)*, 12
dbetabinom *(*betabinom*)*, 13
dbetaprime, *45*
dbetaprime *(*betaprime*)*, 14
dbisa *(*bisa*)*, 16
dbnbinom, *19*
dbnbinom *(*bnbinom*)*, 17
dbnbinom2 *(*bnbinom2*)*, 19
dcard *(*card*)*, 20
dcombinom *(*combinom*)*, 30
dcopula, *22–25*, *27*, *28*, 32
dcopula(), *24*, *29*, *34*, *37*
ddagum *(*gb2*)*, 44
ddcopula, 34
ddcopula(), *33*, *37*
ddirichlet *(*dirichlet*)*, 35
ddirmult *(*dirmult*)*, 36
dexgauss *(*exgauss*)*, 39
dexp, *87*
dfoldnorm *(*foldnorm*)*, 40
dfrechet *(*frechet*)*, 41
dgamma2 *(*gamma2*)*, 43
dgb2 *(*gb2*)*, 44
dgengamma *(*gengamma*)*, 46
dgenpois *(*genpois*)*, 47
dgeom.ad *(*geom.ad*)*, 49
dgev *(*gev*)*, 50
dgompertz *(*gompertz*)*, 51
dgpd *(*gpd*)*, 52
dgumbel *(*gumbel*)*, 54
dhalfcauchy *(*halfcauchy*)*, 55
dhalft *(*halft*)*, 56

dhbetabinom *(*hbetabinom*)*, 58
dhbinom *(*hbinom*)*, 59
dhgeom *(*hgeom*)*, 61
dhnbinom *(*hnbinom*)*, 62
dhnbinom2 *(*hnbinom2*)*, 63
dhpois *(*hpois*)*, 65
dinvchisq *(*invchisq*)*, 66
dinvgamma *(*invgamma*)*, 67
dinvgauss *(*invgauss*)*, 68
dirichlet, 35
dirmult, 36
djsu *(*jsu*)*, 69
djsu2 *(*jsu*)*, 69
dkumar *(*kumar*)*, 72
dlaplace *(*laplace*)*, 74
dllogis, *45*
dllogis *(*llogis*)*, 76
dlomax *(*gb2*)*, 44
dmvcopula, *29*, 37
dmvcopula(), *33, 34*
dmvt *(*mvt*)*, 81
dnbinom2 *(*nbinom2*)*, 82
doibeta *(*oibeta*)*, 83
doibeta2 *(*oibeta2*)*, 84
dpareto *(*pareto*)*, 85
dpgweibull *(*pgweibull*)*, 86
dpig *(*pig*)*, 88
dpowerexp *(*powerexp*)*, 89
dpowerexp2 *(*powerexp*)*, 89
dsichel *(*sichel*)*, 92
dsinmad *(*gb2*)*, 44
dskellam *(*skellam*)*, 94
dskewnorm *(*skewnorm*)*, 95
dskewnorm2 *(*skewnorm2*)*, 96
dskewt *(*skewt*)*, 98
dskewt2 *(*skewt2*)*, 99
dsn, *96*
dst, *99, 100*
dt2 *(*t2*)*, 101
dtruncnorm *(*truncnorm*)*, 102
dtrunct *(*trunct*)*, 103
dtrunct2 *(*trunct2*)*, 104
dvm, *28*, *33*
dvm *(*vm*)*, 105
dvmf *(*vmf*)*, 107
dvmf2 *(*vmf2*)*, 108
dwaring *(*waring*)*, 109
dweibull, *87*
dwishart *(*wishart*)*, 111
dwrpcauchy, *28*, *33*
dwrpcauchy *(*wrpcauchy*)*, 112
dwrpnorm *(*wrpnorm*)*, 114
dyules *(*yules*)*, 115
dzibeta *(*zibeta*)*, 117
dzibeta2 *(*zibeta2*)*, 118
dzibetabinom *(*zibetabinom*)*, 119
dzibinom *(*zibinom*)*, 121
dzigamma *(*zigamma*)*, 122
dzigamma2 *(*zigamma2*)*, 123
dzigeom *(*zigeom*)*, 124
dziinvgauss *(*ziinvgauss*)*, 125
dzilnorm *(*zilnorm*)*, 126
dzinbinom *(*zinbinom*)*, 127
dzinbinom2 *(*zinbinom2*)*, 128
dzipois *(*zipois*)*, 129
dziweibull *(*ziweibull*)*, 130
dzoibeta *(*zoibeta*)*, 131
dzoibeta2 *(*zoibeta2*)*, 132
dztbetabinom *(*ztbetabinom*)*, 133
dztbinom *(*ztbinom*)*, 135
dztgeom *(*ztgeom*)*, 136
dztnbinom *(*ztnbinom*)*, 137
dztnbinom2 *(*ztnbinom2*)*, 138
dztpois *(*ztpois*)*, 139

erf, 38
erfc *(*erf*)*, 38
exgauss, 39

folded normal, *56, 57*
foldnorm, 40, *56*, *58*
frechet, 41, *51*, *54, 55*

gamma2, 43, *47*
gb2, 44
generalised extreme value, *42*
gengamma, *6*, 46, *86*
genpois, 47
geom.ad, 49
gev, *42*, 50, *54, 55*
gompertz, *17*, 51
gpd, *51*, 52, *55*, *86*
Gumbel, *50*
gumbel, *42*, *51*, 54

half-Cauchy, *57*
half-t, *41*, *56*

halfcauchy, *41*, 55, *58*
halft, *41*, *56*, 56
hbetabinom, 58, *120, 121*, *134*
hbinom, *59*, 59, *63*, *66*
hgeom, *49*, 61, *124, 125*, *136*
hnbinom, *60*, *62*, 62, *64*, *66*
hnbinom2, *63*, 63, *66*
hpgweibull *(*pgweibull*)*, 86
hpois, *60*, *62–64*, 65

integrate, *94*, *96, 97*, *99, 100*, *106*
invchisq, 66
invgamma, 67
invgauss, *17*, *47*, 68, *89*

jsu, *40*, 69, *91*

kumar, 72

lambertW, *11*, 73
laplace, 74, *91*
laplace_check, 75
llogis, *45*, 76, *86*

MakeADFun, *78, 79*
mcreport, 78
mean parameterisation, *18*
mean-parameterised beta-negative binomial, *110*
mvt, 81

nbinom2, *18*, *20*, *48*, *64*, 82, *89*, *93*
negative binomial, *19*, *89*, *93*

oibeta, 83
oibeta2, 84
oneStepPredict, *14*, *18*, *20, 21*, *45*, *59*, *89*, *93, 94*, *106*, *113, 114*, *120*, *134*

Pareto, *53*
pareto, *42*, *45*, *54*, 85
pbccg *(*bccg*)*, 5
pbcpe *(*bcpe*)*, 6
pbct *(*bct*)*, 7
pbell *(*bell*)*, 9
pbell2 *(*bell2*)*, 11
pbeta2 *(*beta2*)*, 12
pbetabinom *(*betabinom*)*, 13
pbetaprime *(*betaprime*)*, 14
pbisa *(*bisa*)*, 16
pbnbinom *(*bnbinom*)*, 17
pbnbinom2 *(*bnbinom2*)*, 19
pcard *(*card*)*, 20
pcombinom *(*combinom*)*, 30
pdagum *(*gb2*)*, 44
pexgauss *(*exgauss*)*, 39
pfoldnorm *(*foldnorm*)*, 40
pfrechet *(*frechet*)*, 41
pgamma2 *(*gamma2*)*, 43
pgb2 *(*gb2*)*, 44
pgengamma *(*gengamma*)*, 46
pgenpois *(*genpois*)*, 47
pgeom.ad *(*geom.ad*)*, 49
pgev *(*gev*)*, 50
pgompertz *(*gompertz*)*, 51
pgpd *(*gpd*)*, 52
pgumbel *(*gumbel*)*, 54
pgweibull, *17*, 86
phalfcauchy *(*halfcauchy*)*, 55
phalft *(*halft*)*, 56
phbetabinom *(*hbetabinom*)*, 58
phbinom *(*hbinom*)*, 59
phgeom *(*hgeom*)*, 61
phnbinom *(*hnbinom*)*, 62
phnbinom2 *(*hnbinom2*)*, 63
phpois *(*hpois*)*, 65
pig, 88, *93*
pinvchisq *(*invchisq*)*, 66
pinvgamma *(*invgamma*)*, 67
pinvgauss *(*invgauss*)*, 68
pjsu *(*jsu*)*, 69
pjsu2 *(*jsu*)*, 69
pkumar *(*kumar*)*, 72
plaplace *(*laplace*)*, 74
pllogis *(*llogis*)*, 76
plnorm.ad *(*zilnorm*)*, 126
plomax *(*gb2*)*, 44
pnbinom2 *(*nbinom2*)*, 82
poibeta *(*oibeta*)*, 83
poibeta2 *(*oibeta2*)*, 84
Poisson-inverse Gaussian, *93*
powerexp, *7*, 89
ppareto *(*pareto*)*, 85
ppgweibull *(*pgweibull*)*, 86
ppig *(*pig*)*, 88
ppowerexp *(*powerexp*)*, 89
ppowerexp2 *(*powerexp*)*, 89
print.laplace_check *(*laplace_check*)*, 75

psichel *(sichel)*, 92
psinmad *(gb2)*, 44
pskellam *(skellam)*, 94
pskewnorm *(skewnorm)*, 95
pskewnorm2 *(skewnorm2)*, 96
pskewt *(skewt)*, 98
pskewt2 *(skewt2)*, 99
pst, *98*, *100*
pt.ad *(t2)*, 101
pt2 *(t2)*, 101
ptruncnorm *(truncnorm)*, 102
ptrunct *(trunct)*, 103
ptrunct2 *(trunct2)*, 104
pvm, *23*, *113*
pvm *(vm)*, 105
pwaring *(waring)*, 109
pwrpcauchy, *21*, *23*, *28*, *114*
pwrpcauchy *(wrpcauchy)*, 112
pwrpnorm *(wrpnorm)*, 114
pyules *(yules)*, 115
pzibeta *(zibeta)*, 117
pzibeta2 *(zibeta2)*, 118
pzibetabinom *(zibetabinom)*, 119
pzibinom *(zibinom)*, 121
pzigamma *(zigamma)*, 122
pzigamma2 *(zigamma2)*, 123
pzigeom *(zigeom)*, 124
pziinvgauss *(ziinvgauss)*, 125
pzilnorm *(zilnorm)*, 126
pzinbinom *(zinbinom)*, 127
pzinbinom2 *(zinbinom2)*, 128
pzipois *(zipois)*, 129
pziweibull *(ziweibull)*, 130
pzoibeta *(zoibeta)*, 131
pzoibeta2 *(zoibeta2)*, 132
pztbetabinom *(ztbetabinom)*, 133
pztbinom *(ztbinom)*, 135
pztgeom *(ztgeom)*, 136
pztnbinom *(ztnbinom)*, 137
pztnbinom2 *(ztnbinom2)*, 138
pztpois *(ztpois)*, 139

qbccg *(bccg)*, 5
qbcpe *(bcpe)*, 6
qbct *(bct)*, 7
qbell *(bell)*, 9
qbell2 *(bell2)*, 11
qbeta2 *(beta2)*, 12
qbetaprime *(betaprime)*, 14
qbisa *(bisa)*, 16
qcombinom *(combinom)*, 30
qdagum *(gb2)*, 44
qexgauss *(exgauss)*, 39
qfrechet *(frechet)*, 41
qgamma2 *(gamma2)*, 43
qgb2 *(gb2)*, 44
qgengamma *(gengamma)*, 46
qgenpois *(genpois)*, 47
qgev *(gev)*, 50
qgompertz *(gompertz)*, 51
qgpd *(gpd)*, 52
qgumbel *(gumbel)*, 54
qhalfcauchy *(halfcauchy)*, 55
qhalft *(halft)*, 56
qinvchisq *(invchisq)*, 66
qinvgamma *(invgamma)*, 67
qinvgauss *(invgauss)*, 68
qjsu *(jsu)*, 69
qjsu2 *(jsu)*, 69
qkumar *(kumar)*, 72
qlaplace *(laplace)*, 74
qllogis *(llogis)*, 76
qlomax *(gb2)*, 44
qnbinom2 *(nbinom2)*, 82
qpareto *(pareto)*, 85
qpgweibull *(pgweibull)*, 86
qpowerexp *(powerexp)*, 89
qpowerexp2 *(powerexp)*, 89
qsinmad *(gb2)*, 44
qskewnorm *(skewnorm)*, 95
qskewnorm2 *(skewnorm2)*, 96
qskewt *(skewt)*, 98
qskewt2 *(skewt2)*, 99
qt2 *(t2)*, 101
qtruncnorm *(truncnorm)*, 102
qtrunct *(trunct)*, 103
qtrunct2 *(trunct2)*, 104
qwrpcauchy *(wrpcauchy)*, 112

rbccg *(bccg)*, 5
rbcpe *(bcpe)*, 6
rbct *(bct)*, 7
rbell *(bell)*, 9
rbell2 *(bell2)*, 11
rbeta2 *(beta2)*, 12
rbetabinom *(betabinom)*, 13
rbetaprime *(betaprime)*, 14
rbisa *(bisa)*, 16

rbnbinom (bnbinom), 17
rbnbinom2 (bnbinom2), 19
rcard (card), 20
rcombinom (combinom), 30
rdagum (gb2), 44
rdirichlet (dirichlet), 35
rdirmult (dirmult), 36
rexgauss (exgauss), 39
rfoldnorm (foldnorm), 40
rfrechet (frechet), 41
rgamma2 (gamma2), 43
rgb2 (gb2), 44
rgengamma (gengamma), 46
rgenpois (genpois), 47
rgev (gev), 50
rgmrf, 91
rgompertz (gompertz), 51
rgpd (gpd), 52
rgumbel (gumbel), 54
rhalfcauchy (halfcauchy), 55
rhalft (halft), 56
rhbetabinom (hbetabinom), 58
rhbinom (hbinom), 59
rhgeom (hgeom), 61
rhnbinom (hnbinom), 62
rhnbinom2 (hnbinom2), 63
rhpois (hpois), 65
rinvchisq (invchisq), 66
rinvgamma (invgamma), 67
rinvgauss (invgauss), 68
rjsu (jsu), 69
rjsu2 (jsu), 69
rkumar (kumar), 72
rlaplace (laplace), 74
rllogis (llogis), 76
rlomax (gb2), 44
rmvt (mvt), 81
rnbinom2 (nbinom2), 82
roibeta (oibeta), 83
roibeta2 (oibeta2), 84
rpareto (pareto), 85
rpgweibull (pgweibull), 86
rpig (pig), 88
rpowerexp (powerexp), 89
rpowerexp2 (powerexp), 89
rsichel (sichel), 92
rsinmad (gb2), 44
rskellam (skellam), 94
rskewnorm (skewnorm), 95
rskewnorm2 (skewnorm2), 96
rskewt (skewt), 98
rskewt2 (skewt2), 99
rt2 (t2), 101
rtruncnorm (truncnorm), 102
rtrunct (trunct), 103
rtrunct2 (trunct2), 104
rvm (vm), 105
rvmf (vmf), 107
rvmf2 (vmf2), 108
rwaring (waring), 109
rwishart (wishart), 111
rwrpcauchy (wrpcauchy), 112
rwrpnorm (wrpnorm), 114
ryules (yules), 115
rzibeta (zibeta), 117
rzibeta2 (zibeta2), 118
rzibetabinom (zibetabinom), 119
rzibinom (zibinom), 121
rzigamma (zigamma), 122
rzigamma2 (zigamma2), 123
rzigeom (zigeom), 124
rziinvgauss (ziinvgauss), 125
rzilnorm (zilnorm), 126
rzinbinom (zinbinom), 127
rzinbinom2 (zinbinom2), 128
rzipois (zipois), 129
rziweibull (ziweibull), 130
rzoibeta (zoibeta), 131
rzoibeta2 (zoibeta2), 132
rztbetabinom (ztbetabinom), 133
rztbinom (ztbinom), 135
rztgeom (ztgeom), 136
rztnbinom (ztnbinom), 137
rztnbinom2 (ztnbinom2), 138
rztpois (ztpois), 139

sdreport, *78*, *79*
Sichel, *89*
sichel, *89*, 92
skellam, 94
skewnorm, *40*, *71*, 95, *97*, *99*, *101*
skewnorm2, *96*, 96, *99*, *101*
skewt, *9*, *71*, *96*, *97*, 98, *101*
skewt2, *96*, *97*, *99*, 99
spgweibull (pgweibull), 86

t2, *58*, 101

truncnorm, 102
trunct, *56*, *58*, 103
trunct2, 104

vm, *21*, *24*, *29*, 105, *113*, *115*
vmf, 107
vmf2, 108

Waring, *116*
waring, 109, *116*
wishart, 111
wrpcauchy, *21*, *24*, *29*, *107*, 112, *115*
wrpnorm, *21*, 114

Yule-Simon, *110*
yules, *110*, 115

zero_inflate, 116
zibeta, 117
zibeta2, 118
zibetabinom, *59*, 119, *134*
zibinom, *60*, *121*, 121
zigamma, 122
zigamma2, 123
zigeom, *49*, *62*, 124, *136*
ziinvgauss, 125
zilnorm, 126
zinbinom, *63*, *125*, 127
zinbinom2, *64*, 128
zipois, *48*, *66*, *125*, 129
ziweibull, 130
zoibeta, 131
zoibeta2, 132
ztbetabinom, *59*, *121*, 133
ztbinom, *60*, *134*, 135
ztgeom, *49*, *62*, *125*, 136
ztnbinom, *63*, *136*, 137
ztnbinom2, *64*, 138
ztpois, *66*, *136*, 139