Package: mnorm (via r-universe)

October 25, 2024

Type Package

Title Multivariate Normal Distribution

Version 1.2.2

Date 2023-11-29

Description Calculates and differentiates probabilities and density of (conditional) multivariate normal distribution and Gaussian copula (with various marginal distributions) using methods described in A. Genz (2004) <doi:10.1023/B:STCO.0000035304.20635.31>, A. Genz, F. Bretz (2009) <doi:10.1007/978-3-642-01689-9>, H. I. Gassmann (2003) <doi:10.1198/1061860032283> and E. Kossova, B. Potanin (2018) <https://ideas.repec.org/a/ris/apltrx/0346.html>.

License GPL (>= 2)

Imports Rcpp (>= 1.0.10), hpa (>= 1.3.1)

LinkingTo Rcpp, RcppArmadillo, hpa

RoxygenNote 7.2.3

NeedsCompilation yes

Author Bogdan Potanin [aut, cre, ctb], Sofiia Dolgikh [ctb]

Maintainer Bogdan Potanin <bogdanpotanin@gmail.com>

Repository CRAN

Date/Publication 2023-11-29 07:10:02 UTC

Contents

cmnorm 2
dmnorm 5
fromBase 10
halton 11
pbetaDiff 12
pmnorm 13
qnormFast 23

rmnorm 24
seqPrimes 26
stdt 26
toBase 28

Index **30**

`cmnorm` *Parameters of conditional multivariate normal distribution*

Description

This function calculates mean (expectation) and covariance matrix of conditional multivariate normal distribution.

Usage

```
cmnorm(
  mean,
  sigma,
  given_ind,
  given_x,
  dependent_ind = numeric(),
  is_validation = TRUE,
  is_names = TRUE,
  control = NULL,
  n_cores = 1L
)
```

Arguments

`mean`	numeric vector representing expectation of multivariate normal vector (distribution).
`sigma`	positively defined numeric matrix representing covariance matrix of multivariate normal vector (distribution).
`given_ind`	numeric vector representing indexes of multivariate normal vector which are conditioned at values given by `given_x` argument.
`given_x`	numeric vector which `i`-th element corresponds to the given value of the `given_ind[i]`-th element (component) of multivariate normal vector. If `given_x` is numeric matrix then it's rows are such vectors of given values.
`dependent_ind`	numeric vector representing indexes of unconditional elements (components) of multivariate normal vector.
`is_validation`	logical value indicating whether input arguments should be validated. Set it to `FALSE` to get performance boost (default value is `TRUE`).
`is_names`	logical value indicating whether output values should have row and column names. Set it to `FALSE` to get performance boost (default value is `TRUE`).

control	a list of control parameters. See Details.
n_cores	positive integer representing the number of CPU cores used for parallel computing. Currently it is not recommended to set n_cores > 1 if vectorized arguments include less then 100000 elements.

Details

Consider m-dimensional multivariate normal vector $X = (X_1, ..., X_m)^T \sim N(\mu, \Sigma)$, where $E(X) = \mu$ and $Cov(X) = \Sigma$ are expectation (mean) and covariance matrix respectively.

Let's denote vectors of indexes of conditioned and unconditioned elements of X by I_g and I_d respectively. By $x^{(g)}$ denote deterministic (column) vector of given values of X_{I_g}. The function calculates expected value and covariance matrix of conditioned multivariate normal vector $X_{I_d}|X_{I_g} = x^{(g)}$. For example if $I_g = (1, 3)$ and $x^{(g)} = (-1, 1)$ then $I_d = (2, 4, 5)$ so the function calculates:

$$\mu_c = E\left((X_2, X_4, X_5)\,|X_1 = -1, X_3 = 1\right)$$

$$\Sigma_c = Cov\left((X_2, X_4, X_5)\,|X_1 = -1, X_3 = 1\right)$$

In general case:

$$\mu_c = E\left(X_{I_d}|X_{I_g} = x^{(g)}\right) = \mu_{I_d} + \left(x^{(g)} - \mu_{I_g}\right)\left(\Sigma_{(I_d,I_g)}\Sigma^{-1}_{(I_g,I_g)}\right)^T$$

$$\Sigma_c = Cov\left(X_{I_d}|X_{I_g} = x^{(g)}\right) = \Sigma_{(I_d,I_d)} - \Sigma_{(I_d,I_g)}\Sigma^{-1}_{(I_g,I_g)}\Sigma_{(I_g,I_d)}$$

Note that $\Sigma_{(A,B)}$, where $A, B \in \{d, g\}$, is a submatrix of Σ generated by intersection of I_A rows and I_B columns of Σ.

Below there is a correspondence between aforementioned theoretical (mathematical) notations and function arguments:

- mean - μ.
- sigma - Σ.
- given_ind - I_g.
- given_x - $x^{(g)}$.
- dependent_ind - I_d

Moreover $\Sigma_{(I_g,I_d)}$ is a theoretical (mathematical) notation for sigma[given_ind, dependent_ind]. Similarly μ_g represents mean[given_ind].

By default dependent_ind contains all indexes that are not in given_ind. It is possible to omit and duplicate indexes of dependent_ind. But at least single index should be provided for given_ind without any duplicates. Also dependent_ind and given_ind should not have the same elements. Moreover given_ind should not be of the same length as mean so at least one component should be unconditioned.

If given_x is a vector then (if possible) it will be treated as a matrix with the number of columns equal to the length of mean.

Currently control has no input arguments intended for the users. This argument is used for some internal purposes of the package.

Value

This function returns an object of class "mnorm_cmnorm".

An object of class "mnorm_cmnorm" is a list containing the following components:

- `mean` - conditional mean.
- `sigma` - conditional covariance matrix.
- `sigma_d` - covariance matrix of unconditioned elements.
- `sigma_g` - covariance matrix of conditioned elements.
- `sigma_dg` - matrix of covariances between unconditioned and conditioned elements.
- `s12s22` - equals to the matrix product of `sigma_dg` and `solve(sigma_g)`.

Note that `mean` corresponds to μ_c while `sigma` represents Σ_c. Moreover `sigma_d` is Σ_{I_d,I_d}, `sigma_g` is Σ_{I_g,I_g} and `sigma_dg` is Σ_{I_d,I_g}.

Since Σ_c do not depend on $X^{(g)}$ the output `sigma` does not depend on `given_x`. In particular output `sigma` remains the same independent of whether `given_x` is a matrix or vector. Oppositely if `given_x` is a matrix then output `mean` is a matrix which rows correspond to conditional means associated with given values provided by corresponding rows of `given_x`.

The order of elements of output `mean` and output `sigma` depends on the order of `dependet_ind` elements that is ascending by default. The order of `given_ind` elements does not matter. But, please, check that the order of `given_ind` match the order of given values i.e. the order of `given_x` columns.

Examples

```
# Consider multivariate normal vector:
# X = (X1, X2, X3, X4, X5) ~ N(mean, sigma)

# Prepare multivariate normal vector parameters
  # expected value
mean <- c(-2, -1, 0, 1, 2)
n_dim <- length(mean)
  # correlation matrix
cor <- c(   1,  0.1,  0.2,   0.3,  0.4,
          0.1,    1, -0.1,  -0.2, -0.3,
          0.2, -0.1,    1,   0.3,  0.2,
          0.3, -0.2,  0.3,     1, -0.05,
          0.4, -0.3,  0.2, -0.05,     1)
cor <- matrix(cor, ncol = n_dim, nrow = n_dim, byrow = TRUE)
  # covariance matrix
sd_mat <- diag(c(1, 1.5, 2, 2.5, 3))
sigma <- sd_mat %*% cor %*% t(sd_mat)

# Estimate parameters of conditional distribution i.e.
# when the first and the third components of X are conditioned:
# (X2, X4, X5 | X1 = -1, X3 = 1)
given_ind <- c(1, 3)
given_x <- c(-1, 1)
par <- cmnorm(mean = mean, sigma = sigma,
```

```
              given_ind = given_ind,
              given_x = given_x)
  # E(X2, X4, X5 | X1 = -1, X3 = 1)
par$mean
  # Cov(X2, X4, X5 | X1 = -1, X3 = 1)
par$sigma

# Additionally calculate E(X2, X4, X5 | X1 = 2, X3 = 3)
given_x_mat <- rbind(given_x, c(2, 3))
par1 <- cmnorm(mean = mean, sigma = sigma,
               given_ind = given_ind,
               given_x = given_x_mat)
par1$mean

# Duplicates and omitted indexes are allowed for dependent_ind
# For given_ind duplicates are not allowed
# Let's calculate conditional parameters for (X5, X2, X5 | X1 = -1, X3 = 1):
dependent_ind <- c(5, 2, 5)
par2 <- cmnorm(mean = mean, sigma = sigma,
               given_ind = given_ind,
               given_x = given_x,
               dependent_ind = dependent_ind)
  # E(X5, X2, X5 | X1 = -1, X3 = 1)
par2$mean
  # Cov(X5, X2, X5 | X1 = -1, X3 = 1)
par2$sigma
```

---

dmnorm *Density of (conditional) multivariate normal distribution*

Description

This function calculates and differentiates density of (conditional) multivariate normal distribution.

Usage

```
dmnorm(
  x,
  mean,
  sigma,
  given_ind = numeric(),
  log = FALSE,
  grad_x = FALSE,
  grad_sigma = FALSE,
  is_validation = TRUE,
  control = NULL,
  n_cores = 1L
)
```

Arguments

- `x` numeric vector representing the point at which density should be calculated. If `x` is a matrix then each row determines a new point.
- `mean` numeric vector representing expectation of multivariate normal vector (distribution).
- `sigma` positively defined numeric matrix representing covariance matrix of multivariate normal vector (distribution).
- `given_ind` numeric vector representing indexes of multivariate normal vector which are conditioned at values of `x` with corresponding indexes i.e. `x[given_x]` or `x[, given_x]` if `x` is a matrix.
- `log` logical; if `TRUE` then probabilities (or densities) p are given as log(p) and derivatives will be given respect to log(p).
- `grad_x` logical; if `TRUE` then the vector of partial derivatives of the density function will be calculated respect to each element of `x`. If `x` is a matrix then gradients will be estimated for each row of `x`.
- `grad_sigma` logical; if `TRUE` then the vector of partial derivatives (gradient) of the density function will be calculated respect to each element of `sigma`. If `x` is a matrix then gradients will be estimated for each row of `x`.
- `is_validation` logical value indicating whether input arguments should be validated. Set it to `FALSE` to get performance boost (default value is `TRUE`).
- `control` a list of control parameters. See Details.
- `n_cores` positive integer representing the number of CPU cores used for parallel computing. Currently it is not recommended to set `n_cores` > 1 if vectorized arguments include less then 100000 elements.

Details

Consider notations from the Details section of `cmnorm`. The function calculates density $f(x^{(d)}|x^{(g)})$ of conditioned multivariate normal vector $X_{I_d}|X_{I_g} = x^{(g)}$. Where $x^{(d)}$ is a subvector of x associated with X_{I_d} i.e. unconditioned components. Therefore `x[given_ind]` represents $x^{(g)}$ while `x[-given_ind]` is $x^{(d)}$.

If `grad_x` is `TRUE` then function additionally estimates the gradient respect to both unconditioned and conditioned components:

$$\nabla f(x^{(d)}|x^{(g)}) = \left(\frac{\partial f(x^{(d)}|x^{(g)})}{\partial x_1}, ..., \frac{\partial f(x^{(d)}|x^{(g)})}{\partial x_m}\right),$$

where each x_i belongs either to $x^{(d)}$ or $x^{(g)}$ depending on whether $i \in I_d$ or $i \in I_g$ correspondingly. In particular subgradients of density function respect to $x^{(d)}$ and $x^{(g)}$ are of the form:

$$\nabla_{x^{(d)}} \ln f(x^{(d)}|x^{(g)}) = -\left(x^{(d)} - \mu_c\right)\Sigma_c^{-1}$$

$$\nabla_{x^{(g)}} \ln f(x^{(d)}|x^{(g)}) = -\nabla_{x^{(d)}} f(x^{(d)}|x^{(g)})\Sigma_{d,g}\Sigma_{g,g}^{-1}$$

If `grad_sigma` is `TRUE` then function additionally estimates the gradient respect to the elements of covariance matrix Σ. For $i \in I_d$ and $j \in I_d$ the function calculates:

$$\frac{\partial \ln f(x^{(d)}|x^{(g)})}{\partial \Sigma_{i,j}} = \left(\frac{\partial \ln f(x^{(d)}|x^{(g)})}{\partial x_i} \times \frac{\partial \ln f(x^{(d)}|x^{(g)})}{\partial x_j} - \Sigma_{c,(i,j)}^{-1}\right) / \left(1 + I(i = j)\right),$$

where $I(i = j)$ is an indicator function which equals 1 when the condition $i = j$ is satisfied and 0 otherwise.

For $i \in I_d$ and $j \in I_g$ the following formula is used:

$$\frac{\partial \ln f(x^{(d)}|x^{(g)})}{\partial \Sigma_{i,j}} = -\frac{\partial \ln f(x^{(d)}|x^{(g)})}{\partial x_i} \times \left(\left(x^{(g)} - \mu_g\right) \Sigma_{g,g}^{-1}\right)_{q_g(j)} -$$

$$-\sum_{k=1}^{n_d} (1 + I(q_d(i) = k)) \times (\Sigma_{d,g}\Sigma_{g,g}^{-1})_{k,q_g(j)} \times \frac{\partial \ln f(x^{(d)}|x^{(g)})}{\partial \Sigma_{i,q_d^{-1}(k)}},$$

where $q_g(j) = \sum_{k=1}^{m} I\left(I_{g,k} \leq j\right)$ and $q_d(i) = \sum_{k=1}^{m} I\left(I_{d,k} \leq i\right)$ represent the order of the i-th and j-th elements in I_g and I_d correspondingly i.e. $x_i = x_{q_d(i)}^{(d)} = x_{I_{d,q_d(i)}}$ and $x_j = x_{q_g(j)}^{(g)} = x_{I_{g,q_g(j)}}$. Note that $q_g(j)^{-1}$ and $q_d(i)^{-1}$ are inverse functions. Number of conditioned and unconditioned components are denoted by $n_g = \sum_{k=1}^{m} I(k \in I_g)$ and $n_d = \sum_{k=1}^{m} I(k \in I_d)$ respectively. For the case $i \in I_g$ and $j \in I_d$ the formula is similar.

For $i \in I_g$ and $j \in I_g$ the following formula is used:

$$\frac{\partial \ln f(x^{(d)}|x^{(g)})}{\partial \Sigma_{i,j}} = -\nabla_{x^{(d)}} \ln f(x^{(d)}|x^{(g)}) \times \left(x^{(g)} \times (\Sigma_{d,g} \times \Sigma_{g,g}^{-1} \times I_g^{*} \times \Sigma_{g,g}^{-1})^T\right)^T -$$

$$-\sum_{k_1=1}^{n_d}\sum_{k_2=k_1}^{n_d} \frac{\partial \ln f(x^{(d)}|x^{(g)})}{\partial \Sigma_{q_d(k_1)^{-1},q_d(k_2)^{-1}}} \left(\Sigma_{d,g} \times \Sigma_{g,g}^{-1} \times I_g^{*} \times \Sigma_{g,g}^{-1} \times \Sigma_{d,g}^{T}\right)_{q_d(k_1)^{-1},q_d(k_2)^{-1}},$$

where I_g^{*} is a square n_g-dimensional matrix of zeros except $I_{g,(i,j)}^{*} = I_{g,(j,i)}^{*} = 1$.

Argument `given_ind` represents I_g and it should not contain any duplicates. The order of `given_ind` elements does not matter so it has no impact on the output.

More details on abovementioned differentiation formulas could be found in the appendix of E. Kossova and B. Potanin (2018).

Currently `control` has no input arguments intended for the users. This argument is used for some internal purposes of the package.

Value

This function returns an object of class "mnorm_dmnorm".

An object of class "mnorm_dmnorm" is a list containing the following components:

- `den` - density function value at `x`.
- `grad_x` - gradient of density respect to `x` if `grad_x` or `grad_sigma` input argument is set to `TRUE`.

- grad_sigma - gradient respect to the elements of sigma if grad_sigma input argument is set to TRUE.

If log is TRUE then den is a log-density so output grad_x and grad_sigma are calculated respect to the log-density.

Output grad_x is a Jacobian matrix which rows are gradients of the density function calculated for each row of x. Therefore grad_x[i, j] is a derivative of the density function respect to the j-th argument at point x[i,].

Output grad_sigma is a 3D array such that grad_sigma[i, j, k] is a partial derivative of the density function respect to the sigma[i, j] estimated for the observation x[k,].

References

E. Kossova., B. Potanin (2018). Heckman method and switching regression model multivariate generalization. Applied Econometrics, vol. 50, pages 114-143.

Examples

```
# Consider multivariate normal vector:
# X = (X1, X2, X3, X4, X5) ~ N(mean, sigma)

# Prepare multivariate normal vector parameters
  # expected value
mean <- c(-2, -1, 0, 1, 2)
n_dim <- length(mean)
  # correlation matrix
cor <- c(   1,  0.1,  0.2,   0.3,  0.4,
          0.1,    1, -0.1,  -0.2, -0.3,
          0.2, -0.1,    1,   0.3,  0.2,
          0.3, -0.2,  0.3,     1, -0.05,
          0.4, -0.3,  0.2, -0.05,     1)
cor <- matrix(cor, ncol = n_dim, nrow = n_dim, byrow = TRUE)
  # covariance matrix
sd_mat <- diag(c(1, 1.5, 2, 2.5, 3))
sigma <- sd_mat %*% cor %*% t(sd_mat)

# Estimate the density of X at point (-1, 0, 1, 2, 3)
x <- c(-1, 0, 1, 2, 3)
d.list <- dmnorm(x = x, mean = mean, sigma = sigma)
d <- d.list$den
print(d)

# Estimate the density of X at points
# x=(-1, 0, 1, 2, 3) and y=(-1.2, -1.5, 0, 1.2, 1.5)
y <- c(-1.5, -1.2, 0, 1.2, 1.5)
xy <- rbind(x, y)
d.list.1 <- dmnorm(x = xy, mean = mean, sigma = sigma)
d.1 <- d.list.1$den
print(d.1)

# Estimate the density of Xc=(X2, X4, X5 | X1 = -1, X3 = 1) at
```

```
# point xd=(0, 2, 3) given conditioning values xg=(-1, 1)
given_ind <- c(1, 3)
d.list.2 <- dmnorm(x = x, mean = mean, sigma = sigma,
                   given_ind = given_ind)
d.2 <- d.list.2$den
print(d.2)

# Estimate the gradient of density respect to the argument and
# covariance matrix at points 'x' and 'y'
d.list.3 <- dmnorm(x = xy, mean = mean, sigma = sigma,
                   grad_x = TRUE, grad_sigma = TRUE)
# Gradient respect to the argument
grad_x.3 <- d.list.3$grad_x
   # at point 'x'
print(grad_x.3[1, ])
   # at point 'y'
print(grad_x.3[2, ])
# Partial derivative at point 'y' respect
# to the 3-rd argument
print(grad_x.3[2, 3])
# Gradient respect to the covariance matrix
grad_sigma.3 <- d.list.3$grad_sigma
# Partial derivative respect to sigma(3, 5) at
# point 'y'
print(grad_sigma.3[3, 5, 2])

# Estimate the gradient of the log-density function of
# Xc=(X2, X4, X5 | X1 = -1, X3 = 1) and Yc=(X2, X4, X5 | X1 = -1.5, X3 = 0)
# respect to the argument and covariance matrix at
# points xd=(0, 2, 3) and yd=(-1.2, 0, 1.5) respectively given
# conditioning values xg=(-1, 1) and yg=(-1.5, 0) correspondingly
d.list.4 <- dmnorm(x = xy, mean = mean, sigma = sigma,
                   grad_x = TRUE, grad_sigma = TRUE,
                   given_ind = given_ind, log = TRUE)
# Gradient respect to the argument
grad_x.4 <- d.list.4$grad_x
   # at point 'xd'
print(grad_x.4[1, ])
   # at point 'yd'
print(grad_x.4[2, ])
# Partial derivative at point 'xd' respect to 'xg[2]'
print(grad_x.4[1, 3])
# Partial derivative at point 'yd' respect to 'yd[5]'
print(grad_x.4[2, 5])
# Gradient respect to the covariance matrix
grad_sigma.4 <- d.list.4$grad_sigma
# Partial derivative respect to sigma(3, 5) at
# point 'yd'
print(grad_sigma.4[3, 5, 2])

# Compare analytical gradients from the previous example with
# their numeric (forward difference) analogues at point 'xd'
# given conditioning 'xg'
```

```
delta <- 1e-6
grad_x.num <- rep(NA, 5)
grad_sigma.num <- matrix(NA, nrow = 5, ncol = 5)
for (i in 1:5)
{
  x.delta <- x
  x.delta[i] <- x[i] + delta
  d.list.delta <- dmnorm(x = x.delta, mean = mean, sigma = sigma,
                         grad_x = TRUE, grad_sigma = TRUE,
                         given_ind = given_ind, log = TRUE)
  grad_x.num[i] <- (d.list.delta$den - d.list.4$den[1]) / delta
   for(j in 1:5)
   {
     sigma.delta <- sigma
     sigma.delta[i, j] <- sigma[i, j] + delta
     sigma.delta[j, i] <- sigma[j, i] + delta
     d.list.delta <- dmnorm(x = x, mean = mean, sigma = sigma.delta,
                            grad_x = TRUE, grad_sigma = TRUE,
                            given_ind = given_ind, log = TRUE)
     grad_sigma.num[i, j] <- (d.list.delta$den - d.list.4$den[1]) / delta
   }
}
# Comparison of gradients respect to the argument
h.x <- cbind(analytical = grad_x.4[1, ], numeric = grad_x.num)
rownames(h.x) <- c("xg[1]", "xd[1]", "xg[2]", "xd[3]", "xd[4]")
print(h.x)
# Comparison of gradients respect to the covariance matrix
h.sigma <- list(analytical = grad_sigma.4[, , 1], numeric = grad_sigma.num)
print(h.sigma)
```

---

fromBase *Convert base representation of a number into integer*

Description

Converts base representation of a number into integer.

Usage

```
fromBase(x, base = 2L)
```

Arguments

x	vector of positive integer coefficients representing the number in base that is base.
base	positive integer representing the base.

Value

The function returns a positive integer that is a conversion from base under given coefficients x.

Examples

```
fromBase(c(1, 2, 0, 2, 3), 5)
```

`halton` *Halton sequence*

Description

Calculate elements of the Halton sequence and of some other pseudo-random sequences.

Usage

```
halton(
  n = 1L,
  base = as.integer(c(2)),
  start = 1L,
  random = "NO",
  type = "halton",
  scrambler = "NO",
  is_validation = TRUE,
  n_cores = 1L
)
```

Arguments

`n`	positive integer representing the number of sequence elements.
`base`	vector of positive integers greater then one representing the bases for each of the sequences.
`start`	non-negative integer representing the index of the first element of the sequence to be included in the output sequence.
`random`	string representing the method of randomization to be applied to the sequence. If `random = "NO"` (default) then there is no randomization. If `random = "Tuffin"` then standard uniform random variable will be added to each element of the sequence and the difference between this sum and it's 'floor' will be returned as a new element of the sequence.
`type`	string representing type of the sequence. Default is "halton" that is Halton sequence. The alternative is "richtmyer" corresponding to Richtmyer sequence.
`scrambler`	string representing scrambling method for the Halton sequence. Possible options are `"NO"` (default), `"root"` and `"negroot"` which described in S. Kolenikov (2012).
`is_validation`	logical value indicating whether input arguments should be validated. Set it to `FALSE` to get performance boost (default value is `TRUE`).
`n_cores`	positive integer representing the number of CPU cores used for parallel computing. Currently it is not recommended to set `n_cores` > 1 if vectorized arguments include less then 100000 elements.

Details

Function seqPrimes could be used to provide the prime numbers for the base input argument.

Value

The function returns a matrix which i-th column is a sequence with base base[i] and elements with indexes from start to start + n.

References

J. Halton (1964) <doi:10.2307/2347972>

S. Kolenikov (2012) <doi:10.1177/1536867X1201200103>

Examples

```
halton(n = 100, base = c(2, 3, 5), start = 10)
```

---

pbetaDiff *Differentiate Regularized Incomplete Beta Function.*

Description

Calculate derivatives of the regularized incomplete beta function that is a cumulative distribution function of the beta distribution.

Usage

```
pbetaDiff(x, p = 10, q = 0.5, n = 10L, is_validation = TRUE, control = NULL)
```

Arguments

x	numeric vector of values between 0 and 1. It is similar to q argument of pbeta function.
p	similar to shape1 argument of pbeta function.
q	similar to shape2 argument of pbeta function.
n	positive integer representing the number of iterations used to calculate the derivatives. Greater values provide higher accuracy by the cost of more computational resources.
is_validation	logical; if TRUE then input arguments are validated. Set to FALSE to slightly increase the performance of the function.
control	list of control parameters. Currently not intended for the users.

Details

The function implements differentiation algorithm of R. Boik and J. Robinson-Cox (1998). Currently only first-order derivatives are considered.

Value

The function returns a list which has the following elements:

- dx - numeric vector of derivatives respect to each element of x.
- dp - numeric vector of derivatives respect to p for each element of x.
- dq - numeric vector of derivatives respect to q for each element of x.

References

Boik, R. J. and Robinson-Cox, J. F. (1998). Derivatives of the Incomplete Beta Function. Journal of Statistical Software, 3 (1), pages 1-20.

Examples

```
# Some values from Table 1 of R. Boik and J. Robinson-Cox (1998)
pbetaDiff(x = 0.001,  p = 1.5, q = 11)
pbetaDiff(x = 0.5,  p = 1.5, q = 11)

# Compare analytical and numeric derivatives
delta <- 1e-6
x <- c(0.01, 0.25, 0.5, 0.75, 0.99)
p <- 5
q <- 10
out <- pbetaDiff(x = x, p = p, q = q)
p0 <- pbeta(q = x, shape1 = p, shape2 = q)

# Derivatives respect to x
p1 <- pbeta(q = x + delta, shape1 = p, shape2 = q)
data.frame(numeric = (p1 - p0) / delta, analytical = out$dx)

# Derivatives respect to p
p1 <- pbeta(q = x, shape1 = p + delta, shape2 = q)
data.frame(numeric = (p1 - p0) / delta, analytical = out$dp)

# Derivatives respect to q
p1 <- pbeta(q = x, shape1 = p, shape2 = q + delta)
data.frame(numeric = (p1 - p0) / delta, analytical = out$dq)
```

---

pmnorm *Probabilities of (conditional) multivariate normal distribution*

Description

This function calculates and differentiates probabilities of (conditional) multivariate normal distribution.

Usage

```
pmnorm(
  lower,
  upper,
  given_x = numeric(),
  mean = numeric(),
  sigma = matrix(),
  given_ind = numeric(),
  n_sim = 1000L,
  method = "default",
  ordering = "mean",
  log = FALSE,
  grad_lower = FALSE,
  grad_upper = FALSE,
  grad_sigma = FALSE,
  grad_given = FALSE,
  is_validation = TRUE,
  control = NULL,
  n_cores = 1L,
  marginal = NULL,
  grad_marginal = FALSE,
  grad_marginal_prob = FALSE
)
```

Arguments

`lower`	numeric vector representing lower integration limits. If `lower` is a matrix then each row determines new limits. Negative infinite values are allowed while positive infinite values are prohibited.
`upper`	numeric vector representing upper integration limits. If `upper` is a matrix then each row determines new limits. Positive infinite values are allowed while negative infinite values are prohibited.
`given_x`	numeric vector which `i`-th element corresponds to the given value of the `given_ind[i]`-th element (component) of multivariate normal vector. If `given_x` is numeric matrix then it's rows are such vectors of given values.
`mean`	numeric vector representing expectation of multivariate normal vector (distribution).
`sigma`	positively defined numeric matrix representing covariance matrix of multivariate normal vector (distribution).
`given_ind`	numeric vector representing indexes of multivariate normal vector which are conditioned at values given by `given_x` argument.
`n_sim`	positive integer representing the number of draws from Richtmyer sequence in GHK algorithm. More draws provide more accurate results by the cost of additional computational burden. Alternative types of sequences could be provided via `random_sequence` argument.

method
: string representing the method to be used to calculate multivariate normal probabilities. Possible options are "GHK" and "Gassmann" recommended for high dimensional (more than 5) and low dimensional probabilities correspondingly. Additional option "default" selects optimal method depending on the number of dimensions. See 'Details' for additional information.

ordering
: string representing the method to be used to order the integrals. See Details section below.

log
: logical; if TRUE then probabilities (or densities) p are given as log(p) and derivatives will be given respect to log(p).

grad_lower
: logical; if TRUE then the vector of partial derivatives of the probability will be calculated respect to each element of lower. If lower is a matrix then gradients will be estimated for each row of lower.

grad_upper
: logical; if TRUE then the vector of partial derivatives of the probability will be calculated respect to each element of upper. If upper is a matrix then gradients will be estimated for each row of upper.

grad_sigma
: logical; if TRUE then the vector of partial derivatives (gradient) of the probability will be calculated respect to each element of sigma. If lower and upper are matrices then gradients will be estimated for each row of these matrices.

grad_given
: logical; if TRUE then the vector of partial derivatives of the density function will be calculated respect to each element of given_x. If given_x is a matrix then gradients will be estimated for each row of given_x.

is_validation
: logical value indicating whether input arguments should be validated. Set it to FALSE to get performance boost (default value is TRUE).

control
: a list of control parameters. See Details.

n_cores
: positive integer representing the number of CPU cores used for parallel computing. Currently it is not recommended to set n_cores > 1 if vectorized arguments include less then 100000 elements.

marginal
: list such that marginal[[i]] represents parameters of marginal distribution of the i-th component of the random vector and names(marginal)[i] is a name of this distribution. If names(marginal)[i] = "logistic" or names(marginal)[i] = "normal" then marginal[[i]] should be an empty vector or NULL. If names(marginal)[i] = "student" then marginal[[i]] should contain a single element representing degrees of freedom. If names(marginal)[i] = "PGN" or names(marginal)[i] = "hpa" then marginal[[i]] should be a numeric vector which elements correspond to pc argument of phpa0.

grad_marginal
: logical; if TRUE then the vector of partial derivatives (gradient) of probability will be calculated respect to each parameter of marginal distributions i.e. respect to each element of marginal. The gradient respect to the parameters of the i-th marginal distribution will be stored in the grad_marginal[[i]] output matrix which j-th column stores derivatives respect to marginal[[i]][j].

grad_marginal_prob
: logical; if TRUE then the vector of partial derivatives (gradient) of probability will be calculated respect to each cumulative marginal probability of marginal distributions.

Details

Consider notations from the Details sections of cmnorm and dmnorm. The function calculates probabilities of the form:

$$P\left(x^{(l)} \leq X_{I_d} \leq x^{(u)}|X_{I_g} = x^{(g)}\right)$$

where $x^{(l)}$ and $x^{(u)}$ are lower and upper integration limits respectively i.e. `lower` and `upper` correspondingly. Also $x^{(g)}$ represents `given_x`. Note that `lower` and `upper` should be matrices of the same size. Also `given_x` should have the same number of rows as `lower` and `upper`.

To calculate bivariate probabilities the function applies the method of Drezner and Wesolowsky described in A. Genz (2004). In contrast to the classical implementation of this method the function applies Gauss-Legendre quadrature with 30 sample points to approximate integral (1) of A. Genz (2004). Classical implementations of this method use up to 20 points but requires some additional transformations of (1). During preliminary testing it has been found that approach with 30 points provides similar accuracy being slightly faster because of better vectorization capabilities.

To calculate trivariate probabilities the function uses Drezner method following formula (14) of A. Genz (2004). Similarly to bivariate case 30 points are used in Gauss-Legendre quadrature.

The function may apply the method of Gassmann (2003) for estimation of $m > 3$ dimensional normal probabilities. It uses matrix 5 representation of Gassmann (2003) and 30 points of Gauss-Legendre quadrature.

For m-variate probabilities, where $m > 1$, the function may apply GHK algorithm described in section 4.2 of A. Genz and F. Bretz (2009). The implementation of GHK is based on deterministic Halton sequence with `n_sim` draws and uses variable reordering suggested in section 4.1.3 of A. Genz and F. Bretz (2009). The ordering algorithm may be determined via `ordering` argument. Available options are `"NO"`, `"mean"` (default), and `"variance"`.

Univariate probabilities are always calculated via standard approach so in this case `method` will not affect the output. If `method = "Gassmann"` then the function uses fast (aforementioned) algorithms for bivariate and trivariate probabilities or the method of Gassmann for $m > 3$ dimensional probabilities. If `method = "GHK"` then GHK algorithm is used. If `method = "default"` then `"Gassmann"` is used for bivariate and trivariate probabilities while `"GHK"` is used for $m > 3$ dimensional probabilities. During future updates `"Gassmann"` may become a default method for calculation of the $4-5$ dimensional probabilities.

We are going to provide alternative estimation algorithms during future updates. These methods will be available via `method` argument.

The function is optimized to perform much faster when all upper integration limits `upper` are finite while all lower integration limits `lower` are negative infinite. The derivatives could be also calculated much faster when some integration limits are infinite.

For simplicity of notations further let's consider unconditioned probabilities. Derivatives respect to conditioned components are similar to those mentioned in Details section of dmnorm. We also provide formulas for $m \geq 3$. But the function may calculate derivatives for $m \leq 2$ using some simplifications of the formulas mentioned below.

If `grad_upper` is `TRUE` then function additionally estimates the gradient respect to `upper`:

$$\frac{\partial P\left(x^{(l)} \leq X \leq x^{(u)}\right)}{\partial x_i^{(u)}} = P\left(x_{(-i)}^{(l)} \leq X_{(-i)} \leq x_{(-i)}^{(u)}|X_i = x_i^{(u)}\right) f_{X_i}\left(x_i^{(u)}; \mu_i, \Sigma_{i,i}\right)$$

If grad_lower is TRUE then function additionally estimates the gradient respect to lower:

$$\frac{\partial P\left(x^{(l)}\leq X\leq x^{(u)}\right)}{\partial x_i^{(l)}} = -P\left(x_{(-i)}^{(l)}\leq X_{(-i)}\leq x_{(-i)}^{(u)}|X_i=x_i^{(l)}\right)f_{X_i}\left(x_i^{(l)};\mu_i,\Sigma_{i,i}\right)$$

If grad_sigma is TRUE then function additionally estimates the gradient respect to sigma. For $i \neq j$ the function calculates derivatives respect to the covariances:

$$\frac{\partial P\left(x^{(l)}\leq X\leq x^{(u)}\right)}{\partial \Sigma_{i,j}} =$$

$$= P\left(x_{(-(i,j))}^{(l)}\leq X_{-(i,j)}\leq x_{(-(i,j))}^{(u)}|X_i=x_i^{(u)},X_j=x_j^{(u)}\right)f_{X_i,X_j}\left(x_i^{(u)},x_j^{(u)};\mu_{(i,j)},\Sigma_{(i,j),(i,j)}\right)-$$

$$-P\left(x_{(-(i,j))}^{(l)}\leq X_{-(i,j)}\leq x_{(-(i,j))}^{(u)}|X_i=x_i^{(l)},X_j=x_j^{(u)}\right)f_{X_i,X_j}\left(x_i^{(l)},x_j^{(u)};\mu_{(i,j)},\Sigma_{(i,j),(i,j)}\right)-$$

$$-P\left(x_{(-(i,j))}^{(l)}\leq X_{-(i,j)}\leq x_{(-(i,j))}^{(u)}|X_i=x_i^{(u)},X_j=x_j^{(l)}\right)f_{X_i,X_j}\left(x_i^{(u)},x_j^{(l)};\mu_{(i,j)},\Sigma_{(i,j),(i,j)}\right)+$$

$$+P\left(x_{(-(i,j))}^{(l)}\leq X_{-(i,j)}\leq x_{(-(i,j))}^{(u)}|X_i=x_i^{(l)},X_j=x_j^{(l)}\right)f_{X_i,X_j}\left(x_i^{(l)},x_j^{(l)};\mu_{(i,j)},\Sigma_{(i,j),(i,j)}\right)$$

Note that if some of integration limits are infinite then some elements of this equation converge to zero which highly simplifies the calculations.

Derivatives respect to variances are calculated using derivatives respect to covariances and integration limits:

$$\frac{\partial P\left(x^{(l)}\leq X\leq x^{(u)}\right)}{\partial \Sigma_{i,i}} =$$

$$-\frac{\partial P\left(x^{(l)}\leq X\leq x^{(u)}\right)}{\partial x_i^{(l)}}\frac{x_i^{(l)}}{2\Sigma_{i,i}}-\frac{\partial P\left(x^{(l)}\leq X\leq x^{(u)}\right)}{\partial x_i^{(u)}}\frac{x_i^{(u)}}{2\Sigma_{i,i}}-$$

$$-\sum_{j\neq i}\frac{\partial P\left(x^{(l)}\leq X\leq x^{(u)}\right)}{\partial \Sigma_{i,j}}\frac{\Sigma_{i,j}}{2\Sigma_{i,i}}$$

If grad_given is TRUE then function additionally estimates the gradient respect to given_x using formulas similar to those described in Details section of dmnorm.

More details on abovementioned differentiation formulas could be found in the appendix of E. Kossova and B. Potanin (2018).

If marginal is not empty then Gaussian copula will be used instead of a classical multivariate normal distribution. Without loss of generality let's assume that μ is a vector of zeros and introduce some additional notations:

$$q_i^{(u)} = \Phi^{-1}\left(P_i\left(\frac{x_i^{(u)}}{\sigma_i}\right)\right)$$

$$q_i^{(l)} = \Phi^{-1}\left(P_i\left(\frac{x_i^{(l)}}{\sigma_i}\right)\right)$$

where $\Phi(.)^{-1}$ is a quantile function of a standard normal distribution and P_i is a cumulative distribution function of the standartized (to zero mean and unit variance) marginal distribution which name and parameters are defined by `names(marginal)[i]` and `marginal[[i]]` correspondingly. For example if `marginal[i] = "logistic"` then:

$$P_i(t) = \frac{1}{1 + e^{-\pi t/\sqrt{3}}}$$

Let's denote by X^* random vector which is distributed with Gaussian (its covariance matrix is Σ) copula and marginals defined by `marginal`. Then probabilities for these random vector are calculated as follows:

$$P\left(x^{(l)} \leq X^* \leq x^{(u)}\right) = P\left(\sigma q^{(l)} \leq X \leq \sigma q^{(u)}\right) = P_0\left(\sigma q^{(l)}, \sigma q^{(u)}\right)$$

where $q^{(l)} = (q_1^{(l)}, ..., q_m^{(l)})$, $q^{(u)} = (q_1^{(u)}, ..., q_m^{(u)})$ and $\sigma = (\sqrt{\Sigma_{1,1}}, ..., \sqrt{\Sigma_{m,m}})$. Therefore probabilities of X^* may be calculated using probabilities of multivariate normal random vector X (with the same covariance matrix) by substituting lower and upper integration limits $x^{(l)}$ and $x^{(u)}$ with $\sigma q^{(l)}$ and $\sigma q^{(u)}$ correspondingly. Therefore differentiation formulas are similar to those mentioned above and will be automatically adjusted if `marginal` is not empty (including conditional probabilities).

Argument `control` is a list with the following input parameters:

- `random_sequence` – numeric matrix of uniform pseudo random numbers (like Halton sequence). The number of columns should be equal to the number of dimensions of multivariate random vector. If omitted than this matrix will be generated automatically using `n_sim` number of simulations.

Value

This function returns an object of class "mnorm_pmnorm".

An object of class "mnorm_pmnorm" is a list containing the following components:

- `prob` - probability that multivariate normal random variable will be between `lower` and `upper` bounds.
- `grad_lower` - gradient of probability respect to `lower` if `grad_lower` or `grad_sigma` input argument is set to `TRUE`.
- `grad_upper` - gradient of probability respect to `upper` if `grad_upper` or `grad_sigma` input argument is set to `TRUE`.
- `grad_sigma` - gradient respect to the elements of `sigma` if `grad_sigma` input argument is set to `TRUE`.
- `grad_given` - gradient respect to the elements of `given_x` if `grad_given` input argument is set to `TRUE`.
- `grad_marginal` - gradient respect to the elements of `marginal_par` if `grad_marginal` input argument is set to `TRUE`. Currently only derivatives respect to the parameters of `"PGN"` distribution are available.

If log is TRUE then prob is a log-probability so output grad_lower, grad_upper, grad_sigma and grad_given are calculated respect to the log-probability.

Output grad_lower and grad_upper are Jacobian matrices which rows are gradients of the probabilities calculated for each row of lower and upper correspondingly. Similarly grad_given is a Jacobian matrix respect to given_x.

Output grad_sigma is a 3D array such that grad_sigma[i, j, k] is a partial derivative of the probability function respect to the sigma[i, j] estimated for k-th observation.

Output grad_marginal is a list such that grad_marginal[[i]] is a Jacobian matrice which rows are gradients of the probabilities calculated for each row of lower and upper correspondingly respect to the elements of marginal_par[[i]].

References

Genz, A. (2004), Numerical computation of rectangular bivariate and trivariate normal and t-probabilities, Statistics and Computing, 14, 251-260.

Genz, A. and Bretz, F. (2009), Computation of Multivariate Normal and t Probabilities. Lecture Notes in Statistics, Vol. 195. Springer-Verlag, Heidelberg.

E. Kossova, B. Potanin (2018). Heckman method and switching regression model multivariate generalization. Applied Econometrics, vol. 50, pages 114-143.

H. I. Gassmann (2003). Multivariate Normal Probabilities: Implementing an Old Idea of Plackett's. Journal of Computational and Graphical Statistics, vol. 12 (3), pages 731-752.

Examples

```
# Consider multivariate normal vector:
# X = (X1, X2, X3, X4, X5) ~ N(mean, sigma)

# Prepare multivariate normal vector parameters
  # expected value
mean <- c(-2, -1, 0, 1, 2)
n_dim <- length(mean)
  # correlation matrix
cor <- c(   1,  0.1,  0.2,   0.3,  0.4,
          0.1,    1, -0.1,  -0.2, -0.3,
          0.2, -0.1,    1,   0.3,  0.2,
          0.3, -0.2,  0.3,     1, -0.05,
          0.4, -0.3,  0.2, -0.05,     1)
cor <- matrix(cor, ncol = n_dim, nrow = n_dim, byrow = TRUE)
  # covariance matrix
sd_mat <- diag(c(1, 1.5, 2, 2.5, 3))
sigma <- sd_mat %*% cor %*% t(sd_mat)

# Estimate probability:
# P(-3 < X1 < 1, -2.5 < X2 < 1.5, -2 < X3 < 2, -1.5 < X4 < 2.5, -1 < X5 < 3)
lower <- c(-3, -2.5, -2, -1.5, -1)
upper <- c(1, 1.5, 2, 2.5, 3)
p.list <- pmnorm(lower = lower, upper = upper,
                 mean = mean, sigma = sigma)
p <- p.list$prob
```

```
print(p)

# Additionally estimate a probability
lower.1 <- c(-Inf, 0, -Inf, 1, -Inf)
upper.1 <- c(Inf, Inf, 3, 4, 5)
lower.mat <- rbind(lower, lower.1)
upper.mat <- rbind(upper, upper.1)
p.list.1 <- pmnorm(lower = lower.mat, upper = upper.mat,
                   mean = mean, sigma = sigma)
p.1 <- p.list.1$prob
print(p.1)

# Estimate the probabilities
# P(-1 < X1 < 1, -3 < X3 < 3, -5 < X5 < 5 | X2 = -2, X4 = 4)
lower.2 <- c(-1, -3, -5)
upper.2 <- c(1, 3, 5)
given_ind <- c(2, 4)
given_x <- c(-2, 4)
p.list.2 <- pmnorm(lower = lower.2, upper = upper.2,
                   mean = mean, sigma = sigma,
                   given_ind = given_ind, given_x = given_x)
p.2 <- p.list.2$prob
print(p.2)

# Additionally estimate the probability
# P(-Inf < X1 < 1, -3 < X3 < Inf, -Inf < X5 < Inf | X2 = 4, X4 = -2)
lower.3 <- c(-Inf, -3, -Inf)
upper.3 <- c(1, Inf, Inf)
given_x.1 <- c(-2, 4)
lower.mat.2 <- rbind(lower.2, lower.3)
upper.mat.2 <- rbind(upper.2, upper.3)
given_x.mat <- rbind(given_x, given_x.1)
p.list.3 <- pmnorm(lower = lower.mat.2, upper = upper.mat.2,
                   mean = mean, sigma = sigma,
                   given_ind = given_ind, given_x = given_x.mat)
p.3 <- p.list.3$prob
print(p.3)

# Estimate the gradient of previous probabilities respect various arguments
p.list.4 <- pmnorm(lower = lower.mat.2, upper = upper.mat.2,
                   mean = mean, sigma = sigma,
                   given_ind = given_ind, given_x = given_x.mat,
                   grad_lower = TRUE, grad_upper = TRUE,
                   grad_sigma = TRUE, grad_given = TRUE)
p.4 <- p.list.4$prob
print(p.4)
# Gradient respect to 'lower'
grad_lower <- p.list.4$grad_lower
   # for the first probability
print(grad_lower[1, ])
   # for the second probability
print(grad_lower[2, ])
# Gradient respect to 'upper'
```

```
grad_upper <- p.list.4$grad_upper
   # for the first probability
print(grad_upper[1, ])
   # for the second probability
print(grad_upper[2, ])
# Gradient respect to 'given_x'
grad_given <- p.list.4$grad_given
   # for the first probability
print(grad_given[1, ])
   # for the second probability
print(grad_given[2, ])
# Gradient respect to 'sigma'
grad_given <- p.list.4$grad_given
   # for the first probability
print(grad_given[1, ])
   # for the second probability
print(grad_given[2, ])


# Compare analytical gradients from the previous example with
# their numeric (forward difference) analogues for the first probability
n_dependent <- length(lower.2)
n_given <- length(given_x)
n_dim <- n_dependent + n_given
delta <- 1e-6
grad_lower.num <- rep(NA, n_dependent)
grad_upper.num <- rep(NA, n_dependent)
grad_given.num <- rep(NA, n_given)
grad_sigma.num <- matrix(NA, nrow = n_dim, ncol = n_dim)
for (i in 1:n_dependent)
{
  # respect to lower
  lower.delta <- lower.2
  lower.delta[i] <- lower.2[i] + delta
  p.list.delta <- pmnorm(lower = lower.delta, upper = upper.2,
                         given_x = given_x,
                         mean = mean, sigma = sigma,
                         given_ind = given_ind)
  grad_lower.num[i] <- (p.list.delta$prob - p.list.4$prob[1]) / delta
  # respect to upper
  upper.delta <- upper.2
  upper.delta[i] <- upper.2[i] + delta
  p.list.delta <- pmnorm(lower = lower.2, upper = upper.delta,
                         given_x = given_x,
                         mean = mean, sigma = sigma,
                         given_ind = given_ind)
  grad_upper.num[i] <- (p.list.delta$prob - p.list.4$prob[1]) / delta
}
for (i in 1:n_given)
{
  # respect to lower
  given_x.delta <- given_x
  given_x.delta[i] <- given_x[i] + delta
```

```
  p.list.delta <- pmnorm(lower = lower.2, upper = upper.2,
                         given_x = given_x.delta,
                         mean = mean, sigma = sigma,
                         given_ind = given_ind)
  grad_given.num[i] <- (p.list.delta$prob - p.list.4$prob[1]) / delta
}
for (i in 1:n_dim)
{
  for(j in 1:n_dim)
  {
    # respect to sigma
    sigma.delta <- sigma
    sigma.delta[i, j] <- sigma[i, j] + delta
    sigma.delta[j, i] <- sigma[j, i] + delta
    p.list.delta <- pmnorm(lower = lower.2, upper = upper.2,
                           given_x = given_x,
                           mean = mean, sigma = sigma.delta,
                           given_ind = given_ind)
    grad_sigma.num[i, j] <- (p.list.delta$prob - p.list.4$prob[1]) / delta
  }
}
# Comparison of gradients respect to lower integration limits
h.lower <- cbind(analytical = p.list.4$grad_lower[1, ],
                 numeric = grad_lower.num)
print(h.lower)
# Comparison of gradients respect to upper integration limits
h.upper <- cbind(analytical = p.list.4$grad_upper[1, ],
                 numeric = grad_upper.num)
print(h.upper)
# Comparison of gradients respect to given values
h.given <- cbind(analytical = p.list.4$grad_given[1, ],
                 numeric = grad_given.num)
print(h.given)
# Comparison of gradients respect to the covariance matrix
h.sigma <- list(analytical = p.list.4$grad_sigma[, , 1],
                numeric = grad_sigma.num)
print(h.sigma)

# Let's again estimate probability
# P(-1 < X1 < 1, -3 < X3 < 3, -5 < X5 < 5 | X2 = -2, X4 = 4)
# But assume that standardized (to zero mean and unit variance):
# 1) X1 and X2 have standardized PGN distribution with coefficients vectors
#    pc1 = (0.5, -0.2, 0.1) and pc2 = (0.2, 0.05) correspondingly.
# 2) X3 has standardized student distribution with 5 degrees of freedom
# 3) X4 has standardized logistic distribution
# 4) X5 has standard normal distribution
marginal <- list(PGN = c(0.5, -0.2, 0.1), hpa = c(0.2, 0.05),
                 student = 5, logistic = numeric(), normal = NULL)
p.list.5 <- pmnorm(lower = lower.2, upper = upper.2,
                   mean = mean, sigma = sigma,
                   given_ind = given_ind, given_x = given_x,
                   grad_lower = TRUE, grad_upper = TRUE,
                   grad_sigma = TRUE, grad_given = TRUE,
```

```
                   marginal = marginal, grad_marginal = TRUE)
# Lets investigate derivatives respect to parameters
# of marginal distributions
  # respect to pc1 of X1
p.list.5$grad_marginal[[1]]
  # respect to pc2 of X2
p.list.5$grad_marginal[[2]]
  # derivative respect to degrees of freedom of X5 is
  # currently unavailable and will be set to 0
p.list.5$grad_marginal[[3]]
```

`qnormFast` *Quantile function of a normal distribution*

Description

Calculate quantile of a normal distribution using one of the available methods.

Usage

```
qnormFast(
  p,
  mean = 0L,
  sd = 1L,
  method = "Voutier",
  is_validation = TRUE,
  n_cores = 1L
)
```

Arguments

`p`	numeric vector of values between 0 and 1 representing levels of the quantiles.
`mean`	numeric value representing the expectation of a normal distribution.
`sd`	positive numeric value representing standard deviation of a normal distribution.
`method`	character representing the method to be used for quantile calculation. Available options are "Voutier" (default) and "Shore".
`is_validation`	logical value indicating whether input arguments should be validated. Set it to `FALSE` to get performance boost (default value is `TRUE`).
`n_cores`	positive integer representing the number of CPU cores used for parallel computing. Currently it is not recommended to set `n_cores` > 1 if vectorized arguments include less then 100000 elements.

Details

If `method = "Voutier"` then the method of P. Voutier (2010) is used which maximum absolute error is about 0.000025. If `method = "Shore"` then the approach proposed by H. Shore (1982) is applied which maximum absolute error is about 0.026 for quantiles of level between 0.0001 and 0.9999.

Value

The function returns a vector of `p`-level quantiles of a normal distribution with mean equal to `mean` and standard deviation equal to `sd`.

References

H. Shore (1982) <doi:10.2307/2347972>

P. Voutier (2010) <doi:10.48550/arXiv.1002.0567>

Examples

```
qnormFast(c(0.1, 0.9), mean = 1, sd = 2)
```

`rmnorm` *Random number generator for (conditional) multivariate normal distribution*

Description

This function generates random numbers (i.e. variates) from (conditional) multivariate normal distribution.

Usage

```
rmnorm(
  n,
  mean,
  sigma,
  given_ind = numeric(),
  given_x = numeric(),
  dependent_ind = numeric(),
  is_validation = TRUE,
  n_cores = 1L
)
```

Arguments

`n`	positive integer representing the number of random variates to be generated from (conditional) multivariate normal distribution. If `given_ind` is not empty vector then `n` should be be equal to `nrow(given_x)`.
`mean`	numeric vector representing expectation of multivariate normal vector (distribution).
`sigma`	positively defined numeric matrix representing covariance matrix of multivariate normal vector (distribution).
`given_ind`	numeric vector representing indexes of multivariate normal vector which are conditioned at values given by `given_x` argument.

`given_x` numeric vector which `i`-th element corresponds to the given value of the `given_ind[i]`-th element (component) of multivariate normal vector. If `given_x` is numeric matrix then it's rows are such vectors of given values.

`dependent_ind` numeric vector representing indexes of unconditional elements (components) of multivariate normal vector.

`is_validation` logical value indicating whether input arguments should be validated. Set it to `FALSE` to get performance boost (default value is `TRUE`).

`n_cores` positive integer representing the number of CPU cores used for parallel computing. Currently it is not recommended to set `n_cores` > 1 if vectorized arguments include less then 100000 elements.

Details

This function uses Cholesky decomposition to generate multivariate normal variates from independent standard normal variates.

Value

This function returns a numeric matrix which rows a random variates from (conditional) multivariate normal distribution with mean equal to `mean` and covariance equal to `sigma`. If `given_x` and `given_ind` are also provided then random variates will be from conditional multivariate normal distribution. Please, see details section of `cmnorm` to get additional information on the conditioning procedure.

Examples

```
# Consider multivariate normal vector:
# X = (X1, X2, X3, X4, X5) ~ N(mean, sigma)

# Prepare multivariate normal vector parameters
  # expected value
mean <- c(-2, -1, 0, 1, 2)
n_dim <- length(mean)
  # correlation matrix
cor <- c(   1,  0.1,  0.2,   0.3,  0.4,
          0.1,    1, -0.1,  -0.2, -0.3,
          0.2, -0.1,    1,   0.3,  0.2,
          0.3, -0.2,  0.3,     1, -0.05,
          0.4, -0.3,  0.2, -0.05,     1)
cor <- matrix(cor, ncol = n_dim, nrow = n_dim, byrow = TRUE)
  # covariance matrix
sd_mat <- diag(c(1, 1.5, 2, 2.5, 3))
sigma <- sd_mat %*% cor %*% t(sd_mat)

# Simulate random variates from this distribution
rmnorm(n = 3, mean = mean, sigma = sigma)

# Simulate random variate from (X1, X3, X5 | X1 = -1, X4 = 1)
given_x <- c(-1, 1)
given_ind = c(1, 4)
```

```
rmnorm(n = 1, mean = mean, sigma = sigma,
       given_x = given_x, given_ind = given_ind)

# Simulate random variate from (X1, X3, X5 | X1 = -1, X4 = 1)
# and (X1, X3, X5 | X1 = 2, X4 = 3)
given_x = rbind(c(-1, 1), c(2, 3))
rmnorm(n = nrow(given_x), mean = mean, sigma = sigma,
       given_x = given_x, given_ind = given_ind)
```

---

seqPrimes — *Sequence of prime numbers*

Description

Calculates the sequence of prime numbers.

Usage

```
seqPrimes(n)
```

Arguments

n	positive integer representing the number of sequence elements.

Value

The function returns a numeric vector containing first n prime numbers. The current (naive) implementation of the algorithm is not efficient in terms of speed so it is suited for low n < 10000 but requires just O(n) memory usage.

Examples

```
seqPrimes(10)
```

---

stdt — *Standardized Student t Distribution*

Description

These functions calculate and differentiate a cumulative distribution function and density function of the standardized (to zero mean and unit variance) Student distribution. Quantile function and random numbers generator are also provided.

Usage

```
dt0(x, df = 10, log = FALSE, grad_x = FALSE, grad_df = FALSE)

pt0(x, df = 10, log = FALSE, grad_x = FALSE, grad_df = FALSE, n = 10L)

rt0(n = 1L, df = 10)

qt0(x = 1L, df = 10)
```

Arguments

`x`	numeric vector of quantiles.
`df`	positive real value representing the number of degrees of freedom. Since this function deals with standardized Student distribution, argument `df` should be greater than `2` because otherwise variance is undefined.
`log`	logical; if `TRUE` then probabilities (or densities) p are given as log(p) and derivatives will be given respect to log(p).
`grad_x`	logical; if `TRUE` then function returns a derivative respect to `x`.
`grad_df`	logical; if `TRUE` then function returns a derivative respect to `df`.
`n`	positive integer. If `rt0` function is used then this argument represents the number of random draws. Otherwise `n` states for the number of iterations used to calculate the derivatives associated with `pt0` function via `pbetaDiff` function.

Details

Standardized (to zero mean and unit variance) Student distribution has the following density and cumulative distribution functions:

$$f(x) = \frac{\Gamma\left(\frac{v+1}{2}\right)}{\sqrt{(v-2)\pi}\Gamma\left(\frac{v}{2}\right)}\left(1+\frac{x^2}{v-2}\right)^{-\frac{v+1}{2}},$$

$$F(x) = \begin{cases} 1-\frac{1}{2}I(\frac{v-2}{x^2+v-2}, \frac{v}{2}, \frac{1}{2}), \text{if } x \geq 0 \\ \frac{1}{2}I(\frac{v-2}{x^2+v-2}, \frac{v}{2}, \frac{1}{2}), \text{if } x < 0 \end{cases},$$

where $v > 2$ is the number of degrees of freedom `df` and $I(.)$ is a cumulative distribution function of beta distribution which is calculated by `pbeta` function.

Value

Function `rt0` returns a numeric vector of random numbers. Function `qt0` returns a numeric vector of quantiles. Functions `pt0` and `dt0` return a list which may contain the following elements:

- `prob` - numeric vector of probabilities calculated for each element of `x`. Exclusively for `pt0` function.
- `den` - numeric vector of densities calculated for each each element of `x`. Exclusively for `dt0` function.

- grad_x - numeric vector of derivatives respect to p for each element of x. This element appears only if input argument grad_x is TRUE.
- grad_df - numeric vector of derivatives respect to q for each element of x. This element appears only if input argument grad_df is TRUE.

Examples

```
# Simple examples
pt0(x = 0.3, df = 10, log = FALSE, grad_x = TRUE, grad_df = TRUE)
dt0(x = 0.3, df = 10, log = FALSE, grad_x = TRUE, grad_df = TRUE)
qt0(x = 0.3, df = 10)

# Compare analytical and numeric derivatives
delta <- 1e-6
x <- c(-2, -1, 0, 1, 2)
df <- 5

# For probabilities
out <- pt0(x, df = df, grad_x = TRUE, grad_df = TRUE)
p0 <- out$prob
  # grad_x
p1 <- pt0(x + delta, df = df)$prob
data.frame(numeric = (p1 - p0) / delta, analytical = out$grad_x)
  # grad_df
p1 <- pt0(x, df = df + delta)$prob
data.frame(numeric = (p1 - p0) / delta, analytical = out$grad_df)

# For densities
out <- dt0(x, df = df, grad_x = TRUE, grad_df = TRUE)
p0 <- out$den
  # grad_x
p1 <- dt0(x + delta, df = df)$den
data.frame(numeric = (p1 - p0) / delta, analytical = out$grad_x)
  # grad_df
p1 <- dt0(x, df = df + delta)$den
data.frame(numeric = (p1 - p0) / delta, analytical = out$grad_df)
```

---

toBase *Convert integer value to other base*

Description

Converts integer value to other base.

Usage

```
toBase(x, base = 2L)
```

Arguments

`x`	positive integer representing the number to convert.
`base`	positive integer representing the base.

Value

The function returns a numeric vector containing representation of `x` in a base given in `base`.

Examples

```
toBase(888, 5)
```

Index

cmnorm, 2, *6*, *16*, *25*

dmnorm, 5, *16*, *17*
dt0 *(*stdt*)*, 26

fromBase, 10

halton, 11

pbeta, *12*, *27*
pbetaDiff, 12, *27*
phpa0, *15*
pmnorm, 13
pt0 *(*stdt*)*, 26

qnormFast, 23
qt0 *(*stdt*)*, 26

rmnorm, 24
rt0 *(*stdt*)*, 26

seqPrimes, *12*, 26
stdt, 26

toBase, 28