Package 'mnorm'

April 14, 2026

Type Package

Title Multivariate Normal Distribution

Version 1.2.3

Date 2026-04-15

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 (>= 1.1.1), hpa (>= 1.3.4)

LinkingTo Rcpp, RcppArmadillo, hpa

RoxygenNote 7.3.2

NeedsCompilation yes

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

Maintainer Bogdan Potanin <bogdanpotanin@gmail.com>

Repository CRAN

Date/Publication 2026-04-14 16:00:02 UTC

Contents

`cmnorm` *Parameters of the conditional multivariate normal distribution*

Description

This function calculates the mean (expectation) and the covariance matrix of the 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 an expectation of the multivariate normal vector (distribution).
`sigma`	positively defined numeric matrix representing the covariance matrix of the multivariate normal vector (distribution).
`given_ind`	numeric vector representing indexes of a multivariate normal vector which are conditioned on the values given by the `given_x` argument.
`given_x`	numeric vector whose `i`-th element corresponds to the given value of the `given_ind[i]`-th element (component) of a multivariate normal vector. If `given_x` is a numeric matrix then its rows are such vectors of given values.
`dependent_ind`	numeric vector representing indexes of the unconditioned elements (components) of a multivariate normal vector.
`is_validation`	logical value indicating whether input arguments should be validated. Set it to `FALSE` to get a 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 a 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 than 100000 elements.

Details

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

Let's denote the vectors of indexes of the conditioned and unconditioned elements of X by I_g and I_d respectively. By $x^{(g)}$ denote a deterministic (column) vector of given values of X_{I_g}. The function calculates the expected value and the 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 the 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 the intersection of I_A rows and the 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 one 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 have the same length as `mean`; thus, 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` - a conditional mean.
- `sigma` - a conditional covariance matrix.
- `sigma_d` - a covariance matrix of the unconditioned elements.
- `sigma_g` - a covariance matrix of the conditioned elements.
- `sigma_dg` - a matrix of the covariances between the unconditioned and conditioned elements.
- `s12s22` - equals 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 does 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 a vector. Conversely, if `given_x` is a matrix, then output `mean` is a matrix whose rows correspond to the conditional means associated with the given values provided by the corresponding rows of `given_x`.

The order of the elements of output `mean` and output `sigma` depends on the order of `dependent_ind` elements (which is ascending by default). The order of `given_ind` elements does not matter. However, please check that the order of `given_ind` matches the order of given values, i.e., the order of `given_x` columns.

Examples

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

# Prepare the multivariate normal vector parameters
  # the expected value
mean <- c(-2, -1, 0, 1, 2)
n_dim <- length(mean)
  # the 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)
  # the covariance matrix
sd_mat <- diag(c(1, 1.5, 2, 2.5, 3))
sigma <- sd_mat %*% cor %*% t(sd_mat)

# Estimate the parameters of the 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 the 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 the (conditional) multivariate normal distribution*

Description

This function calculates and differentiates the density of the (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 the density should be calculated. If x is a matrix, then each row determines a new point.
mean	numeric vector representing an expectation of the multivariate normal vector (distribution).
sigma	positively defined numeric matrix representing the covariance matrix of the multivariate normal vector (distribution).
given_ind	numeric vector representing indexes of a multivariate normal vector which are conditioned on the values of x with the corresponding indexes, i.e., x[given_ind] or x[, given_ind] if x is a matrix.
log	logical; if TRUE, then the probabilities (or densities) p are given as log(p), and the derivatives will be given with respect to log(p).
grad_x	logical; if TRUE, then the vector of partial derivatives of the density function will be calculated with respect to each element of x. If x is a matrix, then the 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 with respect to each element of sigma. If x is a matrix, then the 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 a 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 than 100000 elements.

Details

Consider the notations from the Details section of cmnorm. The function calculates the 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., the unconditioned components. Therefore x[given_ind] represents $x^{(g)}$, while x[-given_ind] is $x^{(d)}$.

If grad_x is TRUE, then the function additionally estimates the gradient with respect to both the 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 $i \in I_g$. In particular, the subgradients of the density function with 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 the function additionally estimates the gradient with respect to the elements of covariance matrix Σ. For $i \in I_d$ and $j \in I_d$, the function calculates:

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

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:

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

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

where $q_g(j) = \sum\limits_{k=1}^{m} I\left(I_{g,k} \leq j\right)$ and $q_d(i) = \sum\limits_{k=1}^{m} I\left(I_{d,k} \leq i\right)$ represent the order of the i-th and j-th elements in I_g and I_d, respectively, i.e., $x_i = x^{(d)}_{q_d(i)} = x_{I_{d,q_d(i)}}$ and $x_j = x^{(g)}_{q_g(j)} = x_{I_{g,q_g(j)}}$. Note that $q_g(j)^{-1}$ and $q_d(i)^{-1}$ are inverse functions. The number of conditioned and unconditioned components is denoted by $n_g = \sum\limits_{k=1}^{m} I(k \in I_g)$ and $n_d = \sum\limits_{k=1}^{m} I(k \in I_d)$ 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:

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

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

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 aforementioned 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` - the density function value at `x`.
- `grad_x` - a gradient of the density with respect to `x`, if the `grad_x` or `grad_sigma` input argument is set to `TRUE`.

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

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

Output grad_x is a Jacobian matrix whose rows are the gradients of the density function calculated for each row of x. Therefore, grad_x[i, j] is a derivative of the density function with respect to the j-th argument at the 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 with 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 a multivariate normal vector:
# X = (X1, X2, X3, X4, X5) ~ N(mean, sigma)

# Prepare the multivariate normal vector parameters
  # the expected value
mean <- c(-2, -1, 0, 1, 2)
n_dim <- length(mean)
  # the 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)
  # the 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 the 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=(-1.5, -1.2, 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
```

```
# the point xd=(0, 2, 3) given the 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 the density with 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 with respect to the argument
grad_x.3 <- d.list.3$grad_x
   # at the point 'x'
print(grad_x.3[1, ])
   # at the point 'y'
print(grad_x.3[2, ])
# Partial derivative at the point 'y' with respect
# to the 3-rd argument
print(grad_x.3[2, 3])
# Gradient respect to the covariance matrix
grad_sigma.3 <- d.list.3$grad_sigma
# Partial derivative with 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)
# with respect to the argument and covariance matrix at
# points xd=(0, 2, 3) and yd=(-1.2, 0, 1.5), respectively, given
# the 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$grad_x
   # at point 'xd'
print(grad_x.4[1, ])
   # at point 'yd'
print(grad_x.4[2, ])
# Partial derivative at the point 'xd' with respect to 'xg[2]'
print(grad_x.4[1, 3])
# Partial derivative at the point 'yd' with respect to 'yd[5]'
print(grad_x.4[2, 5])
# Gradient with respect to the covariance matrix
grad_sigma.4 <- d.list.4$grad_sigma
# Partial derivative with 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 the 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 with 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 with respect to the covariance matrix
h.sigma <- list(analytical = grad_sigma.4[, , 1], numeric = grad_sigma.num)
print(h.sigma)
```

`fromBase` *Convert a base representation of a number into an integer*

Description

Converts a base representation of a number into an integer.

Usage

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

Arguments

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

Value

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

Examples

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

`halton` *Halton sequence*

Description

Calculate the 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 than 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 a standard uniform random variable will be added to each element of the sequence, and the difference between this sum and its 'floor' will be returned as a new element of the sequence.
`type`	string representing a type of the sequence. Default is "halton" that is the Halton sequence. The alternative is "richtmyer" corresponding to the Richtmyer sequence.
`scrambler`	string representing a scrambling method for the Halton sequence. Possible options are `"NO"` (default), `"root"` and `"negroot"`, which are described in S. Kolenikov (2012).
`is_validation`	logical value indicating whether input arguments should be validated. Set it to `FALSE` to get a 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 than 100000 elements.

Details

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

Value

The function returns a matrix whose i-th column is a sequence with the base base[i] and the 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 the 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 the q argument of the pbeta function.
p	similar to the shape1 argument of the pbeta function.
q	similar to the shape2 argument of the pbeta function.
n	positive integer representing the number of iterations used to calculate the derivatives. Greater values provide higher accuracy at 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 the control parameters. Currently not intended for the users.

Details

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

Value

The function returns a list which has the following elements:

- dx - a numeric vector of the derivatives with respect to each element of x.
- dp - a numeric vector of the derivatives with respect to p for each element of x.
- dq - a numeric vector of the derivatives with 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 with respect to x
p1 <- pbeta(q = x + delta, shape1 = p, shape2 = q)
data.frame(numeric = (p1 - p0) / delta, analytical = out$dx)

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

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

---

pmnorm *Probabilities of the (conditional) multivariate normal distribution*

Description

This function calculates and differentiates the probabilities of the (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 the 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 the 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 whose `i`-th element corresponds to the given value of the `given_ind[i]`-th element (component) of a multivariate normal vector. If `given_x` is a numeric matrix then its rows are such vectors of given values.
`mean`	numeric vector representing an expectation of the multivariate normal vector (distribution).
`sigma`	positively defined numeric matrix representing the covariance matrix of the multivariate normal vector (distribution).
`given_ind`	numeric vector representing indexes of a multivariate normal vector which are conditioned on the values given by the `given_x` argument.
`n_sim`	positive integer representing the number of draws from the Richtmyer sequence in the GHK algorithm. More draws provide more accurate results at the cost of additional computational burden. Alternative types of sequences could be provided via the `random_sequence` argument.

`method` string representing the method to be used to calculate the multivariate normal probabilities. Possible options are `"GHK"` and `"Gassmann"` recommended for high dimensional (more than 5) and low dimensional probabilities, respectively. An 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 the probabilities (or densities) p are given as log(p), and the derivatives will be given with respect to log(p).

`grad_lower` logical; if `TRUE`, then the vector of partial derivatives of the probability will be calculated with 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 with respect to each element of `upper`. If `upper` is a matrix, then the 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 with respect to each element of `sigma`. If `lower` and `upper` are matrices, then the 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 with respect to each element of `given_x`. If `given_x` is a matrix, then the 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 a 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 than 100000 elements.

`marginal` list such that `marginal[[i]]` represents the parameters of the marginal distribution of the `i`-th component of the random vector, and `names(marginal)[i]` is the 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 whose elements correspond to the `pc` argument of `phpa0`.

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

`grad_marginal_prob` logical; if `TRUE`, then the vector of partial derivatives (gradient) of the probability will be calculated with respect to each cumulative marginal probability of the 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. 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 require some additional transformations of (1). During preliminary testing it has been found that the approach with 30 points provides similar accuracy being slightly faster because of better vectorization capabilities.

To calculate trivariate probabilities, the function uses the 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 the matrix 5 representation of Gassmann (2003) and 30 points of Gauss-Legendre quadrature.

For m-variate probabilities, where $m > 1$, the function may apply the GHK algorithm described in Section 4.2 of A. Genz and F. Bretz (2009). The implementation of GHK is based on the 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 the ordering argument. Available options are "NO", "mean" (default), and "variance".

Univariate probabilities are always calculated via the 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 the 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 the 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 infinity. The derivatives could also be calculated much faster when some integration limits are infinite.

For simplicity of the notations, further, let's consider unconditioned probabilities. Derivatives with respect to conditioned components are similar to those mentioned in the 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 the function additionally estimates the gradient with 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 with 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 the integration limits are infinite, then some elements of this equation converge to zero, which highly simplifies the calculations.

Derivatives with respect to variances are calculated using derivatives with 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 the aforementioned differentiation formulas can 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 standardized (to zero mean and unit variance) marginal distribution whose name and parameters are defined by `names(marginal)[i]` and `marginal[[i]]`, respectively. For example, if `marginal[i]` = `"logistic"`, then:

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

Let's denote by X^* a random vector which is distributed with Gaussian (its covariance matrix is Σ) copula and marginals defined by `marginal`. Then probabilities for this 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 $\sigma q^{(u)}$, respectively. 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, then 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` - the probability that a multivariate normal random variable will be between the `lower` and `upper` bounds.
- `grad_lower` - the gradient of the probability with respect to `lower`, if `grad_lower` or `grad_sigma` the input argument is set to `TRUE`.
- `grad_upper` - the gradient of the probability with respect to `upper`, if `grad_upper` or `grad_sigma` input argument is set to `TRUE`.
- `grad_sigma` - the gradient with respect to the elements of `sigma`, if `grad_sigma` input argument is set to `TRUE`.
- `grad_given` - the gradient with respect to the elements of `given_x`, if `grad_given` input argument is set to `TRUE`.
- `grad_marginal` - the gradient with respect to the elements of `marginal_par`, if `grad_marginal` input argument is set to `TRUE`. Currently only the derivatives with respect to the parameters of the `"PGN"` distribution are available.

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

The output grad_lower and grad_upper are Jacobian matrices whose rows are the gradients of the probabilities calculated for each row of lower and upper, respectively. Similarly, grad_given is a Jacobian matrix with respect to given_x.

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

Output grad_marginal is a list such that grad_marginal[[i]] is a Jacobian matrix whose rows are the gradients of the probabilities calculated for each row of lower and upper, respectively, with 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 a multivariate normal vector:
# X = (X1, X2, X3, X4, X5) ~ N(mean, sigma)

# Prepare the multivariate normal vector parameters
  # the expected value
mean <- c(-2, -1, 0, 1, 2)
n_dim <- length(mean)
  # the 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)
  # the 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
# P(-Inf < X1 < Inf, 0 < X2 < Inf, -Inf < X3 < 3, 1 < X4 < 4, -Inf < X5 < 5)
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 the previous probabilities with respect to
# 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$prob
print(p.4)
# Gradient with respect to 'lower'
grad_lower <- p.list.4$grad_lower
   # for the first probability
print(grad_lower[1, ])
   # for the second probability
```

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

# 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)
{
  # with 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
  # with 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)
{
  # with 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)
  {
    # with 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 with respect to the lower integration limits
h.lower <- cbind(analytical = p.list.4$grad_lower[1, ],
                 numeric = grad_lower.num)
print(h.lower)
# Comparison of gradients with respect to the upper integration limits
h.upper <- cbind(analytical = p.list.4$grad_upper[1, ],
                 numeric = grad_upper.num)
print(h.upper)
# Comparison of gradients with respect to the given values
h.given <- cbind(analytical = p.list.4$grad_given[1, ],
                 numeric = grad_given.num)
print(h.given)
# Comparison of gradients with respect to the covariance matrix
h.sigma <- list(analytical = p.list.4$grad_sigma[, , 1],
                numeric = grad_sigma.num)
print(h.sigma)

# Let's again estimate the probability
# P(-1 < X1 < 1, -3 < X3 < 3, -5 < X5 < 5 | X2 = -2, X4 = 4)
# But assume that the variables are 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), respectively.
# 2) X3 has a standardized Student distribution with 5 degrees of freedom
# 3) X4 has a standardized logistic distribution
# 4) X5 has a 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)
```

```
# Let's investigate derivatives with respect to the parameters
# of marginal distributions
  # with respect to pc1 of X1
p.list.5$grad_marginal[[1]]
  # with respect to pc2 of X2
p.list.5$grad_marginal[[2]]
  # derivative with respect to the degrees of freedom of X5 is
  # currently unavailable and will be set to 0
p.list.5$grad_marginal[[3]]
```

`qnormFast` *The quantile function of a normal distribution*

Description

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

Usage

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

Arguments

`p`	numeric vector of values between 0 and 1 representing the levels of the quantiles.
`mean`	numeric value representing the expectation of a normal distribution.
`sd`	positive numeric value representing the standard deviation of a normal distribution.
`method`	character representing the method to be used for the 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 a 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 than 100000 elements.

Details

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

Value

The function returns a vector of p-level quantiles of a normal distribution with the mean equal to `mean` and the 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 the (conditional) multivariate normal distribution*

Description

This function generates random numbers (i.e. variates) from the (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 the (conditional) multivariate normal distribution. If `given_ind` is not an empty vector, then `n` should be equal to `nrow(given_x)`.
`mean`	numeric vector representing an expectation of the multivariate normal vector (distribution).
`sigma`	positively defined numeric matrix representing the covariance matrix of the multivariate normal vector (distribution).
`given_ind`	numeric vector representing indexes of a multivariate normal vector which are conditioned on the values given by the `given_x` argument.

given_x	numeric vector whose i-th element corresponds to the given value of the given_ind[i]-th element (component) of a multivariate normal vector. If given_x is a numeric matrix then its rows are such vectors of given values.
dependent_ind	numeric vector representing indexes of the unconditioned elements (components) of a multivariate normal vector.
is_validation	logical value indicating whether input arguments should be validated. Set it to FALSE to get a 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 than 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 whose rows are random variates from the (conditional) multivariate normal distribution with the mean equal to mean and the covariance equal to sigma. If given_x and given_ind are also provided, then random variates will be from the conditional multivariate normal distribution. Please, see the details section of cmnorm to get additional information on the conditioning procedure.

Examples

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

# Prepare the multivariate normal vector parameters
  # the expected value
mean <- c(-2, -1, 0, 1, 2)
n_dim <- length(mean)
  # the 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)
  # the 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 the 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 the density function of the standardized (to zero mean and unit variance) Student distribution. Quantile function and random number 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 the standardized Student distribution, the argument df should be greater than 2 because otherwise the variance is undefined.
log	logical; if TRUE, then probabilities (or densities) p are given as log(p) and the derivatives will be given with respect to log(p).
grad_x	logical; if TRUE, then the function returns the derivative with respect to x.
grad_df	logical; if TRUE, then the function returns the derivative with respect to df.
n	positive integer. If the rt0 function is used then this argument represents the number of random draws. Otherwise, n stands for the number of iterations used to calculate the derivatives associated with the pt0 function via the pbetaDiff function.

Details

The 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 the cumulative distribution function of the beta distribution which is calculated by the pbeta function.

Value

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

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

- grad_x - a numeric vector of the derivatives with respect to p for each element of x. This element appears only if the input argument grad_x is TRUE.
- grad_df - a numeric vector of the derivatives with respect to p for each element of x. This element appears only if the 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 an integer value to another base*

Description

Converts an integer value to another 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 the representation of x in a base given in base.

Examples

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

Index