Package 'mnorm'

October 13, 2022

Type Package

Title Multivariate Normal Distribution

Version 1.0.1

Date 2022-05-14

Description Calculates and differentiates probabilities and density of (conditional) multivariate normal distribution 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> and E. Kossova, B. Potanin (2018) <https:ideas.repec.org/a/ris/apltrx/0346.html>.

License GPL (>= 2)

Imports Rcpp (>= 1.0.6)

LinkingTo Rcpp, RcppArmadillo

RoxygenNote 7.2.0

NeedsCompilation yes

Author Bogdan Potanin [aut, cre, ctb]

Maintainer Bogdan Potanin <bogdanpotanin@gmail.com>

Repository CRAN

Date/Publication 2022-05-16 07:10:02 UTC

R topics documented:

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 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:

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

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

Note that $\Sigma_{(I_A,I_B)}$, where $A, B \in \{I_d, I_g\}$, 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 $j \in I_g$ the function calculates:

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

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_g 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:

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

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

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)
```

`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",
  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.
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>

Examples

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

---

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
)
```

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 Halton sequence in GHK algorithm. More draws provide more accurate results by the cost of additional computational burden.
`method`	string representing the method to be used to calculate multivariate normal probabilities. Currently `"default"` is the only available option. See Details section below.
`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.

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 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.

For m-variate probabilities, where $m > 2$, the function applies 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 use 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".

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

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

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:

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

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

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

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).

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_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`.

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.

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.

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
# and increase the accuracy of estimates by increasing the number
# of Halton sequence draws
```

```
n_sim <- 5000
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,
                   n_sim = n_sim)
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,
                        n_sim = n_sim)
  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,
                        n_sim = n_sim)
  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,
                        n_sim = n_sim)
  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,
                           n_sim = n_sim)
    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)
```

---

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)
```

`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)
```

Index