

# Package ‘mvnfast’

August 21, 2014

**Type** Package

**Title** Fast multivariate normal methods

**Version** 0.1.3

**Date** 2014-06-10

**Author** Matteo Fasiolo, using the C++ parallel RNG of Thijs van den Berg and Ziggurat algorithm of Jens Maurer and Steven Watanabe (boost)

**Maintainer** Matteo Fasiolo <matteo.fasiolo@gmail.com>

**Description** The package provides computationally efficient tools related to the multivariate normal distribution. The main functionalities are: simulating multivariate normal random vectors, evaluating multivariate normal densities and Mahalanobis distances. These tools are very efficient thanks to the use of C++ code and of the OpenMP API.

**License** GPL (>= 2.0)

**URL** <https://github.com/mfasiolo/mvnfast>, [www.sitmo.com](http://www.sitmo.com)

**Imports** Rcpp (>= 0.10.4)

**Suggests** knitr, testthat, mvtnorm, microbenchmark, MASS, plyr

**LinkingTo** Rcpp, RcppArmadillo, BH

**VignetteBuilder** knitr

**NeedsCompilation** yes

**Repository** CRAN

**Date/Publication** 2014-08-21 00:24:05

## R topics documented:

dmvn . . . . .	2
maha . . . . .	3
ms . . . . .	4
rmvn . . . . .	6

---

dmvn *Fast computation of the multivariate normal density.*

---

### Description

Fast computation of the multivariate normal density.

### Usage

```
dmvn(X, mu, sigma, log = FALSE, ncores = 1, isChol = FALSE)
```

### Arguments

X	matrix n by d where each row is a d dimensional random vector. Alternatively X can be a d-dimensional vector.
mu	vector of length d, representing the mean the distribution.
sigma	covariance matrix (d x d). Alternatively is can be the cholesky decomposition of the covariance. In that case isChol should be set to TRUE.
log	boolean set to true the logarithm of the pdf is required.
ncores	Number of cores used. The parallelization will take place only if OpenMP is supported.
isChol	boolean set to true is sigma is the cholesky decomposition of the covariance matrix.

### Value

a vector of length n where the i-the entry contains the pdf of the i-th random vector.

### Author(s)

Matteo Fasiolo <matteo.fasiolo@gmail.com>

### Examples

```
N <- 100
d <- 5
mu <- 1:d
X <- t(t(matrix(rnorm(N*d), N, d)) + mu)
tmp <- matrix(rnorm(d^2), d, d)
mcov <- tcrossprod(tmp, tmp)
myChol <- chol(mcov)

head(dmvn(X, mu, mcov), 10)
head(dmvn(X, mu, myChol, isChol = TRUE), 10)

## Not run:
```

```

# Performance comparison
library(mvtnorm)
library(microbenchmark)

a <- cbind(
  dmvn(X, mu, mcov),
  dmvn(X, mu, myChol, isChol = TRUE),
  dmvnorm(X, mu, mcov))

# Check if we get the same output as dmvnorm()
a[, 1] / a[, 3]
a[, 2] / a[, 3]

microbenchmark(dmvn(X, mu, myChol, isChol = TRUE),
  dmvn(X, mu, mcov),
  dmvnorm(X, mu, mcov))

## End(Not run)

```

---

maha	<i>Fast computation of squared mahalanobis distance between all rows of X and the vector mu with respect to sigma.</i>
------	------------------------------------------------------------------------------------------------------------------------

---

## Description

Fast computation of squared mahalanobis distance between all rows of X and the vector mu with respect to sigma.

## Usage

```
maha(X, mu, sigma, ncores = 1, isChol = FALSE)
```

## Arguments

X	matrix n by d where each row is a d dimensional random vector. Alternatively X can be a d-dimensional vector.
mu	vector of length d, representing the central position.
sigma	covariance matrix (d x d). Alternatively is can be the cholesky decomposition of the covariance. In that case isChol should be set to TRUE.
ncores	Number of cores used. The parallelization will take place only if OpenMP is supported.
isChol	boolean set to true is sigma is the cholesky decomposition of the covariance matrix.

## Value

a vector of length n where the i-th entry contains the square mahalanobis distance i-th random vector.

**Author(s)**

Matteo Fasiolo <matteo.fasiolo@gmail.com>

**Examples**

```

N <- 100
d <- 5
mu <- 1:d
X <- t(t(matrix(rnorm(N*d), N, d)) + mu)
tmp <- matrix(rnorm(d^2), d, d)
mcov <- tcrossprod(tmp, tmp)
myChol <- chol(mcov)

rbind(head(maha(X, mu, mcov), 10),
      head(maha(X, mu, myChol, isChol = TRUE), 10),
      head(mahalanobis(X, mu, mcov), 10))

## Not run:
# Performance comparison
library(microbenchmark)

a <- cbind(
  maha(X, mu, mcov),
  maha(X, mu, myChol, isChol = TRUE),
  mahalanobis(X, mu, mcov))

# Same output as mahalanobis
a[ , 1] / a[ , 3]
a[ , 2] / a[ , 3]

microbenchmark(maha(X, mu, mcov),
               maha(X, mu, myChol, isChol = TRUE),
               mahalanobis(X, mu, mcov))

## End(Not run)

```

---

ms

*Mean-shift mode seeking algorithm*


---

**Description**

Given a sample from a d-dimensional distribution, an initialization point and a bandwidth the algorithm finds the nearest mode of the corresponding Gaussian kernel density.

**Usage**

```
ms(X, init, H, tol = 1e-06, ncores = 1, store = FALSE)
```

**Arguments**

<code>X</code>	n by d matrix containing the data.
<code>init</code>	d-dimensional vector containing the initial point for the optimization. By default it is equal to <code>colMeans(X)</code> .
<code>H</code>	Positive definite bandwidth matrix representing the covariance of each component of the Gaussian kernel density.
<code>tol</code>	Tolerance used to assess the convergence of the algorithm, which is stopped if the absolute values of increments along all the dimensions are smaller then <code>tol</code> at any iteration. Default value is <code>1e-6</code> .
<code>ncores</code>	Number of cores used. The parallelization will take place only if OpenMP is supported.
<code>store</code>	If <code>FALSE</code> only the latest iteration is returned, if <code>TRUE</code> the function will return a matrix where the i-th row is the position of the algorithms at the i-th iteration.

**Value**

A list where `estim` is a d-dimensional vector containing the last position of the algorithm, while `traj` is a matrix with d-columns representing the trajectory of the algorithm along each dimension. If `store == FALSE` the whole trajectory is not stored and `traj = NULL`.

**Author(s)**

Matteo Fasiolo <matteo.fasiolo@gmail.com>.

**Examples**

```
set.seed(434)

# Simulating multivariate normal data
N <- 1000
mu <- c(1, 2)
sigma <- matrix(c(1, 0.5, 0.5, 1), 2, 2)
X <- rmvN(N, mu = mu, sigma = sigma)

# Plotting the true density function
steps <- 100
range1 <- seq(min(X[, 1]), max(X[, 1]), length.out = steps)
range2 <- seq(min(X[, 2]), max(X[, 2]), length.out = steps)
grid <- expand.grid(range1, range2)
vals <- dmvn(as.matrix(grid), mu, sigma)

contour(z = matrix(vals, steps, steps), x = range1, y = range2, xlab = "X1", ylab = "X2")
points(X[, 1], X[, 2], pch = '.')

# Estimating the mode from "nrep" starting points
nrep <- 10
index <- sample(1:N, nrep)
for(ii in 1:nrep) {
  start <- X[index[ii], ]
}
```

```

out <- ms(X, init = start, H = 0.1 * sigma, store = TRUE)
lines(out$traj[ , 1], out$traj[ , 2], col = 2, lwd = 2)
points(out$final[1], out$final[2], col = 4, pch = 3, lwd = 3) # Estimated mode (blue)
points(start[1], start[2], col = 2, pch = 3, lwd = 3)      # ii-th starting value
}

```

---

rmvn

*Fast simulations of multivariate normal random variables*


---

## Description

Fast simulations of multivariate normal random variables

## Usage

```
rmvn(n, mu, sigma, ncores = 1, isChol = FALSE)
```

## Arguments

n	number of random vectors to be simulated.
mu	vector of length d, representing the central position.
sigma	covariance matrix (d x d). Alternatively it can be the cholesky decomposition of the covariance. In that case isChol should be set to TRUE.
ncores	Number of cores used. The parallelization will take place only if OpenMP is supported.
isChol	boolean set to true if sigma is the cholesky decomposition of the covariance matrix.

## Details

Notice that this function does not use one of the Random Number Generators (RNGs) provided by R, but one of the parallel cryptographic RNGs described in (Salmon et al., 2011). It is important to point out that this RNG can safely be used in parallel, without risk of collisions between parallel sequence of random numbers. The initialization of the RNG depends on R's seed, hence the set .seed() function can be used to obtain reproducible results. Notice though that changing ncores causes most of the generated numbers to be different even if R's seed is the same (see example below). N.B. At the moment the RNG does not work properly on Solaris OS when ncores>1. Hence, rmvn() checks if the OS is Solaris and, if this the case, it imposes ncores==1.

## Value

A vector of length n where the i-th entry contains the square mahalanobis distance i-th random vector.

## Author(s)

Matteo Fasiolo <matteo.fasiolo@gmail.com>, C++ RNG engine by Thijs van den Berg <<http://sitmo.com/>>.

**References**

John K. Salmon, Mark A. Moraes, Ron O. Dror, and David E. Shaw (2011). Parallel Random Numbers: As Easy as 1, 2, 3. D. E. Shaw Research, New York, NY 10036, USA.

**Examples**

```
d <- 5
mu <- 1:d

# Creating covariance matrix
tmp <- matrix(rnorm(d^2), d, d)
mcov <- tcrossprod(tmp, tmp)

set.seed(414)
rmvn(4, 1:d, mcov)

set.seed(414)
rmvn(4, 1:d, mcov)

set.seed(414)
rmvn(4, 1:d, mcov, ncores = 2) # r.v. generated on the second core are different
```

# Index

dmvn, 2

maha, 3

ms, 4

rmvn, 6