Package 'nvmix'

October 14, 2018

Version 0.0-1

Encoding UTF-8

Title Multivariate Normal Variance Mixtures (Including Student's t Distribution for Non-Integer Degrees of Freedom)

Description Functions for working with multivariate normal variance mixture distributions including evaluating their distribution functions, densities and random number generation.

Author Marius Hofert [aut, cre],
Erik Hintz [aut],
Christiane Lemieux [aut]

Maintainer Marius Hofert <marius.hofert@uwaterloo.ca>

Depends R (>= 3.2.0)

Imports stats, methods, qrng

Suggests knitr, RColorBrewer, lattice

Enhances

License GPL (>= 3) | file LICENCE

NeedsCompilation yes

VignetteBuilder knitr

Repository CRAN

Date/Publication 2018-10-14 17:20:03 UTC

R topics documented:

dnvmix *Density of Multivariate Normal Variance Mixtures*

Description

Evaluating multivariate normal variance mixture densities (including Student t and normal densities).

Usage

```
dnvmix(x, qmix, loc = rep(0, d), scale = diag(d),
       factor = NULL, method = c("sobol", "ghalton", "PRNG"),
       abstol = 0.001, CI.factor = 3.3, fun.eval = c(2^6, 1e8), B = 12,
       log = FALSE, verbose = TRUE, ...)

dStudent(x, df, loc = rep(0, d), scale = diag(d), factor = NULL,
         log = FALSE, verbose = TRUE, ...)
dNorm(x, loc = rep(0, d), scale = diag(d), factor = NULL,
      log = FALSE, verbose = TRUE, ...)
```

Arguments

x	(n, d)-matrix of evaluation points.
qmix	specification of the mixing variable W; see McNeil et al. (2015, Chapter 6). Supported are the following types of specification (see also the examples below):
	character: character string specifying a supported mixing distribution; currently available are "constant" (in which case $W = 1$ and thus the multivariate normal distribution with mean vector loc and covariance matrix scale results) and "inverse.gamma" (in which case W is inverse gamma distributed with shape and rate parameters df/2 and thus the multivariate Student t distribution with df degrees of freedom (required to be provided via the ellipsis argument) results).
	list: list of length at least one, where the first component is a character string specifying the base name of a distribution whose quantile function can be accessed via the prefix "q"; an example is "exp" for which "qexp" exists. If the list is of length larger than one, the remaining elements contain additional parameters of the distribution; for "exp", for example, this can be the parameter rate.
	function: function interpreted as the quantile function of the mixing variable W.
df	positive degress of freedom; can also be Inf in which case the distribution is interpreted as the multivariate normal distribution with mean vector loc and covariance matrix scale).
loc	location vector of dimension d; this equals the mean vector of a random vector following the specified normal variance mixture distribution if and only if the latter exists.

`scale` scale matrix (a covariance matrix entering the distribution as a parameter) of dimension (d, d); this equals the covariance matrix of a random vector following the specified normal variance mixture distribution divided by the expecation of the mixing variable W if and only if the former exists.

`factor` (d, d) lower triangular `matrix` such that `factor %*% t(factor)` equals `scale`; note that for performance reasons, this property is not tested. If not provided, `factor` is internally determined via `t(chol())`.

`method` `character` string indicating the method to be used to compute the integral. Available are:

`"sobol"`: Sobol' sequence

`"ghalton"`: generalized Halton sequence

`"PRNG"`: plain Monte Carlo based on a pseudo-random number generator

`abstol` non-negative `numeric` providing the absolute precision required. If `abstol = 0`, the algorithm will typically run until the total number of function evaluations exceeds `fun.eval[2]`. If $n > 1$ (so `x` has more than one row), the algorithm runs until the precision requirement is reached for all n density estimates.

`CI.factor` multiplier of the Monte Carlo confidence interval bounds. The algorithm runs until `CI.factor` times the estimated standard error is less than `abstol`. If `CI.factor = 3.3` (the default), one can expect the actual absolute error to be less than `abstol` in 99.9% of the cases.

`fun.eval` `numeric(2)` providing the size of the first point set to be used to estimate the probabilities (typically a power of 2) and the maximal number of function evaluations.

`B` number of randomizations for obtaining an error estimate in the randomized quasi-Monte Carlo (RQMC) approach.

`log` `logical` indicating whether the logarithmic density is to be computed.

`verbose` `logical` indicating whether a warning is given if the required precision `abstol` has not been reached.

`...` additional arguments (for example, parameters) passed to the underlying mixing distribution when `qmix` is a `character` string or `function`.

Details

Internally used is `factor`, so `scale` is not required to be provided if `factor` is given.

The default factorization used to obtain `factor` is the Cholesky decomposition via `chol()`. To this end, `scale` needs to have full rank.

The number of rows of `factor` equals the dimension d of the sample. Typically (but not necessarily), `factor` is square.

Internally, an iterative randomized Quasi-Monte Carlo (RQMC) approach is used to estimate the density. It is an iterative algorithm that evaluates the integrand at a randomized Sobol' point-set in each iteration until the pre-specified error tolerance `abstol` is reached for both the density and the log-density. The attribute `"numiter"` gives the worst case number of such iterations needed (over all rows of `x`).

Care should be taken when changing the algorithm-specific parameters, notably `method`, `fun.eval[2]` and `B`. Error estimates will not be reliable for too small `B` and the performance of the algorithm depends heavily on the (quasi-)Monte Carlo point-set used.

If the absolute error tolerance `abstol` cannot be achieved with `fun.eval[2]` function evaluations, an additional warning is thrown.

`dStudent()` and `dNorm()` are wrappers of `dnvmix(, qmix = "inverse.gamma", df = df)` and `dnvmix(, qmix = "constant")`, respectively. In these cases, `dnvmix()` uses the analytical formulas for the density of a multivariate Student *t* and normal distribution, respectively.

Value

`dnvmix()`, `dStudent()` and `dNorm()` return a `numeric` n-vector with the computed (log-)density values and attributes `"error"` (containing the error estimate of the RQMC estimator) and `"numiter"` (containing the number of iterations). If $n > 1$ (so `x` has more than one row), the error estimate reported is the largest error estimate among the n density estimates.

Author(s)

Erik Hintz, Marius Hofert and Christiane Lemieux.

References

McNeil, A. J., Frey, R. and Embrechts, P. (2015). *Quantitative Risk Management: Concepts, Techniques, Tools*. Princeton University Press.

See Also

`pnvmix()`, `rnvmix()`

Examples

```
### Examples for dnvmix() ######################################################

## Generate a random correlation matrix in three dimensions
d <- 3
set.seed(271)
A <- matrix(runif(d * d), ncol = d)
P <- cov2cor(A %*% t(A))

## Evaluate a t_{3.5} density
df <- 3.5
x <- matrix(1:12/12, ncol = d) # evaluation points
dt1 <- dnvmix(x, qmix = "inverse.gamma", df = df, scale = P)
stopifnot(all.equal(dt1, c(0.013266542, 0.011967156, 0.010760575, 0.009648682),
                    tol = 1e-7, check.attributes = FALSE))

## Here is a version providing the quantile function of the mixing distribution
qW <- function(u, df) 1 / qgamma(u, shape = df/2, rate = df/2)
dt2 <- dnvmix(x, qmix = qW, df = df, scale = P)

## Compare
```

```
stopifnot(all.equal(dt1, dt2, tol = 5e-4, check.attributes = FALSE))

## Evaluate a normal density
dn <- dnvmix(x, qmix = "constant", scale = P)
stopifnot(all.equal(dn, c(0.013083858, 0.011141923, 0.009389987, 0.007831596),
                    tol = 1e-7, check.attributes = FALSE))

## Case with missing data
x. <- x
x.[3,2] <- NA
x.[4,3] <- NA
dt <- dnvmix(x., qmix = "inverse.gamma", df = df, scale = P)
stopifnot(is.na(dt) == rep(c(FALSE, TRUE), each = 2))

## Univariate case
x.. <- cbind(1:10/10) # (n = 10, 1)-matrix; note: vectors are taken as rows in dnvmix()
dt1 <- dnvmix(x.., qmix = "inverse.gamma", df = df, factor = 1)
dt2 <- dt(as.vector(x..), df = df)
stopifnot(all.equal(dt1, dt2, check.attributes = FALSE))


### Examples for dStudent() and dNorm() ########################################

## Evaluate a t_{3.5} density
dt <- dStudent(x, df = df, scale = P)
stopifnot(all.equal(dt, c(0.013266542, 0.011967156, 0.010760575, 0.009648682),
                    tol = 1e-7, check.attributes = FALSE))

## Evaluate a normal density
dn <- dNorm(x, scale = P)
stopifnot(all.equal(dn, c(0.013083858, 0.011141923, 0.009389987, 0.007831596),
                    tol = 1e-7, check.attributes = FALSE))
```

---

pnvmix *Distribution Function of Multivariate Normal Variance Mixtures*

Description

Evaluating multivariate normal variance mixture distribution functions (including Student t and normal distributions).

Usage

```
pnvmix(upper, lower = matrix(-Inf, nrow = n, ncol = d), qmix, mean.sqrt.mix = NULL,
       loc = rep(0, d), scale = diag(d), standardized = FALSE,
       method = c("sobol", "ghalton", "PRNG"), precond = TRUE,
       abstol = 1e-3, CI.factor = 3.3, fun.eval = c(2^6, 1e8),
       increment = c("doubling", "num.init"), B = 12, verbose = TRUE, ...)

pStudent(upper, lower = rep(-Inf, d), df, loc = rep(0, d), scale = diag(d),
```

```
         standardized = FALSE, method = c("sobol", "ghalton", "PRNG"),
         precond = TRUE, abstol = 1e-3, CI.factor = 3.3, fun.eval = c(2^6, 1e8),
         B = 12, verbose = TRUE)
pNorm(upper, lower = rep(-Inf, d), loc = rep(0, d), scale = diag(d),
      standardized = FALSE, method = c("sobol", "ghalton", "PRNG"),
      precond = TRUE, abstol = 1e-3, CI.factor = 3.3, fun.eval = c(2^6, 1e8),
      B = 12, verbose = TRUE)
```

Arguments

`upper`	(n, d)-`matrix` of upper integration limits; each row represents a d-vector of upper integration limits.
`lower`	(n, d)-`matrix` of lower integration limits (componentwise less than or equal to `upper`); each row represents a d-vector of lower integration limits.
`qmix`	specification of the mixing variable W; see McNeil et al. (2015, Chapter 6). Supported are the following types of specification (see also the examples below):
	`character`: `character` string specifying a supported distribution; currently available are `"constant"` (in which case $W = 1$ and thus the multivariate normal distribution with mean vector `loc` and covariance matrix `scale` results) and `"inverse.gamma"` (in which case W is inverse gamma distributed with shape and rate parameters `df`/2 and thus the multivariate Student t distribution with `df` degrees of freedom (required to be provided via the ellipsis argument) results).
	`list`: `list` of length at least one, where the first component is a `character` string specifying the base name of a distribution whose quantile function can be accessed via the prefix `"q"`; an example is `"exp"` for which `"qexp"` exists. If the list is of length larger than one, the remaining elements contain additional parameters of the distribution; for `"exp"`, for example, this can be the parameter `rate`.
	`function`: `function` interpreted as the quantile function of the mixing variable W.
`mean.sqrt.mix`	expectation of the square root $\sqrt{(W)}$ of the mixing variable W. If `NULL`, it will be estimated via QMC; this is only needed for determining the reordering of the integration bounds, so a rather crude approximation is fine.
`df`	positive degress of freedom; can also be `Inf` in which case the distribution is interpreted as the multivariate normal distribution with mean vector `loc` and covariance matrix `scale`).
`loc`	location vector of dimension d; this equals the mean vector of a random vector following the specified normal variance mixture distribution if and only if the latter exists.
`scale`	scale matrix (a covariance matrix entering the distribution as a parameter) of dimension (d, d); this equals the covariance matrix of a random vector following the specified normal variance mixture distribution divided by the expecation of the mixing variable W if and only if the former exists.
`standardized`	`logical` indicating whether `scale` is assumed to be a correlation matrix.

`method` — `character` string indicating the method to be used to compute the integral. Available are:

- `"sobol"`: Sobol' sequence
- `"ghalton"`: generalized Halton sequence
- `"PRNG"`: plain Monte Carlo based on a pseudo-random number generator

`precond` — `logical` indicating whether preconditioning is applied, that is, reordering of the integration variables. If `TRUE`, integration limits as well as `scale` are internally re-ordered in a way such that the overall variance of the integrand is usually smaller than with the original ordering; this usually leads smaller run-times.

`abstol` — non-negative `numeric` providing the absolute precision required; if `abstol = NULL`, the algorithm will run until the total number of function evaluations `fun.eval[2]` is reached.

`CI.factor` — multiplier of the Monte Carlo confidence interval bounds. The algorithm runs until `CI.factor` times the estimated standard error is less than `abstol`. If `CI.factor = 3.3` (the default), one can expect the actual absolute error to be less than `abstol` in 99.9% of the cases.

`fun.eval` — `numeric(2)` providing the size of the first point set to be used to estimate the probabilities (typically a power of 2) and the maximal number of function evaluations.

`increment` — `character` string indicating how the sample size should be increased in each iteration. Available are:

- `"doubling"`: next iteration has as many sample points as all the previous iterations combined.
- `"num.init"`: all iterations use an additional `fun.eval[1]`-many points.

`B` — number of randomizations for obtaining an error estimate in the randomized quasi-Monte Carlo (RQMC) approach.

`verbose` — `logical` indicating whether a warning is given if the required precision `abstol` has not been reached; can also be an `integer` in which case 0 is `FALSE`, 1 is `TRUE` and 2 stands for producing a more verbose warning (for each set of provided integration bounds).

`...` — additional arguments (for example, parameters) passed to the underlying mixing distribution when `qmix` is a `character` string or `function`.

Details

One should highlight that evaluating normal variance mixtures is a non-trivial tasks which, at the time of development of **nvmix**, was not available in R before, not even the special case of a multivariate Student t distribution for non-integer degrees of freedom, which frequently appears in applications in finance, insurance and risk management after estimating such distributions.

Note that the procedures call underlying C code. Currently, dimensions $d \geq 16510$ are not supported for the default method `sobol`.

Internally, an iterative randomized Quasi-Monte Carlo (RQMC) approach is used to estimate the probabilities. It is an iterative algorithm that evaluates the integrand at a point-set with size as specified by `increment`) in each iteration until the pre-specified error tolerance `abstol` is reached. The attribute `"numiter"` gives the number of such iterations needed.

Care should be taken when changing the algorithm-specific parameters, notably `method`, `precond`, `fun.eval[2]` and `B`. Error estimates will not be reliable for too small `B` and the performance of the algorithm depends heavily on the (quasi-)Monte Carlo point-set used.

If the absolute error tolerance `abstol` cannot be achieved with `fun.eval[2]` function evaluations, an additional warning is thrown.

`pStudent()` and `pNorm()` are wrappers of `pnvmix(, qmix = "inverse.gamma", df = df)` and `pnvmix(, qmix = "constant")`, respectively. In the univariate case, the functions `pt()` and `pnorm()` are used.

Value

`pnvmix()`, `pStudent()` and `pNorm()` return a `numeric` n-vector with the computed probabilities and corresponding attributes `"error"` (error estimates of the RQMC estimator) and `"numiter"` (number of iterations).

Author(s)

Erik Hintz, Marius Hofert and Christiane Lemieux

References

McNeil, A. J., Frey, R. and Embrechts, P. (2015). *Quantitative Risk Management: Concepts, Techniques, Tools*. Princeton University Press.

Genz, A. and Bretz, F. (1999). Numerical computation of multivariate t-probabilities with application to power calculation of multiple contrasts. *Journal of Statistical Computation and Simulation* 63(4), 103–117.

Genz, A. and Bretz, F. (2002). Comparison of methods for the computation of multivariate *t* probabilities. *Journal of Computational and Graphical Statistics* 11(4), 950–971.

See Also

`dnvmix()`, `rnvmix()`

Examples

```
### Examples for pnvmix() ##################################################

## Generate a random correlation matrix in d dimensions
d <- 3
set.seed(157)
A <- matrix(runif(d * d), ncol = d)
P <- cov2cor(A %*% t(A))

## Evaluate a t_{1/2} distribution function
a <- -3 * runif(d) * sqrt(d) # random lower limit
b <-  3 * runif(d) * sqrt(d) # random upper limit
df <- 0.5 # note that this is *non-integer*
set.seed(1)
pt1 <- pnvmix(b, lower = a, qmix = "inverse.gamma", df = df, scale = P)
```

```
## Here is a version providing the quantile function of the mixing distribution
qW <- function(u, df) 1 / qgamma(u, shape = df/2, rate = df/2)
mean.sqrt.mix <- sqrt(df) * gamma(df/2) / (sqrt(2) * gamma((df+1) / 2))
set.seed(1)
pt2 <- pnvmix(b, lower = a, qmix = qW, mean.sqrt.mix = mean.sqrt.mix, df  = df,
              scale = P)

## Compare
stopifnot(all.equal(pt1, pt2, tol = 7e-4, check.attributes = FALSE))

## mean.sqrt.mix will be approximated by QMC internally if not provided,
## so the results will differ slightly.
set.seed(1)
pt3 <- pnvmix(b, lower = a, qmix = qW, df = df, scale = P)
stopifnot(all.equal(pt3, pt1, tol = 7e-4, check.attributes = FALSE))

## Case with missing data and a matrix of lower and upper bounds
a. <- matrix(rep(a, each = 4), ncol = d)
b. <- matrix(rep(b, each = 4), ncol = d)
a.[2,1] <- NA
b.[3,2] <- NA
pt <- pnvmix(b., lower = a., qmix = "inverse.gamma", df = df, scale = P)
stopifnot(is.na(pt) == c(FALSE, TRUE, TRUE, FALSE))

## Case where upper = (Inf,..,Inf) and lower = (-Inf,...,-Inf)
stopifnot(all.equal(pnvmix(upper = rep(Inf, d), qmix = "constant"), 1,
 check.attributes = FALSE))

### Examples for pStudent() and pNorm() ########################################

## Evaluate a t_{3.5} distribution function
set.seed(271)
pt <- pStudent(b, lower = a, df = 3.5, scale = P)
stopifnot(all.equal(pt, 0.6180, tol = 5e-5, check.attributes = FALSE))

## Evaluate a normal distribution function
set.seed(271)
pn <- pNorm(b, lower = a, scale = P)
stopifnot(all.equal(pn, 0.7001, tol = 1e-4, check.attributes = FALSE))

## pStudent deals correctly with df = Inf:
set.seed(1)
p.St.dfInf <- pStudent(b, df = Inf, scale = P)
set.seed(1)
p.Norm <- pNorm(b, scale = P)
stopifnot(all.equal(p.St.dfInf, p.Norm, check.attributes = FALSE))
```

---

rnvmix *(Quasi-) Random Number Generator for Multivariate Normal Variance Mixtures*

Description

Generate vectors of random variates from multivariate normal variance mixtures (including Student t and normal distributions).

Usage

```
rnvmix(n, rmix = NULL, qmix = NULL, loc = rep(0, d), scale = diag(2),
       factor = NULL, method = c("PRNG", "sobol", "ghalton"),
       skip = 0, ...)

rStudent(n, df, loc = rep(0, d), scale = diag(2), factor = NULL,
         method = c("PRNG", "sobol", "ghalton"), skip = 0)
rNorm(n, loc = rep(0, d), scale = diag(2), factor = NULL,
      method = c("PRNG", "sobol", "ghalton"), skip = 0)
```

Arguments

`n` sample size n (positive integer).

`rmix` specification of the mixing variable W, see McNeil et al. (2015, Chapter 6), via a random number generator. This argument is ignored for `method = "sobol"` and `method = "ghalton"`. Supported are the following types of specification (see also the examples below):

- `character`: `character` string specifying a supported distribution; currently available are `"constant"` (in which case $W = 1$ and thus a sample from the multivariate normal distribution with mean vector `loc` and covariance matrix `scale` results) and `"inverse.gamma"` (in which case W is inverse gamma distributed with shape and rate parameters `df`/2 and thus the multivariate Student t distribution with `df` degrees of freedom (required to be provided via the ellipsis argument) results).
- `list`: `list` of length at least one, where the first component is a `character` string specifying the base name of a distribution which can be sampled via prefix `"r"`; an example is `"exp"` for which `"rexp"` exists for sampling. If the list is of length larger than one, the remaining elements contain additional parameters of the distribution; for `"exp"`, for example, this can be the parameter `rate`.
- `function`: `function` interpreted as a random number generator of the mixing variable W; additional arguments (such as parameters) can be passed via the ellipsis argument.
- `numeric`: `numeric` vector of length `n` providing a random sample of the mixing variable W.

`qmix` specification of the mixing variable W via a quantile function; see McNeil et al. (2015, Chapter 6). This argument is required for `method = "sobol"` and `method = "ghalton"`. Supported are the following types of specification (see also the examples below):

- `character`: `character` string specifying a supported distribution; currently available are `"constant"` (in which case $W = 1$ and thus a sample from the multivariate normal distribution with mean vector `loc` and covariance

matrix scale results) and "inverse.gamma" (in which case W is inverse gamma distributed with shape and rate parameters df/2 and thus the multivariate Student t distribution with df degrees of freedom (required to be provided via the ellipsis argument) results).

list: list of length at least one, where the first component is a character string specifying the base name of a distribution which can be sampled via prefix "q"; an example is "exp" for which "qexp" exists for sampling. If the list is of length larger than one, the remaining elements contain additional parameters of the distribution; for "exp", for example, this can be the parameter rate.

function: function interpreted as the quantile function of the mixing variable W; internally, sampling is then done with the inversion method by applying the provided function to U(0,1) random variates.

df — positive degress of freedom; can also be Inf in which case the distribution is interpreted as the multivariate normal distribution with mean vector loc and covariance matrix scale).

loc — location vector of dimension d; this equals the mean vector of a random vector following the specified normal variance mixture distribution if and only if the latter exists.

scale — scale matrix (a covariance matrix entering the distribution as a parameter) of dimension (d, d) (defaults to $d = 2$); this equals the covariance matrix of a random vector following the specified normal variance mixture distribution divided by the expecation of the mixing variable W if and only if the former exists. Note that scale must be positive definite; sampling from singular normal variance mixtures can be achieved by providing factor.

factor — (d, k)-matrix such that factor %*% t(factor) equals scale; the non-square case $k \neq d$ can be used to sample from singular normal variance mixtures. Note that this notation coincides with McNeil et al. (2015, Chapter 6). If not provided, factor is internally determined via chol() (and multiplied from the right to an (n, k)-matrix of independent standard normals to obtain a sample from a multivariate normal with zero mean vector and covariance matrix scale).

method — character string indicating the method to be used to obtain the sample. Available are:

"PRNG": pseudo-random numbers

"sobol": Sobol' sequence

"ghalton": generalized Halton sequence

If method = "PRNG", either qmix or rmix can be provided. If both are provided, rmix is used and qmix ignored. For the other two methods, sampling is done via inversion, hence qmix has to be provided and rmix is ignored.

skip — integer specifying the number of points to be skipped when method = "sobol", see also example below.

... — additional arguments (for example, parameters) passed to the underlying mixing distribution when rmix or qmix is a character string or function.

Details

Internally used is `factor`, so `scale` is not required to be provided if `factor` is given.

The default factorization used to obtain `factor` is the Cholesky decomposition via `chol()`. To this end, `scale` needs to have full rank.

Sampling from a singular normal variance mixture distribution can be achieved by providing `scale`.

The number of rows of `factor` equals the dimension d of the sample. Typically (but not necessarily), `factor` is square.

`rStudent()` and `rNorm()` are wrappers of `rnvmix(, qmix = "inverse.gamma", df = df)` and `rnvmix(, qmix = "constant", df = df)`, respectively.

Value

`rnvmix()` returns an (n, d)-`matrix` containing n samples of the specified (via `mix`) d-dimensional multivariate normal variance mixture with location vector `loc` and scale matrix `scale` (a covariance matrix).

`rStudent()` returns samples from the d-dimensional multivariate Student t distribution with location vector `loc` and scale matrix `scale`.

`rNorm()` returns samples from the d-dimensional multivariate normal distribution with mean vector `loc` and covariance matrix `scale`.

Author(s)

Erik Hintz, Marius Hofert and Christiane Lemieux

References

McNeil, A. J., Frey, R. and Embrechts, P. (2015). *Quantitative Risk Management: Concepts, Techniques, Tools*. Princeton University Press.

See Also

`dnvmix()`, `pnvmix()`

Examples

```
### Examples for rnvmix() ######################################################

## Generate a random correlation matrix in d dimensions
d <- 3
set.seed(157)
A <- matrix(runif(d * d), ncol = d)
P <- cov2cor(A %*% t(A))

## Draw random variates and compare
df <- 3.5
n <- 1000
set.seed(271)
X  <- rnvmix(n, rmix = "inverse.gamma", df = df, scale = P) # providing scale
```

```
set.seed(271)
X. <- rnvmix(n, rmix = "inverse.gamma", df = df, factor = t(chol(P))) # providing the factor
stopifnot(all.equal(X, X.))

## Checking df = Inf
set.seed(271)
X <- rnvmix(n, rmix = "constant", scale = P) # normal
set.seed(271)
X. <- rnvmix(n, rmix = "inverse.gamma", scale = P, df = Inf) # t_infinity
stopifnot(all.equal(X, X.))

## Univariate case (dimension = number of rows of 'factor' = 1 here)
set.seed(271)
X.1d  <- rnvmix(n, rmix = "inverse.gamma", df = df, factor = 1/2)
set.seed(271)
X.1d. <- rnvmix(n, rmix = "inverse.gamma", df = df, factor = 1)/2 # manual scaling
stopifnot(all.equal(X.1d, X.1d.))

## Checking different ways of providing 'mix'
## 1) By providing a character string (and corresponding ellipsis arguments)
set.seed(271)
X.mix1 <- rnvmix(n, rmix = "inverse.gamma", df = df, scale = P)
## 2) By providing a list; the first element has to be an existing distribution
##    with random number generator available with prefix "r"
rinverse.gamma <- function(n, df) 1 / rgamma(n, shape = df/2, rate = df/2)
set.seed(271)
X.mix2 <- rnvmix(n, rmix = list("inverse.gamma", df = df), scale = P)
## 3) The same without extra arguments (need the extra list() here to
##    distinguish from Case 1))
rinverseGamma <- function(n) 1 / rgamma(n, shape = df/2, rate = df/2)
set.seed(271)
X.mix3 <- rnvmix(n, rmix = list("inverseGamma"), scale = P)
## 4) By providing a quantile function
##    Note: P(1/Y <= x) = P(Y >= 1/x) = 1-F_Y(1/x) = y <=> x = 1/F_Y^-(1-y)
set.seed(271)
X.mix4 <- rnvmix(n, qmix = function(p) 1/qgamma(1-p, shape = df/2, rate = df/2),
                 scale = P)
## 5) By providing random variates
set.seed(271) # if seed is set here, results are comparable to the above methods
W <- rinverse.gamma(n, df = df)
X.mix5 <- rnvmix(n, rmix = W, scale = P)
## Compare (note that X.mix4 is not 'all equal' with X.mix1 or the other samples)
## since rgamma() != qgamma(runif()) (or qgamma(1-runif()))
stopifnot(all.equal(X.mix2, X.mix1),
          all.equal(X.mix3, X.mix1),
          all.equal(X.mix5, X.mix1))

## For a singular normal variance mixture:
## Need to provide 'factor'
A <- matrix( c(1, 0, 0, 1, 0, 1), ncol = 2, byrow = TRUE)
stopifnot(all.equal(dim(rnvmix(n, rmix = "constant", factor = A)),    c(n, 3)))
stopifnot(all.equal(dim(rnvmix(n, rmix = "constant", factor = t(A))), c(n, 2)))
```

```
## Using 'skip'. Need to reset the seed everytime to get the same shifts in "sobol".
## Note that when using method = "sobol", we have to provide 'qmix' instead of 'rmix'.
set.seed(271)
X.skip0 <- rnvmix(n, qmix = "inverse.gamma", df = df, scale = P, method = "sobol")
set.seed(271)
X.skip1 <- rnvmix(n, qmix = "inverse.gamma", df = df, scale = P, method = "sobol",
                  skip = n)
set.seed(271)
X.wo.skip <- rnvmix(2*n, qmix = "inverse.gamma", df = df, scale = P, method = "sobol")
X.skip <- rbind(X.skip0, X.skip1)
all.equal(X.wo.skip, X.skip)


### Examples for rStudent() and rNorm() ########################################

## Draw N(0, P) random variates by providing scale or factor and compare
n <- 1000
set.seed(271)
X.n  <- rNorm(n, scale = P) # providing scale
set.seed(271)
X.n. <- rNorm(n, factor = t(chol(P))) # providing the factor
stopifnot(all.equal(X.n, X.n.))

## Univariate case (dimension = number of rows of 'factor' = 1 here)
set.seed(271)
X.n.1d  <- rNorm(n, factor = 1/2)
set.seed(271)
X.n.1d. <- rNorm(n, factor = 1)/2 # manual scaling
stopifnot(all.equal(X.n.1d, X.n.1d.))

## Draw t_3.5 random variates by providing scale or factor and compare
df <- 3.5
n <- 1000
set.seed(271)
X.t  <- rStudent(n, df = df, scale = P) # providing scale
set.seed(271)
X.t. <- rStudent(n, df = df, factor = t(chol(P))) # providing the factor
stopifnot(all.equal(X.t, X.t.))

## Univariate case (dimension = number of rows of 'factor' = 1 here)
set.seed(271)
X.t.1d  <- rStudent(n, df = df, factor = 1/2)
set.seed(271)
X.t.1d. <- rStudent(n, df = df, factor = 1)/2 # manual scaling
stopifnot(all.equal(X.t.1d, X.t.1d.))

## Check df = Inf
set.seed(271)
X.t <- rStudent(n, df = Inf, scale = P)
set.seed(271)
X.n <- rNorm(n, scale = P)
stopifnot(all.equal(X.t, X.n))
```

Index