Package 'nvmix'

June 12, 2020

Version 0.0-4

Encoding UTF-8

Title Multivariate Normal Variance Mixtures

Description Functions for working with grouped and ungrouped multivariate normal variance mixture
distributions (evaluation of distribution functions and densities,
random number generation and parameter estimation), including
Student's t distribution for non-integer degrees-of-
freedom as well as the grouped t distribution and copula with multiple degrees-of-
freedom parameters.

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, Matrix

Suggests knitr, RColorBrewer, lattice, qrmdata, xts

Enhances

License GPL (>= 3) | file LICENCE

NeedsCompilation yes

VignetteBuilder knitr

Repository CRAN

Date/Publication 2020-06-12 09:10:16 UTC

Repository/R-Forge/Project nvmix

Repository/R-Forge/Revision 283

Repository/R-Forge/DateTimeStamp 2020-06-11 20:31:50

R topics documented:

---

copula	*Functionalities for Normal Variance Mixture Copulas*

Description

Evaluate the density / distribution function of normal variance mixture copulas (including Student *t* and normal copula) and generate vectors of random variates from normal variance mixture copulas.

Usage

```
dnvmixcopula(u, qmix, scale = diag(d), factor = NULL, control = list(),
             verbose = FALSE, log = FALSE, ...)
pnvmixcopula(upper, lower = matrix(0, nrow = n, ncol = d), qmix, scale = diag(d),
             control = list(), verbose = FALSE, ...)
rnvmixcopula(n, qmix, scale = diag(2), factor = NULL,
             method = c("PRNG", "sobol", "ghalton"), skip = 0,
             control = list(), verbose = FALSE, ...)

dStudentcopula(u, df, scale = diag(d), log = FALSE, verbose = TRUE)
pStudentcopula(upper, lower = matrix(0, nrow = n, ncol = d), df, scale = diag(d),
               control = list(), verbose = TRUE)
rStudentcopula(n, df, scale = diag(2), method = c("PRNG", "sobol", "ghalton"),
               skip = 0)

pgStudentcopula(upper, lower = matrix(0, nrow = n, ncol = d), groupings = 1:d,
                df, scale = diag(d), control = list(), verbose = TRUE)
dgStudentcopula(u, groupings = 1:d, df, scale = diag(d), scale.inv = NULL,
                control = list(), verbose = TRUE, log = FALSE)
rgStudentcopula(n, groupings = 1:d, df, scale = diag(2), factor = NULL,
                method = c("PRNG", "sobol", "ghalton"), skip = 0)
```

Arguments

u	(n, d)-[matrix](#) of evaluation points. Have to be in $(0, 1)$.
upper, lower	(n, d)-[matrix](#) of upper/lower evaluation limits. Have to be in $(0, 1)$.
n	sample size n (positive integer).
qmix	specification of the mixing variable W; see [pnvmix](#)() for the ungrouped and [pgnvmix](#)() for the grouped case.
groupings	see [pgnvmix](#)().
df	positive degress of freedom; can also be Inf in which case the copula is interpreted as the Gaussian copula.
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 ungrouped normal variance mixtures can be achieved by providing factor.
scale.inv	inverse of scale; if not provided, computed via solve(scale).
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. For dnvmixcopula(), this has to be a square matrix. 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	see [rnvmix](#)().
skip	see [rnvmix](#)().
control	[list](#) specifying algorithm specific parameters; see [get_set_param](#)().
verbose	[logical](#) indicating whether a warning is given if the required precision abstol has not been reached.
log	[logical](#) indicating whether the logarithmic density is to be computed.
...	additional arguments (for example, parameters) passed to the underlying mixing distribution when rmix or qmix is a [character](#) string or [function](#).

Details

Functionalities for normal variance mixture copulas provided here essentially call [pnvmix](#)(), [dnvmix](#)() and [rnvmix](#)() as well as [qnvmix](#)(), see their documentations for more details.

We remark that computing normal variance mixtures is a challenging task; evaluating normal variance mixture copulas additionally requires the approximation of a univariate quantile function so that for large dimensions and sample sizes, these procedures can be fairly slow. As there are approximations on many levels, reported error estimates for the copula versions of pnvmix() and dnvmix() can be flawed.

The functions [d/p/r]Studentcopula() are user-friendly wrappers for [d/p/r]nvmixcopula(,qmix = "inverse.gamma"), designed for the imporant case of a t copula with degrees-of-freedom df.

Value

The values returned by dnvmixcopula(), rnvmixcopula() and pnvmixcopula() are similar to the ones returned by their non-copula alternatives dnvmix(), rnvmix() and pnvmix().

Author(s)

Erik Hintz, Marius Hofert and Christiane Lemieux

References

Hintz, E., Hofert, M. and Lemieux, C. (2019), Normal variance mixtures: Distribution, density and parameter estimation. https://arxiv.org/abs/1911.03017.

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

See Also

dnvmix(), pnvmix(), qnvmix(), rnvmix()

Examples

```
## Generate a random correlation matrix in d dimensions
d <- 2 # dimension
set.seed(42) # for reproducibility
rho <- runif(1, min = -1, max = 1)
P <- matrix(rho, nrow = d, ncol = d) # build the correlation matrix P
diag(P) <- 1
## Generate two random evaluation points:
u <- matrix(runif(2*d), ncol = d)
## We illustrate using a t-copula
df = 2.1
## Define quantile function which is inverse-gamma here:
qmix. <- function(u) 1/qgamma(1-u, shape = df/2, rate = df/2)


### Example for dnvmixcopula() ##################################################

## If qmix = "inverse.gamma", dnvmix() calls qt and dt:
d1 <- dnvmixcopula(u, qmix = "inverse.gamma", scale = P, df = df)
## Same can be obtained using 'dStudentcopula()'
d2 <- dStudentcopula(u, scale = P, df = df)
stopifnot(all.equal(d1, d2))
## Use qmix. to force the algorithm to use a rqmc procedure:
d3 <- dnvmixcopula(u, qmix = qmix., scale = P)
stopifnot(all.equal(d1, d3, tol = 1e-3, check.attributes = FALSE))


### Example for pnvmixcopula() ##################################################

## Same logic as above:
p1 <- pnvmixcopula(u, qmix = "inverse.gamma", scale = P, df = df)
```

```
p2 <- pnvmixcopula(u, qmix = qmix., scale = P)
stopifnot(all.equal(p1, p2, tol = 1e-3, check.attributes = FALSE))


### Examples for rnvmixcopula() ####################################################

## Draw random variates and compare
n <- 100
set.seed(1)
X  <- rnvmixcopula(n, qmix = "inverse.gamma", df = df, scale = P) # with scale
set.seed(1)
X. <- rnvmixcopula(n, qmix = "inverse.gamma", df = df, factor = t(chol(P))) # with factor
stopifnot(all.equal(X, X.))
```

---

dependencemeasures   *Dependence Measures for grouped normal variance mixture copulas*

Description

Computation of rank correlation coefficients Spearman's rho and Kendall's tau for grouped normal variance mixture copulas as well as computation of the (lower and upper) tail dependence coefficient of a grouped *t* copula.

Usage

```
corgnvmix(scale, qmix, method = c("kendall", "spearman"), groupings = 1:2,
          ellip.kendall = FALSE, control = list(), verbose = TRUE, ...)

lambda_gStudent(df, scale, control = list(), verbose = TRUE)
```

Arguments

scale	n-vector giving the ρ parameters of the copula.
qmix	specification of the mixing variables; see pgnvmix().
method	character indicating if Spearman's rho or Kendall's tau is to be computed.
groupings	vector specifying the grouping structure; either rep(1,2) (ungrouped) or 1:2 (grouped case).
ellip.kendall	logical if the formula for Kendalll's tau for elliptical copulas shall be used; see details below.
df	either scalar or 2-vector giving the degrees-of- freedoms for the t copula; if provided as scalar, the copula is an (ungrouped) t copula and lambda_gStudent() uses a closed formula.
control	list specifying algorithm specific parameters; see get_set_param().
verbose	logical indicating whether a warning is given if the required precision has not been reached.
...	additional arguments (for example, parameters) passed to the underlying mixing distribution when qmix is a character string or function.

Details

For grouped normal variance mixture copulas, including the grouped *t*, there is no closed formula for Kendall's tau and Spearman's rho. The function `corgnvmix()` approximates these dependence measures by numerically approximating an integral representation for these measures.

If no grouping is present (i.e., when `groupings = rep(1,2)`), the copula is an elliptical copula for which the formula $\tau = 2asin(\rho)/pi$ holds. This formula holds only approximately in the grouped case; the quality of the approximation depends on how different the mixing variables for the two components are. When the mixing distributions are not too far apart and when the copula parameter is not close to 1, this approximation is "very accurate", as demonstrated in Daul et al (2003).

In the ungrouped case, `lambda_gStudent()` computes the tail dependence coefficient $lambda$ based on the known formula `2 * pt( -sqrt( (df + 1)*(1 -rho) / (1 + rho)),df = df + 1)` for the tail dependence coefficient of a *t* copula.

In the grouped case, RQMC methods are used to efficiently approximate the integral given in Eq. (26) of Luo and Shevchenko (2010).

Value

`lambda_gStudent()` and `corgnvmix()` return a [numeric](#) n-vector with the computed dependence measure with corresponding attributes `"abs. error"` and `"rel. error"`(error estimates of the RQMC estimator) and `"numiter"` (number of iterations).

Author(s)

Erik Hintz, Marius Hofert and Christiane Lemieux

References

Hintz, E., Hofert, M. and Lemieux, C. (2019), Normal variance mixtures: Distribution, density and parameter estimation. https://arxiv.org/abs/1911.03017.

Luo, X. and Shevchenko, P. (2010). The *t* copula with multiple parameters of degrees of freedom: bivariate characteristics and application to risk management. *Quantitative Finance* 10(9), 1039-1054.

Daul, S., De Giorgi, E. G., Lindskog, F. and McNeil, A (2003). The grouped *t* copula with an application to credit risk. *Available at SSRN 1358956*.

See Also

[dgStudentcopula](#)(), [pgStudentcopula](#)(), [rgStudentcopula](#)()

Examples

```
### Examples for corgnvmix() ##################################################

## Create a plot displaying Spearman's rho for a grouped t copula as a function
## of the copula parameter for various choices of the degrees-of-freedom
qmix <- "inverse.gamma"
df <- matrix( c(1, 2, 1, 5, 1, Inf), ncol = 2, byrow = TRUE)
l.df <- nrow(df)
```

```
scale <- seq(from = 0, to = 1, length.out = 99)
set.seed(1) # for reproducibility
kendalls <- sapply(seq_len(l.df), function(i)
    corgnvmix(scale, qmix = qmix, method = "kendall", df = df[i, ]))
## Include the elliptical approximation (exact when df1 = df2)
kendall_ell <- corgnvmix(scale, method = "kendall", ellip.kendall = TRUE)
## Plot
lgnd <- character(l.df + 1)
lgnd[1] <- "elliptical (equal df)"
plot(NA, xlim = c(0, 1), ylim = c(0, 1), xlab = expression(rho),
     ylab = "Kendall's tau")
lines(scale, kendall_ell, lty = 1)
for(i in 1:l.df){
    lines(scale, kendalls[, i], col = i + 1, lty = i + 1)
    lgnd[i+1] <- paste0("df1 = ", df[i, 1], ", df2 = ", df[i, 2])
}
legend("topleft", lgnd, col = 1:(l.df + 1), lty = 1:(l.df + 1), bty = 'n')


### Examples for lambda_gStudent() ##########################################

## Create a plot displaying 'lambda' as a function of the copula parameter
## for various choices of the degrees-of-freedom
df <- c(3, 6, 9)
df_ <- list( rep(df[1], 2), rep(df[2], 2), rep(df[3], 2), # ungrouped
             c(df[1], df[2]), c(df[1], df[3]), c(df[2], df[3])) # grouped
l.df_ <- length(df_)
scale <- seq(from = -0.99, to = 0.99, length.out = 112) # scale parameters
set.seed(1) # for reproducibilty
lambdas <-
    sapply(seq_len(l.df_), function(i) lambda_gStudent(df_[[i]], scale = scale))
lgnd <- character(length(df_))
plot(NA, xlim = range(scale), ylim = range(lambdas), xlab = expression(rho),
     ylab = expression(lambda))
for(i in seq_len(l.df_)){
    lines(scale, lambdas[, i], col = i, lty = i)
    lgnd[i] <- if(df_[[i]][1] == df_[[i]][2]) paste0("df = ", df_[[i]][1]) else
        paste0("df1 = ", df_[[i]][1], ", df2 = ", df_[[i]][2])
}
legend("topleft", lgnd, col = seq_len(l.df_), lty = seq_len(l.df_),
       bty = 'n')
## If called with 'df' a 1-vector, closed formula for lambda is used => check
lambda.true <- sapply(1:3, function(i) lambda_gStudent(df_[[i]][1], scale = scale))
stopifnot(max(abs( lambda.true - lambdas[, 1:3])) < 4e-4)
```

---

dgnvmix   *Density of Grouped Normal Variance Mixtures*

Description

Evaluating grouped normal variance mixture density functions (including Student t with multiple degrees-of-freedom).

Usage

```
dgnvmix(x, groupings = 1:d, qmix, loc = rep(0, d), scale = diag(d), factor = NULL,
        scale.inv = NULL, control = list(), log = FALSE, verbose = TRUE, ...)
dgStudent(x, groupings = 1:d, df, loc = rep(0, d), scale = diag(d),
          scale.inv = NULL, control = list(), log = FALSE, verbose = TRUE)
```

Arguments

x	see dnvmix().
groupings	see pgnvmix().
qmix	specification of the mixing variables W_i via quantile functions; see pgnvmix().
loc	see pnvmix().
scale	see pnvmix(); must be positive definite.
factor	(d, d) lower triangular matrix such that factor %*% t(factor) equals scale. Internally used is scale.
scale.inv	inverse of scale; if not provided, computed via solve(scale).
df	vector of length length(unique(groupings)) so that variable i has degrees-of-freedom df[groupings[i]]; all elements must be positive and can be Inf, in which case the corresponding marginal is normally distributed.
control	list specifying algorithm specific parameters; see get_set_param().
log	logical indicating whether the logarithmic density is to be computed.
verbose	see pnvmix().
...	additional arguments (for example, parameters) passed to the underlying mixing distribution when qmix is a character string or an element of qmix is a function.

Details

Internally used is scale, so factor is not required to be provided (but allowed for consistency with other functions in the package).

dgStudent() is a wrapper of dgnvmix(, qmix = "inverse.gamma", df = df). If there is no grouping, the analytical formula for the density of a multivariate t distribution is used.

Internally, an adaptive randomized Quasi-Monte Carlo (RQMC) approach is used to estimate the log-density. It is an iterative algorithm that evaluates the integrand at a randomized Sobol' point-set (default) in each iteration until the pre-specified error tolerance control$dnvmix.reltol in the control argument is reached for the log-density. The attribute "numiter" gives the worst case number of such iterations needed (over all rows of x). Note that this function calls underlying C code.

Algorithm specific parameters (such as above mentioned control$dnvmix.reltol) can be passed as a list via the control argument, see get_set_param() for details and defaults.

If the error tolerance cannot be achieved within `control$max.iter.rqmc` iterations and `fun.eval[2]` function evaluations, an additional warning is thrown if `verbose=TRUE`.

Value

`dgnvmix()` and `dgStudent()` return a numeric n-vector with the computed density values and corresponding attributes `"abs. error"` and `"rel. error"` (error estimates of the RQMC estimator) and `"numiter"` (number of iterations).

Author(s)

Erik Hintz, Marius Hofert and Christiane Lemieux

References

Hintz, E., Hofert, M. and Lemieux, C. (2019), Normal variance mixtures: Distribution, density and parameter estimation. <https://arxiv.org/abs/1911.03017>.

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

See Also

rgnvmix(), pgnvmix(), get_set_param()

Examples

```
n <- 100 # sample size to generate evaluation points

### 1. Inverse-gamma mixture
## 1.1. Grouped t with mutliple dof

d <- 3 # dimension
set.seed(157)
A <- matrix(runif(d * d), ncol = d)
P <- cov2cor(A %*% t(A)) # random scale matrix
df <- c(1.1, 2.4, 4.9) # dof for margin i
groupings <- 1:d
x <- rgStudent(n, df = df, scale = P) # evaluation points for the density

### Call 'dgnvmix' with 'qmix' a string:
set.seed(12)
dgt1 <- dgnvmix(x, qmix = "inverse.gamma", df = df, scale = P)
### Version providing quantile functions of the mixing distributions as list
qmix_ <- function(u, df) 1 / qgamma(1-u, shape = df/2, rate = df/2)
qmix <- list(function(u) qmix_(u, df = df[1]), function(u) qmix_(u, df = df[2]),
             function(u) qmix_(u, df = df[3]))
set.seed(12)
dgt2 <- dgnvmix(x, groupings = groupings, qmix = qmix, scale = P)
### Similar, but using ellipsis argument:
qmix <- list(function(u, df1) qmix_(u, df1), function(u, df2) qmix_(u, df2),
             function(u, df3) qmix_(u, df3))
```

```
set.seed(12)
dgt3 <- dgnvmix(x, groupings = groupings, qmix = qmix, scale = P, df1 = df[1],
                df2 = df[2], df3 = df[3])
### Using the wrapper 'dgStudent()'
set.seed(12)
dgt4 <- dgStudent(x, groupings = groupings, df = df, scale = P)
stopifnot(all.equal(dgt1, dgt2, dgt3, dgt4, tol = 1e-5, check.attributes = FALSE))


## 1.2 Classical multivariate t

df <- 2.4
groupings <- rep(1, d) # same df for all components
x <- rStudent(n, scale = P, df = df) # evaluation points for the density
dt1 <- dStudent(x, scale = P, df = df, log = TRUE) # uses analytical formula
## If 'qmix' provided as string and no grouping, dgnvmix() uses analytical formula
dt2 <- dgnvmix(x, qmix = "inverse.gamma", groupings = groupings, df = df, scale = P, log = TRUE)
stopifnot(all.equal(dt1, dt2))
## Provide 'qmix' as a function to force estimation in 'dgnvmix()'
dt3 <- dgnvmix(x, qmix = qmix_, groupings = groupings, df = df, scale = P, log = TRUE)
stopifnot(all.equal(dt1, dt3, tol = 5e-4, check.attributes = FALSE))

### 2. More complicated mixutre

## Let W1 ~ IG(1, 1), W2 = 1, W3 ~ Exp(1), W4 ~ Par(2, 1), W5 = W1, all comonotone
## => X1 ~ t_2; X2 ~ normal; X3 ~ Exp-mixture; X4 ~ Par-mixture; X5 ~ t_2

d <- 5
set.seed(157)
A <- matrix(runif(d * d), ncol = d)
P <- cov2cor(A %*% t(A))
b <- 3 * runif(d) * sqrt(d) # random upper limit
groupings <- c(1, 2, 3, 4, 1) # since W_5 = W_1
qmix <- list(function(u) qmix_(u, df = 2), function(u) rep(1, length(u)),
             list("exp", rate=1), function(u) (1-u)^(-1/2)) # length 4 (# of groups)
x <- rgnvmix(n, groupings = groupings, qmix = qmix, scale = P)
dg <- dgnvmix(x, groupings = groupings, qmix = qmix, scale = P, log = TRUE)
```

---

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, control = list(), 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 pnvmix() for details and examples.
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. Needs to be full rank for the density to exist.
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()).
control	list specifying algorithm specific parameters; see get_set_param().
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 must have full rank.

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

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.

Internally, an adaptive randomized Quasi-Monte Carlo (RQMC) approach is used to estimate the log-density. It is an iterative algorithm that evaluates the integrand at a randomized Sobol' point-set (default) in each iteration until the pre-specified error tolerance control$dnvmix.reltol in the control argument is reached for the log-density. The attribute "numiter" gives the worst case

number of such iterations needed (over all rows of x). Note that this function calls underlying C code.

Algorithm specific parameters (such as above mentioned control$dnvmix.reltol) can be passed as a list via the control argument, see get_set_param() for details and defaults.

If the error tolerance cannot be achieved within control$max.iter.rqmc iterations and fun.eval[2] function evaluations, an additional warning is thrown if verbose=TRUE.

Value

dnvmix(), dStudent() and dNorm() return a numeric n-vector with the computed (log-)density values and attributes "abs. error" and "rel. error" (containing the absolute and relative error estimates of the of the (log-)density) and "numiter" (containing the number of iterations).

Author(s)

Erik Hintz, Marius Hofert and Christiane Lemieux.

References

Hintz, E., Hofert, M. and Lemieux, C. (2019), Normal variance mixtures: Distribution, density and parameter estimation. https://arxiv.org/abs/1911.03017.

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

See Also

pnvmix(), rnvmix(), fitnvmix(), get_set_param().

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(1-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; vectors are considered 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
    x <- x[1,] # use just the first point this time
    dn <- dNorm(x, scale = P)
    stopifnot(all.equal(dn, 0.013083858, tol = 1e-7, check.attributes = FALSE))
```

---

fitnvmix	*Fitting Multivariate Normal Variance Mixtures*

Description

Functionalities for fitting multivariate normal variance mixtures (in particular also Multivariate t distributions) via an ECME algorithm.

Usage

```
fitnvmix(x, qmix, mix.param.bounds, nu.init = NA, loc = NULL, scale = NULL,
         init.size.subsample = min(n, 100), size.subsample = n,
         control = list(), verbose = TRUE)

fitStudent(x, loc = NULL, scale = NULL, mix.param.bounds = c(1e-3, 1e2), ...)
fitNorm(x)
```

Arguments

x	(n, d)-data [matrix](#).
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), "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 results) and "pareto" (in which case W is Pareto distributed with scale equal to unity and shape equal to alpha).

> [function](#): [function](#) interpreted as the quantile function of the mixing variable W. In this case, qmix *must* have the form qmix = function(u,nu), where the argument nu corresponds to the parameter (vector) specifying the distribution of the mixing variable.

mix.param.bounds	
	either [numeric](#)(2) or a [matrix](#) with two columns. The first/second column corresponds to the lower/upper bound of nu_i, the ith component of the parameter vector nu of the mixing variable W. All elements need to be finite, numeric values. Note: The algorithm tends to converge quicker if the parameter ranges supplied are smaller.
nu.init	either NA or an initial guess for the parameter (vector) nu. In the former case an initial estimate is calculated by the algorithm. If nu.init is provided, the algorithm often converges faster; the better the starting value, the faster convergence.
loc	d-[vector](#); if provided, taken as the 'true' location vector in which case loc is not estimated.
scale	positive definite (d, d)-[matrix](#); if provided, taken as the 'true' scale matrix in which case scale is not estimated.
init.size.subsample	
	[numeric](#), non-negative, giving the sub-samplesize used to get an initial estimate for nu. Only used if is.na(nu.init), otherwise ignored.
size.subsample	[numeric](#), non-negative, specifying the size of the subsample that is being used in the ECME iterations to optimize the log-likelihood over nu. Defaults to n, so that the full sample is being used. Decreasing this number can lead to faster run-times (as fewer log-densities need to be estimated) but also to an increase in bias and variance.
control	[list](#) specifying algorithm specific parameters; see below under 'Details' and [get_set_param](#)().
verbose	[numeric](#) or [logical](#) (in which case it is converted to numeric) specifying the amount of tracing to be done. If 0 or FALSE, neither tracing nor warnings are communicated; if 1, only warnings are communicated, if 2 or 3, warnings and (shorter or longer) tracing information is provided.
...	additional arguments passed to the underlying fitnvmix().

Details

The function fitnvmix() uses an ECME algorithm to approximate the MLEs of the parameters nu, loc and scale of a normal variance mixture specified by qmix. The underlying procedure successively estimates nu (with given loc and scale) by maximizing the likelihood which is approximated by dnvmix() (unless qmix is a character string, in which case analytical formulas for the log-densities are used) and scale and loc (given nu) using weights (which again need to be approximated) related to the posterior distribution, details can be found in the first reference below.

It should be highlighted that (unless unless qmix is a character string), every log-likelihood and every weight needed in the estimation is numerically approximated via RQMC methods. For large dimensions and sample sizes this procedure can therefore be slow.

Various tolerances and convergence criteria can be changed by the user via the control argument. For more details, see get_set_param().

Value

fitnvmix() returns a list containing nu, loc, scale as MLEs for nu, the location vector and the scale matrix, respectively; the list also contains iter (number of ECME iterations performed) and max.ll (log-likelihood at the MLEs).

fitStudent() is a wrapper to fitnvmix() for parameter estimation of multivariate Student t distributions; in comparison to fitnvmix(), the fitted degrees of freedom are called "df" (to be consistent with the other wrappers for the Student t distributions).

fitNorm() just returns a list with components loc (columnwise sample means) and scale (sample covariance matrix).

Author(s)

Erik Hintz, Marius Hofert and Christiane Lemieux

References

Hintz, E., Hofert, M. and Lemieux, C. (2019), Normal variance mixtures: Distribution, density and parameter estimation. https://arxiv.org/abs/1911.03017.

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

Liu, C. and Rubin, D. (1994). The ECME algorithm: a simple extension of EM and ECM with faster monotone convergence. *Biometrika* 81(4), 633–648.

See Also

dnvmix(), rnvmix(), pnvmix(), qqplot_maha(), get_set_param().

Examples

```
## Sampling parameters
set.seed(274) # for reproducibility
nu              <- 2.8 # parameter used to sample data
d               <- 4 # dimension
```

```
n                 <- 75 # small sample size to have examples run fast
loc               <- rep(0, d) # location vector
A                 <- matrix(runif(d * d), ncol = d)
diag_vars         <- diag(runif(d, min = 2, max = 5))
scale             <- diag_vars %*% cov2cor(A %*% t(A)) %*% diag_vars # scale matrix
mix.param.bounds <- c(1, 5) # nu in [1, 5]

### Example 1: Fitting a multivariate t distribution ###########################

if(FALSE){
   ## Define 'qmix' as the quantile function of an IG(nu/2, nu/2) distribution
   qmix <- function(u, nu) 1 / qgamma(1 - u, shape = nu/2, rate = nu/2)
   ## Sample data using 'rnvmix'
   x <- rnvmix(n, qmix = qmix, nu = nu, loc = loc, scale = scale)
   ## Call 'fitvnmix' with 'qmix' as a function (so all densities/weights are estimated)
   (MyFit11 <- fitnvmix(x, qmix = qmix, mix.param.bounds = mix.param.bounds))
   ## Call 'fitnvmix' with 'qmix = "inverse.gamma"' in which case analytical formulas
   ## for weights and densities are used:
   (MyFit12 <- fitnvmix(x, qmix = "inverse.gamma",
                           mix.param.bounds = mix.param.bounds))
   ## Alternatively, use the wrapper 'fitStudent()'
   (MyFit13 <- fitStudent(x))
   ## Check
   stopifnot(all.equal(MyFit11$nu, MyFit12$nu, MyFit13$nu, tol = 5e-2))
   ## Can also provide 'loc' and 'scale' in which case only 'nu' is estimated
   (MyFit13 <- fitnvmix(x, qmix = "inverse.gamma", mix.param.bounds = mix.param.bounds,
                           loc = loc, scale = scale))
   (MyFit14 <- fitStudent(x, loc = loc, scale = scale))
   stopifnot(all.equal(MyFit13$nu, MyFit14$df, tol = 1e-6))
}

### Example 2: Fitting a Pareto mixture ########################################

## Define 'qmix' as the quantile function of a Par(nu, 1) distribution
qmix <- function(u, nu) (1-u)^(-1/nu)
## Sample data using 'rnvmix':
x <- rnvmix(n, qmix = qmix, nu = nu, loc = loc, scale = scale)
## Call 'fitvnmix' with 'qmix' as function (=> densities/weights estimated)
(MyFit21 <- fitnvmix(x, qmix = qmix, mix.param.bounds = mix.param.bounds))
## Call 'fitnvmix' with 'qmix = "pareto"' in which case an analytical formula
## for the density is used
(MyFit22 <- fitnvmix(x, qmix = "pareto", mix.param.bounds = mix.param.bounds))
stopifnot(all.equal(MyFit21$nu, MyFit22$nu, tol = 5e-2))
```

---

gammamix   *Functionalities for Gamma Scale Mixture Models*

Description

Evaluating density-, distribution- and quantile-function of Gamma scale mixtures as well as random variate generation.

Usage

```
dgammamix(x, qmix, d, control = list(), verbose = TRUE, log = FALSE, ...)
pgammamix(x, qmix, d, lower.tail = TRUE, control = list(), verbose = TRUE, ...)
qgammamix(u, qmix, d, control = list(), verbose = TRUE, q.only = TRUE,
          stored.values = NULL, ...)
rgammamix(n, rmix, qmix, d, method = c("PRNG", "sobol", "ghalton"),
          skip = 0, ...)
```

Arguments

x	n-vector of evaluation points.
u	n-vector of probabilities.
qmix	see pnvmix().
rmix	see rnvmix().
d	dimension of the underlying normal variance mixture, see also details below.
n	sample size n (positive integer).
lower.tail	logical; if TRUE (default), probabilities are $P(X <= x)$, otherwise $P(X > x)$.
log	logical indicating whether the log-density shall be returned.
q.only	see qnvmix().
stored.values	see qnvmix().
method	see rnvmix().
skip	see rnvmix().
control	list specifying algorithm specific parameters; see get_set_param().
verbose	logical indicating whether a warning is given if the required precision has not been reached.
...	additional arguments (for example, parameters) passed to the underlying mixing distribution when qmix is a character string or function.

Details

We define a Gamma mixture as a random variable Dsq satisfying, in distribution, $Dsq = W * Gamma(d/2, 2)$ where W is specified via qmix. If X follows a $d-$dimensional normal variance mixture, the squared Mahalanobis distance $(X - \mu)^T Sigma^{-1}(X - \mu)$ has the same distribution as Dsq.

The functions presented here are similar to the corresponding functions for normal variance mixtures (d/p/q/rnvmix()), details can be found in the corresponding help-files there.

Value

pgammamix() and dgammamix() return a numeric n-vector with the computed probabilities/densities and corresponding attributes "abs. error" and "rel. error" (error estimates of the RQMC estimator) and "numiter" (number of iterations).

If q.only = TRUE, qgammamix() a vector of the same length as u with entries q_i where q_i satisfies $q_i = inf_x F(x) >= u_i$ where $F(x)$ the df of the Gamma mixture specified via qmix; if q.only = FALSE, see qnvmix.

rgammamix() returns a n-vector containing n samples of the specified (via mix) Gamma mixture.

Author(s)

Erik Hintz, Marius Hofert and Christiane Lemieux

References

Hintz, E., Hofert, M. and Lemieux, C. (2019), Normal variance mixtures: Distribution, density and parameter estimation. https://arxiv.org/abs/1911.03017.

See Also

dnvmix(), pnvmix(), qnvmix(), rnvmix(), get_set_param(), qqplot_maha(), fitnvmix()

Examples

```
## Specify inverse-gamma mixture => results in d * F(d, nu) dist'n,
## handled correctly when 'qmix = "inverse.gamma"' is specified
qmix <- function(u, nu) 1/qgamma(1 - u, shape = nu/2, rate = nu/2)

## Example for rgammamix()
set.seed(271) # for reproducibility
n  <- 25
nu <- 3
d  <- 5
x  <- rgammamix(n, qmix = qmix, d = d, nu = nu)

## Evaluate distribution function at 'x'
p.true_1 <- pgammamix(x, qmix = "inverse.gamma", d = d, df = nu) # calls pf(...)
p.true_2 <- pf(x/d, df1 = d, df2 = nu)
p.estim  <- pgammamix(x, qmix = qmix, d = d, nu = nu)
stopifnot(all.equal(p.true_1, p.true_2, p.estim, tol = 1e-3,
                    check.attributes = FALSE))

## Evaluate density function at 'x'
d.true_1 <- dgammamix(x, qmix = "inverse.gamma", d = d, df = nu)
d.true_2 <- df(x/d, df1 = d, df2 = nu)/d
d.est    <- dgammamix(x, qmix = qmix, d = d, nu = nu)
stopifnot(all.equal(d.true_1, d.true_2, d.est, tol = 5e-4,
                    check.attributes = FALSE))

## Evaluate quantile function
```

```
u <- seq(from = 0.5, to = 0.9, by = 0.1)
q.true_1 <- qgammamix(u, qmix = "inverse.gamma", d = d, df = nu)
q.true_2 <- qf(u, df1 = d, df2 = nu) * d
q.est  <- qgammamix(u, qmix = qmix, d = d, nu = nu)
stopifnot(all.equal(q.true_1, q.true_2, q.est, tol = 5e-4,
                    check.attributes = FALSE))
```

---

get_set_param   *Algorithm-specific Parameters*

Description

Algorithm specific parameters for functionalities in the nvmix package, notably for [fitnvmix](),
[dnvmix](), [pnvmix](), [qnvmix](), [pgammamix](), [dgammamix]() and [ES_nvmix]().

Usage

```
get_set_param(control = list())
```

Arguments

control   [list]() specifying algorithm specific parameters to beset; see below under details.

Details

For most functions in the nvmix package, internally, an iterative randomized Quasi-Monte Carlo
(RQMC) approach is used to estimate probabilities, weights and (log-)densities. There are various
parameters of underlying methods than can be changed.

Algorithm specific parameters can be passed as a list via control. It can contain any of the follow-
ing:

For all algorithms: method [character]() string indicating the method to be used to compute the
integral. Available are:

"sobol": Sobol' sequence (default),

"ghalton": generalized Halton sequence,

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

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 com-
bined,

"num.init": all iterations use an additional fun.eval[1]-many points.

CI.factor multiplier of the Monte Carlo confidence interval bounds. The algorithm runs un-
til CI.factor times the estimated standard error is less than abstol or reltol (whichever
is provided). If CI.factor = 3.5 (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 integrals (typically a power of 2) and the maximal number of function evaluations. `fun.eval` defaults to `c(2^7,1e8)`.

max.iter.rqmc `numeric`, providing the maximum number of iterations allowed in the RQMC approach; the default is 17 if `increment = "doubling"` and `200` otherwise.

B number of randomizations for obtaining an error estimate in the RQMC approach; the default is 15.

For `pnvmix()` **and** `pgnvmix()`: `pnvmix.abstol`, `pnvmix.reltol` non-negative `numeric` providing the relative/absolute precision required for the distribution function. Relative precision via `pnvmix.reltol` is only used when `pnvmix.abstol = NA`; in all other cases, absolute precision will be used. `pnvmix.abstol` defaults to `1e-3`. If `pnvmix.abstol = 0` and `pnvmix.reltol = 0`, the algorithm will typically run until the total number of function evaluations exceeds `fun.eval[2]` or until the total number of iterations exeeds `max.iter.rqmc`, whichever happens first. If $n > 1$ (so upper has more than one row), the algorithm runs until the precision requirement is reached for all n probability estimates.

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.

precond `logical` indicating whether preconditioning is applied, that is, reordering of the integration variables. If `TRUE`, integration limits `lower`, `upper` 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.

cholesky.tol non-negative numeric providing lower threshold for non-zero elements in the computation of the cholesky factor: If calculated $C(i,i)^2 < |cholesky.tol * Scale(i,i)|$, the diagonal element (and all other elements in column i) of the cholesky factor C are set to zero, yielding a singular matrix. `cholesky.tol` defaults to `1e-9`.

For `dnvmix()` **and** `dgnvmix()`: `dnvmix.reltol`, `dnvmix.abstol` non-negative `numeric` providing the relative/absolute precision for the *log-* density required. Absolute precision via `dnvmix.abstol` is only used when `dnvmix.reltol = NA`; in all other cases, relative precision will be used. `dnvmix.reltol` defaults to `1e-2`. If `dnvmix.reltol=0` and `dnvmix.abstol=0`, the algorithm will typically run until the total number of function evaluations exceeds `fun.eval[2]` or until the total number of iterations exeeds `max.iter.rqmc`, whichever happens first. If $n > 1$ (so x has more than one row), the algorithm runs until the precision requirement is reached for all n log-density estimates.

dnvmix.doAdapt `logical` indicating if an adaptive integration procedure shall be used that only samples in relevant subdomains of the mixing variable; defaults to `TRUE`.

dnvmix.max.iter.rqmc.pilot `numeric`, providing the maximum number of unstratified, non-adaptive pilot runs the internal integration procedure performs. Defaults to 4.

dnvmix.tol.int.lower, dnvmix.order.lower both `numeric` and nonnegative. RQMC integration is only performed where the integrand is > than the maximum of `dnvmix.tol.int.lower` and $10^{-c}g_{max}$, where g_{max} is the theoretical maximum of the integrand and c is the specified `dnvmix.order.lower`. Default to `1e-30` and `20`, respectively.

dnvmix.tol.bisec `numeric` vector of length 3 specifying bisection tolerances in the adaptive RQMC algorithm. First/second/third element specify the tolerance on u, W and the log-integrand and default to `1e-16`, `1e-1` and `1e-1`, respectively.

dnvmix.max.iter.bisec `numeric`, maximum number of iterations in the internal bisection procedure to find good cutting points allowed, defaults to `50`.

dnvmix.tol.stratlength [numeric](), nonnegative. If the stratum found by the adaptive integration method has length > dnvmix.tol.stratlength RQMC integration is used there; otherwise a crude approximation. Defaults to 1e-20.

For fitnvmix(): ECMEstep [logical](), if TRUE (default), ECME iteration is performed; if FALSE, no ECME step is performed so that fitnvmix() performs between zero and two optimizations over nu, depending on laststep.do.nu and whether nu.init was provided.

ECMEstep.do.nu [logical](), if TRUE (default), the likelihood is maximized over nu in each ECME iteration; if FALSE, this step is omitted.

laststep.do.nu [logical](), if TRUE another last maximization of the likelihood over nu is performed using all observations after the ECME iterations. Only makes sense if either ECMEstep.do.nu=FALSE or if size.subsample is smaller than the number of observations. Defaults to FALSE.

resample [logical](), if TRUE, a different subsample of x is taken in each optimization over nu in the ECME iterations. Only relevant when size.subsample is smaller than the number of observations. Defaults to FALSE.

ECME.miniter, ECME.maxiter [numeric]() positive, minimum and maximum number of ECME iterations. Default to 5 and 25, respectively.

max.iter.locscaleupdate [numeric]() positive. Maximum number of location-scale updates (while helding nu fixed) in each individual ECME iteration; defaults to 50.

weights.reltol numeric non-negative. Relative tolerance to estimate internal weights used to update loc and $scale$ estimates in the ECME iterations. Defaults to 5e-2.

weights.interpol.reltol numeric non-negative. Some weights can be obtained by interpolating previously calculated weights; if the maximal relative interpolation error is smaller than weights.interpol.reltol, this is done. Defaults to 1e-2.

ECME.rel.conv.tol [numeric]()(3) vector specifying relative convergence tolerances for loc, scale and nu (in this order). Defaults to c(1e-2,1e-2,5e-3).

control.optim [list]() of control parameters passed to the underlying optim in the initial step as well as in the ECME iterations. See [optim]()() for details; defaults to list(maxit=15).

control.optim.laststep like control.optim; this list of control arguments is passed to optim in the last-step. Only relevant when laststep.do.nu = TRUE and defaults to list() (so no defaults of [optim]()() changed).

For qnvmix(): max.iter.newton [numeric](), maximum number of Newton iterations allowed to approximate the quantile; defaults to 45.

newton.conv.abstol [numeric](), convergence tolerance for the Newton proceudre; convergence is detected once the difference of estimated quantiles in two successive iterations is smaller than newton.conv.abstol; defaults to 5e-4.

newton.df.reltol [numeric](), relative error tolerance for estimating the univariate distribution function; defaults to 2.5e-4.

newton.logdens.abstol [numeric](), absolute error tolerance for the internal estimation of the log-density needed for the update; defaults to 1e-2.

newton.df.max.iter.rqmc [numeric](), maximum number of iterations to estimate the univariate distribution function required in the Newton update; defaults to 50. Note that here, internally used is increment = "doubling".

For qqplot_maha(): qqplot.df.reltol [numeric](), corresponds to newton.df.reltol. Defaults to 5e-3.

For ES_nvmix(): riskmeasures.abstol, riskmeasures.reltol [numeric](), absolute or relative
     error tolerance for estimating riskmeasures, notably for ES_nvmix(). By default, riskmeasures.reltol=1e-2
     and riskmeasures.abstol=NA, so that a relative tolerance is used.

Care should be taken when changing algorithm specific parameters, notably tolerances, as the accuracy of the result is heavily influenced by those.

Value

get_set_param() returns a [list]() with more than 30 elements specifying algorithm specific parameters for the functions [fitnvmix](), [dnvmix](), [pnvmix](), [qnvmix](), [pgammamix](), [dgammamix]()
and [ES_nvmix](). Parameter values passed to get_set_param() via the control argument overwrite the defaults; for parameters not specified in the control argument, the default values are
being returned.

Author(s)

Erik Hintz, Marius Hofert and Christiane Lemieux

References

Hintz, E., Hofert, M. and Lemieux, C. (2019), Normal variance mixtures: Distribution, density and
parameter estimation. https://arxiv.org/abs/1911.03017.

See Also

[fitnvmix](), [dnvmix](), [pnvmix](), [qnvmix](), [pgammamix](), [dgammamix](), [ES_nvmix]()

Examples

```
get_set_param() # obtain defaults
```

---

numerical_experiments_data
                         Data Generated by the Demo 'numerical_experiments'

Description

Data generated by the demo('numerical_experiments') of the nvmix package.

Usage

```
data(numerical_experiments_data, package = "nvmix")
```

Format

A list with 10 elements:

`$pnvmix.abserrors` Array as returned by the function `pnvmix_testing_abserr()`, see Section 1.1 of the demo(`'numerical_experiments'`).

`$pnvmix.t.variances` Array as returned by the function `precond_testing_variance()`, see Section 1.1 of the demo(`'numerical_experiments'`).

`$pnvmix.t.sobolind` Array as returned by the function `pnvmix_estimate_sobolind()`, see Section 1.1 of the demo(`'numerical_experiments'`).

`$pnvmix.t.timing` Array as returned by the function `pnvmix_timing_mvt()`, see Section 1.1 of the demo(`'numerical_experiments'`).

`$dnvmix.results` Array as returned by the function `dnvmix_testing()`, see Section 1.2 of the demo(`'numerical_experiments'`).

`$fitnvmix.results` Array as returned by the function `fitnvmix_testing()`, see Section 1.3 of the demo(`'numerical_experiments'`).

`$fit.dj30.anaylytical` Array containing results of `fitnvmix()` applied to DJ30 data using analytical weights/densities, see Section 5 of demo(`'numerical_experiments'`).

`$fit.dj30.estimated` Array containing results of `fitnvmix()` applied to DJ30 data using estimated weights/densities, see Section 5 of demo(`'numerical_experiments'`).

`$qqplots.dj30` Array containing results of `qqplot.maha()` applied to DJ30 data, see Section 5 of the demo(`'numerical_experiments'`).

`$tailprobs.dj30` Array containing estimated quantile shortfall probabilities of models fitted to DJ30 data, see Section 5 of demo(`'numerical_experiments'`).

References

Hintz, E., Hofert, M. and Lemieux, C. (2019), Normal variance mixtures: Distribution, density and parameter estimation. <https://arxiv.org/abs/1911.03017>.

---

pgnvmix	*Distribution Function of Grouped and Generalized Multivariate Normal Variance Mixtures*

Description

Evaluating grouped and generalized multivariate normal variance mixture distribution functions (including Student *t* with multiple degrees-of-freedom).

Usage

```
pgnvmix(upper, lower = matrix(-Inf, nrow = n, ncol = d), groupings = 1:d, qmix,
        rmix, loc = rep(0, d), scale = diag(d), standardized = FALSE,
        control = list(), verbose = TRUE, ...)

pgStudent(upper, lower = matrix(-Inf, nrow = n, ncol = d), groupings = 1:d, df,
          loc = rep(0, d), scale = diag(d), standardized = FALSE,
          control = list(), verbose = TRUE)
```

Arguments

upper	see [pnvmix]().
lower	see [pnvmix]().
groupings	d-[vector](). Specification of the groupings so that variable i has mixing variable W_k where k = groupings[i]. If groupings = 1:d, each variable has a different mixing distribution.
qmix	specification of the mixing variables W_i via quantile functions; 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 "inverse.gamma" (in which case W_i is inverse gamma distributed with shape and rate parameters df[groupings[i]]/2 and a multivariate Student *t* distribution multiple degreess-of-freedom results) and "pareto" (in which case W_i is Pareto distributed with scale equal to unity and shape equal to alpha[groupings[i]]. alpha and df must be of length length(unique(groupings)) and need to be provided via the ellipsis argument).

[list](): [list]() of length length(unique(groupings)) (number of different mixing distributions). Element i of this list specifies the mixing variable for component groupings[i]. Each element of this list can be

[list](): a [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 "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 or random number generator of the mixing variable W_i

rmix	only allowed when groupings = rep(1,d) in which case pgnvmix() is equivalent to [pnvmix](); see [pnvmix]().
df	[vector]() of length length(unique(groupings)) so that variable i has degrees-of-freedom df[groupings[i]]; all elements must be positive and can be Inf, in which case the corresponding marginal is normally distributed.
loc	see [pnvmix]().
scale	see [pnvmix](); must be positive definite.
standardized	see [pnvmix]().
control	[list]() specifying algorithm specific parameters; see [get_set_param]().
verbose	see [pnvmix]().
...	additional arguments (for example, parameters) passed to the underlying mixing distribution when qmix is a [character]() string or an element of qmix is a [function]().

Details

One should highlight that evaluating generalized 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 freedoms, which frequently appears in applications in finance, insurance and risk management after estimating such distributions.

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 control$increment in the control argument) in each iteration until the pre-specified absolute error tolerance control$pnvmix.abstol (or relative error tolerance control$pnvmix.reltol which is used only when control$pnvmix.abstol = NA) is reached. The attribute "numiter" gives the number of such iterations needed. Algorithm specific parameters (such as the above mentioned control$pnvmix.abstol) can be passed as a list via control, see [get_set_param]() for more details. If specified error tolerances are not reached and verbose = TRUE, a warning is thrown.

pgStudent() is a wrapper of pgnvmix(,qmix = "inverse.gamma",df = df).

Value

pgnvmix() and pgStudent() return a [numeric] n-vector with the computed probabilities and corresponding attributes "abs. error" and "rel. error" (error estimates of the RQMC estimator) and "numiter" (number of iterations).

Author(s)

Erik Hintz, Marius Hofert and Christiane Lemieux

References

Hintz, E., Hofert, M. and Lemieux, C. (2019), Normal variance mixtures: Distribution, density and parameter estimation. https://arxiv.org/abs/1911.03017.

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.

Genz, A. and Kwong, K. (2000). Numerical evaluation of singular multivariate normal distributions. *Journal of Statistical Computation and Simulation* 68(1), 1–21.

See Also

[rgnvmix](), [dgnvmix](), [get_set_param]()

Examples

```
### Examples for pgnvmix() ###################################################

## 1. Inverse-gamma mixture (=> distribution is grouped t with mutliple dof)
d <- 3
set.seed(157)
A <- matrix(runif(d * d), ncol = d)
P <- cov2cor(A %*% t(A))
a <- -3 * runif(d) * sqrt(d) # random lower limit
b <-  3 * runif(d) * sqrt(d) # random upper limit
df <- c(1.1, 2.4, 4.9) # dof for margin i
groupings <- 1:d

### Call 'pgnvmix' with 'qmix' a string:
set.seed(12)
(pgt1 <- pgnvmix(b, lower = a, groupings = groupings, qmix = "inverse.gamma",
                 df = df, scale = P))
### Version providing quantile functions of the mixing distributions as list
qmix_ <- function(u, df) 1 / qgamma(1-u, shape = df/2, rate = df/2)
qmix <- list(function(u) qmix_(u, df = df[1]), function(u) qmix_(u, df = df[2]),
             function(u) qmix_(u, df = df[3]))
set.seed(12)
(pgt2 <- pgnvmix(b, lower = a, groupings = groupings, qmix = qmix, scale = P))
### Similar, but using ellipsis argument:
qmix <- list(function(u, df1) qmix_(u, df1), function(u, df2) qmix_(u, df2),
             function(u, df3) qmix_(u, df3))
set.seed(12)
(pgt3 <- pgnvmix(b, lower = a, groupings = groupings, qmix = qmix,
                          scale = P, df1 = df[1], df2 = df[2], df3 = df[3]))
## Version using the user friendly wrapper 'pgStudent()'
set.seed(12)
(pgt4 <- pgStudent(b, lower = a, groupings = groupings, scale = P, df = df))
stopifnot(all.equal(pgt1, pgt2, tol = 1e-4, check.attributes = FALSE),
          all.equal(pgt2, pgt3), all.equal(pgt1, pgt4))


## 2. More complicated mixutre
## Let W1 ~ IG(1, 1), W2 = 1, W3 ~ Exp(1), W4 ~ Par(2, 1), W5 = W1, all comonotone
## => X1 ~ t_2; X2 ~ normal; X3 ~ Exp-mixture; X4 ~ Par-mixture; X5 ~ t_2

d <- 5
set.seed(157)
A <- matrix(runif(d * d), ncol = d)
P <- cov2cor(A %*% t(A))
b <- 3 * runif(d) * sqrt(d) # random upper limit
groupings <- c(1, 2, 3, 4, 1) # since W_5 = W_1
qmix <- list(function(u) qmix_(u, df = 2), function(u) rep(1, length(u)),
             list("exp", rate=1), function(u) (1-u)^(-1/2)) # length 4 (# of groups)
pg1 <- pgnvmix(b, groupings = groupings, qmix = qmix, scale = P)
stopifnot(all.equal(pg1, 0.78711, tol = 5e-6, check.attributes = FALSE))
```

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, rmix,
      loc = rep(0, d), scale = diag(d), standardized = FALSE,
      control = list(), verbose = TRUE, ...)

pStudent(upper, lower = matrix(-Inf, nrow = n, ncol = d), df, loc = rep(0, d),
        scale = diag(d), standardized = FALSE, control = list(), verbose = TRUE)
pNorm(upper, lower = matrix(-Inf, nrow = n, ncol = d), loc = rep(0, d),
      scale = diag(d), standardized = FALSE, control = list(), 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, rmix	specification of the mixing variable W via a quantile function (qmix) *or* random number generator (rmix) ; 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), "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) and "pareto" (in which case W is Pareto distributed with scale equal to unity and shape equal to alpha, which needs to be provided via the ellipsis argument).

> [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 or random number generator can be accessed via the prefix "q" and "r", respectively. 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 or random number generator 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. scale is allowed to be
                   singular in which case the distribution function of the singular normal variance
                   mixture is returned.

standardized   logical indicating whether scale is assumed to be a correlation matrix.

control   [list](#) specifying algorithm specific parameters; see [get_set_param](#)().

verbose   [logical](#) indicating whether a warning is thrown if the required precision pnvmix.abstol
                   or pnvmix.reltol as specified in the control argument has not been reached;
                   can also be an [integer](#) in which case 0 is FALSE, 1 is TRUE and 2 stands for pro-
                   ducing 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 mul-
tivariate 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 sup-
ported 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 speci-
fied by control$increment in the control argument) in each iteration until the pre-specified abso-
lute error tolerance control$pnvmix.abstol (or relative error tolerance control$pnvmix.reltol
which is used only when control$pnvmix.abstol = NA) is reached. The attribute "numiter" gives
the number of such iterations needed. Algorithm specific parameters (such as the above mentioned
control$pnvmix.abstol) can be passed as a list via control, see [get_set_param](#)() for more
details. If specified error tolerances are not reached and verbose = TRUE, a warning is thrown.
If provided scale is singular, pnvmix() estimates the correct probability but throws a warning if
verbose = TRUE.

It is recommended to supply a quantile function via qmix, if available, as in this case efficient
RQMC methods are used to approximate the probability. If rmix is provided, internally used is
plain MC integration, typically leading to slower convergence. If both qmix and rmix are provided,
the latter is ignored.

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 "abs. error" and rel. error (error estimates of the RQMC estimator) and "numiter" (number of iterations).

Author(s)

Erik Hintz, Marius Hofert and Christiane Lemieux

References

Hintz, E., Hofert, M. and Lemieux, C. (2019), Normal variance mixtures: Distribution, density and parameter estimation. https://arxiv.org/abs/1911.03017.

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.

Genz, A. and Kwong, K. (2000). Numerical evaluation of singular multivariate normal distributions. *Journal of Statistical Computation and Simulation* 68(1), 1–21.

See Also

[dnvmix](#)(), [rnvmix](#)(), [fitnvmix](#)(), [pgnvmix](#)(), [get_set_param](#)()

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 <- 1.5 # note that this is *non-integer*
set.seed(123)
pt1 <- pnvmix(b, lower = a, qmix = "inverse.gamma", df = df, scale = P)

## Here is a version providing the quantile function of the mixing distribution
qmix <- function(u, df) 1 / qgamma(1-u, shape = df/2, rate = df/2)
mean.sqrt.mix <- sqrt(df) * gamma(df/2) / (sqrt(2) * gamma((df+1) / 2))
set.seed(123)
pt2 <- pnvmix(b, lower = a, qmix = qmix, df  = df, scale = P,
```

```
                    control = list(mean.sqrt.mix = mean.sqrt.mix))

## 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(123)
pt3 <- pnvmix(b, lower = a, qmix = qmix, df = df, scale = P)
stopifnot(all.equal(pt3, pt1, tol = 7e-4, check.attributes = FALSE))

## Here is a version providing a RNG for the mixing distribution
## Note the significantly larger number of iterations in the attribute 'numiter'
## compared to when 'qmix' was provided (=> plain MC versus RQMC)
set.seed(123)
pt4 <- pnvmix(b, lower = a,
              rmix = function(n, df) 1/rgamma(n, shape = df/2, rate = df/2),
              df = df, scale = P)
stopifnot(all.equal(pt4, pt1, tol = 8e-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))

## An example with singular scale:
A <- matrix( c(1, 0, 0, 0,
               2, 1, 0, 0,
               3, 0, 0, 0,
               4, 1, 0, 1), ncol = 4, nrow = 4, byrow = TRUE)
scale <- A%*%t(A)
upper <- 2:5

pn <- pnvmix(upper, qmix = "constant", scale = scale) # multivariate normal
pt <- pnvmix(upper, qmix = "inverse.gamma", scale = scale, df = df) # multivariate t

stopifnot(all.equal(pn, 0.8581, tol = 1e-3, check.attributes = FALSE))
stopifnot(all.equal(pt, 0.7656, tol = 1e-3, check.attributes = FALSE))

## Evaluate a Exp(1)-mixture
## Specify the mixture distribution parameter
rate <- 1.9 # exponential rate parameter

## Method 1: Use R's qexp() function and provide a list as 'mix'
set.seed(42)
(p1 <- pnvmix(b, lower = a, qmix = list("exp", rate = rate), scale = P))
```

```
## Method 2: Define the quantile function manually (note that
##            we do not specify rate in the quantile function here,
##            but conveniently pass it via the ellipsis argument)
set.seed(42)
(p2 <- pnvmix(b, lower = a, qmix = function(u, lambda) -log(1-u)/lambda,
              scale = P, lambda = rate))

## Check
stopifnot(all.equal(p1, p2))


### 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 = 7e-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(123)
p.St.dfInf <- pStudent(b, df = Inf, scale = P)
set.seed(123)
p.Norm <- pNorm(b, scale = P)
stopifnot(all.equal(p.St.dfInf, p.Norm, check.attributes = FALSE))
```

---

qnvmix   *Quantile Function of a univariate Normal Variance Mixture Distribu-*
                                tion

Description

Evaluating multivariate normal variance mixture distribution functions (including normal and Student *t* for non-integer degrees of freedom).

Usage

```
qnvmix(u, qmix, control = list(),
       verbose = TRUE, q.only = TRUE, stored.values = NULL, ...)
```

Arguments

u   vector of probabilities .

qmix   specification of the mixing variable W; see pnvmix() for details and examples.

control	[list](#) specifying algorithm specific parameters; see [get_set_param](#)().
verbose	[logical](#), if TRUE a warning is printed if one of the error tolerances is not met.
q.only	[logical](#). If TRUE, only the quantiles are returned; if FALSE, see Section 'value' below.
stored.values	[matrix](#) with 3 columns of the form $[x, F(x), log f(x)]$ where $F()$ and $log f()$ are the distribution- and log-density function of the distribution specified in qmix. If provided it is used to determine starting values for internal newton proceudures. Only very basic checking is done.
...	additional arguments containing parameters of mixing distributions when qmix is a [character](#) string.

Details

This function uses a Newton procedure to estimate the quantile of the specified univariate normal variance mixture distribution. Internally, a randomized quasi-Monte Carlo (RQMC) approach is used to estimate the distribution and (log)density function; the method is similar to the one in [pnvmix](#)() and [dnvmix](#)(). The result depends slightly on .random.seed.

Internally, symmetry is used for $u \leq 0.5$. Function values (i.e., df and log-density values) are stored and reused to get good starting values. These values are returned if q.only = FALSE and can be re-used by passing it to qnvmix() via the argument stored.values; this can significantly reduce run-time.

Accuracy and run-time depend on both the magnitude of u and on how heavy the tail of the underlying distributions is. Numerical instabilities can occur for values of u close to 0 or 1, especially when the tail of the distribution is heavy.

If q.only = FALSE the log-density values of the underlying distribution evaluated at the estimated quantiles are returned as well: This can be useful for copula density evaluations where both quantities are needed.

Underlying algorithm specific parameters can be changed via the control argument, see [get_set_param](#)() for details.

Value

If q.only = TRUE a vector of the same length as u with entries q_i where q_i satisfies $q_i = inf_x F(x) \geq u_i$ where $F(x)$ the univariate df of the normal variance mixture specified via qmix;

if q.only = FALSE a list of four:

$q: Vector of quantiles,

$log.density: vector log-density values at q,

$computed.values: matrix with 3 columns [x, F(x), logf(x)]; see details above,

$newton.iterations: vector giving the number of Newton iterations needed for u[i].

Author(s)

Erik Hintz, Marius Hofert and Christiane Lemieux

References

Hintz, E., Hofert, M. and Lemieux, C. (2019), Normal variance mixtures: Distribution, density and parameter estimation. <https://arxiv.org/abs/1911.03017>.

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

See Also

[dnvmix()](), [rnvmix()](), [pnvmix()]()

Examples

```
## Evaluation points
u <- seq(from = 0.05, to = 0.95, by = 0.1)
set.seed(271) # for reproducibility

## Evaluate the t_{1.4} quantile function
df <- 1.4
qmix. <- function(u) 1/qgamma(1-u, shape = df/2, rate = df/2)
## If qmix = "inverse.gamma", qt() is being called
qt1 <- qnvmix(u, qmix = "inverse.gamma", df = df)
## Estimate quantiles (without using qt())
qt1. <- qnvmix(u, qmix = qmix., q.only = FALSE)
stopifnot(all.equal(qt1, qt1.$q, tolerance = 2.5e-3))
## Look at absolute error:
abs.error <- abs(qt1 - qt1.$q)
plot(u, abs.error, type = "l", xlab = "u", ylab = "Absolute error")
## Now do this again but provide qt1.$stored.values, in which case at most
## one Newton iteration will be needed:
qt2 <- qnvmix(u, qmix = qmix., stored.values = qt1.$computed.values, q.only = FALSE)
stopifnot(max(qt2$newton.iterations) <= 1)
```

---

qqplot_maha   *QQ Plot for Multivariate Normal Variance Mixtures*

Description

Visual goodness-of-fit test for multivariate normal variance mixtures: Plotting squared Mahalanobis distances against their theoretical quantiles.

Usage

```
qqplot_maha(x, qmix, loc, scale, plot.diag = TRUE, verbose = TRUE,
            control = list(), ...)
```

Arguments

x	(n, d)-data `matrix`.
qmix	see `pnvmix()`.
loc	see `pnvmix()`.
scale	see `pnvmix()`.
plot.diag	`logical` indicating if the diagonal $y = x$ shall be included in the plot.
verbose	see `pnvmix()`.
control	`list` specifying algorithm specific parameters; see `get_set_param()`.
...	additional arguments (for example, parameters) passed to the underlying mixing distribution when qmix is a `character` string or `function`.

Details

If X follows a multivariate normal variance mixture, the distribution of the Mahalanobis distance $D^2 = (X - \mu)^T \Sigma^{-1}(X - \mu)$ is a gamma mixture whose distribution function can be approximated. The function qqplot_maha() plots the empirical Mahalanobis distances from the data in x (with $\mu =$ loc and $Sigma =$ scale) versus their theoretical quantiles which are internally estimated via the function `qgammamix()`.

Value

qqplot_maha() (invisibly) returns a list of two: maha2, the sorted Mahalanobis distances of the data in x with respect to loc and scale and q, the theoretical quantiles evaluated at ppoints(n) where n=nrow(x).

Author(s)

Erik Hintz, Marius Hofert and Christiane Lemieux

References

Hintz, E., Hofert, M. and Lemieux, C. (2019), Normal variance mixtures: Distribution, density and parameter estimation. https://arxiv.org/abs/1911.03017.

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

`fitnvmix()`, `rnvmix()`, `pnvmix()`

Examples

```
## Setup
n     <- 15
d     <- 2
df    <- 3.1 # degree-of-freedom parameter
loc   <- rep(0, d)
scale <- diag(d)
## Define the quantile function of an IG(nu/2, nu/2) distribution
qmix <- function(u, df) 1 / qgamma(1 - u, shape = df/2, rate = df/2)
## Sample data
set.seed(1) # for reproducibility
x <- rnvmix(n, qmix = qmix, df = df, loc = loc, scale = scale)
## QQ Plot of empirical quantiles vs true quantiles, all values estimated
## via RQMC:
qqplot_maha(x, qmix = qmix, loc = loc, scale = scale, df = df)
## Same could be obtained by specifying 'qmix' as string in which case
## qqplot_maha() calls qf()
qqplot_maha(x, qmix = "inverse.gamma", loc = loc, scale = scale, df = df)
```

---

rgnvmix *(Quasi-)Random Number Generator for Grouped Normal Variance Mixtures*

Description

Generate vectors of random variates from grouped normal variance mixtures (including Student *t* with multiple degrees-of-freedom).

Usage

```
rgnvmix(n, qmix, groupings = 1:d, loc = rep(0, d), scale = diag(2),
        factor = NULL, method = c("PRNG", "sobol", "ghalton"), skip = 0, ...)
rgStudent(n, groupings = 1:d, df, loc = rep(0, d), scale = diag(2),
          factor = NULL, method = c("PRNG", "sobol", "ghalton"), skip = 0)
```

Arguments

n	sample size n (positive integer).
qmix	specification of the mixing variables W_i; see [pgnvmix]().
groupings	[vector]() specifying the group structure; see [pgnvmix]().
df	[vector]() specifying the degrees-of-freedom; see see [pgStudent]().
loc	see [pgnvmix]().
scale	see [pgnvmix](). scale must be positive definite; sampling from singular normal variance mixtures can be achieved by providing factor.
factor	see [rnvmix]().
method	see [rnvmix]().

skip	see [rnvmix]().
...	additional arguments (for example, parameters) passed to the underlying mixing distribution when qmix is a [character]() string or an element of qmix is a [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.

rgStudent() is a wrapper of rgnvmix(,qmix = "inverse.gamma",df = df).

Value

rgnvmix() returns an (n, d)-[matrix]() containing n samples of the specified (via qmix) d-dimensional grouped normal variance mixture with location vector loc and scale matrix scale (a covariance matrix).

rgStudent() returns samples from the d-dimensional multivariate *t* distribution with multiple degrees-of-freedom specified by df, location vector loc and scale matrix scale.

Author(s)

Erik Hintz, Marius Hofert and Christiane Lemieux

References

Hintz, E., Hofert, M. and Lemieux, C. (2019) *Normal variance mixtures: Distribution, density and parameter estimation*. https://arxiv.org/abs/1911.03017

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

See Also

[rnvmix](), [pgnvmix]()

Examples

```
n <- 1000 # sample size

## Generate a random correlation matrix in d dimensions
d <- 2
set.seed(157)
A <- matrix(runif(d * d), ncol = d)
scale <- cov2cor(A %*% t(A))

## Example 1: Exponential mixture
## Let W_1 ~ Exp(1), W_2 ~ Exp(10)
rates <- c(1, 10)
#qmix <- list(list("exp", rate = rates[1]), list("exp", rate = rates[2]))
```

```
qmix <- lapply(1:2, function(i) list("exp", rate = rates[i]))
set.seed(1)
X.exp1 <- rgnvmix(n, qmix = qmix, scale = scale)
## For comparison, consider NVM distribution with W ~ Exp(1)
set.seed(1)
X.exp2 <- rnvmix(n, qmix = list("exp", rate = rates[1]), scale = scale)
## Plot both samples with the same axes
opar <- par(no.readonly = TRUE)
par(mfrow=c(1,2))
plot(X.exp1, xlim = range(X.exp1, X.exp2), ylim = range(X.exp1, X.exp2),
     xlab = expression(X[1]), ylab = expression(X[2]))
mtext("Two groups with rates 1 and 10")
plot(X.exp2, xlim = range(X.exp1, X.exp2), ylim = range(X.exp1, X.exp2),
     xlab = expression(X[1]), ylab = expression(X[2]))
mtext("One group with rate 1")
par(opar)

## Example 2: Exponential + Inverse-gamma mixture
## Let W_1 ~ Exp(1), W_2 ~ IG(1.5, 1.5) (=> X_2 ~ t_3 marginally)
df <- 3
qmix <- list(list("exp", rate = rates[1]),
             function(u, df) 1/qgamma(1-u, shape = df/2, rate = df/2))
set.seed(1)
X.mix1 <- rgnvmix(n, qmix = qmix, scale = scale, df = df)
plot(X.mix1, xlab = expression(X[1]), ylab = expression(X[2]))

## Example 3: Mixtures in d > 2
d <- 5
set.seed(157)
A <- matrix(runif(d * d), ncol = d)
scale <- cov2cor(A %*% t(A))

## Example 3.1: W_i ~ Exp(i), i = 1,...,d
qmix <- lapply(1:d, function(i) list("exp", rate = i))
set.seed(1)
X.mix2 <- rgnvmix(n, qmix = qmix, scale = scale)

## Example 3.2: W_1, W_2 ~ Exp(1), W_3, W_4, W_5 ~ Exp(2)
## => 2 groups, so we need two elements in 'qmix'
qmix <- lapply(1:2, function(i) list("exp", rate = i))
groupings <- c(1, 1, 2, 2, 2)
set.seed(1)
X.mix3 <- rgnvmix(n, qmix = qmix, groupings = groupings, scale = scale)

## Example 3.3: W_1, W_3 ~ IG(1, 1), W_2, W_4 ~ IG(2, 2), W_5 = 1
## => X_1, X_3 ~ t_2; X_2, X_4 ~ t_4, X_5 ~ N(0, 1)
qmix <- list(function(u, df1) 1/qgamma(1-u, shape = df1/2, rate = df1/2),
             function(u, df2) 1/qgamma(1-u, shape = df2/2, rate = df2/2),
             function(u) rep(1, length(u)))
groupings = c(1, 2, 1, 2, 3)
df = c(2, 4, Inf)
set.seed(1)
X.t1 <- rgnvmix(n, qmix = qmix, groupings = groupings, scale = scale,
```

```
                     df1 = df[1], df2 = df[2])

## This is equivalent to calling 'rgnmvix' with 'qmix = "inverse.gamma"'
set.seed(1)
X.t2 <- rgnvmix(n, qmix = "inverse.gamma", groupings = groupings, scale = scale,
                df = df)

## Alternatively, one can use the user friendly wrapper 'rgStudent()'
set.seed(1)
X.t3 <- rgStudent(n, df = df, groupings = groupings, scale = scale)

stopifnot(all.equal(X.t1, X.t2, X.t3))
```

---

riskmeasures   *Risk measures for normal variance mixtures*

Description

Estimation of value-at-risk and expected shortfall for univariate normal variance mixtures

Usage

```
VaR_nvmix(level, qmix, loc = 0, scale = 1, control = list(), verbose = TRUE, ...)
ES_nvmix(level, qmix, loc = 0, scale = 1, control = list(), verbose = TRUE, ...)
```

Arguments

level	n-[vector](#) of confidence levels.
qmix	see [pnvmix](#)().
loc	[numeric](#) location, see also [pnvmix](#)()
scale	[numeric](#) scale, see also [pnvmix](#)()
control	[list](#) specifying algorithm specific parameters; see [get_set_param](#)().
verbose	[logical](#) indicating whether a warning is given if the required precision has not been reached.
...	additional arguments (for example, parameters) passed to the underlying mixing distribution when qmix is a [character](#) string or [function](#), see also [pnvmix](#)()

Details

VaR_nvmix calls [qnvmix](#)().

The function ES_nvmix() estimates the expected shortfall using a randomized quasi Monte Carlo procedure by sampling from the mixing variable specified via qmix and and using the identity $\int_k^\infty x\phi(x)dx = \phi(k)$ where $\phi(x)$ denotes the density of a standard normal distribution. Algorithm specific paramaters (such as tolerances) can be conveniently passed via the control argument, see [get_set_param](#)() for more details.

Value

VaR_nvmix() and ES_nvmix() return a [numeric](#) n-vector with the computed risk measures and in case of ES_nvmix() corresponding attributes "abs. error" and "rel. error"(error estimates of the RQMC estimator) and "numiter" (number of iterations).

Author(s)

Erik Hintz, Marius Hofert and Christiane Lemieux

References

Hintz, E., Hofert, M. and Lemieux, C. (2019), Normal variance mixtures: Distribution, density and parameter estimation. https://arxiv.org/abs/1911.03017.

See Also

[dnvmix](#)(), [pnvmix](#)(), [qnvmix](#)(), [rnvmix](#)(), [get_set_param](#)()

Examples

```
## Example for inverse-gamma mixture (resulting in a t distribution) for
## which the expected shortfall admits a closed formula
set.seed(42) # reproducibility
level <- seq(from = 0.9, to = 0.95, by = 0.01)
df <- 4
## If 'qmix' is provided as string, ES_nvmix() uses the closed formula
ES1 <- ES_nvmix(level, qmix = "inverse.gamma", df = df)
## If 'qmix' is provided as function, the expected shortfall is estimated
ES2 <- ES_nvmix(level, qmix = function(u, df) 1/qgamma(1-u, shape = df/2, rate = df/2),
        df = df)
stopifnot(all.equal(ES1, ES2, tol = 1e-2, check.attributes = FALSE))
```

---

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

Description

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

Usage

```
rnvmix(n, rmix, qmix, 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)
rNorm_sumconstr(n, weights, s, 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.
weights	d-numeric vector of weights.
s	numeric vector of length 1 or n giving the value of the constrained sum; see below under details.
...	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.

The function rNorm_sumconstr() can be used to sample from the multivariate standard normal distribution under a weighted sum constraint; the implementation is based on Algorithm 1 in Vrins (2018). Let $Z = (Z_1, \ldots, Z_d) \ N_d(0, I_d)$. The function rNorm_sumconstr() then samples from $Z|w^T Z = s$ where w and s correspond to the arguments weights and s. If supplied s is a vector of length n, the i'th row of the returned matrix uses the constraint $w^T Z = s_i$ where s_i is the i'th element in s.

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.

rNorm_sumconstr() returns samples from the d-dimensional multivariate normal distribution conditional on the weighted sum being constrained to s.

Author(s)

Erik Hintz, Marius Hofert and Christiane Lemieux

References

Hintz, E., Hofert, M. and Lemieux, C. (2019) *Normal variance mixtures: Distribution, density and parameter estimation*. https://arxiv.org/abs/1911.03017

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

Vrins, E. (2018) *Sampling the Multivariate Standard Normal Distribution under a Weighted Sum Constraint*. Risks 2018, 6(3), 64

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) # with scale
set.seed(271)
X. <- rnvmix(n, rmix = "inverse.gamma", df = df, factor = t(chol(P))) # with 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)
stopifnot(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))

### Examples for rNorm_sumconstr() ############################################
set.seed(271)
weights <- c(1, 1)
Z.constr <- rNorm_sumconstr(n, weights = c(1, 1), s = 2)
stopifnot(all(rowSums(Z.constr ) == 2))
plot(Z.constr , xlab = expression(Z[1]), ylab = expression(Z[2]))
```

Index