Journal of Statistical Software

April 2022, Volume 102, Issue 2. doi: 10.18637/jss.v102.i02

Multivariate Normal Variance Mixtures in R: The R Package nvmix

Erik Hintz
University of Waterloo

Marius Hofert
University of Waterloo

Christiane Lemieux
University of Waterloo

Abstract

We present the features and implementation of the R package **nvmix** for the class of normal variance mixtures including Student t and normal distributions. The package provides functionalities for such distributions, notably the evaluation of the distribution and density function as well as likelihood-based parameter estimation. The distributional family is specified through the quantile function of the underlying mixing random variable. The R package **nvmix** thus allows one to model multivariate distributions well beyond the classical multivariate normal and t case. Additional functionalities include graphical goodness-of-fit assessment, the estimation of the risk measures value-at-risk and expected shortfall for univariate normal variance mixture distributions and functions to work with normal variance mixture copulas, such as sampling and the evaluation of normal variance mixture copulas and their densities. Furthermore, the package **nvmix** also provides functionalities for the evaluation of the distribution and density function as well as random variate generation for the more general class of grouped normal variance mixtures.

Keywords: multivariate normal variance mixtures, Student t, Gauss, distribution function, density, random number generation.

1. Introduction

The class of multivariate normal variance mixtures including the multivariate normal and the (Student) t distributions, belongs to the most widely used classes of multivariate distributions in applications in statistics, finance, insurance and risk management. A d-dimensional random vector $\boldsymbol{X} = (X_1, \dots, X_d)$ follows a normal variance mixture, denoted by $\boldsymbol{X} \sim \mathrm{NVM}_d(\boldsymbol{\mu}, \Sigma, F_W)$, if, in distribution,

$$\boldsymbol{X} = \boldsymbol{\mu} + \sqrt{W} A \boldsymbol{Z}, \tag{1}$$

where $\boldsymbol{\mu} \in \mathbb{R}^d$ denotes the location (vector), $\Sigma = AA^\top$ (for $A \in \mathbb{R}^{d\times k}$) is the scale (matrix) (a covariance matrix), and $W \sim F_W$ is a non-negative random variable independent of $\boldsymbol{Z} \sim \mathrm{N}_k(\mathbf{0}, I_k)$ (where $I_k \in \mathbb{R}^{k\times k}$ denotes the identity matrix); see, for example, McNeil, Frey, and Embrechts (2015, Section 6.2). As $A\boldsymbol{Z} \sim \mathrm{N}_d(\mathbf{0}, \Sigma)$ for any matrix A satisfying $AA^\top = \Sigma$, (1) is a valid definition irrespective of the particular choice of the matrix A. Note that this definition also allows for Σ to be a singular matrix. For some applications, however, such as computing the log-density function or studying the Mahalanobis distance of $\boldsymbol{X}$ from $\boldsymbol{\mu}$ with respect to Σ as discussed below, we do need that Σ has full rank.

It is easy to see from (1) that $\boldsymbol{X} \mid W \sim \mathrm{N}_d(\boldsymbol{\mu}, W\Sigma)$, so that W indeed "mixes" the covariance matrix of a multivariate normal distribution. The mixing variable W can be viewed as a shock affecting the (co)variance matrix of all components in $\boldsymbol{X}$. If $\mathsf{E}(\sqrt{W}) < \infty$, then $\mathsf{E}(\boldsymbol{X}) = \boldsymbol{\mu}$. If $\mathsf{E}(W) < \infty$, then $\mathsf{COV}(\boldsymbol{X}) = \mathsf{E}(W\Sigma) + \mathsf{COV}(\boldsymbol{\mu} + \sqrt{W}\mathbf{0}) = \mathsf{E}(W)\Sigma$, so that $\mathsf{CORR}(\boldsymbol{X}) = P$, the correlation matrix corresponding to Σ.

It is well known that components of a multivariate normal random vector are independent if and only if they are uncorrelated. The multivariate normal distribution is in fact the only normal variance mixture distribution with this property as can be seen from (1): if A (and therefore Σ) is a diagonal matrix, i.e., when the components of $\boldsymbol{X}$ are uncorrelated, they are independent if and only if W is constant almost surely so that $\boldsymbol{X}$ is multivariate normal; see McNeil *et al.* (2015, Lemma 6.5) for a proof. A big advantage of normal variance mixtures is that they can achieve a large range of multivariate distributions well beyond the multivariate normal, with different (joint and marginal) tail behavior, including tail dependence, while keeping some of the desirable properties of the multivariate normal, such as closedness with respect to linear combinations; see McNeil *et al.* (2015, Section 6.2).

If Σ has full rank, the distribution of the squared Mahalanobis distance of $\boldsymbol{X}$ in (1) given by

$$D^2(\boldsymbol{X}; \boldsymbol{\mu}, \Sigma) = (\boldsymbol{X} - \boldsymbol{\mu})^\top \Sigma^{-1} (\boldsymbol{X} - \boldsymbol{\mu}) \tag{2}$$

can be useful for statistical purposes, such as graphical goodness-of-fit assessment. Note that $D^2(\boldsymbol{X}; \boldsymbol{\mu}, \Sigma) \mid W \sim \Gamma(d/2, 2W)$ where $\Gamma(\alpha, \beta)$ denotes a gamma distribution with shape $\alpha > 0$ and scale $\beta > 0$. We therefore refer to $D^2(\boldsymbol{X}; \boldsymbol{\mu}, \Sigma)$ in (2) as a gamma mixture model.

The R (R Core Team 2021) package **nvmix** (Hofert, Hintz, and Lemieux 2022), available from the Comprehensive R Archive Network (CRAN) at `https://CRAN.R-project.org/package=nvmix`, provides functionalities for working with multivariate normal variance mixtures specified via the quantile function $F_W^\leftarrow$ of W which for $u \in (0, 1)$ is defined by $F_W^\leftarrow(u) = \inf\{w : F_W(w) \ge u\}$. In particular, package **nvmix** can, among other things, be used to perform the four main tasks for multivariate distributions:

1. Sampling from $\boldsymbol{X}$ based on (1).
2. Evaluating the cumulative distribution function (cdf) of $\boldsymbol{X}$ in (1).
3. Evaluating the (log-)density function of $\boldsymbol{X}$ in (1).
4. Likelihood-based estimation of the parameters $\boldsymbol{\mu}$, Σ and the parameters of W given an independent and identically distributed (iid) sample $\boldsymbol{X}_1, \dots, \boldsymbol{X}_n$ from $\mathrm{NVM}_d(\boldsymbol{\mu}, \Sigma, F_W)$.

The first task is straightforward based on (1) and the Cholesky factor of Σ. The sampling procedure is implemented in the function `rnvmix()`. The remaining tasks, however, are typically challenging. Evaluating the distribution function, for instance, requires numerical approximation of a d-dimensional integral. In the R package **nvmix**, randomized quasi-Monte Carlo (RQMC) and adaptive algorithms that were derived in Hintz, Hofert, and Lemieux (2021) are implemented to perform the three remaining tasks following the conventional syntax [`r`/`p`/`d`/`fit`]`nvmix()`. The quantile function of the mixing random variable W can be passed to these functions via the argument `qmix`. Here is a first example for the tasks 1) to 4):

```
R> library("nvmix")
R> d <- 3
R> scale <- diag(d)
R> loc <- rep(0, d)
R> df <- 4.1
R> n <- 200
R> x <- 1:d
R> qmix <- function(u, df) 1 / qgamma(1-u, shape = df/2, rate = df/2)
R> rt <- rnvmix(n, qmix = qmix, loc = loc, scale = scale, df = df)
R> pt <- pnvmix(x, qmix = qmix, loc = loc, scale = scale, df = df)
R> dt <- dnvmix(x, qmix = qmix, loc = loc, scale = scale, df = df)
R> fit_t <- fitnvmix(rt, qmix = qmix, mix.param.bounds = c(0.5, 10))
```

Here, `qmix` is the quantile function of an inverse-gamma distribution with shape and rate parameter `df/2`, so that the resulting mixture is multivariate t with `df` degrees of freedom; see page 13. By default, the function `fitnvmix()` uses the variable name `nu` for parameters of the mixing distribution. That is, the variable `nu` in the `fit_t` object corresponds to the estimated degrees of freedom.

In order to work with the squared Mahalanobis distance $D^2(\boldsymbol{X}; \boldsymbol{\mu}, \Sigma)$ in (2), the package **nvmix** also provides the functions [`p`/`d`/`q`/`r`]`gammamix()` which can be used to evaluate the distribution, density and quantile function of gamma-mixture models as well as for sampling.

We highlight that with the functionalities provided by package **nvmix**, one can investigate various normal variance mixture models by merely specifying a quantile function for the mixing distribution. This saves the work of manually implementing lengthy fitting routines and estimation procedures for the distribution and density function tailored to the mixture under consideration. Furthermore, crude estimation procedures for the distribution function (for instance, based on sampling from the normal variance mixture and accepting points that fall within the desired range) typically provide much less efficient estimators, as demonstrated in Genz and Bretz (2002) for the multivariate t distribution.

The remainder of this paper is organized as follows: Section 2 provides a brief overview of other R packages related to **nvmix**. Section 3 explains the algorithms implemented in the more involved functions `dnvmix()`, `pnvmix()` and `fitnvmix()`. Section 4 illustrates how package **nvmix** can be used through various examples. Section 5 includes an example of an application to a financial dataset. Section 6 defines grouped normal variance mixtures and illustrates how package **nvmix** can be used to work with this class of distributions while Section 7 concludes this paper.

2. R packages for multivariate t and related distributions

For estimating multivariate normal and t probabilities in particular, various packages are available, such as the R package **mnormt** (Azzalini and Genz 2020) or the R package **mvtnorm** (Genz, Bretz, Miwa, Mi, and Hothorn 2021). The latter is one of the most widely used packages according to reverse depends (Eddelbuettel 2012) and provides methods for the distribution, density, and equicoordinate quantile functions as well as a sampling procedure. However, evaluating the multivariate t with non-integer degrees of freedom is not possible in **mvtnorm**, and neither is fitting. Another software package focusing on evaluating multivariate normal and t probabilities is the R package **tlrmvnmvt** (Cao, Genton, Keyes, and Turkiyyah 2022). The focus of this package lies on evaluating very high dimensional normal and t probabilities by exploiting low-rank covariance structures. The R packages **sn** (Azzalini 2021) and **tmvtnorm** (Wilhelm and Manjunath 2015) provide functionalities for skew and truncated normal and t distributions, respectively.

The parameters of a multivariate t distribution are typically estimated using expectation-maximization-like algorithms. For instance, an ECME (expectation constrained maximization either) algorithm for estimating the location $\boldsymbol{\mu}$, scale Σ and degrees of freedom ν is provided by the function `fit.mst()` in the R package **QRM**; see Pfaff and McNeil (2020). The R package **MVT** (Osorio 2015) provides the function `studentFit()` which allows one to estimate $\boldsymbol{\mu}$ and Σ for given degrees of freedom.

To the best of our knowledge, there has not been any software available for the R environment for statistical computing and graphics that can be used to perform the four important tasks of sampling, evaluating the density and distribution function as well as fitting for the class of multivariate normal variance mixture distributions when only the quantile function of the mixing random variable W is available in the form of a "black box".

3. The underlying algorithms

This section provides a high-level description of the algorithms underlying the main functions `pnvmix()`, `dnvmix()` and `fitnvmix()` of the R package **nvmix**. For a more detailed derivation and theoretical treatment, see Hintz *et al.* (2021).

3.1. Estimating probabilities

The function `pnvmix()` can be used to estimate the cdf and, more generally, quadrant probabilities of $\boldsymbol{X} \sim \mathrm{NVM}_d(\boldsymbol{\mu}, \Sigma, F_W)$. Here, we are interested in estimating

$$F(\boldsymbol{a}, \boldsymbol{b}) = \mathsf{P}(\boldsymbol{a} < \boldsymbol{X} \le \boldsymbol{b}) = \mathsf{P}(a_1 < X_1 \le b_1, \dots, a_d < X_d \le b_d)$$

for vectors $\boldsymbol{a}, \boldsymbol{b} \in \bar{\mathbb{R}}^d = (\mathbb{R} \cup \{-\infty, \infty\})^d$ and $\boldsymbol{a} < \boldsymbol{b}$ componentwise. Note that the cdf of $\boldsymbol{X}$ is recovered by setting $\boldsymbol{a} = (-\infty, \dots, -\infty)$; this is also the default behavior of the function `pnvmix()`.

Based on ideas of Genz (1992); Genz and Bretz (1999, 2002, 2009), Hintz *et al.* (2021) use a conditioning argument as well as a series of transformations to obtain

$$F(\boldsymbol{a}, \boldsymbol{b}) = \int_{(0,1)^d} g(\boldsymbol{u}) \,\mathrm{d}\boldsymbol{u},$$

where $g(\boldsymbol{u}) = \prod_{i=1}^d g_i(u_0, \dots, u_{i-1})$, $\boldsymbol{u} = (u_0, \dots, u_{d-1}) \in (0,1)^d$ and $g_i(u_0, \dots u_{i-1}) = e_i - d_i$ for $i = 1, \dots, d$. The d_i's are recursively defined by

$$d_1 = d_1(u_0) = \Phi\left(\frac{a_1 - \mu_1}{C_{11}\sqrt{F_W^{\leftarrow}(u_0)}}\right),$$
$$d_i = d_i(u_0, \dots, u_{i-1}) = \Phi\left(\frac{1}{C_{ii}}\left(\frac{a_i - \mu_i}{\sqrt{F_W^{\leftarrow}(u_0)}} - \sum_{j=1}^{i-1} C_{ij}\Phi^{-1}(d_j + u_j(e_j - d_j))\right)\right),$$

for $i = 2, \dots, d$ and the e_i are of the same form as the d_i with a_i replaced by b_i for $i = 1, \dots, d$. Here, $C = (C_{ij})_{i,j=1}^d$ denotes the Cholesky factor of Σ, i.e., a lower triangular matrix satisfying $CC^\top = \Sigma$ and $\Phi(x) = 1/\sqrt{2\pi}\int_{-\infty}^x e^{-t^2/2}\,\mathrm{d}t$ denotes the cdf of the standard normal distribution. The above representation holds for the case when Σ has full rank (so that $C_{ii} > 0$ for $i = 1, \dots, d$). A similar formula can be derived for the singular case; see Hintz *et al.* (2021, Section 3.4) for details.

Internally, the function `pnvmix()` first reorders the limit vectors $\boldsymbol{a}$, $\boldsymbol{b}$, scale matrix Σ and the location vector $\boldsymbol{\mu}$ in a way so that the overall variance of the integrand g is reduced. Then, an iterative RQMC method is used to estimate $F(\boldsymbol{a}, \boldsymbol{b})$: In each iteration, the integrand g is evaluated at a randomized low-discrepancy point-set until the default or user-supplied absolute (or relative) error tolerance is met. For performance reasons, the evaluation of the integrand g is performed in C. For the generation of quasi-random numbers, **nvmix** imports the R package **qrng** of Hofert and Lemieux (2020) and by default uses a randomized Sobol' sequence (obtained by calling `sobol()`). This is also the default for all other RQMC-based methods in package **nvmix**. The type of uniforms used can be changed to a generalized Halton sequence (in which case `pnvmix()` calls `ghalton()` of **qrng**) or to pseudo-random, in which case the point-set is generated via `runif()`; for details on changing such hyperparameters, see `?get_set_param()`.

3.2. Estimating the log-density function

For likelihood-based methods, it is necessary to evaluate the log-density function of a normal variance mixture distribution at a sample. While for some special normal variance mixtures, such as finite mixtures or the multivariate t distribution, the density is available in closed form, this is not the case for a general normal variance mixture distribution. As such, a method capable of estimating the log-density function of any normal variance mixture is needed.

Our proposed method is implemented in the function `dnvmix()`, which efficiently estimates the density $f(\boldsymbol{x})$ for $\boldsymbol{x} \in \mathbb{R}^d$ of $\boldsymbol{X} \sim \mathrm{NVM}_d(\boldsymbol{\mu}, \Sigma, F_W)$. We first outline the main idea behind the method implemented in `dnvmix()`; for more details, see Hintz *et al.* (2021, Section 4).

In order for the density to exist, Σ must have full rank. A conditioning argument then yields

$$f(\boldsymbol{x}) = \int_0^\infty f_{\boldsymbol{X}|W}(\boldsymbol{x} \mid w)\,\mathrm{d}F_W(w) = \int_0^1 h(u)\,\mathrm{d}u,$$

where

$$h(u) = \frac{1}{\sqrt{(2\pi F_W^{\leftarrow}(u))^d|\Sigma|}}\exp\left(-\frac{D^2(\boldsymbol{x}; \boldsymbol{\mu}, \Sigma)}{2F_W^{\leftarrow}(u)}\right), \quad u \in (0,1).$$

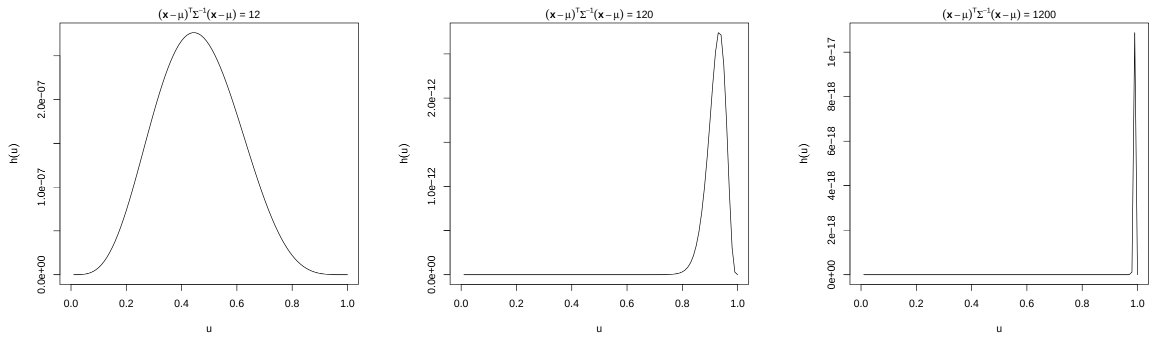

Figure 1: Integrand h for a 10-dimensional t distribution with 2.2 degrees of freedom.

Estimating $f(\boldsymbol{x})$ therefore requires the approximation of a univariate integral.

At first glance, $f(\boldsymbol{x})$ may be approximated via

$$f(\boldsymbol{x}) \approx \widehat{f(\boldsymbol{x})} = \frac{1}{n}\sum_{i=1}^{n} h(u_i),$$

where $u_i \in (0,1)$ for $i = 1,\dots,n$ are either pseudo- or quasi-random numbers and n is chosen so that the error estimate meets some pre-specified tolerance. In practice, however, it is often required to compute the logarithmic density $\log f(\boldsymbol{x})$ rather than $f(\boldsymbol{x})$. In order to obtain a numerically more robust estimator for $\log f(\boldsymbol{x})$ than just taking $\log \widehat{f(\boldsymbol{x})}$, we define the function LSE (logarithmic sum of exponentials) by

$$\mathrm{LSE}(c_1,\dots,c_n) = \log\left(\sum_{i=1}^{n}\exp(c_i)\right) = c_{\max} + \log\left(\sum_{i=1}^{n}\exp(c_i - c_{\max})\right),$$

where $c_1,\dots,c_n \in \mathbb{R}$ and $c_{\max} = \max\{c_1,\dots,c_n\}$. The right-hand side of this equation is numerically more stable than the left-hand side as the sum inside the logarithm is bounded between 1 and n. Applied to our problem, we can use

$$\widehat{\log f(\boldsymbol{x})} = -\log(n) + \mathrm{LSE}\left(\log h(u_1),\dots,\log h(u_n)\right). \tag{3}$$

Note that mathematically, $\widehat{\log f(\boldsymbol{x})} = \log \widehat{f(\boldsymbol{x})}$ but numerically, $\widehat{\log f(\boldsymbol{x})}$ is more stable, especially for large Mahalanobis distances $D^2(\boldsymbol{x};\boldsymbol{\mu},\Sigma)$ that yield very small (negative) values of $\log h(u_i)$. We refer to the estimator in (3) as crude estimator.

It turns out that the estimator $\widehat{\log f(\boldsymbol{x})}$ performs poorly when the input $\boldsymbol{x}$ has a large Mahalanobis distance $D^2(\boldsymbol{x};\boldsymbol{\mu},\Sigma)$. To see why, Figure 1 shows the integrand h in the special case when W follows an inverse-gamma distribution (so that $\boldsymbol{X}$ is multivariate t). When the Mahalanobis distance $D^2(\boldsymbol{x};\boldsymbol{\mu},\Sigma)$ is large, most of the mass of the integrand is concentrated on a small subinterval of $(0,1)$ and this relevant area may be undersampled or not be sampled from at all.

To overcome this issue, we suggest using an adaptive RQMC procedure that samples mostly in a subinterval $(u_l,u_r) \subset (0,1)$ around the peak of $h(u)$ encompassing the maximum, and we use crude trapezoidal rules in the remaining intervals $(0,u_l)$ and $(u_r,1)$ that do not significantly contribute to the value of $f(\boldsymbol{x})$ anyway. Denote by $h_{\max} := \max_{u\in(0,1)}\{h(u)\}$ the maximum value of h and by u^* the maximizer; see Hintz *et al.* (2021, Lemma 5.1) for details. One can show that for any threshold ε_{th} satisfying $0 < \varepsilon_{\text{th}} < h_{\max}$ there exists an interval (u_l, u_r) with $u^* \in (u_l, u_r)$ such that $h(u) > \varepsilon_{\text{th}}$ for $u \in (u_l, u_r)$ and $h(u) \leq \varepsilon_{\text{th}}$ for $u \in (0,1) \setminus (u_l, u_r)$. This result holds for any normal variance mixture with continuous mixing random variable W supported on $[0, \infty)$; note that when W is discrete, the density is known analytically and no estimation is necessary. The idea is then to choose ε_{th} in a way such that the intervals $(0, u_l)$ and $(u_r, 1)$ do not significantly contribute to the density (for instance, if they are $k = 10$ orders smaller than $h_{\max}$), and then obtain the values u_l, u^* and u_r using bisections.

It is typically necessary to evaluate the density function at several inputs $\boldsymbol{x}_1, \dots, \boldsymbol{x}_N$. For instance, for likelihood-based methods the log-density of a sample $\{\boldsymbol{x}_1, \dots \boldsymbol{x}_N\}$ needs to be optimized over some parameter space. For performance reasons it is then highly desirable that the log-density function is vectorized, as otherwise calls to the likelihood function are too time-consuming. On the one hand, the above adaptive procedure is input-dependent (different inputs $\boldsymbol{x}$ will give different maximizers u^* and thus different subintervals (u_l, u_r)). On the other hand, the crude procedure (which always samples over the whole domain of W) can be vectorized easily by using the same random numbers, that is, the same $F_W^{\leftarrow}(u_i)$ for all inputs. Furthermore, the crude estimator from (3) works well for inputs $\boldsymbol{x}$ with small Mahalanobis distance.

Hintz *et al.* (2021, Algorithm 4.3) suggests a combination of the crude and adaptive approach that only uses the adaptive procedure when necessary and also re-uses expensive quantile evaluations $F_W^{\leftarrow}(u)$. The following algorithm gives a high-level summary of the algorithm implemented in the function `dnvmix()` of the R package **nvmix**. Given inputs $\boldsymbol{x}_1, \dots, \boldsymbol{x}_{n'}$, the order k to determine the threshold ε_{th} and an error tolerance ε, estimate $\log f(\boldsymbol{x}_l)$, $l = 1, \dots, n'$, as follows:

1. Use the crude estimator from (3) with small n, to obtain estimates $\widehat{\log f(\boldsymbol{x}_l)}$, $l = 1, \dots, n'$. Store all uniforms and quantile evaluations $(u, F_W^{\leftarrow}(u))$, in a list, say $\mathcal{L}$.

2. If all estimates $\widehat{\log f(\boldsymbol{x}_l)}$, $l = 1, \dots, n'$ meet the error tolerance ε, go to Step 4. If not, assume after reordering that $\boldsymbol{x}_s$, $s = 1, \dots, n''$ with $1 \leq n'' \leq n'$ are the inputs whose error estimates did not meet ε.

3. For each remaining input $\boldsymbol{x}_s$, $s = 1, \dots, n''$, do the following:

   (a) Determine $h_{\max}$ and ε_{th}.

   (b) Use bisections to find u_l, u^* and u_r where starting values are taken from the stored values in $\mathcal{L}$. Add all additional pairs $(u, F_W^{\leftarrow}(u))$ produced in the bisections to the list $\mathcal{L}$.

   (c) Approximate $\log \int_0^{u_l} h(u)\,\mathrm{d}u$ and $\log \int_{u_r}^1 h(u)\,\mathrm{d}u$ via trapezoidal rules using only values $(u, F_W^{\leftarrow}(u))$ in $\mathcal{L}$. Call these approximations $\hat{\mu}_{(0,u_l)}(\boldsymbol{x}_s)$ and $\hat{\mu}_{(u_r,1)}(\boldsymbol{x}_s)$, respectively.

   (d) Approximate $\log \int_{u_l}^{u_r} h(u)\,\mathrm{d}u$ via a RQMC algorithm with error tolerance ε. Call the approximation $\hat{\mu}_{(u_l,u_r)}(\boldsymbol{x}_s)$.

   (e) Combine
$$\widehat{\log f(\boldsymbol{x}_l)} = \mathrm{LSE}\left(\hat{\mu}_{(0,u_l)}(\boldsymbol{x}_s), \hat{\mu}_{(u_l,u_r)}(\boldsymbol{x}_s), \hat{\mu}_{(u_r,1)}(\boldsymbol{x}_s)\right).$$

4. Return $\widehat{\log f(\boldsymbol{x}_l)}$, $l = 1, \dots, n'$.

The default relative error tolerance for log-density estimates is 10^{-2} which, within a small number of iterations, will typically not be met by estimates of log-density values produced in Step 1 that require the usage of the adaptive procedure. Hence, Step 2 indeed detects poor estimates. This would, however, not be the case if a large number of iterations was used, as in this case error estimates of highly peeked integrals become biased and underestimate the error. Furthermore, if in Step 2 at least one estimate does not meet the error tolerance, by choice of defaults this means that 7 680 pairs of $(u, F_W^{\leftarrow}(u))$ were produced in Step 1. Hence, the errors made in the trapezoidal rules performed in Step 3c to approximate the integrals in $(0, u_l)$ and $(u_r, 1)$ are negligible, especially since these areas do not significantly contribute to the density anyway.

3.3. Parameter estimation for multivariate normal variance mixtures

The function `fitnvmix()` of package **nvmix** can be used to tackle the following task: Given $\boldsymbol{X}_1, \dots, \boldsymbol{X}_n \overset{\text{ind.}}{\sim} \mathrm{NVM}_d(\boldsymbol{\mu}, \Sigma, F_W)$ where F_W has quantile function $F_W^{\leftarrow}(u, \boldsymbol{\nu})$ with unknown parameter vector $\boldsymbol{\nu}$ of length $p_{\boldsymbol{\nu}}$, estimate $\boldsymbol{\nu}$, $\boldsymbol{\mu}$ and Σ.

Internally, `fitnvmix()` uses an ECME algorithm; see Liu and Rubin (1994, 1995). Let $\boldsymbol{\theta} = (\boldsymbol{\nu}, \boldsymbol{\mu}, \Sigma)$ and denote by $\boldsymbol{\theta}_k$ the current value of $\boldsymbol{\theta}$ in iteration k. Furthermore, let

$$\log L^{\mathrm{org}}(\boldsymbol{\nu}, \boldsymbol{\mu}, \Sigma; \boldsymbol{X}_1, \dots, \boldsymbol{X}_n) = \sum_{i=1}^{n} \log f_{\boldsymbol{X}}(\boldsymbol{X}_i; \boldsymbol{\nu}, \boldsymbol{\mu}, \Sigma)$$

denote the original log-likelihood and

$$\begin{aligned}\log L^{\mathrm{cpl}}(\boldsymbol{\theta}; \boldsymbol{X}_1, \dots, \boldsymbol{X}_n, W_1, \dots, W_n) &= \sum_{i=1}^{n} \log f_{\boldsymbol{X},W}(\boldsymbol{X}_i, W_i; \boldsymbol{\theta}) \\ &= \sum_{i=1}^{n} \log f_{\boldsymbol{X}|W}(\boldsymbol{X}_i \,|\, W_i; \boldsymbol{\mu}, \Sigma) + \sum_{i=1}^{n} \log f_W(W_i; \boldsymbol{\nu})\end{aligned}$$

the complete log-likelihood, where $W_1, \dots, W_n$ are (unobserved) independent and identically distributed copies of W. Finally, denote by

$$Q(\boldsymbol{\theta}; \boldsymbol{\theta}_k) = \mathsf{E}(\log L^{\mathrm{cpl}}(\boldsymbol{\theta}; \boldsymbol{X}_1, \dots, \boldsymbol{X}_n, W_1, \dots, W_n) \,|\, \boldsymbol{X}_1, \dots, \boldsymbol{X}_n; \boldsymbol{\theta}_k)$$

the expected value of the complete log-likelihood given the (observed) data $\boldsymbol{X}_1, \dots, \boldsymbol{X}_n$.

The main idea implemented in `fitnvmix()` is as follows:

1. Obtain an initial estimate $\boldsymbol{\theta}_0 = (\boldsymbol{\nu}_0, \boldsymbol{\mu}_0, \Sigma_0)$.

2. For $k = 1, 2, \dots$, repeat until convergence:

   (a) Update $\boldsymbol{\mu}_k$ and Σ_k by maximizing $Q(\boldsymbol{\theta}; \boldsymbol{\theta}_k)$ with respect to $\boldsymbol{\mu}$ and Σ with $\boldsymbol{\nu} = \boldsymbol{\nu}_{k-1}$ kept fixed.

   (b) Update $\boldsymbol{\nu}_k$ by maximizing $\log L^{\mathrm{org}}(\boldsymbol{\nu}, \boldsymbol{\mu}_k, \Sigma_k; \boldsymbol{X}_1, \dots, \boldsymbol{X}_n)$ with respect to $\boldsymbol{\nu}$.

As a convergence criterion we suggest stopping once the relative difference in parameter estimates between two iterations is smaller than a given threshold. In Hintz *et al.* (2021), it is shown that the update in Step 2a merely requires the estimation of weights $\delta_{ki} = \mathsf{E}(1/W_i \,|\, \boldsymbol{X}_i; \boldsymbol{\theta}_k)$ for $i = 1, \dots, n$. These weights can be estimated via an adaptive procedure similar to the one implemented in `dnvmix()`. For the optimization in Step 2b, the likelihood estimated via `dnvmix()` is optimized using `optim()`. Unless supplied by the user, a natural starting value for $\boldsymbol{\mu}$ in Step 1 is $\boldsymbol{\mu}_0 = \bar{\boldsymbol{X}}_n$. If S_n denotes the sample covariance matrix of $\boldsymbol{X}_1, \dots, \boldsymbol{X}_n$, it follows that $\mathsf{E}(S_n) = \mathsf{E}(W)\Sigma$, if it exists. This motivates us to numerically solve the $(p_{\boldsymbol{\nu}} + 1)$-dimensional optimization problem

$$(\boldsymbol{\nu}^*, c^*) = \arg\max_{\boldsymbol{\nu}, c>0} L^{\text{org}}(\boldsymbol{\nu}, \boldsymbol{\mu}_0, cS_n; \boldsymbol{X}_1, \dots, \boldsymbol{X}_n)$$

via `optim()` and to set $\boldsymbol{\nu}_0 = \boldsymbol{\nu}^*$ and $\Sigma_0 = c^* S_n$.

4. Usage of the R package nvmix

4.1. Sampling normal variance mixtures using `rnvmix()`

In practice, notably for Monte Carlo studies, it is often required to draw samples from $\boldsymbol{X} \sim \text{NVM}_d(\boldsymbol{\mu}, \Sigma, F_W)$. The function `rnvmix()` provides a function to sample from $\boldsymbol{X}$ based on the stochastic representation (1). Its synopsis is given by:

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

Note that the function `rnvmix()` also allows the user to specify the mixing random variable W via a non-uniform random variate generator (RVG) as argument `rmix`. This is due to the fact that there are distributions for which it is hard to find the quantile function, but for which sampling procedures exist (for example, for stable distributions). As an example call, consider a normal variance mixture with $W \sim \text{Exp}(1)$. In the first case, `rmix` is provided as a `list` and in the second case, `rmix` is a function interpreted as an RVG.

```
R> rate <- 1
R> n <- 500
R> set.seed(42)
R> r.exp.1 <- rnvmix(n, rmix = list("exp", rate = rate))
R> set.seed(42)
R> r.exp.2 <- rnvmix(n, rmix = function(n) rexp(n, rate = rate))
R> stopifnot(all.equal(r.exp.1, r.exp.2))
```

As a second example, let the mixing random variable W follow a stable distribution with skewness $\beta = 1$ so that it is supported on the positive real line. An RVG for stable distributions is provided by the function `rstable1()` from the R package **copula** of Hofert, Kojadinovic, Maechler, and Yan (2020). The two samples generated in the following code are displayed in Figure 2.

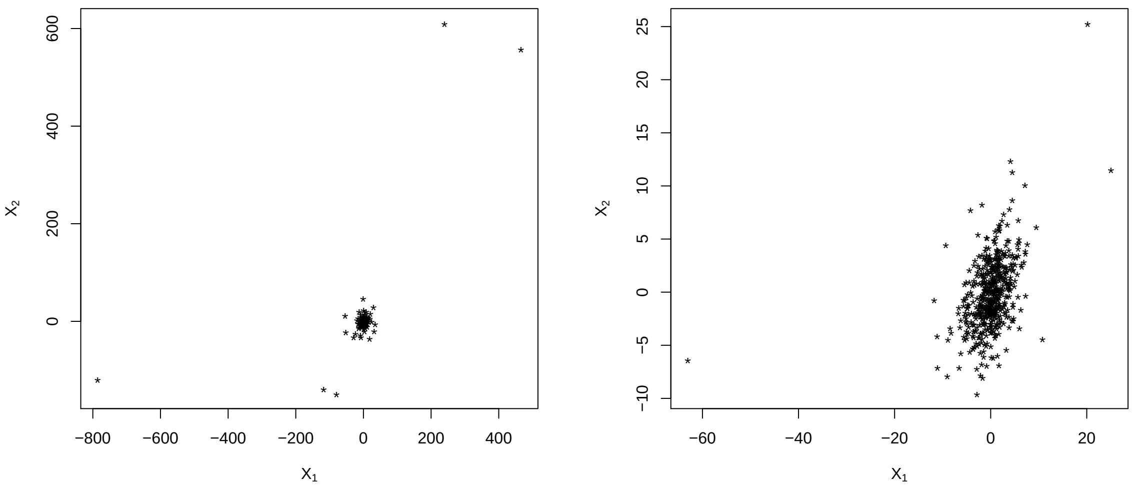

Figure 2: Plots of the samples `r.stable.heavy` and `r.stable.light` from bivariate normal variance mixtures where the mixing variable follows a stable distribution with skewness $\beta = 1$ and characteristic exponents $\alpha = 0.5$ (left) and $\alpha = 0.9$ (right).

```
R> library("copula")
R> scale <- matrix(c(1, 0.5, 0.5, 1), ncol = 2)
R> set.seed(42)
R> r.stable.heavy <- rnvmix(n, rmix = rstable1, scale = scale, alpha = 0.5,
+    beta = 1)
R> r.stable.light <- rnvmix(n, rmix = rstable1, scale = scale, alpha = 0.9,
+    beta = 1)
```

The important argument `method` of `rnvmix()` allows the user to specify whether a pseudo-random sample from $\boldsymbol{X}$ is to be drawn or if instead a low-discrepancy point set should be used to produce the samples. Allowed are the strings `"PRNG"` (the default, classical pseudo-random sampling based on `runif()`) or `"sobol"` or `"ghalton"` (for the inversion method based on the corresponding low-discrepancy point sets). In the latter two cases, `qmix` must be provided. As a third example, consider a three-point mixture distribution with $\boldsymbol{\mu} = (0, 0)$ and $\Sigma = I_2$.

```
R> x <- c(1, 3, 5)
R> p <- c(0.2, 0.3, 0.5)
R> qmix <- function(u) {
+    (u <= p[1]) * x[1] + (u > p[1] & u <= p[1] + p[2]) * x[2] +
+    (u > p[1] + p[2]) * x[3]
+ }
R> r.3pm.pseudo <- rnvmix(n, qmix = qmix)
R> r.3pm.ghalton <- rnvmix(n, qmix = qmix, method = "ghalton")
R> r.3pm.sobol <- rnvmix(n, qmix = qmix, method = "sobol")
```

The samples `r.3pm.pseudo`, `r.3pm.ghalton` and `r.3pm.sobol` are plotted in Figure 3. Note the additional "homogeneity" for the quasi-random samples. We highlight that the quasi-random sample points are typically not independent of each other. That is, while it holds that $\boldsymbol{x}_1, \dots, \boldsymbol{x}_n \overset{\text{ind.}}{\sim} \text{NVM}_d(\boldsymbol{\mu}, \Sigma, F_W)$ for the pseudo-random sample points in `r.3pm.pseudo`,

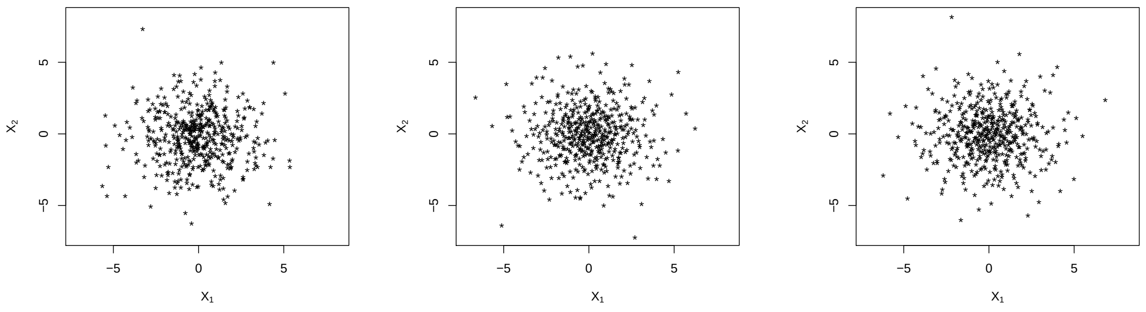

Figure 3: Plots of the samples `r.3pm.pseudo` (pseudo-random sample), `r.3pm.ghalton` (generalized Halton sample) and `r.3pm.sobol` (Sobol' sample) of a bivariate three-point-mixture.

each of the n sample points in `r.3pm.ghalton` and `r.3pm.sobol` follows the specified normal variance mixture but cannot be treated as independent realizations, since the n underlying quasi-random points (that are each mapped to a realization of the normal variance mixture via quantile transformations) are correlated. For more about how to use RQMC in practice, see Lemieux (2009, Section 6.2).

4.2. Estimating probabilities with `pnvmix()`

The main function `pnvmix()`

The function `pnvmix()` has the following structure:

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

Here, `upper` and `lower` correspond to the limit vectors $\boldsymbol{b}$ and $\boldsymbol{a}$ introduced in Section 3.1. The quantile function of W can be specified using the argument `qmix`: This can be either a character string for certain special cases, a list, or a function; see the examples below. As was the case for `rnvmix()`, one can also specify the mixing random variable via an RVG using the argument `rmix`. If `rmix` is supplied, `pnvmix()` uses Monte Carlo (MC) estimation of the probability rather than RQMC. This typically leads to a larger run-time, as RQMC methods substantially outperform purely MC based methods; see Hintz *et al.* (2021, Section 7.2) for details and a numerical study.

As an example, assume that

$$W \sim \mathrm{Exp}(1), \quad \boldsymbol{\mu} = \boldsymbol{0}, \quad \Sigma = \begin{pmatrix} 2 & 0.5 & 0 \\ 0.5 & 2 & 1 \\ 0 & 1 & 2 \end{pmatrix}.$$

The following code estimates $\mathsf{P}(-1 < X_1 \leq 3,\ -2 < X_2 \leq 2,\ -3 < X_3 \leq 1)$ for $\boldsymbol{X} \sim \mathrm{NVM}_3(\boldsymbol{\mu}, \Sigma, F_W)$, first by specifying `qmix` as a `list` and then as a `function`. Due to the random nature of the underlying methods, the result slightly depends on `.Random.seed`.

```
R> a <- -(1:3)
R> b <- 3:1
```

```
R> scale <- matrix(c(2, 0.5, 0, 0.5, 2, 1, 0, 1, 2), ncol = 3)
R> rate <- 1
R> set.seed(42)
R> p1 <- pnvmix(b, lower = a, qmix = list("exp", rate = rate), scale = scale)
R> set.seed(42)
R> p2 <- pnvmix(b, lower = a, qmix = function(u, lambda)
+    -log(1 - u) / lambda, scale = scale, lambda = rate)
R> stopifnot(all.equal(p1, p2))
```

The function `pnvmix()` returns the desired probability, along with an an error estimate and the number of RQMC iterations needed.

```
R> str(p1)
```

```
 num 0.599
 - attr(*, "abs. error")= num 0.000122
 - attr(*, "rel. error")= num 0.000204
 - attr(*, "numiter")= num 1
```

Algorithm-specific parameters can be changed via the argument `control`. For instance, if a higher precision of the computed probability is desired, this can be accomplished by changing the argument `pnvmix.abstol` (which defaults to `1e-3`) in the `control` argument at the expense of a higher run-time.

```
R> (pnvmix(b, lower = a, qmix = function(u, lambda) -log(1 - u) / lambda,
+    lambda = rate, scale = scale, control = list(pnvmix.abstol = 1e-5)))
```

```
[1] 0.5988188
attr(,"abs. error")
[1] 9.851637e-06
attr(,"rel. error")
[1] 1.645178e-05
attr(,"numiter")
[1] 17
```

The wrappers `pNorm()` *and* `pStudent()`

For the two important special cases when the mixing random variable W is constant or inverse-gamma (so that the resulting normal variance mixture follows a multivariate normal or t distribution), `pNorm()` and `pStudent()` are user-friendly wrappers of `pnvmix()`. Note that `pStudent()` works for any positive degrees of freedom (not necessarily integer), a functionality which – to the best of our knowledge – is provided by **nvmix** for the first time in R. This is especially important if t probabilities need to be computed from an estimated t distribution.

We illustrate these wrappers for a singular distribution where the scale matrix Σ is rank deficient. By default, a warning is thrown when the provided scale matrix is singular (unless `verbose = FALSE`).

```
R> A <- matrix(c(1, 0, 0, 0, 2, 1, 0, 0, 3, 0, 0, 0, 4, 1, 0, 1), ncol = 4,
+    byrow = TRUE)
R> scale <- A %*% t(A)
R> upper <- 2:5
R> df <- 1.5
R> pn <- pNorm(upper, scale = scale)

Warning in pgnvmix(upper = upper, lower = lower, qmix = qmix, rmix = rmix, :
  Provided 'scale' is singular.
```

4.3. Estimating the (log-)density with `dnvmix()`

The function `dnvmix()` can be used to compute the (log-)density of any normal variance mixture with full rank scale Σ and has a similar header as `pnvmix()`:

```
dnvmix(x, qmix, loc = rep(0, d), scale = diag(d), factor = NULL,
  control = list(), log = FALSE, verbose = TRUE, ...)
```

But note that there is no argument `rmix()` here, as the underlying algorithm relies on subsampling in certain regions of the support of W for which the quantile function must be provided.

As an example, consider a d-dimensional t distribution with degrees of freedom $\nu > 0$, location $\boldsymbol{\mu}$ and scale Σ, denoted by $\boldsymbol{X} \sim \mathrm{MVT}_d(\nu, \boldsymbol{\mu}, \Sigma)$. The density of $\boldsymbol{X}$ is known and given by

$$f(\boldsymbol{x}) = \frac{\Gamma((\nu+d)/2)}{\Gamma(\nu/2)\sqrt{(\nu\pi)^d|\Sigma|}}\Big(1 + D^2(\boldsymbol{x};\boldsymbol{\mu},\Sigma)\Big)^{-\frac{\nu+d}{2}}, \quad \boldsymbol{x} \in \mathbb{R}^d. \tag{4}$$

The knowledge of this formula can be used to validate our procedure. If we supply `qmix` as a function, the log-density will be estimated; if we supply `qmix = "inverse.gamma"` (or use the wrapper `dStudent()`), `dnvmix()` uses the formula in (4) to compute the density. In the following example, evaluation points are sampled from a 20-dimensional, heavy-tailed t distribution and the log-density of a less heavy-tailed t distribution is computed at the sampled points. For input $\boldsymbol{x}$ with (very) large Mahalanobis distance, it can happen that the procedure fails to provide a reliable error estimate. If this is the case, a warning is thrown.

```
R> set.seed(271)
R> d <- 20
R> df <- 3.9
R> n <- 2000
R> x <- rnvmix(n, qmix = "inverse.gamma", df = df/3, scale = diag(d))
R> dt.1 <- dnvmix(x, qmix = "inverse.gamma", df = df, log = TRUE)
R> dt.2 <- dnvmix(x, qmix = function(u, df) 1 / qgamma(1 - u, shape = df/2,
+    rate = df/2), df = df, log = TRUE)
```

As with `pnvmix()`, algorithm-specific parameters can be passed to `dnvmix()` via the argument `control`. For instance, by setting `dnvmix.doAdapt = FALSE`, `dnvmix()` only uses the crude estimator from (3).

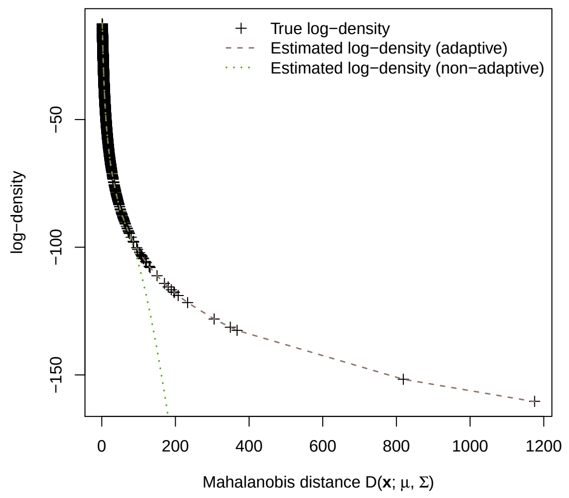

Figure 4: True and estimated log-densities `dt.1`, `dt.2` and `dt.3` for a 20-dimensional t distribution as a function of the Mahalanobis distance $D(\boldsymbol{x}; \boldsymbol{\mu}, \Sigma)$.

```
R> dt.3 <- dnvmix(x, qmix = function(u, df) 1 / qgamma(1 - u, shape = df/2,
+    rate = df/2), df = df, control = list(dnvmix.doAdapt = FALSE),
+    log = TRUE)
```

```
Warning in densmix_(qW, maha2.2 = maha2.2, lconst = lconst, d = d,
  control = control, : Tolerance not reached for all inputs;
  consider increasing 'max.iter.rqmc' in the 'control' argument.
```

As the non-adaptive procedure performs poorly for input $\boldsymbol{x}$ with large Mahalanobis distance, the default error tolerance is not met when the non-adaptive procedure is used. The estimated and true log-densities `dt.1`, `dt.2` and `dt.3` are plotted in Figure 4 as a function of the Mahalanobis distance $D(\boldsymbol{x}; \boldsymbol{\mu}, \Sigma)$. As anticipated, keeping Figure 1 in mind, the non-adaptive approach deteriorates for larger values of $D(\boldsymbol{x}; \boldsymbol{\mu}, \Sigma)$.

4.4. Fitting normal variance mixtures with `fitnvmix()`

The function `fitnvmix()` allows a user to fit normal variance mixtures to data.

```
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)
```

In order to call `fitnvmix()`, three arguments are required: a data matrix `x`, a specification of the mixing random variable W via `qmix`, and the bounds on the mixing parameters as a vector or matrix via `mix.param.bounds`: For a multivariate t distribution, these bounds would correspond to bounds on the degrees of freedom. As before, the mixing variable W is specified via the argument `qmix`, which can be either a character string for certain distributions or a function. In the latter case, `qmix` must be of the form `qmix = function(p, nu)` where `nu` is a `numeric`, so can also be a vector. Furthermore, the user can supply an initial value on the mixing parameter(s) via the argument `nu.init`. The argument `init.size.subsample` determines the size of the subsample used for the $(p_{\boldsymbol{\nu}}+1)$-dimensional optimization to find initial values in Step 1 on page 8. For performance reasons, and since this step is only performed to find an initial value, the default was chosen to be `100`. The argument `size.subsample` determines the size of the subsample used for the optimization in Step 2b on page 8 to find the next estimate of the mixing parameter. By default, `size.subsample = n` so that the full sample is used in these optimizations. While this leads to longer run-times, numerical experiments suggest that using smaller sample sizes leads to flawed estimates.

To illustrate the usage of `fitnvmix()`, consider a Pareto-normal variance mixture, $\boldsymbol{X} \sim \mathrm{PNVM}_d(\nu, \boldsymbol{\mu}, \Sigma)$. Here, $W \sim \mathrm{Par}(\nu, 1)$ has density $f_W(w) = \nu w^{-(\nu+1)}$ for $w \geq 1$. The density of $\boldsymbol{X}$ is then

$$f(\boldsymbol{x}; \mu, \Sigma, \nu) = \frac{\nu}{\sqrt{(2\pi)^d|\Sigma|}} \left(\frac{D^2(\boldsymbol{x}; \boldsymbol{\mu}, \Sigma)}{2}\right)^{-d/2-\nu} \gamma\left(\nu + \frac{d}{2}; \frac{D^2(\boldsymbol{x}; \boldsymbol{\mu}, \Sigma)}{2}\right), \quad \boldsymbol{x} \in \mathbb{R}^d,$$

where $\gamma(z; x) = \int_0^x t^{z-1} e^{-t}\,\mathrm{d}t$ for $z, x > 0$ denotes the (lower) incomplete gamma function. Furthermore, weights $\mathsf{E}(W \mid \boldsymbol{X})$ required for updating $\boldsymbol{\mu}$ and Σ in the ECME iterations can also be derived for $\boldsymbol{X} \sim \mathrm{PNVM}_d(\nu, \boldsymbol{\mu}, \Sigma)$. As an example, we sample $\boldsymbol{X}_1, \ldots, \boldsymbol{X}_n \sim \mathrm{PNVM}_d(\nu, \boldsymbol{\mu}, \Sigma)$ where $\nu = 1.5$, $\boldsymbol{\mu} = (0, \ldots, 0)$ and Σ is randomly generated. We then use `fitnvmix()` to estimate these parameters, first by supplying `qmix` as a string (so that `fitnvmix()` uses closed formulas for weights and densities) and then by supplying the quantile function of W directly so that all weights and densities are estimated.

```
R> set.seed(42)
R> d <- 4
R> n <- 100
R> nu. <- 1.5
R> scale <- cov2cor(tcrossprod(matrix(runif(d * d), ncol = d)))
R> x <- rnvmix(n, qmix = "pareto", alpha = nu., scale = scale)
R> m.p.b <- c(0.1, 50)
R> qmix. <- function(u, nu) (1 - u)^(-1/nu)
R> system.time(fit.par1 <- fitnvmix(x, qmix = "pareto",
+    mix.param.bounds = m.p.b))

   user  system elapsed
  0.053   0.002   0.057

R> system.time(fit.par2 <- fitnvmix(x, qmix = qmix.,
+    mix.param.bounds = m.p.b))

   user  system elapsed
  2.739   0.029   2.784

R> fit.par1

Call: fitnvmix(x = x, qmix = "pareto", mix.param.bounds = m.p.b)
Input data: 100 4-dimensional observations.
```

```
Normal variance mixture specified through quantile function of the mixing
     variable "pareto"
with unknown 'loc' vector and unknown 'scale' matrix.
log-likelihood at reported parameter estimates: -410.348100
Termination after 8 iterations, convergence detected.
Estimated mixing parameter(s) 'nu':
[1] 1.42
Estimated 'loc' vector:
[1]  0.03790 -0.11281  0.02971  0.01733
Estimated 'scale' matrix:
       [,1]   [,2]   [,3]   [,4]
[1,] 0.8960 0.7727 0.7423 0.8947
[2,] 0.7727 0.8389 0.6512 0.7557
[3,] 0.7423 0.6512 0.8311 0.6039
[4,] 0.8947 0.7557 0.6039 1.0230

R> fit.par2

Call: fitnvmix(x = x, qmix = qmix., mix.param.bounds = m.p.b)
Input data: 100 4-dimensional observations.
Normal variance mixture specified through quantile function of the mixing
     variable function (u, nu)  (1 - u)^(-1/nu)
with unknown 'loc' vector and unknown 'scale' matrix.
Approximated log-likelihood at reported parameter estimates: -410.292900
Termination after 16 iterations, convergence detected.
Estimated mixing parameter(s) 'nu':
[1] 1.412
Estimated 'loc' vector:
[1]  0.03800 -0.11296  0.02966  0.01760
Estimated 'scale' matrix:
       [,1]   [,2]   [,3]   [,4]
[1,] 0.8930 0.7701 0.7399 0.8916
[2,] 0.7701 0.8360 0.6491 0.7532
[3,] 0.7399 0.6491 0.8283 0.6019
[4,] 0.8916 0.7532 0.6019 1.0194
```

As expected, the estimates differ slightly when `qmix` is supplied as a function, as in this case, all weights and log-densities are numerically estimated, inevitably leading to small estimation errors and also to a longer run-time.

Note that `fitnvmix()` returns an S3 object of class '`fitnvmix`', which is essentially a list containing, among others, the estimated mixing parameter `nu` as well as the estimated location vector `loc` and scale matrix `scale`. The methods `print()`, `summary()` and `plot()` are defined for the class '`fitnvmix`'.

The function `qqplot_maha()` plots the empirical quantiles $D^2(\boldsymbol{x}_i; \boldsymbol{\mu}, \Sigma)$, $i = 1, \dots, n$, versus their theoretical counterparts; the latter are approximated internally by calling `qgammamix()`. This serves as a graphical goodness-of-fit (GoF) assessment. The structure is as follows:

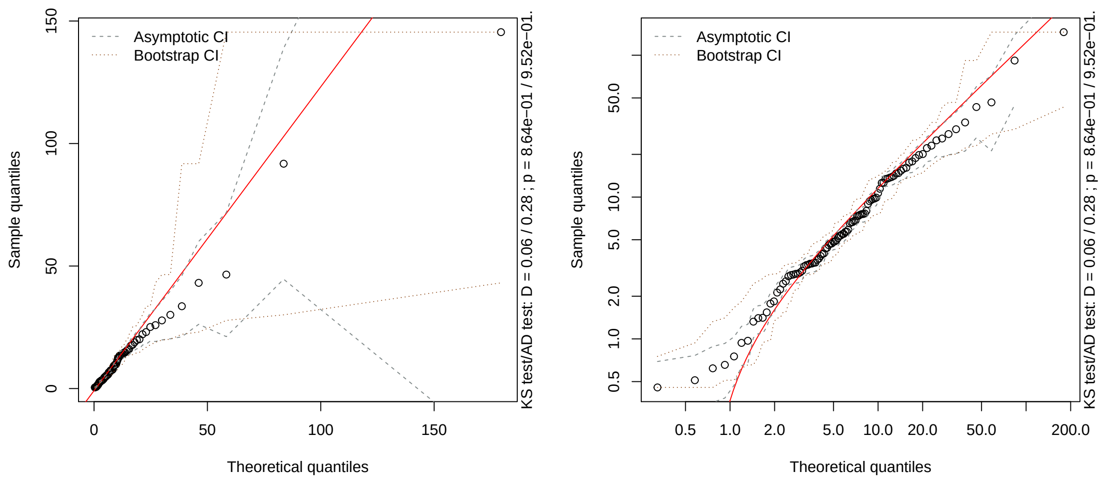

Figure 5: Q-Q plot of the empirical quantiles $D^2(\boldsymbol{x}_i; \boldsymbol{\mu}, \Sigma)$ versus their theoretical counterparts on ordinary scale (left) and log-log scale (right).

```
qqplot_maha(x, qmix, loc, scale, fitnvmix_object, trafo.to.normal = FALSE,
  test = c("KS.AD", "KS", "AD", "none"), boot.pars = list(B = 500,
  level = 0.95), plot = TRUE, verbose = TRUE, control = list(),
  digits = max(3, getOption("digits") - 4), plot.pars = list(), ...)
```

If the argument `trafo.to.normal` is `TRUE`, a probability-quantile transformation is used to construct the Q-Q plot on a normal axis. Besides approximating the theoretical quantiles, the function also computes asymptotic standard errors as in Fox (2015, p. 35–36) and a Bootstrap confidence interval with level `boot.pars$level` and `boot.pars$B` repetitions for the empirical quantiles. Including these confidence intervals helps address the problem of large variations in the ordered samples. As Q-Q plots can merely be used as a first graphical assessment, the function additionally performs statistical GoF tests. In particular, depending on the argument `test`, a Kolmogorov-Smirnov GoF test is performed via `ks.test()` on the univariate Mahalanobis distances, or an Anderson-Darling GoF test in which case `qqplot_maha()` calls `ad.test()` from the R package **ADGofTest**; see Bellosta (2011). By default, both tests are performed. The p values for testing the null hypothesis of having specified the correct distribution and the values of the test-statistic are displayed on the second y-axis. Finally, the argument `plot.pars` which is passed to the underlying `plot()` method can be used to specify plotting parameters, such as logarithmic axes and colors. See `?qqplot_maha()` and `?get_set_qqplot_param()` for details and examples. The output of the following code is displayed in Figure 5.

```
R> set.seed(1)
R> qq.par <- qqplot_maha(x, qmix = "pareto", alpha = fit.par1$nu,
+    loc = fit.par1$loc, scale = fit.par1$scale, plot = FALSE)
R> plot(qq.par)
R> plot(qq.par, plot.pars = list(log = "xy"))
```

The return value of the function `qqplot_maha()` is an object of class '`qqplot_maha`' for which the methods `plot()` and `print()` are defined:

```
R> qq.par

Call: qqplot_maha(x = x, qmix = "pareto", loc = fit.par1$loc,
  scale = fit.par1$scale, plot = FALSE, alpha = fit.par1$nu)

Input: 100 squared Mahalanobis distances.

KS test: D = 0.06, p = 8.65e-01
AD test: D = 0.281, p = 9.52e-01.

Computed results stored in the object:
- theoretical quantiles in $theo_quant;
- sorted, squared Mahalanobis distances in $maha2;
- estimated, asymptotic standard errors in $asymptSE;
- Bootstrap CIs (estimated from 500 resamples) in $boot_CI;
- GoF test results in $testout;
```

As a second example, consider $W = 1/\tilde{W}$ for $\tilde{W} \sim \mathrm{Burr}(\nu_1, \nu_2)$ having cdf $F_{\tilde{W}}(w) = 1 - (1 + w^{\nu_1})^{-\nu_2}$ for $w > 0$; we refer to the resulting mixture as inverse-Burr mixture. It can be easily computed that $F_W^{\leftarrow}(u) = (u^{-1/\nu_2} - 1)^{-1/\nu_1}$ for $u \in (0, 1)$. Note that in this case, there are two mixing parameters (so that $\boldsymbol{\nu}$ has length $p_{\boldsymbol{\nu}} = 2$) and bounds for both, ν_1 and ν_2, need to be passed to `fitnvmix()` via the argument `mix.param.bounds`.

```
R> qmix <- function(u, nu) (u^(-1/nu[2]) - 1)^(-1/nu[1])
R> set.seed(274)
R> nu <- c(2, 5)
R> d <- 5
R> n <- 500
R> scale <-cov2cor(tcrossprod(matrix(runif(d * d), ncol = d)))
R> m.p.b <- matrix(c(0.1, 0.1, 8, 8), ncol = 2)
R> x <- rnvmix(n, qmix = qmix, nu = nu, scale = scale)
R> fit.burr <- fitnvmix(x, qmix = qmix, mix.param.bounds = m.p.b)
R> fit.burr

Call: fitnvmix(x = x, qmix = qmix, mix.param.bounds = mix.param.bounds)
Input data: 500 5-dimensional observations.
Normal variance mixture specified through quantile function of the mixing
     variable function (u, nu)  (u^(-1/nu[2]) - 1)^(-1/nu[1])
with unknown 'loc' vector and unknown 'scale' matrix.
Approximated log-likelihood at reported parameter estimates: -3021.139300
Termination after 27 iterations, convergence detected.
Estimated mixing parameter(s) 'nu':
[1] 2.055 5.494
Estimated 'loc' vector:
[1]  0.009835 -0.020870 -0.036802  0.027952 -0.030785
Estimated 'scale' matrix:
        [,1]    [,2]    [,3]    [,4]    [,5]
```

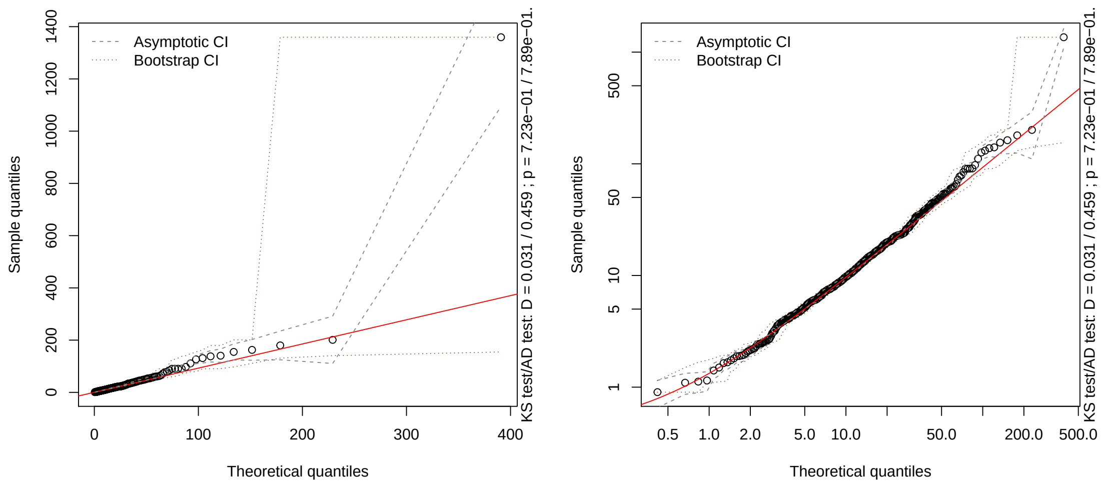

Figure 6: Q-Q plot of the empirical quantiles $D^2(\boldsymbol{x}_i; \boldsymbol{\mu}, \Sigma)$ versus their theoretical counterparts on ordinary scale (left) and log-log scale (right).

```
[1,] 1.0051 0.7493 0.7909 0.7903 0.7403
[2,] 0.7493 1.0268 0.8677 0.2242 0.5882
[3,] 0.7909 0.8677 1.0275 0.3870 0.8727
[4,] 0.7903 0.2242 0.3870 0.9365 0.4883
[5,] 0.7403 0.5882 0.8727 0.4883 1.0067
```

Finally, we call function `qqplot_maha()` by supplying the object `fit.burr` to the argument `fitnvmix_object`, which already contains all necessary information.

```
R> set.seed(1)
R> qq.burr <- qqplot_maha(fitnvmix_object = fit.burr, plot = FALSE)
R> plot(qq.burr)
R> plot(qq.burr, plot.pars = list(log = "xy"))
```

5. Example application

In this section, we demonstrate how package **nvmix** can be used to analyze a multivariate financial dataset. We consider daily return data from the 15 real estate investment trusts (REITs) which are constituents of the S&P 500 index between 2010 and 2012 ($n = 753$ data points). The data are obtained from the R package **qrmdata**; see Hofert, Hornik, and McNeil (2019). We fit marginal ARMA(1, 1)-GARCH(1, 1) models using the packages **qrmtools** (Hofert, Hornik, and McNeil 2021) and **rugarch** (Galanos 2022) and then fit normal variance mixture models to the resulting standardized residuals.

```
R> library("qrmtools")
R> library("rugarch")
R> set.seed(123)
R> data("SP500_const", package = "qrmdata")
```

```
R> time <- c("2010-01-01", "2012-12-31")
R> x <- SP500_const[paste0(time, collapse = "/"),
+    SP500_const_info$Subsector == "REITs"]
R> X <- -returns(x)
R> uspec <- rep(list(ugarchspec(distribution.model = "std")), ncol(X))
R> fit.ARMA.GARCH <- fit_ARMA_GARCH(X, ugarchspec.list = uspec,
+    verbose = FALSE)
R> fits <- fit.ARMA.GARCH$fit
R> resi <- lapply(fits, residuals, standardize = TRUE)
R> X <- as.matrix(do.call(merge, resi))
R> colnames(X) <- colnames(x)
R> n <- nrow(X)
```

In particular, we consider four different models: a multivariate normal, multivariate t (so that W follows an inverse-gamma distribution), a Pareto mixture and an inverse-Burr mixture. We need to specify both the mixing random variable and bounds on its parameters. Note that, with the exception of the inverse-Burr mixture, `qmix` can be provided as a string so that `fitnvmix()` uses closed formulas for densities and weights as opposed to estimating them internally via RQMC methods.

```
R> qmix_ <- list(constant = "constant", inverse.gamma = "inverse.gamma",
+    inverse.burr = function(u, nu) (u^(-1/nu[2]) - 1)^(-1/nu[1]),
+    pareto = "pareto")
R> m.p.b_ <- list(constant = c(0, 1e8), inverse.gamma = c(1, 8),
+    inverse.burr = matrix(c(0.1, 0.1, 8, 8), ncol = 2), pareto = c(1, 8))
```

Now all the ingredients are defined so that `fitnvmix()` can be called with the different specifications of `qmix` and `mix.param.bounds`.

```
R> fit.results <- lapply(1:4, function(i)
+    fitnvmix(X, qmix = qmix_[[i]], mix.param.bounds = m.p.b_[[i]]))
```

In order to compare the models, we plot the fitted log-densities (computed via `dnvmix()`) as functions of the Mahalanobis distances $D(\boldsymbol{x}; \hat{\mu}, \hat{\Sigma})$.

```
R> l.dens <- matrix(NA_real_, ncol = 4, nrow = n)
R> mahas  <- matrix(NA_real_, ncol = 4, nrow = n)
R> for (i in 1:4) {
+    mahas[, i] <- sqrt(mahalanobis(X, center = fit.results[[i]]$loc,
+      cov = fit.results[[i]]$scale))
+    order.maha <- order(mahas[, i])
+    mahas[, i] <- mahas[order.maha, i]
+    l.dens[, i] <- dnvmix(X[order.maha, ], qmix = qmix_[[i]],
+      loc = fit.results[[i]]$loc, scale = fit.results[[i]]$scale,
+      nu = fit.results[[i]]$nu, log = TRUE)
+ }
```

The results are displayed in Figure 7. Evidently, the non-trivial mixtures show heavy tails whereas the fitted multivariate normal does not.

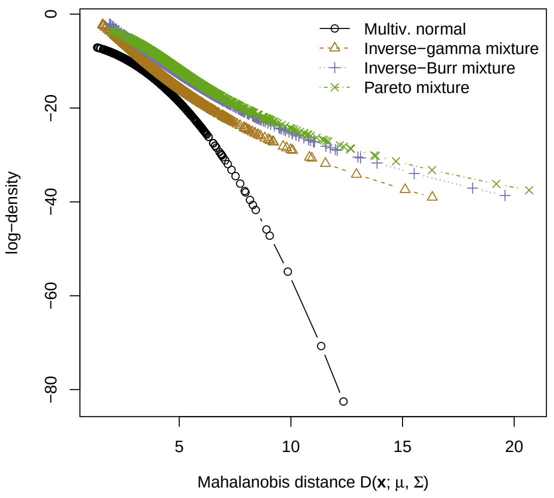

Figure 7: Log-densities as functions of the Mahalanobis distance for four fitted normal variance mixture models using a 15 stock REIT portfolio with data from the SP500 dataset from 2010-01-01 to 2012-12-31 after deGARCHing.

With the estimated parameters at hand, one can call `qqplot_maha()` to plot the observed Mahalanobis distances $D^2(\boldsymbol{x}_i, \hat{\mu}, \hat{\Sigma})$ versus their theoretical quantiles and perform a GoF test. The plots produced by the following code are displayed in Figure 8. As expected, the multivariate normal provides a poor fit as it lacks heavy tails. On the other hand, the fitted Pareto-mixture seems too heavy-tailed to provide a good fit to the data. The inverse-Burr mixture fits better than both the normal and the Pareto, and roughly as well as the inverse-gamma mixture (corresponding to a multivariate t distribution). Based on the provided p values, we can reject the multivariate normal and the Pareto-mixture.

```
R> qq.results <- lapply(1:4, function(i)
+    qqplot_maha(fitnvmix_object = fit.results[[i]]))
```

An important quantity in quantitative risk management is the joint quantile exceedance probability, which for a d-dimensional random vector $\boldsymbol{X} = (X_1, \dots, X_d)$ is defined by

$$Q(u) = \mathsf{P}(X_1 > F^{\leftarrow}_{X_1}(u), \dots, X_d > F^{\leftarrow}_{X_d}(u)), \quad u \in (0,1).$$

That is, $Q(u)$ is the probability that each component in $\boldsymbol{X}$ exceeds its u quantile. As we regard the random variables X_j, $j = 1, \dots, d$, as losses, $Q(u)$ is the probability of a joint large loss, which is a rare event.

By radial symmetry of $\boldsymbol{X} \sim \mathrm{NVM}_d(\boldsymbol{\mu}, \Sigma, F_W)$ and continuity of the marginal distribution functions F_{X_j}, $j = 1, \dots, d$, it follows that for any $u \in (0,1)$,

$$\begin{aligned} Q(u) &= \mathsf{P}(X_1 > F^{\leftarrow}_{X_1}(u), \dots, X_d > F^{\leftarrow}_{X_d}(u)) = \mathsf{P}(X_1 \le F^{\leftarrow}_{X_1}(1-u), \dots, X_d \le F^{\leftarrow}_{X_d}(1-u)) \\ &= \mathsf{P}(F_{X_1}(X_1) \le 1-u, \dots, F_{X_d}(X_d) \le 1-u) = C(1-u, \dots, 1-u) \end{aligned}$$

where $C : [0,1]^d \to [0,1]$ with $C(\boldsymbol{u}) = \mathsf{P}(F_{X_1}(X_1) \le u_1, \dots, F_{X_d}(X_d) \le u_d)$ is the copula of $\boldsymbol{X}$. Such a copula can be evaluated with the function `pnvmixcop()`, which first calls `qnvmix()` to estimate the quantiles $F^{\leftarrow}_{X_j}(u)$ and then calls `pnvmix()` with argument `upper` set to the corresponding quantile estimates.

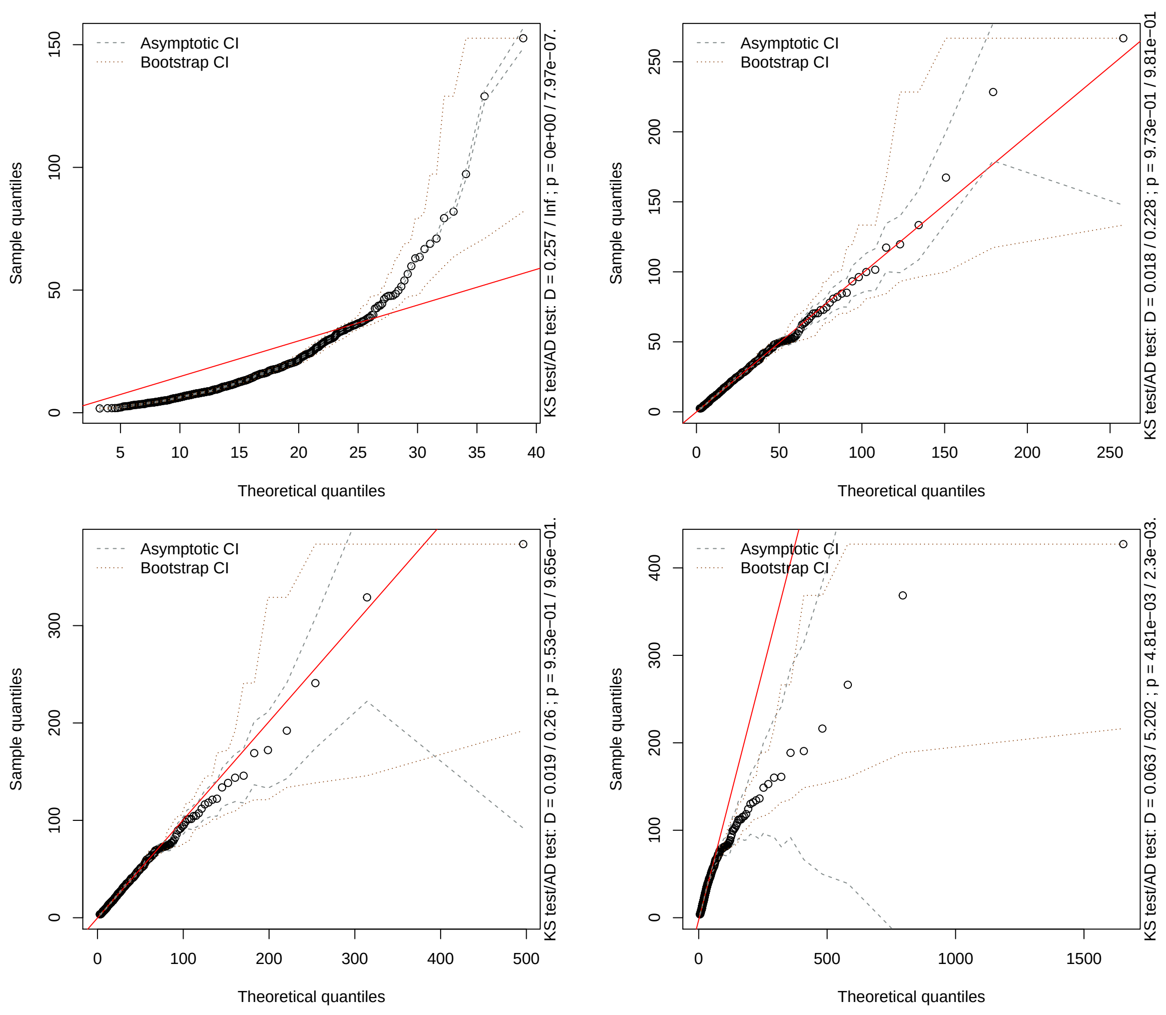

Figure 8: Q-Q plots for four fitted normal variance mixture models using the 15 stock REIT portfolio.

```
R> n. <- 50
R> u. <- seq(0.95, to = 0.999, length.out = n.)
R> u.matrix <- matrix(u., nrow = n., ncol = ncol(X))
R> tailprobs <- sapply(1:4, function(i) pnvmixcopula(1 - u.matrix,
+    qmix = qmix_[[i]], scale = cov2cor(fit.results[[i]]$scale),
+    nu = fit.results[[i]]$nu, control = list(pnvmix.abstol = 1e-5)))
```

Figure 9 displays the estimated probabilities $Q(u)$ for a range of values of u for the four different models, once as is and once standardized by the corresponding multivariate normal probability. As expected from the previous discussion, the heavy-tailed Pareto mixture gives the largest exceedance probabilities. Note that when u is large, the quantile exceedance probability for the non-normal models is larger by orders of magnitude. For instance, $Q^{\text{inverse-Burr}}(0.99)/Q^{\text{Normal}}(0.99) \approx 10$ meaning that such a joint large loss is 10 times more likely if we assume that $\boldsymbol{X}$ follows an inverse-Burr mixture as opposed to a multivariate normal. This highlights the well known fact that multivariate normal variance mixtures are

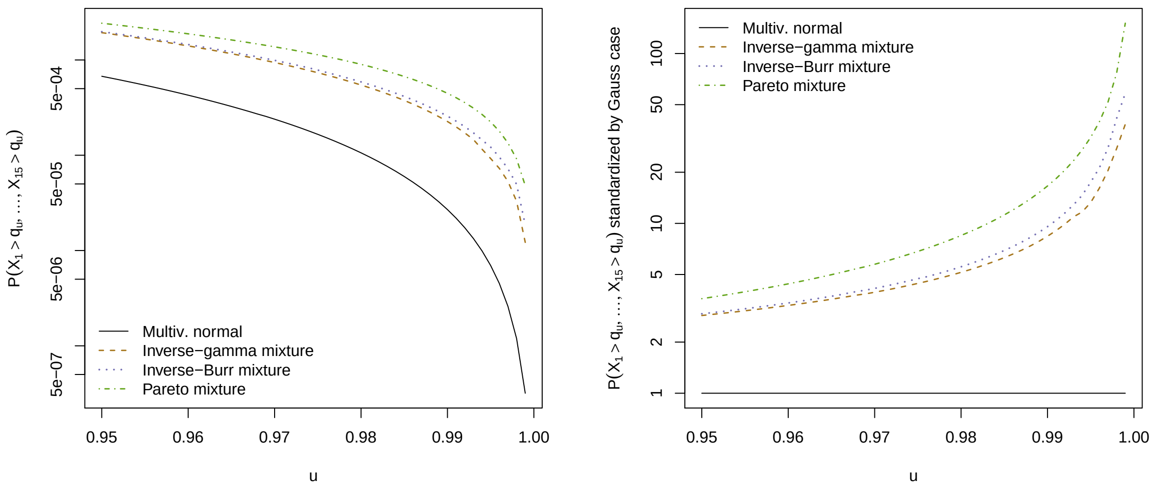

Figure 9: Estimated quantile exceedance probabilities $Q(u)$ for the four different models. The right figure displays the same probabilities $Q(u)$ standardized by the corresponding normal probability.

better suited than the multivariate normal for financial data, due to their inherent heavy tailedness.

To conclude this section, we consider the problem of estimating risk measures for an equal weight portfolio. If $\boldsymbol{X} \sim \mathrm{NVM}_d(\boldsymbol{\mu}, \Sigma, F_W)$, it follows from the stochastic representation (1) of $\boldsymbol{X}$ that $\boldsymbol{a}^\top \mathbf{X} \sim \mathrm{NVM}_1(\boldsymbol{a}^\top\boldsymbol{\mu}, \boldsymbol{a}^\top \Sigma \boldsymbol{a}, F_W)$ for any $\boldsymbol{a} \in \mathbb{R}^d$. If we regard $\boldsymbol{X}$ as a vector of losses from different investments, $\boldsymbol{a}^\top \boldsymbol{X}$ models the loss of a portfolio with portfolio weights given by $\boldsymbol{a}$.

Besides the cdf of $X := \boldsymbol{a}^\top \boldsymbol{X}$, interest often lies in the two important risk measures value-at-risk and expected shortfall; see McNeil *et al.* (2015, Chapter 2). The value-at-risk is defined as the α quantile of the cdf of X, i.e., $\mathrm{VaR}_\alpha(X) = \inf\{x \in \mathbb{R} : F_X(x) \geq \alpha\}$ for $\alpha \in (0,1)$. Such quantile can be estimated with the function `qnvmix()` or with its wrapper `VaRnvmix()`. And the expected shortfall of X at confidence level $\alpha \in (0,1)$ is, provided the integral converges, given by

$$\mathrm{ES}_\alpha(X) = \frac{1}{1-\alpha} \int_\alpha^1 \mathrm{VaR}_u(X)\,\mathrm{d}u;$$

if F_X is continuous, which is the case for normal variance mixtures, one can show that

$$\mathrm{ES}_\alpha(X) = \mathrm{E}(X \mid X > \mathrm{VaR}_\alpha(X)).$$

The function `ESnvmix()` in the R package **nvmix** can be used to estimate expected shortfall for univariate normal variance mixtures. In the case of the normal and t distributions, closed formulas for expected shortfall are known; these formulas are used by `ESnvmix()` if `qmix` is provided as a string.

In the following code, we estimate risk measures of $\boldsymbol{a}^\top \boldsymbol{X}$ for $\boldsymbol{a} = (1, \dots, 1)$ where $\boldsymbol{X} \sim \mathrm{NVM}_{15}(\hat{\mu}, \hat{\Sigma}, F_W)$ for the four different models fitted earlier.

```
R> alpha <- 1 - 1/10^seq(0.5, 3.5, by = 0.05)
R> VaRs <- matrix(NA, ncol = 5, nrow = length(alpha))
```

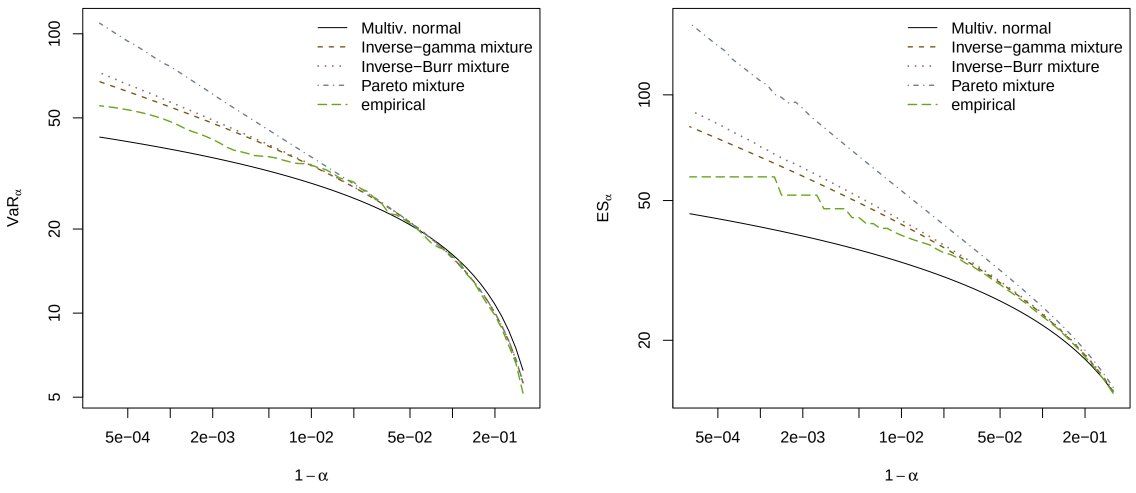

Figure 10: Estimated value-at-risk (left) and expected shortfall (right) for a 15 stock REIT portfolio based on the four different normal variance mixture models fitted to data obtained from the SP500 dataset from 2010-01-01 to 2012-12-31 after deGARCHing.

```
R> ESs <- matrix(NA, ncol = 5, nrow = length(alpha))
R> for (i in 1:4) {
+    VaRs[, i] <- VaR_nvmix(alpha, qmix = qmix_[[i]],
+      loc = sum(fit.results[[i]]$loc), scale = sum(fit.results[[i]]$scale),
+      nu = fit.results[[i]]$nu)
+    ESs[, i] <- ES_nvmix(alpha, qmix = qmix_[[i]],
+      loc = sum(fit.results[[i]]$loc), scale = sum(fit.results[[i]]$scale),
+      nu = fit.results[[i]]$nu)
+ }
```

We can also estimate these risk measures non-parametrically from the data by their respective sample versions for comparison.

```
R> sum.obs <- rowSums(X)
R> VaRs[, 5] <- quantile(sum.obs, probs = alpha)
R> ESs[, 5] <- sapply(seq_along(alpha),
+    function(i) mean(sum.obs[sum.obs > VaRs[i, 5]]))
```

The results are plotted in Figure 10 on log-log scale. For large α, the normal model clearly underestimates both risk measures whereas both the multivariate t and Burr-mixture models provide a better fit. The Pareto-mixture gives too conservative estimates.

We highlight again that all estimation was carried out using the function `qmix` as a "black box". For instance, the inverse-Burr mixture is a model that has not been studied in the literature yet; without functionalities provided by package **nvmix**, rather involved estimation procedures would need to be derived for this model to perform the data analysis undertaken in this section.

6. Grouped normal variance mixtures

An even larger class of multivariate distributions than normal variance mixtures is obtained when the scalar random variable W in (1) is replaced by a vector of comonotone random variables. We say that $\boldsymbol{X}$ follows a grouped normal variance mixture, denoted by $\boldsymbol{X} \sim \mathrm{gNVM}_d(\boldsymbol{\mu}, \Sigma, F_{\boldsymbol{W}})$, if $\boldsymbol{X}$ has stochastic representation

$$\boldsymbol{X} = \boldsymbol{\mu} + \mathrm{diag}(\sqrt{W_1}, \dots, \sqrt{W_d}) A\boldsymbol{Z}, \tag{5}$$

where $\boldsymbol{W} = (W_1, \dots, W_d) \overset{\mathrm{d}}{=} (F^{\leftarrow}_{W_1}(U), \dots, F^{\leftarrow}_{W_d}(U))$ for $U \sim \mathrm{U}(0,1)$ and $F_{\boldsymbol{W}}(\boldsymbol{w}) = \mathsf{P}(\boldsymbol{W} \leq \boldsymbol{w})$. Similarly to normal variance mixtures from (1) where the scalar random variable W can be interpreted as a shock affecting all (co)variances of the underlying multivariate normal, the comonotone random vector $\boldsymbol{W}$ can be regarded as a shock mixing different components of $\boldsymbol{X}$ using different distributions. This gives rise to non-elliptical models well beyond the classical normal variance mixture case. As such, this class includes the multivariate t distribution with multiple degrees of freedom.

Note that the definition of $\boldsymbol{X}$ in (5) does not indicate any grouping yet. We can speak of groups when certain margins of $\boldsymbol{W}$ have the same distribution. More precisely, let $\boldsymbol{W}$ be split into S subvectors $\boldsymbol{W} = (\boldsymbol{W}_1, \dots, \boldsymbol{W}_S)$ where $\boldsymbol{W}_j$ has dimension d_j and $\sum_{j=1}^S d_j = d$. If each $\boldsymbol{W}_j$ has stochastic representation $\boldsymbol{W}_j = (F^{\leftarrow}_{W_j}(U), \dots, F^{\leftarrow}_{W_j}(U))$, all margins of $\boldsymbol{W}_j$ are identical, and the corresponding subvector of $\boldsymbol{X}$ is then a normal variance mixture with mixing distribution F_{W_j}. Even if there is no grouping present, i.e., when $S = d$, we refer to $\boldsymbol{X}$ in (5) as a grouped mixture.

The grouped t copula proposed in Daul, Giorgi, Lindskog, and McNeil (2003) (see also Demarta and McNeil 2005 and Luo and Shevchenko 2010) is the copula of a grouped normal variance mixture where the W_j follow inverse-gamma distributions with potentially different degrees of freedom. Unlike the classical t copula, the grouped t copula allows different pairwise margins to be modeled by different t copulas, thereby leading to more flexible models. The original motivation to derive grouped t copulas led to our definition in (5); see also Hintz, Hofert, and Lemieux (2020) for a theoretical treatment.

Moving from a scalar mixing random variable W to a vector of comonotone random variables $\boldsymbol{W}$ introduces additional computational challenges. Not even the density of a grouped t copula is available in closed form. Package **nvmix** also provides functionalities for grouped normal variance mixtures with the functions [`p/d/r`]`gnvmix()`. The mixing distribution is again specified via the argument `qmix`, which now can be a list to reflect the grouping, along with an additional vector `groupings`. For instance, the function `rgnvmix()` for sampling from grouped mixtures has the following structure:

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

The argument `qmix` is typically a `list` of length S, whereas `groupings` is a vector of length d such that variable j has mixing distribution specified in `qmix[[groupings[j]]]`.

As an illustration, assume $\boldsymbol{X} \sim \mathrm{gNVM}_5(0, \Sigma, F_{\boldsymbol{W}})$ where $W_1 \sim \mathrm{IG}(1,1)$, $W_2 = 1$ a.s., $W_3 \sim \mathrm{Exp}(1)$, $W_4 \sim \mathrm{Par}(2,1)$ and $W_5 = W_1$. That is, marginally, $X_1, X_5 \sim t_2$, $X_2 \sim \mathrm{N}(0,1)$, and X_3 and X_4 are Exponential and Pareto mixtures. The following code samples 1000 iid copies of $\boldsymbol{X}$; see Figure 11 for a pairs plot of the sample. Note how different bivariate margins are of

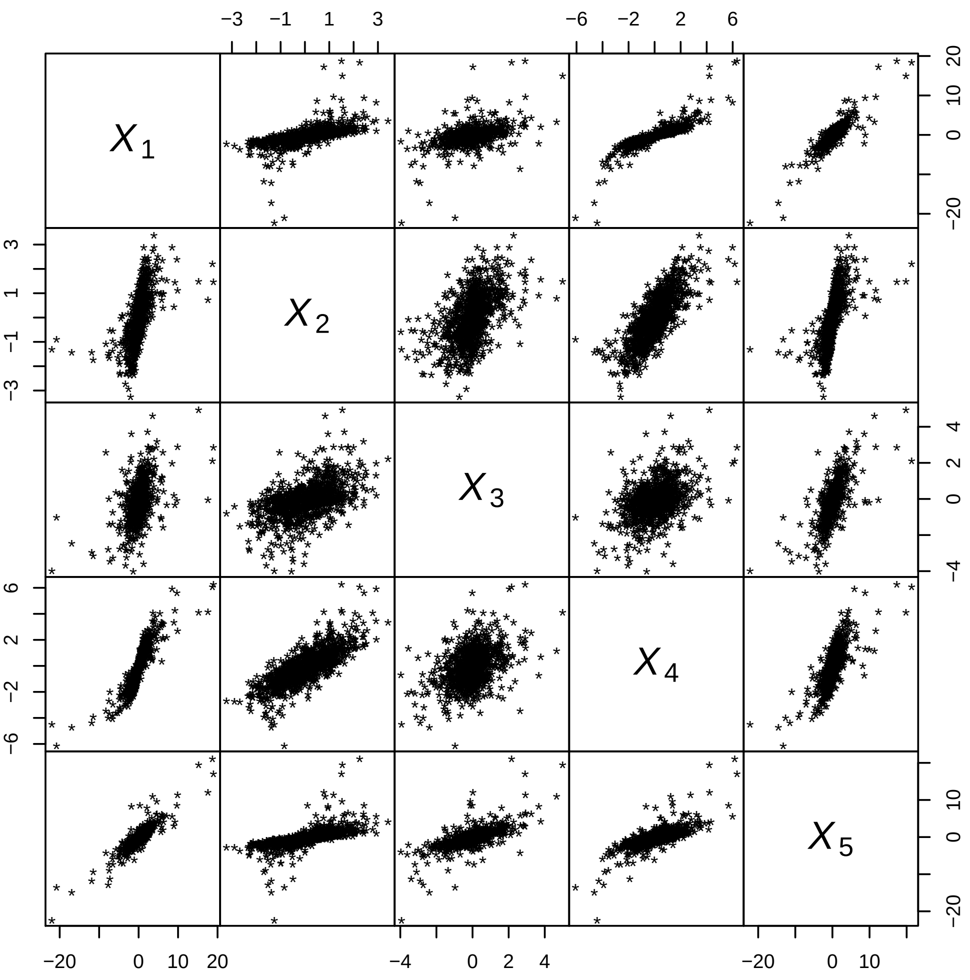

Figure 11: Plot of the samples `r.gnvm` from a 5-dimensional grouped normal variance mixture.

quite different types; this cannot be the case for normal variance mixtures which are elliptical; see McNeil *et al.* (2015, Chapter 6).

```
R> d <- 5
R> df <- 2
R> n <- 1e3
R> set.seed(157)
R> A <- matrix(runif(d * d), ncol = d)
R> scale <- cov2cor(A %*% t(A))
R> groupings <- c(1, 2, 3, 4, 1)
R> qmix <- list(function(u, df) 1 / qgamma(1 - u, shape = df/2,
+    rate = df/2), function(u) rep(1, length(u)), list("exp", rate = 1),
+    function(u) (1 - u)^(-1/2))
R> r.gnvm <- rgnvmix(n, groupings = groupings, qmix = qmix, scale = scale,
+    df = df)
```

The distribution and density function of $\boldsymbol{X}$ can be evaluated using the functions `pgnvmix()` and `dgnvmix()`:

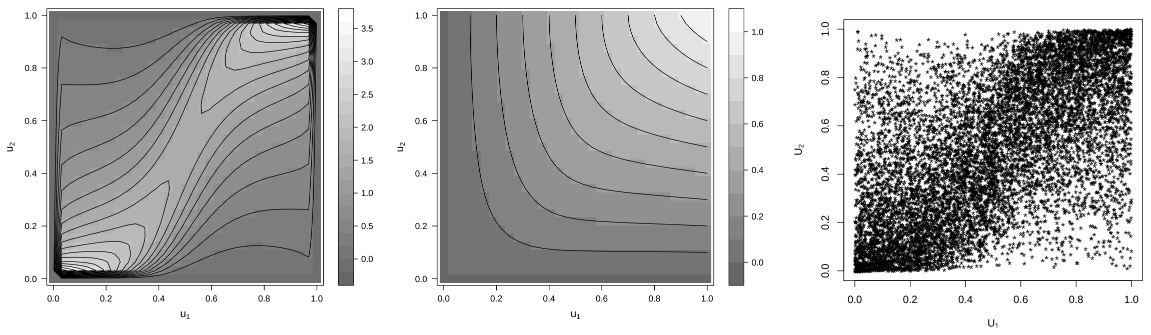

Figure 12: Density (left), distribution function (middle) and a sample plot (right) of a bivariate grouped t-copula with parameter $\rho = 0.7$ and degrees of freedom $(0.5, 35)$.

```
R> b <- 3 * runif(d) * sqrt(d)
R> (pg <- pgnvmix(b, groupings = groupings, qmix = qmix, scale = scale,
+    df = df))

[1] 0.6913258
attr(,"abs. error")
[1] 0.0008712519
attr(,"rel. error")
[1] 0.001260262
attr(,"numiter")
[1] 1

R> (dg <- dgnvmix(b, groupings = groupings, qmix = qmix, scale = scale,
+    df = df))

[1] 9.770276e-06
attr(,"abs. error")
[1] 2.270454e-11
attr(,"rel. error")
[1] 2.323838e-06
attr(,"numiter")
[1] 1
```

Functionalities for the important special case of a grouped t copula are implemented in the functions [d/p/r]gStudentcopula(); not even this important special case has been treated before. In the following example, we compute the density and distribution function of a bivariate grouped t copula with degrees of freedom $(0.5, 35)$; see Figure 12. The degrees of freedom were chosen far apart to make the effect of having two different parameters more visible.

```
R> set.seed(123)
R> d <- 2
R> rho <- 0.7
```

```
R> scale <- matrix(c(1, rho, rho, 1), ncol = 2)
R> df <- c(0.5, 35)
R> r.gStdcop <- rgStudentcopula(1e4, df = df, scale = scale)
R> n.grid <- 33
R> u <- seq(0, 1, length.out = n.grid)
R> grid <- expand.grid("u[1]" = u, "u[2]" = u)
R> dC <- dgStudentcopula(as.matrix(grid), df = df, scale = scale)
R> pC <- pgStudentcopula(as.matrix(grid), df = df, scale = scale)
R> val.pC <- cbind(grid, "C(u[1],u[2])" = pC)
R> val.dC <- cbind(grid, "c(u[1],u[2])" = dC)
```

7. Conclusion

The present paper introduced the R package **nvmix**, which offers a range of functionalities for the class of (grouped and ungrouped) multivariate normal variance mixture distributions. The package can be used to perform main statistical tasks, namely sampling, the estimation of the distribution and density functions as well as parameter estimation. Additionally, some functionalities for graphical goodness-of-fit assessment, for the estimation of the risk measures value-at-risk and expected shortfall as well as for working with normal variance copulas are implemented.

As multivariate normal variance mixtures belong to the most widely used distributions in disciplines such as actuarial science and quantitative risk management, package **nvmix** enjoys a potential wide applicability in practice: Functions provided in **nvmix** allow one to model multivariate data well beyond the classical multivariate normal and t distributions.

Besides demonstrating the algorithms underlying the main functions [`p`/`d`/`fit`]`nvmix`, several examples illustrating the usage of the package were provided. Furthermore, a data analysis on a portfolio of 15 real estate investment trusts was performed; besides the multivariate normal and t, rather new models such as a Pareto and an inverse-Burr mixture model were fitted to the data and studied in more detail.

Even more flexible models than classical normal variance mixtures are obtained by moving from a scalar mixing random variable to a vector of comonotone random variables, giving rise to grouped normal variance mixtures. The package **nvmix** also provides functionalities for this larger class of multivariate distributions; this includes the special case of a grouped t copula.

Acknowledgments

We thank the associate editor and an anonymous reviewer for their comments and suggestions, which helped us improve this paper and our software. The second and third authors would like to thank NSERC (Natural Sciences and Engineering Research Council of Canada) for financial support for this work through Discovery Grant RGPIN-5010-2015 and RGPIN-238959, respectively.

References

Azzalini A (2021). ***sn**: The Skew-Normal and Related Distributions Such as the Skew-t and the SUN.* R package version 2.0.1, URL https://CRAN.R-project.org/package=sn.

Azzalini A, Genz A (2020). ***mnormt**: The Multivariate Normal and t Distributions, and Their Truncated Versions.* R package version 2.0.2, URL https://CRAN.R-project.org/package=mnormt.

Bellosta CJG (2011). ***ADGofTest**: Anderson-Darling GoF Test.* R package version 0.3, URL https://CRAN.R-project.org/package=ADGofTest.

Cao J, Genton MG, Keyes DE, Turkiyyah GM (2022). "**tlrmvnmvt**: Computing High-Dimensional Multivariate Normal and Student-t Probabilities with Low-Rank Methods in R." *Journal of Statistical Software*, **101**(4), 1–25. doi:10.18637/jss.v101.i04.

Daul S, Giorgi ED, Lindskog F, McNeil A (2003). "The Grouped t-Copula with an Application to Credit Risk." doi:10.2139/ssrn.1358956. Available at SSRN 1358956.

Demarta S, McNeil A (2005). "The t Copula and Related Copulas." *International Statistical Review*, **73**(1), 111–129. doi:10.1111/j.1751-5823.2005.tb00254.x.

Eddelbuettel D (2012). "Counting CRAN Package Depends, Imports and LinkingTo." URL http://dirk.eddelbuettel.com/blog/2012/08/05/.

Fox J (2015). *Applied Regression Analysis and Generalized Linear Models.* Sage Publications.

Galanos A (2022). ***rugarch**: Univariate GARCH Models.* R package version 1.4-6, URL https://CRAN.R-project.org/package=rugarch.

Genz A (1992). "Numerical Computation of Multivariate Normal Probabilities." *Journal of Computational and Graphical Statistics*, **1**(2), 141–149. doi:10.1080/10618600.1992.10477010.

Genz A, 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. doi:10.1080/00949659908811962.

Genz A, Bretz F (2002). "Comparison of Methods for the Computation of Multivariate t Probabilities." *Journal of Computational and Graphical Statistics*, **11**(4), 950–971. doi:10.1198/106186002394.

Genz A, Bretz F (2009). *Computation of Multivariate Normal and t Probabilities.* Lecture Notes in Statistics. Springer-Verlag. doi:10.1007/978-3-642-01689-9.

Genz A, Bretz F, Miwa T, Mi X, Hothorn T (2021). ***mvtnorm**: Multivariate Normal and t Distributions.* R package version 1.1-3, URL http://CRAN.R-project.org/package=mvtnorm.

Hintz E, Hofert M, Lemieux C (2020). "Grouped Normal Variance Mixtures." *Risks*, **8**(4), 103. doi:10.3390/risks8040103.

Hintz E, Hofert M, Lemieux C (2021). "Normal Variance Mixtures: Distribution, Density and Parameter Estimation." *Computational Statistics & Data Analysis*, **157**, 107175. doi:10.1016/j.csda.2021.107175.

Hofert M, Hintz E, Lemieux C (2022). ***nvmix**: Multivariate Normal Variance Mixtures.* R package version 0.1-0, URL https://CRAN.R-project.org/package=nvmix.

Hofert M, Hornik K, McNeil A (2021). ***qrmtools**: Tools for Quantitative Risk Management.* R package version 0.0-14, URL https://CRAN.R-project.org/package=qrmtools.

Hofert M, Hornik K, McNeil AJ (2019). ***qrmdata**: Data Sets for Quantitative Risk Management Practice.* R package version 2019-12-03-1, URL https://CRAN.R-project.org/package=qrmdata.

Hofert M, Kojadinovic I, Maechler M, Yan J (2020). ***copula**: Multivariate Dependence with Copulas.* R package version 1.0-1, URL https://CRAN.R-project.org/package=copula.

Hofert M, Lemieux C (2020). ***qrng**: (Randomized) Quasi-Random Number Generators.* R package version 0.0-8, URL https://CRAN.R-project.org/package=qrng.

Lemieux C (2009). *Monte Carlo and Quasi-Monte Carlo Sampling.* Springer-Verlag.

Liu C, Rubin D (1994). "The ECME Algorithm: A Simple Extension of EM and ECM with Faster Monotone Convergence." *Biometrika*, **81**(4), 633–648. doi:10.1093/biomet/81.4.633.

Liu C, Rubin D (1995). "ML Estimation of the t Distribution Using EM and Its Extensions, ECM and ECME." *Statistica Sinica*, **5**(1), 19–39.

Luo X, 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. doi:10.1080/14697680903085544.

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

Osorio F (2015). ***MVT**: Estimation and Testing for the Multivariate t-Distribution.* R package version 0.3, URL https://CRAN.R-project.org/package=MVT.

Pfaff B, McNeil A (2020). ***QRM**: Provides R-Language Code to Examine Quantitative Risk Management Concepts.* R package version 0.4-31, URL https://CRAN.R-project.org/package=QRM.

R Core Team (2021). *R : A Language and Environment for Statistical Computing.* R Foundation for Statistical Computing, Vienna, Austria. URL https://www.R-project.org.

Wilhelm S, Manjunath B (2015). ***tmvtnorm**: Truncated Multivariate Normal and Student t Distribution.* R package version 1.4-10, URL http://CRAN.R-project.org/package=tmvtnorm.

Affiliation:

Erik Hintz, Marius Hofert, Christiane Lemieux
Department of Statistics and Actuarial Science
University of Waterloo
200 University Avenue West
Waterloo, ON, Canada N2L 3G1
E-mail: erik.hintz@uwaterloo.ca, marius.hofert@uwaterloo.ca,
clemieux@uwaterloo.ca
URL: http://uwaterloo.ca/scholar/ehintz/
http://www.math.uwaterloo.ca/~mhofert/
http://www.math.uwaterloo.ca/~clemieux/

Journal of Statistical Software https://www.jstatsoft.org/
published by the Foundation for Open Access Statistics https://www.foastat.org/

April 2022, Volume 102, Issue 2 *Submitted:* 2020-07-02
doi:10.18637/jss.v102.i02 *Accepted:* 2021-09-20