tlrmvnmvt: Computing High-Dimensional Multivariate Normal and Student- t Probabilities with Low-Rank Methods in R

Journal of Statistical Software

January 2022, Volume 101, Issue 4. *doi: 10.18637/jss.v101.i04*

tlrmvnmvt: Computing High-Dimensional Multivariate Normal and Student-t Probabilities with Low-Rank Methods in R

Jian Cao
King Abdullah University of Science and Technology

Marc G. Genton
King Abdullah University of Science and Technology

David E. Keyes
King Abdullah University of Science and Technology

George M. Turkiyyah
American University of Beirut

Abstract

This paper introduces the usage and performance of the R package **tlrmvnmvt**, aimed at computing high-dimensional multivariate normal and Student-t probabilities. The package implements the tile-low-rank methods with block reordering and the separation-of-variable methods with univariate reordering. The performance is compared with two other state-of-the-art R packages, namely the **mvtnorm** and the **TruncatedNormal** packages. Our package has the best scalability and is likely to be the only option in thousands of dimensions. However, for applications with high accuracy requirements, the **TruncatedNormal** package is more suitable. As an application example, we show that the excursion sets of a latent Gaussian random field can be computed with the **tlrmvnmvt** package without any model approximation and hence, the accuracy of the produced excursion sets is improved.

Keywords: excursion sets, high dimensions, multivariate normal, multivariate Student-t, **tlrmvnmvt**.

1. Introduction

The multivariate normal distribution (MVN) is probably the most well-known probability model due to its tractable analytical properties, principally being closed under conditioning and marginalization. The MVN probability arises in many applications. It amounts to a challenging numerical integration problem and becomes the computation bottleneck in high

dimensions. Such examples include Bayes classification (Durante 2019), finding excursion and contour uncertainty regions (Bolin and Lindgren 2015), and maximum likelihood estimation (Cao, Genton, Keyes, and Turkiyyah 2021; Davison, Huser, and Thibaud 2013; Genton, Ma, and Sang 2011), among others. However, the normal density decays quickly from the mean value, which is not suitable for modeling the tail of the distribution, hence more heavy-tailed elliptical models are needed. The multivariate Student-t (MVT) distribution belongs to this category and can be viewed as a scale mixture of the MVN distribution.

In this paper, we introduce an R (R Core Team 2021) package that is motivated by high-dimensional MVN probabilities but also tackles MVT probabilities. MVN and MVT probabilities are defined as:

$$\Phi_n(\mathbf{a}, \mathbf{b}; \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \int_{\mathbf{a}}^{\mathbf{b}} \frac{1}{\sqrt{(2\pi)^n |\boldsymbol{\Sigma}|}} \exp\left\{-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^\top \boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})\right\} \mathrm{d}\mathbf{x}, \tag{1}$$

$$T_n(\mathbf{a}, \mathbf{b}; \boldsymbol{\Delta}, \boldsymbol{\Sigma}, \nu) = \frac{2^{1-\frac{\nu}{2}}}{\Gamma(\frac{\nu}{2})} \int_0^\infty s^{\nu-1} e^{-s^2/2} \Phi_n\left\{\frac{s\mathbf{a}}{\sqrt{\nu}} - \boldsymbol{\Delta}, \frac{s\mathbf{b}}{\sqrt{\nu}} - \boldsymbol{\Delta}; \mathbf{0}, \boldsymbol{\Sigma}\right\} \mathrm{d}s, \tag{2}$$

respectively. Here, $\mathbf{a}$ and $\mathbf{b}$ are n-dimensional vectors denoting the lower and the upper integration limits, while $\boldsymbol{\mu}$ and $\boldsymbol{\Delta}$ are the mean and the location parameters. Notice that we use the "Kshirsagar" definition of MVT probabilities in Equation 2. MVT probabilities of the "shifted" type can be transformed into "Kshirsagar" MVT probabilities with $\boldsymbol{\Delta} = \mathbf{0}$. The type names of "Kshirsagar" and "shifted" are aligned with the naming convention of the **mvtnorm** package (Genz *et al.* 2021; Genz and Bretz 2009). The matrix $\boldsymbol{\Sigma} \in \mathbb{R}^{n \times n}$ is the positive-definite covariance matrix and ν is a positive scalar denoting the degrees of freedom.

Many methods have been proposed for estimating MVN probabilities among which Genz (1992) suggested the first numerical solution based on Monte Carlo simulation. Later developments focused on two aspects of the numerical methods, with Miwa, Hayter, and Kuriki (2003), Craig (2008), Nomura (2014), and Botev (2017), among others, focusing on the estimation accuracy, and Trinh and Genz (2015) and Genton, Keyes, and Turkiyyah (2018), among others, focusing on scalability; see Genz and Bretz (2009) for an overview of the various early approximation methods. Most accuracy-oriented methods are confined to $n < 100$ with the exception of Botev (2017), while the speed-oriented conditioning methods in Trinh and Genz (2015) and Cao, Genton, Keyes, and Turkiyyah (2019) usually have insufficient accuracy. We think that overall, the methods from Botev (2017) and Genton *et al.* (2018) have the widest applicability. Botev (2017) used the importance sampling technique to increase the convergence rate while Genton *et al.* (2018) improved the computation efficiency with a hierarchical representation of the covariance matrix. Cao *et al.* (2021) refined Genton *et al.* (2018) with the block reordering from Cao *et al.* (2019) and a tile-low-rank (TLR; Weisbecker 2013; Mary 2017; Akbudak, Ltaief, Mikhalev, and Keyes 2017; Cao *et al.* 2021) representation of the covariance matrix. In this paper, we introduce **tlrmvnmvt** (Cao, Genton, Keyes, and Turkiyyah 2022), an R package that implements the methods introduced in Cao *et al.* (2021) and that is available from the Comprehensive R Archive Network (CRAN) at `https://CRAN.R-project.org/package=tlrmvnmvt`, and compare the package with the two state-of-the-art alternatives, namely, the **mvtnorm** and the **TruncatedNormal** (Botev and Belzile 2019) packages, that implement the algorithms from Genz (1992) and from Botev (2017), respectively.

Because the TLR Monte Carlo method in Cao *et al.* (2021) is based on Genz (1992), the **tlrmvnmvt** package also includes an efficient implementation of the cumulative probability

functions from the package **mvtnorm**. Additionally, the algorithms modified for MVT probabilities are also implemented. Section 2 reviews the algorithms of the TLR Monte Carlo methods for MVN and MVT probabilities. Section 3 introduces the interfaces of the five functions included in this new package and provides examples of their usage. Section 4 compares the package performance with two state-of-the-art alternatives. Section 5 describes an application of computing the excursion sets defined in Bolin and Lindgren (2015), where the package **tlrmvnmvt** makes the computation feasible in thousands of dimensions without any model approximation. Section 6 concludes the paper.

2. TLR Monte Carlo with block reordering

The TLR Monte Carlo method with block reordering, as indicated by its name, adds two techniques to the original Monte Carlo method in Genz (1992), namely the TLR representation for the covariance matrix and the block reordering technique. The method in Genz (1992) is also known as the separation-of-variable (SOV) transformation that transforms the integration region into the unit hypercube. In this section, we first review the original Monte Carlo method and then we demonstrate how it is combined with the TLR representation and the block reordering. Without loss of generality, we assume $\boldsymbol{\mu} = \mathbf{0}$ and $\boldsymbol{\Delta} = \mathbf{0}$ for this section and denote MVN and MVT probabilities with $\Phi_n(\mathbf{a}, \mathbf{b}; \boldsymbol{\Sigma})$ and $T_n(\mathbf{a}, \mathbf{b}; \boldsymbol{\Sigma}, \nu)$, respectively, to simplify the expressions.

2.1. Monte Carlo for MVN and MVT

Genz (1992) used SOV to transform Equation 1 into

$$\Phi_n(\mathbf{a}, \mathbf{b}; \boldsymbol{\Sigma}) = \int_{\Phi(\tilde{a}_1)}^{\Phi(\tilde{b}_1)} \int_{\Phi(\tilde{a}_2)}^{\Phi(\tilde{b}_2)} \cdots \int_{\Phi(\tilde{a}_n)}^{\Phi(\tilde{b}_n)} \mathrm{d}\mathbf{z}, \quad \tilde{a}_i = \frac{a_i - \sum_{j=1}^{i-1} L_{ij} y_j}{L_{ii}}, \quad \tilde{b}_i = \frac{b_i - \sum_{j=1}^{i-1} L_{ij} y_j}{L_{ii}}. \tag{3}$$

The sequence of transformations from Equation 1 to Equation 3 are $\mathbf{x} = \mathbf{L}\mathbf{y}$ and $y_i = \Phi^{-1}(z_i)$, where $\mathbf{L}$ is the lower Cholesky factor of $\boldsymbol{\Sigma}$, $\boldsymbol{\Sigma} = \mathbf{L}\mathbf{L}^\top$; refer to Genz (1992) for further details. Here, we use the unbolded letter with subscripts to denote the corresponding coefficient in the matrix (vector). With a subsequent standardization, $z_i = \Phi(\tilde{a}_i) + \{\Phi(\tilde{b}_i) - \Phi(\tilde{a}_i)\}w_i$, the integration region can be transformed to the unit hypercube in $\mathbb{R}^n$, where quasi-Monte Carlo rules are applicable to $\mathbf{w}$. Here, the computation of the MVN integrand with $\mathbf{w}$ is also referred to as Monte Carlo sampling. In Equation 3, the integration limits for the ith integration variable depend on the first to the $(i-1)$th integration variables, which allows variable reordering to increase the Monte Carlo convergence rate but limits the degree of parallelism. The MVT probability of Equation 2 can be transformed into

$$T_n(\mathbf{a}, \mathbf{b}; \boldsymbol{\Sigma}, \nu) = \int_0^1 \Phi_n\left(\frac{\chi_\nu^{-1}(t)\mathbf{a}}{\sqrt{\nu}}, \frac{\chi_\nu^{-1}(t)\mathbf{b}}{\sqrt{\nu}}; \boldsymbol{\Sigma}\right) \mathrm{d}t, \tag{4}$$

through $s = \chi_\nu^{-1}(t)$, where $\chi_\nu^{-1}(\cdot)$ is the inverse of the Chi distribution with ν degrees of freedom. Equation 4 suggests that $(n+1)$-dimensional quasi-Monte Carlo rules can be used to generate $(t, \mathbf{w})$ as one Monte Carlo sample.

Detailed Monte Carlo algorithms for the two integrations of Equations 3 and 4 can be found in Genz (1992) and Genz and Bretz (2009), respectively. Note that a Cholesky factorization is

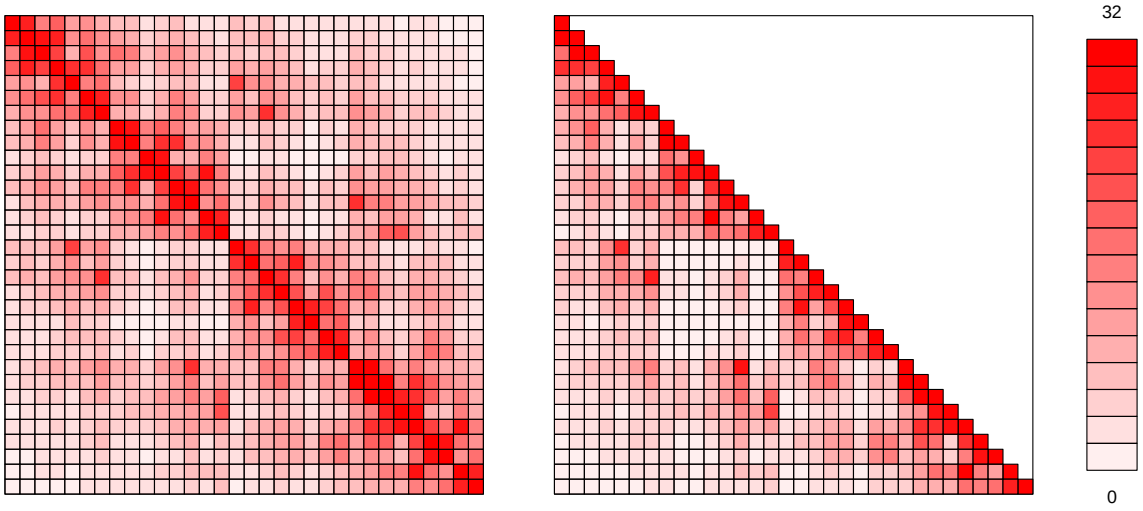
needed before estimating the integration with Monte Carlo samples, which has a complexity of $\mathcal{O}(n^3)$ and can be challenging by itself. It is recognized by Genz and Bretz (2009), Botev (2017), and Cao *et al.* (2021), among others, that ordering variables based on their conditional probabilities, measured by $\Phi(\tilde{b}_i) - \Phi(\tilde{a}_i)$, in an ascending order increases the convergence rate. The univariate reordering introduced in Trinh and Genz (2015) is one example. It has the same order of complexity as a Cholesky factorization and computes the Cholesky factor while ordering the integration variables, for which it is used to substitute the Cholesky factorization before the Monte Carlo procedures in the **mvtnorm** and **tlrmvnmvt** packages.

2.2. TLR representation for covariance matrices

As shown in Equation 3, the computation of $\tilde{a}_i$ and $\tilde{b}_i$ sums up to the matrix-vector multiplication between $\mathbf{L}$ and $\mathbf{y}$ and hence, the complexity for each Monte Carlo sample is $\mathcal{O}(n^2)$. This can readily be reduced through a structured representation of $\mathbf{L}$. The TLR representation breaks down a large matrix into small blocks, of which a large proportion can be closely approximated by the multiplication of two thin matrices, $\mathbf{U}\mathbf{V}^\top$. This representation is also referred to as the low-rank representation for the block (tile) (Weisbecker 2013; Mary 2017; Akbudak *et al.* 2017; Cao *et al.* 2021) and typically, all blocks have the same numbers of rows and columns to reduce the complexity of matrix operations. In this paper, we refer to the number of columns of $\mathbf{U}$ and $\mathbf{V}$ as local ranks, denoted by k, for which a smaller value yields greater computation savings. Since the matrix we approximate is a covariance matrix in $\mathbb{R}^{n\times n}$, we assume that the number of rows is equal to the number of columns for each block, both of which are denoted by m.

Figure 1 visualizes the local rank distribution across blocks in a Whittle correlation matrix and the corresponding Cholesky factor, where the pair-wise sum is simply two times the local ranks, also the sum of the ranks of $\mathbf{U}$ and $\mathbf{V}$. The Whittle kernel with a range parameter of 0.1 has an effective range of 0.4 on the unit square and is implemented with the `matern()` function from the **geoR** (Ribeiro Jr and Diggle 2020) package. For most blocks, the pair-wise local ranks are much smaller than 32 and this difference usually becomes more significant when n increases (Cao *et al.* 2021). With the TLR Cholesky factor, the complexity for each Monte Carlo sample is reduced to $\mathcal{O}(nm + kn^2/m)$, whose order is minimized approximately at $m = \sqrt{n}$ because the average k across the blocks is usually a small value, for example, below three.

Aligned with the TLR representation, a TLR Cholesky factorization can be designed at a complexity of $\mathcal{O}(n^3/m^2 + nm^2)$, which also reaches its minimum order, $\mathcal{O}(n^2)$, when $m = \sqrt{n}$; see Akbudak *et al.* (2017) for the algorithm of the TLR Cholesky factorization. However, the success of the TLR Cholesky factorization is not guaranteed because truncation is involved in the block addition part of the algorithm. Typically, the covariance matrix is close to singular when correlation is strong, in which case the Schur complement has a much smaller magnitude in its entries than the original covariance matrix, and small truncation errors may cause the failure of the Cholesky factorization. To address this problem, either a lower truncation error or a correction mechanism can be resorted to. The correction mechanism generally induces bias because it makes the matrix "more positive-definite". One common example is adding a nugget effect while Xia and Gu (2010) provided a more sophisticated example of the correction mechanism.



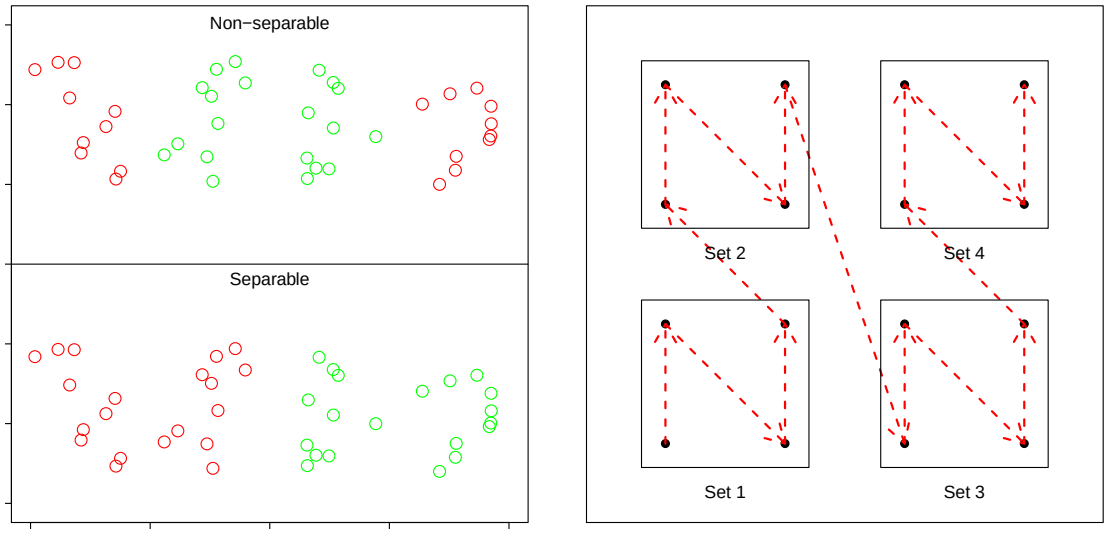
(a) (b)

Figure 1: Color-coded pair-wise local ranks of (a) the TLR covariance matrix and (b) the TLR Cholesky factor. Local ranks are truncated at the absolute error of 1e−3 and represented as the pair-wise sum. The covariance matrix is built with the Whittle correlation kernel with a range parameter of 0.1 for 1,024 spatial locations on a regular grid in the unit square. The order of the locations is Morton's order implemented by the `zorder()` function from the **tlrmvnmvt** package.

2.3. Block reordering

The block reordering is a variable reordering scheme compatible with the TLR representation of the covariance matrix. For a block to have a small local rank, the locations corresponding to the rows of the block should be spatially separated from those of the columns, which is also referred to as the admissibility condition (Börm, Grasedyck, and Hackbusch 2003b). Figure 2(a) shows two cases, where the two sets of spatial locations are regarded as non-separable and separable, respectively. For implementation, an explicit admissibility condition specifies the minimum ratio of the distance between the two sets to the minimum diameter of the sets (Börm *et al.* 2003b). A good ordering for the TLR representation aims to make all pairs of two different sets of locations separable while an ordering based on the integration limits, $\mathbf{a}$ and $\mathbf{b}$, increases the convergence rate of Monte Carlo sampling (Genz and Bretz 2009).

To benefit from both aspects, the block reordering, visualized in Figure 2(b), first divides the locations into sets based on their indices from Morton's order and then orders again the sets and the locations in each set based on their marginal probabilities; refer to Cao *et al.* (2019) for the detailed algorithm of the block reordering and Cao *et al.* (2021) for a recursive version of the block reordering. In this manner, the block reordering does not change the separability of any block in $\boldsymbol{\Sigma}$ while improving the convergence rate significantly. Cao *et al.* (2021) also discussed other potential structured representations for $\boldsymbol{\Sigma}$ and $\mathbf{L}$ and concluded that the TLR structure combined with the block reordering leads to the best efficiency.



(a) (b)

Figure 2: (a) Illustration of non-separable and separable pairs of sets. The separability is given by the bounding boxes defined in Börm *et al.* (2003a). (b) Illustration of the block reordering. The dotted red arrows indicate the sequence, in which the locations are ordered.

3. Package structure and implementation

The **tlrmvnmvt** package has two major functions for computing MVN and MVT probabilities, namely `pmvn()` and `pmvt()`. Both functions provide the flexibility of choosing the dense-matrix-based SOV algorithm or the TLR-matrix-based quasi-Monte Carlo algorithm. The third function, `zorder()`, generates Morton's order in 2D that is helpful for constructing covariance matrices with the low-rank feature. The last two functions, `GenzBretz()` and `TLRQMC()`, are used to specify the internal algorithm called by the `pmvn()` and the `pmvt()` functions. This package has an efficient underlying C++ implementation interfaced through **Rcpp** (Eddelbuettel and Balamuta 2018) and utilizes the **Eigen** library (Guennebaud and Jacob 2010), a template library for linear algebra, via **RcppEigen** (Bates and Eddelbuettel 2013). Currently, the **Boost** library (Schling 2011) is also linked through the **BH** package (Eddelbuettel, Emerson, and Kane 2021) to provide modified Bessel functions but this dependency can be spared when C++17 is supported by R.

3.1. Function interfaces

The interfaces of functions `pmvn()` and `pmvt()` are:

```
pmvn(lower = -Inf, upper = Inf, mean = 0, sigma = NULL,
  uselog2 = FALSE, algorithm = GenzBretz(), ...)

pmvt(lower = -Inf, upper = Inf, delta = 0, df = 1, sigma = NULL,
  uselog2 = FALSE, algorithm = GenzBretz(), type = "Kshirsagar", ...)
```

The two interfaces closely resemble those of the `pmvnorm()` and the `pmvt()` functions from the **mvtnorm** package. The first two arguments, `lower` and `upper`, define a rectangular in-

tegration region. The `mean` parameter for MVN probabilities and the `delta` parameter for MVT probabilities simply shift the integration limits, `lower` and `upper`, when the `type` of MVT probabilities is "shifted" whereas MVT probabilities of `type = "Kshirsagar"` are defined in Equation 2 and do not simply shift `lower` and `upper` with `delta`. The covariance matrix $\mathbf{\Sigma}$ can be either given by the `sigma` parameter or constructed by the package through specifying the underlying geometry `geom`, the covariance kernel `kernelType`, and the associated parameters `para`. Here, `geom`, `kernelType`, and `para` are optional parameters. The parameter `geom` should be a matrix with n rows, each of which is a coordinate vector and denoted with $\mathbf{s}_i$. Currently, the only supported covariance model is the Matérn covariance function that accepts `para` as a vector of length four, storing the scale parameter $\sigma > 0$, the range parameter $\beta > 0$, the smoothness parameter $\kappa > 0$, and the nugget parameter $\delta \geq 0$. Specifically, we assume the following parameterization for the Matérn covariance function:

$$\sigma_{ij} = \sigma^2 \{2^{\kappa-1}\Gamma(\kappa)\}^{-1} \left(\frac{\|\mathbf{s}_i - \mathbf{s}_j\|}{\beta}\right)^{\kappa} H_\kappa \left(\frac{\|\mathbf{s}_i - \mathbf{s}_j\|}{\beta}\right) + \mathbb{1}_{\{\mathbf{s}_i = \mathbf{s}_j\}} \cdot \delta, \tag{5}$$

where σ_{ij} is the (i, j)th coefficient of $\mathbf{\Sigma}$, $\|\cdot\|$ is the L_2-norm that leads to the Euclidean distance, $\mathbb{1}$ is the indicator function, and $H_\kappa(\cdot)$ denotes the modified Bessel function of the second kind of order κ. The parameter `uselog2` determines whether the probability is returned as its logarithm to the base two, which is necessary when the probability estimate is smaller than the machine precision. The `df` parameter should be a positive number representing the degrees of freedom that are needed only for MVT probabilities.

The `algorithm` parameter should be either of class '`GenzBretz`' or '`TLRQMC`', which are returned by the `GenzBretz()` and the `TLRQMC()` functions, respectively. When `algorithm` is of class '`GenzBretz`', the `pmvn()` and the `pmvt()` functions implement the univariate reordering and the SOV algorithm with a dense representation of the covariance matrix. This is recommended for MVN/MVT problems in relatively low dimensions, e.g., no more than 4,000 and is not constrained by the covariance structure. When `algorithm` is of the '`TLRQMC`' class, the two functions implement the recursive block reordering (Cao *et al.* 2021) and the TLR Monte Carlo algorithm. The latter combination can solve probabilities in tens of thousands of dimensions with reasonable time costs but requires the existence of the low-rank feature in the covariance matrix.

Both the `GenzBretz()` and the `TLRQMC()` functions accept parameters that control the aspects of the `pmvn()` and the `pmvt()` functions associated with the adopted algorithm:

```
GenzBretz(N = 499)
TLRQMC(N = 499, m = 64, epsl = 1e-4)
```

The `N` parameter controls how many Monte Carlo samples are used for estimating MVN/MVT probabilities. Specifically, we use the Richtmyer rule (Richtmyer 1951) for sampling $\mathbf{w}$ that is defined after Equation 3. The Richtmyer rule specifies a batch size N and the number of batches n_s, which amounts to a total sample size of $N \times n_s$. In our implementation, the `N` parameter is the batch size and n_s is set internally to 20. Two additional parameters are needed when the `TLRQMC` algorithm is used. The `m` parameter specifies the block (tile) size m and the `epsl` parameter is the absolute truncation error used in the construction of the TLR covariance matrix and the TLR Cholesky factorization. It is worth mentioning that the adaptive cross approximation (ACA; Bebendorf and Rjasanow 2003) algorithm is used

for constructing the TLR covariance matrix and that the low-rank matrix addition in the TLR Cholesky factorization is based on singular value decomposition (SVD); see Börm *et al.* (2003b) for a detailed discussion on low-rank matrix addition. Although $m = \sqrt{n}$ leads to the optimal order of complexity, the choice of m should guarantee that most off-diagonal blocks of dimension $m \times m$ in $\mathbf{\Sigma}$ can be closely approximated by the low-rank representation described in Section 2.2. This requirement motivates a choice of m depending on the structure of the input `sigma` or `geom` but generally, the smaller m is, the more closely the blocks are approximated by the low-rank representation.

When `algorithm` is of the class '`GenzBretz`', the univariate reordering, described in Algorithm 2.2 of Trinh and Genz (2015) is first applied, which reorders the integration variables and simultaneously computes the Cholesky factor $\mathbf{L}$ of $\mathbf{\Sigma}$. Next, the MVN and the MVT integrands are computed for each $\mathbf{w}$ generated from the Monte Carlo rule through the algorithms described in Section 3 of Genz (1992) and Section 4.2.2 of Genz and Bretz (2009), respectively. When `algorithm` is of the class '`TLRQMC`', the `pmvn()` and the `pmvt()` functions first construct the TLR representation of the covariance matrix based on either `sigma` or the optional parameters `geom`, `kernelType`, and `para`. Next, similar to the univariate reordering, the recursive block reordering is applied to improve the convergence rate, which also implicitly performs the TLR Cholesky factorization. The algorithms for computing the MVN and MVT integrands under the TLR representation are described in Cao *et al.* (2021), where it is also concluded that the computation costs per sample are proportional to the storage costs of the Cholesky factor. Therefore, the TLR representation significantly improves the time efficiency as large off-diagonal blocks are reduced to their low-rank representations.

For both, the `pmvn()` and the `pmvt()` functions, the probability estimate from one sample is the product of n one-dimensional probabilities and we store the mantissa and exponent of the product after each multiplication separately. This allows the storage of values much smaller than the machine precision and is also compatible with the computation of the mean and the standard deviation. Both functions return a numeric value of the estimated probability, either as its original value or as its logarithm to the base two if `uselog2` is `TRUE`. When `uselog2` is `FALSE`, the returned value has an attribute of the estimation error, which is not available otherwise because the standard deviation of the logarithm of the probability cannot be directly estimated. However, this is amenable to multiple instances of Monte Carlo sampling with the same arguments. Cao *et al.* (2021) showed that the logarithm of the probability has a much smaller relative error than the probability itself and hence, is preferred in high dimensions.

The last function to introduce from the **tlrmvnmvt** package is `zorder()` that implements Morton's ordering in the 2D plane:

```
zorder(geom)
```

`zorder()` accepts a set of locations from the `geom` parameter, which should be an n-by-2 matrix and returns the vector that matches the old indices to the new indices, e.g., `geom <- geom[zorder(geom)]`. This function aims to align the row indices of `geom` with their spatial locations, based on which the constructed covariance matrices are likely to possess the low-rank feature. This is useful when the covariance matrix cannot be constructed internally. For example, the covariance matrix is equal to the summation of two Matérn covariance matrices. In such cases, users can first construct a dense covariance matrix based on the order given by `zorder()`; then call the `pmvn()` and the `pmvt()` functions and set the `sigma` parameter equal to the previously computed covariance matrix. The time costs for constructing the dense

covariance matrix is typically much less than that of the Monte Carlo sampling, for which the `zorder()` function practically broadens the class of MVN/MVT probabilities that can be processed by the TLR methods.

3.2. Computation with dense matrices

In this section, we present the computation of MVN and MVT probabilities with the dense representation of the covariance matrix. The covariance matrix here is constructed from n locations on a perturbed grid in the unit square and a Whittle correlation function with the range parameter $\beta = 0.1$. The integration limits, $\mathbf{a}$ and $\mathbf{b}$, are generated from uniform distributions $\mathrm{U}(-5, -1)$ and $\mathrm{U}(1, 5)$, respectively. We set a fixed seed value for the results to be reproducable.

```
R> set.seed(123)
R> nx <- 20
R> ny <- 20
R> n <- nx * ny
R> vecx <- c(1:nx) - 1
R> vecy <- c(1:ny) - 1
R> geom <- cbind(kronecker(vecx, rep(1, ny)), kronecker(rep(1, nx), vecy))
R> geom <- geom + matrix(runif(n * 2), n, 2)
R> geom <- geom / max(nx, ny)
R> a <- runif(n, -5, -1)
R> b <- runif(n, 1, 5)
```

Since we will use the dense representation of the covariance matrix, we do not need to reorder the locations in `geom`. The covariance matrix is constructed with the `matern()` function from the **geoR** package.

```
R> library("geoR")
R> distM <- as.matrix(dist(geom))
R> covM <- matern(distM, 0.1, 1.0)
```

Note that the Whittle covariance function is a special case of the Matérn covariance function from Equation 5 when $\kappa = 1$. By default, the `uselog2` is `FALSE` and the estimation error is returned.

```
R> library("tlrmvnmvt")
R> sprintf("Time costs: %f seconds", system.time(
+    ret <- pmvn(a, b, 0, covM))[[3]])

[1] "Time costs: 1.203000 seconds"

R> ret

[1] 0.0001066559
attr(,"error")
[1] 3.328883e-06
```

The output error is the absolute error, similar to the **mvtnorm** package but different from the **TruncatedNormal** package. Similar results can be produced for MVT probabilities, for which we need to define the degrees of freedom.

```
R> nu <- 7
R> sprintf("Time costs: %f seconds", system.time(
+    ret <- pmvt(a, b, 0, nu, covM, uselog2 = TRUE))[[3]])

[1] "Time costs: 1.352000 seconds"

R> ret

[1] -6.987227
```

The log-probability is computed without an error estimation, which, if desired, can be estimated through multiple instances of the same problem. The above two examples can be also specified with the geometry and the covariance structure. Here, we only use the `pmvn()` function as an example.

```
R> sprintf("Time costs: %f seconds", system.time(
+    ret <- pmvn(a, b, 0, geom = geom, kernelType = "matern",
+      para = c(1, 0.1, 1, 0), algorithm = GenzBretz(N = 737)))[[3]])

[1] "Time costs: 1.723000 seconds"

R> ret

[1] 0.0001073855
attr(,"error")
[1] 3.155405e-06
```

In this example, we increase the sample size of Monte Carlo sampling from the default 499×20 to 737×20. The computed probability is very close to that from the first call of the `pmvn()` function and the difference is due to the randomness of Monte Carlo sampling.

3.3. Computation with TLR matrices

In this section, we first show that the computation of relatively low-dimensional MVN and MVT problems does not benefit much from the TLR methods, with the same examples used in Section 3.2. Next, we provide two higher-dimensional comparisons that highlight the computation efficiency of the TLR methods, where we construct the covariance matrix based on thousands of locations on a perturbed grid. The following code defines the block size to be 20 and uses the TLR method to compute the same MVN problem that appeared before.

```
R> sprintf("Time costs: %f seconds", system.time(
+    ret <- pmvn(a, b, 0, covM, algorithm = TLRQMC(m = 20)))[[3]])
```

```
[1] "Time costs: 1.723000 seconds"

R> ret

[1] 0.0001026763
attr(,"error")
[1] 4.489981e-06
```

For the problem dimension of 400, the `TLRQMC` algorithm actually has lower efficiency than the `GenzBretz` algorithm. This is because the block size `m` is 20, relative to which the local ranks cannot be much smaller. The storage and computation savings from the `TLRQMC` algorithm are significant only if the local ranks are much smaller than the block size. The same relationship is also true for MVT probabilities in 400 dimensions.

```
R> sprintf("Time costs: %f seconds", system.time(
+    ret <- pmvt(a, b, 0, nu, covM, uselog2 = TRUE,
+      algorithm = TLRQMC(m = 20)))[[3]])

[1] "Time costs: 1.740000 seconds"

R> ret

[1] -6.862828
```

For the same MVN/MVT probability, the result computed with the `TLRQMC` algorithm is very close to that computed with the `GenzBretz` algorithm in Section 3.2. From many similar numerical experiments, we conclude that the error caused by the TLR approximation is negligible compared with that from the Monte Carlo sampling. At this point, we clear the environment variables and build a covariance matrix based on 4,000 locations on a perturbed grid in $(0, 1)$. We do not choose a purely random geometry because the covariance matrix may appear singular under the truncation error, ϵ, if there are locations too close to each other. Here, we use the same correlation function, the Whittle correlation function with a range parameter of 0.1, and generate the integration limits also from $\mathrm{U}(-5, -1)$ and $\mathrm{U}(1, 5)$.

```
R> set.seed(123)
R> n <- 4000
R> geom <- c(0:(n - 1)) / n
R> geom <- geom + runif(n) / n * 0.8
R> distM <- as.matrix(dist(geom))
R> covM <- matern(distM, 0.1, 1.0)
R> a <- runif(n, -5, -1)
R> b <- runif(n, 1, 5)
```

In the 1D domain, the initial grid has a unit distance of $1/4000$, to which we add a random perturbation whose magnitude is uniformly distributed between zero and 0.8 times the unit distance. Due to the small unit distance and hence, the strong correlation between neighbor locations, we set the truncation error to 1e$-$6 when using the `TLRQMC` algorithm.

```
R> sprintf("Time costs: %f seconds", system.time(
+    ret <- pmvn(a, b, 0, covM,
+      algorithm = TLRQMC(m = 64, epsl = 1e-6)))[[3]])

[1] "Time costs: 10.730000 seconds"

R> ret

[1] 0.01164927
attr(,"error")
[1] 0.0002779126
```

The block size is chosen to be close to $\sqrt{n}$ for the optimal asymptotic complexity as discussed in Section 2.2. Typically, the covariance matrix from a 1D geometry and a smooth kernel has a strong low-rank feature, where the TLR representation achieves higher savings than what it achieves for problems defined in higher spatial dimensions. The computation for this 4,000-dimensional MVN problem takes less than 11 seconds, which is faster than the time of the next example where the problem size is the same but the spatial locations are in 2D. For the last problem, we consider a 4,000-dimensional MVN probability based on a 2D perturbed grid. The original grid is 50×80 with a unit distance of $1/80$, and the perturbation is also uniformly distributed between zero and 0.8 times the unit distance but in both x and y directions. The covariance matrix is designed to be the summation of a Whittle correlation matrix and an exponential correlation matrix, both with the range parameter $\beta = 0.1$. Note that to generate the low-rank feature, we index the locations based on Morton's order before constructing the two correlation matrices. Since the variance becomes two, we scale the range from which we generate the integration limits proportionally.

```
R> set.seed(123)
R> nx <- 50
R> ny <- 80
R> n <- nx * ny
R> vecx <- c(1:nx) - 1
R> vecy <- c(1:ny) - 1
R> geom <- cbind(kronecker(vecx, rep(1, ny)), kronecker(rep(1, nx), vecy))
R> geom <- geom + matrix(runif(n * 2), n, 2) * 0.8
R> geom <- geom / max(nx, ny)
R> idxZ <- zorder(geom)
R> geom <- geom[idxZ, ]
R> distM <- as.matrix(dist(geom))
R> covM1 <- matern(distM, 0.1, 1.0)
R> covM2 <- matern(distM, 0.1, 0.5)
R> covM <- covM1 + covM2
R> a <- runif(n, -10, -2)
R> b <- runif(n, 2, 10)
```

This type of covariance matrix cannot be generated by the **tlrmvnmvt** package internally, for which the `zorder()` function becomes necessary for constructing the low-rank feature. For this MVN problem, we use a truncation error of 1e−5.

```
R> sprintf("Time costs: %f seconds", system.time(
+    ret <- pmvn(a, b, 0, covM,
+      algorithm = TLRQMC(m = 64, epsl = 1e-5)))[[3]])

[1] "Time costs: 29.583000 seconds"

R> ret

[1] 1.754891e-05
attr(,"error")
[1] 2.471964e-06
```

The local ranks are not shown through the R interface but internally we find that the average local rank is one for the previous example in 1D space while it is five for this example in 2D space, which explains the increase of computation time. Specifically, the time costs of both the recursive block reordering and the Monte Carlo sampling are affected by the average local rank. It is worth noticing that here, the relative error is bigger than for previous examples because the "acceptance rate", defined in Botev (2017), is low. The **TruncatedNormal** package can produce smaller relative errors when the "acceptance rates" of the SOV and the TLR Monte Carlo algorithms are low but it also has significantly higher computation costs.

4. Performance comparison

We use five examples from Botev (2017) and Genton *et al.* (2018) to compare the performances of the three packages, i.e., **mvtnorm**, **TruncatedNormal**, and **tlrmvnmvt**. The first example features a slow-decaying probability, where the correlation is a constant, 0.5, and the integration limits are all $(-\infty, 0)$. This example was used as the high-dimensional example in Botev (2017). The performance of the three packages up to 16,384 dimensions is shown in Table 1, where the accuracy is measured with the relative error of the log-probability. Here, "N.A." indicates that either the package does not support the input problem dimension or the computation cost exceeds our capacity. The same reason for using "N.A." also applies to Tables 2 and 3. The true probability under the constant correlation structure is computed through a one-dimensional integration of

$$\Phi_n(-\boldsymbol{\infty}, \mathbf{b}; \boldsymbol{\Sigma}_\rho) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} \exp\left(-\frac{1}{2}t^2\right) \prod_{i=1}^{n} \Phi\left(\frac{b_i + \sqrt{\rho}t}{\sqrt{1-\rho}}\right) \mathrm{d}t, \tag{6}$$

which can be found in Genz (1992). Here, $\boldsymbol{\Sigma}_\rho$ denotes the constant correlation matrix whose correlation is denoted by ρ. This integration is computed by the Gauss quadrature (Golub and Welsch 1969) rule, specifically, the Gauss-Hermite rule (Liu and Pierce 1994) with 200 nodes. With the `gauss.quad()` function from the **statmod** package (Giner and Smyth 2016), the function for computing Equation 6 is defined as

```
R> library("statmod")
R> nnode <- 200
R> nodeWeight <- gauss.quad(nnode, "hermite")
```

n	16	64	128	512	1,024	2,048	4,096	16,384
mvtnorm	0.0%	0.1%	0.2%	0.5%	N.A.	N.A.	N.A.	N.A.
	0.0s	*0.2s*	*0.6s*	*4.1s*	N.A.	N.A.	N.A.	N.A.
TruncatedNormal	0.2%	0.1%	0.1%	0.1%	0.1%	0.1%	N.A.	N.A.
	0.1s	*0.3s*	*0.9s*	*9.9s*	*38.4s*	*164.1s*	N.A.	N.A.
tlrmvnmvt (`GenzBretz`)	0.1%	0.2%	0.4%	0.7%	1.2%	1.7%	2.8%	N.A.
	0.0s	*0.1s*	*0.2s*	*1.2s*	*3.3s*	*10.5s*	*58.3s*	N.A.
tlrmvnmvt (`TLRQMC`)	0.1%	0.2%	0.4%	0.7%	1.4%	1.9%	3.8%	5.7%
	0.0s	*0.1s*	*0.2s*	*1.1s*	*2.2s*	*6.0s*	*21.5s*	*133.8s*

Table 1: Performance analysis under the constant correlation structure. The lower integration limits are set to $-\infty$, the upper integration limits are set to 0.0 and the constant correlation to $\rho = 0.5$. The default sample sizes are used for all functions involved. In each cell, the upper row shows the relative error of the log-probability and the lower row shows the computation time. The results are the average over 10 replicates.

```
R> intfct <- function(x, b, rho) {
+    y <- rep(0, length(x))
+    for (i in 1:length(x)) {
+      y[i] <- 1 / sqrt(pi) * prod(pnorm((b + sqrt(2 * rho) * x[i]) /
+        sqrt(1 - rho)))
+    }
+    return(y)
+  }
R> constRhoProb <- function(b, rho) {
+    sum(nodeWeight$weights * intfct(nodeWeight$nodes, b, rho))
+  }
```

The `constRhoProb()` function is used to compute the true probabilities. The constant correlation structure is one of the ideal cases for low-rank representations that include the TLR structure, since all off-diagonal blocks have a numerical rank of one. All methods have relatively low errors up to 2,048 dimensions, with **TruncatedNormal** being the most accurate. The `TLRQMC` algorithm produces higher relative error than the `GenzBretz` algorithm because the recursive block reordering, used in the former, reorders only on the block level, which, heuristically, increases the convergence rate less than the univariate reordering used in the latter. The **tlrmvnmvt** package has the lowest time cost and the TLR method, specifically, finishes the estimation in 16,384 dimensions within one hundred seconds. All experiments in this paper are run on an Intel Xeon E5-2680 v4 @ 2.40GHz CPU; see the section on the computation details at the end of the paper for the versions of packages used.

We use the relative error of the log-probability because the relative error of the probability is no longer informative. As shown in Botev (2017), the relative error for tail probabilities already becomes large in moderately high dimensions, e.g., hundreds of dimensions, if the Monte Carlo methods without importance sampling are used. This may lead to the impression that probability estimates become meaningless in high dimensions but Cao *et al.* (2021) claim that the estimates of the logarithm of the probabilities are much more robust because the distribution of the probability estimates is skewed, and show that their relative errors are

n	16	64	128	512	1,024	2,048	4,096	16,384
mvtnorm	0.0%	0.3%	0.7%	2.8%	N.A.	N.A.	N.A.	N.A.
	0.0s	*0.2s*	*0.5s*	*4.1s*	N.A.	N.A.	N.A.	N.A.
TruncatedNormal	0.1%	0.1%	0.1%	0.1%	0.1%	0.1%	N.A.	N.A.
	0.1s	*0.3s*	*0.7s*	*8.5s*	*35.5s*	*148.1s*	N.A.	N.A.
tlrmvnmvt (`GenzBretz`)	0.1%	0.8%	1.0%	4.4%	6.8%	11.2%	17.8%	N.A.
	0.0s	*0.1s*	*0.2s*	*1.1s*	*2.9s*	*9.2s*	*41.1s*	N.A.
tlrmvnmvt (`TLRQMC`)	0.0%	0.5%	1.4%	4.4%	9.0%	15.4%	17.6%	108.4%
	0.0s	*0.1s*	*0.2s*	*0.9s*	*1.9s*	*4.4s*	*14.4s*	*105.3s*

Table 2: Performance analysis under the constant correlation structure. The lower integration limits are set to $-\infty$, the upper integration limits are set to -1.0 and the constant correlation to $\rho = 0.5$. The default sample sizes are used for all functions involved except for $n = 16{,}384$, where the sample size is 4e4. In each cell, the upper row shows the relative error of the log-probability and the lower row shows the computation time. The results are the average over 10 replicates.

usually small. We choose the relative error of the log-probability to show that the probability estimates are still close to the true values in terms of orders of magnitude and that the **tlrmvnmvt** package provides a viable option for high-dimensional applications with reasonably high accuracy requirements.

The second example considers tail probabilities, changing the integration limits from $(-\infty, 0)$ used in Table 1 to $(-\infty, -1)$, which becomes identical to the Example IV in Botev (2017). The relative errors of the log-probabilities and computation times are listed in Table 2. This problem design is considered more challenging than the first example because the integration regions are farther towards the tail. Compared with Table 1, the relative error increases faster with n, for which we use four times the default sample size in 16,384 dimensions. The methods from the **tlrmvnmvt** package become less reliable when the problem size exceeds two thousand. Based on these results, the accuracy of the **TruncatedNormal** package is the least affected by the shift of the integration region towards the tail.

The third example uses the same integration region as the second one but increases the constant correlation ρ to 0.8, which resembles Table 8 in Genton *et al.* (2018). The corresponding results for $\rho = 0.8$ are shown in Table 3. The relative errors become smaller than those in Tables 1 and 2. By increasing the correlation strength, the Monte Carlo sampling has a higher convergence rate, which is possibly due to the reduced number of effective integration variables. The first three examples consider only the constant correlation scenarios, where the true probability can be efficiently and accurately solved with Equation 6. The logarithm of such computed true probabilities is used as the benchmark for computing the relative errors. For these three examples, we do not experiment with the MVT functions because of the lack of their true probabilities.

The fourth example uses randomly generated covariance matrices as described in Davies and Higham (2000) and defines the integration limits to be $(-\infty, 0.5)$, which is similar to Example III in Botev (2017) but experiments are in 1,000 dimensions. The code snippet for generating random correlation matrices is based on the `rGivens()` function from the **fungible** package (Waller 2021).

n	16	64	128	512	1,024	2,048	4,096	16,384
mvtnorm	0.0%	0.1%	0.1%	0.1%	N.A.	N.A.	N.A.	N.A.
	0.0s	*0.2s*	*0.5s*	*4.1s*	N.A.	N.A.	N.A.	N.A.
TruncatedNormal	0.1%	0.1%	0.1%	0.1%	0.2%	0.2%	N.A.	N.A.
	0.1s	*0.3s*	*0.7s*	*8.5s*	*35.1s*	*147.6s*	N.A.	N.A.
tlrmvnmvt (`GenzBretz`)	0.0%	0.1%	0.1%	0.3%	0.4%	0.4%	0.4%	N.A.
	0.0s	*0.1s*	*0.2s*	*1.1s*	*2.9s*	*9.3s*	*40.6s*	N.A.
tlrmvnmvt (`TLRQMC`)	0.0%	0.1%	0.2%	0.3%	0.3%	0.4%	0.6%	1.2%
	0.0s	*0.1s*	*0.2s*	*0.9s*	*1.9s*	*4.4s*	*14.4s*	*105.4s*

Table 3: Performance analysis under the constant correlation structure. The lower integration limits are set to $-\infty$, the upper integration limits are set to -1.0 and the constant correlation to $\rho = 0.8$. The default sample sizes are used for all functions involved. In each cell, the upper row shows the relative error of the log-probability and the lower row shows the computation time. The results are the average over 10 replicates.

	Min	1st quartile	Median	3rd quartile	Max	Time
MVN probabilities						
mvtnorm	0.23%	0.28%	0.38%	0.45%	0.70%	12.3s
TruncatedNormal	0.03%	0.05%	0.07%	0.09%	0.12%	27.1s
tlrmvnmvt (`GenzBretz`)	0.23%	0.35%	0.40%	0.51%	0.56%	2.7s
MVT probabilities						
mvtnorm	4.82%	5.64%	6.24%	7.07%	9.35%	12.3s
TruncatedNormal	0.02%	0.03%	0.03%	0.04%	0.06%	41.7s
tlrmvnmvt (`GenzBretz`)	4.35%	6.93%	7.89%	8.90%	11.55%	2.7s

Table 4: Quartiles of the relative errors of the log-probabilities under random correlation matrices in 1,000 dimensions. The correlation matrix is randomly generated based on Davies and Higham (2000). The lower integration limits are set to $-\infty$ and the upper integration limits are set to 0.5. The degrees of freedom for MVT probabilities are $\nu = 7$. The default sample sizes are used for all functions involved. The statistics are computed from 20 simulated problems and the relative errors are based on 10 estimations of each problem.

```
R> library("fungible")
R> lambda <- runif(n)
R> lambda <- lambda * n / sum(lambda)
R> covM <- rGivens(lambda, Seed = i)$R
```

Here, `i` is the index of the current correlation matrix under simulation, ranging from 1 to 20. The five-number summaries and the computation times are shown in Table 4. We also include the MVT functions from the three packages, whose names are all `pmvt()`. The methods from the **mvtnorm** and **tlrmvnmvt** packages perform much better for MVN probabilities than for MVT probabilities. We reckon that the former has a higher convergence rate than the latter because the MVT algorithm (Genz and Bretz 1999) with univariate reordering is more affected by the negative correlation randomly generated. The relative errors of all three packages are

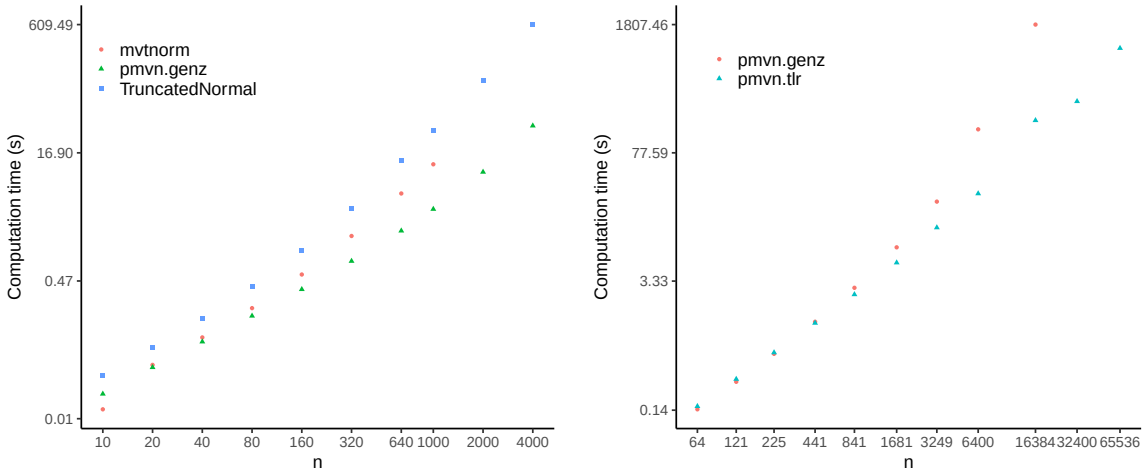
	Min	1st quartile	Median	3rd quartile	Max	Time
MVN probabilities						
mvtnorm	0.11%	0.18%	0.20%	0.23%	0.33%	11.8s
TruncatedNormal	0.13%	0.20%	0.26%	0.34%	0.43%	24.2s
tlrmvnmvt (`GenzBretz`)	0.13%	0.28%	0.36%	0.38%	0.50%	3.1s
MVT probabilities						
mvtnorm	0.34%	0.45%	0.50%	0.60%	0.86%	11.9s
TruncatedNormal	0.15%	0.19%	0.25%	0.29%	0.41%	34.1s
tlrmvnmvt (`GenzBretz`)	0.40%	0.64%	0.79%	0.93%	1.29%	3.1s

Table 5: Quartiles of the relative errors of the log-probabilities under Whittle correlation matrices in 900 dimensions. The correlation matrix is generated based on a 30×30 perturbed grid in the unit square. The Whittle correlation function has a range parameter $\beta = 0.1$. The lower integration limits are independently generated from $\mathrm{U}(-5, -1)$ and the upper integration limits are independently generated from $\mathrm{U}(1, 5)$. The degrees of freedom for MVT probabilities are $\nu = 7$. The default sample sizes are used for all functions involved. The statistics are computed from 20 simulated problems, each with a different geometry and integration limits, and the relative errors are based on 10 estimations of each problem.

again low for MVN probabilities, which renders the package **tlrmvnmvt** the preferred choice due to its shorter computation times. For applications involving MVT probabilities with irregular correlation, the **TruncatedNormal** package is a more reliable option.

The last example assumes a 2D Whittle correlation structure that is used in Section 3 of this paper and in Table 3 of Genton *et al.* (2018). The correlation matrices such generated are more representative than the previous four for spatial covariance matrices. Unlike Genton *et al.* (2018), we simulate integration limits from $\mathrm{U}(-5, -1)$ and $\mathrm{U}(1, 5)$, which is more challenging as the true probabilities become smaller. The performance of the three packages in 900 dimensions is summarized in Table 5. The difference in the convergence rate between MVN probabilities and MVT probabilities is less obvious than that in Table 4, which indicates that the Monte Carlo methods with variable reordering for MVT probabilities work more favorably under positive correlation. The relative errors are small for all methods while the computation time of the **tlrmvnmvt** package is less than a quarter of that of the **mvtnorm** package and approximately one twentieth of that of the **TruncatedNormal** package. We think that overall, the **tlrmvnmvt** package has a good tradeoff between accuracy and computation time for this category of MVN/MVT problems.

For Tables 4 and 5, the true probabilities are no longer available and hence, for each simulated problem, we repeat the computation for ten times, based on which the standard deviation of the log-probabilities are computed. This standard deviation is then used to compute the relative error of the log-probabilities. It is also worth mentioning that the TLR methods are not included in these two tables because the random correlation matrix does not have the low-rank feature while under the 2D Whittle correlation structure, the computation savings from the TLR representation is not significant in under 1,000 dimensions. Figure 3 shows the scalability of the `GenzBretz` and the `TLRQMC` algorithms, featuring the time growth with n when the Monte Carlo sample size is fixed at 1e4.



(a) (b)

Figure 3: Computation times of MVN probabilities in n dimensions under (a) the random correlation structure and (b) the 2D Whittle correlation structure. For (a), the problems are simulated in the same way as those in Table 4. For (b), the problems are simulated in the same way as those in Table 5 but in higher dimensions with an additional nugget effect of 0.05. The underlying geometry is a $\sqrt{n} \times \sqrt{n}$ perturbed grid in the unit square. Both x and y-axes are on the logarithm scale.

Figure 3(a) suggests that the `pmvn()` function may serve as a more efficient implementation of the `pmvnorm()` function from the **mvtnorm** package when the `GenzBretz` algorithm is used and that the cost of finding the proposal density (Botev 2017) used in the **TruncatedNormal** package grows quickly with n. Figure 3(b) compares the efficiencies of the `GenzBretz` and the `TLRQMC` algorithms under the 2D Whittle correlation structure. The difference in time is not significant when $n < 1{,}000$ but their ratio increases to 20 when $n = 16{,}384$. We conclude from the five examples above that the `TLRQMC` algorithms from the **tlrmvnmvt** package can efficiently estimate MVN/MVT probabilities in tens of thousands of dimensions but their accuracy may depend on the integration region and the correlation structure. For tail probabilities with weak correlation strength, the **TruncatedNormal** package is probably the best option. However, for most other cases, the functions in the **tlrmvnmvt** package can be the preferred choice, especially for high-dimensional applications.

5. Application in finding excursion sets

For applications that need to compute MVN probabilities, their models are often simplified to be computationally feasible in high dimensions. Bolin and Lindgren (2015) provided such an example that computed the excursion sets of a latent Gaussian random field. In their second example, they created a latent Gaussian random field in 2D and approximated it with a Gaussian Markov random field (GMRF). Although this approximation made the computation feasible in 6,400 dimensions, the accuracy was shown to be one-order-of-magnitude lower than their first example, where the exact posterior distribution was used. In this section, we show that with the **tlrmvnmvt** package, we can compute the excursion sets of the 2D latent Gaussian

random field with its exact posterior distribution and that the accuracy of our excursion sets reaches the same level as the first example in Bolin and Lindgren (2015).

Using the definition in Bolin and Lindgren (2015), a positive level u excursion set with probability $1-\alpha$ is defined by:

$$E^+_{u,\alpha}(X) = \text{argmax}_D\{|D| : P\{D \subseteq A^+_u(X)\} \geq 1-\alpha\},$$

where $A^+_u(f) = \{\mathbf{s} \in \mathbf{\Omega}; f(\mathbf{s}) > u\}$ and f is a deterministic function in $\mathbf{\Omega}$. The negative excursion sets are symmetrically defined and not elaborated upon here. In the implementation, $E^+_{u,\alpha}(X)$ is computed as the largest D that satisfies $P\{D \subseteq A^+_u(X)\} \geq 1-\alpha$ in a family of expanding sets:

$$D^+_1(\rho) = \{\mathbf{s}; P\{X(\mathbf{s}) > u\} \geq 1-\rho\},$$

which is referred to as the one-parameter family in Bolin and Lindgren (2015). Based on $E^+_{u,\alpha}(X)$, the positive excursion function is defined as:

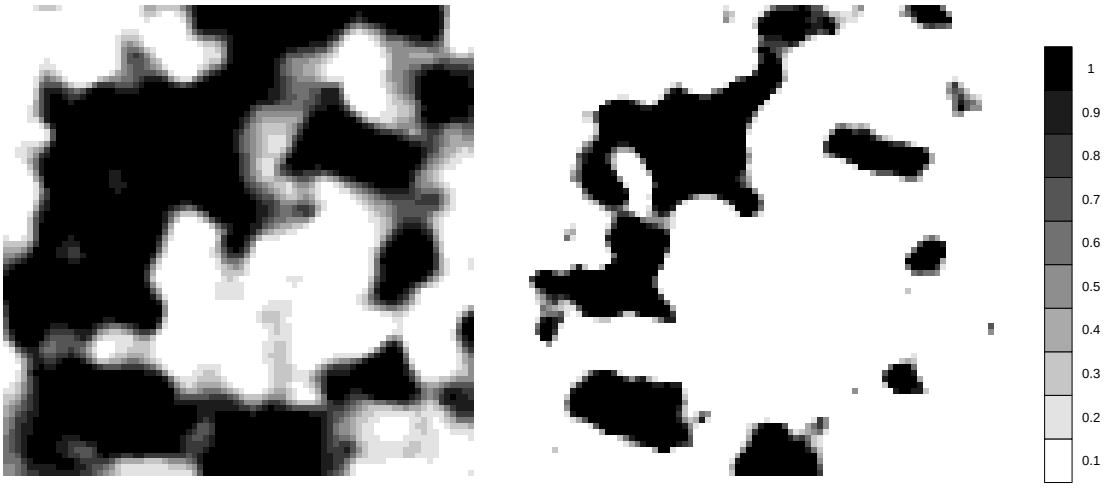
$$F^+_u(\mathbf{s}) = \sup\{1-\alpha; \mathbf{s} \in E^+_{u,\alpha}(X)\},$$

which can be significantly different from the posterior marginal exceedance probability. To compute $E^+_{u,\alpha}(X)$ and $F^+_u(\mathbf{s})$, we need to estimate $P\{D \subseteq A^+_u(X)\}$ for all D in the chosen one-parameter family, whose size can range from hundreds to thousands. Therefore, the efficient estimation of $P\{D \subseteq A^+_u(X)\}$ is crucial to the computation of the excursion sets.

The structure of the latent Gaussian random field is the same as that in Bolin and Lindgren (2015). Specifically, $X(\mathbf{s})$ is a Gaussian random field with a zero mean structure and the Matérn covariance structure, described in Equation 5, with $\sigma = 1$, $\beta = \sqrt{2}/20$, $\kappa = 1$, and $\delta = 0$. Our goal is to predict one realization of this random field on a 80×80 grid in the unit square based on one thousand observations on this grid under the additive Gaussian noise $N(0, 0.5^2)$. Using $\mathbf{y}$ to denote the observations, $X(\mathbf{s}) \mid \mathbf{y}$ is the 2D latent Gaussian random field whose excursion sets are of interest. The posterior distribution of $X(\mathbf{s}) \mid \mathbf{y}$ is another MVN distribution with

$$\begin{aligned}
\mathbf{Q}_{\text{post}} &= \mathbf{Q} + \frac{1}{\epsilon^2}\mathbf{A}^\top\mathbf{A}, \\
\boldsymbol{\mu}_{\text{post}} &= \boldsymbol{\mu} + \frac{1}{\epsilon^2}\mathbf{Q}^{-1}_{\text{post}}\mathbf{A}^\top(\mathbf{y} - \mathbf{A}\boldsymbol{\mu}),
\end{aligned}$$

where $\boldsymbol{\mu} = \mathbf{0}$ and $\mathbf{Q}$ are the mean and the precision matrix of $\mathbf{x}$, $\boldsymbol{\mu}_{\text{post}}$ and $\mathbf{Q}_{\text{post}}$ are the posterior mean and precision matrix of $\mathbf{x} \mid \mathbf{y}$, $\mathbf{A}$ is the matrix such that the coefficients in $\mathbf{y} - \mathbf{A}\mathbf{x}$ have i.i.d. $N(0, \epsilon^2)$ distributions and $\epsilon = 0.5$ is the magnitude of the additive Gaussian noise. Here, we use $\mathbf{x}$ for the realization of $X(\mathbf{s})$ on the grid. When the level u is zero, the posterior marginal probability, $P\{X(\mathbf{s}) > 0 \mid \mathbf{y}\}$, and the positive excursion function, $F^+_0(\mathbf{s})$, are shown in Figure 4. The posterior marginal probability is greater than or equal to the positive excursion function and in fact, $E^+_{u,0.99}(X \mid \mathbf{y}) \subseteq D^+_1(0.05)$, which indicates a strong mismatch between $P\{X(\mathbf{s}) > 0 \mid \mathbf{y}\}$ and $F^+_0(\mathbf{s})$. Similar to Bolin and Lindgren (2015), we simulate 50,000 realizations from $X(\mathbf{s}) \mid \mathbf{y}$ and compute the empirical probability of $\min(\{X(\mathbf{s}); \mathbf{s} \in E^+_{0,\alpha}(X)\}) > 0$, denoted by $\hat{p}(\alpha)$. Figure 5 describes the difference between $\hat{p}(\alpha)$ and $1-\alpha$ as a function of $1-\alpha$. The differences are smaller than those in Bolin and



(a) (b)

Figure 4: (a) The posterior marginal probability for $P\{X(\mathbf{s}) > 0 \mid \mathbf{y}\}$ and (b) the positive excursion function $F_0^+(\mathbf{s})$.

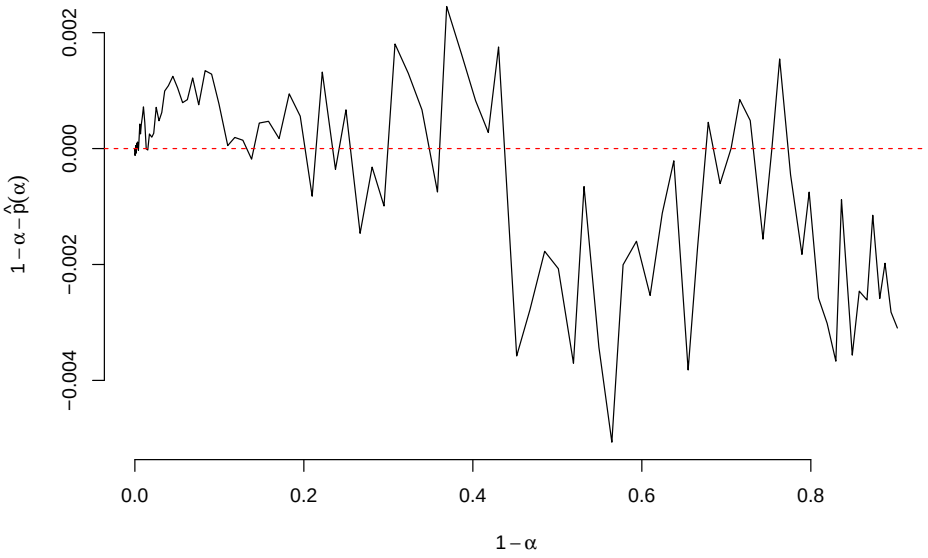


Figure 5: The difference between the empirical probability of $\min(\{X(\mathbf{s}); \mathbf{s} \in E_{0,\alpha}^+(X)\}) > 0$, denoted by $\hat{p}(\alpha)$, and $1 - \alpha$.

Lindgren (2015) by one-order of magnitude, which highlights the extra accuracy gained from using the exact posterior distribution over using the GMRF approximation.

It is worth mentioning that the `TLRQMC` algorithm is used for computing $E_{0,\alpha}^+(X)$ and $F_0^+(\mathbf{s})$ because the low-rank feature is strong if we order the locations on the 80×80 grid with the `zorder()` function. The run time is reduced by a factor between two and three compared with that using the `GenzBretz` algorithm. For higher-dimensional applications, this computation saving is expected to be more significant. The following code snippet is used to generate the observations `Y`.

```
R> n.obs <- 1000
R> sigma.e <- 0.5
R> m <- 80; n <- m * m
R> y <- x <- seq(from = 0, to = 1, length.out = m)
R> geom <- cbind(kronecker(x, rep(1, m)), kronecker(rep(1, m), y))
R> odrMorton <- zorder(geom)
R> geom <- geom[odrMorton, ]
R> distM <- as.matrix(dist(geom))
R> covM <- matern(distM, phi = sqrt(2)/20, kappa = 1)
R> cholM <- t(chol(covM))
R> mu <- rep(0, n)
R> set.seed(120)
R> y0 <- as.vector(cholM %*% rnorm(n)) + mu
R> obsIdx <- sample(1:n, n.obs)
R> Y <- y0[obsIdx] + rnorm(n.obs) * sigma.e
```

The next code snippet constructs the parameters for the posterior distribution of $X(\mathbf{s})$, which is a MVN distribution with mean `mu.post` and covariance matrix `covM`.

```
R> A <- matrix(0, n.obs, n)
R> A[cbind(seq_len(n.obs), obsIdx)] <- 1
R> Q <- chol2inv(chol(covM))
R> Q.post <- Q
R> Q.post <- Q.post + crossprod(A) / (sigma.e^2)
R> mu.post <- as.vector(mu + solve(Q.post,
+    (t(A) %*% (Y - mu[obsIdx]))/(sigma.e^2)))
R> covM <- chol2inv(chol(Q.post))
```

Knowing the posterior distribution, the marginal probability at each location on the grid can be readily achieved to build the one-parameter family, $D_1^+(\rho)$, based on which the excursion function is computed.

```
R> pMar <- 1 - pnorm(0, mean = mu.post, sd = sqrt(diag(covM)))
R> FMar <- rep(0, n)
R> ttlNum <- sum(pMar > 0.95)
R> pMarOdr <- order(pMar, decreasing = TRUE)
R> numVec <- round(seq(from = 3180, to = 1000, by = -10))
R> for (num in numVec) {
+    selectIdx <- pMarOdr[1 : num]
+    tmpLower <- rep(-Inf, n)
+    tmpLower[selectIdx] <- 0
+    FMar[selectIdx] <- pmvn.tlr(lower = tmpLower, mean = mu.post,
+      sigma = covM, m = 80)[[1]]
+  }
```

`FMar` stores the excursion function $F_0^+(\mathbf{s})$ and $E_{0,\alpha}^+(X)$ can be simultaneously constructed as $\{\mathbf{s}; F_0^+(\mathbf{s}) \geq 1 - \alpha\}$.

6. Summary and discussion

We introduced the R package **tlrmvnmvt** that computes MVN/MVT probabilities with the TLR Monte Carlo methods described in Cao *et al.* (2021). This package also provides a more efficient implementation of the SOV methods introduced in Genz (1992). Through comparison with the two state-of-the-art packages, i.e., the **mvtnorm** and the **TruncatedNormal** packages, we conclude that our package has the highest time efficiency and is probably the only viable option in thousands of dimensions. The TLR Monte Carlo methods improve the computation efficiency compared with the dense SOV methods but require the low-rank feature. Fortunately, the covariance matrices constructed from a spatial domain under common covariance functions typically possess this block-level low-rank feature.

The relative error of the probability estimates may increase quickly with the number of dimensions, depending on the covariance structure and the integration limits. However, we showed that the relative error of the log-probability is usually much smaller than that of the probability estimates. Even for cases where the relative error of the probability estimates exceeds 50%, the relative error of the log-probability can still be small, e.g., below 1%. The **tlrmvnmvt** and the **mvtnorm** packages work less favorably compared with the **TruncatedNormal** package when solving tail probabilities or probabilities with negative correlation structures. Although the **TruncatedNormal** package produces the most accurate results among the three, it has the highest time costs and is typically not applicable in high dimensions.

We used the computation of the excursion sets of a latent Gaussian random field (Bolin and Lindgren 2015) as an example to which the **tlrmvnmvt** package can be applied. Bolin and Lindgren (2015) approximated the posterior MVN distribution with a Gaussian Markov random field to make the computation of MVN probabilities feasible in thousands of dimensions. With the **tlrmvnmvt** package, we utilized the low-rank feature of the posterior covariance matrix and computed the excursion sets using the exact posterior distribution. We showed that the accuracy of the excursion sets was improved by one order of magnitude compared with Bolin and Lindgren (2015).

Computational details

The results in this paper were obtained using R 3.6.3 with the **fields** 10.3 package, the **fungible** 1.95.2 package, the **geoR** 1.8.1 package, the **ggplot2** 3.2.1 package, the **mvtnorm** 1.0.11 package, the **scales** 1.1.0 package, the **statmod** 1.4.33 package, the **tlrmvnmvt** 1.1.1 package, and the **TruncatedNormal** 2.1 package. R itself and all packages used are available from CRAN at `https://CRAN.R-project.org/`.

References

Akbudak K, Ltaief H, Mikhalev A, Keyes D (2017). "Tile Low Rank Cholesky Factorization for Climate/Weather Modeling Applications on Manycore Architectures." In *International Supercomputing Conference*, pp. 22–40. Springer-Verlag.

Bates D, Eddelbuettel D (2013). "Fast and Elegant Numerical Linear Algebra Using the **RcppEigen** Package." *Journal of Statistical Software*, **52**(5), 1–24. `doi:10.18637/jss.v052.i05`.

Bebendorf M, Rjasanow S (2003). "Adaptive Low-Rank Approximation of Collocation Matrices." *Computing*, **70**(1), 1–24. doi:10.1007/s00607-002-1469-6.

Bolin D, Lindgren F (2015). "Excursion and Contour Uncertainty Regions for Latent Gaussian Models." *Journal of the Royal Statistical Society B*, **77**(1), 85–106. doi:10.1111/rssb.12055.

Börm S, Grasedyck L, Hackbusch W (2003a). "Hierarchical Matrices." *Lecture note 21*, Max-Planck-Institut für Mathematik in den Naturwissenschaften Leipzig. URL https://www.mis.mpg.de/de/publications/andere-reihen/ln/lecturenote-2103.html.

Börm S, Grasedyck L, Hackbusch W (2003b). "Introduction to Hierarchical Matrices with Applications." *Engineering Analysis with Boundary Elements*, **27**(5), 405–422. doi:10.1016/s0955-7997(02)00152-2.

Botev ZI (2017). "The Normal Law Under Linear Restrictions: Simulation and Estimation via Minimax Tilting." *Journal of the Royal Statistical Society B*, **79**(1), 125–148. doi:10.1111/rssb.12162.

Botev ZI, Belzile L (2019). ***TruncatedNormal**: Truncated Multivariate Normal and Student Distributions*. R package version 2.1, URL https://CRAN.R-project.org/package=TruncatedNormal.

Cao J, Genton MG, Keyes DE, Turkiyyah GM (2019). "Hierarchical-Block Conditioning Approximations for High-Dimensional Multivariate Normal Probabilities." *Statistics and Computing*, **29**(3), 585–598. doi:10.1007/s11222-018-9825-3.

Cao J, Genton MG, Keyes DE, Turkiyyah GM (2021). "Exploiting Low Rank Covariance Structures for Computing High-Dimensional Normal and Student-t Probabilities." *Statistics and Computing*, **31**(1), 1–16. doi:10.1007/s11222-020-09978-y.

Cao J, Genton MG, Keyes DE, Turkiyyah GM (2022). ***tlrmvnmvt**: Low-Rank Methods for MVN and MVT Probabilities*. R package version 1.1.1, URL https://CRAN.R-project.org/package=tlrmvnmvt.

Craig P (2008). "A New Reconstruction of Multivariate Normal Orthant Probabilities." *Journal of the Royal Statistical Society B*, **70**(1), 227–243. doi:10.1111/j.1467-9868.2007.00625.x.

Davies PI, Higham NJ (2000). "Numerically Stable Generation of Correlation Matrices and Their Factors." *BIT Numerical Mathematics*, **40**(4), 640–651. doi:10.1023/a:1022384216930.

Davison AC, Huser R, Thibaud E (2013). "Geostatistics of Dependent and Asymptotically Independent Extremes." *Mathematical Geosciences*, **45**(5), 511–529. doi:10.1007/s11004-013-9469-y.

Durante D (2019). "Conjugate Bayes for Probit Regression via Unified Skew-Normal Distributions." *Biometrika*, **106**(4), 765–779. doi:10.1093/biomet/asz034.

Eddelbuettel D, Balamuta JJ (2018). "Extending R with C++: A Brief Introduction to **Rcpp**." *The American Statistician*, **72**(1), 28–36.

Eddelbuettel D, Emerson JW, Kane MJ (2021). ***BH**: **Boost** C++ Header Files*. R package version 1.78.0-0, URL https://CRAN.R-project.org/package=BH.

Genton MG, Keyes DE, Turkiyyah G (2018). "Hierarchical Decompositions for the Computation of High-Dimensional Multivariate Normal Probabilities." *Journal of Computational and Graphical Statistics*, **27**(2), 268–277. doi:10.1080/10618600.2017.1375936.

Genton MG, Ma Y, Sang H (2011). "On the Likelihood Function of Gaussian Max-Stable Processes." *Biometrika*, **98**(2), 481–488. doi:10.1093/biomet/asr020.

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

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

Giner G, Smyth GK (2016). "**statmod**: Probability Calculations for the Inverse Gaussian Distribution." *The R Journal*, **8**(1), 339–351. doi:10.32614/rj-2016-024.

Golub GH, Welsch JH (1969). "Calculation of Gauss Quadrature Rules." *Mathematics of Computation*, **23**(106), 221–230. doi:10.1090/s0025-5718-69-99647-1.

Guennebaud G, Jacob B (2010). "**Eigen** V3." URL https://eigen.tuxfamily.org/.

Liu Q, Pierce DA (1994). "A Note on Gauss-Hermite Quadrature." *Biometrika*, **81**(3), 624–629. doi:10.1093/biomet/81.3.624.

Mary T (2017). *Block Low-Rank Multifrontal Solvers: Complexity, Performance, and Scalability*. Ph.D. thesis, King Abdullah University of Science and Technology.

Miwa T, Hayter AJ, Kuriki S (2003). "The Evaluation of General Non-Centred Orthant Probabilities." *Journal of the Royal Statistical Society B*, **65**(1), 223–234. doi:10.1111/1467-9868.00382.

Nomura N (2014). "Computation of Multivariate Normal Probabilities with Polar Coordinate Systems." *Journal of Statistical Computation and Simulation*, **84**(3), 491–512. doi:10.1080/00949655.2012.717224.

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/.

Ribeiro Jr PJ, Diggle PJ (2020). ***geoR**: Analysis of Geostatistical Data*. R package version 1.8-1, URL https://CRAN.R-project.org/package=geoR.

Richtmyer RD (1951). "The Evaluation of Definite Integrals, and a Quasi-Monte-Carlo Method Based on the Properties of Algebraic Numbers." *Technical report*, Los Alamos Scientific Lab.

Schling B (2011). *The **Boost** C++ Libraries*. XML Press.

Trinh G, Genz A (2015). "Bivariate Conditioning Approximations for Multivariate Normal Probabilities." *Statistics and Computing*, **25**(5), 989–996. doi:10.1007/s11222-014-9468-y.

Waller NG (2021). ***fungible**: Psychometric Functions from the Waller Lab*. R package version 1.99, URL https://CRAN.R-project.org/package=fungible.

Weisbecker C (2013). *Improving Multifrontal Solvers by Means of Algebraic Block Low-Rank Representations*. Ph.D. thesis, King Abdullah University of Science and Technology.

Xia J, Gu M (2010). "Robust Approximate Cholesky Factorization of Rank-Structured Symmetric Positive Definite Matrices." *SIAM Journal on Matrix Analysis and Applications*, **31**(5), 2899–2920. doi:10.1137/090750500.

Affiliation:

Jian Cao, Marc G. Genton, David E. Keyes
CEMSE Division, Extreme Computing Research Center
King Abdullah University of Science and Technology
Thuwal 23955-6900, Saudi Arabia
E-mail: jian.cao@kaust.edu.sa, marc.genton@kaust.edu.sa,
david.keyes@kaust.edu.sa

George M. Turkiyyah
Department of Computer Science
American University of Beirut
Beirut, Lebanon
E-mail: gt02@aub.edu.lb

Journal of Statistical Software https://www.jstatsoft.org/
published by the Foundation for Open Access Statistics https://www.foastat.org/
January 2022, Volume 101, Issue 4 *Submitted:* 2020-03-11
doi:10.18637/jss.v101.i04 *Accepted:* 2021-02-16