Hierarchical-Block Conditioning Approximations for High-Dimensional Multivariate Normal Probabilities

Jian Cao, Marc G. Genton, David E. Keyes and George M. Turkiyyah[1]

January 9, 2018

Abstract

This paper presents a new method to estimate large-scale multivariate normal probabilities. The approach combines a hierarchical representation with processing of the covariance matrix that decomposes the n-dimensional problem into a sequence of smaller m-dimensional ones. It also includes a d-dimensional conditioning method that further decomposes the m-dimensional problems into smaller d-dimensional problems. The resulting two-level hierarchical-block conditioning method requires Monte Carlo simulations to be performed only in d dimensions, with $d \ll n$, and allows the complexity of the algorithm's major cost to be $O(n \log n)$. The run-time cost of the method depends on two parameters, m and d, where m represents the diagonal block size and controls the sizes of the blocks of the covariance matrix that are replaced by low-rank approximations, and d allows a trade-off of accuracy for expensive computations in the evaluation of the probabilities of m-dimensional blocks. We also introduce an inexpensive block reordering strategy to provide improved accuracy in the overall probability computation. Numerical simulations on problems from 2D spatial statistics with dimensions up to 16,384 indicate that the algorithm achieves a 1% error level and improves the run time over a one-level hierarchical Quasi-Monte Carlo method by a factor between 5 and 10.

Keywords: Block reordering; d-dimensional conditioning; Hierarchical representation; Spatial covariance functions; Univariate reordering.

Short title: Hierarchical-block Conditioning for high-dimensional MVN probabilities

[1] CEMSE Division, Extreme Computing Research Center, King Abdullah University of Science and Technology, Thuwal 23955-6900, Saudi Arabia.
E-mail: {jian.cao, marc.genton, david.keyes, george.turkiyyah}@kaust.edu.sa
This research was supported by King Abdullah University of Science and Technology (KAUST).

1 Introduction

The computation of high-dimensional multivariate normal (MVN) probabilities is required for a variety of applications. For example, in spatial extreme statistics, the commonly used parameterizations of the max-stable process involve the multivariate normal distribution (Smith et al., 1990; Schlather, 2002; Brown and Resnick, 1977). Moreover, the multivariate t-distribution probability can be transformed into a multivariate normal probability (Genz and Bretz, 2009) and the multivariate skew-normal distribution (Genton, 2004; Azzalini and Capitanio, 2014) is defined based on the multivariate normal distribution, both of which have important applications in science and engineering (Gupta and Brown, 2001; Kotz and Nadarajah, 2004).

The multivariate normal probability is defined as:

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

where $\mathbf{a}$ and $\mathbf{b}$ are integration bounds, $\mathbf{\Sigma}$ is a positive-definite matrix produced by a covariance function, and n is the problem size. Because most covariance functions in spatial statistics are smooth at points excluding 0, the blocks of the covariance matrix that represent the correlation between two groups of well-separated variables are usually well approximated by low-rank blocks. This approximation provides the potential for using the hierarchical matrix ($\mathcal{H}$ matrix) representation (Hackbusch, 2015) for the covariance matrix, $\mathbf{\Sigma}$, which substantially reduces the complexity of the matrix-associated operations.

MVN probabilities are generally not analytically tractable and various approximation methods have been studied to generate reliable estimations. Genz (1992) used separation of variables to transform the integration region to the unit hypercube in which a Monte Carlo simulation is performed. Niederreiter (1992), Caflisch (1998) and Owen and Zhou (2000) studied more efficient sampling methods for Monte Carlo simulation. Genz and Bretz (2009) provided a summary of recently developed methods for estimating MVN probabilities. Two state-of-the-art methods, among others, are the bivariate conditioning method (Trinh and Genz, 2015) and the hierar-

chical Quasi-Monte Carlo method (Genton et al., 2018). The bivariate conditioning method provides insights into estimating high-dimensional MVN problems with low-dimensional Monte-Carlo simulations but the univariate reordering has a complexity of $O(n^3)$ which makes working in high dimensions difficult. The hierarchical Quasi-Monte Carlo method utilizes the hierarchical representation of the covariance matrix and reduces the complexity of one Monte Carlo sample to $O(mn + kn\log(n/m))$ from $O(n^2)$, where m and k represent the diagonal block size and the rank of the off-diagonal blocks, respectively.

The purpose of this paper is to generalize the bivariate conditioning method to a d-dimensional conditioning method and combine it with the hierarchical representation of the covariance matrix. Through this combination, we introduce a two-level hierarchical-block conditioning method. In addition, we develop a reordering scheme with a complexity of $O(m^2n)$ that can maintain the low-rank feature of the off-diagonal blocks, thus making it applicable to high-dimensional MVN problems. The downside of conditioning methods, including this one, is that error estimates of the computed MVN probability approximations are not generated as part of the computations. As a result, apriori assessments are needed to insure that the method is reliable in the contexts where it will be used (Trinh and Genz, 2015).

The remainder of the paper is organized as follows. In Section 2, we introduce the construction of the hierarchical covariance matrix. In Section 3, we propose a generalized d-dimensional conditioning algorithm and present the derivation of the truncated normal expectations and the numerical simulation results for different d. In Section 4, we combine the conditioning method with the hierarchical representation of the covariance matrix to develop a two-level hierarchical-block conditioning method that speeds up the MVN integration problem relative to the one-level hierarchical Quasi-Monte Carlo method. In Section 5, we introduce a block-wise reordering scheme that significantly improves the accuracy level with negligible run-time cost. We discuss our experimental results in Section 6.

2 Building Hierarchical Representations

2.1 Generating a hierarchical covariance matrix

Hierarchical covariance matrices are shown in Figures 1(a) and 1(b). The number of diagonal blocks, denoted by r, expands with a factor of 2 while the off-diagonal part has a total number of $r-1$ low-rank blocks. In actual construction and storage, the $\mathcal{H}$-matrix assumes a quad-tree structure in which each parent node has two child nodes as well as two low-rank matrices. The green blocks are low-rank representations and the blue blocks signify dense matrices. Hierarchical representation enables faster matrix arithmetic and decreases the storage costs. These improvements are especially significant for high-dimensional problems. Specifically, when two groups are well separated in space, the corresponding off-diagonal block usually has few singular values that are larger than the machine precision. However, the low-rank feature depends on the separation of samples. When the samples are not very distant, the ranks of the off-diagonal blocks grow. Even so, the hierarchical representation still delivers significant computational savings relative to the standard dense representation.

Building a hierarchical covariance matrix requires the input of a covariance model, the spatial geometry of the samples and the indices of the samples. For general covariance models, the low-rank representation of off-diagonal blocks is generated from the dense representation through singular value decomposition (Hackbusch, 2015). This is computationally expensive but achieves the goal of examining the efficiencies of different algorithms for solving the MVN problem in high dimensions. We can use the orthonormal polynomial expansions of common covariance functions as an approximation to generate the low-rank representation directly. Both methods approximate the off-diagonal blocks individually and thus require validation for positive-definiteness. In the case where a non-positive-definite approximation is produced, we can decrease the error tolerance level, $\|\mathbf{L}-\mathbf{U}\mathbf{V}^T\|/\|\mathbf{L}\|$, where $\mathbf{L}$ is some matrix block, $\mathbf{U}\mathbf{V}^T$ is the low-rank approximation of $\mathbf{L}$, and $\|\cdot\|$ denotes a valid norm for matrices, or we can increase the number of terms in the orthonormal polynomial expansions.

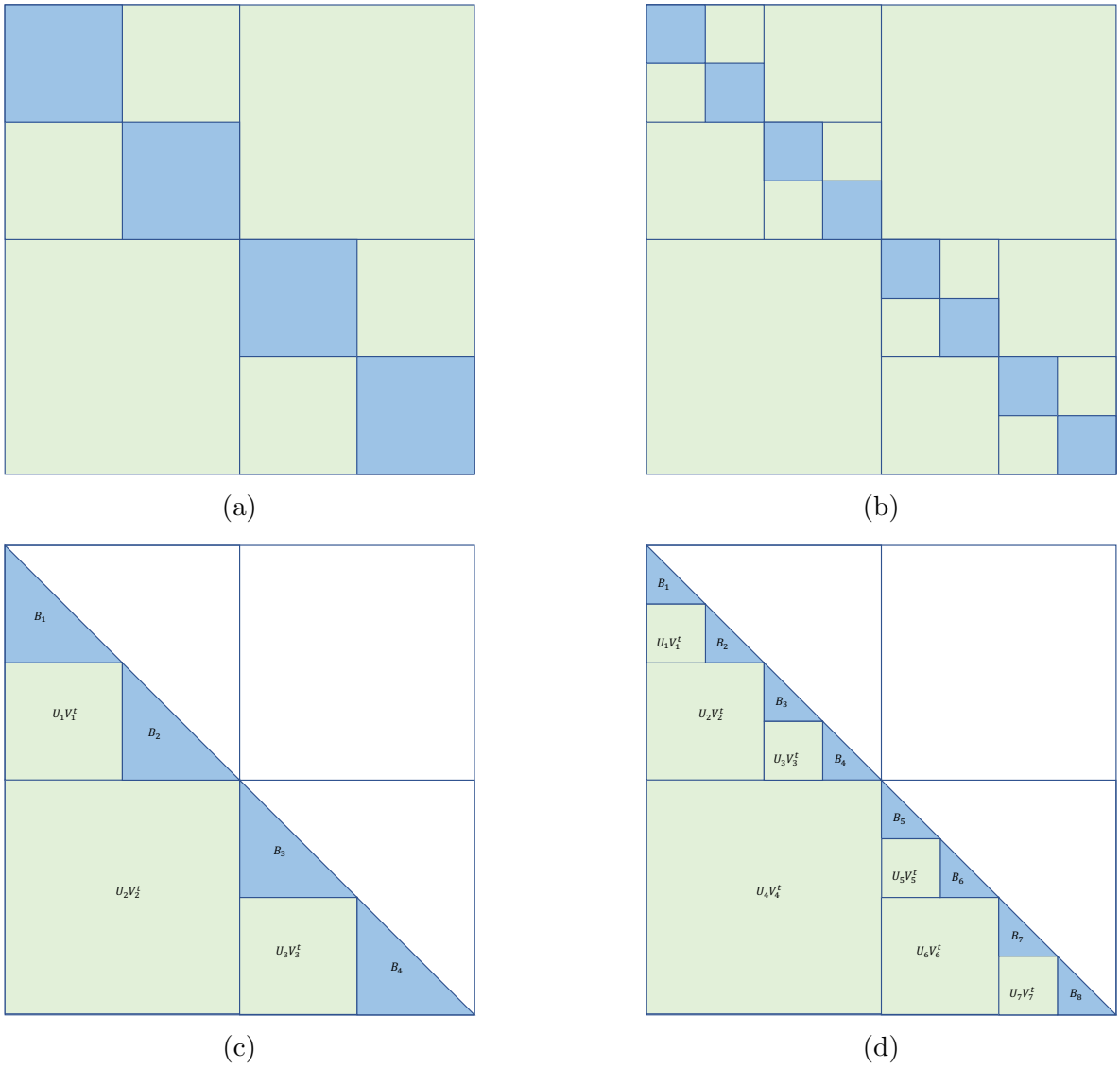

Figure 1: Hierarchical covariance matrices and Cholesky factors. The green blocks are approximated by low-rank matrices while the blue blocks are stored in the dense form. (a) and (b) are hierarchical matrices with maximum tree depth equal to 2 and 3. (c) and (d) are hierarchical Cholesky factors with maximum tree depth equal to 2 and 3.

2.2 Hierarchical Cholesky factorization

Both the conditioning method and the Quasi-Monte Carlo method require the Cholesky factor of the covariance matrix. The dense Cholesky factorization has a complexity of $O(n^3)$ and requires $O(n^2)$ amount of memory, which becomes prohibitive on typical workstations when n is much larger than 10^4. Hierarchical Cholesky factorization is much faster even under a strong spatial correlation structure. Table 1 compares two implementations of dense Cholesky factorization

Table 1: Time required for Cholesky factorization (seconds). The covariance matrix is generated from the exponential covariance function with $\beta = 0.3$, based on n points uniformly distributed in the unit square indexed with Morton's order. *chol* refers to the Cholesky factorization implemented in R. *dpotrf* is the Cholesky factorization from the LAPACK library. *choldecomp_hmatrix* is the hierarchical Cholesky factorization from the H2Lib library.

n	256	1,024	4,096	16,384
chol	0.004	0.23	12.66	737.0
dpotrf	0.001	0.02	0.61	41.5
choldecomp_hmatrix	0.003	0.03	0.46	9.3

and one of hierarchical Cholesky factorization under different matrix dimensions, n. We selected a 2D exponential covariance structure, $\text{corr}(\mathbf{x}, \mathbf{y}) = \exp(-\|\mathbf{x} - \mathbf{y}\|/\beta)$, where β is set to 0.3, to compare the time required for these three Cholesky factorization methods. The underlying geometry consists of n points evenly distributed on a grid in the unit square and indexed with Morton's order (Morton, 1966). Three implementations, namely the *chol* from R (R Core Team, 2016), the *dpotrf* from LAPACK (Anderson et al., 1999), and the *choldecomp_hmatrix* from H2Lib (Hackbusch, 2015), are compared as run on an Intel Xeon(R) E5-2680 CPU.

Since we focus on high-dimensional problems, we can reasonably conclude that hierarchical Cholesky factorization is the most efficient among the three although *dpotrf* is highly optimized. The commonly used function, *chol* from R, has a much slower performance than the other two. The cost of hierarchical Cholesky factorization depends on the ranks of the off-diagonal blocks, which usually grow at a much slower rate than $O(n)$ under 2D geometries. The algorithm may involve truncation of the off-diagonal blocks to maintain the low-rank feature. This would introduce little error compared with the overall estimation method discussed in the paper. The resulting hierarchical Cholesky factor has the hierarchical structure shown in Figures 1(c) and 1(d).

3 d-dimensional Conditioning Approximation

With knowledge of the hierarchical Cholesky factor, $\mathbf{L}$, the n-dimensional MVN problem can be separated into r m-dimensional MVN problems through the change of variable $\mathbf{Y} = \mathbf{L}\mathbf{X}$. For each m-dimensional problem, the Cholesky factor of its covariance matrix is already given from the diagonal blocks of $\mathbf{L}$. In this section, we extend the bivariate conditioning method of Trinh and Genz (2015) to a d-dimensional conditioning method and use it to compute the m-dimensional MVN probabilities and truncated expectations that become the building blocks for solving the n-dimensional MVN problem. To avoid confusion, we will add a tilde to the notations in the n dimension to represent the counterpart in the m dimension. For example, $\tilde{\mathbf{X}}$ represents the normal random variables in the m dimension.

3.1 d-dimensional LDL decomposition

LDL decomposition is a generalization of the classical Cholesky factorization. In fact, we can produce the Cholesky factors from LDL decomposition and vice versa without tracing back to the original covariance matrix. In calculating MVN probabilities, the variable transformation using the Cholesky factor can make the new integration variables independent from each other. Similarly, transformation with the $\tilde{\mathbf{L}}$ matrix in LDL decomposition can generate integration variables that are block-wise independent. The bivariate conditioning method from Trinh and Genz (2015) applied the 2-dimensional LDL decomposition and separated the m integration variables into uncorrelated blocks of size 2. We use the LDL decomposition in dimension d to separate the m integration variables into uncorrelated blocks of size d. When m is a multiple of d, the matrices $\tilde{\mathbf{L}}$ and $\tilde{\mathbf{D}}$ can be written as

$$\tilde{\mathbf{L}} = \begin{bmatrix} \mathbf{I}_d & \mathbf{O}_d & \cdots & \mathbf{O}_d \\ \tilde{\mathbf{L}}_{2,1} & \ddots & \ddots & \vdots \\ \vdots & \ddots & \mathbf{I}_d & \mathbf{O}_d \\ \tilde{\mathbf{L}}_{s,1} & \cdots & \tilde{\mathbf{L}}_{s,s-1} & \mathbf{I}_d \end{bmatrix}, \tilde{\mathbf{D}} = \begin{bmatrix} \tilde{\mathbf{D}}_1 & \mathbf{O}_d & \cdots & \mathbf{O}_d \\ \mathbf{O}_d & \ddots & \ddots & \vdots \\ \vdots & \ddots & \tilde{\mathbf{D}}_{s-1} & \mathbf{O}_d \\ \mathbf{O}_d & \cdots & \mathbf{O}_d & \tilde{\mathbf{D}}_s \end{bmatrix},$$

Algorithm 1 LDL decomposition

procedure LDL($\tilde{\mathbf{\Sigma}}$)
  $\tilde{\mathbf{L}} \leftarrow \mathbf{I}_m$, $\tilde{\mathbf{D}} \leftarrow \mathbf{O}_m$
  for $i = 1 : d : m - d + 1$ **do**
    $\tilde{\mathbf{D}}[i : i + d - 1, i : i + d - 1] \leftarrow \tilde{\mathbf{\Sigma}}[i : i + d - 1, i : i + d - 1]$
    $\tilde{\mathbf{L}}[i + d : m, i : i + d - 1] \leftarrow \tilde{\mathbf{\Sigma}}[i + d : m, i : i + d - 1]\tilde{\mathbf{D}}^{-1}[i : i + d - 1, i : i + d - 1]$
    $\tilde{\mathbf{\Sigma}}[i + d : m, i + d : m] \leftarrow \tilde{\mathbf{\Sigma}}[i + d : m, i + d : m] - \tilde{\mathbf{L}}[i + d : m, i : i + d - 1]\tilde{\mathbf{D}}[i : i + d - 1, i : i + d - 1]\tilde{\mathbf{L}}^T[i + d : m, i : i + d - 1]$
    if $i + d < m$ **then**
      $\tilde{\mathbf{D}}[i + d : m, i + d : m] \leftarrow \tilde{\mathbf{\Sigma}}[i + d : m, i + d : m]$
    end if
  end for
  return $\tilde{\mathbf{L}}$ and $\tilde{\mathbf{D}}$
end procedure

where $s = \frac{m}{d}$, $\mathbf{O}_d$ and $\mathbf{I}_d$ are zero and unit matrices of dimension d, $\tilde{\mathbf{D}}_i, i = 1, \ldots, s$, are positive-definite matrices, and $\tilde{\mathbf{L}}_{ij}, i = 2, \ldots, s, j = 1, \ldots, s - 1$, are matrix blocks. If m is not a multiple of d, then the last row of $\tilde{\mathbf{L}}$ and $\tilde{\mathbf{D}}$ has the dimension of the remainder. The algorithm for LDL decomposition is outlined as Algorithm 1. With the change of variable $\tilde{\mathbf{Y}} = \tilde{\mathbf{L}}\tilde{\mathbf{X}}$, we can rewrite an m-dimensional MVN probability as the product of s d-dimensional MVN probabilities

$$\Phi_m(\tilde{\mathbf{x}}; \tilde{\mathbf{a}}, \tilde{\mathbf{b}}, \tilde{\mathbf{\Sigma}}) = \int_{\tilde{\mathbf{a}}_1'}^{\tilde{\mathbf{b}}_1'} \phi_d(\tilde{\mathbf{y}}_1; \tilde{\mathbf{D}}_1) \int_{\tilde{\mathbf{a}}_2'}^{\tilde{\mathbf{b}}_2'} \phi_d(\tilde{\mathbf{y}}_2; \tilde{\mathbf{D}}_2) \cdots \int_{\tilde{\mathbf{a}}_s'}^{\tilde{\mathbf{b}}_s'} \phi_d(\tilde{\mathbf{y}}_s; \tilde{\mathbf{D}}_s) \, \mathrm{d}\tilde{\mathbf{y}}_s \cdots \mathrm{d}\tilde{\mathbf{y}}_2 \, \mathrm{d}\tilde{\mathbf{y}}_1, \qquad (2)$$

where $\tilde{\mathbf{a}}_i' = \tilde{\mathbf{a}}_i - \sum_{j=1}^{i-1} \tilde{\mathbf{L}}_{ij}\tilde{\mathbf{y}}_j$, $\tilde{\mathbf{b}}_i' = \tilde{\mathbf{b}}_i - \sum_{j=1}^{i-1} \tilde{\mathbf{L}}_{ij}\tilde{\mathbf{y}}_j$, $\tilde{\mathbf{a}}_i$ and $\tilde{\mathbf{b}}_i$, $i = 1, \ldots, s$, are the corresponding segments of $\tilde{\mathbf{a}}$ and $\tilde{\mathbf{b}}$.

3.2 d-dimensional truncated expectations

The integration limits in Equation (2) depend only on the integration variables to their left. The conditioning method separates the m-dimensional integration into d-dimensional integrations and updates the integration limits with the truncated expectations of the integration variables on the left side. When $d = 2$, Trinh and Genz (2015) employed the method in Muthen (1990) to calculate the bivariate truncated expectations. In this paper, we generalize the truncated normal expectation formula to d-dimensions based on the work of Kan and Robotti (2017). The

truncated expectation is expressed as

$$\mathrm{E}(\hat{\mathbf{X}}^{\mathbf{e}_j}) = \frac{1}{\Phi_d(\hat{\mathbf{a}}, \hat{\mathbf{b}}; \hat{\boldsymbol{\mu}}, \hat{\boldsymbol{\Sigma}})} \int_{\hat{\mathbf{a}}}^{\hat{\mathbf{b}}} \hat{x}_j \phi_d(\hat{\mathbf{x}}; \hat{\boldsymbol{\mu}}, \hat{\boldsymbol{\Sigma}}) \, \mathrm{d}\hat{\mathbf{x}}, \tag{3}$$

where $\hat{\mathbf{X}}$ is a d-dimensional MVN random vector, $\phi_d(\cdot)$ denotes the d-dimensional normal probability density, $\mathbf{e}_j$ is a unit vector of length d with 1 in the jth position and 0 elsewhere, and $\hat{\mathbf{X}}^{\mathbf{e}_j}$ denotes the jth random variable in $\hat{\mathbf{X}}$. Here, $\hat{\mathbf{a}}$ and $\hat{\mathbf{b}}$ are the corresponding lower and upper integration bounds of dimension d. In this section, we add a hat to the notations to signify the counterparts in d-dimensions. We define

$$F_j^d(\hat{\mathbf{a}}, \hat{\mathbf{b}}; \hat{\boldsymbol{\mu}}, \hat{\boldsymbol{\Sigma}}) = \int_{\hat{\mathbf{a}}}^{\hat{\mathbf{b}}} \hat{x}_j \phi_d(\hat{\mathbf{x}}; \hat{\boldsymbol{\mu}}, \hat{\boldsymbol{\Sigma}}) \, \mathrm{d}\hat{\mathbf{x}},$$

which is the numerator of Equation (3). A recurrence relation for F_j^d can be derived by differentiating the MVN density function, $\phi_d(\hat{\mathbf{x}}; \hat{\boldsymbol{\mu}}, \hat{\boldsymbol{\Sigma}})$, with respect to $\hat{\mathbf{x}}$, then multiplying $\hat{x}_j$ on both sides and integrating the random vector, $\hat{\mathbf{x}}$, from $\hat{\mathbf{a}}$ to $\hat{\mathbf{b}}$. The detailed derivation of the recurrence relation can be found in Kan and Robotti (2017). We use parentheses around the subscript to denote the exclusion of the element from the subscript set. Following this relation, we deduce that

$$F_j^d(\hat{\mathbf{a}}, \hat{\mathbf{b}}; \hat{\boldsymbol{\mu}}, \hat{\boldsymbol{\Sigma}}) = \hat{\mu}_j \Phi_d(\hat{\mathbf{a}}, \hat{\mathbf{b}}; \hat{\boldsymbol{\mu}}, \hat{\boldsymbol{\Sigma}}) + \mathbf{e}_j^T \hat{\boldsymbol{\Sigma}} \mathbf{c}, \tag{4}$$

where $\mathbf{c}$ is a vector with the lth coefficient defined as

$$\begin{aligned}
c_l &= \phi_1(\hat{a}_l; \hat{\mu}_l, \hat{\sigma}_l^2) \Phi_{d-1}(\hat{\mathbf{a}}_{(l)}, \hat{\mathbf{b}}_{(l)}; \check{\boldsymbol{\mu}}_l^1, \check{\boldsymbol{\Sigma}}_l) - \phi_1(\hat{b}_l; \hat{\mu}_l, \hat{\sigma}_l^2) \Phi_{d-1}(\hat{\mathbf{a}}_{(l)}, \hat{\mathbf{b}}_{(l)}; \check{\boldsymbol{\mu}}_l^2, \check{\boldsymbol{\Sigma}}_l), \\
\check{\boldsymbol{\mu}}_l^1 &= \hat{\boldsymbol{\mu}}_{(l)} + \hat{\boldsymbol{\Sigma}}_{(l),l} \frac{\hat{a}_l - \hat{\mu}_l}{\hat{\sigma}_l^2}, \quad \check{\boldsymbol{\mu}}_l^2 = \hat{\boldsymbol{\mu}}_{(l)} + \hat{\boldsymbol{\Sigma}}_{(l),l} \frac{\hat{b}_l - \hat{\mu}_l}{\hat{\sigma}_l^2}, \quad \check{\boldsymbol{\Sigma}}_l = \hat{\boldsymbol{\Sigma}}_{(l),(l)} - \frac{1}{\hat{\sigma}_l^2} \hat{\boldsymbol{\Sigma}}_{(l),l} \hat{\boldsymbol{\Sigma}}_{l,(l)}.
\end{aligned}$$

When $d = 2$, $\hat{\boldsymbol{\mu}} = \mathbf{0}$ and $\hat{\sigma}_l = 1, l = 1, \ldots, d$, the above formula turns into the bivariate version used by Trinh and Genz (2015). The computation of the d-dimensional truncated expectation requires the computation of the $(d-1)$-dimensional MVN probability for $2d$ times which becomes the major computational cost for the conditioning algorithm and for the two-level hierarchical-

block conditioning algorithm introduced later. Future developments for efficiently estimating the truncated normal expectations would substantially improve the performance of this algorithm.

3.3 The d-dimensional conditioning algorithm

In the m-dimensional MVN probability problem represented by Equation (2), the d-dimensional conditioning method iterates through s integrations from the left to the right while updating the integration limits after each iteration. The d-dimensional conditioning algorithm is presented as Algorithm 2: A minor change is needed to the algorithm when the remainder of m divided by d is not 0. In the algorithm, $\Phi_d(\mathbf{Y}'; \tilde{\mathbf{a}}', \tilde{\mathbf{b}}', \tilde{\mathbf{D}}')$ is approximated with Quasi-Monte Carlo simulation and $\mathrm{E}[\tilde{\mathbf{Y}}']$ is computed from Equations (3) and (4). Since $(d-1)$-dimensional MVN probabilities are generated through Quasi-Monte Carlo simulations, the result unavoidably introduces a small error that can be made negligible compared with the error from the conditioning method.

To illustrate the improved accuracy and the effect on computation time when d becomes larger, we simulate 250 MVN problems for different values of m and d. The covariance matrix, $\tilde{\mathbf{\Sigma}}$, is simulated through $\tilde{\mathbf{\Sigma}} = \tilde{\mathbf{Q}}\check{\mathbf{J}}\tilde{\mathbf{Q}}^T$, where $\tilde{\mathbf{Q}}$ is simulated from the Haar distribution over the orthogonal matrix group (Stewart, 1980) and $\tilde{\mathbf{J}}$ is a diagonal matrix with the diagonal coefficients independently drawn from $\mathrm{U}(0,1)$. The integration limits are set with $\tilde{\mathbf{a}} = \mathbf{0}$ and $\tilde{\mathbf{b}} \overset{i.i.d}{\sim}$

Algorithm 2 d-dimensional conditioning algorithm

procedure CMVN($\tilde{\mathbf{\Sigma}}, \tilde{\mathbf{a}}, \tilde{\mathbf{b}}, d$)
  $\tilde{\mathbf{y}} \leftarrow \mathbf{0}$ and $P \leftarrow 1$
  $[\tilde{\mathbf{L}}, \tilde{\mathbf{D}}] = \mathrm{LDL}(\tilde{\mathbf{\Sigma}})$ as in Algorithm 1
  for $i = 1 : s$ **do**
    $j \leftarrow (i-1)d$
    $\tilde{\mathbf{g}} \leftarrow \tilde{\mathbf{L}}[j+1 : j+d, 1 : j]\tilde{\mathbf{y}}[1 : j]$
    $\tilde{\mathbf{a}}' \leftarrow \tilde{\mathbf{a}}[j+1 : j+d] - \tilde{\mathbf{g}}$
    $\tilde{\mathbf{b}}' \leftarrow \tilde{\mathbf{b}}[j+1 : j+d] - \tilde{\mathbf{g}}$
    $\tilde{\mathbf{D}}' \leftarrow \tilde{\mathbf{D}}[j+1 : j+d, j+1 : j+d]$
    $P \leftarrow P \cdot \Phi_d(\tilde{\mathbf{Y}}'; \tilde{\mathbf{a}}', \tilde{\mathbf{b}}', \tilde{\mathbf{D}}')$
    $\tilde{\mathbf{y}}[j+1 : j+d] \leftarrow \mathrm{E}[\tilde{\mathbf{Y}}']$
  end for
  return P and $\tilde{\mathbf{y}}$
end procedure

Table 2: Errors and execution times of the d-dimensional conditioning method. The upper half of the table reports results of the non-reordered conditioning method and the lower half reports the results of the reordered conditioning method. The correlation matrix is randomly generated from the correlation matrix space. The absolute error and the time cost are shown for each combination of m and d based on 250 replicates.

Without univariate reordering					
(m, d)	1	2	4	8	16
16	2.6%	3.6%	2.9%	1.5%	0.0%
	0.00s	*0.00s*	*0.04s*	*0.28s*	*2.32s*
32	2.8%	4.7%	4.3%	3.5%	2.2%
	0.00s	*0.01s*	*0.07s*	*0.55s*	*4.69s*
64	2.6%	4.0%	3.7%	3.4%	2.7%
	0.00s	*0.02s*	*0.13s*	*1.08s*	*9.23s*
128	2.8%	4.3%	4.2%	3.9%	3.3%
	0.00s	*0.04s*	*0.24s*	*1.88s*	*16.17s*
With univariate reordering					
16	0.1%	0.1%	0.0%	0.0%	0.0%
	0.001s	*0.005s*	*0.042s*	*0.285s*	*2.318s*
32	0.0%	0.0%	0.0%	0.0%	0.0%
	0.003s	*0.011s*	*0.076s*	*0.560s*	*4.682s*
64	0.0%	0.0%	0.0%	0.0%	0.0%
	0.008s	*0.025s*	*0.144s*	*1.092s*	*9.222s*
128	0.0%	0.0%	0.0%	0.0%	0.0%
	0.029s	*0.069s*	*0.270s*	*1.907s*	*16.15s*

$\mathrm{U}(0, m)$ which correspond to the conditions used in Trinh and Genz (2015) for straightforward comparison. Our simulation results are presented in the upper half of Table 2, where the relative errors are benchmarked against the Quasi-Monte Carlo results. The estimation error is reduced when d increases and when m remains unchanged because less correlation information is discarded by conditioning when d becomes larger. In the extreme case where $m = d$, we perform the Quasi-Monte Carlo simulation in m-dimensions to approximate the MVN probability directly and no error is caused by the loss of correlation information. The time cost grows close to a linear fashion with m. However, the curse of dimensionality is significant when d increases since the simulation is performed in dimension d.

3.4 Reordered d-dimensional conditioning

Trinh and Genz (2015) found that reordering of the integration variables can improve the estimation accuracy of the conditioning method. The reordering precedes the running of Algorithm 2 and arranges the integration variables with smaller truncated probabilities on the left side. Since the integration limits are affected only by the integration variables to their left as shown in Equation (2), intuitively the integration variables further to the left have more impact on the estimation accuracy. Univariate reordering is preferred to multivariate reordering because the marginal improvement for the multivariate reordering is insignificant but the increased computational complexity is substantial. The univariate reordering algorithm is presented as Algorithm 2.2 in Trinh and Genz (2015). Here, we extract and rephrase the univariate reordering algorithm for completeness. The d-dimensional conditioning method with univariate reordering is presented as Algorithm 3 here.

The lower half of Table 2 presents the results for the d-dimensional conditioning algorithm with preceding reordering. The simulation conditions are the same as in the upper half of the

Algorithm 3 d-dimensional conditioning algorithm with univariate reordering

procedure RCMVN($\tilde{\mathbf{\Sigma}}, \tilde{\mathbf{a}}, \tilde{\mathbf{b}}, d$)
  $\tilde{\mathbf{y}} \leftarrow \mathbf{0}$, $\tilde{\mathbf{C}} \leftarrow \tilde{\mathbf{\Sigma}}$
  for $i = 1 : m$ **do**
    if $i > 1$ **then**
      $\tilde{\mathbf{y}}[i-1] \leftarrow \frac{\phi(\tilde{a}') - \phi(\tilde{b}')}{\Phi(\tilde{b}') - \Phi(\tilde{a}')}$
    end if
    $j \leftarrow \arg\min_{i \le j \le m} \left\{ \Phi\left(\frac{\tilde{\mathbf{b}}[j] - \tilde{\mathbf{C}}[j,1:i-1]\tilde{\mathbf{y}}[1:i-1]}{\sqrt{\tilde{\mathbf{\Sigma}}[j,j] - \tilde{\mathbf{C}}[j,1:i-1]\tilde{\mathbf{C}}^T[j,1:i-1]}} \right) - \Phi\left(\frac{\tilde{\mathbf{a}}[j] - \tilde{\mathbf{C}}[j,1:i-1]\tilde{\mathbf{y}}[1:i-1]}{\sqrt{\tilde{\mathbf{\Sigma}}[j,j] - \tilde{\mathbf{C}}[j,1:i-1]\tilde{\mathbf{C}}^T[j,1:i-1]}} \right) \right\}$
    $\tilde{\mathbf{\Sigma}}[:,(i,j)] \leftarrow \tilde{\mathbf{\Sigma}}[:,(j,i)], \tilde{\mathbf{\Sigma}}[(i,j),:] \leftarrow \tilde{\mathbf{\Sigma}}[(j,i),:]$
    $\tilde{\mathbf{C}}[:,(i,j)] \leftarrow \tilde{\mathbf{C}}[:,(j,i)], \tilde{\mathbf{C}}[(i,j),:] \leftarrow \tilde{\mathbf{C}}[(j,i),:]$
    $\tilde{\mathbf{a}}[(i,j)] \leftarrow \tilde{\mathbf{a}}[(j,i)]$, $\tilde{\mathbf{b}}[(i,j)] \leftarrow \tilde{\mathbf{b}}[(j,i)]$
    $\tilde{\mathbf{C}}[i,i] \leftarrow \sqrt{\tilde{\mathbf{\Sigma}}[i,i] - \tilde{\mathbf{C}}[i,1:i-1]\tilde{\mathbf{C}}^T[i,1:i-1]}$
    $\tilde{\mathbf{C}}[j,i] \leftarrow (\tilde{\mathbf{\Sigma}}[j,i] - \tilde{\mathbf{C}}[i,1:i-1]\tilde{\mathbf{C}}^T[j,1:i-1])/\tilde{\mathbf{C}}[i,i]$, for $j = i+1, \ldots, m$
    $\tilde{a}' \leftarrow (\tilde{\mathbf{a}}[i] - \tilde{\mathbf{C}}[i,1:i-1]\tilde{\mathbf{y}}[1:i-1])/\tilde{\mathbf{C}}[i,i]$
    $\tilde{b}' \leftarrow (\tilde{\mathbf{b}}[i] - \tilde{\mathbf{C}}[i,1:i-1]\tilde{\mathbf{y}}[1:i-1])/\tilde{\mathbf{C}}[i,i]$
  end for
  CMVN($\tilde{\mathbf{\Sigma}}, \tilde{\mathbf{a}}, \tilde{\mathbf{b}}, d$) as in Algorithm 2
end procedure

table. Reordering results in significant improvement in estimation accuracy if we compare the errors from the non-reordered conditioning method with those from the reordered conditioning method in Table 2. The time cost for the univariate reordering is approximately 0.06s when $m = 128$, which accounts for only a small proportion of the total time cost. The complexity of univariate reordering is $O(m^3)$ assuming that the complexity of the matrix multiplication, $\mathbf{M}^1_{m_1 \times m_2} \mathbf{M}^2_{m_2 \times m_3}$, is $O(m_1 m_2 m_3)$. Although this complexity increases rapidly as m increases, the size of each diagonal block in the hierarchical covariance matrix, $\mathcal{H}$, is usually much smaller than the problem size, n, in high-dimensional cases, which makes univariate reordering a feasible option for solving the small MVN problem presented by each diagonal block.

4 The Hierarchical-Block Conditioning Method

The simulations for the d-dimensional conditioning method in Section 3 indicate that the algorithm coupled with univariate reordering could be an efficient alternative to Monte-Carlo-based methods. Genton et al. (2018) found that building the Quasi-Monte Carlo method on top of a hierarchical representation reduces the computational complexity per sample from $O(n^2)$ to $O(mn + kn \log(n/m))$, which is especially significant when the problem size, n, is large. As an additional benefit, the memory required for storage is also minimized when a hierarchical representation is used. In this section, we solve the n-dimensional MVN problem with the hierarchical covariance matrix and compare the efficiency of using the d-dimensional conditioning method with that of the Monte-Carlo-based method for solving the m-dimensional MVN problems presented by the diagonal blocks.

4.1 The hierarchical-block conditioning algorithm

The hierarchical-block conditioning method uses the conditioning technique with the hierarchical representation of the covariance matrix, which decomposes the n-dimensional integration into:

$$\Phi_n(\mathbf{a}, \mathbf{b}, \mathbf{\Sigma}) = \int_{\mathbf{a}'_1}^{\mathbf{b}'_1} \phi_m(\mathbf{x}_1; \mathbf{B}_1\mathbf{B}_1^T) \int_{\mathbf{a}'_2}^{\mathbf{b}'_2} \phi_m(\mathbf{x}_2; \mathbf{B}_2\mathbf{B}_2^T) \cdots \int_{\mathbf{a}'_r}^{\mathbf{b}'_r} \phi_m(\mathbf{x}_r; \mathbf{B}_r\mathbf{B}_r^T)\, \mathrm{d}\mathbf{x}_r \cdots \mathrm{d}\mathbf{x}_2\, \mathrm{d}\mathbf{x}_1, \quad (5)$$

where $\mathbf{a}'_i$ and $\mathbf{b}'_i$, $i = 1, \ldots, r$, are the corresponding segments of the updated $\mathbf{a}$ and $\mathbf{b}$. Truncated expectations are computed for each diagonal block, $\mathbf{B}_i\mathbf{B}_i^T$, and used for updating the integration limits to the right as shown in Equation (5). The hierarchical-block conditioning algorithm is presented as Algorithm 4. It transforms the n-dimensional MVN problem into r m-dimensional problems. For clarity of presentation, we assume that n and m are both powers of 2. The dimension function, dim, therefore returns an integer and the offset refers to the number of rows or columns leading the matrix block, $\mathbf{U}_{i-1}\mathbf{V}_{i-1}^T$. The function *choldecomp_hmatrix* implements the hierarchical Cholesky factorization and returns $\mathbf{B}$ and $\mathbf{U}\mathbf{V}^T$ as a vector of matrices. A slight modification is needed when n and m assume arbitrary values from the set of positive integers.

To compute the MVN probability and truncated expectations in m-dimensions, we can either perform the Monte-Carlo-based method directly or use the d-dimensional conditioning algorithm introduced in Section 3 as a second level of conditioning. In fact, the former method is a special

Algorithm 4 Hierarchical-block conditioning algorithm

procedure HCMVN($\mathbf{a}, \mathbf{b}, \mathbf{\Sigma}, d$)
  $\mathbf{x} \leftarrow \mathbf{0}$ and $P \leftarrow 1$
  $[\mathbf{B}, \mathbf{UV}] \leftarrow choldecomp\_hmatrix(\mathbf{\Sigma})$
  for $i = 1 : r$ **do**
    $j \leftarrow (i-1)m$
    if $i > 1$ **then**
      $o_r \leftarrow$row offset of $\mathbf{U}_{i-1}\mathbf{V}_{i-1}^T$, $o_c \leftarrow$column offset of $\mathbf{U}_{i-1}\mathbf{V}_{i-1}^T$
      $l \leftarrow \dim(\mathbf{U}_{i-1}\mathbf{V}_{i-1}^T)$
      $\mathbf{g} \leftarrow \mathbf{U}_{i-1}\mathbf{V}_{i-1}^T\mathbf{x}[o_c + 1 : o_c + l]$
      $\mathbf{a}[o_r + 1 : o_r + l] \leftarrow \mathbf{a}[o_r + 1 : o_r + l] - \mathbf{g}$
      $\mathbf{b}[o_r + 1 : o_r + l] \leftarrow \mathbf{b}[o_r + 1 : o_r + l] - \mathbf{g}$
    end if
    $\mathbf{a}_i \leftarrow \mathbf{a}[j + 1 : j + m]$, $\mathbf{b}_i \leftarrow \mathbf{b}[j + 1 : j + m]$
    $P \leftarrow P \cdot \Phi_m(\mathbf{X}_i; \mathbf{a}_i, \mathbf{b}_i, \mathbf{B}_i\mathbf{B}_i^T)$
    $\mathbf{x}[j + 1 : j + m] \leftarrow \mathbf{B}_i^{-1}\mathrm{E}[\mathbf{X}_i]$
  end for
  return P
end procedure

case of the latter when $d = m$ and has much higher complexity because the simulations are performed in m-dimensions. The latter method further transforms each m-dimensional MVN problem into s d-dimensional MVN problems as shown in Equation (2) although additional error can be introduced by the conditioning technique on the second level. In Sections 4.2 and 4.3, we compare the two methods for solving the m-dimensional problem under simple covariance structures.

4.2 Simulation with a constant covariance matrix

We first use the constant covariance structure to compare the performance of applying the d-dimensional conditioning method with that of the Monte-Carlo-based method to each m-dimensional MVN problem. The covariance matrix is ideal for hierarchical representation because any off-diagonal block can be written exactly as the product of two rank-1 matrices. However, the correlation does not decay with the distance between indices and a stronger correlation usually leads to a larger error for the conditioning method (Trinh and Genz, 2015). In this section, we progressively define three methods to highlight the efficiencies gained from the conditioning method and the accuracy gained from the univariate reordering. Method 1 applies Equations (3) and (4) to compute the m-dimensional MVN probability and truncated expectations directly. Method 2 employs the d-dimensional conditioning method as described in Algorithm 2 to compute the m-dimensional MVN probability and truncated expectations. Method 3 begins with univariate reordering as described in Algorithm 3 and then uses the d-dimensional conditioning method. We select $d = 4$ for Method 2 and Method 3 because it provides a more balanced tradeoff between the error and the computation time given the results in Table 2.

In this experiment, the correlation of the constant covariance matrix is set at 0.7 for a medium correlation strength. The lower integration bound, $\mathbf{a}$, is set at $-\infty$ and the upper bound is independently generated from $\mathrm{U}(0, n)$, which makes the expectation of the simulated probability roughly 0.7. The upper half of Table 3 summarizes the time and relative error of the

Table 3: Errors and execution times under the constant covariance structure and 1D exponential covariance structure. Method 1 depends on the Monte-Carlo-based method for the MVN probability and truncated expectations. Method 2 applies the d-dimensional conditioning method. And Method 3 employs the d-dimensional conditioning method with univariate reordering. The constant correlation is set at 0.7. The time and absolute error are based on 20 replicates.

Constant covariance structure

m		16			32			64	
n	512	1,024	2,048	512	1,024	2,048	512	1,024	2,048
M1	11.9%	13.6%	10.9%	11.8%	13.6%	10.9%	11.9%	13.6%	10.9%
	7.6s	*14.7s*	*30.7s*	*16.7s*	*33.0s*	*59.9s*	*41.2s*	*78.1s*	*133.3s*
M2	11.9%	13.6%	10.9%	11.8%	13.7%	10.9%	11.9%	13.7%	11.0%
	0.1s	*0.3s*	*0.7s*	*0.1s*	*0.3s*	*0.6s*	*0.1s*	*0.3s*	*0.6s*
M3	11.9%	13.6%	10.9%	11.8%	13.7%	10.9%	11.9%	13.6%	11.0%
	0.1s	*0.3s*	*0.7s*	*0.1s*	*0.3s*	*0.6s*	*0.2s*	*0.3s*	*0.6s*
1D exponential covariance structure									
M1	8.0%	8.5%	8.7%	4.3%	5.2%	3.3%	1.7%	2.8%	1.4%
	21.4s	*42.7s*	*76.8s*	*40.3s*	*83.4s*	*153.9s*	*74.0s*	*144.1s*	*298.7s*
M2	19.9%	20.5%	25.8%	20.4%	20.9%	25.7%	20.6%	20.5%	25.6%
	0.5s	*0.9s*	*1.8s*	*0.5s*	*0.9s*	*1.8s*	*0.4s*	*0.8s*	*1.7s*
M3	7.0%	8.3%	8.7%	4.4%	5.2%	3.3%	1.8%	2.6%	1.3%
	0.5s	*0.9s*	*1.8s*	*0.4s*	*0.9s*	*1.8s*	*0.4s*	*0.8s*	*1.9s*

three methods under the constant covariance structure based on 20 replicates. We use 20 as the sample size instead of 250 as in Table 2 because the covariance structure is fixed, leading to a much smaller standard deviation for the estimators. Unlike for other covariance structures, the benchmark for the constant correlation case can be accurately calculated with a 1-dimensional integration

$$\Phi_n(\mathbf{X}; -\infty, \mathbf{b}, \mathbf{\Sigma}_{const}) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} e^{-\frac{1}{2}x^2} \prod_{i=1}^{n} \Phi_1 \left(\frac{b_i + \sqrt{\theta} x}{1-\theta} \right) \mathrm{d}x,$$

where θ is the constant correlation coefficient. The estimation errors from the three methods are very close under the same set of constant-covariance MVN problems, indicating that the dominant error comes from the first level of conditioning and the d-dimensional conditioning methods can estimate the MVN probability of size m almost as accurately as the Quasi-Monte Carlo method. Thus the estimation accuracy of the three methods cannot be clearly distinguished. By comparing the time costs, we can conclude that the d-dimensional conditioning method leads to significant

Table 4: Complexity decomposition of the three methods. Method 1 depends on the Monte-Carlo-based method for the MVN probability and truncated expectations. Method 2 applies the d-dimensional conditioning method. Method 3 employs the d-dimensional conditioning method with univariate reordering. The three parts of the complexity are the calculation of the MVN probability (MVN prob), the calculation of the truncated expectations (Trunc exp), and the update of the integration limits with truncated expectations (Upd limits). The latter two share the same asymptotic order in all three complexity terms. The updating cost is independent of the method.

	MVN prob	Trunc exp	Upd limits
M1	$\frac{n}{m}M(m)$	$2nM(m) + O(nm^2)$	$O(mn + kn\log(n/m))$
M2	$\frac{n}{d}M(d) + O(m^2n)$	$2nM(d) + O(nd^2)$	$O(mn + kn\log(n/m))$
M3	$\frac{n}{d}M(d) + O(m^2n)$	$2nM(d) + O(nd^2)$	$O(mn + kn\log(n/m))$

efficiencies. In addition, the complexities of all methods appear to be in a linear relationship with n while the complexities of Method 2 and Method 3 are not sensitive to m.

For a clearer comparison of the complexities, we decompose the complexity of Algorithm 4 into three parts and list the complexity for each part in Table 4, where $M(\cdot)$ denotes the complexity of the Quasi-Monte Carlo simulation in the given dimension. Table 4 shows that the time efficiency of the d-dimensional conditioning algorithm mainly comes from lowering the dimension in which the Quasi-Monte Carlo simulation is performed. The complexity of the univariate reordering is $O(m^2n)$, the same as the complexity of computing the MVN probabilities in Method 2, resulting in an identical major complexity component for Method 2 and Method 3. Since Methods 2 and 3 perform the Quasi-Monte Carlo simulation in d-dimensions, these two methods are not greatly affected by the choice of m.

4.3 Simulation with 1D exponential covariance matrix

The second covariance structure used for comparing the accuracy and efficiency of the three methods is the 1D exponential covariance structure. Under this structure, the correlation is $\rho(X_i, X_j) = \exp(-d_{ij}/\beta)$, where d_{ij} is the distance between X_i and X_j, and β is the coefficient controling the correlation decay rate. After the random variables are indexed with a geometrically increasing order, the off-diagonal blocks can be written accurately as the product of two rank-

1 matrices as in the case of the constant covariance. However, unlike the constant covariance example, the 1D exponential correlation decays quickly along with the distance between indices. In this experiment, n points are selected on the real line with neighboring distance 1, β is set to 10. Here, $\mathbf{a}, \mathbf{b}$ and d are selected in the same fashion as in Section 4.2. Results from the Quasi-Monte Carlo simulation with standard error smaller than 5×10^{-3} are used as the benchmark for calculating the absolute errors. The lower half of Table 3 describes the average time and error of the three methods based on 20 replicates.

We find that the errors of the three methods under the 1D exponential structure fall into two bins. The error from Method 1 is close to that from Method 3, and both are much smaller than that from Method 2, indicating that univariate reordering effectively reduces the estimation error and even makes up for the loss of information resulting from the second level of conditioning. In contrast, under the constant covariance structure, a relatively large error is already generated from the first level of conditioning, which makes the improvement from univariate conditioning insignificant. Since there are two levels of conditioning in Method 2 and Method 3, we refer to both methods as two-level hierarchical-block conditioning methods hereafter. A similar conclusion about algorithmic efficiencies can be drawn from the results of the 1D exponential covariance structure that Methods 2 and 3 are more efficient. However, it is worth noting that the computation time for the same combination of (n, m) and the computational method is smaller under the 1D exponential covariance structure, probably because the Quasi-Monte Carlo simulation is faster under the weaker covariance structure. By comparing the results of the two covariance structures in Table 3, we may also conclude that increasing the diagonal block size, m, reduces the estimation error under a decaying correlation structure because, intuitively, a larger m captures more correlation information. This is less obvious for Method 2 and Method 3, however, because the second level of conditioning unavoidably causes some of the correlation information within each diagonal block to be discarded.

Based on the results from the constant covariance structure and the 1D exponential covariance

structure, we argue that Method 3 has the best combination of efficiency and accuracy. Hence from this point on, we consider only Method 3, the two-level hierarchical-block conditioning method proceded by univariate reordering for each diagonal block. We compare results from Method 3 with those from the hierarchical Quasi-Monte Carlo method (Genton et al., 2018), which could be considered the state-of-the-art technique for high-dimensional MVN problems at the time of this writing. To test the efficiencies of both methods on general MVN problems, we use a covariance structure in two dimensions.

4.4 Simulation with a 2D exponential covariance matrix

The covariance matrix from 1-dimensional geometry has a bounded rank for off-diagonal blocks under most covariance models because the sample points are distinctly separable. However, in two and higher dimensions, a proper indexing scheme needs to be implemented to maintain the locality of the samples despite the fact that the rank of the off-diagonal blocks usually grows in a log fashion. Morton's order is extensively used for reducing the dimensionality of data to one while leaving the sample points in the geometric vicinity still closer in indices. In this section, we assume a 2D exponential covariance structure and use Morton's order for indexing the sample points on the plane. As a result, the ranks of the off-diagonal blocks grow with their block sizes but at a much slower rate than $O(n)$. This is visible in the second column of Table 5. The diagonal block size m and the second conditioning dimension, d, collectively determine the amount of correlation information that is captured. Here, m reduces the estimation error at the cost of increased univariate reordering time for each diagonal block while d improves the result by performing Quasi-Monte Carlo simulation in higher dimensions. In Sections 4.2 and 4.3, we found that the two-level hierarchical-block conditioning methods are insensitive to the choice of m. We therefore fix $m = 64$ for the 2D covariance structure in this section and examine the effectiveness of our algorithm when the second conditioning dimension, d, is set to 2, 4, and 6. To test the sensitivity with respect to the correlation strength, we perform the estimation under

Table 5: Performance of the two-level hierarchical-block conditioning method without reordering. The covariance matrix is generated from the exponential covariance function based on n points randomly distributed in the unit square indexed with Morton's order. The Quasi-Monte Carlo method (mvn), the hierarchical Quasi-Monte Carlo method (hmvn) and the two-level hierarchical-block conditioning method (hccmvn$_d$) with $d = 2, 4$ and 6 are compared.

			$\beta = 0.3$				
n	$k_{min}, k_{avg}, k_{max}$	$\mathcal{H}$(Megabytes)	mvn	hmvn	hccmvn$_2$	hccmvn$_4$	hccmvn$_6$
256	15, 20, 31	0.22	0.4% *0.3s*	0.4% *0.6s*	22.2% *0.1s*	20.3% *0.2s*	20.5% *1.0s*
1,024	14, 24, 60	1.58	0.7% *2.1s*	0.7% *2.7s*	27.0% *0.5s*	25.1% *1.1s*	25.0% *5.0s*
4,096	14, 26, 116	11.67	0.8% *30.8s*	0.8% *20.0s*	37.3% *2.9s*	36.1% *5.2s*	36.6% *25.6s*
16,384	13, 26, 222	87.05	0.8% *1255.5s*	0.8% *91.5s*	45.2% *14.1s*	44.6% *25.9s*	44.4% *121.5s*
			$\beta = 0.1$				
n	$k_{min}, k_{avg}, k_{max}$	$\mathcal{H}$(Megabytes)	mvn	hmvn	hccmvn$_2$	hccmvn$_4$	hccmvn$_6$
256	14, 18, 28	0.21	0.2% *0.3s*	0.2% *0.6s*	10.5% *0.1s*	8.5% *0.2s*	9.0% *0.7s*
1,024	12, 23, 58	1.55	0.5% *2.0s*	0.5% *2.5s*	20.8% *0.5s*	18.2% *1.1s*	18.4% *4.8s*
4,096	14, 26, 117	11.83	0.7% *27.0s*	0.7% *11.2s*	30.0% *2.2s*	28.3% *4.6s*	28.9% *20.6s*
16,384	13, 27, 229	89.75	0.8% *1219.3s*	0.9% *53.9s*	41.5% *10.1s*	39.3% *19.6s*	39.3% *87.8s*
			$\beta = 0.03$				
n	$k_{min}, k_{avg}, k_{max}$	$\mathcal{H}$(Megabytes)	mvn	hmvn	hccmvn$_2$	hccmvn$_4$	hccmvn$_6$
256	9, 12, 18	0.18	0.0% *0.3s*	0.0% *0.6s*	0.9% *0.1s*	0.6% *0.2s*	0.7% *0.7s*
1,024	12, 20, 50	1.39	0.2% *2.0s*	0.2% *2.6s*	5.3% *0.4s*	4.1% *1.0s*	4.2% *3.7s*
4,096	13, 25, 111	11.22	0.4% *26.7s*	0.4% *11.2s*	13.2% *2.2s*	11.3% *4.6s*	11.6% *18.9s*
16,384	14, 27, 227	88.90	0.5% *1225.1s*	0.5% *43.2s*	24.7% *9.5s*	21.2% *19.2s*	21.7% *83.5s*

$\beta = 0.3, 0.1$, and 0.03, representing strong, medium, and weak correlation strengths.

Table 5 presents the results for the two-level hierarchical-block conditioning method. The algorithm is implemented in C++ and compared with that in Genton et al. (2018). Table 5 shows the relative error and time, averaged from the same set of 20 problem replicates for each

dimension n. For each replicate, the upper bound, $\mathbf{b}$, is generated from $\mathrm{U}(0, n)$ and the lower bound $\mathbf{a}$ is assumed to be $-\infty$ as in Trinh and Genz (2015). We construct the hierarchical covariance matrix, $\mathcal{H}$, prior to Cholesky factorization, and we use an adaptively decreasing tolerance level, starting at 5×10^{-4}, for building $\mathcal{H}$ to guarantee the positive-definiteness of $\mathcal{H}$. The second column describes the ranks of the off-diagonal blocks of $\mathcal{H}$ that grow approximately in a $\log n$ fashion. The last five columns correspond to the error and time for the three techniques evaluated in this section, namely the Quasi-Monte Carlo method, the hierarchical Quasi-Monte Carlo method, and the two-level hierarchical-block conditioning method with $d = 2, 4$, and 6. The two-level hierarchical-block conditioning method performs the worst, providing an extremely poor estimation despite having a higher computational efficiency. After comparing the estimation errors under the three types of correlation strengths, we conclude that increasing the second-level conditioning dimension, d, can slightly improve the estimation accuracy and that the conditioning method is more sensitive to the correlation strength than the Monte-Carlo-based methods. For example, the estimation error for the conditioning method is negligible under $n = 256$ and $\beta = 0.03$ but grows more rapidly than the other two methods when n or β increases. In fact, the correlation strength is essentially increased when n increases while β remains unchanged. The method, when used with general covariance matrices without a reordering strategy, does not produce sufficiently accurate results. This motivated the development of the reordering strategy described in the next section. We note that the time difference between this paper and Genton et al. (2018) for the Quasi-Monte Carlo method and the hierarchical Quasi-Monte Carlo method is due to different hardware and machine-level implementations.

5 Block Reordering

In the context of conditioning algorithms, the integration variables on the farther left, which also have higher priority in terms of calculation, tend to have more impact on the accuracy of the estimation. Univariate reordering can effectively reduce the estimation error for low-

Algorithm 5 Block-wise reordering

procedure BLOCKREORDER($\mathbf{G}, \rho, \mathbf{a}, \mathbf{b}, m, \mathbf{ind}$)
  for $i = 1 : m : n - m + 1$ **do**
    $\mathbf{subind} \leftarrow \mathbf{ind}[i : i + m - 1]$
    $\mathbf{A} \leftarrow \rho(\mathbf{G}, \mathbf{subind})$
    $\mathbf{a}' \leftarrow \mathbf{a}[\mathbf{subind}]$
    $\mathbf{b}' \leftarrow \mathbf{b}[\mathbf{subind}]$
    $\mathbf{P} = [\mathbf{P}, \text{RCMVN}(\mathbf{A}, \mathbf{a}', \mathbf{b}', 1).P]$ as in Algorithm 3
  end for
  $\mathbf{sort}(\mathbf{ind}, \mathbf{P}, m)$
  return ind
end procedure

dimensional MVN problems (Trinh and Genz, 2015) but applying it to n integration variables has a complexity of $O(n^3)$ and is likely to spoil the low-rank feature of the off-diagonal blocks, which is not desirable for an efficiency-oriented algorithm. The construction of the hierarchical covariance matrix essentially assumes that the correlation within each m-sized block of variables is strong while the correlation between these blocks is weak. Building on this idea, we rearrange the diagonal blocks of size m with their probabilities increasing from left to right, which is consistent with the idea of univariate reordering. The reordering of the blocks is implemented only once instead of recursively due to the high complexity of calculating the truncated expectations in m-dimensions. The probability for each block is estimated with the conditioning method introduced in Algorithm 2 and d is set to 1. Hence, the univariate reordering is performed at the same time. The algorithm for the block-wise reordering is summarized as Algorithm 5. For clarity, we use $\rho(\mathbf{G}, \mathbf{ind})$ for constructing a correlation matrix based on the correlation function, ρ, geometry, $\mathbf{G}$, and indices, $\mathbf{ind}$. Here, $\mathbf{sort}(\mathbf{ind}, \mathbf{P}, m)$ stands for rearranging the size-m segments of $\mathbf{ind}$ based on the vector, $\mathbf{P}$, in an increasing order.

To examine the increased accuracy from the preceding block reordering, we implemented the 2D exponential experiments the same way as in Section 4.4 with an extra reordering layer before the two-level hierarchical-block conditioning algorithm. Table 6 compares the reordered conditioning algorithm with the Monte-Carlo-based algorithms and the additional time cost from block

Table 6: Performance of the two-level hierarchical-block conditioning method with reordering. The covariance matrix is generated from the exponential covariance function based on n points randomly distributed in the unit square indexed first with Morton's order and then with block reordering as in Algorithm 5. The Quasi-Monte Carlo method (mvn), the hierarchical Quasi-Monte Carlo method (hmvn) and the two-level hierarchical-block conditioning method (hccmvn$_d$) with $d = 2, 4$ and 6 are compared. MB in the third column is short for megabytes.

$\beta = 0.3$

n	$k_{min}, k_{avg}, k_{max}$	$\mathcal{H}$(MB)	mvn	hmvn	hccmvn$_2$	hccmvn$_4$	hccmvn$_6$	reorder
256	11, 20, 37	0.22	0.4% *0.3s*	0.4% *0.6s*	1.0% *0.1s*	1.0% *0.2s*	1.0% *0.9s*	0.00s
1,024	1, 20, 82	1.71	0.7% *2.1s*	0.7% *2.7s*	2.8% *0.5s*	2.7% *1.1s*	2.7% *5.0s*	0.01s
4,096	1, 22, 151	13.94	0.8% *30.8s*	0.8% *20.0s*	2.1% *2.5s*	2.1% *5.2s*	2.1% *25.0s*	0.03s
16,384	1, 24, 287	114.14	0.8% *1255.5s*	0.8% *91.5s*	3.0% *13.1s*	3.0% *24.1s*	3.0% *114.9s*	0.14s

$\beta = 0.1$

n	$k_{min}, k_{avg}, k_{max}$	$\mathcal{H}$(MB)	mvn	hmvn	hccmvn$_2$	hccmvn$_4$	hccmvn$_6$	reorder
256	10, 18, 35	0.22	0.2% *0.3s*	0.2% *0.6s*	0.3% *0.1s*	0.3% *0.2s*	0.3% *0.7s*	0.00s
1,024	1, 19, 82	1.70	0.5% *2.0s*	0.5% *2.5s*	0.9% *0.5s*	0.8% *1.0s*	0.8% *4.4s*	0.01s
4,096	1, 22, 151	14.03	0.7% *27.0s*	0.7% *11.2s*	0.3% *2.0s*	0.3% *4.4s*	0.3% *19.5s*	0.03s
16,384	1, 25, 287	115.75	0.8% *1219.3s*	0.9% *53.9s*	0.6% *10.1s*	0.6% *19.7s*	0.6% *86.6s*	0.13s

$\beta = 0.03$

n	$k_{min}, k_{avg}, k_{max}$	$\mathcal{H}$(MB)	mvn	hmvn	hccmvn$_2$	hccmvn$_4$	hccmvn$_6$	reorder
256	6, 12, 24	0.19	0.0% *0.3s*	0.0% *0.6s*	0.0% *0.1s*	0.0% *0.1s*	0.0% *0.5s*	0.00s
1,024	1, 16, 82	1.59	0.2% *2.0s*	0.2% *2.6s*	0.0% *0.4s*	0.0% *0.9s*	0.0% *3.6s*	0.01s
4,096	1, 20, 151	13.70	0.4% *26.7s*	0.4% *11.2s*	0.0% *1.9s*	0.0% *4.1s*	0.0% *18.0s*	0.03s
16,384	1, 25, 287	115.74	0.5% *1225.1s*	0.5% *43.2s*	0.0% *9.3s*	0.0% *18.5s*	0.0% *78.9s*	0.09s

reordering is measured separately and shown in the last column. The original Morton's order is disturbed by the block reordering. As a result, the ranks and the memory cost of the off-diagonal blocks increase as indicated in the second and third columns. The resulting estimation error for the two-level hierarchical-block conditioning method is significantly reduced by the preceding

block reordering. The error is below 3% overall with strong correlation, $\beta = 0.3$, while below 1% with medium and weak correlations. Errors of such magnitude can make the conditioning method a good substitute for Monte-Carlo-based methods. The run time costs for the two-level hierarchical-block conditioning method are not sensitive to the ranks of the off-diagonal blocks. Columns 6 to 8 of Table 6 indicate similar time performance compared with Table 5 because each matrix-vector multiplication is performed only once in the conditioning algorithm as described in Algorithm 4. Since the choice of d has little impact on the estimation accuracy when the preceding block reordering exists, we use $d = 2$ as the benchmark for comparing the time efficiencies of the hierarchical-block conditioning method and the hierarchical Quasi-Monte Carlo method. On average, the two-level hierarchical-block conditioning method, *hccmvn*, is five times faster than the hiararchical Quasi-Monte Carlo method, *hmvn*, but slightly more sensitive to the correlation strength. The extra time used for block reordering is negligible compared with the overall time costs of the conditioning methods.

6 Discussion

We presented a d-dimensional conditioning algorithm as an extension of the bivariate conditioning algorithm from Trinh and Genz (2015) and, based on it, we described a hierarchical-block conditioning method for estimating MVN probabilities that is suitable for high-dimensional problems. The d-dimensional conditioning algorithm delivers more accurate estimation under randomly generated covariance structures as d increases. The hierarchical technique takes advantage of the low-rank features of the common covariance models used in spatial statistics, which significantly reduce the computation time and the storage cost. We also introduced a block reordering scheme that preserves the low-rank feature and significantly improves the estimation accuracy with little additional cost. Combining the three, we introduced a two-level hierarchical-block conditioning algorithm that can further shorten the computation time of MVN probabilities based on the hierarchical Quasi-Monte Carlo method. There are two parameters for the algorithm, namely

the diagonal block size, m, and the conditioning dimension, d, which collectively control the accuracy and the complexity of the algorithm. The dimension, m, should be large enough to yield savings from low-rank structures in the off-diagonal blocks yet not too large to make computation of the diagonal blocks too costly. The dimension, d, increases the estimation accuracy for the probability and truncated expectations within each diagonal block but also increases the complexity quickly because of the calculations of truncated expectations. A value of 2 or 4 for d can produce a sufficiently small error when a block reordering is performed. The algorithm provides a practical method for calculating the MVN probabilities in tens of thousands of dimensions.

The estimation for the truncated expectations makes up the largest computational complexity in the current algorithm and is calculated based on Equations (3) and (4). Future progress in estimating the truncated expectations of MVN random variables is expected to reduce the run time of this algorithm significantly.

References

Anderson, E., Bai, Z., Bischof, C., Blackford, S., Demmel, J., Dongarra, J., Du Croz, J., Greenbaum, A., Hammerling, S., McKenney, A., et al. (1999), *LAPACK Users' Guide Third Edition*, SIAM Publications, Philadelphia, PA.

Azzalini, A. and Capitanio, A. (2014), *The Skew-Normal and Related Families*, Cambridge University Press, Cambridge.

Brown, B. M. and Resnick, S. I. (1977), "Extreme values of independent stochastic processes," *Journal of Applied Probability*, 14, 732–739.

Caflisch, R. E. (1998), "Monte Carlo and quasi-Monte Carlo methods," *Acta Numerica*, 7, 1–49.

Genton, M. G. (2004), *Skew-Elliptical Distributions and Their Applications: A Journey Beyond Normality*, CRC Press.

Genton, M. G., Keyes, D. E., and Turkiyyah, G. M. (2018), "Hierarchical decompositions for the computation of high-dimensional multivariate normal probabilities," *Journal of Computational and Graphical Statistics, in press.*

Genz, A. (1992), "Numerical computation of multivariate normal probabilities," *Journal of Computational and Graphical Statistics*, 1, 141–149.

Genz, A. and Bretz, F. (2009), *Computation of Multivariate Normal and t Probabilities*, vol. 195, Springer Science & Business Media.

Gupta, R. C. and Brown, N. (2001), "Reliability studies of the skew-normal distribution and its application to a strength-stress model," *Communications in Statistics-Theory and Methods*, 30, 2427–2445.

Hackbusch, W. (2015), *Hierarchical Matrices: Algorithms and Analysis*, vol. 49, Springer.

Kan, R. and Robotti, C. (2017), "On moments of folded and truncated multivariate normal distributions," *Journal of Computational and Graphical Statistics, in press*.

Kotz, S. and Nadarajah, S. (2004), *Multivariate t-Distributions and Their Applications*, Cambridge University Press.

Morton, G. M. (1966), *A Computer Oriented Geodetic Data Base and A New Technique In File Sequencing*, International Business Machines Company, New York.

Muthen, B. (1990), "Moments of the censored and truncated bivariate normal distribution," *British Journal of Mathematical and Statistical Psychology*, 43, 131–143.

Niederreiter, H. (1992), "New methods for pseudorandom numbers and pseudorandom vector generation," in *Proceedings of the 24th Conference on Winter Simulation*, New York, NY, USA: ACM, WSC '92, pp. 264–269.

Owen, A. and Zhou, Y. (2000), "Safe and effective importance sampling," *Journal of the American Statistical Association*, 95, 135–143.

R Core Team (2016), *R: A Language and Environment for Statistical Computing*, R Foundation for Statistical Computing, Vienna, Austria.

Schlather, M. (2002), "Models for stationary max-stable random fields," *Extremes*, 5, 33–44.

Smith, R., Tawn, J., and Yuen, H. (1990), "Statistics of multivariate extremes," *International Statistical Review*, 58, 47–58.

Stewart, G. W. (1980), "The efficient generation of random orthogonal matrices with an application to condition estimators," *SIAM Journal on Numerical Analysis*, 17, 403–409.

Trinh, G. and Genz, A. (2015), "Bivariate conditioning approximations for multivariate normal probabilities," *Statistics and Computing*, 25, 989–996.