Computation of High-Dimensional Multivariate Normal and Student-t Probabilities Based on Matrix Compression Schemes

Dissertation by
Jian Cao

In Partial Fulfillment of the Requirements
For the Degree of
Doctor of Philosophy

King Abdullah University of Science and Technology
Thuwal, Kingdom of Saudi Arabia

April, 2020
EXAMINATION COMMITTEE PAGE

The dissertation of Jian Cao is approved by the examination committee:

Committee Chairperson: Marc G. Genton
Committee Members: Marc G. Genton, David E. Keyes, Håvard Rue, Victor Panaretos
ABSTRACT

Computation of High-Dimensional Multivariate Normal and Student-\(t\) Probabilities Based on Matrix Compression Schemes

Jian Cao

The first half of the thesis focuses on the computation of high-dimensional multivariate normal (MVN) and multivariate Student-\(t\) (MVT) probabilities. Chapter 2 generalizes the bivariate conditioning method to a \(d\)-dimensional conditioning method and combines it with a hierarchical representation of the \(n \times n\) covariance matrix. 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 dominant complexity term of the algorithm to be \(O(n \log n)\). Chapter 3 improves the block reordering scheme from Chapter 2 and integrates it into the Quasi-Monte Carlo simulation under the tile-low-rank representation of the covariance matrix. Simulations up to dimension 65,536 suggest that this method can improve the run time by one order of magnitude compared with the hierarchical Monte Carlo method. The second half of the thesis discusses a novel matrix compression scheme with Kronecker products, an R package that implements the methods described in Chapter 3, and an application study with the probit Gaussian random field. Chapter 4 studies the potential of using the sum of Kronecker products (SKP) as a compressed covariance matrix representation. Experiments show that this new SKP representation can save the memory footprint by one order of magnitude compared with the hierarchical representation for covariance matrices from large grids and the Cholesky factorization in one million dimensions can be achieved within 600 seconds. In Chapter 5, an R package is introduced that implements the methods in Chapter 3 and show how the
package improves the accuracy of the computed excursion sets. Chapter 3 derives the posterior properties of the probit Gaussian random field, based on which model selection and posterior prediction are performed. With the tlmvnmvt package, the computation becomes feasible in tens of thousands of dimensions, where the prediction errors are significantly reduced.
ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to my advisor Prof. Marc G. Genton, who has kept me on track on the lengthy path towards a Ph.D. with his patience and guidance. Besides my advisor, I would like to thank my collaborators, Prof. David Keyes, Prof. George Turkiyyah, and Prof. Daniele Durante for their inspiring ideas, critical comments, and academic writing suggestions. I would like to say thank you to the rest of my thesis committee: Prof. Håvard Rue and Prof. Victor Panaretos, for their insightful comments and encouragement.

I would like to thank my colleagues and the friends I met at KAUST for filling my four years’ Ph.D. life with warmth and joy. Finally, I would like to thank my parents for being 24/7 at the phone whatever difficulty I encountered on a different longitude. The march to this milestone would not have been feasible without you all.
## TABLE OF CONTENTS

[Examination Committee Page](#) 2

[Copyright](#) 3

[Abstract](#) 4

[Acknowledgements](#) 6

[List of Symbols](#) 10

[List of Figures](#) 17

[List of Tables](#) 21

1 Introduction 26

1.1 Research motivations 26

1.2 Contributions and outline of the thesis 29

2 Hierarchical-Block Conditioning Approximations for High-Dimensional Multivariate Normal Probabilities 32

2.1 Chapter overview 32

2.2 Building hierarchical representations 33

2.2.1 Generating a hierarchical covariance matrix 33

2.2.2 Hierarchical Cholesky factorization 35

2.3 $d$-dimensional conditioning approximation 37

2.3.1 $d$-dimensional LDL decomposition 37

2.3.2 $d$-dimensional truncated expectations 39

2.3.3 The $d$-dimensional conditioning algorithm 41

2.3.4 Reordered $d$-dimensional conditioning 43

2.4 The hierarchical-block conditioning method 44

2.4.1 The hierarchical-block conditioning algorithm 45

2.4.2 Simulation with a constant covariance matrix 46

2.4.3 Simulation with 1D exponential covariance matrix 49
2.4.4 Simulation with a 2D exponential covariance matrix 

2.5 Block reordering

2.6 Discussion

2.7 Chapter summary

3 Computing High-Dimensional Normal and Student-t Probabilities with Tile-Low-Rank Quasi-Monte Carlo and Block Reordering

3.1 Chapter overview

3.2 SOV for MVN and MVT probabilities

3.2.1 SOV for MVN integrations

3.2.2 SOV for MVT integrations

3.3 Low-rank representation and reordering for MVN and MVT probabilities

3.3.1 Section overview

3.3.2 TLR as a practical representation for MVN and MVT

3.3.3 Reordering schemes and TLR factorizations

3.3.4 Preconditioned TLR Quasi-Monte Carlo algorithms

3.4 Numerical simulations

3.5 Application to stochastic generators

3.5.1 A skew-normal stochastic generator

3.5.2 Estimation with simulated data

3.5.3 Estimation with wind data from Saudi Arabia

3.6 Chapter summary

4 Sum of Kronecker Products Representation and Its Cholesky Factorization for Spatial Covariance Matrices from Large Grids

4.1 Notations and relevant literature

4.2 Nearest sum of Kronecker products

4.2.1 SVD framework

4.2.2 ACA framework

4.3 Quasi-optimal sum of Kronecker products

4.3.1 Indexing and partitioning

4.3.2 Storage costs comparison

4.3.3 On perturbed grids

4.4 Cholesky factorization under SKP

4.4.1 Basic ideas and notations

4.4.2 Algorithm and complexity

4.4.3 Numerical results under expanding domain
## 4.4.4 Correction mechanism for a fixed domain

- Page 110

## 4.5 Simulation of large Gaussian random fields

- Page 111

## 4.6 Chapter summary

- Page 114

## 5 The tlmvnmvt R Package

- Page 116

### 5.1 R and R packages for MVN/MVT probabilities

- Page 116

### 5.2 Package functions

#### 5.2.1 Function interfaces

- Page 118

#### 5.2.2 Computation with dense matrices

- Page 122

#### 5.2.3 Computation with TLR matrices

- Page 125

### 5.3 Performance comparison

- Page 129

### 5.4 Application in finding excursion sets

- Page 136

### 5.5 Chapter summary

- Page 141

## 6 Computation of High-dimensional Probit Gaussian Random Field for Binary Classification

- Page 143

### 6.1 Chapter overview

- Page 143

### 6.2 Properties

- Page 144

### 6.3 MVN probabilities and truncated distributions

- Page 150

### 6.4 Monte Carlo simulations

- Page 153

### 6.5 Chapter summary

- Page 156

## 7 Concluding Remarks

- Page 159

## References

- Page 162
**LIST OF SYMBOLS**

- $n$: the problem size/dimension
- $m$: $m \leq n$, the diagonal block size in the hierarchical representation and the block/tile size in the tile-low-rank representation
- $d$: $d \leq m$, the second conditioning dimension
- $r$: equal to $n/m$, the number of diagonal blocks in the hierarchical representation
- $s$: in Chapter 2, it is equal to $m/d$, denoting the number of diagonal blocks in the $D$ factor of the LDL decomposition. In Chapter 3, it is an integration variable
- $a, b$: $a, b \in \mathbb{R}^n$, the lower and the upper integration limits
- $\mu$: $\mu \in \mathbb{R}^n$, the mean parameter
- $\Sigma$: $\Sigma \in \mathbb{R}^{n \times n}$, the covariance matrix
- $\Phi_n(a, b; \mu, \Sigma)$: an $n$-dimensional multivariate normal probability with the lower integration limits $a$, the upper integration limits $b$, the mean parameter $\mu$, and the covariance matrix $\Sigma$. $\mu$ is occasionally omitted when it is zero. $a$ is occasionally omitted when it is $-\infty$
- $\phi_n(x; \mu, \Sigma)$: the $n$-dimensional multivariate normal probability density at $x$ with the mean parameter $\mu$, and the covariance matrix $\Sigma$. $\mu$ is occasionally omitted when it is zero
- $L$: the lower Cholesky factor or the $L$ factor in the LDL decomposition
- $D$: in Chapter 2, it is the $D$ factor in the LDL decomposition. In Chapter 4, it is the inverse of the lower Cholesky factor $L$. In Chapter 6, it is equal to $\text{diag}(2y - 1)$
in Chapters 2 and 3, \( x \in \mathbb{R}^n \) is the vector of integration variables. In Chapter 4, \( x_i \) is the first coordinate of \( s_i \). In Chapter 5, \( x \) is the realization of \( X(\cdot) \) at a set of spatial locations. In Chapter 6, \( x \) is the predictor vector

in Chapters 2 and 3, \( y = Lx \) is the vector of transformed integration variables. In Chapter 5, \( y \) is a segment of \( x \) added with Gaussian noise. In Chapter 6, \( y \in \{0, 1\}^n \) is the vector of binary responses

in Chapter 4, \( z_i \) is the spatial variable observed at the location \( s_i \)

\( a_i, b_i \) \( a_i, b_i \in \mathbb{R}^m \), the length-\( m \) segment of \( a \) and \( b \), the same notation rule is also applied to \( x \), and \( \mu \)

\( a'_i, b'_i \) \( a'_i, b'_i \in \mathbb{R}^m \), the transformed \( a_i \) and \( b_i \)

\( \tilde{x} \) the counterpart of \( x \) for an \( m \)-dimensional problem, the same notation rule is also applied to \( y, a, b, \) and \( \Sigma \). In Chapter 4, \( \tilde{\Sigma} \) refers to the rearranged \( \Sigma \)

\( \tilde{a}'_i, \tilde{b}'_i \) \( \tilde{a}'_i, \tilde{b}'_i \in \mathbb{R}^d \), the counterpart of \( a'_i \) and \( b'_i \) for an \( m \)-dimensional problem

\( \tilde{D}_i \) \( \tilde{D}_i \in \mathbb{R}^{d \times d} \), the diagonal block of the \( D \) factor in the LDL decomposition of an \( m \)-dimensional covariance matrix

\( \hat{x} \) the counterpart of \( x \) for a \( d \)-dimensional problem, the same notation rule is also applied to \( a, b, \mu, \) and \( \Sigma \)

\( \hat{x}_j \) the \( j \)-th coefficient in \( \hat{x} \)

\( \tilde{X} \) a random vector of length \( d \)

\( \tilde{X}^o_j \) the \( j \)-th random variable in \( \tilde{X} \)

\( \tilde{L}_{i,j} \) \( \tilde{L}_{i,j} \in \mathbb{R}^{d \times d} \), the \( (i, j) \)-th block of the \( L \) factor in the LDL decomposition of an \( m \)-dimensional covariance matrix

\( F^d \) an intermediate value used for the computation of truncated multivariate normal expectations

\( c \) \( c \in \mathbb{R}^d \), an intermediate vector used for the computation of truncated multivariate normal expectations
$\tilde{\mu}_i^1, \tilde{\mu}_i^2 \in \mathbb{R}^{d-1}$, intermediate vectors used for the computation of truncated multivariate normal expectations

$\tilde{\Sigma}_i \in \mathbb{R}^{(d-1) \times (d-1)}$, an intermediate covariance matrix used for the computation of truncated multivariate normal expectations

$B_i$ the $i$-th diagonal block in the hierarchical representation

$U_i, V_i$ in Chapters 2 and 3, $U_i, V_i$ are matrices with $k_i$ columns and $U_i V_i^\top$ is the low-rank representation. In Chapter 4, $U_i \in \mathbb{R}^{m_1 \times n_1}$ and $V_i \in \mathbb{R}^{m_2 \times n_2}$ such that $U_i \otimes V_i \in \mathbb{R}^{n \times n}$

$k, k_i$ in Chapters 2 and 3, $k, k_i$ denote the local rank. In Chapter 4, they denote the number of Kronecker products

$\tau^r$ the row cluster of a low-rank block in the hierarchical representation

$\tau^c$ the column cluster of a low-rank block in the hierarchical representation

$0, 1$ a vector filled with 0 or 1

$e_i$ a vector whose $i$-th coefficient is 1 and other coefficients are zero

$\beta$ the range parameter of the covariance function, controlling the correlation strength

$\hat{\mu}_i$ an estimator for the truncated multivariate normal expectation

$\Phi_m$ one sample estimate of the $m$-dimensional multivariate normal probability

$\Phi_m^j$ the estimate of an $m$-dimensional multivariate normal probability with the $j$-th sample

$M, N$ Monte Carlo sample sizes

$X, Y, Z, Z^*$ in Chapters 2 to 5, they are random vectors of length $n$. In Chapter 6, $X$ is the design matrix

$A, B, C$ in Chapter 3, $A, B, C \in \mathbb{R}^{n \times n}$ are matrices used in the skew-normal model. In Chapter 4, $A, B, C$ are general matrices used in different contexts
\( \mathbf{w} \) 
\( \mathbf{w} \in \mathbb{R}^n \), the integration variables transformed into the unit hypercube

\( \nu \) 
the degrees of freedom

\( T_n(\mathbf{a}, \mathbf{b}, \boldsymbol{\mu}, \boldsymbol{\Sigma}, \nu) \) 
an \( n \)-dimensional multivariate Student-\( t \) probability with the lower integration limits \( \mathbf{a} \), the upper integration limits \( \mathbf{b} \), the location parameter \( \boldsymbol{\mu} \), the covariance matrix \( \boldsymbol{\Sigma} \), and the degrees of freedom \( \nu \).

\( \mu \) 
is occasionally omitted when it is zero

\( \Gamma(\cdot) \) 
the gamma function

\( S \) 
a random variable that has the \( \chi \)-distribution

\( w_0 \) 
an integration variable in \((0, 1)\)

\( \mathbf{h} \) 
the vector connecting the locations of a pair of spatial variables

\( \Sigma_{i_1:i_2,j_1:j_2} \) 
in Chapter 3, it refers to the intersection of the \( i_1 \)-th to \( i_2 \)-th block rows and the \( j_1 \)-th to \( j_2 \)-th block columns, the same notation rule is also applied to \( \mathbf{L} \)

\( \Sigma_{i,j} \) 
the \((i, j)\)-th block of \( \boldsymbol{\Sigma} \), the same notation rule is also applied to \( \mathbf{L} \)

\( \Leftarrow \) 
the switching of coefficients, (block) rows, or (block) columns

\( \odot, \otimes \) 
low-rank matrix multiplication and subtraction that involve truncation

\( \mathbf{U}_{i,j}, \mathbf{V}_{i,j} \) 
\( \mathbf{U}_{i,j}, \mathbf{V}_{i,j} \in \mathbb{R}^{m \times k_{i,j}} \), where \( k_{i,j} \ll m \). \( \mathbf{U}_{i,j} \mathbf{V}_{i,j}^T \) is the low-rank representation of the \((i, j)\)-th block of a tile-low-rank matrix

\( \mathbf{U}, \mathbf{V} \) 
the collection \( \{\mathbf{U}_i\} \) or \( \{\mathbf{U}_{i,j}\} \), the same notation rule is also applied to \( \mathbf{V} \)

\( \xi, \sigma_1, \beta_1, \sigma_2, \beta_2 \) 
in Chapter 3, they are parameters used in the skew-normal model. In Chapter 4, \( \sigma_i \) denotes the \( i \)-th singular value. In Chapter 6, \( \xi(\cdot) \) is the mean structure

\( G(\cdot) \) 
a \( 2n \)-dimensional distribution density function

\( \mathcal{K}(\cdot), \mathcal{K}(\cdot) \) 
covariance functions
\( s, s_i \) in Chapters 2 to 5, they are locations of spatial variables. In Chapter 6, \( s \) is a diagonal matrix used for regularization

\( \tilde{A} \) rearranged \( A \)

\( q \) the number of non-zero singular values

\( \| \cdot \|_F \) Frobenius norm

\( m_1, n_1, m_2, n_2 \) \((m_1, n_1)\) and \((m_2, n_2)\) are the dimensions of the \( U \) and the \( V \) factors, respectively, in the sum-of-Kronecker-product representation.

\( \tilde{U}_i, \tilde{V}_i \) the singular vectors of \( \tilde{\Sigma} \)

\( \text{vec}(\cdot) \) vectorization of a matrix (block)

\( p \) the extra sampling size used in the randomized singular value decomposition

\( \epsilon \) the truncation level

\( s_1, s_2 \) \((s_1, s_2)\) are the dimensions of a 2D grid

\( \Sigma_{\text{ns}} \) a non-stationary covariance matrix

\( U \) a random variable uniformly distributed in a given interval

\( U_i^\Sigma, V_i^\Sigma, k_i^\Sigma \) \( U_i^\Sigma \in \mathbb{R}^{m_1 \times n_1}, V_i^\Sigma \in \mathbb{R}^{m_2 \times n_2} \) such that \( \Sigma \approx \sum_{i=1}^{k_i^\Sigma} U_i^\Sigma \otimes V_i^\Sigma \), the same notation rule is also applied to \( U_i^L, V_i^L, k_i^L \) and \( U_i^D, V_i^D, k_i^D \)

\( S, H \) collections of the matrix multiplications between \( V_i^\Sigma, V_i^L, \) and \( V_i^D \)

\( S, H \) matrices whose columns are the vectorization of the matrices in \( S \) and \( H \), respectively

\( Q^S, R^S \) the \( Q \) and the \( R \) factors in the QR decomposition of \( S \), the same notation rule is also applied to \( Q^H, R^H \)

\( Q \) in Chapter 4, it is a matrix with orthogonal columns. In Chapter 5, it is the precision matrix

\( G \) a matrix whose entries are generated independently from the standard normal distribution

\( k_A^\Sigma \) the numerical rank of the matrix \( A \)

\( C_i^\Sigma, C_i^L \) the coordinate matrices for \( V_i^\Sigma \) and \( V_i^L \) relative to \( Q^S \) and \( Q^H \), respectively

\( k_{\text{max}} \) the maximum of \( k_i^\Sigma, k_i^L, \) and \( k_i^D \)
\( \beta_0 \) the correlation strength parameter for the non-stationary covariance function

\( \beta_i \) the correlation strength coefficient for the \( i \)-th spatial variable

\( \sigma_{ij} \) the \((i,j)\)-th coefficient of \( \Sigma \)

\( \sigma \) the scale parameter for the Matérn covariance kernel

\( \kappa \) the smoothness parameter for the Matérn covariance kernel

\( H_\kappa(\cdot) \) the modified Bessel function of the second kind of order \( \kappa \)

\( \Sigma_\rho \) a constant correlation matrix, where the correlation between two different variables is \( \rho \)

\( \rho \) the correlation, especially for the constant-correlation scenarios

\( U(a, b), \text{Uniform}(a, b) \) the uniform distribution between \( a \) and \( b \)

\( u \) a value used as the variable threshold

\( \alpha \) in Chapter 5, it is a value used as the probability threshold. In Chapter 6, it is the correlation strength parameter for the squared exponential correlation function

\( E_{u,\alpha}^+(X) \) a positive level \( u \) excursion set with probability \( 1-\alpha \)

\( A_u^+(f) \) \( \{s \in \Omega; f(s) > u\} \), where \( f \) is a deterministic function

\( F_u^+(s) \) the positive excursion function

\( \Omega \) a spatial domain

\( \mu_{\text{post}} \) the posterior mean vector

\( \mathcal{Q}_{\text{post}} \) the posterior precision matrix

\( \hat{p}(\alpha) \) the empirical probability of \( \min(\{X(s); s \in E_{0,\alpha}^+(X)\}) > 0 \)

\( X(\cdot), g(\cdot) \) a Gaussian random field

\( \mathbf{x}_i \) the predictor vector for the binary response \( y_i \)

\( \xi, \Delta, \gamma, \Gamma \) parameters for the unified skew-normal distribution, the same notation rule is also applied to \( \xi_{\text{post}}, \Delta_{\text{post}}, \gamma_{\text{post}}, \Gamma_{\text{post}} \)

\( g \) the realization of \( g(\cdot) \)
$\mathcal{M}_i$ the $i$-th model

$d_i, e_i$ intermediate integration variables in the separation-of-variable algorithm

$l_{ij}$ the $(i, j)$-th coefficient of the lower Cholesky factor $L$

$p_i$ $e_i - d_i$

$p_i^{(j)}$ $p_i$ computed with the $j$-th sample
LIST OF FIGURES

2.1 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 ........................................ 34

3.1 Structures of hierarchical (left) and tile-low-rank (right) matrices ................................................................. 68

3.2 Increase in local rank and memory footprint of the Cholesky factor of a hierarchical matrix due to integration-oriented ordering. In each subfigure, the left panel is under Morton order while the right panel is under the block reordering. The diagonal block size is \( \sqrt{n} \). The storage cost for the lower triangular part of the Cholesky factor is marked in each subfigure. The color for each block is coded by the logarithm of the rank-to-block-size ratio, linearly transformed into \((0, 1)\). ...................................................... 70

3.3 Change in local rank and memory footprint of the Cholesky factor of TLR matrix due to integration-oriented ordering. In each subfigure, the left panel is under Morton order while the right panel is under the block reordering. The diagonal block size is \( \sqrt{n} \). The storage cost for the lower triangular part of the Cholesky factor is marked in each subfigure. The color for each block is coded by the logarithm of the rank-to-block-size ratio, linearly transformed into \((0, 1)\). ...................................................... 71

3.4 The relative error for probabilities and log-probabilities. For each \( n \), the three boxplots, from left to right, correspond to the TLR method, the TLR with the block reordering method, and the TLR with the iterative block reordering method. The relative error for log-probabilities is based on 10 estimations of the same replicate. Each boxplot consists of 20 replicates. The covariance matrix, degrees of freedom, and lower integration limits are the same as Table 3.1 \((\beta = 0.1)\) while the upper limits are generated from \(N(4.0, 1.5^2)\). ...................................................... 79
3.5 Boxplots of 30 estimation results. Each estimation is based on one realization of the $n$-dimensional skew-normal model. The red dashed line marks the true value used for generating random vectors from the skew-normal model.

3.6 The heatmap based on one simulation and the functional boxplot of the empirical semivariogram based on 100 simulations. Top to bottom are the fitted skew-normal model and the Gaussian random field. The green curve denotes the empirical semivariogram based on the wind speed data. The distance is computed as the Euclidean distance in the longitudinal and latitudinal coordinate system.

4.1 Illustrations of (a) $y$-major order, (b) $z$-curve order, and (c) Hilbert curve order.

4.2 Efficiency of ordering and partitioning methods. The $x$-axis denotes the number of Kronecker products and the $y$-axis (ndbl) denotes the number of double-precision values needed by $\{U_i\}$ and $\{V_i\}$. Green triangular dots assume the $y$-major order, blue square dots assume the $z$-curve order, and red circular dots assume the Hilbert curve order. The spatial locations are on a $10 \times 32$ grid for (a) and (c) and a $32 \times 32$ grid for (b) and (d), with a unit distance of $1/32$. The covariance functions are stationary (Equation (4.1)) for (a) and (b) and non-stationary (Equation (4.2)) for (c) and (d). All possible partitions are listed but only the ones having the smallest five (six) ndbl are labelled with $(m_2, n_2, \text{ndbl})$. 

84

87

96

97
4.3 (a) An illustration of the hierarchical representation of the covariance matrix used in Figure 4.2(b), built with the standard admissibility condition and truncated at a relative error of $1e-5$. The green blocks are low-rank while the red ones are dense. The number denotes the rank of the block. (b) The ratio between the number of double precision values allocated and the number of spatial locations under the hierarchical representation (blue, top four) and the SKP representation (brown, bottom four). The spatial locations are on a $s_1 \times s_2$ grid, where $s_1 = s_2$ for ‘×’ and ‘△’ shaped dots and $s_1 = \frac{3}{16}s_2$ for ‘+’ and ‘◦’ shaped dots. The spatial correlation matrix is built with Equation (4.1) for ‘◦’ and ‘△’ cases while with Equation (4.2) for ‘+’ and ‘×’ cases. The black dotted line is $ndbl = s_1^2 s_2$, denoting the growth of the block Toeplitz representation.

4.4 (a) The convergence of relative error with the number of Kronecker products. The approximated matrix is built with the exponential (Equation (4.1)) or the non-stationary (Equation (4.2)) correlation function on a regular or a perturbed $32 \times 32$ grid in the unit square.

(b) The number of Kronecker products needed to reach a relative error of $1e-1$, $1e-2$, $1e-3$, $1e-4$, and $1e-5$. The approximated matrix is built with the exponential (Equation (4.1)) correlation function on a $\sqrt{n} \times \sqrt{n}$ perturbed grid in the unit square.

4.5 Time costs for Cholesky factorization under the SKP representation. The exponential and the non-stationary correlation functions from Equation (4.1) and Equation (4.2) are used for building the correlation matrix. The same algorithm is run over 8 threads to show the parallel capacity.
4.6 Mean, variance and empirical semivariogram of spatial variables located on a $1024 \times 1024$ regular grid from a Gaussian random field. Each plot is based on 100 replicates. The simulation parameters are domain types (fixed domain or expanding domain) and correlation structures (exponential correlation function or Whittle correlation function). The unit distance is $1/1024$ for the fixed domain and $1/32$ for the expanding domain. Both correlation functions have a range parameter $\beta = 0.1$. (a) and (b) are the sample mean and sample variance boxplots; (c) and (d) are the functional boxplots of the empirical semivariogram under the Whittle correlation function with fixed and expanding domains. The true underlying variogram is the dashed green line.

5.1 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 5.4. For (b), the problems are simulated in the same way as those in Table 5.5 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.

5.2 (a) The posterior marginal probability for $P\{X(s) > 0 \mid y\}$ and (b) The positive excursion function $F_{\alpha}^+(s)$. The difference between the empirical probability of $\min(\{X(s); s \in E_{\alpha}^+(X)\}) > 0$, denoted by $\hat{p}(\alpha)$, and $1 - \alpha$.

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

6.1 (a) Simulated binary observations, $\{y_i\}_{i=1}^{10,000}$, on the $100 \times 100$ grid in the unit square. (b) Simulated probit Gaussian random field on the $100 \times 100$ grid in the unit square under a zero mean structure and the squared exponential correlation structure. The 100 purple circles denote the unknown locations used for prediction.

6.2 Marginal probabilities and posterior predictions of $\{\text{pr}(y_i = 1)\}_{i=10,000}^{10,100}$ based on 625 observations. The results on the top row are computed with the \textit{tlrmvnmvt} package and those on the bottom row are computed with the \textit{TruncatedNormal} package.

6.3 Marginal probabilities and posterior predictions of $\{\text{pr}(y_i = 1)\}_{i=10,000}^{10,100}$ based on 2,500, 5,625, and 10,000 observations (from top to bottom).
2.1 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. \textit{chol} refers to the Cholesky factorization implemented in R. \textit{dpotrf} is the Cholesky factorization from the Intel LAPACK library. \textit{choldecomp_hmatrix} is the hierarchical Cholesky factorization from the H2Lib library.

2.2 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. The bold font represents the smallest error × time under each \( m \).

2.3 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. The bold font represents the smallest error × time under each combination of \( m \) and \( n \).
2.4 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.

2.5 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 $d = 1, 2$ and 4 are compared. The bold font represents the smallest error time under each $n$.

2.6 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 2.5. The Quasi-Monte Carlo method (mvn), the hierarchical Quasi-Monte Carlo method (hmvn) and the two-level hierarchical-block conditioning method (hccmvn$_d$) with $d = 1, 2$ and 4 are compared. MB in the third column is short for megabytes. The bold font represents the smallest error time under each $n$. 
3.1 Relative error and time of the three algorithms. ‘mvt 1’, ‘mvt 2’, and ‘mvn’ refer to Algorithm 3.2, Algorithm 3.3, and Algorithm 3.1. The covariance matrix is generated from a 2D exponential model, \( \exp(-\|h\|/\beta) \), where \( \beta = 0.1 \), based on \( n \) random points in the unit square. The lower integration limits are fixed at \(-\infty\) and the upper limits are generated from \( N(5.5, 1.25^2) \). \( \nu \) is set as 10 for the ‘mvt’ algorithms. The upper row is the average relative estimation error and the lower row is the average computation time over 20 iterations. All three algorithms have the same sample size of \( N = 10^4 \).

3.2 Performance of the eight methods under strong, medium, and weak correlations. ‘mvn’ and ‘mvt’ are the dense Quasi-Monte Carlo methods, ‘tlrmvn’ and ‘tlrmvt’ are the TLR Quasi-Monte Carlo methods, ‘rtlrmvn’ and ‘rtlrmvt’ add the block reordering preconditioner to ‘tlrmvn’ and ‘tlrmvt’, while ‘rtlrmvn’ and ‘rtlrmvt’ use the iterative block reordering preconditioner. The covariance matrix, integration limits, and the degrees of freedom are generated the same way as in Table 3.1. The upper row is the average relative estimation error and the lower row is the average computation time over 20 replicates.

3.3 Parameter specifications and estimations based on the skew-normal (SN) model and the Gaussian random field (GRF).

3.4 Empirical moments and BIC comparison. SN denotes the skew-normal model and GRF denotes the Gaussian random field. The intervals represent the 5% to 95% quantile intervals based on 100 simulations.

4.1 Approximation of \( \Sigma \) with SVD and ACA frameworks. \( \Sigma \) is the covariance matrix built with an exponential correlation function, \( \exp(-\|x\|/\beta) \), \( \beta = 0.3 \) and \( s_1s_2 \) locations distributed on an \( s_1 \times s_2 \) grid whose unit distance is \( 1/s_2 \). The choice for \((m_1, n_1, m_2, n_2)\) is \((s_1, s_1, s_2, s_2)\) and the spatial indexing is along the second dimension of the grid. Error measures \( \| \Sigma - \sum_{i=1}^k U_i \otimes V_i \|_F / \| \Sigma \|_F \) and \( k \) is the number of Kronecker products. The unit for time is seconds. Within each cell, the SVD framework is on the left while the ACA framework is on the right.

4.2 Major complexity estimates for Algorithm 4.3

4.3 Relative error and time cost for Cholesky factorization of the exponential and non-stationary kernel matrices. Here, \( n \) spatial variables are located on a regular grid in the unit square.
5.1 Performance analysis under the constant correlation structure. The lower integration limits are set at $-\infty$ and the upper integration limits are set at 0.0. The constant correlation $\rho = 0.5$. 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.

5.2 Performance analysis under the constant correlation structure. The lower integration limits are set at $-\infty$ and the upper integration limits are set at $-1.0$. The constant correlation $\rho = 0.5$. The default sample sizes are used for all functions involved with the exception of $n = 16,384$, 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.

5.3 Performance analysis under the constant correlation structure. The lower integration limits are set at $-\infty$ and the upper integration limits are set at $-1.0$. The constant correlation $\rho = 0.8$. 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.

5.4 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 at $-\infty$ and the upper integration limits are set at 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.
5.5 Quartiles of the relative errors of the log-probabilities under Whittle correlation matrices in 900 dimensions. The correlation matrix is generated based on a $30 \times 30$ 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 $U(-5, -1)$ and the upper integration limits are independently generated from $U(1, 5)$. 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.
Chapter 1

Introduction

1.1 Research motivations

The multivariate normal (MVN) probability appears frequently in statistical applications. These applications include but are not limited to the estimation of statistical models, statistical inferences, and the computation of excursion sets (Bolin and Lindgren, 2013). In terms of model estimation, the probability densities of many Gaussian-based models involve the MVN probability, for example, the skew-normal models described in Genton (2004), Arellano-Valle et al. (2006), and Zhang and El-Shaarawi (2010). Zhang and El-Shaarawi (2010) studied the skew-normal model \( X + jY \), where \( X \) and \( Y \) are independent MVN random vectors with predefined covariance structure. However, this distribution has an intractable density function. In Chapter 3, I study a skew-normal model with the representation \( AX + B|Y| \), where \( A \) is a lower Cholesky factor, \( B \) is a covariance matrix, and \( X \) and \( Y \) are standard normal random vectors. Unlike Zhang and El-Shaarawi (2010), this model has a tractable density function and a similar degree of flexibility. Specifically, its probability density involves the MVN probability with a non-trivial covariance structure. I illustrate that the computational methods in this thesis improve the estimation of this model in high dimensions.

One example of MVN probabilities in statistical inference is the posterior inference for the probit regression model discussed in Durante (2019). The key finding in Durante (2019) is that the posterior distribution of the regression coefficients under
the Gaussian prior and the probit response function belongs to the family of unified skew-normal (SUN) distribution (Arellano-Valle and Azzalini, 2006), whose density and moments involve the MVN probability. In fact, the combination of the Gaussian prior and the probit response function leads to the SUN posterior distribution. In Chapter 6, I show that the probit Gaussian random field (GRF), which is more flexible than the probit regression model, also has a SUN posterior distribution. This model is formulated as: \( g(\cdot) \) is a Gaussian random field, \( \{s_i\} \) are the spatial locations, \( y_i \) is the Bernoulli response at \( s_i \), and \( \Pr(y_i = 1) = \Phi(g(s_i)) \). MVN probabilities are again involved in the posterior inference for the probit GRF.

An excursion set (Bolin and Lindgren, 2015), \( E_{u,\alpha} \), is defined as the spatial region with the largest norm among a collection of spatial regions whose probabilities of having the spatial random variables in them all above or below the threshold \( u \) are above \( 1 - \alpha \), where \( \alpha \) is the significance level. Therefore, the probabilities of many different sets of spatial variables being above or below \( u \) need to be computed for finding \( E_{u,\alpha} \). When the spatial variables are jointly normal, these probabilities amount to MVN probabilities and in Chapter 5, I illustrate that compared with Bolin and Lindgren (2015), the efficient estimation of MVN probabilities improves the accuracy of the computed excursion sets of a GRF.

MVN probabilities are generally not analytically tractable and many numerical methods have been proposed for approximating them, among which Genz (1992) introduced the first algorithm based on Monte Carlo sampling that has good efficiency and accuracy. Later-developed numerical methods can be divided into two categories based on their focuses, either accuracy or scalability. Miwa et al. (2003), Craig (2008), Nomura (2014), and Botvich (2017) belong to the first category while Trinh and Genz (2013) and Genton et al. (2018) belong to the second category; see Genz and Bretz (2009) for an overview of the various early approximation methods. In my thesis, I mainly focus on the scalability aspect of these numerical algorithms. Trinh and
Genz (2015) improved the univariate conditioning method, proposed in Mendell and Elston (1974) and discussed with more details in Kamakura (1989), to a bivariate conditioning method, which has higher accuracy than its predecessor. Genton et al. (2018) improved the separation-of-variable (SOV) algorithm proposed in Genz (1992) with a hierarchical representation (Hackbusch, 2015) of the covariance matrix that reduces the cost per Monte Carlo sample from $O(n^2)$ to asymptotically $O(n \log n)$, where $n$ is the problem dimension. Chapter 2 further studies the conditioning methods by extending the conditioning dimension from two to higher and combining the conditioning methods with the hierarchical representation. One aspect where the hierarchical Monte Carlo method in Genton et al. (2018) can be improved is variable reordering, which has been shown to improve the estimation accuracy in the literature (Schervish, 1984; Gibson et al., 1994; Trinh and Genz, 2015). Chapter 3 revisits the Monte Carlo methods, aiming to combine the low-rank representations that include the hierarchical representation, with variable reordering and thus, to benefit from both lower costs per sample and a higher convergence rate for the Monte Carlo sampling.

Existing R (R Core Team, 2019) packages for MVN probabilities are the mvtnorm (Genz et al., 2019; Genz and Bretz, 2009) and the TruncatedNormal (Botev and Belzile, 2019) packages. The mvtnorm package currently only accepts arguments up to $n = 1,000$ dimensions and the TruncatedNormal package, although highly accurate, takes hundreds or even thousands of seconds in few thousand dimensions. The methods from Chapter 3 have their unique advantage of being able to compute MVN probabilities in tens of thousands of dimensions but may not be easily accessible through their C++ interface. Therefore, I develop an R package in Chapter 5, integrating the computational methods from Chapter 3 into the R environment.

As a contribution to statistical computing, not limited to the computation of MVN probabilities, I study a novel matrix compression method for spatial covariance...
matrices from large grids. Common low-rank representations include hierarchical matrices (Hackbusch, 2015) and hierarchical semiseparable matrices (Martinsson, 2011) among others, which are effective for reducing the storage costs of matrices and the complexity of matrix operations. Parallel to the low-rank representations, the Kronecker product provides another perspective for compressing matrices. One example is the nearest Kronecker product (NKP) problem (Van Loan and Pitsianis, 1993; Van Loan, 2000): \[ \arg\min_{U, V} \| \Sigma - U \otimes V \|_F, \] where \( \Sigma \) is the matrix to approximate, which in this case, is a spatial covariance matrix, \( \| \cdot \|_F \) is the Frobenius norm, \( \otimes \) is the Kronecker product, and \( U \) and \( V \) are matrices of given dimensions such that their Kronecker product has the same dimensions as \( \Sigma \). In Chapter 4, I study the efficiency of a more general sum-of-Kronecker-product (SKP) representation, \( \sum_{i=1}^{k} U_i \otimes V_i \), for compressing kernel matrices and compare it with the hierarchical matrix.

1.2 Contributions and outline of the thesis

In Chapter 2, I extend the bivariate conditioning method to an arbitrary \( d \)-dimensional conditioning method and combine it with the hierarchical representation. The hierarchical representation defines a minimum block size \( m \) that is typically larger than \( d \). Therefore, the covariance matrix is first compressed to decompose the \( n \)-dimensional problem into a sequence of smaller \( m \)-dimensional ones. Then, I apply a \( d \)-dimensional conditioning method that further decomposes the \( m \)-dimensional problems into smaller \( d \)-dimensional ones. The resulting two-level hierarchical-block conditioning method requires Monte Carlo simulations to be performed only in \( d \) dimensions, with \( d \ll n \), and restricts the complexity of the algorithm’s major cost to \( O(n \log n) \). An inexpensive block reordering scheme is also introduced 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 hier-
architectural Quasi-Monte Carlo method (Genton et al., 2018) by a factor between 10 and 15.

In Chapter 3, I discuss the drawbacks of the conditioning methods and illustrate that the tile-low-rank (TLR) representation (Weisbecker, 2013; Mary, 2017; Akbudak et al., 2017) is more compatible with the block reordering than the previously-used hierarchical representation. An improved version of the block-reordering scheme is also proposed to further improve the convergence rate. These findings lead to the design of a TLR Quasi-Monte Carlo method with iterative block reordering for MVN probabilities. Simulations up to dimension 65,536 suggest that this method can improve the run time for reaching the same accuracy by one order of magnitude compared with the hierarchical Quasi-Monte Carlo method in Genton et al. (2018) and two orders of magnitude compared with the Monte Carlo method in Genz (1992). The corresponding low-rank Quasi-Monte Carlo methods are also developed for MVT probabilities. It is concluded that the benefits of an improved convergence rate from the iterative block reordering scheme significantly outweigh the storage savings of the hierarchical representation over the TLR representation. As an application of the computational methods, I estimate the parameters of a multivariate skew-normal model in dimensions up to 16,384.

Chapter 4 explores the potential of the SKP representation for approximating spatial covariance matrices and shows that it can reduce the memory footprint by one order of magnitude compared with existing structured representations. The outer product, a special case of the Kronecker product, has been widely applied for constructing low-rank representations, e.g., hierarchically semiseparable representations (Chandrasekaran et al., 2005) and hierarchical representations (Hackbusch, 2015), where the compression comes from the matrix multiplication of two thin matrices. Since the multiplication of two matrices is equivalent to the summation of the outer products between the columns and rows of the corresponding matrices, the SKP
representation can also be viewed as an extension of several other low-rank representations. However, this extension complicates the associated linear algebra operations because the matrix multiplication between two sums of Kronecker products is not as intuitive as the matrix multiplication between four thin matrices. Despite this challenge, Chapter 4 develops the Cholesky factorization algorithm for the SKP representation of a spatial covariance matrix from a large grid that can scale up to one million dimensions.

In Chapter 5, I introduce a new R package, `tlrmvnmvt`, that is published on R CRAN (R Core Team, 2019), where the TLR Quasi-Monte Carlo method with the iterative block reordering, introduced in Chapter 3, is implemented. Additionally, the existing implementation of the Genz (1992) algorithm from the `mvtnorm` package is also improved in the `tlrmvnmvt` package, making the dense-matrix-based method feasible in thousands of dimensions. I include the computation of the excursion sets for a Gaussian random field that was described in Bolin and Lindgren (2015), as an example where the new package improves existing results.

As a more comprehensive application of the `tlrmvnmvt` package, in Chapter 6, I generalize the linear probit model with a Gaussian prior from Durante (2019) to the probit Gaussian random field that considers the spatial correlation between variables and show that its posterior distribution is still tractable. Specifically, I derive the posterior distribution of the probit Gaussian random field as well as the corollaries regarding posterior simulation, posterior prediction, and model selection. With the explicit posterior properties and the `tlrmvnmvt` package, it is shown that model selection and posterior prediction are still feasible when tens of thousands of observations are used.
Chapter 2

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

2.1 Chapter overview

The importance of computing MVN probabilities has been discussed in Chapter 1 as well as in the literature, e.g., Brown and Resnick (1977); Schlather (2002); Arellano-Valle and Azzalini (2006), and Durante (2019). Without further addressing its importance, I introduce the notation of the MVN probability:

$$
\Phi_n(a, b; \mu, \Sigma) = \int_a^b \frac{1}{\sqrt{(2\pi)^n|\Sigma|}} \exp \left\{ -\frac{1}{2}(x - \mu)^\top \Sigma^{-1}(x - \mu) \right\} \, dx, \quad (2.1)
$$

where \(a, b,\) and \(x \in \mathbb{R}^n\) and \(\Sigma \in \mathbb{R}^{n \times n}\). \(a\) and \(b\) are integration limits, defining a rectangular integration region. Without loss of generality, the mean vector \(\mu\) can be assumed 0. \(\Sigma\) is a spatial covariance matrix and \(n\) is the integration dimension. Because most covariance functions in spatial statistics are smooth, the covariance block between two groups of well separated spatial variables can be closely approximated by a low-rank representation. This provides the potential for using the hierarchical matrix (H matrix) (Hackbusch, 2015) to approximate \(\Sigma\), which was used in Genton et al. (2015) to reduce the costs of Monte Carlo sampling from \(O(n^2)\) to asymptotically \(O(n \log n)\). The bivariate conditioning method from Trinh and Genz (2015) that decomposes the \(n\)-dimensional problem, \(\Phi_n(a, b; 0, \Sigma)\), into a sequence of two-dimensional problems, provides another perspective for improving efficiency.
The purpose of this chapter is to generalize the bivariate conditioning method to a $d$-dimensional conditioning method, where $d$ can be an arbitrary integer that is no greater than $n$ and combine it with the hierarchical representation of the covariance matrix. The final algorithm is a two-level hierarchical-block conditioning method. In addition, I develop a reordering scheme with a complexity of $O(m^2n)$ that better maintains the low-rank feature of the off-diagonal blocks than the univariate reordering (Trinh and Genz, 2015), thus applicable to high-dimensional MVN problems. Here, $m$ is the diagonal block size in the hierarchical representation of the covariance matrix. The downside of the 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 ensure that the method is reliable in the contexts where it will be used (Trinh and Genz, 2015).

The remainder of the chapter is organized as follows. In Section 2, I introduce the construction of the hierarchical covariance matrix. In Section 3, I 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, I 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, I introduce a block-wise reordering scheme that significantly improves the accuracy level with negligible run-time cost. I discuss the method further in Section 6 and conclude in Section 7.

### 2.2 Building hierarchical representations

#### 2.2.1 Generating a hierarchical covariance matrix

Two visualizations of hierarchical matrices are shown in Figures 2.1(a) and 2.1(b), where I implicitly use the weak admissibility condition (Hackbusch, 2015) unless
Figure 2.1: 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.

specified otherwise. The diagonal blocks are denoted with $B_1, \ldots, B_r$ while the $r - 1$ off-diagonal blocks are indexed along their intersections with the diagonal and compressed as $\{U_i V_i^T\}$, where $U_i, V_i \in \mathbb{R}^{r_i \times k_i}$ and $r_i$ and $k_i$ are the row cluster (Hackbusch, 2015) and the local rank of $U_i V_i^T$, respectively. In actual construction and storage, the $H$-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 the spatial locations in the row cluster are well separated from those in the column cluster (Hackbusch, 2015), the corresponding off-diagonal block’s singular values typically have a fast decay rate. However, not all off-diagonal blocks have well separated row and column clusters under the weak admissibility condition, which may lead to relatively high local ranks and compromise the efficiency of the hierarchical representation.

A spatial covariance matrix $\Sigma$ is typically built from a set of spatial locations, a spatial covariance function, and an indexing method for the spatial locations. An intuitive way for generating the low-rank representations, $\{U_i V_i^\top\}$, is to truncate based on the singular value decomposition (SVD) of the dense representation of $U_i V_i^\top$. 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, $\|\Sigma_{\pi,\pi} - UV^\top\|/\|\Sigma_{\pi,\pi}\|$, where $\Sigma_{\pi,\pi}$ is a dense matrix block, $UV^\top$ is its low-rank approximation, and $\|\cdot\|$ denotes a valid norm for matrices, or increase the number of terms in the orthonormal polynomial expansions.

2.2.2 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 (Hackbusch, 2015) is much faster in large dimensions (e.g., more than 10,000) even
Table 2.1: 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 Intel LAPACK library. choldecomp_hmatrix is the hierarchical Cholesky factorization from the H2Lib library.

<table>
<thead>
<tr>
<th>n</th>
<th>chol</th>
<th>dpotrf</th>
<th>choldecomp_hmatrix</th>
</tr>
</thead>
<tbody>
<tr>
<td>256</td>
<td>0.004</td>
<td>0.001</td>
<td>0.003</td>
</tr>
<tr>
<td>1,024</td>
<td>0.23</td>
<td>0.02</td>
<td>0.03</td>
</tr>
<tr>
<td>4,096</td>
<td>12.66</td>
<td>0.61</td>
<td>0.46</td>
</tr>
<tr>
<td>16,384</td>
<td>737.0</td>
<td>41.5</td>
<td>9.3</td>
</tr>
</tbody>
</table>

under a strong spatial correlation structure. Table 2.1 compares two implementations of dense Cholesky factorization and one of hierarchical Cholesky factorization under different matrix dimensions, $n$. I selected a 2D exponential covariance structure, $\text{corr}(x, y) = \exp(-||x - y||/\beta)$, where $\beta$ 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, 2019), 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 I focus on high-dimensional problems, I 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, is slower than the dpotrf function because of its extra overhead and less degree of optimization. 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 but this would introduce little error compared with the overall estimation method discussed in the chapter. The resulting hierarchical Cholesky factor has the
hierarchical structure shown in Figures 2.1(c) and 2.1(d).

2.3 \(d\)-dimensional conditioning approximation

Knowing the hierarchical representation of \(L\), the lower Cholesky factor of \(\Sigma\), the \(n\)-dimensional MVN problem can be separated into \(r\) \(m\)-dimensional MVN problems through the change of variable \(Y = LX:\)

\[
\Phi_n(a, b; 0, \Sigma) = \int_{a'}^{-b'} \phi_m(y_1; B_1B_1^\top) \int_{a_2'}^{-b_2'} \phi_m(y_2; B_2B_2^\top) \cdots \\
\int_{a_r'}^{-b_r'} \phi_m(y_r; B_rB_r^\top) \, dy_r \cdots dy_2 \, dy_1, \tag{2.2}
\]

where the formulas of \(a'_i\) and \(b'_i\) are indicated in Equation (2.3). Here, \(\{B_i\}\) are the diagonal blocks in the hierarchical representation of \(L\) and \(m\) is the diagonal block size. For each \(m\)-dimensional problem, the Cholesky factor of its covariance matrix is already given by \(\{B_i\}\). In this section, I extend the bivariate conditioning method of Trinh and Genz (2015) to a \(d\)-dimensional conditioning method and use it to compute these \(m\)-dimensional MVN probabilities and truncated expectations, which becomes the building block for solving the \(n\)-dimensional MVN problem. Here, \(d\) should be a positive integer no greater than \(m\). In Section 2.3.1, I consider only one single \(m\)-dimensional MVN problem, \(\Phi_m(\tilde{a}, \tilde{b}; 0, \tilde{\Sigma})\) and use a tilde to represent the counterparts in \(m\) dimensions.

2.3.1 \(d\)-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 \(\tilde{L}\) matrix
Algorithm 2.1 LDL decomposition

1: procedure LDL(\(\Sigma\))
2: \(\hat{L} \leftarrow I_m, \hat{D} \leftarrow O_m\)
3: for \(i = 1 : d : m - d + 1\) do
4: \(\hat{D}[i : i + d - 1, i : i + d - 1] \leftarrow \hat{\Sigma}[i : i + d - 1, i : i + d - 1]\)
5: \(\hat{L}[i+d : m, i : i+d-1] \leftarrow \hat{\Sigma}[i+d : m, i : i+d-1]^{-1}[i : i+d-1, i : i+d-1]\)
6: \(\hat{\Sigma}[i + d : m, i + d : m] \leftarrow \hat{\Sigma}[i + d : m, i + d : m] - \hat{L}[i + d : m, i : i + d - 1]\)
7: \(\hat{\Sigma}[i + d : m, i + d : m] \leftarrow \hat{\Sigma}[i + d : m, i + d : m] - \hat{\Sigma}[i + d : m, i : i + d - 1] \hat{L}[i+d : m, i : i+d-1]^{-1}\)
8: if \(i + d < m\) then
9: \(\hat{D}[i + d : m, i + d : m] \leftarrow \hat{\Sigma}[i + d : m, i + d : m] - \hat{\Sigma}[i + d : m, i : i + d - 1] \hat{L}[i+d : m, i : i+d-1]^{-1}\)
10: end if
11: end for
12: return \(\hat{L}\) and \(\hat{D}\)
13: end procedure

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 integration variables into uncorrelated blocks of size 2. I 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 \(\hat{L}\) and \(\hat{D}\) can be written as

\[
\hat{L} = \begin{bmatrix}
I_d & O_d & \cdots & O_d \\
\hat{L}_{2,1} & \ddots & \ddots & \vdots \\
\vdots & \ddots & I_d & O_d \\
\hat{L}_{s,1} & \cdots & \hat{L}_{s,s-1} & I_d
\end{bmatrix}, \hat{D} = \begin{bmatrix}
\hat{D}_1 & O_d & \cdots & O_d \\
O_d & \ddots & \ddots & \vdots \\
\vdots & \ddots & \hat{D}_{s-1} & O_d \\
O_d & \cdots & O_d & \hat{D}_s
\end{bmatrix},
\]

where \(s = \frac{m}{d}\), \(O_d\) and \(I_d\) are zero and unit matrices of dimension \(d\), \(\hat{D}_i, i = 1, \ldots, s\), are positive-definite matrices, and \(\hat{L}_{ij}, i = 2, \ldots, s, j = 1, \ldots, s-1\), are matrix blocks.

If \(m\) is not a multiple of \(d\), then the last row of \(\hat{L}\) and \(\hat{D}\) has the dimension of the remainder. The algorithm for LDL decomposition is outlined as Algorithm 2.1. With the change of variable \(\tilde{Y} = \hat{L}\tilde{X}\), I can rewrite an \(m\)-dimensional MVN probability as
the product of \( s \) \( d \)-dimensional MVN probabilities

\[
\Phi_m(\tilde{\mathbf{a}}, \tilde{\mathbf{b}}; 0, \tilde{\Sigma}) = \int_{\tilde{a}_1'}^\tilde{b}_1' \phi_d(\tilde{y}_1; \tilde{D}_1) \int_{\tilde{a}_2'}^\tilde{b}_2' \phi_d(\tilde{y}_2; \tilde{D}_2) \cdots \int_{\tilde{a}_s'}^\tilde{b}_s' \phi_d(\tilde{y}_s; \tilde{D}_s) d\tilde{y}_s \cdots d\tilde{y}_2 d\tilde{y}_1,
\]

(2.3)

where \( \tilde{a}_i' = \tilde{a}_i - \sum_{j=1}^{i-1} \tilde{L}_{ij} \tilde{y}_j, \tilde{b}_i' = \tilde{b}_i - \sum_{j=1}^{i-1} \tilde{L}_{ij} \tilde{y}_j, \tilde{a}_i, \tilde{b}_i, i = 1, \ldots, s, \) are the corresponding segments of \( \tilde{a} \) and \( \tilde{b} \). When \( d = 1 \), Equation (2.3) leads to the univariate conditioning method (Mendell and Elston, 1974; Kamakura, 1989), and when \( d = 2 \), it corresponds to the bivariate conditioning method (Trinh and Genz, 2015), for which both accuracy and complexity increase.

### 2.3.2 \( d \)-dimensional truncated expectations

The integration limits in Equation (2.3) 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 chapter, I generalize the truncated normal expectation formula to \( d \)-dimensions based on the work of Kan and Robotti (2017). The truncated expectation is expressed as

\[
E(\hat{\mathbf{X}}^{e_j}) = \frac{1}{\Phi_d(\tilde{a}, \tilde{b}; \tilde{\mu}, \tilde{\Sigma})} \int_{\tilde{a}}^{\tilde{b}} \hat{x}_j \phi_d(\hat{x}; \tilde{\mu}, \tilde{\Sigma}) d\hat{x},
\]

(2.4)

where \( \hat{\mathbf{X}} \) is a \( d \)-dimensional MVN random vector, \( \phi_d(\cdot) \) denotes the \( d \)-dimensional normal probability density, \( e_j \) is a unit vector of length \( d \) with 1 in the \( j \)th position and 0 elsewhere, and \( \hat{\mathbf{X}}^{e_j} \) denotes the \( j \)th random variable in \( \hat{\mathbf{X}} \). In this section, I add a hat to the notations to signify the counterparts in \( d \)-dimensions. For example, \( \tilde{a} \) and \( \tilde{b} \) are the corresponding lower and upper integration limits of dimension \( d \) while \( \tilde{\mu} \) and \( \tilde{\Sigma} \) denote some general mean vector and covariance matrix of the \( d \)-dimensional
random vector $\hat{X}$. I define

$$F^d_j(\hat{a}, \hat{b}; \hat{\mu}, \hat{\Sigma}) = \int_{\hat{a}}^{\hat{b}} \hat{x}_j \phi_d(\hat{x}; \hat{\mu}, \hat{\Sigma}) \, d\hat{x},$$

which is the numerator of Equation (2.4). A recurrence relation for $F^d_j$ can be derived by differentiating the MVN density function, $\phi_d(\hat{x}; \hat{\mu}, \hat{\Sigma})$, with respect to $\hat{x}$, then multiplying $\hat{x}_j$ on both sides and integrating the random vector, $\hat{x}$, from $\hat{a}$ to $\hat{b}$. The detailed derivation of the recurrence relation can be found in Kan and Robotti (2017). I use parentheses around the subscript to denote the exclusion of the element from the subscript set. Following this relation, I deduce that

$$F^d_j(\hat{a}, \hat{b}; \hat{\mu}, \hat{\Sigma}) = \hat{\mu}_j \Phi_d(\hat{a}, \hat{b}; \hat{\mu}, \hat{\Sigma}) + e_j^\top \hat{\Sigma} c,$$  

(2.5)

where $c$ is a vector with the $l$th coefficient defined as

$$c_l = \phi_1(\hat{a}_l; \hat{\mu}_l, \hat{\sigma}_l^2) \Phi_{d-1}(\hat{a}_{(l)}; \hat{b}_{(l)}; \hat{\mu}_l^1, \hat{\Sigma}_l) - \phi_1(\hat{b}_l; \hat{\mu}_l, \hat{\sigma}_l^2) \Phi_{d-1}(\hat{b}_{(l)}; \hat{a}_{(l)}; \hat{\mu}_l^2, \hat{\Sigma}_l),$$

$$\hat{\mu}_l^1 = \hat{\mu}_{(l)} + \hat{\Sigma}_{(l),l} \frac{\hat{a}_l - \hat{\mu}_l}{\hat{\sigma}_l^2}, \quad \hat{\mu}_l^2 = \hat{\mu}_{(l)} + \hat{\Sigma}_{(l),l} \frac{\hat{b}_l - \hat{\mu}_l}{\hat{\sigma}_l^2}, \quad \hat{\Sigma}_l = \hat{\Sigma}_{(l),(l)} - \frac{1}{\hat{\sigma}_l^2} \hat{\Sigma}_{(l),l} \hat{\Sigma}_{l,(l)}.$$

When $d = 2$, $\hat{\mu} = \mathbf{0}$ and $\hat{\sigma}_l = 1, l = 1, \ldots, d$, the above formula turns into the bivariate version used by Trinh and Genz (2013). The computation of the $d$-dimensional truncated expectation requires the computation of the $(d-1)$-dimensional MVN probability $2d$ 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.
2.3.3 The $d$-dimensional conditioning algorithm

In the $m$-dimensional MVN probability problem represented by Equation (2.3), 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 2.2. A minor change is needed to the algorithm when the remainder of $m$ divided by $d$ is not 0. In the algorithm, $\Phi_d(\tilde{a}', \tilde{b}'; 0, \tilde{D}_i)$ is approximated with Quasi-Monte Carlo simulation and $E[\tilde{Y}']$ is computed from Equations (2.4) and (2.5). Since $(d - 1)$-dimensional MVN probabilities are generated through Quasi-Monte Carlo simulations, the results are unavoidably subject to the Monte Carlo error but this error is on a smaller magnitude compared with that from the conditioning method.

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

\textbf{Algorithm 2.2} $d$-dimensional conditioning algorithm

```
procedure cmvn($\tilde{\Sigma}$, $\tilde{a}$, $\tilde{b}$, $d$)
    $\tilde{y} \leftarrow 0$ and $P \leftarrow 1$
    $[\tilde{L}, \tilde{D}] = \text{LDL}(\tilde{\Sigma})$ as in Algorithm 2.1
    for $i = 1 : s$ do
        $j \leftarrow (i - 1)d$
        $\tilde{g} \leftarrow \tilde{L}[j + 1 : j + d, 1 : j]\tilde{y}[1 : j]$
        $\tilde{a}' \leftarrow \tilde{a}[j + 1 : j + d] - \tilde{g}$
        $\tilde{b}' \leftarrow \tilde{b}[j + 1 : j + d] - \tilde{g}$
        $\tilde{D}_i \leftarrow \tilde{D}[j + 1 : j + d, j + 1 : j + d]$
        $P \leftarrow P \cdot \Phi_d(\tilde{a}', \tilde{b}'; 0, \tilde{D}_i)$
        $\tilde{y}[j + 1 : j + d] \leftarrow E[\tilde{Y}']$
    end for
    return $P$ and $\tilde{y}$
end procedure
```
Table 2.2: 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. The bold font represents the smallest error×time under each $m$.

<table>
<thead>
<tr>
<th>(m, d)</th>
<th>1</th>
<th>2</th>
<th>4</th>
<th>8</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>error</td>
<td>time</td>
<td>error</td>
<td>time</td>
<td>time</td>
</tr>
<tr>
<td>16</td>
<td>2.1%</td>
<td>0.00s</td>
<td>1.9%</td>
<td>0.00s</td>
<td>1.6%</td>
</tr>
<tr>
<td>32</td>
<td>2.5%</td>
<td>0.00s</td>
<td>2.4%</td>
<td>0.01s</td>
<td>2.2%</td>
</tr>
<tr>
<td>64</td>
<td>2.8%</td>
<td>0.00s</td>
<td>2.8%</td>
<td>0.01s</td>
<td>2.7%</td>
</tr>
<tr>
<td>128</td>
<td>2.6%</td>
<td>0.00s</td>
<td>2.6%</td>
<td>0.02s</td>
<td>2.5%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>error</th>
<th>time</th>
<th>error</th>
<th>time</th>
<th>time</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>0.09%</td>
<td>0.00s</td>
<td>0.05%</td>
<td>0.00s</td>
<td>0.01%</td>
</tr>
<tr>
<td>32</td>
<td>0.05%</td>
<td>0.00s</td>
<td>0.02%</td>
<td>0.00s</td>
<td>0.02%</td>
</tr>
<tr>
<td>64</td>
<td>0.03%</td>
<td>0.00s</td>
<td>0.03%</td>
<td>0.00s</td>
<td>0.02%</td>
</tr>
<tr>
<td>128</td>
<td>0.02%</td>
<td>0.00s</td>
<td>0.02%</td>
<td>0.00s</td>
<td>0.02%</td>
</tr>
</tbody>
</table>

Integration limits are set with $\tilde{a} = -\infty$ and $\tilde{b}^{i.i.d.} \sim U(0, m)$, which correspond to the conditions used in Trinh and Genz (2015). Here, i.i.d. is the acronym for independent and identically distributed. The simulation results are presented in the upper half of Table 2.2, where the relative errors are benchmarked against the Quasi-Monte Carlo results with a sample size of $10^4$, which can provide an estimation error below $10^{-4}$.

Since $(d - 1)$-dimensional MVN probabilities need to be evaluated for $2d \times s$ times, I use a progressively increasing sample size for the Quasi-Monte Carlo simulations within the $d$-dimensional conditioning algorithm. Specifically, when $d \leq 4$, 5 × 200 samples are selected, when $4 < d \leq 8$, 10 × 400 samples are selected, and when $d > 8$, 10 × 1000 samples are selected. The estimation error is reduced when $d$ increases
with $m$ remaining unchanged because less correlation information is discarded by conditioning when $d$ becomes larger. In the extreme case when $d = m$, I 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 $m$ but exponentially with $d$.

### 2.3.4 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 2.3 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 (2.3), 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

**Algorithm 2.3** $d$-dimensional conditioning algorithm with univariate reordering

1: procedure $\text{rcmvn}(\Sigma, \tilde{a}, \tilde{b}, d)$
2: $\tilde{y} \leftarrow 0$, $\tilde{C} \leftarrow \Sigma$
3: for $i = 1 : m$ do
4: \quad if $i > 1$ then
5: \quad \quad $\tilde{y}[i - 1] \leftarrow \phi(\tilde{a}^\prime) - \phi(\tilde{b}^\prime)$
6: \quad end if
7: \quad $j \leftarrow \arg \min_{1 \leq j \leq m} \left\{ \Phi \left( \frac{\tilde{b}[j] - \tilde{C}[j, 1 : i - 1] \tilde{y}[1 : i - 1]}{\sqrt{\Sigma[j, j] - \tilde{C}[j, 1 : i - 1] \tilde{C}^\top[j, 1 : i - 1]}} \right) - \Phi \left( \frac{\tilde{a}[j] - \tilde{C}[j, 1 : i - 1] \tilde{y}[1 : i - 1]}{\sqrt{\Sigma[j, j] - \tilde{C}[j, 1 : i - 1] \tilde{C}^\top[j, 1 : i - 1]}} \right) \right\}$
8: \quad $\tilde{\Sigma}[:, (i, j)] \leftarrow \tilde{\Sigma}[:, (j, i)]$, $\tilde{\Sigma}[(i, j), :] \leftarrow \Sigma[(j, i), :]
9: \quad \tilde{C}[:, (i, j)] \leftarrow \tilde{C}[:, (j, i)]$, $\tilde{C}[(i, j), :] \leftarrow C[(j, i), :]
10: \quad \tilde{a}[(i, j)] \leftarrow \tilde{a}[(j, i)]$, $\tilde{b}[(i, j)] \leftarrow \tilde{b}[(j, i)]$
11: \quad $\tilde{C}[i, i] \leftarrow \sqrt{\Sigma[i, i]} - \tilde{C}[i, 1 : i - 1] \tilde{C}^\top[i, 1 : i - 1]$
12: \quad $\tilde{C}[j, i] \leftarrow (\tilde{\Sigma}[j, i] - \tilde{C}[i, 1 : i - 1] \tilde{C}^\top[i, 1 : i - 1]) / \tilde{C}[i, i]$, for $j = i + 1, \ldots, m$
13: \quad $\tilde{a}' \leftarrow (\tilde{\tilde{a}}[i] - C[i, 1 : i - 1] \tilde{y}[1 : i - 1]) / C[i, i]$
14: \quad $\tilde{b}' \leftarrow (\tilde{\tilde{b}}[i] - C[i, 1 : i - 1] \tilde{y}[1 : i - 1]) / C[i, i]$
15: end for
16: $\text{cmvn}((\hat{\Sigma}, \hat{a}, \hat{b}, d)$ as in Algorithm 2.3
17: end procedure
as Algorithm 2.2 in [Trinh and Genz (2015)]. Here, I extract and rephrase the univariate reordering algorithm for completeness. The $d$-dimensional conditioning method with univariate reordering is presented as Algorithm 2.3 here.

The lower half of Table 2.2 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 table. Reordering results in significant improvement in estimation accuracy if I compare the errors with those from the non-reordered conditioning method. I cannot distinguish the conditioning method error from the Monte Carlo error when their estimate difference is below $5 \times 10^{-4}$. The time costs, including those from the univariate reordering, decrease on average compared with the case without the univariate reordering, indicating that the univariate reordering can speed up the $d$-dimensional Monte Carlo simulation performed as part of the conditioning method. The univariate reordering has a complexity of $O(m^3)$ but is still insignificant up to 128-dimensions. Although this complexity increases rapidly as $m$ increases, the size of each diagonal block in the hierarchical covariance matrix, $H$, is usually taken from a range much smaller than the problem size, $n$, which makes univariate reordering a feasible option for solving the small MVN problem presented by each diagonal block.

### 2.4 The hierarchical-block conditioning method

The simulations for the $d$-dimensional conditioning method in Section 2.3 indicate that the algorithm coupled with univariate reordering could be an efficient alternative to Monte-Carlo-based methods. Now I step back to revisit the $n$-dimensional MVN problem. [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. Notice that $k$ is the (average) local rank in the hierarchical representation. As an additional benefit, the memory required for storage is also
minimized when a hierarchical representation is used. In this section, I 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.

### 2.4.1 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 as described in Equation (2.2). Truncated expectations are computed for each diagonal block, $B_iB_i^T$, and used for updating the integration limits to the right as shown in Equation (2.2). The hierarchical-block conditioning algorithm is presented as Algorithm 2.4. It transforms the $n$-dimensional MVN problem into $r$ $m$-dimensional problems. For clarity of presentation, I assume that $n$ and $m$ are both

**Algorithm 2.4** Hierarchical-block conditioning algorithm

```plaintext
1: procedure HCMVN(a, b, Σ, d)
2: x ← 0 and P ← 1
3: [B, UV] ← choldecomp_hmatrix(Σ)
4: for i = 1 : r do
5:   j ← (i - 1)m
6:   if i > 1 then
7:     o_r ← row offset of $U_{i-1}V_{i-1}^T$, o_c ← column offset of $U_{i-1}V_{i-1}^T$
8:     l ← dim($U_{i-1}V_{i-1}^T$)
9:     g ← $U_{i-1}V_{i-1}^T x[o_c + 1 : o_c + l]$
10:    a[o_r + 1 : o_r + l] ← a[o_r + 1 : o_r + l] - g
11:    b[o_r + 1 : o_r + l] ← b[o_r + 1 : o_r + l] - g
12:   end if
13:   a_i ← a[j + 1 : j + m], b_i ← b[j + 1 : j + m]
14:   P ← P : Φ_m(a_i, b_i; 0, B_iB_i^T)
15:   x[j + 1 : j + m] ← $B_i^{-1}E[X_i]$
16: end for
17: return P
18: end procedure
```
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, $U_{i-1}V_{i-1}^T$. The function `choldecomp_hmatrix` implements the hierarchical Cholesky factorization and returns $B$ and $UV^T$ 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 2.3 as a second level of conditioning. In fact, the former method is a special 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 (2.3) although additional error can be introduced by the conditioning technique on the second level. In Sections 2.4.2 and 2.4.3, I compare the two methods for solving the $m$-dimensional problem under simple covariance structures.

### 2.4.2 Simulation with a constant covariance matrix

I first use the constant correlation structure, where the correlations between all pairs of spatial variables are equal, 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, 2013](#)). In this section, I 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 (2.4)
and (2.3) to compute the $m$-dimensional MVN probability and truncated expectations directly. Method 2 employs the $d$-dimensional conditioning method as described in Algorithm 2.2 to compute the $m$-dimensional MVN probability and truncated expectations. Method 3 begins with univariate reordering as described in Algorithm 2.3 and then uses the $d$-dimensional conditioning method. I 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 2.2.

In this experiment, the correlation of the constant covariance matrix is set at 0.7 for a medium correlation strength. The lower integration bound, $a$, is set at $-\infty$ and the upper bound is independently generated from $U(0, n)$, which makes the expectation of the simulated probability roughly 0.7. The upper half of Table 2.3 summarizes the time and relative error of the three methods under the constant covariance structure based on 20 replicates. I use 20 as the sample size instead of 250 as in Table 2.2 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

$$
\Phi_n(-\infty, b; 0, \Sigma_{\text{const}}) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} e^{-\frac{1}{2}x^2} \prod_{i=1}^{n} \Phi_i \left( \frac{b_i + \sqrt{\theta} x}{1 - \theta} \right) \, dx,
$$

where $\theta$ 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, I conclude that the $d$-dimensional conditioning method leads to significant efficiencies.
Table 2.3: 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. The bold font represents the smallest error \(\times\) time under each combination of \(m\) and \(n\).

<table>
<thead>
<tr>
<th></th>
<th>Constant covariance structure</th>
<th></th>
<th>1D exponential covariance structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>(m) (n)</td>
<td>512</td>
<td>1,024</td>
<td>2,048</td>
</tr>
<tr>
<td>M1</td>
<td>11.9%</td>
<td>13.6%</td>
<td>10.9%</td>
</tr>
<tr>
<td></td>
<td>7.6s</td>
<td>14.7s</td>
<td>30.7s</td>
</tr>
<tr>
<td>M2</td>
<td>11.9%</td>
<td>13.6%</td>
<td>10.9%</td>
</tr>
<tr>
<td></td>
<td>0.1s</td>
<td>0.3s</td>
<td>0.7s</td>
</tr>
<tr>
<td>M3</td>
<td>11.9%</td>
<td>13.6%</td>
<td>10.9%</td>
</tr>
<tr>
<td></td>
<td>0.1s</td>
<td>0.3s</td>
<td>0.7s</td>
</tr>
</tbody>
</table>

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, I decompose the complexity of Algorithm 2.4 into three parts and list the complexity for each part in Table 2.4, where \(M(\cdot)\) denotes the complexity of the Quasi-Monte Carlo simulation in the given dimension. Table 2.4 shows that the time efficiency of the \(d\)-dimensional conditioning algorithm mainly comes from reducing 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
Table 2.4: 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.

<table>
<thead>
<tr>
<th></th>
<th>MVN prob</th>
<th>Trunc exp</th>
<th>Upd limits</th>
</tr>
</thead>
<tbody>
<tr>
<td>M1</td>
<td>( \frac{n}{m} M(m) )</td>
<td>( 2nM(m) + O(nm^2) )</td>
<td>( O(mn + kn \log(n/m)) )</td>
</tr>
<tr>
<td>M2</td>
<td>( \frac{n}{d} M(d) + O(m^2n) )</td>
<td>( 2nM(d) + O(nd^2) )</td>
<td>( O(mn + kn \log(n/m)) )</td>
</tr>
<tr>
<td>M3</td>
<td>( \frac{n}{d} M(d) + O(m^2n) )</td>
<td>( 2nM(d) + O(nd^2) )</td>
<td>( O(mn + kn \log(n/m)) )</td>
</tr>
</tbody>
</table>

methods are not greatly affected by the choice of \( m \).

2.4.3 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 Euclidean distance between
the spatial variables \( X_i \) and \( X_j \), and \( \beta \) is the coefficient controlling the correlation
decay rate. After the spatial variables are indexed along their one-dimensional co-
dinates, 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 con-
stant 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, \( \beta \) is set to 10. Here, \( a, b \) and \( d \) are selected in the same
fashion as in Section 2.4.2. Results from the Quasi-Monte Carlo simulation with a
sample size of \( 100 \times 10^4 \) are used as the benchmark, whose error level is below \( 10^{-4} \)
in general. The lower half of Table 2.3 describes the average time and error of the
three methods based on 20 replicates.

I find that the errors of the three methods under the 1D exponential structure
fall into two groups. 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, I 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 2.3, I 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, I argue that Method 3 has the best combination of efficiency and accuracy. Hence from this point on, I consider only Method 3, the two-level hierarchical-block conditioning method preceded by univariate reordering for each diagonal block. I 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, I use a
covariance structure in two dimensions.

### 2.4.4 Simulation with a 2D exponential covariance matrix

The covariance matrix from 1-dimensional geometry typically has low local ranks because of the intuitive indexing method in 1D. However, in two and higher-dimensional spatial domains, a locality-preserving indexing method is needed to restrict the local ranks from growing large. 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, I 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 2.5. 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 2.4.2 and 2.4.3, I found that the two-level hierarchical-block conditioning methods are insensitive to the choice of $m$. I therefore fix $m = 64$ for the 2D covariance structure in this section and examine the effectiveness of the algorithm when the second conditioning dimension, $d$, is set to 1, 2, and 4. To test the sensitivity with respect to the correlation strength, I perform the estimation under $\beta = 0.3, 0.1,$ and 0.03, representing strong, medium, and weak correlation strengths.

Table 2.5 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 2.5 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, $b$, is generated from $U(0,n)$ and the lower bound $a$ is assumed to be $-\infty$ as
Table 2.5: 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 $d = 1, 2$ and $4$ are compared. The bold font represents the smallest error $\times$ time under each $n$.

### $\beta = 0.3$

<table>
<thead>
<tr>
<th>$n$</th>
<th>$k_{\text{min}}, k_{\text{avg}}, k_{\text{max}}$</th>
<th>$\mathcal{H}$(MB)</th>
<th>mvn</th>
<th>hmvn</th>
<th>hccmvn$_1$</th>
<th>hccmvn$_2$</th>
<th>hccmvn$_4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>256</td>
<td>15, 20, 30</td>
<td>0.22</td>
<td>0.4%</td>
<td>0.4%</td>
<td>19.7%</td>
<td>18.5%</td>
<td>16.1%</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0.2s</td>
<td>0.6s</td>
<td>0.0s</td>
<td>0.1s</td>
<td>0.2s</td>
</tr>
<tr>
<td>1,024</td>
<td>14, 24, 60</td>
<td>1.58</td>
<td>0.5%</td>
<td>0.5%</td>
<td>27.6%</td>
<td>26.0%</td>
<td>24.7%</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>2.0s</td>
<td>2.7s</td>
<td>0.1s</td>
<td>0.5s</td>
<td>1.1s</td>
</tr>
<tr>
<td>4,096</td>
<td>14, 26, 116</td>
<td>11.67</td>
<td>0.9%</td>
<td>0.9%</td>
<td>47.8%</td>
<td>46.8%</td>
<td>43.6%</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>30.7s</td>
<td>20.0s</td>
<td>1.2s</td>
<td>2.9s</td>
<td>5.4s</td>
</tr>
<tr>
<td>16,384</td>
<td>13, 26, 222</td>
<td>87.09</td>
<td>0.7%</td>
<td>0.7%</td>
<td>52.8%</td>
<td>50.9%</td>
<td>48.7%</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1161.1s</td>
<td>91.5s</td>
<td>6.9s</td>
<td>14.4s</td>
<td>26.2s</td>
</tr>
</tbody>
</table>

### $\beta = 0.1$

<table>
<thead>
<tr>
<th>$n$</th>
<th>$k_{\text{min}}, k_{\text{avg}}, k_{\text{max}}$</th>
<th>$\mathcal{H}$(MB)</th>
<th>mvn</th>
<th>hmvn</th>
<th>hccmvn$_1$</th>
<th>hccmvn$_2$</th>
<th>hccmvn$_4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>256</td>
<td>13, 18, 28</td>
<td>0.21</td>
<td>0.2%</td>
<td>0.2%</td>
<td>14.5%</td>
<td>12.1%</td>
<td>9.3%</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0.2s</td>
<td>0.6s</td>
<td>0.0s</td>
<td>0.1s</td>
<td>0.2s</td>
</tr>
<tr>
<td>1,024</td>
<td>14, 23, 58</td>
<td>1.55</td>
<td>0.5%</td>
<td>0.5%</td>
<td>31.9%</td>
<td>29.5%</td>
<td>24.8%</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1.8s</td>
<td>2.5s</td>
<td>0.1s</td>
<td>0.4s</td>
<td>1.1s</td>
</tr>
<tr>
<td>4,096</td>
<td>14, 26, 117</td>
<td>11.81</td>
<td>0.6%</td>
<td>0.6%</td>
<td>26.6%</td>
<td>24.5%</td>
<td>22.9%</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>26.6s</td>
<td>11.2s</td>
<td>0.4s</td>
<td>2.1s</td>
<td>4.7s</td>
</tr>
<tr>
<td>16,384</td>
<td>13, 27, 229</td>
<td>89.63</td>
<td>0.8%</td>
<td>0.8%</td>
<td>54.9%</td>
<td>53.5%</td>
<td>51.7%</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1161.8s</td>
<td>53.9s</td>
<td>4.1s</td>
<td>10.4s</td>
<td>20.9s</td>
</tr>
</tbody>
</table>

### $\beta = 0.03$

<table>
<thead>
<tr>
<th>$n$</th>
<th>$k_{\text{min}}, k_{\text{avg}}, k_{\text{max}}$</th>
<th>$\mathcal{H}$(MB)</th>
<th>mvn</th>
<th>hmvn</th>
<th>hccmvn$_1$</th>
<th>hccmvn$_2$</th>
<th>hccmvn$_4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>256</td>
<td>8, 12, 18</td>
<td>0.18</td>
<td>0.0%</td>
<td>0.0%</td>
<td>1.2%</td>
<td>0.8%</td>
<td>0.6%</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0.2s</td>
<td>2.6s</td>
<td>0.0s</td>
<td>0.0s</td>
<td>0.1s</td>
</tr>
<tr>
<td>1,024</td>
<td>12, 20, 50</td>
<td>1.39</td>
<td>0.1%</td>
<td>0.1%</td>
<td>6.2%</td>
<td>5.7%</td>
<td>3.5%</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>2.0s</td>
<td>2.6s</td>
<td>0.1s</td>
<td>0.4s</td>
<td>1.0s</td>
</tr>
<tr>
<td>4,096</td>
<td>13, 25, 112</td>
<td>11.22</td>
<td>0.5%</td>
<td>0.5%</td>
<td>26.8%</td>
<td>24.0%</td>
<td>19.8%</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>26.8s</td>
<td>11.2s</td>
<td>0.5s</td>
<td>2.1s</td>
<td>4.5s</td>
</tr>
<tr>
<td>16,384</td>
<td>14, 27, 227</td>
<td>88.90</td>
<td>0.5%</td>
<td>0.5%</td>
<td>28.3%</td>
<td>26.5%</td>
<td>24.0%</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1166.8s</td>
<td>43.2s</td>
<td>3.4s</td>
<td>9.3s</td>
<td>19.6s</td>
</tr>
</tbody>
</table>

in [Trinh and Genz (2013)](https://doi.org/10.1007/s11041-013-9509-z). I construct the hierarchical covariance matrix, $\mathcal{H}$, prior to Cholesky factorization, and I use an adaptively decreasing tolerance level, starting at $5 \times 10^{-4}$, 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 (with a sample size of 10,000), the hierarchical Quasi-Monte Carlo method (with a sample size of 10,000), and the two-level hierarchical-block conditioning method with $d = 1, 2, \text{ and } 4$. The benchmarks are more accurate hierarchical Quasi-Monte Carlo simulations with increased sample size. Specifically, the sample size is $10^6$ for $n = 1,024$ or 4,096 and $5 \times 10^6$ for $n = 16,384$, which produces estimation error below $5 \times 10^{-4}$ under all correlation strengths and below $1 \times 10^{-4}$ under the weak correlation. The normal Quasi-Monte Carlo simulation is not used as a benchmark because of the huge computational cost in 16,384 dimensions (around 35,000 seconds). The two-level hierarchical-block conditioning method provides a poor estimation, indicating that the conditioning method by itself is not very suitable for high-dimensional MVN probability estimation when the correlation strength exceeds the level corresponding to $\beta = 0.03$ and $n = 256$. It is worth mentioning that the correlation strength is essentially increased when $n$ increases while $\beta$ remains unchanged because the samples are from the unit hypercube. Also the increase of $d$ will reduce the estimation error but is still unable to reach a satisfactory level. The method, when used without a reordering strategy, does not produce sufficiently accurate results. This motivated the development of the reordering strategy described in the next section. Notice that the time difference between this chapter 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.

2.5 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
Algorithm 2.5 Block-wise reordering

1: procedure BLOCKREORDER($G$, $\rho$, $a$, $b$, $m$, $\text{ind}$)
2:     for $i = 1 : m : n - m + 1$ do
3:         $\text{subind} \leftarrow \text{ind}[i : i + m - 1]$
4:         $\text{A} \leftarrow \rho(G, \text{subind})$
5:         $a' \leftarrow a[\text{subind}]$
6:         $b' \leftarrow b[\text{subind}]$
7:         $P = [P, \text{RCMVN}(\text{A}, a', b', 1), P]$ as in Algorithm 2.3
8:     end for
9:     sort($\text{ind}$, $P$, $m$)
10:    return $\text{ind}$
11: end procedure

on the accuracy of the estimation. Univariate reordering can effectively reduce the estimation error for low-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 damage 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, I 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 2.2 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 2.3. For clarity, I use $\rho(G, \text{ind})$ for constructing a correlation matrix based on the correlation function $\rho$, geometry $G$, and indices $\text{ind}$. Here, sort($\text{ind}$, $P$, $m$) stands for rearranging the size-$m$ segments of $\text{ind}$ based on the vector, $P$, in an increasing order. For example, if $\text{ind} = \{1, 2 | 5, 6 | 3, 4 | 7, 8\}$, $P = \{0.3, 0.2, 0.1, 0.4\}$, and $m = 2$, sort($\text{ind}$, $P$, $m$) outputs $\{3, 4 | 5, 6 | 1, 2 | 7, 8\}$. 
Table 2.6: 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 2.5. The Quasi-Monte Carlo method (mvn), the hierarchical Quasi-Monte Carlo method (hmvn) and the two-level hierarchical-block conditioning method (hccmvn) with \( d = 1, 2 \) and 4 are compared. MB in the third column is short for megabytes. The bold font represents the smallest error \( \times \) time under each \( n \).

\[
\begin{array}{cccccccc}
\beta = 0.3 & & & & & & & \\
 n & k_{min}, k_{avg}, k_{max} & H(MB) & \text{mvn} & \text{hmvn} & \text{hccmvn}_1 & \text{hccmvn}_2 & \text{hccmvn}_4 & \text{reorder} \\
256 & 11, 20, 37 & 0.22 & 0.4\% & 0.4\% & 1.3\% & 1.1\% & 1.1\% & 0.00s \\
1,024 & 1, 21, 82 & 1.73 & 0.2s & 0.6s & 0.0s & 0.1s & 1.0s & 5.2s \\
4,096 & 1, 22, 151 & 13.94 & 2.0s & 2.7s & 1.0s & 0.5s & 5.1s & 0.01s \\
16,384 & 1, 24, 287 & 114.14 & 30.7s & 20.0s & 1.2s & 2.9s & 5.4s & 0.03s \\
\end{array}
\]

\[
\begin{array}{cccccccc}
\beta = 0.1 & & & & & & & \\
 n & k_{min}, k_{avg}, k_{max} & H(MB) & \text{mvn} & \text{hmvn} & \text{hccmvn}_1 & \text{hccmvn}_2 & \text{hccmvn}_4 & \text{reorder} \\
256 & 10, 18, 34 & 0.21 & 0.2\% & 0.2\% & 0.7\% & 0.6\% & 0.6\% & 0.00s \\
1,024 & 1, 19, 82 & 1.70 & 0.5s & 0.6s & 0.0s & 0.1s & 0.1s & 0.01s \\
4,096 & 1, 22, 151 & 13.9 & 1.8s & 5.2s & 0.1s & 0.5s & 1.0s & 0.03s \\
16,384 & 1, 25, 287 & 115.75 & 26.8s & 11.2s & 0.5s & 2.1s & 4.4s & 0.14s \\
\end{array}
\]

\[
\begin{array}{cccccccc}
\beta = 0.03 & & & & & & & \\
 n & k_{min}, k_{avg}, k_{max} & H(MB) & \text{mvn} & \text{hmvn} & \text{hccmvn}_1 & \text{hccmvn}_2 & \text{hccmvn}_4 & \text{reorder} \\
256 & 6, 12, 23 & 0.18 & 0.0\% & 0.0\% & 0.0\% & 0.0\% & 0.0\% & 0.00s \\
1,024 & 1, 17, 81 & 1.60 & 0.1s & 0.6s & 0.0s & 0.0s & 0.1s & 0.01s \\
4,096 & 1, 20, 149 & 13.57 & 1.8s & 2.6s & 0.1s & 0.4s & 1.0s & 0.04s \\
16,384 & 1, 25, 287 & 115.71 & 26.8s & 11.2s & 0.6s & 2.0s & 4.4s & 0.12s \\
\end{array}
\]

To examine the increased accuracy from the preceding block reordering, I implemented the 2D exponential experiments the same way as in Section 2.4.2 with an extra reordering layer before the two-level hierarchical-block conditioning algorithm. The benchmarks are computed the same way as in Table 2.3. Table 2.6 compares the reordered conditioning algorithm with the Monte-Carlo-based algorithms and the additional time cost from block reordering is measured separately and shown in the last column. The original Morton’s order is changed by the block reordering. As a result, the ranks and the memory cost of the off-diagonal blocks increase as indi-
cated 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 but is still affected by the correlation strength. For isotropic exponential covariance models, I use their effective ranges as a proxy for the correlation strength. When $\beta$ is equal to 0.3, 0.1, and 0.03 respectively, the effective ranges are 0.90, 0.30, and 0.09, which can provide a reference for the level of estimation error when the geometry is scaled to the unit hypercube. The error is below 3% overall with strong correlation, 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 2.6 indicate similar time performance compared with Table 2.5 because each matrix-vector multiplication is performed only once in the conditioning algorithm as described in Algorithm 2.4. The choice of $d$ has little impact on the estimation accuracy when the preceding block reordering exists due to the fact that the accuracy level is already very high when $d = 1$. Thus, I use $d = 1$ 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, $hccmn_1$, has the best combination of error×time and is eleven times faster than the hierarchical 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.

2.6 Discussion

Despite the broad success of conditioning approximation methods (Kamakura, 1989) for computing MVN probabilities, one problem with them is that additional guidance is needed as to when the approximations can be used reliably and when one needs to
fall back on more expensive Monte Carlo simulation methods that can produce approximations to a specified accuracy. This issue has been recognized in the literature, and the general practice (Connors et al., 2014; Trinh and Genz, 2015) is to perform careful comparisons with MC methods to assess empirically whether the conditioning approximations are appropriate for the problem at hand, as I have done, for example, in Section 2.3.

However it is possible in principle to perform additional computations to obtain indicators for the errors that are introduced by the basic substitutions adopted in these approximations. One potential strategy combines the conditioning approximation substitution with sampling at the block level, to provide a means for producing error estimates. Consider for example the “conditioned” form of an $m$-dimensional MVN problem:

$$
\Phi_m(a, b; 0, \Sigma) = \int_{a_1'}^{b_1'} \phi(x_1; D_1) \int_{a_2'(x_1)}^{b_2'(x_1)} \phi(x_2; D_2) \cdots \int_{a_m'(x_1, \ldots, x_{m-1})}^{b_m'(x_1, \ldots, x_{m-1})} \phi(x_m; D_m) \, dx_m \cdots dx_2 \, dx_1.
$$

The key substitution in the univariate ($d = 1$) conditioning approximation replaces the $x_i$ values in the limits of the integrals by their expected values $\mu_i$, instead of generating them as random samples from a truncated normal and using these values to update all integration limits for every sample, as one would do in a Monte Carlo simulation. The justification for these substitutions is that these $\mu_i$ values are the average values that the $x_i$ would have if sampled from the truncated normal in $[a'_i, b'_i]$.

This suggests a possible way to assess the quality of this substitution: instead of using the expected value $\mu_i$, one may use $N$ samples for every integration and use the mean $\bar{\mu}_i$ of the samples to update the integration limits to produce an approximation $\Phi_{m,j}$ of the MVN integration. Repeating this process for $j = 1 : M$ produces estimates whose variance is an error indicator for $\Phi_m$. A large value of the error indicator implies that the conditioning approximation is not adequate for the problem at hand. One
would either need to increase the conditioning dimension $d$ to increase the coupling that was implicitly ignored, or fall back on Monte Carlo simulations.

For the proposed hierarchical block-conditioning method, the MVN integration was written in Equation (2.2) as:

$$
\Phi_n(a, b; 0, \Sigma) = \int_{a_1}^{b_1'} \phi_m(x_1; B_1B_1^\top) \int_{a_2'(<x_1)}^{b_2'(x_1)} \phi_m(x_2; B_2B_2^\top) \cdots \\
\int_{a_r'(<x_{r-1})}^{b_r'(x_1, \ldots, x_{r-1})} \phi_m(x_r; B_rB_r^\top) \, dx_r \cdots dx_2 \, dx_1.
$$

The key substitution here replaced $x_i$ in the integration limits by their expected values $\mu_i \in \mathbb{R}^m$. In order to assess the effect of these substitutions, one can similarly compute approximations by using $N$ samples for every $m$-dimensional integration $i \in 1 : r$, and update the integration limits on the right of $i$ using the sample expectation. Repeating this process $M$ times provides a variance that is indicative of the error. Large error indicators would require using larger block sizes (for example $2m$) or falling back on a Monte Carlo simulation. I note here that this computation is far less costly than a full Monte Carlo solution, because the integration limits affected by $x_i$ are not updated for every sample but only once for the $N$ samples used in the $i$th integration.

It remains for future work to develop general error indicators and estimates to be used with these conditioning approximations, in order to make them more general-purpose black boxes. In the current state of the practice, the hierarchical-block conditioning method proposed here and the class of approximations to which it belongs are intended to be used when fast computations are desired, but on problems where the user has established that the approximations introduced are of acceptable accuracy for the intended application purpose.

The suitability of the hierarchical conditioning method depends on the covariance structure, the correlation strength, and the accuracy requirement. When the co-
variance structure is random, for example in Table 2.2, the non-hierarchical method works better because the cost of building the hierarchical covariance matrix may not outweigh the savings from matrix manipulations. However, in most circumstances, the covariance function is smooth and adopting the hierarchical representation will greatly improve the efficiency. In cases where the estimation error or a high accuracy level is required, the Monte Carlo method is more suitable than the conditioning method since the latter method in its basic version does not generate an internal estimate of its error. When the MVN probabilities need to be estimated a large number of times, the conditioning method can be based on the correlation strength and the practitioner’s error tolerance level. The estimation error is generally below 1% when the correlation strength is lower than or comparable to the medium correlation strength for the exponential covariance, as discussed in Section 2.5. Unlike the Monte Carlo method, the estimate of the conditioning method is not stochastic given a MVN specific problem, $\Phi_n(a, b; 0, \Sigma)$, assuming the errors from the $d$-dimensional Monte Carlo simulations are ignored while the Monte Carlo method can produce an estimate arbitrarily close to the true value through increasing the sample size.

2.7 Chapter summary

In this chapter, I presented a $d$-dimensional conditioning algorithm as an extension of the bivariate conditioning algorithm from Trinh and Genz (2015) and, based on it, I 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. I 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, I 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. 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 (2.4) and (2.5). Future progress in estimating the truncated expectations of MVN random variables is expected to reduce the run time of this algorithm significantly.
Chapter 3

Computing High-Dimensional Normal and Student-t Probabilities with Tile-Low-Rank Quasi-Monte Carlo and Block Reordering

3.1 Chapter overview

The conditioning methods introduced in Chapter 2 and Trinh and Genz (2015) are more scalable than the Monte-Carlo-based methods of Genz (1992), Botev (2017), and Genton et al. (2018) for estimating MVN probabilities. However, the estimation accuracy of conditioning methods depends on the design of the covariance structure and the integration limits of the MVN problems. Moreover, conditioning methods cannot provide an estimate for the estimation error, which imposes the requirement for apriori assessments to ensure its performance. In contrast, the Monte-Carlo-based methods are more accurate and able to provide an error estimate. Genton et al. (2018) succeeded in reducing the complexity of the Monte-Carlo-based integration of Equation (2.11) through assuming the hierarchical representation (Hackbusch, 2015) for the covariance matrix (also its Cholesky factor). In this chapter, I aim to further improve the Monte Carlo convergence rate while keeping the computation complexity similar to Genton et al. (2018).

With the hierarchical representation, the cost per Monte Carlo sample is reduced from $O(n^2)$ to $O(nk \log n)$, where $k$ is the local rank of the low-rank blocks. Additionally, the Cholesky factorization under the hierarchical representation is shown to be faster than that under the dense representation in Chapter 2. However, the hierarchi-
cal Monte Carlo method from Genton et al. (2018) has the disadvantage of conflicting with variable reordering and hence is unable to benefit fully from an improved convergence rate. Although block reordering was successfully applied in Chapter 2, the local rank $k$ grows after introducing the block reordering. This may have two negative implications on the computation cost. On one hand, it makes the construction of the hierarchical representation more reliant on the SVD that becomes prohibitively expensive in high dimensions. On the other hand, the cost of the Monte-Carlo based methods is more sensitive to the complexity of the matrix-vector multiplication that is $O(nk \log n)$, because this multiplication is applied only once in the conditioning methods but tens of thousands of times in the Monte Carlo methods.

The main contribution of this chapter is finding a middle ground between low-rank representations and variable reordering. I show that the TLR (Weisbecker, 2013; Mary, 2017; Akbudak et al., 2017) structure is more compatible with the block reordering from Chapter 2 than the hierarchical representation. I propose a TLR Monte Carlo method with block reordering for MVN probabilities that has a similar cost per sample to the hierarchical Quasi-Monte Carlo method while benefiting from a higher convergence rate. Additionally, an iterative version of the block reordering from Chapter 2 is proposed that further improves the convergence and produces the TLR Cholesky factor simultaneously. I also develop the corresponding algorithm for MVT probabilities. I include two studies on the maximum likelihood estimation for the skew-normal model based on simulated data and wind data to demonstrate the applicability of my methods in tens of thousands of dimensions.

The remainder of this chapter is structured as follows. In Section 3.2, I introduce the separation-of-variable (SOV) technique (Genz and Bretz, 2009) for MVN and MVT problems and describe dense Quasi-Monte Carlo algorithms for both probabilities. In Section 3.3, I demonstrate the conflict between the hierarchical structure and the block reordering, which leads to the choice of the TLR structure. An improved
version of the block reordering from Chapter 2 is also proposed in Section 3.3. In Section 3.4, I compare the dense Quasi-Monte Carlo method, the TLR Quasi-Monte Carlo method, and the preconditioned TLR Quasi-Monte Carlo methods with a focus on high-dimensional MVN and MVT probabilities. In Section 3.5, I estimate the parameters for simulated high-dimensional skew-normal random fields as well as fit the skew-normal model to a large wind speed dataset of Saudi Arabia as examples where the methods developed in this chapter can be applied. Section 3.6 concludes the chapter.

3.2 SOV for MVN and MVT probabilities

The SOV technique transforms the integration region into the unit hypercube, where efficient Quasi-Monte Carlo rules can improve the convergence rate. The SOV for MVN probabilities is based on the Cholesky factor of the covariance matrix (Genz, 1992) and this naturally leads to the second form of SOV for MVT probabilities (Genz and Bretz, 2002). The two forms of SOV for MVT probabilities have been derived in Genz (1992) and Genz and Bretz (2002). In this chapter, I summarize the derivations for completeness.

3.2.1 SOV for MVN integrations

The MVN probability has been defined in Equation (2.1) with the assumption of \( \mu = 0 \). I leave out the mean parameter and denote the \( n \)-dimensional MVN probability with \( \Phi_n(a, b; \Sigma) \) in this chapter. Here in this chapter, I use \( L \) to represent the lower Cholesky factor of \( \Sigma = LL^T \) and \( l_{ij} \) to represent the element on the \( i \)-th row and \( j \)-th column of \( L \). Following the procedure in Genz (1992), we can transform \( \Phi_n(a, b; \Sigma) \) into:

\[
\Phi_n(a, b; \Sigma) = (e_1 - d_1) \int_0^1 (e_2 - d_2) \cdots \int_0^1 (e_n - d_n) \int_0^1 dw, \quad (3.1)
\]
where \( d_i = \Phi\left(\sum_{j=1}^{i-1} l_{ij} y_j / l_{ii}\right) \), \( e_i = \Phi\left(\sum_{j=1}^{i-1} l_{ij} y_j / l_{ii}\right) \), \( y_j = \Phi^{-1}\{d_j + w_j(e_j - d_j)\} \), and \( \Phi(\cdot) \) is the cumulative distribution function of the standard normal distribution.

The integration region is transformed into \([0, 1]^n\) and efficient sampling rules can be applied to simulate \( w \), although the integrand is difficult to compute in parallel because \( d_i \) and \( e_i \) depend on \( \{y_j, j = 1, \ldots, i-1\} \) while \( y_i \) depends on \( d_i \) and \( e_i \). Only univariate standard normal probabilities and quantile functions are needed, which can be readily obtained with the high efficiency of scientific computing libraries, for example, the Intel MKL. The Cholesky factorization has a complexity of \( O(n^3) \) but modern CPUs and libraries have been developed to handle matrices with more than 10,000 dimensions with ease.

I use ‘mvn’ to denote the integrand function of Equation (3.1), whose pseudocode was originally proposed in [Genz (1992)]. Because the ‘mvn’ function is also the subroutine in other functions of this chapter, I summarize it here in Algorithm 3.1. The algorithm returns \( P \), the probability estimate from one Monte Carlo sample and \( y \) whose coefficients are described in Equation (3.1). Keeping \( a, b, \) and \( L \) unchanged,

**Algorithm 3.1 Quasi-Monte Carlo for MVN probabilities**

1: \( \text{mvn}(L, a, b, w) \)
2: \( n \leftarrow \text{dim}(L) \), \( s \leftarrow 0 \), \( y \leftarrow 0 \), and \( P \leftarrow 1 \)
3: for \( i = 1 : n \) do
4: \( \text{if} \ i > 1 \ \text{then} \)
5: \( s \leftarrow L(i, 1 : i - 1) y(1 : i - 1) \)
6: \( \text{end if} \)
7: \( a' \leftarrow \frac{a}{C_{ii}}, \) and \( b' \leftarrow \frac{b}{C_{ii}} \)
8: \( y_i \leftarrow \Phi^{-1}\left[w_i\{\Phi(b') - \Phi(a')\}\right] \)
9: \( P \leftarrow P \cdot \{\Phi(b') - \Phi(a')\} \)
10: end for
11: return \( P \) and \( y \)

the mean and standard deviation of the outputs \( P \) from a set of well designed \( w \), usually conforming to a Quasi-Monte Carlo rule, form the probability and error estimates. In actual implementation, I employ the Richtmyer Quasi-Monte Carlo rule
Richtmyer, 1951), where the batch number is usually much smaller than the batch size.

3.2.2 SOV for MVT integrations

I denote an $n$-dimensional MVT probability with $T_n(a, b; \mu, \Sigma, \nu)$, where $\nu$ is the degrees of freedom. Here, $\mu$ is the mean vector and $\Sigma$ is the scale matrix. To simplify the notations, $\mu$ is again assumed to be $0$ and left out of the notation of $T_n(\cdot)$. There are two common equivalent definitions for $T_n(\cdot)$, of which the first one is:

$$T_n(a, b; \Sigma, \nu) = \frac{\Gamma\left(\frac{\nu+n}{2}\right)}{\Gamma\left(\frac{\nu}{2}\right)\sqrt{|\Sigma|}^{(\nu)\pi^{n/2}}} \int_{a_1}^{b_1} \cdots \int_{a_n}^{b_n} \left(1 + \frac{x^T \Sigma^{-1} x}{\nu}\right)^{-\frac{\nu+n}{2}} d\mathbf{x}, \quad (3.2)$$

where $\Gamma(\cdot)$ is the gamma function. Based on this definition, Genz and Bretz (1999) transformed the integration into the $n$-dimensional hypercube, where the inner integration limits depend on the outer integration variables. However, the integration needs to compute the cumulative distribution function and the quantile function of the univariate Student-$t$ distribution at each integration variable. A second equivalent form defines $T_n$ as a scale mixture of the MVN probability, specifically:

$$T_n(a, b; \Sigma, \nu) = \frac{2^{1-\frac{\nu}{2}}}{\Gamma\left(\frac{\nu}{2}\right)} \int_0^\infty s^{\nu-1} e^{-s^2/2} \Phi_n \left( \frac{Sa}{\sqrt{\nu}}, \frac{Sb}{\sqrt{\nu}}; \Sigma \right) ds, \quad (3.3)$$

$$= E \left[ \Phi_n \left( \frac{Sa}{\sqrt{\nu}}, \frac{Sb}{\sqrt{\nu}}; \Sigma \right) \right]. \quad (3.4)$$

The density of a $\chi$-distribution random variable, $S$, with degrees of freedom $\nu$, is

$$\frac{2^{1-\frac{\nu}{2}}}{\Gamma\left(\frac{\nu}{2}\right)} s^{\nu-1} e^{-s^2/2}, \quad s > 0.$$ Thus, $T_n(a, b; \Sigma, \nu)$ can be also written as Equation (3.3).

The integrand boils down to the MVN probability discussed in the previous section. Hence, we can apply a Quasi-Monte Carlo rule in the $(n+1)$-dimensional hypercube to approximate this expectation, where only the cdf and the quantile function of the univariate standard normal distribution are involved. It is worth pointing out that
considering $T_n$ as a one-dimensional integration of $\Phi_n$ and applying quadrature is much more expensive than integrating directly in $(n + 1)$ dimensions.

I describe the integrand functions based on the two SOV schemes in Algorithm 3.2 and Algorithm 3.3, corresponding to Equation (3.2) and Equation (3.3), respectively.

Algorithm 3.2 calls the univariate Student-t cumulative distribution function and

Algorithm 3.3 Quasi-Monte Carlo for MVT probabilities based on Equation (3.3)

1. $\text{mvt}(L, a, b, \nu, w)$
2. $n \leftarrow \dim(L)$, $s \leftarrow 0$, $ssq \leftarrow 0$, $y \leftarrow 0$, and $P \leftarrow 1$
3. for $i = 1 : n$ do
4. 
5. 
6. 
7. 
8. 
9. 
10. 
11. 
12. return $P$

Algorithm 3.3 Quasi-Monte Carlo for MVT probabilities based on Equation (3.3)

1. $\text{mvt}(L, a, b, \nu, w_0, w)$
2. $a' \leftarrow \frac{\chi_{(w_0)}^{-1}(a)}{\sqrt{\nu}}$, $b' \leftarrow \frac{\chi_{(w_0)}^{-1}(w_0)}{\sqrt{\nu}}$
3. return $\text{mvn}(L, a', b', w)$

the quantile function with an increasing value of degrees of freedom at each iteration whereas Algorithm 3.3 relies on $(w_0, w)$ from an $(n + 1)$-dimensional Quasi-Monte Carlo rule and calls the ‘mvn’ kernel from Algorithm 3.1 with the scaled integration limits. I use single-quoted ‘mvn’ and ‘mvt’ to denote the corresponding algorithms to distinguish them from the uppercase MVN and MVT used for multivariate normal and Student-t in this chapter.

A numerical comparison between Algorithm 3.2 and Algorithm 3.3 is shown in Table 3.1. The counterpart for MVN probabilities (Algorithm 3.1) is included as a benchmark. The table indicates that the first definition as in Equation (3.2) leads to
Table 3.1: Relative error and time of the three algorithms. ‘mvt 1’, ‘mvt 2’, and ‘mvn’ refer to Algorithm 3.2, Algorithm 3.3, and Algorithm 3.1. The covariance matrix is generated from a 2D exponential model, $\exp(-\|\mathbf{h}\|/\beta)$, where $\beta = 0.1$, based on $n$ random points in the unit square. The lower integration limits are fixed at $-\infty$ and the upper limits are generated from $N(5.5, 1.25^2)$. $\nu$ is set as 10 for the ‘mvt’ algorithms. The upper row is the average relative estimation error and the lower row is the average computation time over 20 iterations. All three algorithms have the same sample size of $N = 10^4$.

<table>
<thead>
<tr>
<th>$n$</th>
<th>16</th>
<th>64</th>
<th>256</th>
<th>1,024</th>
<th>4,096</th>
</tr>
</thead>
<tbody>
<tr>
<td>mvt 1</td>
<td>0.0%</td>
<td>0.2%</td>
<td>0.7%</td>
<td>1.4%</td>
<td>4.2%</td>
</tr>
<tr>
<td></td>
<td>0.7s</td>
<td>3.0s</td>
<td>13.3s</td>
<td>58.7s</td>
<td>283.1s</td>
</tr>
<tr>
<td>mvt 2</td>
<td>0.0%</td>
<td>0.0%</td>
<td>0.2%</td>
<td>0.4%</td>
<td>1.3%</td>
</tr>
<tr>
<td></td>
<td>0.0s</td>
<td>0.0s</td>
<td>0.2s</td>
<td>2.0s</td>
<td>40.8s</td>
</tr>
<tr>
<td>mvn</td>
<td>0.0%</td>
<td>0.0%</td>
<td>0.1%</td>
<td>0.4%</td>
<td>1.2%</td>
</tr>
<tr>
<td></td>
<td>0.0s</td>
<td>0.0s</td>
<td>0.2s</td>
<td>2.0s</td>
<td>40.1s</td>
</tr>
</tbody>
</table>

an implementation slower by one order of magnitude. Additionally, the convergence rate from Equation (3.2) is also worse than that from Equation (3.3). Although the univariate Student-t cumulative distribution function and quantile function are computed the same number of times as their standard normal counterparts, their computation takes much more time and probably produces lower accuracy due to the lack of optimized libraries. Due to its performance advantage, I refer to Algorithm 3.3 as the ‘mvt’ algorithm from this point on. It has negligible marginal complexity over the ‘mvn’ algorithm since the only additional step is scaling the integration limits.

### 3.3 Low-rank representation and reordering for MVN and MVT probabilities

#### 3.3.1 Section overview

More flexible than quadrature methods, Monte Carlo (MC) procedures provide several viable options for computing MVN and MVT probabilities. The cost of these computations depends on the product of the number of MC samples, $N$, needed to
achieve a desired accuracy and the cost per MC sample. Under the standard dense representation of covariance, the computational complexity for each sample is $O(n^2)$ as shown in Algorithm 3.1 and Algorithm 3.3. Genton et al. (2018) proposed using the hierarchical representation for the Cholesky factor, illustrated in Figure 3.1, which reduced the complexity per sample to $O(kn \log n)$, where $k$ is a nominal local rank of the matrix blocks. Using nested bases in the hierarchical representation (Boukaram et al., 2019), it is possible to reduce this cost further to an asymptotically optimal $O(kn)$.

Small local ranks $k$ in the hierarchical representation depend on the separability of the underlying geometry and are directly affected by the ordering of the underlying point set. When the row cluster and the column cluster of an off-diagonal matrix block are well separated spatially, the ranks of these blocks tend to be rather small, growing very weakly with the problem dimension $n$. When the geometry is a subset of $\mathbb{R}^2$ or $\mathbb{R}^3$, a space-filling curve, or a spatial partitioning method in combination with a space-filling curve, may be used for indexing to keep the index distances reasonably consistent with the spatial distances. The point set is then further divided into blocks (clusters) according to these indices to build the hierarchical representation.
The optimal ordering for reducing the cost per Monte Carlo sample however is unfortunately generally not the optimal ordering for reducing the total number of samples \(N\). A proper reordering scheme that takes into account the widths of the integration limits of the MVN and MVT probabilities can have a substantial effect on reducing the variance of the estimates, making the numerical methods far more effective relative to a default ordering (Schervish, 1984; Genz and Bretz, 2009). Trinh and Genz (2015) analyzed ordering heuristics and found that a univariate reordering scheme, that sorts the variables so that the outermost integration variables have the smallest expected values, significantly increased the estimation accuracy. This heuristic was more effective overall than more expensive bivariate reordering schemes that might further reduce the number of samples needed. In Chapter 4, a block reordering scheme was proposed with the hierarchical matrix representations used in high dimensions. Specifically, within each diagonal block \(B_i\), univariate reordering was applied and the blocks were reordered based on their estimated probabilities using this univariate reordering scheme.

The important point here is that these reordering schemes shuffle the variables based on their integration limits to achieve better convergence for the integration, measured by the number of samples needed. They produce different orders from the geometry-oriented ordering obtained by spatial partitioning methods or space-filling curves. The reordering increases the local ranks \(k\) of the hierarchical representation making the per-sample computation more expensive.

In this chapter, I seek a better middle ground between the geometry-oriented and the integration-oriented orderings by combining a block reordering scheme with the tile-low-rank representation of covariance illustrated in Figure 3.11. I also introduce the TLR versions of the Quasi-Monte Carlo algorithms for computing MVN and MVT probabilities.
Figure 3.2: Increase in local rank and memory footprint of the Cholesky factor of a hierarchical matrix due to integration-oriented ordering. In each subfigure, the left panel is under Morton order while the right panel is under the block reordering. The diagonal block size is $\sqrt{n}$. The storage cost for the lower triangular part of the Cholesky factor is marked in each subfigure. The color for each block is coded by the logarithm of the rank-to-block-size ratio, linearly transformed into $(0, 1)$.

3.3.2 TLR as a practical representation for MVN and MVT

To illustrate the rank increase under the hierarchical representation due to integration-oriented orderings, I consider an MVN problem, where the integration limits are randomly generated and independent from the geometry. I use Morton order (Morton, 1966) as a geometry-oriented ordering to compare with the integration-oriented block ordering scheme proposed in Chapter 2 with respect to constructing the hierarchical Cholesky factor. Figure 3.2 highlights the sharp increase in local ranks, represented by the storage costs when Morton order is substituted with the block reordering. The initial covariance matrices are built with the 2D exponential covariance model, $\exp(-\|\mathbf{h}\|/\beta)$ based on a perturbed grid in the unit square as described in Section 3.5.2. Here, $\beta = 0.3$ and $\mathbf{h}$ is the vector connecting the locations of a pair of spatial variables. Similar to Genton et al. (2018) and those in Chapter 4, the hierarchical matrix is built under the weak admissibility condition but here, the rank of each block is defined as the number of singular values above an absolute threshold of $10^{-2}$.

Every off-diagonal block touching the main diagonal is represented as $\mathbf{U}_i\mathbf{V}_i^\top$, where $\mathbf{U}_i$ and $\mathbf{V}_i$ are thin matrices. This representation is beneficial only if the ranks
of the off-diagonal blocks are small, which typically originates from well-defined separability (Hackbusch, 2015). However, the two sets of spatial locations corresponding to the rows and columns of large off-diagonal blocks become less separable under the block reordering compared with Morton order because the former is equivalent to block shuffling when each block’s integration limits are independent and identically distributed. The changes in the storage cost and the rank-to-block-size ratio in Figure 3.2 is a numerical proof of the incompatibility between the hierarchical structure and the integration-oriented orderings.

I now consider the rank impact on the TLR structure from the same change in the ordering scheme. Figure 3.3 shows that the average rank of the off-diagonal blocks in the TLR structure even decreases when applying the block reordering scheme, which shuffles the diagonal blocks. This numerical result may appear odd but an explanation is given from two perspectives.

Firstly, the collection of local ranks in the TLR covariance matrix does not change. The block reordering changes the order of blocks but does not switch variables from different blocks and hence, the new TLR covariance matrix merely rearranges the previous off-diagonal blocks. Secondly, the magnitude of the Schur complement decreases faster along with the block column index under the block reordering. Intuitively, the
Schur complement at block column \( i \),
\[
\Sigma_{i,r,i} - \Sigma_{i,r,1:i-1} \Sigma_{1:i-1,1:i-1}^{-1} \Sigma_{i,r,1:i-1}^\top,
\]

is the conditional covariance matrix for blocks \( i \) to \( r \) given blocks 1 to \( i - 1 \). I argue that for most spatial correlation matrices, the Schur complement given a clustered set of spatial locations is bigger in magnitude than that given a set of scattered locations of the same size. This is treated as a heuristic without proof since it is not the focus of the chapter. One measure for magnitude is the Frobenius norm. The Frobenius norm of a Cholesky factor is equal to the square root of the trace of the corresponding covariance matrix. Since \( L_{i,r,i} \) is the Cholesky factor of the corresponding Schur complement, the magnitude of the Cholesky factor also decreases faster along with the block column index under the block reordering. Overall, these two aspects lead to a smaller average local rank for the block reordering. It is worth noting that the absolute truncation usually demonstrates better efficiency than the relative one for correlation matrices because all coefficients belong to \([-1, 1]\] and the magnitude ratio between different blocks varies significantly.

I thus make the ansatz that the TLR structure is compatible and even creates a synergy with the block reordering. There are also two practical benefits compared with the hierarchical structure with the weak admissibility. First, fast approximation algorithms, for example, the adaptive cross approximation (ACA) (Bebendorf, 2011), can be more reliably applied under TLR due to its lower ranks. Second, the regularity of the flat structure of TLR benefits more directly from modern hardware architectures. The reordering for MVT problems shares the same principle. Specifically, because the expectation of \( S \) from Equation (3.3) is \( \sqrt{2} \Gamma((\nu + 1)/2)/\Gamma(\nu/2) \), converging to \( \sqrt{\nu} \) quickly as \( \nu \) increases, Genz and Bretz (2002) proposed substituting \( S \) with \( \sqrt{\nu} \) and the reordering becomes exactly the same as that for the MVN probability.
3.3.3 Reordering schemes and TLR factorizations

The block reordering scheme has been proposed in Chapter 2 and shown to improve the estimation accuracy of the conditioning method with lower complexity than the univariate or bivariate reordering scheme introduced in [Trinh and Genz (2015)]. In this chapter, I introduce a version of the block reordering scheme computed during Cholesky factorization. The block reordering can be also viewed as the block version of the univariate reordering scheme in [Trinh and Genz (2015)].

Algorithm 2.5 describes the block reordering scheme proposed in Chapter 2 while Algorithm 3.4 is the iterative version that produces the Cholesky factor. I use $\Sigma_{ij}$ to represent the $(i, j)$-th size-$m$ block of $\Sigma$. Similar notations are also used for $a$ and $b$. The symbol $\leftrightarrow$ indicates the switching of coefficients, rows, or columns. Variables can be overwritten by themselves after computations for computational benefits. When $i \neq j$, $\Sigma_{ij}$ is stored in the low-rank format. The blue lines in Algorithm 3.4 mark the matrix operations that are also in the TLR Cholesky factorization ([Versbeeken, 2013; Mary, 2017; Akbudak et al., 2017]). If I ignore the cost for steps 5 and 9, the complexity of Algorithm 3.4 is the same as the TLR Cholesky factorization. Although the complexity for accurately computing $\Phi_m$ and the truncated expectations is high, the univariate conditioning method ([Trinh and Genz, 2013]), with a complexity of $O(m^3)$, can provide an estimate for both that is indicative enough. Algorithm 2.5 ignores the correlation between the $m$-dimensional blocks and also uses the univariate conditioning method for approximating $\Phi_m$. Therefore, the block reordering scheme has a total complexity of $O(nm^2)$ but requires an additional Cholesky factorization while the block reordering has additional complexity of $O(n^2m)$ over the TLR Cholesky factorization but produces the Cholesky factor simultaneously.

The truncated product and subtraction operations, $\odot$ and $\ominus$, indicate the corresponding matrix operations which involve truncation to smaller ranks to maintain required accuracy. Here, $\Sigma_{i,j} \odot \Sigma_{ji,j}^T$ and $\Sigma_{i,j} \ominus \Sigma_{ji,j}^{-T}$ have complexities of $O(mk^2)$.
Algorithm 3.4 Block reordering during Cholesky factorization

1: rbodr($\Sigma, a, b, m$)
2: $r = n/m$
3: for $j = 1 : r$ do
4:   for $l = j : r$ do
5:     $p[l] \approx \Phi_m(a_l, b_l; \Sigma_{l,l})$
6:   end for
7: $\tilde{j} = \text{argmin}_l(p[l]), l = j, \ldots, r$
8: Block-wise $\Sigma[j = \tilde{j}, j = \tilde{j}]$, $a[j = \tilde{j}], b[j = \tilde{j}]$
9: $y_j \approx E_m[Y | Y \sim N_m(0, \Sigma_{j,j}), Y \in (a_j, b_j)]$
10: $\Sigma_{j,j} = \text{Cholesky}(\Sigma_{j,j})$
11: for $i = j + 1 : r$ do
12:   $\Sigma_{i,j} = \Sigma_{i,j} \odot \Sigma_{j,j}^\top$
13:   $a_i = a_i - \Sigma_{i,j} \odot y_j$, $b_i = b_i - \Sigma_{i,j} \odot y_j$
14: end for
15: for $j_1 = j + 1 : r$ do
16:   for $i_1 = j + 1 : r$ do
17:     $\Sigma_{i_1,j_1} = \Sigma_{i_1,j_1} \odot \Sigma_{i_1,j} \odot \Sigma_{j_1,j}^\top$
18:   end for
19: end for
20: end for

and $O(m^2k)$ respectively, where $m$ is the tile size and $k$ is the local rank. The $\odot$ operation uses ACA truncated at an absolute tolerance to keep the result low-rank. For the studies in Section 3.4 and Section 3.5, I set the tolerance to $10^{-5}$. Prior to the TLR Cholesky factorization, I construct the TLR covariance matrix with ACA given the covariance kernel, the underlying geometry and the indices of variables. Therefore, the total memory needed for computing MVN and MVT probabilities is $O(kn^2/m)$.

3.3.4 Preconditioned TLR Quasi-Monte Carlo algorithms

Algorithm 3.5 and Algorithm 3.6 describe the TLR versions of the ‘mvn’ and ‘mvt’ algorithms. To distinguish them from the dense ‘mvn’ and ‘mvt’ algorithms, I expand the storage structure of $L$, the TLR Cholesky factor, as the interface of the TLR algorithms. The definitions of $B_i$, $U_{i,j}$, and $V_{i,j}$ are shown in Figure 3.1.
Similar to Algorithm 3.4, I use subscripts to represent the size-$m$ segment of $a$, $b$, $y$, and $w$. The two algorithms compute the integrand given one sample $w$ in the $n$-dimensional unit hypercube. In the implementation, the Richtmyer rule ([Richtmyer, 1951]), used in Chapter 2, is employed for choosing $w$. Here, ‘tlrmvn’ is called by ‘tlrmvt’, where the additional inputs, $\nu$ and $w_0$, bear the same meaning as those in Algorithm 3.3. The TLR structure reduces dense matrix-vector multiplication to low rank matrix-vector multiplication when factoring the correlation between blocks into the integration limits. The TLR structure reduces the complexity of matrix-vector multiplication, hence the cost per MC sample, at the step of block updating the integration limits (Lines 6 and 7 in Algorithm 3.5). The TLR Quasi-Monte Carlo is a variant of the SOV algorithm from Genz (1992) that belongs to the same category as the hierarchical Quasi-Monte Carlo (Genton et al, 2018). Algorithm 3.5 and Algorithm 3.6 can be either preconditioned by the block reordering or the iterative block

---

**Algorithm 3.5** TLR Quasi-Monte Carlo for MVN probabilities

```latex
1: tlrmvn($B, U, V, a, b, w$)
2: $y \leftarrow 0$, and $P \leftarrow 1$
3: for $i = 1 : r$ do
4: \hspace{1em} if $i > 1$ then
5: \hspace{2em} for $j = i : r$ do
6: \hspace{3em} $\Delta = U_{j,i-1} (V^T_{j,i-1} y_{i-1})$
7: \hspace{3em} $a_j = a_j - \Delta$, $b_j = b_j - \Delta$
8: \hspace{2em} end for
9: \hspace{1em} end if
10: \hspace{1em} $(P', y_i) \leftarrow$ MVN($B_i, a_i, b_i, w_i$)
11: \hspace{1em} $P \leftarrow P \cdot P'$
12: end for
13: return $P$
```

**Algorithm 3.6** TLR Quasi-Monte Carlo for MVT probabilities

```latex
1: tlrmvt($B, U, V, a, b, \nu, w_0, w$)
2: $a' \leftarrow \frac{\chi^{-1}(w_0)}{\sqrt{\nu}} a$, $b' \leftarrow \frac{\chi^{-1}(w_0)}{\sqrt{\nu}} b$
3: return TLRMVN($B, U, V, a', b', w$)
```
reordering. I examine the performance of the TLR Quasi-Monte Carlo algorithms in Section 3.4.

### 3.4 Numerical simulations

Table 3.2 summarizes the performance of the dense (Genz, 1992) and the TLR Quasi-Monte Carlo methods for computing MVN and MVT probabilities, measured on a workstation with 50 GB memory and 8 Xeon(R) E5-2670 CPUs. Methods are assessed over 20 simulated problems for each combination of problem dimension $n$ and correlation strength $\beta$. The highest dimension in my experiment is $2^{16}$. Considerations for higher dimensions include the truncation level required for the success of the TLR Cholesky factorization and the number of Quasi-Monte Carlo samples needed to reach the desired accuracy. As in Tables 2.5 and 2.6, $\beta = 0.3$, 0.1, and 0.03 correspond to strong, medium, and weak correlation strengths. The tile size $m$ for the TLR Quasi-Monte Carlo methods is set as $\sqrt{n}$ but other reasonable choices also suffice. The listed time covers only the integration part while the time for constructing the covariance matrix, the block reordering, and the Cholesky factorization is not included. The costs for the block reorderings are trivial compared with that for the integration, which is indicated in the time difference between the TLR Quasi-Monte Carlo method with reordering and that without reordering. The sample size is set at $N = 10^4$ for the methods without any preconditioner while at $N = 10^3$ for the four preconditioned methods to highlight the time efficiency of the preconditioned methods for reaching a similar accuracy. Note that different from Tables 2.5 and 2.6, here, I generate the integration limits from a univariate normal distribution instead of $U(0, n)$.

Table 3.2 shows that the preconditioned methods achieve an even lower estimation error with one-tenth of the sample size compared with the ones without any preconditioner. The scalability of the TLR methods is better than the dense methods and
the time saving already reaches two orders of magnitude in 16,384 dimensions. The iterative block reordering has a marginal improvement on the convergence rate over

<table>
<thead>
<tr>
<th>$\beta = 0.3$ (strong correlation)</th>
<th>$n$</th>
<th>mvn</th>
<th>tlrmvn</th>
<th>rtlrmvn</th>
<th>rrtlrmvn</th>
<th>mvt</th>
<th>tlrmvt</th>
<th>rtlrmvt</th>
<th>rrtlrmvt</th>
</tr>
</thead>
<tbody>
<tr>
<td>1024</td>
<td>0.5%</td>
<td>0.5%</td>
<td>0.4%</td>
<td>0.5%</td>
<td>0.6%</td>
<td>1.3%</td>
<td>1.3%</td>
<td>1.4%</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2.1s</td>
<td>1.0s</td>
<td>0.1s</td>
<td>0.1s</td>
<td>2.2s</td>
<td>1.1s</td>
<td>0.1s</td>
<td>0.1s</td>
<td></td>
</tr>
<tr>
<td>4096</td>
<td>1.0%</td>
<td>1.0%</td>
<td>0.9%</td>
<td>1.0%</td>
<td>1.1%</td>
<td>1.3%</td>
<td>1.2%</td>
<td>1.2%</td>
<td></td>
</tr>
<tr>
<td></td>
<td>44.8s</td>
<td>9.9s</td>
<td>0.9s</td>
<td>0.9s</td>
<td>44.7s</td>
<td>9.1s</td>
<td>0.8s</td>
<td>0.8s</td>
<td></td>
</tr>
<tr>
<td>16384</td>
<td>2.4%</td>
<td>1.9%</td>
<td>1.9%</td>
<td>1.7%</td>
<td>2.1%</td>
<td>3.1%</td>
<td>2.4%</td>
<td>2.3%</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1227.0s</td>
<td>60.1s</td>
<td>5.0s</td>
<td>5.0s</td>
<td>1214.9s</td>
<td>57.8s</td>
<td>4.8s</td>
<td>4.8s</td>
<td></td>
</tr>
<tr>
<td>65536</td>
<td>NaN</td>
<td>5.7%</td>
<td>3.7%</td>
<td>3.2%</td>
<td>NaN</td>
<td>7.9%</td>
<td>5.9%</td>
<td>5.4%</td>
<td></td>
</tr>
<tr>
<td></td>
<td>NaN</td>
<td>308.6s</td>
<td>28.0s</td>
<td>28.8s</td>
<td>NaN</td>
<td>307.1s</td>
<td>27.5s</td>
<td>28.2s</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$\beta = 0.1$ (medium correlation)</th>
<th>$n$</th>
<th>mvn</th>
<th>tlrmvn</th>
<th>rtlrmvn</th>
<th>rrtlrmvn</th>
<th>mvt</th>
<th>tlrmvt</th>
<th>rtlrmvt</th>
<th>rrtlrmvt</th>
</tr>
</thead>
<tbody>
<tr>
<td>1024</td>
<td>0.5%</td>
<td>0.5%</td>
<td>0.3%</td>
<td>0.3%</td>
<td>0.5%</td>
<td>0.6%</td>
<td>0.6%</td>
<td>0.6%</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2.0s</td>
<td>0.9s</td>
<td>0.1s</td>
<td>0.1s</td>
<td>1.9s</td>
<td>0.9s</td>
<td>0.1s</td>
<td>0.1s</td>
<td></td>
</tr>
<tr>
<td>4096</td>
<td>1.3%</td>
<td>1.2%</td>
<td>1.1%</td>
<td>1.2%</td>
<td>1.3%</td>
<td>1.3%</td>
<td>1.1%</td>
<td>1.0%</td>
<td></td>
</tr>
<tr>
<td></td>
<td>40.2s</td>
<td>5.6s</td>
<td>0.4s</td>
<td>0.4s</td>
<td>40.8s</td>
<td>5.8s</td>
<td>0.5s</td>
<td>0.5s</td>
<td></td>
</tr>
<tr>
<td>16384</td>
<td>4.1%</td>
<td>4.5%</td>
<td>3.9%</td>
<td>3.5%</td>
<td>3.6%</td>
<td>4.1%</td>
<td>3.7%</td>
<td>3.8%</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1213.5s</td>
<td>44.1s</td>
<td>3.5s</td>
<td>3.4s</td>
<td>1204.3s</td>
<td>44.6s</td>
<td>3.5s</td>
<td>3.5s</td>
<td></td>
</tr>
<tr>
<td>65536</td>
<td>NaN</td>
<td>29.2%</td>
<td>11.0%</td>
<td>10.4%</td>
<td>NaN</td>
<td>19.7%</td>
<td>10.6%</td>
<td>9.6%</td>
<td></td>
</tr>
<tr>
<td></td>
<td>NaN</td>
<td>302.9s</td>
<td>26.2s</td>
<td>26.4s</td>
<td>NaN</td>
<td>287.7s</td>
<td>24.3s</td>
<td>24.5s</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$\beta = 0.03$ (weak correlation)</th>
<th>$n$</th>
<th>mvn</th>
<th>tlrmvn</th>
<th>rtlrmvn</th>
<th>rrtlrmvn</th>
<th>mvt</th>
<th>tlrmvt</th>
<th>rtlrmvt</th>
<th>rrtlrmvt</th>
</tr>
</thead>
<tbody>
<tr>
<td>1024</td>
<td>0.1%</td>
<td>0.1%</td>
<td>0.1%</td>
<td>0.1%</td>
<td>0.2%</td>
<td>0.2%</td>
<td>0.3%</td>
<td>0.3%</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2.0s</td>
<td>0.9s</td>
<td>0.1s</td>
<td>0.1s</td>
<td>2.1s</td>
<td>0.9s</td>
<td>0.1s</td>
<td>0.1s</td>
<td></td>
</tr>
<tr>
<td>4096</td>
<td>0.7%</td>
<td>0.7%</td>
<td>0.4%</td>
<td>0.4%</td>
<td>0.6%</td>
<td>0.7%</td>
<td>0.6%</td>
<td>0.7%</td>
<td></td>
</tr>
<tr>
<td></td>
<td>39.6s</td>
<td>5.6s</td>
<td>0.4s</td>
<td>0.4s</td>
<td>39.8s</td>
<td>5.7s</td>
<td>0.4s</td>
<td>0.4s</td>
<td></td>
</tr>
<tr>
<td>16384</td>
<td>3.7%</td>
<td>4.1%</td>
<td>2.7%</td>
<td>2.4%</td>
<td>3.8%</td>
<td>3.4%</td>
<td>1.9%</td>
<td>2.0%</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1202.3s</td>
<td>37.3s</td>
<td>2.2s</td>
<td>2.2s</td>
<td>1197.7s</td>
<td>37.3s</td>
<td>2.2s</td>
<td>2.2s</td>
<td></td>
</tr>
<tr>
<td>65536</td>
<td>NaN</td>
<td>87.7%</td>
<td>15.0%</td>
<td>14.0%</td>
<td>NaN</td>
<td>23.7%</td>
<td>13.1%</td>
<td>10.6%</td>
<td></td>
</tr>
<tr>
<td></td>
<td>NaN</td>
<td>219.7s</td>
<td>16.6s</td>
<td>16.6s</td>
<td>NaN</td>
<td>223.2s</td>
<td>16.7s</td>
<td>16.9s</td>
<td></td>
</tr>
</tbody>
</table>
the non-iterative version. It is worth noting that the block reorderings are more effective when heterogeneity is strong among the spatial variables. They would make no difference in the extreme scenario, where the correlation is constant and the integration limits are the same. The upper limits I choose for Table 3.2 are relatively large, whereas MVN and MVT problems with much smaller probabilities may appear in applications. In fact with $10^4$ samples, the non-preconditioned methods already fail to provide a meaningful estimate when $n = 65,536$ and $\beta = 0.03$. The convergence rate generally decreases for small integration limits, where importance sampling techniques, for example, Botev (2017), become necessary in reducing the sample size. However, in this chapter, I do not resort to importance sampling but suggest that the preconditioned methods provide a decent estimate for the log-probability in relatively high dimensions.

Figure 3.4 compares the relative error for the probabilities and the log-probabilities, where the simulated problems have smaller integration limits generated from $N(4.0, 1.5^2)$. The ‘mvn’ and ‘mvt’ methods are not included because they share the same error level as the ‘tlrmvn’ and ‘tlrmvt’ methods and the sample size $N$ for all listed methods is $10^4$. The error for the log-probability cannot be directly estimated, for which the same problem is estimated 10 times to provide replicates of the log-probabilities. In terms of the relative error for probabilities, all methods fail to provide a meaningful estimate when $n = 16,384$. However, the relative error for the log-probability is significantly smaller, which indicates that the methods are relatively reliable if I use the logarithm of the estimates and that the distribution of the estimates has significant skewness. Consistent with Table 3.2, the iterative block reordering outperforms the non-iterative version while both have a considerably higher convergence rate than the non-preconditioned versions. I illustrate that the log-probability estimated from the preconditioned methods is sufficient for the maximum likelihood procedure in Section 3.5.
Figure 3.4: The relative error for probabilities and log-probabilities. For each $n$, the three boxplots, from left to right, correspond to the TLR method, the TLR with the block reordering method, and the TLR with the iterative block reordering method. The relative error for log-probabilities is based on 10 estimations of the same replicate. Each boxplot consists of 20 replicates. The covariance matrix, degrees of freedom, and lower integration limits are the same as Table 3.1 ($\beta = 0.1$) while the upper limits are generated from $N(4.0, 1.5^2)$.

3.5 Application to stochastic generators

3.5.1 A skew-normal stochastic generator

Stochastic generators are typically used to reproduce the physical processes that are otherwise emulated through partial differential equation systems. The emulation of the system requires tens of variables and a very fine grid in the spatio-temporal domain, which is extremely time-and-storage demanding (Castruccio and Genton, 2016). For example, the Community Earth System Model (CESM) Large ENSsemble project (LENS) required ten million CPU hours and more than four hundred terabytes
of storage to emulate one initial condition (Jeong et al., 2018). Castruccio and Genton (2016) found statistical models could form efficient surrogates for reproducing the physical processes in climate science and concluded that additional model flexibilities would facilitate the modeling on a finer scale; see Castruccio and Genton (2018) for a recent account.

The MVN and MVT methods developed in this chapter allow for more complexity in the construction of stochastic generators. A significant improvement in flexibility is to introduce skewness since the majority of the statistical models used nowadays are Gaussian-based, i.e. they rely on a symmetric distribution. Generally speaking, there are three ways of introducing skewness to an elliptical distribution, all of which involve the cumulative distribution function of the distribution. The first is through reformulation, which multiplies the elliptical probability density function by its cumulative distribution function. The second method introduces skewness via selection. Assuming $(X^\top, Y^\top)^\top$ have a joint multivariate elliptical distribution, $X|Y > \mu$, where $\mu$ is an $n$-dimensional vector, has a skew-elliptical distribution. Arellano-Valle and Genton (2010) studied a general class of skewed reformulations and introduced its link to the selection representation. The third method is defined by the stochastic representation $Z = X + |Y|$, where $X$ and $Y$ are two independent elliptical random vectors. Zhang and El-Shaarawi (2010) studied the skew-normal random field based on this construction assuming a general correlation structure for $Y$, because of which a direct maximum likelihood estimation is almost impossible. Instead, $Y$ was taken as a latent random variable and the EM algorithm was applied. In the M-step, the conditional expectations of $X$ were computed through Markov chain Monte Carlo. Thus, the cost of maximizing the likelihood is expectedly high.

The three methods have equivalent forms in the one-dimensional case but extend differently into higher dimensions. The first method is flexible but provides little information on the underlying stochastic process. The second method has a clear
underlying model and its probability distribution function is usually more tractable than that from the third method but the choice for $Y$ is usually not obvious, especially when $X$ is in high dimensions. In the third method, the parameterization is usually more intuitive and the model can be also applied in spatial statistics as a random field. However, the probability distribution function is a summation of a number of terms exponentially growing with the number of locations $n$, which renders the model difficult to scale. Weighing an intuitive stochastic representation against the probability distribution function complexity, I modify the third construction method based on the $C$ random vector properties introduced in Arellano-Valle et al. (2002). A $C$ random vector can be written as the Hadamard product of two independent random vectors, representing the sign and the magnitude, respectively. When $Y$ is a $C$ random vector and $X$ is independent from $Y$, $G(X, Y)|Y > 0$ has the same distribution as $G(X, |Y|)$ for any function $G(\cdot)$ (Arellano-Valle et al., 2002). Similar to these authors, my model assumes a stochastic representation where the matrix-vector multiplication that models the dependence structure among the skewness components follows the absolute value operation:

$$Z^* = \xi 1_n + AX + B|Y|,$$  \hspace{1cm} (3.5)

where $\xi \in \mathbb{R}$ is the location parameter, $\{X_i| i = 1, \ldots, n\} \cup \{Y_i| i = 1, \ldots, n\}$ are independent and identically distributed standard normal random variables. Hence, $AX + B|Y|$ has the same distribution as $AX + BY|Y > 0$ since I can choose $G(X, Y)$ to be $AX + BY$. The probability distribution function of $Z^*$ avoids the $2^n$-term summation, which was the hinge in Zhang and El-Shaarawi (2010), making the probability distribution function computation more scalable.

The marginal representation shown in Equation (3.5) is difficult to extend to the multivariate skewed-$t$ version because the sufficient condition for the equivalence
between the conditional representation and the marginal representation is that \( X \) and \( Y \) are independent (Arellano-Valle et al., 2002). However, the sum of independent Student-\( t \) random variables does not necessarily lead to another Student-\( t \) random variable. Another issue with this representation is the difficulty to generalize as a random field. When \( A \) is a Cholesky factor, \( AX \) coincides with the classical Gaussian random field but it is not obvious that \( B|Y| \) can be derived from any well-defined random field. However, for stochastic generators, the model is usually simulated on a fixed spatial domain without the need for prediction at unknown locations and therefore, Equation (3.5) may serve as the surrogate model for a physical system. In general, this stochastic representation has better-rounded properties due to its advantage in estimation, simulation, and flexibility. Specifically,

- the probability distribution function avoids the summation of \( 2^n \) terms as in the model \( AX + |BY| \), which makes the probability distribution function estimable;
- the marginal representation in Equation (3.5) allows for more efficient simulation compared with conditional representations;
- the correlation structure between the skewness components \( B|Y| \) has full flexibility controlled by \( B \), which can adapt to different datasets for model fitting.

Considering the reasons above, I simulate \( Z^* \) based on the skew-normal distribution without studying the skewed Student-\( t \) counterpart for the simulation study and use the same model as a stochastic generator for the Saudi wind speed dataset that has more than 18,000 spatial locations.

### 3.5.2 Estimation with simulated data

I construct \( A \) and \( B \) before simulating \( Z^* \), where \( A \) controls the correlation strength of the symmetric component while \( B \) adjusts the level of skewness and the correlation between the skewness component. To have a parsimonious model, \( A \) is assumed to be
the lower Cholesky factor of a covariance matrix constructed from the 2D exponential kernel, \( \sigma_1^2 \exp(-\|h\|/\beta_1) \), \( \beta_1 > 0 \), and \( B \) takes the form of a covariance matrix from the kernel, \( \sigma_2^2 \exp(-\|h\|/\beta_2) \), \( \beta_2 > 0 \), where \( h \) is the vector connecting the two spatial variables’ locations. I choose the form of a covariance matrix instead of a Cholesky factor for \( B \) out of two reasons. Numerically, the row sum of a Cholesky factor usually increases with the row index, which produces a large difference between the sum of the first row and that of the last row when the dimension is high. This would cause the coefficients of \( Z^* \) to have a varying order of magnitude. Secondly, due to the first reason, the likelihood would depend on the ordering of the random variables in \( Z^* \).

When \( B \) is a covariance matrix, the row sums usually have similar magnitudes and the likelihood function becomes independent from the ordering scheme. The probability distribution function of \( Z^* \) can be derived based on the results in [Arellano-Valle et al. (2002)] to be:

\[
2^n \phi_n(z - \xi_1, AA^\top + BB^\top) \Phi_n\{-\infty; (I_n + C^\top C)^{-1}C^\top A^{-1}(z - \xi_1); (I_n + C^\top C)^{-1}\},
\]

(3.6)

where \( C = A^{-1}B \). Assuming a dense representation, the matrix operations have \( O(n^3) \) complexity. However, the TLR representation can closely approximate \( AA^\top \) and \( B \) due to the 2D exponential covariance model. The subsequent Cholesky factorization, matrix multiplication, and matrix inversion can be performed at adequate accuracy and the complexity can be reduced by one order of magnitude. For each \( n = 4^r, r = 4, 5, 6, 7 \), I generate the geometry in the \([0, 2^{r-4}] \times [0, 2^{r-4}] \) square, mimicking an expanding domain. The spatial locations are on a perturbed grid, where the grid’s length unit is \( 1/15 \) and the perturbation is uniformly distributed within \((-0.4/15, 0.4/15)^2 \). Here, \( A \) and \( B \) are constructed based on the covariance kernel and the simulated geometry. The likelihood function is the probability distribution function of \( Z^* \) shown in Equation (3.6) and the optimization seeks to find the param-
Figure 3.5: Boxplots of 30 estimation results. Each estimation is based on one realization of the \( n \)-dimensional skew-normal model. The red dashed line marks the true value used for generating random vectors from the skew-normal model.

The outliers may indicate that there is a local maximum, where \( \sigma_1 \) and \( \beta_1 \) are large and \( \sigma_2 \) is small, on a similar magnitude level with the global maximum. In this case, the estimation result is closer to a Gaussian random field.

In the application study, I found that the likelihood was extremely small when the dimension \( n \) was high because the order of magnitude cumulated through the multiplication of one-dimensional probabilities as shown in Equation (3.1), exceeding the lower boundary of double-precision numbers. I extracted the exponent after each...
multiplication step described on Line 9 of Algorithm 3.1. This mechanism allows for a minimum value of $2^{\text{MIN}_\text{INT}}$, where $\text{MIN}_\text{INT}$ is the minimum integer allowed.

### 3.5.3 Estimation with wind data from Saudi Arabia

The dataset I use for modeling is the daily wind speed over a region in the Kingdom of Saudi Arabia on August 5th, 2013, produced by the weather research and forecasting (WRF) model (Yip, 2018), which numerically predicts the weather system based on partial differential equations on the mesoscale and features strong computation capacity to serve meteorological applications (Skamarock et al., 2008). The dataset has an underlying geometry with 155 longitudinal and 122 latitudinal bands. Specifically, the longitude increases from 40.034 to 46.960 and the latitude increases from 16.537 to 21.979, both with an increment size of 0.045. Before fitting the skew-normal model, I subtract the wind speed at each location with its mean over a six-year window (six replicates in total) to increase the homogeneity across the locations. The vectorized centered wind speed data is used as the input dataset, $Z^*$, for the maximum likelihood estimation. The dataset has a skewness of $-0.45$ and is likely to benefit from the skewness flexibility introduced by the model in Equation (3.5). It is worth noting that $B|Y|$ has a negative skewness under my parameterization for $B$ although all its coefficients are non-negative.

Table 3.3: Parameter specifications and estimations based on the skew-normal (SN) model and the Gaussian random field (GRF)

<table>
<thead>
<tr>
<th></th>
<th>$\xi$</th>
<th>$\sigma_1$</th>
<th>$\beta_1$</th>
<th>$\sigma_2$</th>
<th>$\beta_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Range</td>
<td>$(-2,2)$</td>
<td>$(0.1,2.0)$</td>
<td>$(0.1,5.0)$</td>
<td>$(0.0,2.0)$</td>
<td>$(0.01,1.0)$</td>
</tr>
<tr>
<td>Initial Value</td>
<td>0.000</td>
<td>1.000</td>
<td>0.100</td>
<td>1.000</td>
<td>0.010</td>
</tr>
<tr>
<td>SN</td>
<td>-1.211</td>
<td>1.028</td>
<td>4.279</td>
<td>0.419</td>
<td>0.065</td>
</tr>
<tr>
<td>GRF</td>
<td>0.338</td>
<td>1.301</td>
<td>4.526</td>
<td>N.A.</td>
<td>N.A.</td>
</tr>
</tbody>
</table>
The likelihood function is described in Equation (3.6), where the parameterization of \( A \) and \( B \) also remains unchanged. The optimization involves five parameters, namely \( \xi, \sigma_1, \beta_1, \sigma_2, \) and \( \beta_2 \), whose search ranges, initial values, and optimized values are listed in Table 3.3. Since the likelihood requires the inverse of \( A \) as shown in Equation (3.6), I set the lower limit of \( \sigma_1 \) to 0.1 to avoid the singularity. The correlation-strength parameters \( \beta_1 \) and \( \beta_2 \) can theoretically be close to zero but setting a lower limit above zero can avoid boundary issues. The convergence level is set at \( 10^{-3} \) and the optimization produces the results shown in Table 3.3, which has a (negative) log-likelihood of 11,508. I compare the optimized skew-normal model with the optimized classical Gaussian random field, which is also a simplified version of Equation (3.6), where \( \sigma_2 \) is fixed to zero: \( \mathbf{Z}^* = \xi \mathbf{1}_n + \mathbf{A} \mathbf{X} \). The estimation of the Gaussian random field thus involves three parameters, \( (\sigma_1, \beta_1, \xi) \), for which the optimization setups are the same as those for the skew-normal model. The estimated parameter values are also summarized in Table 3.3, which has a (negative) log-likelihood of 10,797. The functional boxplots (Sun and Genton, 2011) of the empirical semivariogram based on 100 simulations of the fitted skew-normal model and the Gaussian random field are shown in Figure 3.6. The skew-normal model has significantly smaller band width than the Gaussian random field in the semivariogram plot, although both cover the semivariogram of the original data. The BIC values of the two models and the quantile intervals of the empirical moments based on the same 100 simulations are illustrated in Table 3.4. The BIC values strongly indicate that

Table 3.4: Empirical moments and BIC comparison. SN denotes the skew-normal model and GRF denotes the Gaussian random field. The intervals represent the 5% to 95% quantile intervals based on 100 simulations.

<table>
<thead>
<tr>
<th></th>
<th>Mean</th>
<th>Variance</th>
<th>Skewness</th>
<th>Kurtosis</th>
<th>BIC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wind data</td>
<td>0.042</td>
<td>0.932</td>
<td>-0.445</td>
<td>2.873</td>
<td>N.A.</td>
</tr>
<tr>
<td>SN</td>
<td>(-1.079, 1.360)</td>
<td>(0.308, 1.054)</td>
<td>(-0.644, 0.449)</td>
<td>(2.274, 3.595)</td>
<td>-22986</td>
</tr>
<tr>
<td>GRF</td>
<td>(-1.644, 1.911)</td>
<td>(0.612, 2.594)</td>
<td>(-0.717, 0.489)</td>
<td>(2.116, 3.705)</td>
<td>-21565</td>
</tr>
</tbody>
</table>
Figure 3.6: The heatmap based on one simulation and the functional boxplot of the empirical semivariogram based on 100 simulations. Top to bottom are the fitted skew-normal model and the Gaussian random field. The green curve denotes the empirical semivariogram based on the wind speed data. The distance is computed as the Euclidean distance in the longitudinal and latitudinal coordinate system.

the skew-normal model is a better fit than the Gaussian random field. This can be also seen from the variance quantile intervals and the functional boxplots of the empirical semivariogram, where the former has smaller variance and its semivariogram is more aligned with that of the Saudi wind data. The empirical moments ignore the connection between the spatial locations, thus may not be a comprehensive measure for the fitting quality. Except for the one for the variance, the two models are not significantly different in terms of the other three quantile intervals.

3.6 Chapter summary

In this chapter, I first summarized the SOV methods from Genz (1992) and Genz and Bretz (1999) for MVN and MVT probabilities. Two definitions of the MVT
probability were compared and one was shown to have better numerical properties than the other. Next, I demonstrated that the hierarchical representation (Hackbusch, 2013) is not suitable for the block reordering introduced in Chapter 2 while the TLR structure (Weisbecker, 2013; Mary, 2017; Akbudak et al., 2017) creates a synergy. An iterative block reordering was proposed that further improves the convergence rate compared with the non-iterative version. Later, I showed that the relative error grows quickly with the integration dimension $n$ when integration limits are non-trivial but the log-probability can be much more reliably estimated. The absolute error of the log-probability is more related to the relative error of the probability than the relative error of the log-probability. However, the relative error of the log-probability is indicative of the smoothness of the likelihood curve and how much we can trust the maximum likelihood estimators. Finally, I used both simulated and physical datasets to illustrate that the preconditioned methods can provide meaningful estimates for high-dimensional likelihood estimations. The new methods reduce the computation time by an order of magnitude compared with Genton et al. (2018) and two orders of magnitude compared with dense versions (Genz, 1992).
Chapter 4

Sum of Kronecker Products Representation and Its Cholesky Factorization for Spatial Covariance Matrices from Large Grids

4.1 Notations and relevant literature

In this chapter, I study a novel matrix compression method, the sum of Kronecker products (SKP) representation, \( \sum_{i=1}^{k} U_i \otimes V_i \), which utilizes the linear resemblance between blocks of the approximated matrix. The covariance matrices from large grids, which are frequently used in Gaussian random fields, are the major motivation of the study of the SKP representation. Gaussian random fields are widely applied to data modeling in spatial statistics. In addition to the mean structure, they are defined by a positive definite covariance function \( K(\cdot, \cdot) \) and a spatial domain. Given any set of \( n \) locations, \( \{s_1, \ldots, s_n\} \), within the domain, the corresponding \( n \) spatial variables have a joint normal distribution, \( N_n(0, \Sigma) \) with \( \Sigma(i, j) = K(s_i, s_j) \). I show that when \( \{s_1, \ldots, s_n\} \) are located on a regular grid, the linear resemblance between all blocks of \( \Sigma \) is strong enough for most covariance functions to yield significant storage savings.

The SKP approximation can be viewed as one application of the nearest Kronecker product (NKP) problem (Van Loan and Pitsianis, 1993; Van Loan, 2000). The general NKP problem is formulated as: \( \text{argmin}_{B, C} \| A - B \otimes C \|_F \), where \( \| \cdot \|_F \) denotes the Frobenius norm. Implicitly, I assume that the dimensions of \( A, B, \) and \( C \) are conformable. One method for finding \( B \) and \( C \) is through the SVD of the rearrangement of \( A \) defined in Van Loan and Pitsianis (1993), denoted by \( \tilde{A} \). In
In this case, \( \| \mathbf{A} - \mathbf{B} \otimes \mathbf{C} \|_F^2 = \sum_{i=2}^{q} \sigma_i^2 \), where \( \sigma_i \)'s are the singular values of \( \tilde{\mathbf{A}} \) in descending order and \( q \) is the number of singular values of \( \tilde{\mathbf{A}} \). A natural extension is to select the top \( k \) singular vectors of \( \tilde{\mathbf{A}} \) to further reduce the approximation error. When the required accuracy level is achieved with \( k \ll q \) and \( \mathbf{B} \) and \( \mathbf{C} \) have a similar number of coefficients, this representation can lead to linear storage cost with respect to the dimensions of \( \mathbf{A} \). This representation also coincides with the compression of multidimensional arrays, referred to as tensors, with Kronecker products. \cite{Grasedyck2013} surveyed low-rank tensor approximation techniques with a focus on the discretization of multivariate functions. However, these techniques usually target ultra-high dimensional applications, for example, \( 2^{50} \), and apply a series of Kronecker products for the approximation, which is likely to increase the approximation error for spatial covariance matrices. Moreover, matrix operations become more expensive and less robust when the Kronecker chain extends. For example, the truncation used in matrix addition to keep a small \( k \) typically involves an optimization routine \cite{Grasedyck2013} and matrix factorization is generally difficult for such representations.

In this chapter, I show that the memory footprint of the SKP representation can be reduced by one order of magnitude compared with the widely used hierarchical matrices \cite{Hackbusch2015}. Additionally, I develop a corresponding Cholesky factorization algorithm that can scale up to one million dimensions. Section 4.2 describes an ACA \cite{Bebendorf2000} based framework with linear complexity for building the Kronecker factors \( \{ \mathbf{U}_i \} \) and \( \{ \mathbf{V}_i \} \). Section 4.3 discusses the optimal strategy for indexing the spatial locations and choosing the dimensions of the Kronecker factors. In Section 4.4, I propose a Cholesky factorization algorithm that recycles the matrix multiplication between base blocks and provide a detailed complexity analysis. Section 4.5 illustrates one application of the computed Cholesky factor: the simulation of Gaussian random fields in one million dimensions. Section 4.6 concludes and
discusses the potential limitations of this new representation.

4.2 Nearest sum of Kronecker products

Since this chapter focuses on spatial covariance matrices, I use \(\|\Sigma - U \otimes V\|_F\) to denote the NKP problem. Van Loan and Pitsianis (1993) introduced two frameworks for computing \(U\) and \(V\): the SVD framework and the separable least squares framework. The separable least squares framework considers \(U\) or \(V\) as fixed and optimizes the other as in a linear least squares problem. This iterative method needs extra consideration on the convergence rate when the NKP problem is generalized to \(\|\Sigma - \sum_{i=1}^{k} U_i \otimes V_i\|_F\). In this section, I summarize the SVD framework for completeness and introduce an ACA-based variant that reduces the complexity by two orders.

4.2.1 SVD framework

Assuming \(U \in \mathbb{R}^{m_1 \times n_1}\) and \(V \in \mathbb{R}^{m_2 \times n_2}\), the SVD framework starts with partitioning \(\Sigma\) into blocks of the same dimensions as \(V\). Next, it rearranges \(\Sigma\) into \(\tilde{\Sigma}\), where the rows correspond to the transpose of the vectorized blocks of \(\Sigma\) in the column-major order. For example, when \(m_2 = n_2 = n/2\):

\[
\Sigma = \begin{bmatrix} \Sigma_{11} & \Sigma_{12} \\ \Sigma_{21} & \Sigma_{22} \end{bmatrix} \Rightarrow \tilde{\Sigma} = \begin{bmatrix} \text{vec}(\Sigma_{11})^\top \\ \text{vec}(\Sigma_{21})^\top \\ \text{vec}(\Sigma_{12})^\top \\ \text{vec}(\Sigma_{22})^\top \end{bmatrix}.
\]

After the rearrangement, an SVD is applied to \(\tilde{\Sigma} = [\tilde{U}_1, \ldots, \tilde{U}_{m_1n_1}]\Lambda[\tilde{V}_1, \ldots, \tilde{V}_{m_2n_2}]^\top\); where \([\tilde{U}_1, \ldots, \tilde{U}_{m_1n_1}]\) and \([\tilde{V}_1, \ldots, \tilde{V}_{m_2n_2}]\) are orthogonal matrices and \(\Lambda\) is a diagonal matrix containing the singular values of \(\tilde{\Sigma}\), \(\sigma_1, \ldots, \sigma_q, q = \min(m_1n_1, m_2n_2)\), in descending order. Finally, \(\sigma_1 \tilde{U}_1\) and \(\tilde{V}_1\) are matricized to form \(U\) and \(V\). This
framework leads to the intuitive extension of selecting the first \( k \) singular vectors (values) when one considers approximating with a sum of Kronecker products. The approximation error in squared Frobenius norm, \( \| \Sigma - \sum_{i=1}^{k} U_i \otimes V_i \|_F^2 \), amounts to \( \sum_{i=k+1}^{q} \sigma_i^2 \).

When linear dependence between the rows or columns of a matrix (block) is strong, the magnitude of the singular values decreases rapidly, which lays the ground for structured representations. The linear dependence, also known as the low-rank feature, typically builds on some separability conditions and creates computation savings when stored in the structured form. The hierarchically semi-separable (HSS) matrix (Chandrasekaran et al., 2005) and the hierarchical matrix (Hackbusch, 2015) are well-studied examples of this category. Along this thread, the sum of Kronecker products approximation can be viewed as utilizing the linear dependence between the blocks of \( \Sigma \). Proper choices of \((m_1, n_1, m_2, n_2)\) and the indexing scheme for the spatial locations are crucial to the block-wise linear dependence, which is discussed in Section 3.

### 4.2.2 ACA framework

Although the SVD framework provides the approximation error, \( \| \Sigma - \sum_{i=1}^{k} U_i \otimes V_i \|_F \), and a precise solution for \( \{U_i\} \) and \( \{V_i\} \), its complexity grows at a cubic rate and becomes prohibitively expensive when \( \min(m_1 n_1, m_2 n_2) > 10^4 \). To make the construction process more scalable, I consider both the random sampling method (Halko et al., 2011) and the ACA method (Bebendorf, 2000). The random sampling method has a complexity of \( O(km_1 n_1 m_2 n_2) \) because it multiplies \( \Sigma \) with a \( (m_2 n_2) \times (k + p) \) random matrix, where \( p \) is a small constant denoting the extra sampling size. The ACA method has a linear complexity of \( O(k^2 (m_1 n_1 + m_2 n_2)) \) (Zhao et al., 2005) because it accesses only \( k \) rows and columns of the approximated matrix. However, it lacks an error bound apart from specific assumptions on the bounds and smoothness of the kernel function (Bebendorf and Rjasanow, 2003). I choose the latter since cor-
relation functions are bounded and most spatial covariance functions are sufficiently smooth.

The ACA algorithm has been used in Chapters 2 and 3; see also Zhao et al. (2005) for the pseudocode of the ACA algorithm. Here, I describe a block version of it in Algorithm 4.1 for completeness. I use the notation \( \Sigma_{ib,jb} \) to denote the \((i_b,j_b)\)-th block of \( \Sigma \).

**Algorithm 4.1 Block ACA Algorithm**

1: procedure BLKACA(\( \Sigma \), \( \epsilon \))
2: \( q \leftarrow 0 \), \( k \leftarrow 0 \), \( i_b \leftarrow 1 \), and \( j_b \leftarrow 1 \)
3: while true do
4: \( V_{k+1} \leftarrow \Sigma_{ib,jb} - \sum_{l=1}^{k} U_l(i_b,j_b)V_l \)
5: \((i,j) \leftarrow \text{argmax}_{i,j} (|\Sigma_{ib,jb}(\hat{i},\hat{j})|); \ V_{k+1} \leftarrow V_{k+1}/\Sigma_{ib,jb}(\hat{i},\hat{j}) \)
6: \( U_{k+1}(i_b,j_b) \leftarrow \Sigma_{ib,jb}(\hat{i},\hat{j}), 1 \leq i_b \leq m_1, 1 \leq j_b \leq n_1 \)
7: \( U_{k+1} \leftarrow \sum_{l=1}^{k} V_l(i,j)U_l \)
8: \( q \leftarrow q + 2 \sum_{l=1}^{k} |\langle U_l, U_{k+1} \rangle \langle V_l, V_{k+1} \rangle| + \|U_{k+1}\|_F^2 + \|V_{k+1}\|_F^2 \)
9: if \( ||U_{k+1}||_F||V_{k+1}||_F \leq \epsilon \sqrt{q} \) then
10: \( \text{return } \{U_i\}_{i=1}^{k+1}, \{V_i\}_{i=1}^{k+1}, \text{ and } k + 1 \)
11: end if
12: \((i_b,j_b) \leftarrow \text{argmax}_{i,j} (1_{(i_b,j_b) \neq (i_b,j_b)} |U_{k+1}(\hat{i},\hat{j})|), k \leftarrow k + 1 \)
13: end while
14: end procedure

The \( \Sigma \) and parentheses to refer to a matrix (block) coefficient. The \( \langle \cdot, \cdot \rangle \) operator applied to matrices computes the inner product between the two vectorized matrices. The \{\( U_i \)\} and \{\( V_i \)\} returned by this algorithm are guaranteed to be not empty, which is easier for implementation. When orthogonality is desired, the modified Gram-Schmidt (MGS) algorithm (Björck, 1994) can be used to normalize \{\( V_i \)\} and transform \{\( U_i \)\} at a cost of \( O((m_1n_1 + m_2n_2)k^2) \), where \( k \) is the number of elements in \{\( U_i \)\} and \{\( V_i \)\}. MGS is numerically as stable as the ordinary Householder QR factorization (Björck, 1994) but leads to a more intuitive transformation for \{\( U_i \)\}. Similar to Householder QR factorization, MGS has the disadvantage of reduced parallelism compared with the ordinary Gram-Schmidt algorithm.

I refer to the approximation of \( \Sigma \) with block ACA followed by a normalizing MGS routine as the ACA framework. Table \( \Box \) compares the SVD and ACA frameworks.
Table 4.1: Approximation of $\Sigma$ with SVD and ACA frameworks. $\Sigma$ is the covariance matrix built with an exponential correlation function, $\exp(-\|x\|/\beta)$, $\beta = 0.3$ and $s_1 s_2$ locations distributed on an $s_1 \times s_2$ grid whose unit distance is $1/s_2$. The choice for $(m_1, n_1, m_2, n_2)$ is $(s_1, s_1, s_2, s_2)$ and the spatial indexing is along the second dimension of the grid. Error measures $\|\Sigma - \sum_{i=1}^{k} U_i \otimes V_i\|_F/\|\Sigma\|_F$ and $k$ is the number of Kronecker products. The unit for time is seconds. Within each cell, the SVD framework is on the left while the ACA framework is on the right.

<table>
<thead>
<tr>
<th>$(s_1, s_2)$</th>
<th>(64, 64)</th>
<th>(64, 128)</th>
<th>(128, 128)</th>
<th>(128, 256)</th>
<th>(256, 256)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Method</td>
<td>SVD</td>
<td>ACA</td>
<td>SVD</td>
<td>ACA</td>
<td>SVD</td>
</tr>
<tr>
<td>Error</td>
<td>5.5e-6</td>
<td>2.8e-5</td>
<td>2.3e-6</td>
<td>2.0e-6</td>
<td>1.4e-6</td>
</tr>
<tr>
<td>$k$</td>
<td>9</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>11</td>
</tr>
<tr>
<td>Time</td>
<td>4.4e+1</td>
<td>1.4e-2</td>
<td>9.5e+1</td>
<td>1.4e-2</td>
<td>2.7e+3</td>
</tr>
</tbody>
</table>

in terms of relative error, computation time, and the size of $\{U_i\}$ and $\{V_i\}$. The simulations in this chapter are compiled with the Intel(R) C++ compiler with level three optimization and run on the Intel(R) Xeon(R) E5-2680 v4 @ 2.40GHz CPU. The times in Table 4.1 are without multithreading. The computation time of the SVD framework, mainly consisting of the SVD routine from the Intel(R) MKL, already becomes prohibitive when $\Sigma$ is of dimension 16,384. Hence, an alternative that computes $\{U_i\}$ and $\{V_i\}$ at a lower complexity level is critical to the SKP representation. The ACA framework visits only a small proportion of $\Sigma$ and may leave the unvisited insufficiently approximated. Here, however, I consider the relative error from the ACA framework small enough for demonstrating the merits of the SKP representation for spatial covariance matrices. For applications with higher accuracy requirement or a less smooth kernel, the randomized SVD can be a more proper alternative.

4.3 Quasi-optimal sum of Kronecker products

Fixing the covariance function and the spatial locations, I am left with the choice of a spatial indexing scheme and the dimensions of $\{U_i\}$ and $\{V_i\}$ to build an efficient SKP representation. It is desirable to find the optimal indexing and partitioning that lead to the strongest linear dependence between the blocks of $\Sigma$ but this can
be a challenging problem. Meanwhile, there is a straightforward choice that typically produces an efficient SKP representation, which indexes the spatial locations along one dimension and makes the block size a multiple of that dimension’s length. In fact, Table 4.1 partially proves this point and this section shows that this choice of indexing and partitioning is quasi-optimal. I also compare it with alternative structured representations in terms of storage costs. Finally, I show the limitation of approximating with the SKP representation when spatial locations are on a perturbed grid, leading to irregularly spaced locations.

4.3.1 Indexing and partitioning

Common space-filling curves used for indexing multidimensional locations include the z-curve (Orenstein, 1986), the Gray-coded curve (Faloutsos, 1986), and the Hilbert curve (Faloutsos and Roseman, 1989), among which the last one, under most circumstances, best preserves locality (Jagadish, 1990; Moon et al., 2001). Therefore, I add the Hilbert curve to the comparison group. I also include the z-curve since it was used in Genton et al. (2018) and Chapters 2 and 3 for building the low-rank representations. A third order under consideration is what Table 4.1 is based on: grouping locations based on the x-coordinate and indexing them along the y-axis, which is referred to as the y-major order in this chapter. An illustration of the three indexing methods is shown in Figure 4.1, where the spatial locations are on a 4 × 4 grid with the x coordinate increasing from left to right and the y coordinate increasing from bottom to top.

Given a covariance function and spatial locations, I investigate which combination of ordering and partitioning leads to the most efficient SKP representation. I use the number of Kronecker products, k, and the number of double-precision values needed by \{U_i\} and \{V_i\} as the proxy for efficiency. To account for general spatial covariance matrices, I experiment with both stationary and non-stationary correlation functions.
Figure 4.1: Illustrations of (a) y-major order, (b) z-curve order, and (c) Hilbert curve order

Specifically, I use the technique from Paciorek and Schervish (2004) to transform the exponential correlation function:

$$
\Sigma(i, j) = \exp \left( -\frac{\| (x_i, y_i) - (x_j, y_j) \|}{\beta_0} \right),
$$

$$
\Sigma_{\text{ns}}(i, j) = \beta_i^{1/2} \beta_j^{1/2} \left\{ \frac{\beta_i^2 + \beta_j^2}{2} \right\}^{-1/2} \exp \left( -\sqrt{\frac{2\| (x_i, y_i) - (x_j, y_j) \|^2}{\beta_i^2 + \beta_j^2}} \right),
$$

$$
\beta_0 = 0.3, \quad \beta_i = 0.1 + 0.2 * x_i,
$$

where $\Sigma$ and $\Sigma_{\text{ns}}$ are respectively the stationary and non-stationary exponential correlation matrices, $(x_i, y_i)$ are coordinates of the spatial location $s_i$ in the 2D unit square, and $\beta_i$ is the correlation strength parameter that varies with the $x$-coordinate in the range of $(0.1, 0.3)$. When $\beta = 0.3$, the exponential correlation function $\exp(-\|x\|/\beta)$ has an effective range of 0.9, which is considered strong in the unit square. Similarly, $\beta = 0.1$ is considered a medium correlation strength.

Figure 4.2 compares all combinations of ordering and partitioning under four types of covariance matrices. The truncation is at the smallest $k$ that has a relative error, $\| \Sigma - \sum_{i=1}^{k} U_i \otimes V_i \|_F / \| \Sigma \|_F$, less than $1e-5$ based on the SVD framework. The y-major order is clearly the most suitable choice for the SKP representation, with significantly higher efficiency under almost all partitions. The Hilbert curve also outperforms the z-curve in this example. I find that when $m_2$ and $n_2$ are equal to
Figure 4.2: Efficiency of ordering and partitioning methods. The $x$-axis denotes the number of Kronecker products and the $y$-axis (ndbl) denotes the number of double-precision values needed by $\{U_j\}$ and $\{V_i\}$. Green triangular dots assume the $y$-major order, blue square dots assume the $z$-curve order, and red circular dots assume the Hilbert curve order. The spatial locations are on a $10 \times 32$ grid for (a) and (c) and a $32 \times 32$ grid for (b) and (d), with a unit distance of $1/32$. The covariance functions are stationary (Equation (4.1)) for (a) and (b) and non-stationary (Equation (4.2)) for (c) and (d). All possible partitions are listed but only the ones having the smallest five (six) ndbl are labelled with $(m_2, n_2, \text{ndbl})$.

either $s_2$ or half of $s_2$, the length of the grid along the $y$-axis, the storage costs become close to optimal or quasi-optimal. In fact,

$$\text{ndbl} = k \times (m_1 n_1 + m_2 n_2) = k \times \left( \frac{n^2}{m_2 n_2} + m_2 n_2 \right),$$

where $n = s_1 s_2$ is the number of spatial locations. The part in parentheses reaches minimum when $m_2 n_2 = n$ and the linear dependence between the blocks is stronger when the length of the indexing dimension, $s_2$, can be divided by $m_2$ and $n_2$, which leads to a smaller $k$. I refer to the combination of the $y$-major order and the choice
of \( m_2 = n_2 \) being the same factor of \( s_2 \) that is close to \( \sqrt{n} \) as the quasi-optimal SKP representation.

### 4.3.2 Storage costs comparison

I demonstrate the efficiency of the SKP representation through comparison with the storage costs of approximating with a hierarchical matrix. Both should grow at a log-linear rate assuming that \( k \) grows at \( O(\log n) \) and the admissibility condition can be met without too many partitions. The hierarchical matrices are built with HLIBpro v2.7.2 (Borm et al., 2003; Kriemann, 2003; Grasedyck et al., 2008) under the geometrically balanced partitioning strategy and the standard admissibility condition with the ratio between the diameter and the distance equal to 2; see Hackbusch (2015) for a detailed introduction on hierarchical matrices or Genton et al. (2018) for examples in spatial statistics. Figure 4.3(a) provides a concrete image of the hierarchical matrix that approximates the same correlation matrix as that in Figure 4.2(b). In this context, the hierarchical representation cannot achieve an \( O(n \log n) \) memory growth because the partition is fine and the number of low-rank blocks is much higher than \( n \). Figure 4.3(b) compares the memory growths of the two representations, both of which truncate at a relative error of \( 1e-5 \). Similar to Figure 4.2, the approximated correlation matrix changes in terms of the dimensions of the grid and the stationarity of the correlation function. The SKP representation costs significantly less memory and \( k \) grows even slower than \( \log n \) whereas the hierarchical representation has a memory footprint of one magnitude higher. The block Toeplitz structure is valid when the correlation function is stationary. However, its cost is \( s_1^2 s_2 \sim O(n^{3/2}) \), which is less scalable than the hierarchical and SKP representations.
Figure 4.3: (a) An illustration of the hierarchical representation of the covariance matrix used in Figure 4.2(b), built with the standard admissibility condition and truncated at a relative error of $10^{-5}$. The green blocks are low-rank while the red ones are dense. The number denotes the rank of the block. (b) The ratio between the number of double precision values allocated and the number of spatial locations under the hierarchical representation (blue, top four) and the SKP representation (brown, bottom four). The spatial locations are on a $s_1 \times s_2$ grid, where $s_1 = s_2$ for ‘×’ and ‘△’ shaped dots and $s_1 = \frac{5}{16} s_2$ for ‘+’ and ‘◦’ shaped dots. The spatial correlation matrix is built with Equation (4.1) for ‘◦’ and ‘△’ cases while with Equation (4.2) for ‘+’ and ‘×’ cases. The black dotted line is $\text{ndbl} = s_1^2 s_2$, denoting the growth of the block Toeplitz representation.

4.3.3 On perturbed grids

With the quasi-optimal SKP representation, the number of Kronecker products required on a regular grid is small under both stationary and non-stationary correlation functions. For example, a one-million dimensional correlation matrix can be approximated at a precision of $10^{-5}$ with only $k = 15$. However, the linear dependence between the blocks of $\Sigma$ becomes weaker when spatial locations are randomly distributed. I show that the block-wise similarity is still strong but the convergence rate becomes slower on a random geometry. In my experiments, I use a perturbed grid as a surrogate for the random geometry to avoid singularity and inherit the $y$-major
Figure 4.4: (a) The convergence of relative error with the number of Kronecker products. The approximated matrix is built with the exponential (Equation (4.1)) or the non-stationary (Equation (4.2)) correlation function on a regular or a perturbed 32 × 32 grid in the unit square. (b) The number of Kronecker products needed to reach a relative error of 1e−1, 1e−2, 1e−3, 1e−4, and 1e−5. The approximated matrix is built with the exponential (Equation (4.1)) correlation function on a √n × √n perturbed grid in the unit square.

Figure 4.4 compares the convergence rate for the regular grid and the perturbed grid. The perturbed grid adds a perturbation of $\frac{U}{\sigma_2}$ to the x and y coordinates of each location on the regular grid, where $U \sim U(0.0, 0.8)$. The relative errors are below 10% with only two Kronecker products, which indicates a strong similarity between the blocks. However, the perturbed grid has a much lower convergence rate to zero, requiring a large $k$ to achieve the relative error of 1e−5. This demonstrates that the linear dependence between the blocks is weak on a random geometry. Strong block-wise similarity and weak linear dependence can seemingly be paradoxical but one example is that all matrix blocks are equal to one common block added with some independent and identical distributed random noise of a much smaller magnitude. Because of the increase in the Frobenius norm of the correlation matrix, the $k$ needed for a relative error of 1e−2 even decreases with $n$ but this may not be an evidence for practicability. The correlation matrix often becomes closer to singular when $n$ increases and requires an even higher approximation accuracy to stay positive definite.
For example in Figure 4.4, the unit distance becomes smaller whereas $\beta$, the spatial range, remains the same, which produces a correlation closer to one as more spatial locations are included. Overall, I think the SKP is not suitable for the random geometry.

4.4 Cholesky factorization under SKP

The construction of covariance matrices is usually the first step in statistical applications. Matrix addition, multiplication, and factorization are involved in the majority of statistical methods. Structured representations benefit from reduced storage costs but can be less intuitive in terms of the aforementioned matrix operations. In this section, I discuss the feasibility of the Cholesky factorization under the SKP representation, focusing on its scalability and robustness.

4.4.1 Basic ideas and notations

Given a SKP representation, each block of $\Sigma$ can be approximated by a linear combination of the same set of blocks, $\{V_i\}_{i=1}^k$. I refer to $\{V_i\}_{i=1}^k$ as the base blocks of $\Sigma$, whereas $\{U_i\}_{i=1}^k$ can be interpreted as the coefficients with respect to $\{V_i\}_{i=1}^k$. Recall that I define the inner product between two matrix blocks the same as that between the two vectorized matrix blocks in Section 4.2. When $\{V_i\}_{i=1}^k$ are orthonormal, the Frobenius norm of the $(i_b, j_b)$-th block is directly $\|\Sigma_{i_b, j_b}\|_F = \sqrt{\sum_{i=1}^k U_i(i_b, j_b)^2}$.

Three SKP representations are involved in the factorization algorithm, i.e., $\Sigma$, the lower Cholesky factor, $L$, and the inverse of $L$, $D$. I use the following notations to distinguish the three:

$$
\Sigma \approx \sum_{i=1}^{k\Sigma} U_i^\Sigma \otimes V_i^\Sigma, \quad L \approx \sum_{i=1}^{kL} U_i^L \otimes V_i^L, \quad D \approx \sum_{i=1}^{kD} U_i^D \otimes V_i^D.
$$

Here, I assume $m_1 = n_1$ and $m_2 = n_2$ to align with Cholesky factorization.
Algorithm 4.2 Block Cholesky Algorithm

1: procedure BLKCHOL($\Sigma$) 
2:    for $j_b = 1 : m_1$ do 
3:        if $j_b > 1$ then 
4:            for $i_b = j_b : m_1$ do 
5:                $\Sigma_{i_b,j_b} \leftarrow \Sigma_{i_b,j_b} - L_{i_b,1:j_b-1}L_{j_b,1:j_b-1}^T$ 
6:            end for 
7:        end if 
8:        $L_{j_b,j_b} \leftarrow \text{cholesky} (\Sigma_{j_b,j_b})$ 
9:        $D_{j_b,j_b} \leftarrow \text{inverse} (L_{j_b,j_b})$ 
10:       for $i_b = (j_b + 1) : m_1$ do 
11:            $L^\Sigma_{i_b,j_b} \leftarrow \Sigma_{i_b,j_b}D_{j_b,j_b}^T$ 
12:       end for 
13:    end for 
14: end procedure 

The fundamental idea is a block Cholesky factorization that recycles the matrix multiplication between the base blocks while discovering new base blocks. Algorithm 4.2 describes the block Cholesky algorithm, where the colored segments correspond to those in Algorithm 4.3 with respect to their functionalities. Matrix multiplication is involved in Steps 5 and 11 and I use Step 11 as an example of recycling matrix multiplications:

$$
\Sigma_{i_b,j_b}D^T_{j_b,j_b} = \left( \sum_{l_1=1}^{k^\Sigma} U_{l_1}^{\Sigma}(i_b, j_b) V_{l_1}^{\Sigma} \right) \left( \sum_{l_2=1}^{k^D} U_{l_2}^{D}(j_b, j_b) V_{l_2}^{D^T} \right)
= \sum_{l_1=1}^{k^\Sigma} \sum_{l_2=1}^{k^D} U_{l_1}^{\Sigma}(i_b, j_b) U_{l_2}^{D}(j_b, j_b) V_{l_1}^{\Sigma} V_{l_2}^{D^T}. \quad (4.3)
$$

Thus, $V_{l_1}^{\Sigma} V_{l_2}^{D^T}$ can be stored for later iterations. The complete set of base blocks for $D$ and $L$ are initially unknown but discovered through the computation in Steps 9 and 11 respectively. The Schur complement computed through Step 5 is designed to share the same set of base blocks as the initial covariance matrix and hence, the set of base blocks for the input $\Sigma$ is augmented to account for its Schur complement.

The second technique from which I create savings is a relative coordinate system.
Equation (4.3) indicates that each off-diagonal block in $L$ can be stored by $k^\Sigma k^D$ coordinates with respect to $S = \{V_i^{\Sigma}V_j^{D^T}, l_1 = 1, \ldots, k^\Sigma, l_2 = 1, \ldots, k^D\}$. It is worth noting that here, $k^\Sigma$, $k^L$, and $k^D$ are interpreted as the current number of base blocks and increased by one when a new base block is introduced. Therefore, all base blocks for the off-diagonal part of $L$ belong to the linear space spanned by $S$. Similarly, the additional base blocks needed for the Schur complements (Step 5) besides the base blocks for the input $\Sigma$ are in the linear space spanned by $H = \{V_i^{L}V_j^{L^T}, l_1 = 1, \ldots, k^L, l_2 = 1, \ldots, k^L\}$. However, neither $S$ or $H$ is orthonormal, which poses difficulty for projection and norm computation. I further apply a rank-revealing QR decomposition (Chan, 1987) to the vectorized $S$ and $H$ to transform the coordinates under $S$ and $H$ into a cartesian system. Denote:

$$S = Q^S R^S = \begin{bmatrix} \text{vec}(V_1^{\Sigma}V_1^{D^T}) & \ldots & \text{vec}(V_1^{\Sigma}V_k^{D^T}) & \ldots & \text{vec}(V_k^{\Sigma}V_k^{D^T}) \end{bmatrix}, \quad (4.4)$$

$$H = Q^H R^H = \begin{bmatrix} \text{vec}(V_1^{L}V_1^{L^T}) & \ldots & \text{vec}(V_1^{L}V_k^{L^T}) & \ldots & \text{vec}(V_k^{L}V_k^{L^T}) \end{bmatrix}. \quad (4.5)$$

Multiplication with the $R$ factor leads to the coordinates relative to the $Q$ factor, whose columns are a set of orthonormal basis. By definition, the $Q$ and $R$ factors grow with the base block augmentation, where a new base block is added to $\{V_i^{\Sigma}\}_{i=1}^{k^\Sigma}$, $\{V_i^{L}\}_{i=1}^{k^L}$, or $\{V_i^{D}\}_{i=1}^{k^D}$. To include $\{\text{vec}(V_i^{\Sigma})\}_{i=1}^{k^\Sigma}$ in the column space of $Q^H$, I use the vectorized base blocks of the input $\Sigma$ as the initial state of $Q^H$. Hence, we can write:

$$\begin{bmatrix} \text{vec}(V_1^{\Sigma}) & \text{vec}(V_2^{\Sigma}) & \ldots & \text{vec}(V_k^{\Sigma}) \end{bmatrix} = Q^HC^\Sigma, \quad (4.6)$$

$$\begin{bmatrix} \text{vec}(V_1^{L}) & \text{vec}(V_2^{L}) & \ldots & \text{vec}(V_k^{L}) \end{bmatrix} = Q^SC^L, \quad (4.7)$$

where $C$ is the coordinate matrix. Thus, after computing the coordinates of the new blocks, $\Sigma_{ib,jb}$ and $L_{ib,jb}$, under $S$ and $H$ (Steps 5 and 11), I transform them with the $R$ factor. The transformed coordinate vectors are then projected onto the existing base
blocks represented by the columns of $C$ and the resolutes are normalized to append $C$ if their norm is above the tolerance. This base block augmentation is similar to the Gram-Schmidt orthogonalization method (Björck, 1994) that produces orthonormal columns in $C$. It is worth mentioning that $R^H$ is not upper triangular because the initial setup of $Q^H$ is not empty but the later-developed Cholesky factorization has no structural requirement for $R^H$.

The third technique employed is the projection of the Gaussian mixture, as described in Martinsson (2011), of the blocks onto the base blocks. Using Step 11 as an example, $m_1 - j_b$ blocks are projected onto $C^L$ sequentially in the MGS framework but any projection order will suffice as these computations are independent. However, the number of base blocks discovered, which is also the numerical rank of the vectorized blocks, can be affected by the order of the projections. For example, consider applying the MGS algorithm to the columns of

$$
\begin{bmatrix}
1 & 1e^{-3} & 2 \\
1e^{-3} & 1 & 1 \\
1e^{-3} & 1e^{-3} & 1e^{-3}
\end{bmatrix}
$$

from left to right. The numerical rank is three at the truncation level of $1e^{-3}$ but becomes two if the first and third columns are switched. To achieve the smallest number of base blocks, I apply the random sampling method (Halko et al., 2011) to the blocks to be projected with an extra sampling size of 10 (Martinsson, 2011). The samples are then sequentially projected and the resolutes are appended to existing base blocks if they exceed an error tolerance. The random sampling method, in some sense, projects the computed blocks ‘simultaneously’, which tends to achieve a lower numerical rank than projecting them sequentially. Additionally, random sampling distributes the approximation error more evenly across blocks, whereas sequential projection has no error for blocks whose resolutes are used as new base blocks, leaving
the error distribution a point mass at zero.

### 4.4.2 Algorithm and complexity

Algorithm 4.3 outlines the workflow of Cholesky factorization under the SKP representation, featuring the relative coordinate system and the discovery of new base blocks. The relative coordinate system complicates the algorithm but the functionalities of the colored segments essentially match those in Algorithm 4.2. The input SKP representation of $\Sigma$ can be efficiently produced by the ACA framework described in Algorithm 4.1 that already includes an MGS routine to orthogonalize $\{V_1^\Sigma\}_{i=1}^{k^\Sigma}$. The updates in Steps 14, 24, and 30 are based on the definitions from Equations (4.4), (4.5), (4.6), and (4.7), where the implementation details are left to readers. The projection of the diagonal blocks in $L$ (Steps 35 to 37), separated from off-diagonal blocks, guarantees that $\{\text{vec}(V_i^L)\}_{i=1}^{k^L}$ falls in the column space of $S$ until Step 34.

The overloaded subroutine PROJECT for projection and discovery of new base blocks is described in Algorithms 4.4 and 4.5. Algorithm 4.4 accepts a group of coordinates relative to the columns of $Q$, stored in the columns of $A$, whereas Algorithm 4.5 is given a new block of the same dimensions as the base blocks. The random sampling technique is applied at Step 2 in Algorithm 4.4, where $A$ is right-multiplied with a $(kA + 10)$-column matrix $G$. Here, $kA$ is the numerical column rank of $A$ and $G$’s coefficients are independent and identically generated from $N(0,1)$. Actual implementation sets upper limits for $k^\Sigma$, $k^L$, and $k^D$, which are used in place of $kA$. The modified Gram-Schmidt subroutine sequentially projects onto the columns of the second argument with the previous resolute. The inner products from all projections are sequentially stored in $u$ and the final resolute is in $v$. The vector norm involved is $L_2$-norm, equal to the Frobenius norm of the matricized vector. Here, $\epsilon$ denotes the error tolerance, above which a new base block is discovered. A proper choice of $\epsilon$ may depend on the dimension of $\Sigma$ and the upper limits for $k^\Sigma$, $k^L$, and $k^D$. 


Algorithm 4.3 Cholesky factorization under the SKP representation

1: procedure KCHOL(\(\{U_i^l\}_{i=1}^{k^E}, \{V_i^l\}_{i=1}^{k^E}\) )
2: Initialize \(Q^H = [\text{vec}(V^l) \cdots \text{vec}(V_{k^E}^l)]\), \(R^H\), \(Q^S\) and \(R^S\) as empty matrices
3: Initialize \(C^E = I_{k^E}\) and \(C^L\) as an empty matrix
4: Initialize \(\{L_{i,i}\}_{i=1}^{m_i}\) and \(\{D_{i,i}\}_{i=1}^{m_i}\) as empty matrices
5: for \(j_b = 1 : m_1\) do
6:   if \(j_b > 1\) then
7:     Initialize temporary storage \(A \in \mathbb{R}^{k^E \times k^E, m_1 - j_b + 1}\)
8:     for \(i_b = j_b : m_1\), \(l_1 = 1 : k^L\), \(l_2 = 1 : k^L\) do
9:       \(l_3 \leftarrow (l_1 - 1)k^L + l_2\)
10:      \(A(l_3, i_b + 1 - j_b) \leftarrow U_{l_1}^L(l_3, i_b + 1 - j_b - 1)\)
11:     end for » Each block in \(L_{j_b,m_1:j_b-1}^L, L_{j_b,j_b-1}^L\) ← each column in \(HA\)
12:      \(A \leftarrow R^H A, \text{PROJECT}(A, Q^H, C^E, \{U_i^l\}_{i=1}^{k^E}, \{V_i^l\}_{i=1}^{k^E}, k^E)\)
13:      \(A \leftarrow C^E A\)
14:      Compute new columns in \(S\) and update \(Q^S\), \(R^S\), and \(C^L\)
15:   end if
16:   Initialize \(A \in \mathbb{R}^{k^E, m_1 - j_b + 1}, A \leftarrow 0\)
17: end for
18: for \(l = 1 : k^E\) do
19:      \(A(l, 1 : m_1 - j_b + 1) \leftarrow U_{l}^E(j_b : m_1, j_b) - A(l, 1 : m_1 - j_b + 1)\)
20: end for » A: the coefficients of the Schur complement relative to \(\{V^l_i\}\)
21: Initialize \(B \leftarrow \sum_{l=1}^{k^E} A(l, 1) V_{l}^E\)
22: \(L_{j_b,j_b} \leftarrow \text{cholesky}(B), D_{j_b,j_b} \leftarrow \text{inverse}(L_{j_b,j_b})\)
23: \(\text{PROJECT}(D_{j_b,j_b} \{U_{i}^{D}\}_{i=1}^{k^D}, \{V_{i}^{D}\}_{i=1}^{k^D}, k^D, j_b)\)
24: Compute new columns in \(S\) and update \(Q^S\), \(R^S\), and \(C^L\)
25: Reset \(B \in \mathbb{R}^{k^E, k^D, m_1 - j_b}\)
26: for \(i_b = j_b + 1 : m_1\), \(l_1 = 1 : k^E\), \(l_2 = 1 : k^D\) do
27:     \(l_3 \leftarrow (l_1 - 1)k^D + l_2\)
28:     \(B(l_3, i_b + 1 - j_b) \leftarrow A(l_3, i_b + 1 - j_b - 1)U_{l_1}^D(j_b, j_b)\)
29: end for » Each block in \(\sum_{j_b+1}^{m_1} L_{j_b,j_b}^D\) ← each column in \(SB\)
30: \(B \leftarrow R^SB, \text{PROJECT}(B, Q^S, C^L, \{U_{i}^{L}\}_{i=1}^{k^L}, \{V_{i}^{L}\}_{i=1}^{k^L}, k^L), B \leftarrow C^LT B\)
31: Compute new columns in \(H\) and update \(Q^H, R^H, \text{and } C^E\)
32: for \(l = 1 : k^L\) do
33:     \(U_{l}^E(j_b + 1 : m_1, j_b) \leftarrow B^T(l, 1 : m_1 - j_b)\)
34: end for » B: the coefficients of the Cholesky factor relative to \(\{V_i^L\}\)
35: for \(j_b = 1 : m_1\) do
36:     \(\text{PROJECT}(L_{j_b,j_b}^S, \{U_{i}^{L}\}_{i=1}^{k^L}, \{V_{i}^{L}\}_{i=1}^{k^L}, k^L, j_b)\)
37: end for » Project dense diagonal blocks to \(\{V^L_i\}_{i=1}^{k^L}\)
38: return \(\{U_{i}^{L}\}_{i=1}^{k^L}, \{V_{i}^{L}\}_{i=1}^{k^L}, \{U_{i}^{D}\}_{i=1}^{k^D}, \{V_{i}^{D}\}_{i=1}^{k^D}\)
39: end procedure
Algorithm 4.4 Discover new base blocks I

1: procedure PROJECT(A, Q, C, \{U_i\}_{i=1}^k, \{V_i\}_{i=1}^k, k)
2: Initialize temporary storage B ← random column sampling(A)
3: for j = 1 : ncol(A) do
4:    u, v ← modified Gram-Schmidt(A(:, j), C)
5:    if \|v\| > \epsilon then
6:        C_1 ← [C, v/\|v\|], U_{k+1} ← 0, vec(V_{k+1}) ← Qv, k ← k + 1
7:    end if
8: end for
9: end procedure

Algorithm 4.5 Discover new base blocks II

1: procedure PROJECT(A, \{U_i\}_{i=1}^k, \{V_i\}_{i=1}^k, k, j)
2:    u, v ← modified Gram-Schmidt(vec(A), [vec(V_1), ..., vec(V_k)])
3:    for l = 1 : k do
4:        U_l(j, j) ← u(l)
5:    end for
6:    if \|v\| > \epsilon then
7:        U_{k+1} ← 0, U_{k+1}(j, j) ← 1, vec(V_{k+1}) ← Qv, k ← k + 1
8:    end if
9: end procedure

Major complexity estimates for Algorithm 4.3 are shown in Table 4.2, where the complexity is aggregated across the outer iterations. Here, \(k_{\text{max}}\) denotes the maximum of the upper limits for \(k^\Sigma\), \(k^L\), and \(k^D\). Therefore, these values serve as the upper bounds of the complexity orders. Assuming \(k_{\text{max}} \leq m_1^{1/2} \leq m_2^{1/2}\), the total complexity of Cholesky factorization is dominated by dense Cholesky factorization and triangular matrix inversion on the diagonal blocks, which reaches an optimal order of \(O(n^2)\) when \(m_1 = m_2\). This assumption is reasonable because \(m_2\) can be chosen as the factor of \(s_2\) closest to but no smaller than \(\sqrt{n}\) as discussed in Section 4.3.1 and \(k_{\text{max}}\) is typically below 20 for \(n\) in hundreds of thousands of dimensions. Due to

<table>
<thead>
<tr>
<th>Step</th>
<th>Projection</th>
<th>Schur Complement</th>
<th>S and H</th>
<th>Diagonal</th>
</tr>
</thead>
<tbody>
<tr>
<td>Complexity</td>
<td>(O(k_{\text{max}}^3(m_1^2 + m_2^2)))</td>
<td>(O(k_{\text{max}}^2m_1^3))</td>
<td>(O(k_{\text{max}}^2m_2^3 + k_{\text{max}}^4m_2^2))</td>
<td>(O(m_1m_2^3))</td>
</tr>
</tbody>
</table>
the nature of Cholesky factorization, Algorithm 4.3 has limited parallel capacity. However, the matrix multiplication for constructing $S$ and $H$, which amounts to $O(k_{\text{max}}^2 m_3^3)$ complexity, can be directly parallelized.

4.4.3 Numerical results under expanding domain

The typical challenge for low-rank Cholesky factorizations is guaranteeing positive-definiteness of the Schur complement. Xia and Gu (2010) introduced a robust factorization for HSS matrices through Schur compensation, which truncates the orthogonal factorization of the off-diagonal block. However, this technique is not applicable under the SKP representation. As shown in Step 5 of Algorithm 4.2, the error of $L_{ib,1:jb-1}^T L_{jb,1:jb-1}$ cumulates with $j_b$ because truncation is involved for each block of $L$. When $m_1$ becomes larger, $j_b$ increases from 1 to $m_1$, during which this error may cumulate to the point of breaking the positive-definiteness of $\Sigma_{ib,jb}$. Therefore, the requirement for accuracy becomes higher when $n$, as well as $m_1$ and $m_2$, increase.

In the previous experiments, I considered geometries in the unit square but in this case, the correlation between neighbors can be arbitrarily strong as $n$ grows, which further reduces the error tolerance. For example, when $s_1 = s_2 = 2^{10}$ and $\beta = 0.1$ in the exponential covariance function, the correlation between two neighbor locations is more than 0.99 and the Schur complement’s magnitude is two orders smaller than that of the correlation matrix, where a small truncation error on the computed part of $L$ may lead to the break-down of its positive-definiteness.

In this section, I apply Algorithm 4.3 to correlation matrices in the context of an expanding domain to keep the magnitudes of $k^\Sigma$, $k^L$, and $k^D$ small. In the following section, I introduce a correction scheme to improve the robustness of the algorithm. Here, the domain is considered to be a regular grid with a unit distance of 1/32. I also consider only $s_1 = s_2$ because it already contains the Cholesky factor for the cases where $s_1 < s_2$. A stationary kernel, Equation (4.11) with $\beta_0 = 0.1$, and a
Figure 4.5: Time costs for Cholesky factorization under the SKP representation. The exponential and the non-stationary correlation functions from Equation (4.1) and Equation (4.2) are used for building the correlation matrix. The same algorithm is run over 8 threads to show the parallel capacity.

non-stationary kernel, Equation (4.2) with $\beta_i = 0.03 + 0.07 \times x_i$, are used to build the correlation matrix. Figure 4.5 shows the time costs for factorizing the SKP representation of the correlation matrix with single thread and eight threads.

The factorization in one million dimensions takes 450 seconds with 8 threads while 950 seconds with a single thread. Hence, more than half of the total work benefits from parallelism. I also compute the relative error defined in Section 4.3.1 for the Cholesky factors up to $2^{14}$ dimensions and the errors range from 1.5% to 1.6% for all cases, which is $\sqrt{n}$-times the average coefficient-wise error. When $n = 2^{14}$, the average coefficient-wise error is smaller than $1.6e-4$. The tolerance for discovering new base blocks, $\epsilon$, is set at $1e-2$ if $n \leq 2^{16}$ and at $5e-3$ otherwise. It is worth noting that I impose upper limits for $k^\Sigma$, $k^L$, and $k^D$ to keep computation cost under control and for most cases, the upper limits for $k^\Sigma$ and $k^L$ are reached. The upper limits for $k^\Sigma$, $k^L$, and $k^D$ are $(20, 20, 5)$ if $n \leq 2^{16}$ and $(25, 25, 6)$ otherwise.
4.4.4 Correction mechanism for a fixed domain

The Schur complement is the conditional covariance matrix given the spatial variables in previous diagonal blocks. Each substractions in computing the Schur complement, 
\[ \Sigma_{i_b} - \sum_{j_b=1}^{i_b-1} L_{i_b,j_b} L_{i_b,j_b}^T, \]
indicates the conditioning on an additional block of spatial variables. Hence, we can reduce the number of substractions through conditioning on only the closest blocks of spatial variables. The correction mechanism I propose is that when the Cholesky factorization fails for \((\Sigma_{i_b} | z_{1:i_b-1})\), compute \((\Sigma_{i_b} | z_{i_b-k_0:i_b-1})\) instead, where \(k_0 \geq 1\) defines the number of blocks that \(\Sigma_{i_b}\) is conditioned on, and use it to proceed from Step 21 in Algorithm 4.3. Here, \(z_{i_b}\) denotes the spatial variables in the \(i_b\)-th diagonal block. Similarly, the Schur complement for later block columns are conditioned from \(z_{i_b-k_0}\) until the previous block of spatial variables, where \(\hat{i}_b\) is the diagonal block index to which the previous failure happens.

Another interpretation of this correction mechanism is that I restart the Cholesky factorization for \(\Sigma_{i_b-k_0:m_1,i_b-k_0:m_1}\) with pre-computed base blocks \(\{V^\Sigma_i\}_{i=1}^{k_\Sigma}, \{V^L_i\}_{i=1}^{k_L}, \) and \(\{V^D_i\}_{i=1}^{k_D}\), but return only \(L_{i_b:m_1,i_b:m_1}\). This mechanism works well for stationary correlation kernels because the computation at each restart point is a subset of that at the previous restart point and hence, existing base blocks can be reused. However, this is not valid for the non-stationary correlation function of Equation (4.2) since the correlation strength changes from left to right along the grid. If I limit the magnitudes of \(k^\Sigma, k^L, \) and \(k^D\), new base blocks are no longer added at later restart points and this correction mechanism may cause significant error or even failure.

Numerical experiments on the fixed domain are performed with larger upper limits for \(k^\Sigma, k^L, \) and \(k^D\) than those in Section 4.4.3. Here, the unit distance is \(1/s_1 = 1/\sqrt{n}\), the tolerance \(\epsilon\) is \(1e^{-2}\) if \(n < 2^{14}\) and \(1e^{-3}\) otherwise, and the upper limits for \(k^\Sigma, k^L, \) and \(k^D\) are \((20, 20, 5)\) if \(n < 2^{14}\) and \((30, 30, 10)\) otherwise. Table 4.3 describes the relative error of the computed Cholesky factor up to \(2^{14}\) dimensions and the time cost of the factorization in one million dimensions with 8 cores. The relative
Table 4.3: Relative error and time cost for Cholesky factorization of the exponential and non-stationary kernel matrices. Here, $n$ spatial variables are located on a regular grid in the unit square.

<table>
<thead>
<tr>
<th></th>
<th>Err for $n = 2^{10}$</th>
<th>Err for $n = 2^{12}$</th>
<th>Err for $n = 2^{14}$</th>
<th>Time for $n = 2^{20}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exponential</td>
<td>1.6%</td>
<td>4.1%</td>
<td>3.7%</td>
<td>596s</td>
</tr>
<tr>
<td>Non-stationary</td>
<td>1.3%</td>
<td>1.8%</td>
<td>8.0%</td>
<td>N.A.</td>
</tr>
</tbody>
</table>

error for the stationary kernel can be controlled through an increased number of base blocks. However, under the non-stationary kernel, increasing the limits of $k^S$, $k^L$, and $k^D$ from $(20, 20, 5)$ to $(30, 30, 10)$ is insufficient for keeping the relative error at the same level. In fact, the correction mechanism fails for $n = 2^{16}$ and above with the upper limits of $(30, 30, 10)$. Therefore, I conclude that Cholesky factorization for non-stationary kernels in the fixed domain requires a higher number of base blocks, which rapidly brings up the computation cost.

The fixed domain with increasing density, in general, is a challenge for low-rank methods, including the tile-low-rank structure (Weisbecker, 2013; Mary, 2017; Akbudak et al., 2017) and the hierarchical structure (Hackbusch, 2013). The robust factorization in Xia and Gu (2010) is only applicable to the HSS structure and the construction of the HSS matrix becomes more expensive under higher density. In this chapter, I consider only up to one million locations in the fixed unit square.

### 4.5 Simulation of large Gaussian random fields

One representative application of the Cholesky factor is the simulation of Gaussian random fields, which has been broadly used for statistical inference and stochastic generators (e.g., Castruccio and Genton (2018)). More efficient and less constrained simulation methods are constantly sought after to meet increasing demands. To my knowledge, the most efficient method currently is the circulant embedding method (Dietrich and Newsam, 1995, 1997) that embeds the simulation domain into a torus lattice and has a total complexity of $O(n \log n)$. Its drawback is the requirement
that the circulant embedding should be nonnegative definite but for two-dimensional simulations, unfortunately, this is not frequently the case (Gneiting et al., 2006). Gneiting et al. (2006) compared intrinsic embedding with cut-off embedding and concluded that the former could embed broader covariance functions at the cost of losing stationarity. However, both were unable to embed smooth covariance functions, for example, the Matérn covariance function with smoothness greater than 0.5, which can be needed in real applications (North et al., 2011). Rue and Held (2005) proposed a nested dissection method that has a complexity of $O(n^{3/2})$ for factorization and $O(n \log n)$ for simulation but this method depends on the Markov structure.

Based on the Cholesky algorithm in Section 4.4, I propose a method to simulate Gaussian random fields on a regular grid that accommodates broader covariance functions than the circulant embedding method and does not depend on the Markov structure. The method I propose has a complexity of $O(n^2)$ for factorization and $O(n^{3/2})$ for simulation. Although they are higher than state-of-the-art methods, this method is arguably more flexible and has proved to achieve the factorization in one million dimensions in under 600 seconds on a standard scientific workstation while the time cost for simulation is trivial compared with factorization. Here, I demonstrate this method by simulating $1024 \times 1024$ dimensional Gaussian random fields. In addition to the exponential kernel, I use the Whittle kernel with the same range parameter $\beta = 0.1$ as an example for smooth covariance functions. It is worth mentioning that the Whittle kernel has an effective range of 0.4 and gives a correlation of 0.9997 at the unit distance in the fixed domain described in Figure 4.6 and hence, I add a nugget effect of $1e{-2}$ to the Whittle kernel to reduce singularity.

Figure 4.6 shows the statistics of simulated Gaussian random fields. The upper limits for $k^\Sigma$, $k^L$, and $k^D$ are still $(30, 30, 10)$, under which the time cost for simulating one replicate with 8 cores is less than 5 seconds. In fact, all operations involved in simulating the Gaussian random field are level 1 or level 2 basic linear algebra
Figure 4.6: Mean, variance and empirical semivariogram of spatial variables located on a $1024 \times 1024$ regular grid from a Gaussian random field. Each plot is based on 100 replicates. The simulation parameters are domain types (fixed domain or expanding domain) and correlation structures (exponential correlation function or Whittle correlation function). The unit distance is $1/1024$ for the fixed domain and $1/32$ for the expanding domain. Both correlation functions have a range parameter $\beta = 0.1$. (a) and (b) are the sample mean and sample variance boxplots, where the red dashed lines mark the underlying truth; (c) and (d) are the functional boxplots of the empirical semivariogram under the Whittle correlation function with fixed and expanding domains. The true underlying variogram is the dashed green line.

Subprograms (BLAS) ([Blackford et al., 2002]), which indicates further improvement if the procedures are integrated into a batch simulation with level 3 BLAS. The mean estimator has a zero expectation and a variance equal to the coefficient-wise average of $\Sigma_n \frac{1}{n^2} \mathbf{1}_n \Sigma \mathbf{1}_n$ with $\mathbf{1}_n^\top = (1, \ldots, 1)_{1 \times n}$, which is not asymptotically zero under the fixed domain. Since the Whittle kernel is bigger than the exponential kernel in $(0, \sqrt{2})$, the mean estimator under the former has a larger variance when the domain is fixed. The expectation of the variance estimator is $1 - \text{Var}(\bar{z})$, where $\bar{z}$ is the mean estimator and
due to the same reason, the empirical mean of the variance estimator is smaller under the Whittle kernel. The semivariogram is computed by randomly sampling $10^8$ pairs of spatial variables due to the large size of full sampling. The empirical semivariogram of the exponential kernel is similar to that of the Whittle kernel and not listed. The mean curve of the functional boxplots (Sun and Genton, 2011) is consistent with its true value of $1.01 - \mathcal{K}(\|x\|)$, where $\mathcal{K}(\cdot)$ denotes the correlation function. I conclude that the Cholesky factor computed from Algorithm 4.3 provides a scalable option for simulating Gaussian random fields under a wide range of stationary covariance functions. Furthermore, Algorithm 4.3 may lay the groundwork for SKP-based linear algebra, leading to broader matrix operations and statistical applications.

4.6 Chapter summary

This chapter introduced a novel sum of Kronecker products (SKP) representation for spatial covariance matrices from a regular grid. This representation has a $O(nk)$ storage cost and can be built with $O(nk^2)$ floating-point operations with the ACA framework, where $k$ is the number of Kronecker products in the sum. Stationary and non-stationary kernels were tested and proven feasible with this representation. Additionally, the indexing of spatial locations can be simply along one dimension, which I refer to as the $y$-major indexing, without the need for a complex spatial partitioning scheme to guarantee separability. This feature would make the SKP representation applicable to grids in 3D or even higher dimensions without much adaptation. I also designed a Cholesky factorization algorithm for the SKP representation as an example for matrix operations. Here, the overarching idea is that both the covariance matrix and its Cholesky factor can be closely approximated with a small number of Kronecker products. The factorization of a one million-dimensional covariance matrix can be achieved within 600 seconds on a standard scientific workstation. A study was provided to show that the factorization algorithm could be used to simulate large
Gaussian random fields for a wide range of spatial covariance kernels.

One limitation of this new representation is that it depends on the regular grid assumption, without which the approximation error decreases slowly with the number of Kronecker products. Hence, it may be less general than hierarchical matrices (Hackbusch, 2015) or HSS matrices (Chandrasekaran et al., 2005). The Cholesky factorization algorithm is sensitive to the singularity condition of the covariance matrix. The factorization for a fixed domain and an increasing number of locations may fail at a certain variable density level. I proposed a correction mechanism that made the factorization feasible for stationary covariance kernels. More robust correction mechanisms may be developed for non-stationary kernels. The Kronecker product is not directly compatible with matrix multiplication, due to which the SKP representation has higher complexity for Cholesky factorization and Gaussian random field simulation than other state-of-the-art methods. The superlinearity is not directly attacked by the algorithms of this chapter, but I point out that matrix multiplication within my framework is amenable to massive hybrid distributed-shared memory parallelism.
Chapter 5

The tlrmvnmvt R Package

5.1 R and R packages for MVN/MVT probabilities

Despite that the library containing the methods discussed in Chapter 3 has been developed in C++, the fact that the interfaces of the functions exist only in C++ may prevent many practitioners from using the library and hence, the developed scalable methods for computing MVN and MVT probabilities. The difficulty from compiling the library, formatting the input data, and connecting other parts of the overall application to the library is the main challenge for the library to become more broadly accessible. R (R Core Team, 2019) provides a unified data format and language syntax that significantly simplify the integration of functions developed by different authors. It is the most extensively used language among statisticians and field experts, whose work involves statistics. Therefore, I modify my existing library that implements the TLR methods with the recursive block reordering into an R package, named tlrmvnmvt, and publish it on CRAN. Because the TLR Monte Carlo method is based on Genz (1992), the tlrmvnmvt package also includes an efficient implementation of the cumulative probability functions from the R package, mvtnorm (Genz et al., 2019; Genz and Bretz, 2009).

Currently, there are two dedicated packages in R for the computation of the MVN and MVT probabilities, implementing the methods from Genz (1992) and from Botev (2017), respectively. The mvtnorm package, implementing the Monte Carlo method from Genz (1992), is currently the most popular package (based on the number of re-
verse dependencies) in statistical applications. This package has a relatively efficient implementation, able to estimate a one-thousand-dimensional MVN/MVT probability in tens of seconds. However, it limits the input dimension to no bigger than one thousand and its estimates for tail probabilities are not as accurate as those for others (Botev, 2017). The TruncatedNormal package, based on Botev (2017), improved the Monte Carlo sampling with the minimax tilting technique that significantly increased the convergence rate of the Monte Carlo sampling. With the same number of Monte Carlo samples, the functions in TruncatedNormal produce more accurate results than those in mvtnorm. Additionally, TruncatedNormal provides functions for the simulation of truncated MVN random vectors, which is a challenging problem when the acceptance probability is low. One drawback of the TruncatedNormal package is that the cost of the minimax tilting procedure is shown in Botev (2017) to scale at a cubic rate with the problem size \( n \). Both packages are not ideal for probabilities in more than one thousand dimensions, leaving high-dimensional applications unattended, while the TLR Monte Carlo method with recursive block reordering can handle MVN/MVT probabilities in tens of thousands of dimensions, significantly extending the computation limits of those applications.

The remainder of this chapter is constructed as follows: Section 5.2 introduces the interfaces of the five functions from the package tlrmvnmvt and describes the scenarios where each method should be applied. Section 5.2 compares the performance of tlrmvnmvt with the two alternatives, i.e., mvtnorm and TruncatedNormal. Section 5.3 concludes this chapter.

### 5.2 Package functions

The tlrmvnmvt package has four major functions for computing MVN and MVT probabilities, namely \( pmvn\text{.genz} \), \( pmvt\text{.genz} \), \( pmvn\text{.tlr} \), and \( pmvt\text{.tlr} \), each of which has two interfaces that specify how the covariance matrix is constructed. Additionally,
there is a fifth function, \textit{zorder}, that implements Morton’s order in 2D. This package has an efficient underlying C++ implementation, interfaced through \texttt{Rcpp (Eddelbuettel and Balamuta, 2017)}, and utilizes the Eigen library (Guennebaud et al, 2010), a template library for linear algebra, via \texttt{RcppEigen (Bates and Eddelbuettel, 2013)}. Currently, the Boost library (Schling, 2011) is also linked through the \texttt{BH} package (Eddelbuettel et al, 2019) to provide modified Bessel functions but this dependency can be spared when C++17 is supported by R.

5.2.1 Function interfaces

The functions, \texttt{pmvn.genz} and \texttt{pmvt.genz}, perform the univariate reordering and the Monte Carlo sampling with a dense representation of the covariance matrix. They aim to solve MVN and MVT probabilities in relatively low and moderately high dimensions, e.g., below 4,000. Their interfaces are:

\begin{verbatim}
pmvn.genz(lower=-Inf, upper=Inf, mean=0, Sigma=NULL, geom=NULL, 
        type="matern", para=NULL, uselog2=FALSE, N=499)
\end{verbatim}

\begin{verbatim}
pmvt.genz(lower=-Inf, upper=Inf, nu=NULL, mean=0, Sigma=NULL, geom=NULL, 
        type="matern", para=NULL, uselog2=FALSE, N=499)
\end{verbatim}

The first three arguments are the integration limits, \(a\) and \(b\), and the mean parameter, \(\mu\), which are described in Equations (2.1) and (3.3) except for the mean parameter. The mean parameter directly shifts \(a\) and \(b\) for MVN probabilities but shifts the scaled \(a\) and \(b\) for MVT probabilities; see Genz and Bretz (2009) for details. Each of them should be either a vector of length \(n\) or a scalar if the same value is shared between all the coefficients of the vector. The covariance matrix \(\Sigma\) can be either directly given by the \texttt{Sigma} parameter or constructed by the package through specifying the underlying geometry with \texttt{geom}, the covariance model with \texttt{type}, and the covariance parameters with \texttt{para}. Here, \texttt{geom} should be a matrix with \(n\) rows, each of which is treated as a coordinate vector in the domain, e.g., \(\mathbb{R}^2\). I denote the \(i\)-th row of \texttt{geom} with \texttt{s}_i. Currently, the only supported covariance model is the Matérn covariance function.
that accepts \texttt{para} as a vector of length four, storing the scale parameter $\sigma$, the range parameter $\beta$, the smoothness parameter $\kappa$, and the nugget parameter $\delta$. Specifically, I assume the following parameterization for the Matérn covariance function:

\[
\sigma_{ij} = \sigma^2 \left\{ 2^{\kappa - 1} \Gamma(\kappa) \right\}^{-1} \left( \frac{\|s_i - s_j\|}{\beta} \right)^\kappa H_\kappa \left( \frac{\|s_i - s_j\|}{\beta} \right) + \mathbb{1}_{s_i = s_j} \cdot \delta,
\]

where $\sigma_{ij}$ is the $(i, j)$-th coefficient of $\Sigma$, $\| \cdot \|$ is the $L_2$-norm that leads to the Euclidean distance, and $H_\kappa(\cdot)$ denotes the modified Bessel function of the second kind of order $\kappa > 0$. The $\texttt{nu}$ parameter should be a positive number representing the degrees of freedom that are needed only for MVT probabilities. The last two parameters, $\texttt{uselog2}$ and $\texttt{N}$, determine whether the probability is returned as its logarithm to the base two and how many Monte Carlo samples are used for estimating the integration, respectively. In extreme cases, the probability can be smaller than the machine precision while the logarithm of the probability can still be correctly stored. Note that I use the Richtmyer rule (Richtmyer, 1951) for sampling $w$ that was introduced in Section 5.2, which specifies a batch size, $N$, and the number of batches, $ns$, typically much smaller than $N$. In my implementation, $N$ is set equal to the input $\texttt{N}$ parameter while $ns$ is internally chosen as 20. Therefore, the total number of Monte Carlo samples amounts to $20N$.

Internally, 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 $L$ of $\Sigma$. Next, the MVN and the MVT integrands are computed for each $w$ generated from the Monte Carlo rule through the algorithms defined in Section 3 of Genz (1992) and Section 4.2.2 of Genz and Bretz (2009), respectively. In both algorithms, the probability estimate from one sample is the product of $n$ one-dimensional probabilities and I store the mantissa and exponent of the product after each multiplication separately in one double-precision variable and
one integer variable, respectively. This allows values much smaller than the machine
precision to be correctly represented and is also compatible with the computation
for the mean and the standard deviation. Both functions return a list with four
fields, i.e., the probability estimate, the estimation error, the time for the univariate
reordering, and the time for the Monte Carlo sampling, when useLog2 is FALSE
and Sigma is given. When geom, type, and para are given instead of Sigma, an
extra field of the time cost for constructing the covariance matrix is returned. When
useLog2 is TRUE, the estimation error is not returned because the standard deviation
of the logarithm cannot be directly estimated. However, this is amenable to multiple
multiple instances of Monte Carlo sampling with the same arguments. Chapter 3
shows that the logarithm of the estimated probability has a much smaller relative
error than the estimated probability itself and hence, is preferred in high dimensions.

The functions, pmvn.tlr and pmvt.tlr, perform recursive block reordering and the
Monte Carlo sampling with the TLR representations of the covariance matrix and
the Cholesky factor. They are designed for cases where the off-diagonal blocks of $\Sigma$
have the low-rank feature. When the TLR representation can closely approximate
$\Sigma$, the two functions create computation savings by reducing the costs of Cholesky
factorization and Monte Carlo sampling. Their interfaces are:

```
pmvn.tlr(lower=-Inf, upper=Inf, mean=0, Sigma=NULL, geom=NULL,
           type="matern", para=NULL, useLog2=FALSE, m=64, epsl=1e-3, N=499)
```

```
pmvt.tlr(lower=-Inf, upper=Inf, nu=NULL, mean=0, Sigma=NULL, geom=NULL,
           type="matern", para=NULL, useLog2=FALSE, m=64, epsl=1e-3, N=499)
```

There are two additional parameters compared with their dense-matrix-based coun-
terparts. The $m$ parameter specifies the block (tile) size, $m$, in the TLR representation,
and the $\epsilon$ parameter is the absolute truncation error, $\epsilon$, used in the construction of
the covariance matrix and the low-rank matrix addition. The adaptive cross approx-
imation (ACA) (Bebendorf and Rjasanow, 2003) algorithm is used for constructing
the TLR representation of the covariance matrix while the low-rank matrix addition
involves singular value decomposition (SVD), where the singular vectors corresponding to the singular values smaller than the truncation error are descretized; see Borm et al. (2003) 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 size \( m \times m \) in \( \Sigma \) can be closely approximated by the low-rank representation described in Section 3.3. This requirement motivates a choice of \( m \) depending on the structure of the input \( \text{Sigma} \) or \( \text{geom} \) but generally, the smaller \( m \) is, the more closely the blocks are approximated by the low-rank representation. The \( \text{geom} \) parameter generally conforms to the same format described in the \text{pmvn.genz} and \text{pmvt.genz} functions except that the coordinates for each location should be in the 2D unit square. A future extension of this package will remove this current limitation to allow arbitrary point distributions in two and three spatial dimensions.

For the two TLR functions, when \( \text{geom} \) is given instead of \( \text{Sigma} \), Morton’s order is initially applied as my choice of a geometry-oriented indexing method that preserves locality. Next, the functions construct the TLR covariance matrix based on the geometry, the indices, the covariance kernel, the block size \( m \), and the truncation error, \( \epsilon \). When \( \text{Sigma} \) is given, the construction of the TLR covariance matrix directly reads the coefficients of \( \Sigma \) and hence, needs only the information of \( m \) and \( \epsilon \). After the construction, the two input scenarios share the same procedures that apply the recursive block reordering and compute the integrands for each \( w \) sampled from the unit hypercube in \( n \) dimensions. The recursive block reordering introduced in Chapter 3 produces the TLR Cholesky factor and is shown to improve the convergence rate of the Monte Carlo sampling more than the non-recursive version from Chapter 2. The algorithms for computing the MVN and MVT integrands under the TLR representation of the Cholesky factor are described in Chapter 3. Similar to the two functions based on dense matrices, the probability estimate from each sample \( w \) is stored in a double-precision variable and an integer variable. After computing the
probability estimates from all samples, the functions return the probability estimation and its error estimate (when `uselog2` is `FALSE`) together with four performance indicators, e.g., the times for building the TLR covariance matrix, the recursive block reordering, and Monte Carlo sampling, and the average rank among the off-diagonal blocks.

The last function to introduce from the `tlrmvnmvt` package is `zorder` that implements Morton’s order on the 2D plane:

```r
zorder(geom)
```

`zorder` accepts a set of locations from the `geom` parameter, whose specifications should be the same as those for the `pmvn.tlr` and `pmvt.tlr` functions, and returns the vector that matches the old indices to the new indices, e.g., `geom<-geom[zorder(geom)]`. This function is provided for cases where the TLR covariance matrix cannot be constructed solely with the Matérn covariance function, but the low-rank feature can still be valid when the spatial locations, if any, are properly ordered. For example, the covariance matrix is equal to the summation of two Matérn covariance matrices. In such cases, users can first construct dense covariance matrices based on the order given by `zorder`; then perform the required matrix operations to reach the final covariance matrix; lastly call the `pmvn.tlr` and `pmvt.tlr` functions with the `Sigma` parameter equal to the final covariance matrix. Although it is more time-and-memory consuming to build the dense covariance matrix, its overall cost is typically much less than that of the Monte Carlo sampling. Therefore, `zorder` broadens the class of covariance matrices that can be processed by the TLR methods, in dimensions where the `pmvn.tlr` and `pmvt.tlr` functions are still feasible.

### 5.2.2 Computation with dense matrices

In this section, I present the computation of MVN and MVT probabilities with the dense representation of the covariance matrix. The covariance matrix here is con-
structured 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, $a$ and $b$, are generated from $U(-5, -1)$ and $U(1, 5)$, respectively. I set a fixed seed value for the results to be reproduced.

```r
R> rm(list = ls())
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)),
R> +   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 I will use the dense representation of the covariance matrix, I do not need to reorder the locations in $\text{geom}$. The covariance matrix is constructed with the \texttt{matern} function from the \texttt{geoR} package.

```r
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 (5.1) when $\nu = 1$. By default, the \texttt{uselog2} is \texttt{FALSE} and the estimation error is returned.

```r
R> library(tlrmvnmvt)
R> pmvn.genz(a, b, 0, covM)
```
The output error is the absolute error and the unit for the time measurements is seconds. Similar results can be produced for MVT probabilities, for which I need to define the degrees of freedom.

\texttt{R> nu <- 7}
\texttt{R> pmvt.genz(a, b, nu, 0, covM, uselog2 = T)}

$\text{Estimation}$
[1] -6.914408
$\text{\textquoteleft Univariate reordering time\textquoteright}$
[1] 0.03028711
$\text{\textquoteleft Monte Carlo time\textquoteright}$
[1] 1.234697

The log-probability is computed without an error estimation, which, if desired, can be estimated through multiple computations of the same problem. The same problems can be also computed through specifying the geometry and the covariance structure. Here, I only use the \textit{pmvn.genz} function as an example.

\texttt{pmvn.genz(a, b, 0, NULL, geom, "matern", c(1.0, 0.1, 1.0, 0.0))}

$\text{Estimation}$
[1] 0.0001071391
$\text{Error}$
[1] 3.807667e-06
$\text{\textquoteleft Building covariance matrix\textquoteright}$
[1] 0.01073234
$\text{\textquoteleft Univariate reordering time\textquoteright}$
[1] 0.02170856
$\text{\textquoteleft Monte Carlo time\textquoteright}$
[1] 1.309238
The third field is the time (in seconds) used for building the dense covariance matrix that has a complexity of $O(n^2)$. The probability estimate is very close to the previous estimate of the same problem, where the difference is from the randomness of Monte Carlo sampling. The results also support the validity of the output estimation error.

### 5.2.3 Computation with TLR matrices

In this section, I 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 5.2.2. Next, I provide two higher-dimensional comparisons that highlight the computation efficiency of the TLR methods, where I 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
R> pmvn.tlr(a, b, 0, covM, m = 20)

$Estimation
[1] 0.0001027184

$Error
[1] 4.503308e-06

$‘Building TLR covariance matrix time’
[1] 0.004808933

$‘Recursive block reordering time’
[1] 0.117401

$‘Monte Carlo time’
[1] 1.366923

$‘Average rank’
[1] 8
```

There is no obvious difference between the computation times of the `pmvn.genz` and `pmvn.tlr` functions. In fact, the average rank of the off-diagonal blocks is eight, close to $m/2 = 10$, which indicates little storage and computation savings. The `pmvt.tlr` function also fails to outperform the `pmvt.genz` function in 400 dimensions.
$\text{Estimation}$

\[
\begin{align*}
\text{[1]} & \quad -6.863394 \\
\text{\$Building TLR covariance matrix time'} & \quad 0.004179112 \\
\text{\$Recursive block reordering time'} & \quad 0.1126912 \\
\text{\$Monte Carlo time'} & \quad 1.378512 \\
\text{\$Average rank'} & \quad 8
\end{align*}
\]

The difference between the probability estimates from the above two TLR function calls can be well contained in their estimates of the estimation error. From many similar numerical experiments, I conclude that the error caused by the TLR approximation is negligible compared with that from the Monte Carlo sampling. At this point, I clear the environment variables and build a covariance matrix based on 4,000 locations on a perturbed grid in (0,1). I do not choose a purely random geometry because the covariance matrix may appear singular under the truncation error, $\epsilon$, if there are locations too close to each other. Here, I use the same correlation function, the Whittle correlation function with a range parameter of $\beta = 0.1$, and generate the integration limits also from U($-5, -1$) and U(1,5).

R> rm(list = ls())
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 I add a random perturbation of the magnitude of 0.8 times the unit distance. Due to the
small unit distance and hence, the strong correlation between neighbor locations, I set the truncation error to $10^{-6}$ when calling the `pmvn.tlr` function.

```r
R> pmvn.tlr(a, b, 0, covM, m = 64, epsl = 1e-6)

$Estimation
[1] 0.01164927
$Error
[1] 0.0002779126
$Building TLR covariance matrix time'
[1] 0.0345531
$Recursive block reordering time'
[1] 0.9096951
$Monte Carlo time'
[1] 11.2869
$Average rank'
[1] 1
```

The block size is chosen to be close to $\sqrt{n}$ for optimal complexity. Typically, the covariance matrix from a 1D geometry and a smooth kernel has a strong low-rank feature and in the case above, the TLR representation achieves the highest savings possible because the average rank is one. The computation for a 4,000-dimensional MVN problem takes less than 13 seconds in total, which is considered fast compared with other similar packages. Finally, I consider a 4,000-dimensional MVN problem based on a 2D perturbed grid. The original grid is $50 \times 80$ with a unit distance of $1/80$, and the perturbation is still 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 of $\beta = 0.1$. Note that to generate the low-rank feature, I index the locations based on Morton’s order before constructing the two correlation matrices. Since the variance becomes two, I scale the range from which I generate the integration limits proportionally.

```r
R> rm(list = ls())
R> set.seed(123)
```
```r
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 can be only generated by users externally, for which the `zorder` becomes necessary for the low-rank feature. For the MVN problem above, I use a truncation error of $1e^{-5}$.

```r
R> pmvn.tlr(a, b, 0, covM, m = 64, epsl = 1e-5)
```

$Estimation$

[1] 1.754891e-05

$Error$

[1] 2.471964e-06

$'Building TLR covariance matrix time'$

[1] 0.1217158

$'Recursive block reordering time'$

[1] 5.991929

$'Monte Carlo time'$

[1] 24.20458

$'Average rank'$

[1] 5

Different from the 1D example, the average rank increases to five, which increases the total computation time to thirty seconds. Since the recursive block reordering operates on the TLR covariance matrix, its time is also affected by the average rank
of the off-diagonal blocks. It is worth noticing that here the probability estimate is smaller than that from the 1D example, which leads to a larger relative error. **TruncatedNormal** produces a smaller relative error for marginal probabilities but it is mainly used in lower dimensions.

### 5.3 Performance comparison

I use five examples from [Botev (2017)](https://doi.org/10.1007/s11222-016-9796-7) and [Genton et al. (2018)](https://doi.org/10.1080/01621459.2017.1388178) to compare the performances of the three packages, i.e., **mvtnorm**, **TruncatedNormal**, and **tlrvmvnmvt**.

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)](https://doi.org/10.1007/s11222-016-9796-7). The performance of the three packages up to 16,384 dimensions is shown in Table 5.1, where the accuracy is measured with the relative error of the log-probability. The true probability under the constant correlation structure is computed through a one-dimensional integration of

\[
\Phi_n(-\infty, \mathbf{b}; \Sigma_\rho) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} \exp \left( -\frac{1}{2} t^2 \right) \prod_{i=1}^{n} \Phi \left( \frac{b_i + \sqrt{\rho} t}{\sqrt{1 - \rho}} \right) dt, \tag{5.2}
\]
which can be found in Genz (1992). Here, $\Sigma_\rho$ denotes the constant correlation matrix whose correlation is denoted by $\rho$. I continue the notation for $\Phi(\cdot)$ from Chapter 3 that the mean parameter is assumed zero and omitted. This integration is computed by the Gauss quadrature (Golub and Welsch, 1969) rule, specifically, the Gauss-Hermite rule (Liu and Piered, 1994) with 200 nodes. With the `gauss.quad` function from the statmod package (Giner and Smyth, 2016), the function for computing Equation (5.2) is defined as

```r
library(statmod)
nnode <- 200
nodeWeight <- gauss.quad(nnode, "hermite")
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)
}
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 `pmvn.tlr` function produces higher relative error than the `pmvn.genz` function because the recursive block reordering, used in the former, reorders only on the block level, which is less flexible and effective than the univariate reordering used in the latter. The tlmvnmvmt 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 chapter are run on the Intel(R) Xeon(R) E5-2680 v4 @ 2.40GHz CPU.
Table 5.2: Performance analysis under the constant correlation structure. The lower integration limits are set at $-\infty$ and the upper integration limits are set at $-1.0$. The constant correlation $\rho = 0.5$. The default sample sizes are used for all functions involved with the exception of $n = 16,384$, 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.

<table>
<thead>
<tr>
<th></th>
<th>n</th>
<th>16</th>
<th>64</th>
<th>128</th>
<th>512</th>
<th>1,024</th>
<th>2,048</th>
<th>4,096</th>
<th>16,384</th>
</tr>
</thead>
<tbody>
<tr>
<td>mvtnorm</td>
<td>0.0%</td>
<td>0.3%</td>
<td>0.7%</td>
<td>2.8%</td>
<td>N.A.</td>
<td>N.A.</td>
<td>N.A.</td>
<td>N.A.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.1s</td>
<td>0.2s</td>
<td>0.5s</td>
<td>4.0s</td>
<td>N.A.</td>
<td>N.A.</td>
<td>N.A.</td>
<td>N.A.</td>
<td></td>
</tr>
<tr>
<td>TruncatedNormal</td>
<td>0.1%</td>
<td>0.1%</td>
<td>0.1%</td>
<td>0.1%</td>
<td>0.1%</td>
<td>0.1%</td>
<td>N.A.</td>
<td>N.A.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.0s</td>
<td>0.2s</td>
<td>0.7s</td>
<td>12.1s</td>
<td>67.6s</td>
<td>349.7s</td>
<td>N.A.</td>
<td>N.A.</td>
<td></td>
</tr>
<tr>
<td>pmvn.genz</td>
<td>0.1%</td>
<td>0.8%</td>
<td>1.0%</td>
<td>4.4%</td>
<td>6.8%</td>
<td>11.2%</td>
<td>17.8%</td>
<td>N.A.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.0s</td>
<td>0.1s</td>
<td>0.2s</td>
<td>1.2s</td>
<td>3.2s</td>
<td>9.9s</td>
<td>42.1s</td>
<td>N.A.</td>
<td></td>
</tr>
<tr>
<td>pmvn.tlr</td>
<td>0.0%</td>
<td>0.5%</td>
<td>1.4%</td>
<td>4.4%</td>
<td>9.0%</td>
<td>15.4%</td>
<td>17.6%</td>
<td>52.7%</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.0s</td>
<td>0.1s</td>
<td>0.2s</td>
<td>1.0s</td>
<td>2.1s</td>
<td>5.3s</td>
<td>14.0s</td>
<td>419.9s</td>
<td></td>
</tr>
</tbody>
</table>

I 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 significant 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 in Chapter 3, I found that the estimates of the logarithm of the probabilities are much more robust because the distribution of the probability estimates is skewed. I 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 \texttt{tlrmvnmvt} 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 5.1 to $(-\infty, -1)$, which becomes identical to Example IV in Botev (2017). The relative errors of the log-probabilities and computation times are listed in Table 5.2. This problem design is considered more challenging than the first example by Botev (2017), where the method from Genz (1992) already produces
a relative error of 100% when \( n = 100 \). Compared with Table 5.1, the relative error increases faster with \( n \), for which I use four times the default sample size in 16,384 dimensions. The methods from the \texttt{tlrnvmvt} package become less reliable when the problem size exceeds two thousand. The accuracy of the \texttt{TruncatedNormal} package is the least affected by the shift of the integration limits towards the tail based on these results.

The third example keeps the integration limits from the second one unchanged but increases the correlation \( \rho \) to 0.8, which resembles Table 8 in Genton et al. (2018). The corresponding results for \( \rho = 0.8 \) are shown in Table 5.3. The relative errors become smaller than those in Tables 5.1 and 5.2. 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 (5.2). The logarithm of such computed true probabilities is used as the benchmarks for computing the relative errors. For these three examples, I do not experiment with the MVT functions because of the lack of

<table>
<thead>
<tr>
<th>( n )</th>
<th>16</th>
<th>64</th>
<th>128</th>
<th>512</th>
<th>1,024</th>
<th>2,048</th>
<th>4,096</th>
<th>16,384</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{mvtnorm}</td>
<td>0.0%</td>
<td>0.1%</td>
<td>0.1%</td>
<td>0.1%</td>
<td>N.A.</td>
<td>N.A.</td>
<td>N.A.</td>
<td>N.A.</td>
</tr>
<tr>
<td></td>
<td>0.0s</td>
<td>0.2s</td>
<td>0.5s</td>
<td>4.0s</td>
<td>N.A.</td>
<td>N.A.</td>
<td>N.A.</td>
<td>N.A.</td>
</tr>
<tr>
<td>\texttt{TruncatedNormal}</td>
<td>0.1%</td>
<td>0.1%</td>
<td>0.1%</td>
<td>0.1%</td>
<td>0.2%</td>
<td>0.2%</td>
<td>N.A.</td>
<td>N.A.</td>
</tr>
<tr>
<td></td>
<td>0.0s</td>
<td>0.2s</td>
<td>0.7s</td>
<td>12.5s</td>
<td>59.0s</td>
<td>350.5s</td>
<td>N.A.</td>
<td>N.A.</td>
</tr>
<tr>
<td>\texttt{pmvn.genz}</td>
<td>0.0%</td>
<td>0.1%</td>
<td>0.1%</td>
<td>0.3%</td>
<td>0.4%</td>
<td>0.4%</td>
<td>0.4%</td>
<td>N.A.</td>
</tr>
<tr>
<td></td>
<td>0.0s</td>
<td>0.1s</td>
<td>0.2s</td>
<td>1.2s</td>
<td>3.2s</td>
<td>10.1s</td>
<td>43.0s</td>
<td>N.A.</td>
</tr>
<tr>
<td>\texttt{pmvn.tlr}</td>
<td>0.0%</td>
<td>0.1%</td>
<td>0.2%</td>
<td>0.3%</td>
<td>0.3%</td>
<td>0.4%</td>
<td>0.6%</td>
<td>1.2%</td>
</tr>
<tr>
<td></td>
<td>0.0s</td>
<td>0.1s</td>
<td>0.2s</td>
<td>1.0s</td>
<td>2.1s</td>
<td>5.4s</td>
<td>14.1s</td>
<td>93.7s</td>
</tr>
</tbody>
</table>
their true probabilities.

The fourth example uses randomly generated covariance matrices as described in [Davies and Higham (2000)](https://doi.org/10.1016/S0005-1444(00)00008-0) and defines the integration limits to be \((-\infty, 0.5)\), which is similar to Example III in [Botev (2017)](https://arxiv.org/abs/1706.08581) but experiments in 1,000 dimensions. The code snippet for generating random correlation matrices is based on the \texttt{rGivens} function from the \texttt{fungible} package ([Waller, 2019](https://doi.org/10.1016/j.amstat.2019.07.008)).

```r
library(fungible)
lambda <- runif(n)
lambda <- lambda * n / sum(lambda)
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 5.4.

I also include the MVT functions from the three packages, e.g., the \texttt{pmvt} function.

Table 5.4: 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)](https://doi.org/10.1016/S0005-1444(00)00008-0). The lower integration limits are set at \(-\infty\) and the upper integration limits are set at 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.

<table>
<thead>
<tr>
<th>Methods</th>
<th>Min</th>
<th>1st quartile</th>
<th>Median</th>
<th>3rd quartile</th>
<th>Max</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>MVN probabilities</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>mvtnorm</td>
<td>0.11%</td>
<td>0.23%</td>
<td>0.30%</td>
<td>0.43%</td>
<td>0.58%</td>
<td>14.4s</td>
</tr>
<tr>
<td>TruncatedNormal</td>
<td>0.03%</td>
<td>0.05%</td>
<td>0.05%</td>
<td>0.07%</td>
<td>0.15%</td>
<td>59.4s</td>
</tr>
<tr>
<td>pmvn.genz</td>
<td>0.16%</td>
<td>0.25%</td>
<td>0.33%</td>
<td>0.40%</td>
<td>0.60%</td>
<td>3.1s</td>
</tr>
<tr>
<td><strong>MVT probabilities</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>mvtnorm</td>
<td>3.42%</td>
<td>5.43%</td>
<td>6.31%</td>
<td>7.94%</td>
<td>15.98%</td>
<td>14.5s</td>
</tr>
<tr>
<td>TruncatedNormal</td>
<td>0.01%</td>
<td>0.02%</td>
<td>0.03%</td>
<td>0.03%</td>
<td>0.06%</td>
<td>76.5s</td>
</tr>
<tr>
<td>pmvtn.genz</td>
<td>2.06%</td>
<td>5.80%</td>
<td>7.70%</td>
<td>8.98%</td>
<td>11.45%</td>
<td>3.1s</td>
</tr>
</tbody>
</table>

from the \texttt{mvtnorm} package, the \texttt{pmvt} function from the \texttt{TruncatedNormal} package, and the \texttt{pmvtn.genz} function from the \texttt{tlrmvnmvnt} package. The methods from the \texttt{mvtnorm} and \texttt{tlrmvnmvnt} packages perform much better for MVN probabilities than
for MVT probabilities. I 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 again low for MVN probabilities, which renders tlmvnmvmt the preferred choice due to its shorter computation times. For applications involving MVT probabilities with weak irregular correlation, the TruncatedNormal package is a more reliable option.

The last example assumes a 2D Whittle correlation structure that is used in Section 5.2 of this chapter 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), I simulate integration limits from $U(-5, -1)$ and $U(1, 5)$, which is more challenging as the true probabilities become smaller. The performance of the three packages in 900 dimensions is summarized in Table 5.5. The Table 5.5: Quartiles of the relative errors of the log-probabilities under Whittle correlation matrices in 900 dimensions. The correlation matrix is generated based on a $30 \times 30$ 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 $U(-5, -1)$ and the upper integration limits are independently generated from $U(1, 5)$. 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.

<table>
<thead>
<tr>
<th>Methods</th>
<th>Min</th>
<th>1st quartile</th>
<th>Median</th>
<th>3rd quartile</th>
<th>Max</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>MVN probabilities</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>mvtnorm</td>
<td>0.09%</td>
<td>0.17%</td>
<td>0.20%</td>
<td>0.22%</td>
<td>0.29%</td>
<td>13.0s</td>
</tr>
<tr>
<td>TruncatedNormal</td>
<td>0.12%</td>
<td>0.21%</td>
<td>0.26%</td>
<td>0.29%</td>
<td>0.37%</td>
<td>46.3s</td>
</tr>
<tr>
<td>pmvn.genz</td>
<td>0.15%</td>
<td>0.27%</td>
<td>0.33%</td>
<td>0.38%</td>
<td>0.48%</td>
<td>3.3s</td>
</tr>
<tr>
<td>MVT probabilities</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>mvtnorm</td>
<td>0.31%</td>
<td>0.43%</td>
<td>0.48%</td>
<td>0.63%</td>
<td>0.70%</td>
<td>12.7s</td>
</tr>
<tr>
<td>TruncatedNormal</td>
<td>0.18%</td>
<td>0.23%</td>
<td>0.25%</td>
<td>0.26%</td>
<td>0.33%</td>
<td>62.0s</td>
</tr>
<tr>
<td>pmvt.genz</td>
<td>0.58%</td>
<td>0.75%</td>
<td>0.92%</td>
<td>1.07%</td>
<td>1.22%</td>
<td>3.2s</td>
</tr>
</tbody>
</table>

difference in the convergence rate between MVN probabilities and MVT probabilities
is less obvious than that in Table 5.4, which indicates that the Monte Carlo methods without importance sampling for MVT probabilities work more favorably under positive correlation. The relative errors are small for all methods while the computation time of the \texttt{tlrmvnmvt} package is less than a quarter of that of the \texttt{mvtnorm} package and approximately one-twentieth of that of the \texttt{TruncatedNormal} package. I think that overall, the \texttt{tlrmvnmvt} package has a good tradeoff between accuracy and computation time for this category of MVN/MVT problems.

For Tables 5.4 and 5.5, the true probabilities are no longer available and hence, for each simulated problem, I repeat the computation for ten times, based on which the errors are computed as the standard deviation of the ten estimates. 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 5.1 shows the scalability of the \texttt{pmvn.genz} and \texttt{pmvn.tlr} functions, featuring the time growth with $n$ when the Monte Carlo sample size is fixed at 1e4. Figure 5.1(a) suggests that the \texttt{pmvn.genz} function may serve as a more efficient implementation of the \texttt{pmvnorm} function from the \texttt{mvtnorm} package and that the cost of the minimax tilting procedure used in the \texttt{TruncatedNormal} package grows quickly with $n$. Figure 5.1(b) compares the costs of the \texttt{pmvn.genz} and the \texttt{pmvn.tlr} functions under the 2D Whittle correlation structure. The difference in time is not significant when $n < 1,000$ but their ratio increases to 20 when $n = 16,384$. I conclude from the five examples above that the TLR methods from the \texttt{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 \texttt{TruncatedNormal} package is probably the best option. However, for most other cases, the functions in the \texttt{tlrmvnmvt} package can be the preferred choice for high-
Figure 5.1: 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 5.4. For (b), the problems are simulated in the same way as those in Table 5.5 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.

5.4 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, I show that with the tlrmvnmvt package, it is feasible to 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:

$$E_{u,\alpha}^+(X) = \arg\max_D \{|D| : P\{D \subseteq A_u^+(X)\} \geq 1 - \alpha\},$$

where $A_u^+(f) = \{s \in \Omega; f(s) > u\}$ and $f$ is a deterministic function in $\Omega$. 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:

$$D_1^+(\rho) = \{s; P\{X(s) > u\} \geq 1 - \rho\},$$

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:

$$F_u^+(s) = \sup\{1 - \alpha; s \in E_{u,\alpha}^+(X)\},$$

which can be significantly different from the posterior marginal exceedance probability. To compute $E_{u,\alpha}^+(X)$ and $F_u^+(s)$, it requires the estimation of $P\{D \subseteq A_u^+(X)\}$ 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, $X(s)$ is a Gaussian random field with a zero mean structure and the Matérn covariance structure, described in Equation (5.1), with $\sigma = 1$, $\beta = \sqrt{2}/20$, $\nu = 1$, and $\delta = 0$. Our goal is to predict one realization of this random field on a $80 \times 80$ grid in the unit square based on one thousand observations on this grid under the additive Gaussian noise $N(0, 0.5^2)$. Using $y$ to denote the
observations, \( X(s) \mid y \) is the 2D latent Gaussian random field whose excursion sets are of interest. The posterior distribution of \( X(s) \mid y \) is another MVN distribution with

\[
Q_{\text{post}} = Q + \frac{1}{\epsilon^2} A^\top A, \\
\mu_{\text{post}} = \mu + \frac{1}{\epsilon^2} Q_{\text{post}}^{-1} A^\top (y - A\mu),
\]

where \( \mu = 0 \) and \( Q \) are the mean and the precision matrix of \( x \), \( \mu_{\text{post}} \) and \( Q_{\text{post}} \) are the posterior mean and precision matrix of \( x \mid y \), \( A \) is the matrix such that \( y - Ax \) \( \sim \) \( N(0, \epsilon^2) \) and \( \epsilon = 0.5 \) is the magnitude of the additive Gaussian noise. Here, I use \( x \) for the realization of \( X(s) \) on the grid. When the level \( u \) is zero, the posterior marginal probability, \( P\{X(s) > 0 \mid y\} \), and the positive excursion function, \( F_0^+(s) \), are shown in Figure 5.2. The posterior marginal probability is greater than or equal to the positive excursion function and in fact, \( E_{x.0.99}^+(X \mid y) \subseteq D_1^+(0.05) \), which indicates a strong mismatch between \( P\{X(s) > 0 \mid y\} \) and \( F_0^+(s) \). Similar to Bolin and Lindgren (2015), I simulate 50,000 realizations from \( X(s) \mid y \) and compute the empirical probability of \( \min(\{X(s); s \in E_0^+(X)\}) > 0 \), denoted by \( \hat{\rho}(\alpha) \). Figure 5.3 describes the difference between \( \hat{\rho}(\alpha) \) and \( 1 - \alpha \) as a function of \( 1 - \alpha \). The differences are smaller than those in Bolin and 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 \texttt{pmvn.tlr} function is used for computing \( E_{0,\alpha}^+(X) \) and \( F_0^+(s) \) because the low-rank feature is strong if the locations on the \( 80 \times 80 \) grid are ordered with the \texttt{zorder} function. The run time is reduced by a factor between two and three compared with that using the \texttt{pmvn.genz} function. For higher-dimensional applications, this computation saving is expected to be more significant. The following code snippet is used to generate the observations and compute the posterior MVN
Figure 5.2: (a) The posterior marginal probability for $P\{X(s) > 0 \mid y\}$ and (b) The positive excursion function $F_0^+(s)$.

parameters.
	n.obs <- 1000
	sigma.e <- 0.5

m <- 80

n <- m * m

Figure 5.3: The difference between the empirical probability of $\min\{X(s); s \in E_{0,\alpha}(X)\} > 0$, denoted by $\hat{p}(\alpha)$, and $1 - \alpha$. 
x <- seq(from = 0, to = 1, length.out = m)
y <- x
geom <- cbind(kronecker(x, rep(1, m)), kronecker(rep(1, m), y))
odrMorton <- zorder(geom)
geom <- geom[odrMorton, ]
distM <- as.matrix(dist(geom))
covM <- matern(distM, phi = sqrt(2) / 20, kappa = 1)
cholM <- t(chol(covM))
mu <- rep(0, n)
set.seed(120)
y0 <- as.vector(cholM %*% rnorm(n)) + mu
obsIdx <- sample(1 : n, n.obs)
Y <- y0[obsIdx] + rnorm(n.obs) * sigma.e
A <- matrix(0, n.obs, n)
for(i in 1 : n.obs)
  A[i, obsIdx[i]] <- 1
Q <- solve(covM)
Q.post <- Q + t(A) %*% A / (sigma.e^2)
mu.post <- as.vector(mu +
solve(Q.post, (t(A) %*% (Y - mu[obsIdx])) / (sigma.e^2)))
covM <- solve(Q.post)

Here, Q.post is inverted to produce the posterior covariance matrix. Knowing the posterior mean and covariance matrix, the marginal probabilities on the grid can readily be achieved to build the one-parameter family, \( D_1^+(\rho) \), based on which the excursion function is computed.

\[
pMar <- 1 - pnorm(0, \text{mean}=\text{mu.post}, \text{sd} = \text{sqrt(diag(covM))})
\]

\[
FMar <- \text{rep}(0, n)
\]

\[
\text{ttlNum} <- \text{sum}(pMar > 0.95)
\]

\[
pMar0dr <- \text{order}(pMar, \text{decreasing} = \text{T})
\]

\[
\text{numVec} <- \text{round}(\text{seq(from} = 3180, \text{to} = 1000, \text{by} = -10))
\]

\[
\text{for(\text{num in numVec})}
\{
  \text{selectIdx} <- \text{pMar0dr}\{1 : \text{num}\}
  \text{tmpLower} <- \text{rep}(-\text{Inf}, n)
  \text{tmpLower[selectIdx]} <- 0
  \text{FMar[selectIdx]} <- \text{pmvn.tlr} (\text{lower} = \text{tmpLower}, \text{mean} = \text{mu.post},
  \text{sigma} = \text{covM}, \text{m} = 80)[[1]]
\}

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

In this chapter, I introduced the five functions from the R package \texttt{tlrmvnmvt} that is based on the results in Chapter 3, and compared their performances with two other similar packages. This package provides possibly the only option for applications that need to compute MVN/MVT probabilities in thousands of dimensions. The dense-matrix-based functions, \texttt{pmvn.genz} and \texttt{pmvt.genz}, are based on the same algorithms as those used for the \texttt{pmvnorm} and \texttt{pmvt} functions from the \texttt{mvtnorm} package but have a more efficient implementation. Their difference in terms of convergence rate is likely due to the number of Monte Carlo samples and the Quasi-Monte Carlo rules. The TLR methods, e.g., \texttt{pmvn.tlr} and \texttt{pmvt.tlr}, further improve the computation efficiency, making the computation of MVN/MVT probabilities in tens of thousands of dimensions feasible on a standard scientific workstation. However, the TLR methods are only valid if the covariance matrix can be closely approximated by the TLR structure. Fortunately, common spatial covariance structures and constant covariance structures, among others, are covered in this category.

I used the computation of the excursion sets of a latent Gaussian random field \citep{Bolin2015} as an example to which the \texttt{tlrmvnmvt} package can be applied. \cite{Bolin2015} 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 \texttt{tlrmvnmvt} package, the low-rank feature of the posterior covariance matrix could be utilized and the excursion sets using the exact posterior distribution could be more efficiently computed. I showed that the accuracy of the excursion sets was improved by one order of magnitude compared with \cite{Bolin2015}.

Consistent with the conclusions from \citep{Botev2017}, the errors of the estimates from the Monte Carlo methods without importance sampling increase more quickly with the problem size for tail probabilities than for non-tail probabilities. The Trun-
catedNormal package effectively controls the errors of tail probabilities but its cost for the minimax tilting procedure becomes prohibitively expensive in thousands of dimensions. The tail probabilities under a weak-to-medium correlation strength in thousands of dimensions remain a challenging problem to study.
Chapter 6

Computation of High-dimensional Probit Gaussian Random Field for Binary Classification

6.1 Chapter overview

A wide variety of applications can be viewed as binary classification, where \( y \in \{0, 1\} \) is predicted given a predictor vector \( \mathbf{x} \in \mathbb{R}^p \). One common technique for modeling these applications is adding a link function to the classic models as a generalization. For models in this category, some consider each predictor separately to make model fitting and interpretation easy while others consider the coupling effect between predictors. Generalized linear models \((\text{Nelder and Wedderburn, 1972})\) belong to the first category; whereas the second category includes smoothing splines \((\text{Wahba, 1990})\), Bayesian additive regression trees \((\text{Chipman et al., 2010})\), and generalized Gaussian processes \((\text{Girolami and Rogers, 2006})\). Models in the second category are typically more complicated and do not have tractable posterior distributions. However, in this chapter, I show that the Gaussian random field generalized through the probit link function has a tractable posterior distribution that involves the MVN probabilities and that the R package described in Chapter 5 greatly facilitate the model selection and the posterior prediction.

As mentioned in Chapter 6, \(\text{Durante (2019)}\) recently discovered that the posterior distribution of the linear coefficients in the generalized linear model with a probit link function under the Gaussian prior belongs to the class of unified skew-normal (SUN) distribution \((\text{Arellano-Valle and Azzalini, 2006})\). Specifically, this result can
be summarized as:

\[
\text{pr}(y = 1 \mid \mathbf{x}, \mathbf{\beta}) = \Phi(\mathbf{x}^\top \mathbf{\beta}), \quad \mathbf{\beta} \sim \mathcal{N}_p(\mathbf{\xi}, \mathbf{\Sigma})
\]

\[
\Rightarrow (\mathbf{\beta} \mid y, \mathbf{x}) \sim \text{SUN}_{p,n}(\mathbf{\xi}_{\text{post}}, \mathbf{\Sigma}_{\text{post}}, \mathbf{\Delta}_{\text{post}}, \mathbf{\gamma}_{\text{post}}, \mathbf{\Gamma}_{\text{post}}),
\] (6.1)

where the definitions of SUN(·) and the posterior parameters can be found in Equations (6.2) and (6.3), respectively; see Arellano-Valle and Azzalini (2006) and Durante (2019) for a detailed reference. The probit Gaussian random field used in this Chapter is generalized in a similar way to the linear probit model with Gaussian priors and this chapter derives the posterior distribution of the probit Gaussian random field in a similar manner.

The scalability of the SUN distribution is mainly limited by the estimation of MVN probabilities that appear in its probability density function. Besides, the simulation from the SUN distribution needs sampling from the truncated MVN distribution, which is also challenging when the acceptance probability is small. This chapter focuses on model selection and posterior prediction, which do not involve the simulation from the posterior distribution. Therefore, only the first limitation is tackled and I show that the tlrmvnmvt package introduced in Chapter 5 makes model selection and posterior prediction feasible for \( n \) in thousands of dimensions.

### 6.2 Properties

Suppose there are \( n \) observations, \( \{(\mathbf{x}_i, y_i)\}_{i=1}^n \), each of which consists of a vector of covariates \( \mathbf{x}_i \in \mathbb{R}^p \) and a binary response \( y_i \in \{0, 1\} \). The model under study is
\[ \text{pr}(y_i = 1) = \Phi\{g(x_i)\} \]

\[
\begin{bmatrix}
g(x_1) \\
g(x_2) \\
\vdots \\
g(x_n)
\end{bmatrix} \sim N_n
\begin{bmatrix}
\xi(x_1) \\
\xi(x_2) \\
\vdots \\
\xi(x_n)
\end{bmatrix}
K(x_1, x_1) & K(x_1, x_2) & \cdots & K(x_1, x_n) \\
K(x_2, x_1) & K(x_2, x_2) & \cdots & K(x_2, x_n) \\
\vdots & \vdots & \ddots & \vdots \\
K(x_n, x_1) & K(x_n, x_2) & \cdots & K(x_n, x_n)
\end{bmatrix}
\]

Here, \( g(\cdot) \) is a Gaussian random field while \( \xi(\cdot) \) and \( K(\cdot, \cdot) \) are deterministic functions representing the mean and the covariance structures of \( g(\cdot) \). This model adds the probit response function, \( \Phi(\cdot) \), to a Gaussian random field, for which it is referred as the \textit{probit Gaussian random field}. The following notations are used in this Chapter to derive the posterior distribution of \( g \):

\[
X = [x_1^T, x_2^T, \ldots, x_n^T]^T, \quad y = [y_1, y_2, \ldots, y_n]^T, \quad D = \text{diag}(2\gamma - 1),
\]

\[
\phi_n(x; \xi, \Sigma) : \text{the probability density function of } N_n(\xi, \Sigma),
\]

\[
\Phi_n(x; \xi, \Sigma) : \text{the cumulative distribution function of } N_n(\xi, \Sigma).
\]

I use the unbolded font with one subscript for the coefficient in a vector and the unbolded font with two subscripts for the coefficient in a matrix. Before the derivation of the posterior distribution of \( g \), I first describe the SUN distribution that was initially introduced in \cite{Arellano-Valle and Azzalini, 2006}.

\textbf{Lemma 1. If } \text{X} \sim \text{SUN}_{p,n}(\xi_p, \Sigma_{p \times p}, \Delta_{p \times n}, \gamma_n, \Gamma_{n \times n}), \text{ then }

\[
\pi(x) = \phi_p(x - \xi; 0, \Sigma) \frac{\Phi_n\{\gamma + \Delta^\top \Sigma^{-1} \omega^{-1}(x - \xi); 0, \Gamma - \Delta^\top \Sigma^{-1} \Delta\}}{\Phi_n(\gamma; 0, \Gamma)}. (6.2)
\]

The SUN distribution is closed under marginalization, linear combination, and conditioning \cite{Gupta et al., 2013}, which facilitates statistical inferences. However,
high-dimensional inferences are still subject to the computation of MVN probabilities. With the formal definition of the SUN distribution, Theorem 1 describes the posterior distribution of \( g \):

**Theorem 1.** If \( \{y_i\}_{i=1}^n \) are conditionally independent given \( g \) with

\[
(y_i \mid g) \sim \text{Bernoulli}(\Phi(g_i)) \quad \text{and} \quad (g \mid X) \sim N_n(\xi, \Sigma).
\]

The posterior distribution of \( g \) is:

\[
(g \mid y, X) \sim \text{SUN}_{n,n}(\xi_{\text{post}}, \Sigma_{\text{post}}, \Delta_{\text{post}}, \gamma_{\text{post}}, \Gamma_{\text{post}})
\]

where

\[
\begin{align*}
\xi_{\text{post}} &= \xi, \\
\Sigma_{\text{post}} &= \Sigma, \\
\Delta_{\text{post}} &= \Sigma \omega D^\top s^{-1}, \\
\gamma_{\text{post}} &= s^{-1} D \xi, \\
\Gamma_{\text{post}} &= s^{-1} (D \Sigma D^\top + I_n)s^{-1}.
\end{align*}
\]

Here, \( \omega \) and \( s \) are the diagonal matrices that make \( \Sigma = \omega^{-1} \Sigma \omega^{-1} \) and \( \Gamma_{\text{post}} = s^{-1} (D \Sigma D^\top + I_n)s^{-1} \) correlation matrices, respectively.

**Proof.**

\[
\pi(g \mid y, X) \propto \pi(y \mid g) \pi(g \mid X) = \left[ \prod_{i=1}^n \Phi((2y_i - 1)g_i) \right] \phi_n(g; \xi, \Sigma)
\]

\[
= \phi_n(g; \xi, \Sigma) \Phi_n\{Dg; 0, I_n\}
\]

\[
= \phi_n(g; \xi, \Sigma) \Phi_n\{s^{-1}D \xi + s^{-1}D \omega \Sigma \Sigma^{-1} \omega^{-1}(g - \xi); 0, s^{-1}s^{-1}\}
\]

\[
\propto \text{SUN}_{n,n}(\xi_{\text{post}}, \Sigma_{\text{post}}, \Delta_{\text{post}}, \gamma_{\text{post}}, \Gamma_{\text{post}})
\]

The SUN parameters of the posterior distribution are similar to those in Durante.
but $\xi$, $\Sigma$, and $D$ bear different meanings in this chapter. Here, unlike Durante (2019) that models the linear coefficients of the mean structure of $g$, the distribution of $g$ is modeled directly. Without standardizing the covariance matrices, Equation (6.3) can be simplified into:

$$
\pi(g | y, X) = \phi_n(g; \xi, \Sigma) \frac{\Phi_n\{Dg; 0, I_n\}}{\Phi_n(D\xi; 0, D\Sigma D^T + I_n)}.
$$

Based on the stochastic representation of the SUN distribution introduced in Arellano-Valle and Azzalini (2006), Theorem 1 directly leads to the stochastic representation of the posterior distribution of $g$.

**Corollary 1.** If $(g | y, X) \sim \text{SUN}_{n,n}(\xi_{\text{post}}, \Sigma_{\text{post}}, \Delta_{\text{post}}, \gamma_{\text{post}}, \Gamma_{\text{post}})$, then

$$
(g | y, X) \overset{d}{=} \xi + \omega\{V_0 + \Sigma_1 D \Sigma_2 D^T (D\Sigma D^T + I_n)^{-1}s V_1\},
$$

(6.5)

where $V_0$ and $V_1$ are independent, $V_0 \sim N_n\{0, \Sigma - \Sigma_1 D \Sigma_2 D^T (D\Sigma D^T + I_n)^{-1}D \Sigma \}$, and $V_1 \overset{d}{=} (\tilde{V}_1 | \tilde{V}_1 > -s^{-1}D\xi)$ with $\tilde{V}_1 \sim N_n\{0, s^{-1}(D\Sigma D^T + I_n)s^{-1}\}$.

**Proof.** This proof can be achieved by substituting the $\xi$, $\Sigma$, $\Delta$, $\gamma$, and $\Gamma$ in Equation (7.4) from Arellano-Valle and Azzalini (2006) with their posterior counterparts from Theorem 1.

To simulate from this posterior distribution, efficient sampling from the truncated MVN distribution is needed, which is challenging in high dimensions because the acceptance probability decreases exponentially with $n$. Two existing packages in R (R Core Team, 2019) provide this functionality, whose scalability is discussed in Section 6.3. Similar to Durante (2019), I consider the posterior prediction for a new vector of covariates, $x_{n+1}$, given observations $(X, y)$. $X_{\text{new}}$ and $y_{\text{new}}$ are used to denote the extensions of $X$ and $y$ with $x_{n+1}$ and $y_{n+1}$, based on which $D_{\text{new}} = \text{diag}(2y_{\text{new}} - 1)$. $\xi_{\text{new}}$ and $\Sigma_{\text{new}}$ are the mean and the covariance matrix of the Gaussian random field
Corollary 2. If \((y \mid g) \sim \text{Bernoulli}\{\Phi(g_i)\}\) independently for \(i = 1, \ldots, n\) and \(g(\cdot)\) is a Gaussian random field defined by its mean and covariance structure, then

\[
P(y_{n+1} = 1 \mid y, X_{\text{new}}) = \frac{\Phi_{n+1}\left(s_{\text{new}}^{-1}D_{\text{new}}\xi_{\text{new}}; 0, s_{\text{new}}^{-1}(D_{\text{new}}\Sigma_{\text{new}}D_{\text{new}}^\top + I_{n+1})s_{\text{new}}^{-1}\right)}{\Phi_n\left(s^{-1}D\xi; 0, s^{-1}(D\Sigma D^\top + I_n)s^{-1}\right)},
\]

where \(s_{\text{new}}^{-1}\) is the diagonal matrix so that \(s_{\text{new}}^{-1}(D_{\text{new}}\Sigma_{\text{new}}D_{\text{new}}^\top + I_{n+1})s_{\text{new}}^{-1}\) is a correlation matrix.

Proof.

\[
\pi(g_{\text{new}} \mid y, X_{\text{new}}) \propto \phi_{n+1}(g_{\text{new}}; \xi_{\text{new}}, \Sigma_{\text{new}}) \prod_{i=1}^n \Phi\{(2y_i - 1)g_i\}
\]

\[
= \phi_{n+1}(g_{\text{new}}; \xi_{\text{new}}, \Sigma_{\text{new}}) \Phi_n\{\hat{D}g_{\text{new}}; 0, I_n\},
\]

where \(\hat{D}\) is the first \(n\) rows of \(D_{\text{new}}\). Similar to the proof of Theorem 3, \((g_{\text{new}} \mid y, X_{\text{new}})\) has an \(\text{SUN}_{n+1,n}\) distribution with:

\[
\xi = \xi_{\text{new}}, \quad \Sigma = \Sigma_{\text{new}}, \quad \Delta = \Sigma_{\text{new}}\omega_{\text{new}}\hat{D}^\top s^{-1},
\]

\[
\gamma = s^{-1}\hat{D}\xi_{\text{new}}, \quad \Gamma = s^{-1}\{\hat{D}\Sigma_{\text{new}}\hat{D}^\top + I_n\}s^{-1}.
\]

Here, \(\omega_{\text{new}}\) and \(s\) are the diagonal matrices that make \(\Sigma_{\text{new}} = \omega_{\text{new}}^{-1}\Sigma_{\text{new}}\omega_{\text{new}}^{-1}\) and \(\Gamma = s^{-1}\{\hat{D}\Sigma_{\text{new}}\hat{D}^\top + I_n\}s^{-1}\) correlation matrices, respectively. The posterior predictive
probability \( \text{pr}(y_{n+1} = 1 \mid y, X_{\text{new}}) \) is:

\[
\int \Phi(g_{n+1}) \pi(g_{\text{new}} \mid y, X_{\text{new}}) \, dg_{\text{new}} \\
= \Phi_n\{s^{-1}\hat{D}\xi_{\text{new}}; 0, s^{-1}(\hat{D}\Sigma_{\text{new}}\hat{D}^\top + I_n)s^{-1}\}^{-1} \times \\
\int \Phi(g_{n+1})\phi_{n+1}(g_{\text{new}}; \xi_{\text{new}}, \Sigma_{\text{new}})\Phi_n\{\hat{D}g_{\text{new}}; 0, I_n\} \, dg_{\text{new}} \\
= \Phi_n\{s^{-1}D\xi; 0, s^{-1}(D\Sigma D^\top + I_n)s^{-1}\}^{-1} \times \\
\int \phi_{n+1}(g_{\text{new}}; \xi_{\text{new}}, \Sigma_{\text{new}})\Phi_{n+1}\{D_{\text{new}}g_{\text{new}}; 0, I_{n+1}\} \, dg_{\text{new}} \\
= \frac{\Phi_{n+1}\{s^{-1}_{\text{new}}D_{\text{new}}\xi_{\text{new}}; 0, s^{-1}_{\text{new}}(D_{\text{new}}\Sigma_{\text{new}}D_{\text{new}}^\top + I_{n+1})s^{-1}_{\text{new}}\}}{\Phi_n\{s^{-1}D\xi; 0, s^{-1}(D\Sigma D^\top + I_n)s^{-1}\}}.
\]

Here, Corollary 2 differs from that in [Durante (2019)] with respect to the numerator because of the extension of the mean and covariance structure. The computation of this ratio can be simplified into computing only one MVN probability in \((n + 1)\)-dimensions since the denominator is a marginalization of the numerator. In fact, merging the computation of the numerator and the denominator reduces the estimation error, which is shown in Section 6.4. Finally, I show the derivation of the marginal probability in Corollary 3 that can be used for model selection. I use \( M_k \) to denote the \( k \)-th model that specifies the mean and the covariance structure of the Gaussian process \( g(\cdot) \).

**Corollary 3.** If \((y_i \mid g) \sim \text{Bernoulli}(\Phi(g_i)) \) independently for \( i = 1, \ldots, n \) and \( g(\cdot) \) is a Gaussian random field whose mean and covariance structure are defined by \( M_k \), then

\[
\text{pr}(y \mid M_k, X) = \Phi_n\{s^{-1}D\xi; 0, s^{-1}(D\Sigma D^\top + I_n)s^{-1}\}.
\]

(6.7)
Proof.

\[ \text{pr}(y \mid M_k, X) = \int \text{pr}(y \mid g)\pi(g \mid M_k, X) \, dg, \]

where the integrand is essentially Equation (6.4), which can be normalized to a probability density function by the factor \( \Phi_n\{s^{-1}D\xi; 0, s^{-1}(D\Sigma D^\top + I_n)s^{-1}\}^{-1} \). Therefore, this integration is equal to \( \Phi_n\{s^{-1}D\xi; 0, s^{-1}(D\Sigma D^\top + I_n)s^{-1}\}. \)

\[ \square \]

### 6.3 MVN probabilities and truncated distributions

Three corollaries are derived based on Theorem 1 that lead to efficient algorithms for the simulation of the posterior distribution, the prediction given a new predictor vector, and the selection of prior models, respectively. However, Corollary 1 requires the simulation of random vectors from the truncated MVN distribution in \( n \) dimensions while Corollaries 2 and 3 involve the estimation of \( n \)-dimensional MVN probabilities.

For the posterior simulation, two R packages within my knowledge can simulate from the truncated MVN distribution, namely the `tmvtnorm` (Wilhelm and Manjunath, 2013) and the `TruncatedNormal` (Botev and Belzile, 2019) packages. Both implement the accept-reject sampling method but the latter has a more effective proposal density that is found through the minimax exponential tilting method (Botev, 2017).

In fact, the `tmvtnorm` package already reaches low acceptance probability in tens of dimensions while the `TruncatedNormal` package can maintain a proper acceptance probability even in hundreds of dimensions. The `tmvtnorm` package also implements a Gibbs sampler. However, it is less favorable for the purpose of simulating independent samples.

In terms of model selection, I compare three packages that are dedicated to the computation of MVN probabilities. The first one is the `mvtnorm` (Genz and Bretz, 2009; Genz et al., 2019) package, which implements the celebrated separation-of-
variable (SOV) algorithm introduced in Genz (1992). It is the first package for the computation of MVN probabilities in R (R Core Team, 2019) and the SOV algorithm is the foundation for many later-developed algorithms dedicated to MVN probabilities. This package is typically used for MVN probabilities in less than one thousand dimensions as required by its input argument check. The second one is the TruncatedNormal (Botev and Belzile, 2019) package. Unlike Genz and Bretz (2009) that improved the convergence rate with variable reordering, this package uses importance sampling to enhance the convergence rate of the SOV algorithm, which was shown in Botev (2017) to be more effective, especially for tail probabilities. However, the package needs to solve a minimax optimization problem for constructing the proposal density, which increases its time costs. This package is recommended for tail probabilities or for applications with high accuracy requirements but the number of dimensions where the package is applicable may be limited to hundreds. The last one is the tlrmvnmvt package introduced in Chapter 5 that uses a tile-low-rank (TLR) representation to improve the time efficiency of the SOV algorithm as described in Chapter 3. Under the low-rank representation, time efficiency can be gained from both the SOV algorithm and the Cholesky factorization that is needed by the SOV algorithm. The package has an accuracy level that is comparable with the mvtnorm package since the block reordering used under the TLR representation bears the same heuristic as the univariate reordering used in the mvtnorm package. The tlrmvnmvt package can tackle MVN problems up to tens of thousands of dimensions but the convergence rate generally deteriorates as the number of dimensions increases, for which prior investigation is needed to decide whether the estimation accuracy is sufficient.

The prediction based on the posterior distribution appears to involve two MVN probabilities as described in Equation (6.7). However, if they are computed separately, the error of the ratio is approximately the error of the numerator scaled by the inverse of the magnitude of the denominator and hence, the estimation error of the ratio is
likely to be significant. Instead, the two MVN probabilities can be combined in terms of computation, which halves the computation costs and improves the estimation accuracy. Specifically, the SOV algorithm from Genz (1992) transforms $\Phi_n(a, b; 0, \Sigma)$ into:

$$
\Phi_n(a, b; 0, \Sigma) = (e_1 - d_1) \int_0^1 (e_2 - d_2) \cdots \int_0^1 (e_n - d_n) \int_0^1 \mathrm{d}w,
$$

where $d_i = \Phi\{(a_i - \sum_{j=1}^{i-1} l_{ij} y_j)/l_{ii}\}$, $e_i = \Phi\{(b_i - \sum_{j=1}^{i-1} l_{ij} y_j)/l_{ii}\}$, $y_j = \Phi^{-1}\{d_j + w_j(e_j - d_j)\}$, $l_{ij}$ is the $(i, j)$-th coefficient in the lower Cholesky factor of $\Sigma$, and $w$ is a vector in the $(n - 1)$-dimensional unit hypercube. Here in this section, I use an additional parameter $a$ in $\Phi_n(\cdot)$ as the lower integration limits to align with the original SOV algorithm but omit it elsewhere in this chapter because only MVN cumulative distribution functions are involved. Denote $e_i - d_i$ with $p_i$, which is a deterministic function of $w_1, \ldots, w_{i-1}$. It follows that $\Phi_n(a, b; 0, \Sigma) = E_w[p_1 \cdots p_n]$ and hence, the ratio in Equation (6.7) can be abstracted as:

$$
\frac{\Phi_n(a, b; \Sigma)}{\Phi_{n-1}(a_{-n}, b_{-n}; \Sigma_{-n,-n})} = \frac{E_w[p_1 \cdots p_n]}{E_{w_{-n}}[p_1 \cdots p_{n-1}]};
$$

where I use the minus sign in the subscript to denote the elimination of the corresponding row or column. Note that $p_1, \ldots, p_{n-1}$ have the same deterministic form on the numerator and the denominator. The actual implementation selects $N$ samples, $\{w^{(j)} \mid j = 1, \ldots, N\}$, from the $(n - 1)$-dimensional unit hypercube and computes

$$
\frac{\sum_{j=1}^N p_i^{(j)} \cdots p_n^{(j)} p_n^{(j)}}{\sum_{j=1}^N p_i^{(j)} \cdots p_n^{(j)} p_n^{(j)}},
$$

with $p_i^{(j)} = p_i(w^{(j)})$, which guarantees the validity of the computed probability and is shown to reduce the estimation error in the numerical simulations.

In this chapter, I focus on the improvement of the prediction accuracy when the number of observations in the probit Gaussian random field is increased. In relatively low dimensions where the processing time of the TruncateNormal package is still manageable, I compare the model selection and posterior prediction results from
the \textit{tlrmvnmvt} package with those from the \textit{TruncatedNormal} package. For higher dimensions, I show only the results from the \textit{tlrmvnmvt} package and illustrate the improvement of prediction quality when the number of observations increases.

6.4 Monte Carlo simulations

In this section, I show that efficient model selection and posterior prediction for binary random fields can be achieved based on Corollaries \ref{corollary_2} and \ref{corollary_3}, and facilitated by the TLR method from Chapter \ref{chapter_3}. The binary responses are generated from the same model as described in Section \ref{section_6.2}. Specifically, \(\xi(\cdot)\) is assumed to be zero and \(K(\cdot, \cdot)\) is assumed as the squared exponential correlation kernel:

\[ K(x_i, x_j) = \exp(-\|x_i - x_j\|^2 \times 30) \]

that has an effective range of 0.31. The squared exponential kernel is a smooth kernel and frequently used in machine learning, for which it is chosen for the simulation study. First, 10,100 locations, \(\{x_i\}_{i=1}^{10,100}\), are selected in the 2D unit square, among which the first 10,000 locations are on a \(100 \times 100\) grid and used as known observations, while the other 100 are randomly generated, representing the unknown locations used for prediction. Next, a probit Gaussian random field is generated at the 10,100 locations as the probabilities, \(\{\text{pr}(y_i = 1)\}_{i=1}^{10,100}\), with the mean and covariance structures described above. Finally, \(\{y_i\}_{i=1}^{10,000}\) are simulated as independent Bernoulli random variables. The image plots of the probit Gaussian random field and the binary random field are shown in Figure \ref{figure_6.1}.

With the knowledge of \(\{x_i\}_{i=1}^{10,000}\) and \(\{y_i\}_{i=1}^{10,000}\), I select the optimal model from the pool of squared exponential kernels, \(\{\exp(-\alpha\|x_i - x_j\|^2) \mid 15 \leq \alpha \leq 45\}\), by maximizing the marginal probability of Equation (\ref{equation_6.7}). Using the optimal model, posterior prediction is performed at the 100 unknown locations, \(\{x_i\}_{i=10,001}^{10,100}\), based on Equa-
Figure 6.1: (a) Simulated binary observations, \( \{ y_i \}_{i=1}^{10,000} \), on the 100 \times 100 grid in the unit square. (b) Simulated probit Gaussian random field on the 100 \times 100 grid in the unit square under a zero mean structure and the squared exponential correlation structure. The 100 purple circles denote the unknown locations used for prediction.

To demonstrate the improvement of prediction accuracy through using a larger number of observations, I sequentially use \( n \in \{25^2, 50^2, 75^2, 100^2\} \) observations out of the total 10,000 known observations for the model selection and the prediction of \( \{ y_i \}_{i=10,001}^{10,100} \). When \( n < 10,000 \), the locations are selected evenly from the 100 \times 100 grid to guarantee that the unknown locations, \( \{ x_i \}_{i=10,001}^{10,100} \), are contained by the selected observations. When \( n = 25^2 \), I compare the performance of the tlrvmnmt package with that of the TruncatedNormal package, whose importance sampling technique largely improves the estimation accuracy of tail probabilities. For higher dimensions, however, the marginal probability becomes smaller than the minimum value allowed by a double-precision number, for which the TruncatedNormal package simply returns zero and is unable to distinguish the optimal model in the pool.

Figure 6.2 shows the results of model selection and posterior prediction when \( n = 625 \). The TruncatedNormal package is able to maintain high accuracy for MVN probabilities of extremely small magnitudes \( (2^{-517}) \), for which its log-probability curve is smooth, whereas the tlrvmnmt package produces estimates that only indicate the
Figure 6.2: Marginal probabilities and posterior predictions of \( \{ \text{pr}(y_i = 1) \}_{i=10,001}^{10,100} \) based on 625 observations. The results on the top row are computed with the \textit{tlrmvnmvt} package and those on the bottom row are computed with the \textit{TruncatedNormal} package.

The shape of the curve. The two packages result in different values of the maximum-likelihood estimator but it is shown in the likelihood curve produced by the \textit{TruncatedNormal} package that there is a likelihood plateau when \( 28 < \alpha < 32 \). In terms of posterior prediction, the two packages have similar performance, indicating that the variance cancellation technique introduced in Section 6.3 reduces the variance of the \textit{tlrmvnmvt} package to the same level as that of the \textit{TruncatedNormal} package for the computation of marginal MVN ratios. The prediction errors are largely due to the number of observations used for prediction and hence, likely to be improved through incorporating more observations. Furthermore, the \textit{TruncatedNormal} package costs more than 20 seconds per probability estimation while the \textit{tlrmvnmvt} needs less than
5 seconds in 625 dimensions. The time difference is more significant for the estimation in higher dimensions, for example, when \( n = 50^2 \), the former takes approximately 500 seconds while the latter needs 27 seconds on average. For the ‘pmvn.thr’ function from the \texttt{tlrmvnmet} package, I use the block size of \( \sqrt{n} \), the truncation level of \( 10^{-4} \), and the Monte Carlo sample size of 20,000 and for the ‘pmvnorm’ function from the \textit{TruncatedNormal} package, I use its default settings. The computation is run on 2.5 GHz Intel Core i7.

The model selection and posterior prediction in higher dimensions are shown in Figure 6.3. It is worth noticing that the likelihood curve and the posterior prediction results are different with different numbers of observations and that the magnitudes of the likelihoods are too small for the current \textit{TruncatedNormal} package to distinguish from zero. The smoothness of the likelihood curve improves with the number of observations \( n \) but the estimation error may still lead to a non-optimal prior model. If the likelihood is assumed to be a smooth function of the parameter, proper smoothing techniques can be applied to identify the optimal model. The prediction accuracy at the same 100 unknown locations, \( \{ \text{pr}(x_i = 1) \}_{i=10}^{100} \), becomes higher as \( n \) increases, which proves that the prediction error in Figure 6.2 is principally from the lack of observations and highlights the necessity of computational methods applicable in higher dimensions. The computation times per likelihood estimation with 20,000 Monte Carlo samples are 27, 76, and 168 seconds in 2,500, 5,625, and 10,000 dimensions, respectively. Overall, it is concluded that the quality of model selection and posterior prediction is enhanced by the number of observations in the fixed domain.

### 6.5 Chapter summary

In this chapter, I extended the results from Durante (2019) to the probit Gaussian random field and showed that its posterior distribution belongs to the SUN distribution (Arellano-Valle and Azzalini, 2006). Arellano-Valle and Azzalini (2006) pro-
Figure 6.3: Marginal probabilities and posterior predictions of \( \{ pr(y_i = 1) \}_{i=10,001} \) based on 2,500, 5,625, and 10,000 observations (from top to bottom).

provided the stochastic representation and the marginal probability density that are the foundation of the efficient posterior sampling and the model selection in both Durante (2019) and this chapter. However, despite the explicit marginal probability, the model selection still involves the estimation of MVN probabilities, which becomes challenging in high dimensions. This can be tackled by the TLR method introduced...
in Chapter 3. Additionally, the posterior prediction of the probit Gaussian random field involves the computation of marginal MVN probability ratios that would have higher variability than MVN probabilities if the numerator and the denominator are computed separately. In this chapter, a variant of the SOV algorithm is proposed to reduce the estimation variability and to halve the computational costs of computing the marginal MVN probability ratios.
Chapter 7

Concluding Remarks

This thesis is centered on the efficient computation of high-dimensional multivariate normal (MVN) probabilities. As an natural extension, the multivariate Student-\(t\) (MVT) probabilities were also discussed in Chapters 3 and 5. MVN probabilities appear frequently in applications, among which Chapters 3, 5, and 6 provided three examples of the skew-normal model, the excursion sets of a Gaussian random field and the probit Gaussian random field, respectively. Therefore, the efficient estimation of MVN probabilities is directly related to the scalability of these applications. In Chapter 2, I studied the hierarchical conditioning method for MVN probabilities as an extension of the bivariate conditioning method from Trinh and Genz (2015). The method is faster than the Monte Carlo methods but it has less favorable accuracy and lacks an error estimate. With the desire for an error estimate, in Chapter 3, I improved the hierarchical Monte Carlo method from Genton et al. (2018) and introduced a tile-low-rank (TLR) Monte Carlo method with recursive block reordering that reduced the computation time for reaching the same accuracy by one order of magnitude.

The studies in Chapters 2 and 3 indicated that structured representations could yield significant computation savings for the computation of MVN probabilities, which amounts to the computation savings from associated matrix operations. In Chapter 4, I studied a novel sum-of-Kronecker-product (SKP) representation that could efficiently approximate spatial covariance matrices from large grids. The SKP representation reduces the storage cost by one order of magnitude compared with the hierarchical representation and it benefits from a log-linear construction cost. I also
showed that the Cholesky factorization could scale up to one million dimensions under stationary covariance functions on a standard scientific workstation.

Chapter 5 described an R package (R Core Team, 2019) published on CRAN that is based on the results in Chapter 3 and provided a detailed comparison with two other R packages for computing MVN probabilities. This package, tlmvnmvt, is the fastest among the three and probably the only viable option for MVN probabilities in thousands of dimensions. As an application, I showed how the computation of excursion sets of a 2D Gaussian random field was improved with the tlmvnmvt package compared with the original work in Bolin and Lindgren (2015). Chapter 6 extended the results of Durante (2019) to the probit Gaussian random field, where the efficient estimation of MVN probabilities was also needed. I showed that the posterior distribution of the probit Gaussian random field belonged to the unified skew-normal (SUN) distribution (Arellano-Valle and Azzalini, 2006) and that the model selection and posterior prediction could be greatly facilitated by the tlmvnmvt package.

Overall, this thesis contributed to the efficient computation of high-dimensional MVN probabilities based on which researchers in spatial statistics and similar fields may find their models more scalable. With the tlmvnmvt package, this scalability can be simply accessed through the R interface. The SKP representation introduced in Chapter 4 has low construction and storage costs, which may lead to a new structured representation system for kernel matrices from large grids. The theorem derived in Chapter 6 enhances the scalability of a popular model for binary classification and highlights the importance of efficient estimation of MVN probabilities.

For future improvement in computing MVN probabilities, importance sampling techniques can substitute or be combined with the block reorderings currently used in Chapter 3. When computing tail MVN probabilities, the block reordering may not be able to improve the estimation error to a satisfactory level. On the other hand, the importance sampling method introduced in Botev (2017) is highly effective for
tail probabilities but needs to solve a non-trivial optimization problem to find the proposal density, which restricts its applicability to low dimensions. Therefore, the challenge is how to find an efficient proposal density with minimal computation costs. Except for the constant and the random covariance matrices, the covariance matrices in this thesis are based on the 2D spatial domain. The challenge for approximating the covariance matrices generated from 3D or spherical spatial locations with the TLR structure is finding an indexing method that can still produce low-rank off-diagonal blocks. Assuming the validity of the low-rank feature, the TLR Quasi-Monte Carlo method is applicable. The approximation of kernel matrices generated from 3D grids with the SKP representation would be a meaningful extension of the current research. The MVN probabilities discussed in this thesis have rectangular integration regions, which after the SOV transformation, have the feature that the $i$-th integration variable is independent of the $(i + 1)$-th to the $n$-th integration variables. For non-rectangular integration regions, it is generally difficult to create this feature, for which the Monte Carlo sampling probably becomes less efficient.
REFERENCES


Biography

Jian Cao is a Ph.D. candidate supervised by Prof. Marc Genton in the Statistics Program at King Abdullah University of Science and Technology. He received his Master of Finance from Shanghai Jiao Tong University in 2016 and his Bachelor in Applied Mathematics from the University of Science and Technology of China in 2014. His research interests include scientific computing, high-performance computing, spatial statistics, and sparse matrix operations.

Publications:


• Cao, J., Genton, M. G., Keyes, D. E., & Turkiyyah, G. M. (2020). “Sum of Kronecker Products Representation for Spatial Covariance Matrices and Its Factorization,” manuscript under review