Computational Statistics and Data Analysis 157 (2021) 107165

Contents lists available at ScienceDirect

Computational Statistics and Data Analysis

journal homepage: www.elsevier.com/locate/csda

Sum of Kronecker products representation and its Cholesky factorization for spatial covariance matrices from large grids

Jian Cao [a,*], Marc G. Genton [a], David E. Keyes [a], George M. Turkiyyah [b]

[a] CEMSE Division, Extreme Computing Research Center, King Abdullah University of Science and Technology, Thuwal 23955-6900, Saudi Arabia
[b] Department of Computer Science, American University of Beirut, Beirut, Lebanon

ARTICLE INFO

Article history:
Received 4 January 2020
Received in revised form 17 December 2020
Accepted 24 December 2020
Available online 6 January 2021

Keywords:
Adaptive-cross-approximation
Cholesky factorization
Matérn covariance function
Spatial statistics
Sum of Kronecker products

ABSTRACT

The sum of Kronecker products (SKP) representation for spatial covariance matrices from gridded observations and a corresponding adaptive-cross-approximation-based framework for building the Kronecker factors are investigated. The time cost for constructing an n-dimensional covariance matrix is $O(nk^2)$ and the total memory footprint is $O(nk)$, where k is the number of Kronecker factors. The memory footprint under the SKP representation is compared with that under the hierarchical representation and found to be one order of magnitude smaller. A Cholesky factorization algorithm under the SKP representation is proposed and shown to factorize a one-million dimensional covariance matrix in under 600 seconds on a standard scientific workstation. With the computed Cholesky factor, simulations of Gaussian random fields in one million dimensions can be achieved at a low cost for a wide range of spatial covariance functions.

1. Introduction

Sparse and low-rank representations are frequently used in statistics for approximating the covariance structure, for example, the Cholesky factor of the inverse covariance matrix under the Vecchia approximation (Vecchia, 1988) and the covariance block between two separable (Hackbusch, 2015) sets of spatial locations, respectively. They typically benefit from smaller memory footprint than the default dense representation and their applications in statistics are supported by the linear algebra functionalities developed for structured matrices. However, sparse matrices are not suitable for directly approximating kernel matrices while the low-rank representation was shown in Tsiligkaridis and Hero (2013) to be less efficient than the sum of Kronecker products (SKP) representation in terms of approximating spatio-temporal covariance matrices. Specifically, Tsiligkaridis and Hero (2013) and Greenewald et al. (2013) studied the SKP and the diagonally loaded SKP representations, respectively, concluding their suitability for estimating spatio-temporal covariance structures due to their higher energy concentration in the first few principal components than the low-rank representation. Nonetheless, the spatial dimensions in Tsiligkaridis and Hero (2013) and Greenewald et al. (2013) were generally limited to below one hundred by the lack of linear algebra algorithms under the SKP representation. This paper aims to narrow this gap through developing an SKP-based Cholesky factorization algorithm that brings the applicability of the SKP representation to much higher dimensions. Furthermore, we assume only one temporal replicate and discuss the indexing and partitioning of the 2D spatial domain for a most efficient SKP representation to compare with the state-of-the-art low-rank representation,

* Corresponding author.
E-mail addresses: jian.cao@kaust.edu.sa (J. Cao), marc.genton@kaust.edu.sa (M.G. Genton), david.keyes@kaust.edu.sa (D.E. Keyes), gt02@aub.edu.lb (G.M. Turkiyyah).

https://doi.org/10.1016/j.csda.2020.107165
0167-9473/

e.g., the hierarchical matrix (Hackbusch, 2015), in up to one million dimensions, which is new in the literature. Notice that we use the term 'dimension' to interchangeably refer to the number of spatial locations and the number of rows/columns of a matrix (block), which in fact, have a corresponding relationship introduced in Section 2.

The prototype of the SKP representation is the nearest Kronecker product (NKP) problem defined in Van Loan and Pitsianis (1993), $\text{argmin}_{\mathbf{U}_1,\mathbf{V}_1}\|\boldsymbol{\Sigma}-\mathbf{U}_1\otimes\mathbf{V}_1\|_F$, with $\|\cdot\|_F$ being the Frobenius norm, which can be intuitively extended to the nearest SKP problem with k terms, $\text{argmin}_{\{\mathbf{U}_i\}_{i=1}^k,\{\mathbf{V}_i\}_{i=1}^k}\|\boldsymbol{\Sigma}-\sum_{i=1}^k\mathbf{U}_i\otimes\mathbf{V}_i\|_F$. Here, we denote the matrix under approximation by $\boldsymbol{\Sigma}$. It is different from the original work because we focus on covariance matrices. Given predefined dimensions of $\{\mathbf{U}_i\}_{i=1}^k$ and $\{\mathbf{V}_i\}_{i=1}^k$, the solution can be found through singular value decomposition (SVD) (Van Loan, 2000) and in fact, Tsiligkaridis and Hero (2013) pointed out that computing the singular vectors corresponding to the k largest singular values is sufficient, hence reducing the complexity of constructing the SKP representation from $O(n^3)$ to $O(n^2)$ for the n-dimensional matrix $\boldsymbol{\Sigma}$. In this paper, we introduce an $O(nk^2)$ construction algorithm for the SKP representation that is based on adaptive cross approximation (ACA) (Bebendorf, 2000), suitable for spatial covariance matrices from large 2D grids. The impression of 'large' when describing the matrix dimension is usually supplemented by the research topic, for example, Grasedyck et al. (2013) studied low-rank tensor approximation targeting matrices of 2^{50} dimensions while here, we consider dimensions above 10^5 as large.

The paper is organized as follows: Section 2 introduces an ACA-based algorithm for constructing the SKP representation, which amounts to a block variant of the classic ACA (Bebendorf, 2000). Section 3 discusses the quasi-optimal strategy for indexing the 2D spatial locations and choosing the dimensions of the Kronecker factors, $\{\mathbf{U}_i\}_{i=1}^k$ and $\{\mathbf{V}_i\}_{i=1}^k$, to minimize the memory footprint. In Section 4, a Cholesky factorization algorithm under the SKP representation is proposed together with its complexity analysis as a prototype for the potential linear algebra operations under the SKP representation. Section 5 provides an application example of the SKP Cholesky factor, i.e., simulation of Gaussian random fields in up to one million dimensions, which is challenging for smooth covariance kernels. Section 6 discusses the storage efficiency and the computational limitations of the SKP representation. A list of symbols containing the variables necessary for the pseudo-algorithm of the Cholesky factorization is provided in the Appendix A.

2. Nearest sum of Kronecker products

Given an indexed set of n spatial locations, $\{\mathbf{s}_1,\ldots,\mathbf{s}_n\}$, the covariance matrix $\boldsymbol{\Sigma}$ is computed with $\boldsymbol{\Sigma}(i,j)=\mathcal{K}(\mathbf{s}_i,\mathbf{s}_j)$, where $\mathcal{K}(\cdot,\cdot)$ is a spatial covariance kernel. Van Loan and Pitsianis (1993) proposed two frameworks, namely the separable least squares and the SVD frameworks, for solving the NKP problem, $\text{argmin}_{\mathbf{U}_1,\mathbf{V}_1}\|\boldsymbol{\Sigma}-\mathbf{U}_1\otimes\mathbf{V}_1\|_F$, which can be extended to the construction of the SKP representation. The separable least squares framework iteratively optimizes one Kronecker factor while fixing the others until convergence, whose rate will conceivably deteriorate as the number of terms k in the SKP representation grows. Conversely, the SVD framework produces the solution with one pass and can be directly generalized to the SKP representation. In this section, we first summarize the SVD framework, extending from the NKP problem to the construction of the SKP representation and then we introduce a new ACA framework, featuring a linear complexity and high accuracy for spatial covariance matrices from 2D grids.

2.1. SVD framework

Assuming that $\mathbf{U}_1\in\mathbb{R}^{m_1\times n_1}$, $\mathbf{V}_1\in\mathbb{R}^{m_2\times n_2}$, and the matrix dimension $n=m_1m_2=n_1n_2$, the SVD framework partitions $\boldsymbol{\Sigma}$ into blocks of the same dimensions as $\mathbf{V}_1$ to build the rearranged covariance matrix $\tilde{\boldsymbol{\Sigma}}$, whose SVD gives the solution of $\mathbf{U}_1$ and $\mathbf{V}_1$ in the form of vectors. To illustrate the rearrangement for $\tilde{\boldsymbol{\Sigma}}$ and the matricization for $\mathbf{U}_1$ and $\mathbf{V}_1$, we consider an example with $m_2=n_2=n/2$ and $m_1=n_1=2$:

$$\boldsymbol{\Sigma}=\begin{bmatrix}\boldsymbol{\Sigma}_{11} & \boldsymbol{\Sigma}_{12}\\ \boldsymbol{\Sigma}_{21} & \boldsymbol{\Sigma}_{22}\end{bmatrix}\Rightarrow\tilde{\boldsymbol{\Sigma}}=\begin{bmatrix}\text{vec}(\boldsymbol{\Sigma}_{11})^\top\\ \text{vec}(\boldsymbol{\Sigma}_{21})^\top\\ \text{vec}(\boldsymbol{\Sigma}_{12})^\top\\ \text{vec}(\boldsymbol{\Sigma}_{22})^\top\end{bmatrix}.$$

Notice that the vectorization, $\text{vec}(\cdot)$, stacks matrix columns from left to right into a single column and that $\tilde{\boldsymbol{\Sigma}}$ stacks block columns from left to right instead of block rows from top to bottom. After the rearrangement, an SVD is applied to $\tilde{\boldsymbol{\Sigma}}=[\tilde{\mathbf{U}}_1,\ldots,\tilde{\mathbf{U}}_{m_1n_1}]\boldsymbol{\Lambda}[\tilde{\mathbf{V}}_1,\ldots,\tilde{\mathbf{V}}_{m_2n_2}]^\top$, where $[\tilde{\mathbf{U}}_1,\ldots,\tilde{\mathbf{U}}_{m_1n_1}]$ and $[\tilde{\mathbf{V}}_1,\ldots,\tilde{\mathbf{V}}_{m_2n_2}]$ are orthogonal matrices and $\boldsymbol{\Lambda}$ is a diagonal matrix containing the singular values of $\tilde{\boldsymbol{\Sigma}}$, $\sigma_1,\ldots,\sigma_q$, $q=\min(m_1n_1,m_2n_2)$, in descending order. Finally, $\sigma_1\tilde{\mathbf{U}}_1$ and $\tilde{\mathbf{V}}_1$ are matricized to form $\mathbf{U}$ and $\mathbf{V}$. 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, $\|\boldsymbol{\Sigma}-\sum_{i=1}^k\mathbf{U}_i\otimes\mathbf{V}_i\|_F^2$, is $\sum_{i=k+1}^q\sigma_i^2$.

The complexity of the SVD framework is $O(n^3)$, which is prohibitively expensive in more than ten thousand dimensions but this is amenable to random sampling techniques (Halko et al., 2011) that transforms SVD into randomized SVD, hence reducing the complexity to $O(n^2)$. However, both SVD and randomized SVD fail to utilize the smoothness of spatial covariance kernels that are frequently used in statistical applications. Next, we introduce the ACA framework that exploits kernel smoothness for an $O(n)$ construction algorithm.

Table 1
Approximation of $\boldsymbol{\Sigma}$ with SVD and ACA frameworks. $\boldsymbol{\Sigma}$ is the covariance matrix built with an exponential correlation function, $\exp(-\|\mathbf{x}\|/\beta)$, $\beta = 0.3$ 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 edges of length s_2. Error measures $\|\boldsymbol{\Sigma} - \sum_{i=1}^{k} \mathbf{U}_i \otimes \mathbf{V}_i\|_F / \|\boldsymbol{\Sigma}\|_F$ and k is the number of Kronecker factors. The unit for time is seconds. Within each cell, the SVD framework is on the left while the ACA framework is on the right.

$(s_1\,,\,s_2)$	(64 , 128)		(128 , 128)		(128 , 256)		(256 , 256)	
Method	SVD	ACA	SVD	ACA	SVD	ACA	SVD	ACA
Error	2.3e−6	2.0e−6	1.4e−6	8.8e−6	*NA*	1.4e−5	*NA*	4.1e−6
k	10	11	11	11	*NA*	11	*NA*	13
Time	9.5e+1	1.4e−2	2.7e+3	3.7e−2	*NA*	6.0e−2	*NA*	2.7e−1

2.2. ACA framework

The ACA algorithm for constructing the low-rank representation of a matrix (block) can be found in Zhao et al. (2005). It has a complexity of $O(nk^2)$ because it accesses only selected rows/columns instead of all coefficients, thus utilizing the smoothness property of the kernel function. Here, we describe a block variant of the ACA algorithm that is equivalent to applying the classic ACA (Zhao et al., 2005) to $\tilde{\boldsymbol{\Sigma}}$ in Algorithm 1. We use the notation $\boldsymbol{\Sigma}_{i_b j_b}$ to denote the (i_b, j_b)th m_2-by-n_2 dimensional block of $\boldsymbol{\Sigma}$ and parentheses for indexing a coefficient. The $\langle\,,\,\rangle$ operator applied to matrices computes the inner product between the two vectorized matrices. The number of Kronecker factors k is the smallest integer such that the relative error of the SKP representation is no bigger than ϵ but unlike in the SVD framework, the measurement of relative error here is heuristic. The sensitivity of k w.r.t. ϵ is plotted in Fig. 4 of Section 3.3, where the decay of the relative error is discussed on regular grids and perturbed grids. The $\{\mathbf{U}_i\}_{i=1}^k$ and $\{\mathbf{V}_i\}_{i=1}^k$ returned by this algorithm are guaranteed to be not empty, which can be normalized by the modified Gram–Schmidt (MGS) algorithm (Björck, 1994) at a cost of $O((m_1 n_1 + m_2 n_2)k^2)$. We choose MGS over Householder QR factorization because they have similar numerical stability (Björck, 1994) but transforming a basis set with MGS produces more intuitive transformation of the coefficients, $\{\mathbf{U}_i\}_{i=1}^k$, relative to the bases than with Householder QR factorization.

Algorithm 1 Block ACA Algorithm

procedure BLKACA($\boldsymbol{\Sigma}$, m_2, n_2, ϵ)
  $n = \dim(\boldsymbol{\Sigma})$, $m_1 = \frac{n}{m_2}$, $n_1 = \frac{n}{n_2}$. Partition $\boldsymbol{\Sigma}$ into m_2-by-n_2 dimensional blocks
  $q \leftarrow 0$, $k \leftarrow 0$, $i_b \leftarrow 1$, and $j_b \leftarrow 1$
  while true **do**
    Initialize $\mathbf{U}_{k+1}$ and $\mathbf{V}_{k+1}$ as m_1-by-n_1 and m_2-by-n_2 dimensional matrices, respectively
    $\mathbf{V}_{k+1} \leftarrow \boldsymbol{\Sigma}_{i_b j_b} - \sum_{l=1}^{k} \mathbf{U}_l(i_b, j_b)\mathbf{V}_l$
    $(i, j) \leftarrow \operatorname{argmax}_{\tilde{i}\tilde{j}}(|\boldsymbol{\Sigma}_{i_b j_b}(\tilde{i}, \tilde{j})|)$, $\mathbf{V}_{k+1} \leftarrow \mathbf{V}_{k+1} / \boldsymbol{\Sigma}_{i_b j_b}(i, j)$
    $\mathbf{U}_{k+1}(i_b, j_b) \leftarrow \boldsymbol{\Sigma}_{i_b j_b}(i, j)$, $1 \le i_b \le m_1$, $1 \le j_b \le n_1$
    $\mathbf{U}_{k+1} \leftarrow \mathbf{U}_{k+1} - \sum_{l=1}^{k} \mathbf{V}_l(i, j)\mathbf{U}_l$
    $q \leftarrow q + 2\sum_{l=1}^{k} |\langle \mathbf{U}_l, \mathbf{U}_{k+1} \rangle \langle \mathbf{V}_l, \mathbf{V}_{k+1} \rangle| + \|\mathbf{U}_{k+1}\|_F^2 \|\mathbf{V}_{k+1}\|_F^2$
    if $\|\mathbf{U}_{k+1}\|_F \|\mathbf{V}_{k+1}\|_F \le \epsilon \sqrt{q}$ **then**
      return $\{\mathbf{U}_l\}_{l=1}^{k+1}$, $\{\mathbf{V}_l\}_{l=1}^{k+1}$, and $k + 1$
    end if
    $(i_b, j_b) \leftarrow \operatorname{argmax}_{\tilde{i}_b \tilde{j}_b}(\mathbf{1}_{(\tilde{i}_b, \tilde{j}_b) \neq (i_b, j_b)} |\mathbf{U}_{k+1}(\tilde{i}_b, \tilde{j}_b)|)$, $k \leftarrow k + 1$
  end while
end procedure

We refer to the approximation of $\boldsymbol{\Sigma}$ with Algorithm 1 followed by a normalizing MGS routine as the ACA framework. Table 1 compares the SVD and ACA frameworks in terms of relative error, computation time, and the number of terms k. The simulations in this paper are compiled with the Intel(R) C++ compiler with level three optimization and run on the Intel(R) Xeon(R) E5-2680 v4 @ 2.40 GHz CPU. The times in Table 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 $\boldsymbol{\Sigma}$ is of dimensions 128×128. In contrast, the ACA framework accesses only $k(m_1 n_1 + m_2 n_2)$ coefficients of $\boldsymbol{\Sigma}$, greatly reducing the construction time and is able to maintain high approximation accuracy due to the smoothness and the stationarity of the exponential kernel.

3. Quasi-optimal sum of Kronecker products

Fixing the covariance function and the spatial locations, we can choose the spatial ordering/indexing method and the dimensions of $\{\mathbf{U}_i\}_{i=1}^k$ and $\{\mathbf{V}_i\}_{i=1}^k$ for stronger linear dependence between the blocks of $\boldsymbol{\Sigma}$, which indicates a smaller k for

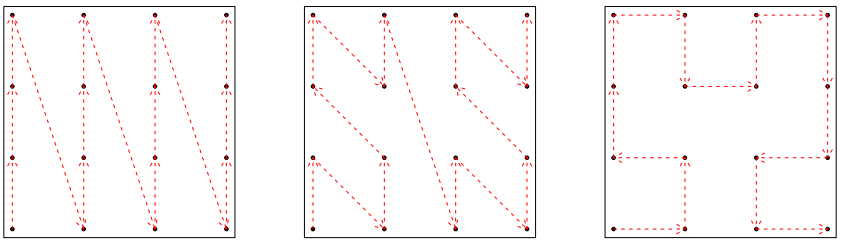


Fig. 1. Illustrations of (a) y-major order, (b) z-curve order, and (c) Hilbert curve order.

better efficiency. While the analytical solution to the above choices is not obvious, the naive choice of indexing locations along each edge of the grid and making (m_2, n_2) a factor or multiple of s_2 typically produces strong block-wise linear dependence as indicated in Table 1. In this section, we provide a more systematic analysis on the choice of indexing and partitioning, supplemented by a comparison with the hierarchical matrices (Hackbusch, 2015). Additionally, we show one limitation of the SKP representation of converging slowly when approximating covariance matrices from irregularly spaced locations.

3.1. Indexing and partitioning

Common space-filling curves used for indexing locations in multidimensional domains, e.g., $\mathbb{R}^p$ for $p \geq 2$, 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, we add the Hilbert curve to our comparison group. We also include the z-curve since it was frequently applied when using the hierarchical matrices to approximate spatial covariance matrices (Genton et al., 2018; Cao et al., 2019). A third order under consideration is the naive method that uses the x-coordinate as the primary key and the y-coordinate as the secondary key in an ascending order, which is referred to as the *y-major order* in this paper. An illustration of the three indexing methods is shown in Fig. 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, we investigate which combination of ordering and partitioning leads to the most memory-efficient SKP representation. We use the number of Kronecker products, k, and the number of double-precision values needed by $\{\mathbf{U}_i\}_{i=1}^k$ and $\{\mathbf{V}_i\}_{i=1}^k$ as two indicators for efficiency. To account for both stationary and non-stationary kernels, we first apply the exponential kernel used in Table 1 and transform it with the technique from Paciorek and Schervish (2004) to obtain a non-stationary exponential kernel. Specifically,

$$\boldsymbol{\Sigma}(i,j) = \exp\left(-\frac{\|(x_i, y_i) - (x_j, y_j)\|}{\beta_0}\right), \tag{1}$$

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

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

where $\boldsymbol{\Sigma}$ and $\boldsymbol{\Sigma}_{\text{ns}}$ are stationary and non-stationary exponential correlation matrices, respectively, (x_i, y_i) are coordinates of the spatial location $\mathbf{s}_i$ in the 2D unit square, and β_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 $\exp(-\|\mathbf{x}\|/\beta)$ 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.

Fig. 2 compares all combinations of ordering and partitioning under four types of covariance matrices. The truncation occurs at the smallest k that enables a relative error, $\|\boldsymbol{\Sigma} - \sum_{i=1}^k \mathbf{U}_i \otimes \mathbf{V}_i\|_F / \|\boldsymbol{\Sigma}\|_F$, less than 1e−5. In terms of location ordering, the y-major order is most aligned with the SKP representation, having the smallest memory footprint given most choices of (m_1, n_1, m_2, n_2), which is followed by the Hilbert curve order. Regarding partitioning, when $m_2 = n_2 = s_2$ or $m_2 = n_2 = s_2/2$, the storage costs become close to optimal. More generally, the number of double-precision values (ndbl) used is:

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

which reaches a minimum when $m_2 n_2 = n$ if k is fixed. On the other hand, k depends on the linear dependence between the partitioned blocks and typically, choosing m_2 and n_2 as a factor or multiple of s_2, the length of the edge along which the index grows by one, leads to strong block-wise linear dependence. We 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.

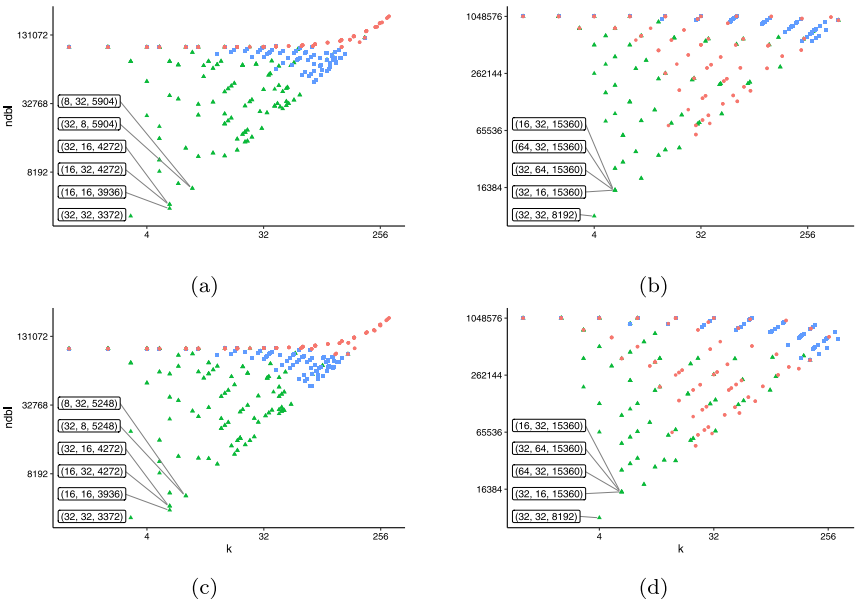


Fig. 2. Efficiency of ordering and partitioning methods. The x-axis denotes the number of Kronecker products and the y-axis (ndbl) denotes the number of double-precision values needed by $\{\mathbf{U}_i\}_{i=1}^k$ and $\{\mathbf{V}_i\}_{i=1}^k$. Green triangular dots use the y-major order, blue square dots use the z-curve order, and red circular dots use the Hilbert curve order. The spatial locations are on a 10×32 grid for (a) and (c) and a 32×32 grid for (b) and (d), with a unit distance of 1/32. The covariance functions are stationary (Eq. (1)) for (a) and (b) and non-stationary (Eq. (2)) for (c) and (d). All possible partitions are listed but only the ones having the smallest five/six ndbl are labeled with (m_2, n_2, ndbl). (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

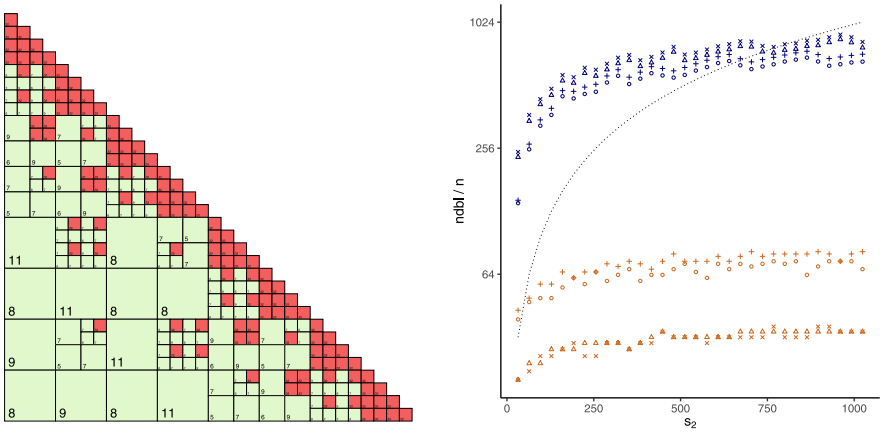
3.2. Storage costs comparison

We demonstrate the efficiency of the SKP representation through a comparison with the hierarchical matrix. The hierarchical representation of a spatial covariance matrix is constructed based on the covariance kernel and the pre-computed hierarchical block cluster tree that amounts to a recursive partition of the spatial locations into smaller clusters until either the admissibility condition is met or the partition is smaller than a predefined level; see Hackbusch (2015) for a detailed description. Its memory footprint asymptotically grows at a log-linear rate assuming regularity conditions for the underlying geometry and the covariance kernel. The whole procedure for constructing the hierarchical representation has been implemented in HLIBpro v2.7.2 (Börm et al., 2003; Kriemann, 2005; Grasedyck et al., 2008), which is used with its default hyperparameter values for the numerical studies in this section.

Fig. 3(a) is a visualization of the hierarchical matrix constructed under the same covariance kernel and 1024 spatial locations used in Fig. 2(b) with a minimum block size of 32, under which the partition becomes too fine to have a log-linear storage complexity. Relaxing the admissibility condition may integrate smaller blocks into bigger low-rank blocks but could also cause higher local ranks, potentially increasing the construction cost and the overall memory footprint. Fig. 3(b) compares the quasi-optimal SKP representation with the hierarchical representation in terms of storage growth with n, both using a relative-error-based truncation at 1e−5. Similar to Fig. 2, the comparison considers four scenarios, with different combinations of kernel stationarity and grid aspect ratio, under which the SKP representation's memory footprints are one-to-two orders smaller than those of the hierarchical representation. This is consistent with the findings in Tsiligkaridis and Hero (2013) except that here, we approximate covariance matrices of much higher dimensions.

3.3. On perturbed grids

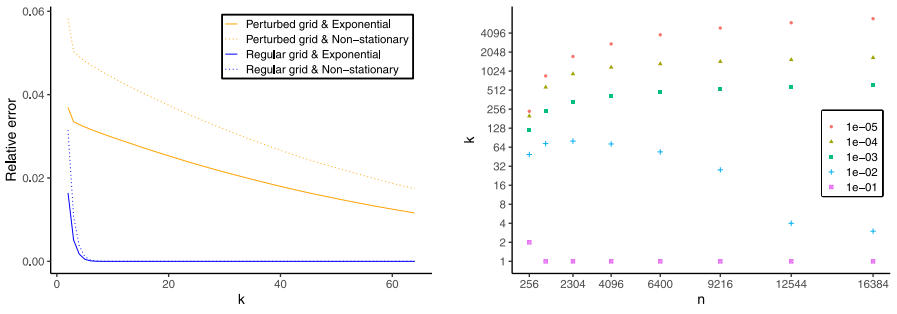
The quasi-optimal SKP representation converges quickly to the true covariance structure when the spatial locations are on a 2D regular grid. For example, a one-million dimensional stationary or non-stationary exponential correlation matrix can be closely approximated with only $k = 15$ Kronecker products based on Fig. 3(b). However, this convergence rate comes from strong linear dependence between the blocks of $\boldsymbol{\Sigma}$, which would be undermined if the spatial locations are irregularly distributed. Tsiligkaridis and Hero (2013) concluded that the SKP representation has the advantage of higher 'energy' concentration in the first few Kronecker products compared with other low-rank representations. We further this conclusion by observing that the convergence rate of the SKP representation is higher when the locations are on a 2D regular grid but lower when the spatial locations are irregularly distributed. In our experiments, we perturb a regular grid to represent the irregular geometry, which also avoids singularity and inherits the y-major order.



(a) (b)

Fig. 3. (a) An illustration of the hierarchical representation of the covariance matrix used in Fig. 2(b), built with the standard admissible condition and truncated at a relative error of 1e−5. The green and red colors denote low-rank and dense blocks, respectively, with the bottom-left numbers indicating the (local) ranks. (b) The ratio between the number of double precision values allocated and the number of spatial locations under the hierarchical representation (blue, top four curves) and the SKP representation (brown, bottom four curves). 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 'o' shaped dots. The spatial correlation kernel is Eq. (1) for 'o' and '△' and Eq. (2) for '+ ' and '×'. The black dotted line is $\text{ndbl} = s_1^2 s_2$, denoting an $O(n^{3/2})$ growth rate. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

Fig. 4 compares the convergence rate for the regular grid and the perturbed grid that is obtained through adding $\frac{U}{s_2}$ to the coordinates of each location on the regular grid, where $U \sim \text{Uniform}(0.0, 0.8)$. For all four combinations of grid type and stationarity in Fig. 4(a), the relative errors reduce to below 10% with only two Kronecker products, indicating high 'energy' concentration at the beginning of the SKP representation. However, the convergence rate differs significantly between regular and perturbed grids, with the former converging quickly to zero and the latter still having a relative error greater than 1% at $k = 60$. Fig. 4(b) shows that using the SKP representation to achieve a decent relative error, e.g., 1e−5, under irregular geometries is already unrealistic in thousands of dimensions because a k value of hundreds typically produces very limited storage and computation savings. On the other hand, a larger relative error is more likely to cause non-positive definite issues, especially when the spatial locations are compactly distributed. The reason for the slow convergence rate under irregular geometries resembles adding independent noises to the blocks of a matrix, for which the SKP representation converges quickly. Consider a simplified example where all matrix blocks are equal to one common block perturbed by independent noises of a smaller magnitude. There is high 'energy' concentration in the first



(a) (b)

Fig. 4. (a) The convergence of relative error with the number of Kronecker products. The approximated covariance matrices are built with the exponential kernel Eq. (1) and the non-stationary exponential kernel Eq. (2) based on regular and perturbed 32 × 32 grids 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 kernel Eq. (1) based on a $\sqrt{n} \times \sqrt{n}$ perturbed grid in the unit square.

Kronecker product if $\mathbf{V}_1$ is the common block but each following Kronecker product only has marginal impact on the relative error, hence we think that the SKP representation is not suitable for irregular geometries in 2D.

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, we discuss the feasibility of Cholesky factorization under the SKP representation, focusing on its scalability and robustness.

4.1. Basic ideas and notations

Given an SKP representation, each block of $\boldsymbol{\Sigma}$ is approximated by a linear combination of $\{\mathbf{V}_i\}_{i=1}^{k}$ and hence, we refer to $\{\mathbf{V}_i\}_{i=1}^{k}$ as the basis blocks and $\{\mathbf{U}_i\}_{i=1}^{k}$ as the coefficient blocks relative to $\{\mathbf{V}_i\}_{i=1}^{k}$. Recall that we defined the inner product between two matrix blocks as the inner product between the two vectorized blocks in Section 2.1. When $\{\mathbf{V}_i\}_{i=1}^{k}$ are orthonormal, the Frobenius norm of the (i_b, j_b)th block in the SKP representation is $\sqrt{\sum_{i=1}^{k} \mathbf{U}_i(i_b, j_b)^2}$. Three matrices are involved in the Cholesky factorization algorithm, namely the covariance matrix $\boldsymbol{\Sigma}$, its lower Cholesky factor, $\mathbf{L}$, and the inverse of $\mathbf{L}$, $\mathbf{D}$, whose SKP representations are respectively denoted by:

$$\boldsymbol{\Sigma} \approx \sum_{i=1}^{k^{\boldsymbol{\Sigma}}} \mathbf{U}_i^{\boldsymbol{\Sigma}} \otimes \mathbf{V}_i^{\boldsymbol{\Sigma}}, \quad \mathbf{L} \approx \sum_{i=1}^{k^{\mathbf{L}}} \mathbf{U}_i^{\mathbf{L}} \otimes \mathbf{V}_i^{\mathbf{L}}, \quad \mathbf{D} \approx \sum_{i=1}^{k^{\mathbf{D}}} \mathbf{U}_i^{\mathbf{D}} \otimes \mathbf{V}_i^{\mathbf{D}}.$$

Additionally, the Schur complement is computed at each block column as the difference between $\boldsymbol{\Sigma}$ and a non-negative definite matrix, for which we use $\boldsymbol{\Sigma}_{[j_b]}$ and $k_{j_b}^{\boldsymbol{\Sigma}}$ to denote the Schur complement at block column j_b and the number of basis blocks for $\boldsymbol{\Sigma}_{[j_b]}$, respectively and $\boldsymbol{\Sigma}_{[1]}$ is defined as the original covariance matrix $\boldsymbol{\Sigma}$. The SKP representation for $\boldsymbol{\Sigma}_{[1]}$ is the same as that for $\boldsymbol{\Sigma}$ and the SKP representation for $\boldsymbol{\Sigma}_{[j_b]}$ is updated based on the SKP representation for $\boldsymbol{\Sigma}_{[j_b-1]}$:

$$\boldsymbol{\Sigma}_{[j_b]} \approx \sum_{i=1}^{k_{j_b}^{\boldsymbol{\Sigma}}} \mathbf{U}_i^{\boldsymbol{\Sigma}_{[j_b]}} \otimes \mathbf{V}_i^{\boldsymbol{\Sigma}_{[j_b]}}, \quad \{\mathbf{V}_i\}_{i=1}^{k^{\boldsymbol{\Sigma}}} = \{\mathbf{V}_i^{\boldsymbol{\Sigma}_{[1]}}\}_{i=1}^{k^{\boldsymbol{\Sigma}_{[1]}}} \subset \cdots \subset \{\mathbf{V}_i^{\boldsymbol{\Sigma}_{[n_1]}}\}_{i=1}^{k^{\boldsymbol{\Sigma}_{[n_1]}}}.$$

When considering the Cholesky factorization algorithm under the SKP representation, we assume $m_2 = n_2$ since the diagonal blocks must be square for the dense Cholesky factorization and matrix inverse.

The fundamental idea is a block Cholesky factorization where each block is represented by a coordinate vector relative to basis blocks. Furthermore, the basis blocks are initially incomplete, with new basis blocks dynamically discovered during the iterations through the block columns. At the discovery of a new basis block, its multiplications with existing basis blocks are computed and stored for the remaining computation, which is referred to as recycling the matrix multiplication between basis blocks in this paper. Algorithm 2 describes the block Cholesky factorization algorithm (Akbudak et al., 2017), to which we associate code sections of the SKP Cholesky factorization in Algorithm 3. Specifically, the colored segments in Algorithms 2 (Lines 3-7 and Lines 10-12) and 3 (Lines 6-19 and Lines 23-31) correspond to each other in terms of functionalities, i.e., computing the Schur complement and the Cholesky factor at block column j_b, respectively.

Algorithm 2 Block Cholesky Algorithm

```
1: procedure BLKCHOL(Σ)
2:     for j_b = 1 : m_1 do
3:         if j_b > 1 then
4:             for i_b = j_b : m_1 do
5:                 Σ_{i_b j_b} ← Σ_{i_b j_b} − L_{i_b,1:j_b−1} L^⊤_{j_b,1:j_b−1}
6:             end for
7:         end if
8:         L_{j_b j_b} ← cholesky(Σ_{j_b j_b})
9:         D_{j_b j_b} ← inverse(L_{j_b j_b})
10:        for i_b = (j_b + 1) : m_1 do
11:            L_{i_b j_b} ← Σ_{i_b j_b} D^⊤_{j_b j_b}
12:        end for
13:    end for
14: end procedure
```

We first describe how the newly computed blocks of $\boldsymbol{\Sigma}_{[j_b]}$ and $\mathbf{L}$ are represented with coordinate systems. Since the basis blocks for $\boldsymbol{\Sigma}_{[j_b]}$ are obtained by adding new basis blocks to $\{\mathbf{V}_i^{\boldsymbol{\Sigma}}\}_{i=1}^{k^{\boldsymbol{\Sigma}}}$, we add the new basis blocks directly to the

Algorithm 3 Cholesky factorization under the SKP representation

1: **procedure** KCHOL($\{\mathbf{U}_i^{\mathbf{\Sigma}}\}_{i=1}^{k^{\mathbf{\Sigma}}}$, $\{\mathbf{V}_i^{\mathbf{\Sigma}}\}_{i=1}^{k^{\mathbf{\Sigma}}}$, $k^{\mathbf{\Sigma}}$)
2: Initialize $\mathbf{Q}^{\mathbf{H}} = [\mathrm{vec}(\mathbf{V}_1^{\mathbf{\Sigma}}) \cdots \mathrm{vec}(\mathbf{V}_{k^{\mathbf{\Sigma}}}^{\mathbf{\Sigma}})]$, $\mathbf{R}^{\mathbf{H}}$, $\mathbf{Q}^{\mathbf{S}}$ and $\mathbf{R}^{\mathbf{S}}$ as empty matrices
3: Initialize $\mathbf{C}^{\mathbf{\Sigma}} = \mathbf{I}_{k^{\mathbf{\Sigma}}}$ and $\mathbf{C}^{\mathbf{L}}$ as an empty matrix
4: Initialize $\{\mathbf{L}_{i,i}\}_{i=1}^{m_1}$ and $\{\mathbf{D}_{i,i}\}_{i=1}^{m_1}$ as empty matrices, recall that $m_1 = n_1$
5: **for** $j_b = 1 : m_1$ **do**
6: **if** $j_b > 1$ **then**
7: Initialize temporary storage $\mathbf{A} \in \mathbb{R}^{k^{\mathbf{L}} \times k^{\mathbf{L}}, m_1 - j_b + 1}$
8: **for** $i_b = j_b : m_1$, $l_1 = 1 : k^{\mathbf{L}}$, $l_2 = 1 : k^{\mathbf{L}}$ **do**
9: $l_3 \leftarrow (l_1 - 1)k^{\mathbf{L}} + l_2$
10: $\mathbf{A}(l_3, i_b + 1 - j_b) \leftarrow \mathbf{U}_{l_1}^{\mathbf{L}}(i_b, 1 : j_b - 1)\mathbf{U}_{l_2}^{\mathbf{L}\top}(j_b, 1 : j_b - 1)$
11: **end for** ▷ Columns of $\mathbf{A}$: coordinates of $\mathbf{L}_{j_b:m_1,1:j_b-1}\mathbf{L}_{j_b,1:j_b-1}^{\top}$ w.r.t. $\mathbf{H}_{j_b}$
12: $\mathbf{A} \leftarrow \mathbf{R}^{\mathbf{H}}\mathbf{A}$, PROJ($\mathbf{A}, \mathbf{Q}^{\mathbf{H}}, \mathbf{C}^{\mathbf{\Sigma}}, \{\mathbf{U}_i^{\mathbf{\Sigma}}\}_{i=1}^{k^{\mathbf{\Sigma}}}, \{\mathbf{V}_i^{\mathbf{\Sigma}}\}_{i=1}^{k^{\mathbf{\Sigma}}}, k^{\mathbf{\Sigma}}$), $\mathbf{A} \leftarrow \mathbf{C}^{\mathbf{\Sigma}\top}\mathbf{A}$
13: If $k^{\mathbf{\Sigma}}$ is increased in Line 12, update $\mathbf{Q}^{\mathbf{S}}$, $\mathbf{R}^{\mathbf{S}}$, and $\mathbf{C}^{\mathbf{L}}$
14: **else**
15: Initialize $\mathbf{A} \in \mathbb{R}^{k^{\mathbf{\Sigma}}, m_1 - j_b + 1}$, $\mathbf{A} \leftarrow \mathbf{0}$
16: **end if**
17: **for** $l = 1 : k^{\mathbf{\Sigma}}$ **do**
18: $\mathbf{A}(l, 1 : m_1 - j_b + 1) \leftarrow \mathbf{U}_l^{\mathbf{\Sigma}\top}(j_b : m_1, j_b) - \mathbf{A}(l, 1 : m_1 - j_b + 1)$
19: **end for** ▷ Columns of $\mathbf{A}$: coordinates of 1st block column in $\mathbf{\Sigma}_{[j_b]}$ w.r.t. $\{\mathbf{V}_i^{\mathbf{\Sigma}}\}_{i=1}^{k_{j_b}^{\mathbf{\Sigma}}}$
20: $\mathbf{B} \leftarrow \sum_{l=1}^{k^{\mathbf{\Sigma}}} \mathbf{A}(l, 1)\mathbf{V}_l^{\mathbf{\Sigma}}$, $\mathbf{L}_{j_b,j_b} \leftarrow$ cholesky($\mathbf{B}$), $\mathbf{D}_{j_b,j_b} \leftarrow$ inverse($\mathbf{L}_{j_b,j_b}$)
21: PROJ($\mathbf{D}_{j_b,j_b}, \{\mathbf{U}_i^{\mathbf{D}}\}_{i=1}^{k^{\mathbf{D}}}, \{\mathbf{V}_i^{\mathbf{D}}\}_{i=1}^{k^{\mathbf{D}}}, k^{\mathbf{D}}, j_b$)
22: If $k^{\mathbf{D}}$ is increased in Line 21, update $\mathbf{Q}^{\mathbf{S}}$, $\mathbf{R}^{\mathbf{S}}$, and $\mathbf{C}^{\mathbf{L}}$
23: Reset $\mathbf{B} \in \mathbb{R}^{k^{\mathbf{\Sigma}}k^{\mathbf{D}}, m_1 - j_b}$
24: **for** $i_b = j_b + 1 : m_1$, $l_1 = 1 : k^{\mathbf{\Sigma}}$, $l_2 = 1 : k^{\mathbf{D}}$ **do**
25: $l_3 \leftarrow (l_1 - 1)k^{\mathbf{D}} + l_2$, $\mathbf{B}(l_3, i_b - j_b) \leftarrow \mathbf{A}(l_1, i_b - j_b + 1)\mathbf{U}_{l_2}^{\mathbf{D}}(j_b, j_b)$
26: **end for** ▷ Columns of $\mathbf{B}$: coordinates of $\mathbf{L}_{j_b+1:m_1,j_b}$ w.r.t. $\mathbf{S}_{j_b}$
27: $\mathbf{B} \leftarrow \mathbf{R}^{\mathbf{S}}\mathbf{B}$, PROJ($\mathbf{B}, \mathbf{Q}^{\mathbf{S}}, \mathbf{C}^{\mathbf{L}}, \{\mathbf{U}_i^{\mathbf{L}}\}_{i=1}^{k^{\mathbf{L}}}, \{\mathbf{V}_i^{\mathbf{L}}\}_{i=1}^{k^{\mathbf{L}}}, k^{\mathbf{L}}$), $\mathbf{B} \leftarrow \mathbf{C}^{\mathbf{L}\top}\mathbf{B}$
28: If $k^{\mathbf{L}}$ is increased in Line 27, update $\mathbf{Q}^{\mathbf{H}}$, $\mathbf{R}^{\mathbf{H}}$, and $\mathbf{C}^{\mathbf{\Sigma}}$
29: **for** $l = 1 : k^{\mathbf{L}}$ **do**
30: $\mathbf{U}_l^{\mathbf{L}}(j_b + 1 : m_1, j_b) \leftarrow \mathbf{B}^{\top}(l, 1 : m_1 - j_b)$
31: **end for** ▷ Columns of $\mathbf{B}$: coordinates of $\mathbf{L}_{j_b+1:m_1,j_b}$ w.r.t. $\{\mathbf{V}_i^{\mathbf{L}}\}_{i=1}^{k_{j_b}^{\mathbf{L}}}$
32: **end for**
33: **for** $j_b = 1 : m_1$ **do**
34: PROJ($\mathbf{L}_{j_b,j_b}, \{\mathbf{U}_i^{\mathbf{L}}\}_{i=1}^{k^{\mathbf{L}}}, \{\mathbf{V}_i^{\mathbf{L}}\}_{i=1}^{k^{\mathbf{L}}}, k^{\mathbf{L}}, j_b$)
35: **end for** ▷ Project dense diagonal blocks to $\{\mathbf{V}_i^{\mathbf{L}}\}_{i=1}^{k^{\mathbf{L}}}$
36: **return** $\{\mathbf{U}_i^{\mathbf{L}}\}_{i=1}^{k^{\mathbf{L}}}$, $\{\mathbf{V}_i^{\mathbf{L}}\}_{i=1}^{k^{\mathbf{L}}}$, $\{\mathbf{U}_i^{\mathbf{D}}\}_{i=1}^{k^{\mathbf{D}}}$, $\{\mathbf{V}_i^{\mathbf{D}}\}_{i=1}^{k^{\mathbf{D}}}$
37: **end procedure**

existing basis blocks of $\mathbf{\Sigma}$ for better efficiency and pad existing coordinate vectors relative to $\{\mathbf{V}_i^{\mathbf{\Sigma}}\}_{i=1}^{k^{\mathbf{\Sigma}}}$ with zeros so that its linear space contains blocks of $\mathbf{\Sigma}_{[j_b]}$. Because the numbers of basis blocks, $k^{\mathbf{\Sigma}}$, $k^{\mathbf{L}}$, and $k^{\mathbf{D}}$, grow with the block column index j_b, we use $k_{j_b}^{\mathbf{\Sigma}}$, $k_{j_b}^{\mathbf{L}}$, and $k_{j_b}^{\mathbf{D}}$ to denote the number of basis blocks for $\mathbf{\Sigma}_{[j_b]}$, $\mathbf{L}$, and $\mathbf{D}$ before computing the Cholesky factor at the j_bth block column, which leads to the following relationship:

$$k^{\mathbf{\Sigma}} = k_1^{\mathbf{\Sigma}} \le \cdots \le k_{n_1}^{\mathbf{\Sigma}}, \quad 0 = k_1^{\mathbf{L}} \le \cdots \le k_{n_1}^{\mathbf{L}} = k^{\mathbf{L}}, \quad 0 = k_1^{\mathbf{D}} \le \cdots \le k_{n_1}^{\mathbf{D}} = k^{\mathbf{D}}.$$

Line 11 of Algorithm 2 indicates that the off-diagonal blocks in the j_bth block column of $\mathbf{L}$ can be stored by $k_{j_b}^{\mathbf{\Sigma}}k_{j_b}^{\mathbf{D}}$ coordinates with respect to $\mathcal{S}_{j_b} = \{\mathbf{V}_{l_1}^{\mathbf{\Sigma}}\mathbf{V}_{l_2}^{\mathbf{D}\top}, l_1 = 1, \ldots, k_{j_b}^{\mathbf{\Sigma}}, l_2 = 1, \ldots, k_{j_b}^{\mathbf{D}}\}$, which is implemented at Line 25 of Algorithm 3. Similarly, the blocks of the Schur complement computed in Line 5 of Algorithm 2 belong to the linear space spanned by $\mathcal{H}_{j_b} = \{\mathbf{V}_i^{\mathbf{\Sigma}}\}_{i=1}^{k^{\mathbf{\Sigma}}} \cup \{\mathbf{V}_{l_1}^{\mathbf{L}}\mathbf{V}_{l_2}^{\mathbf{L}\top}, l_1 = 1, \ldots, k_{j_b}^{\mathbf{L}}, l_2 = 1, \ldots, k_{j_b}^{\mathbf{L}}\}$ and Line 10 of Algorithm 3 computes the coordinates of $\mathbf{L}_{i_b,1:j_b-1}\mathbf{L}_{j_b,1:j_b-1}^{\top}$ relative to $\{\mathbf{V}_{l_1}^{\mathbf{L}}\mathbf{V}_{l_2}^{\mathbf{L}\top}, l_1 = 1, \ldots, k_{j_b}^{\mathbf{L}}, l_2 = 1, \ldots, k_{j_b}^{\mathbf{L}}\}$. However, not knowing an orthogonal basis set of $\mathcal{S}_{j_b}$ and $\mathcal{H}_{j_b}$ poses difficulties for discovering new bases through projection, for which we further apply a rank-revealing QR

decomposition (Chan, 1987) to the vectorized default bases of $\mathcal{S}_{j_b}$ and $\mathcal{H}_{j_b}$, denoted by $\mathbf{S}_{j_b}$ and $\mathbf{H}_{j_b}$, respectively:

$$\mathbf{S}_{j_b} = \mathbf{Q}^{\mathbf{S}}_{j_b}\mathbf{R}^{\mathbf{S}}_{j_b} = \left[\mathrm{vec}(\mathbf{V}^{\mathbf{\Sigma}}_1\mathbf{V}^{\mathbf{D}\top}_1)\cdots\mathrm{vec}(\mathbf{V}^{\mathbf{\Sigma}}_1\mathbf{V}^{\mathbf{D}\top}_{k^{\mathbf{D}}_{j_b}})\cdots\mathrm{vec}(\mathbf{V}^{\mathbf{\Sigma}}_{k^{\mathbf{\Sigma}}_{j_b}}\mathbf{V}^{\mathbf{D}\top}_{k^{\mathbf{D}}_{j_b}})\right], \tag{3}$$

$$\mathbf{H}_{j_b} = \mathbf{Q}^{\mathbf{H}}_{j_b}\tilde{\mathbf{R}}^{\mathbf{H}}_{j_b} = \left[\mathrm{vec}(\mathbf{V}^{\mathbf{\Sigma}}_1)\cdots\mathrm{vec}(\mathbf{V}^{\mathbf{\Sigma}}_{k^{\mathbf{\Sigma}}})\ \mathrm{vec}(\mathbf{V}^{\mathbf{L}}_1\mathbf{V}^{\mathbf{L}\top}_1)\cdots\mathrm{vec}(\mathbf{V}^{\mathbf{L}}_1\mathbf{V}^{\mathbf{L}\top}_{k^{\mathbf{L}}_{j_b}})\cdots\mathrm{vec}(\mathbf{V}^{\mathbf{L}}_{k^{\mathbf{L}}_{j_b}}\mathbf{V}^{\mathbf{L}\top}_{k^{\mathbf{L}}_{j_b}})\right], \tag{4}$$

$$\tilde{\mathbf{R}}^{\mathbf{H}}_{j_b} = \left[\begin{pmatrix}\mathbf{I}_{k^{\mathbf{\Sigma}}\times k^{\mathbf{\Sigma}}}\\ \mathbf{0}\end{pmatrix}\quad \mathbf{R}^{\mathbf{H}}_{j_b}\right],\quad \left[\mathrm{vec}(\mathbf{V}^{\mathbf{L}}_1\mathbf{V}^{\mathbf{L}\top}_1)\cdots\mathrm{vec}(\mathbf{V}^{\mathbf{L}}_{k^{\mathbf{L}}}\mathbf{V}^{\mathbf{L}\top}_{k^{\mathbf{L}}})\right] = \mathbf{Q}^{\mathbf{H}}_{j_b}\mathbf{R}^{\mathbf{H}}_{j_b}.$$

Left multiplication with the $\mathbf{R}^{\mathbf{S}}_{j_b}$ and $\mathbf{R}^{\mathbf{H}}_{j_b}$ transforms the coordinates computed at Lines 25 and 10 of Algorithm 3 to coordinates relative to column-orthogonal $\mathbf{Q}^{\mathbf{S}}_{j_b}$ and $\mathbf{Q}^{\mathbf{H}}_{j_b}$, which is implemented in the first part of Lines 27 and 12 of Algorithm 3, respectively. Notice that $\{\mathrm{vec}(\mathbf{V}^{\mathbf{\Sigma}_{[j_b]}}_i)\}_{i=1}^{k^{\mathbf{\Sigma}}_{j_b}}$ and $\{\mathrm{vec}(\mathbf{V}^{\mathbf{L}}_i)\}_{i=1}^{k^{\mathbf{L}}_{j_b}}$ are in the linear space spanned by $\mathbf{Q}^{\mathbf{H}}_{j_b}$ and $\mathbf{Q}^{\mathbf{S}}_{j_b}$, for which we can create computation savings by storing the basis blocks as coordinates relative to $\mathbf{Q}^{\mathbf{H}}_{j_b}$ and $\mathbf{Q}^{\mathbf{S}}_{j_b}$:

$$\left[\mathrm{vec}(\mathbf{V}^{\mathbf{\Sigma}_{[j_b]}}_1)\quad \mathrm{vec}(\mathbf{V}^{\mathbf{\Sigma}_{[j_b]}}_2)\quad \cdots\quad \mathrm{vec}(\mathbf{V}^{\mathbf{\Sigma}_{[j_b]}}_{k^{\mathbf{\Sigma}}_{j_b}})\right] = \mathbf{Q}^{\mathbf{H}}_{j_b}\mathbf{C}^{\mathbf{\Sigma}}_{j_b}, \tag{5}$$

$$\left[\mathrm{vec}(\mathbf{V}^{\mathbf{L}}_1)\quad \mathrm{vec}(\mathbf{V}^{\mathbf{L}}_2)\quad \cdots\quad \mathrm{vec}(\mathbf{V}^{\mathbf{L}}_{k^{\mathbf{L}}_{j_b}})\right] = \mathbf{Q}^{\mathbf{S}}_{j_b}\mathbf{C}^{\mathbf{L}}_{j_b}. \tag{6}$$

Here, the $\mathbf{C}$ matrices also have orthogonal columns. The transformed coordinates computed in the first part of Lines 27 and 12 of Algorithm 3 are then projected onto the existing basis blocks of $\mathbf{C}^{\mathbf{L}}_{j_b}$ and $\mathbf{C}^{\mathbf{\Sigma}}_{j_b}$, whose vector rejections are normalized and appended as a new basis at the last column if their Frobenius norm is above a predefined tolerance level, which corresponds to the second part of Lines 27 and 12 in Algorithm 3. Left multiplication with the transpose of $\mathbf{C}^{\mathbf{\Sigma}}_{j_b}$ and $\mathbf{C}^{\mathbf{L}}_{j_b}$ transforms the coordinates relative to $\mathbf{Q}^{\mathbf{H}}_{j_b}$ and $\mathbf{Q}^{\mathbf{S}}_{j_b}$ to those relative to $\{\mathrm{vec}(\mathbf{V}^{\mathbf{\Sigma}_{[j_b]}}_i)\}_{i=1}^{k^{\mathbf{\Sigma}}_{j_b}}$ and $\{\mathrm{vec}(\mathbf{V}^{\mathbf{L}}_i)\}_{i=1}^{k^{\mathbf{L}}_{j_b}}$, concluding the coordinate representation of newly computed blocks in $\mathbf{\Sigma}_{[j_b]}$ and $\mathbf{L}$ as shown in the last part of Lines 27 and 12 in Algorithm 3. It is worth mentioning that $\mathbf{R}^{\mathbf{H}}$ is not upper triangular and hence, general matrix–matrix multiplication should be used for the left multiplication with $\mathbf{R}^{\mathbf{H}}$.

The second technique for generating computation savings is recycling the matrix multiplication results, which is involved in Lines 5 and 11 of Algorithm 2. Here, we expand Line 11 to elaborate the recycling process:

$$\begin{aligned}\mathbf{\Sigma}_{i_bj_b}\mathbf{D}^{\top}_{j_bj_b} &= \left(\sum_{l_1=1}^{k^{\mathbf{\Sigma}}_{j_b}}\mathbf{U}^{\mathbf{\Sigma}}_{l_1}(i_b,j_b)\mathbf{V}^{\mathbf{\Sigma}}_{l_1}\right)\left(\sum_{l_2=1}^{k^{\mathbf{D}}_{j_b}}\mathbf{U}^{\mathbf{D}}_{l_2}(j_b,j_b)\mathbf{V}^{\mathbf{D}\top}_{l_2}\right)\\ &= \sum_{l_1=1}^{k^{\mathbf{\Sigma}}_{j_b}}\sum_{l_2=1}^{k^{\mathbf{D}}_{j_b}}\mathbf{U}^{\mathbf{\Sigma}}_{l_1}(i_b,j_b)\mathbf{U}^{\mathbf{D}}_{l_2}(j_b,j_b)\mathbf{V}^{\mathbf{\Sigma}}_{l_1}\mathbf{V}^{\mathbf{D}\top}_{l_2}.\end{aligned} \tag{7}$$

Thus, $\mathbf{V}^{\mathbf{\Sigma}}_{l_1}\mathbf{V}^{\mathbf{D}\top}_{l_2}$ can be stored and reused for later iterations, which is implemented in the definitions of $\mathbf{Q}^{\mathbf{S}}_{j_b}\mathbf{R}^{\mathbf{S}}_{j_b}$ and $\mathbf{Q}^{\mathbf{H}}_{j_b}\tilde{\mathbf{R}}^{\mathbf{H}}_{j_b}$.

The third technique for improving efficiency and numerical stability is projecting the Gaussian mixture (Martinsson, 2011) of newly computed blocks in $\mathbf{\Sigma}_{[j_b]}$ and $\mathbf{L}$ onto existing basis blocks represented by $\mathbf{C}^{\mathbf{\Sigma}}_{j_b}$ and $\mathbf{C}^{\mathbf{L}}_{j_b}$. Consider Line 12 in Algorithm 3, $m_1 - j_b$ blocks are projected onto $\mathbf{C}^{\mathbf{\Sigma}}_{j_b}$. If the projection occurs sequentially, each adding a new basis if its vector rejection is greater than the tolerance level, then the approximation errors for the $m_1 - j_b$ blocks are likely to be distributed unevenly, with the projections leading to new bases having zero approximation error and others having higher errors. On the other hand, the number of new bases may depend on the order of projection among the $m_1 - j_b$ blocks. To illustrate this point, we provide a simplified example of applying the MGS algorithm to the columns of

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

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 a stable and potentially smaller number of basis blocks, we obtain a Gaussian mixture of the blocks to be projected prior to projecting them onto existing basis blocks, which is indicated by 'random column sampling' in Line 2 of Algorithm 4. Additionally, this helps distribute the approximation error evenly across the blocks in the j_bth block column.

Algorithm 4 Discover new basis blocks I

1: **procedure** PROJ($\mathbf{A}$, $\mathbf{Q}$, $\mathbf{C}$, $\{\mathbf{U}_i\}_{i=1}^k$, $\{\mathbf{V}_i\}_{i=1}^k$, k)
2: Initialize temporary storage $\mathbf{B} \leftarrow$ random column sampling($\mathbf{A}$)
3: **for** $j = 1 : \text{ncol}(\mathbf{B})$ **do**
4: $\mathbf{u}, \mathbf{v} \leftarrow$ modified Gram–Schmidt($\mathbf{B}(:, j)$, $\mathbf{C}$)
5: **if** $\|\mathbf{v}\| > \epsilon$ **then**
6: $\mathbf{C} \leftarrow [\mathbf{C}, \mathbf{v}/\|\mathbf{v}\|]$, $\mathbf{U}_{k+1} \leftarrow \mathbf{0}$, $\text{vec}(\mathbf{V}_{k+1}) \leftarrow \mathbf{Q}\mathbf{v}$, $k \leftarrow k + 1$
7: **end if**
8: **end for**
9: **end procedure**

Algorithm 5 Discover new basis blocks II

1: **procedure** PROJ($\mathbf{A}$, $\{\mathbf{U}_i\}_{i=1}^k$, $\{\mathbf{V}_i\}_{i=1}^k$, k, j)
2: $\mathbf{u}, \mathbf{v} \leftarrow$ modified Gram–Schmidt($\text{vec}(\mathbf{A})$, $[\text{vec}(\mathbf{V}_1), \ldots, \text{vec}(\mathbf{V}_k)]$)
3: **for** $l = 1 : k$ **do**
4: $\mathbf{U}_l(j, j) \leftarrow \mathbf{u}(l)$
5: **end for**
6: **if** $\|\mathbf{v}\| > \epsilon$ **then**
7: $\mathbf{U}_{k+1} \leftarrow \mathbf{0}$, $\mathbf{U}_{k+1}(j, j) \leftarrow 1$, $\text{vec}(\mathbf{V}_{k+1}) \leftarrow \mathbf{Q}\mathbf{v}$, $k \leftarrow k + 1$
8: **end if**
9: **end procedure**

4.2. Implementation details and complexity analysis

The input SKP representation of the covariance matrix for Algorithm 3 can be efficiently produced by the ACA framework introduced in Section 2.2. Notice that $\{\mathbf{V}_i^{\mathbf{\Sigma}}\}_{i=1}^{k^{\mathbf{\Sigma}}}$ are orthogonal to each other based on the inner product between matrix blocks defined in this paper because the MGS algorithm is included in the ACA framework. In Algorithm 3, the block column index j_b, defined in Line 5, is implicitly included in $k^{\mathbf{\Sigma}}$, $k^{\mathbf{L}}$, $k^{\mathbf{D}}$, $\mathbf{Q}^{\mathbf{H}}$, $\mathbf{R}^{\mathbf{H}}$, $\mathbf{Q}^{\mathbf{S}}$, $\mathbf{R}^{\mathbf{S}}$, $\mathbf{C}^{\mathbf{\Sigma}}$, and $\mathbf{C}^{\mathbf{L}}$ because they are updated by either larger values or appended rows/columns when j_b increases and hence, stored in the same variables for better efficiency. The matrices $\mathbf{A}$ and $\mathbf{B}$ in Algorithm 3 are temporary variables used to store intermediate coordinates or dense matrices. Specifically, in Line 10, the $(i_b - j_b + 1)$th column of $\mathbf{A}$ stores the coordinates of $\mathbf{L}_{i_b,1:j_b-1}\mathbf{L}_{j_b,1:j_b-1}^{\top}$ relative to $\mathbf{H}_{j_b}$, which is transformed to coordinates relative to $\{\mathbf{V}_i^{\mathbf{\Sigma}}\}_{i=1}^{k_{j_b}^{\mathbf{\Sigma}}}$ after Line 12; in Line 18, the jth column of $\mathbf{A}$ stores the coordinates of the $(j, 1)$th block in $\mathbf{\Sigma}_{[j_b]}$ relative to $\{\mathbf{V}_i^{\mathbf{\Sigma}}\}_{i=1}^{k_{j_b}^{\mathbf{\Sigma}}}$; in Line 20, $\mathbf{B}$ stores the first diagonal block of $\mathbf{\Sigma}_{[j_b]}$ in the dense format; in Line 25, the $(i_b - j_b)$th column of $\mathbf{B}$ stores the coordinates of $\mathbf{L}_{i_b,j_b}$ relative to $\mathbf{S}_{j_b}$, which is transformed to coordinates relative to $\{\mathbf{V}_i^{\mathbf{L}}\}_{i=1}^{k_{j_b}^{\mathbf{L}}}$ after Line 27. The updates in Lines 13, 22, and 28 are based on the definitions in Eqs. (3), (4), (5), and (6), involving dense matrix multiplications. Projecting the diagonal blocks of $\mathbf{L}$ in Line 34 after the main loop that spans from Line 5 to Line 32 guarantees that $\{\text{vec}(\mathbf{V}_i^{\mathbf{L}})\}_{i=1}^{k_{j_b}^{\mathbf{L}}}$ falls in the column space of $\mathbf{S}_{j_b}$ until the end of the main loop.

The overloaded subroutine PROJ used in Algorithm 3 for projection and discovery of new basis blocks is described in Algorithms 4 and 5. Algorithm 4 accepts a set of coordinates from the columns of $\mathbf{A}$ to project onto the existing basis set $\mathbf{C}$, while both coordinates are relative to the orthogonal columns of $\mathbf{Q}$. When a new basis is discovered, it is normalized and appended to $\mathbf{C}$ and the inputs $\{\mathbf{U}_i\}_{i=1}^k$, $\{\mathbf{V}_i\}_{i=1}^k$, and k are updated. Algorithm 5 accepts the jth diagonal block $\mathbf{A}$ in the dense format to project onto existing basis blocks $\{\mathbf{V}_i\}_{i=1}^k$, whose projection coordinates are stored in $\{\mathbf{U}_i\}_{i=1}^k$ and rejection is used for discovering new bases. The modified Gram–Schmidt subroutine used in the two PROJ functions projects the first argument onto the columns of the second, returning the projection coordinates in $\mathbf{u}$ and the rejection in $\mathbf{v}$, whose L_2-norm is compared with the tolerance level ϵ to decide if a new basis will be added. In Algorithm 4, the random column sampling right-multiplies $\mathbf{A}$ by a Gaussian matrix with $(k^{\mathbf{A}} + 10)$ columns, where $k^{\mathbf{A}}$ is the estimated numerical column rank of $\mathbf{A}$ and an extra sample size of 10 is typically sufficient (Martinsson, 2011). We set upper limits for $k_{j_b}^{\mathbf{\Sigma}}$, $k_{j_b}^{\mathbf{L}}$, and $k_{j_b}^{\mathbf{D}}$ to control the number of basis blocks, which can be used in place of $k^{\mathbf{A}}$.

The complexity of Algorithm 3 is broken down into categories as shown in Table 2, whose values are aggregated across the outer iterations starting at Line 5. Here, $k_{\max}$ is defined as $\max(k_{n_1}^{\mathbf{\Sigma}}, k_{n_1}^{\mathbf{L}}, k_{n_1}^{\mathbf{D}})$. Assuming $k_{\max} \leq m_1^{1/2} \leq m_2^{1/2}$, the total complexity of Cholesky factorization is dominated by dense Cholesky factorization and triangular matrix inversion on the diagonal blocks, which reach the optimal complexity of $O(n^2)$ when $m_1 = m_2 = n_1 = n_2 = \sqrt{n}$. This optimal complexity

Table 2
Major complexity components for Algorithm 3. 'Projection' means computing the coordinates and discovering new bases. 'Diagonal' refers to performing dense Cholesky factorization and triangular matrix inverse on the diagonal blocks.

	Projection	Schur	**S** and **H**	Diagonal
Line	11, 20, 26, 33	9, 17	12, 21, 27	19
Complexity	$O(k_{\max}^3(m_1^2+m_2^2))$	$O(k_{\max}^2 m_1^3)$	$O(k_{\max}^2 m_2^3 + k_{\max}^4 m_2^2)$	$O(m_1 m_2^3)$

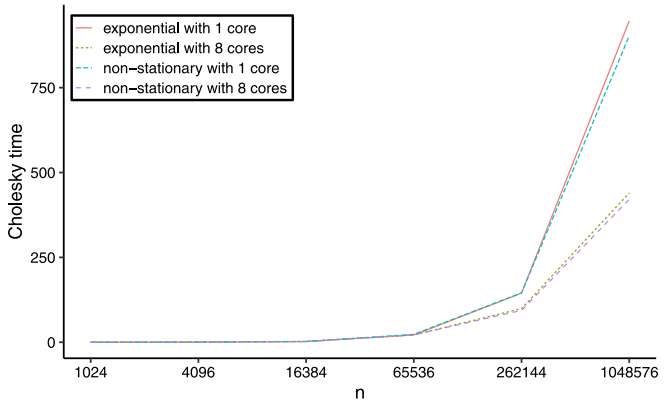


Fig. 5. Time costs (seconds) for Cholesky factorization under the SKP representation. The exponential and the non-stationary correlation functions from Eqs. (1) and (2) are used for building the correlation matrix. The same algorithm is run over 8 threads to show the parallel capacity.

is naturally achieved for square grids while for rectangular grids, it can be asymptotically achieved by choosing (m_2, n_2) close to but no smaller than $\sqrt{n}$. The value of $k_{\max}$ is normally below 20 for n in hundreds of thousands of dimensions, which satisfies $k_{\max} \leq \min(m_1^{1/2}, m_2^{1/2})$. The outer loop from Line 5 in Algorithm 3 has limited parallel capacity but the matrix multiplications for constructing $\mathbf{S}_{j_b}$ and $\mathbf{H}_{j_b}$ can be readily parallelized.

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 hierarchically semiseparable (HSS) matrices through Schur compensation, which truncates the orthogonal factorization of the off-diagonal block. However, we find this technique difficult to be extended to the SKP representation. As $\mathbf{L}_{j_b,1:j_b-1}\mathbf{L}_{j_b,1:j_b-1}^\top = \sum_{j=1}^{j_b-1}\mathbf{L}_{j_b,j}\mathbf{L}_{j_b,j}^\top$, the approximation error for the first diagonal block in $\mathbf{\Sigma}_{[j_b]}$ accumulates with the block column index j_b. Unlike Xia and Gu (2010), it is difficult to keep the approximation error, defined as the difference between the approximation and the truth, of $\mathbf{L}_{j_b,1:j_b-1}\mathbf{L}_{j_b,1:j_b-1}^\top$ non-negative definite and under strong correlation, $\mathbf{\Sigma}_{[j_b]}$ is already close to numerical singularity assuming no approximation error, which in turn requires high approximation accuracy that grows with j_b to maintain positive definiteness.

In the construction of the SKP representation, we considered the fixed spatial domain of a unit square where the unit distance becomes smaller when the number of locations n increases, bringing up the accuracy requirement. 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, for which a small approximation error of $\mathbf{L}_{j_b,1:j_b-1}\mathbf{L}_{j_b,1:j_b-1}^\top$ may cause $\mathbf{\Sigma}_{[j_b]}$ to be non-positive definite. In this section, we control the singularity issue by constructing the covariance matrix based on an expanding domain, i.e., fixing the unit distance while increasing n, which is considered easier compared with the fixed spatial domain discussed in the next section. Specifically, we assume a square grid with the unit distance of $1/32$, which also includes rectangular grids in the sense that they can be embedded into square grids and hence, their Cholesky factors can be obtained as a submatrix. Both stationary and non-stationary kernels, namely, Eq. (1) with $\beta_0 = 0.1$ and Eq. (2) with $\beta_i = 0.03 + 0.07 * x_i$, are used to build the SKP representations of the correlation matrices, whose factorization times based on Algorithm 3 are plotted in Fig. 5.

The factorization in one million dimensions takes 450 s with 8 threads while 950 s with a single thread, indicating that more than half of the workload can be parallelized. To compare with other methods, Cholesky factorization in one million dimensions is generally impossible for standard scientific workstations when assuming the dense-matrix format while Litvinenko et al. (2020) used the hierarchical matrix to achieve a two million dimensional factorization in 473 s with 40 cores. However, it is worth noticing that the storage of the two million dimensional hierarchical Cholesky factor requires 19 GB, potentially exceeding the memory capacity of standard workstations and similar to the SKP representation,

Table 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.

	Err, $n = 2^{10}$	Err, $n = 2^{12}$	Err, $n = 2^{14}$	Time, $n = 2^{20}$
Exponential	1.6%	4.1%	3.7%	596 s
Non-stationary	1.3%	1.8%	8.0%	N.A.

the factorization of the hierarchical covariance matrix may fail due to singularity issues when locations are very close. The relative errors of the Cholesky factors computed by Algorithm 3 range from 1.5% to 1.6% for all cases in up to $n = 2^{14}$ dimensions, suggesting an average coefficient-wise error smaller than 1.6e−4 when $n = 2^{14}$. The tolerance level for discovering new basis blocks, ϵ, is set at 1e−2 if $n \leq 2^{16}$ and at 5e−3 otherwise. Additionally, we impose upper limits for $k^{\mathbf{\Sigma}}_{j_b}$, $k^{\mathbf{L}}_{j_b}$, and $k^{\mathbf{D}}_{j_b}$ to control the overall memory footprint and computation cost, which are 20, 20, and 5, respectively, if $n \leq 2^{16}$ and 25, 25, and 6 otherwise.

4.4. Correction mechanism for a fixed domain

Under fixed domains, the singularity of the covariance matrix generally increases with the number of spatial locations n, for which the first diagonal block of $\mathbf{\Sigma}_{[j_b]}$, given by $\mathbf{\Sigma}_{j_b j_b} - \sum_{j=1}^{j_b-1} \mathbf{L}_{j_b,j}\mathbf{L}_{j_b,j}^\top$, may fail the dense Cholesky factorization as shown in Line 20 of Algorithm 3. Denoting the spatial variables whose marginal covariance matrix is the jth diagonal block of $\mathbf{\Sigma}$ with $\mathbf{z}_j$, $\mathbf{\Sigma}_{[j_b]}$ is the conditional covariance matrix of $\mathbf{z}_{j_b:n_1}$ given $\mathbf{z}_{1:j_b-1}$. One idea to reduce singularity is to subtract 'less' from $\mathbf{\Sigma}_{j_b j_b}$, which can be achieved by substituting $\mathbf{\Sigma}_{[j_b]}$ with the conditional covariance matrix of $\mathbf{z}_{j_b:n_1}$ given $\mathbf{z}_{j_b-k_0:j_b-1}$, where k_0 defines the number of conditioning dimensions. Notice that this leads to a different mechanism from subtracting the last k_0 terms in $\sum_{j=1}^{j_b-1} \mathbf{L}_{i_b,j}\mathbf{L}_{j_b,j}^\top$ from $\mathbf{\Sigma}_{i_b j_b}$. To summarize, the correction mechanism we propose is that if the Cholesky factorization in Line 20 of Algorithm 3 fails at the j_0th block column, we substitute $\mathbf{\Sigma}_{[j_b]}$ with the covariance matrix of $(\mathbf{z}_{j_b:n_1} \mid \mathbf{z}_{j_0-k_0:j_b-1})$ for all $j_b \geq j_0$ and the substitution is only applied to the blocks of $\mathbf{\Sigma}_{[j_b]}$ that are involved in the computation.

Another interpretation of this correction mechanism is that we restart the Cholesky factorization with the marginal covariance matrix starting at the $(j_0 - k_0)$th block column/row of $\mathbf{\Sigma}$ until the end but inherit the previously computed $k^{\mathbf{\Sigma}}$, $k^{\mathbf{L}}$, $k^{\mathbf{D}}$, $\mathbf{Q}^{\mathbf{H}}$, $\mathbf{R}^{\mathbf{H}}$, $\mathbf{Q}^{\mathbf{S}}$, $\mathbf{R}^{\mathbf{S}}$, $\mathbf{C}^{\mathbf{\Sigma}}$, and $\mathbf{C}^{\mathbf{L}}$ defined in Algorithm 3. Based on this interpretation, the correction mechanism can be implemented through calling Algorithm 3 again with different inputs. This correction works better for stationary kernels because the linear spaces of the previously computed basis blocks readily contain the newly computed blocks after the correction while for non-stationary kernels, new basis blocks may be needed to maintain the approximation accuracy.

The Cholesky factorization on the fixed domain considers a $\sqrt{n} \times \sqrt{n}$ grid in the unit square and is performed with bigger upper limits for $k^{\mathbf{\Sigma}}_{j_b}$, $k^{\mathbf{L}}_{j_b}$, and $k^{\mathbf{D}}_{j_b}$ as well as a smaller tolerance level for discovering new basis blocks than on the expanding domain. Specifically, the tolerance level ϵ is 1e−2 if $n < 2^{14}$ and 1e−3 otherwise while the upper limits for $k^{\mathbf{\Sigma}}_{j_b}$, $k^{\mathbf{L}}_{j_b}$, and $k^{\mathbf{D}}_{j_b}$ are 20, 20, and 5 if $n < 2^{14}$ and 30, 30, and 10 otherwise. Table 3 describes the relative errors of the computed Cholesky factors up to $n = 2^{14}$ and the time cost of the factorization in one million dimensions with 8 threads. The relative error stabilizes between 3% and 4% for the stationary kernel using the correction mechanism introduced above while the Cholesky factorization fails for the non-stationary kernel when $n \geq 2^{16}$ because the existing basis blocks discovered through the computation at smaller block column indices are not suitable for the factorization at bigger column indices. In most cases, $k^{\mathbf{\Sigma}}_{j_b}$, $k^{\mathbf{L}}_{j_b}$, and $k^{\mathbf{D}}_{j_b}$ already reach their upper limits before the Cholesky factorization finishes and hence, new basis blocks cannot be added for bigger j_b. On the other hand, using larger upper limits of $k^{\mathbf{\Sigma}}_{j_b}$, $k^{\mathbf{L}}_{j_b}$, and $k^{\mathbf{D}}_{j_b}$ may substantially increase the computation cost, exceeding the computation capacity of normal scientific workstations. The Cholesky factorization of covariance matrices based on fixed domains is generally a challenge for low-rank methods, including the tile-low-rank structure (Akbudak et al., 2017) and the hierarchical structure (Hackbusch, 2015). We leave the improvement of the correction mechanism as a direction for future works.

5. Simulation of large Gaussian random fields

One representative application of the Cholesky factor is the simulation of Gaussian random fields, where a number n of spatial variables located at $\{\mathbf{s}_1, \ldots, \mathbf{s}_n\}$ in the domain have a jointly Gaussian distribution defined by the mean parameter $\boldsymbol{\mu} = \{\mu(\mathbf{s}_1), \ldots, \mu(\mathbf{s}_n)\}^\top$ and the covariance matrix $\mathbf{\Sigma}$ with $\mathbf{\Sigma}(i,j) = \mathcal{K}(\mathbf{s}_i, \mathbf{s}_j)$. Here, $\mu(\cdot)$ is a real function and $\mathcal{K}(\cdot,\cdot)$ was defined in Section 2. Gaussian random fields are broadly used in statistical applications, e.g., inference based on datasets generated from climate model ensembles (Castruccio and Genton, 2018), state estimation in linear discrete-time systems (Ackerson and Fu, 1970), and analysis of spanning avalanches (Pérez-Reche and Vives, 2004), where replicates of the estimated model would be of practical use. Therefore, more efficient and less constrained simulation methods are constantly sought after to meet the challenge of a large number of spatial variables n. To our knowledge, the most efficient method currently is the circulant embedding method (Dietrich and Newsam, 1993, 1997) that embeds the simulation

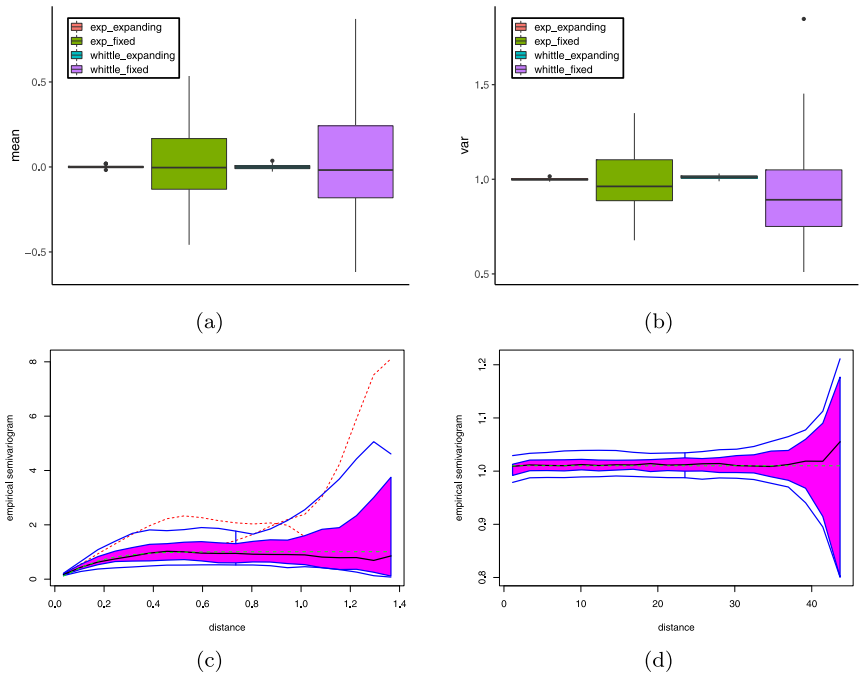


Fig. 6. Mean, variance and empirical semivariogram of spatial variables located on a 1024 × 1024 regular grid from a Gaussian random field. Each plot is based on 100 replicates. The simulation parameters are domain types (fixed domain or expanding domain) and correlation structures (exponential correlation 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.

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 simulations in $\mathbb{R}^p, p \geq 2$, unfortunately, this is frequently not 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 factorization in Section 4, we propose a method to simulate Gaussian random fields on a 2D regular grid that accommodates broader covariance functions than the circulant embedding method and does not depend on the Markov structure. Our method has a complexity of $O(n^2)$ for the Cholesky factorization and $O(n^{3/2})$ for simulating one replicate. Although they are higher than state-of-the-art methods, our method is arguably more flexible and can finish the factorization in one million dimensions in under 600 s on a standard scientific workstation as shown in Table 3, after which the time cost for simulation is trivial. Here, we demonstrate our method by simulating Gaussian random fields on 1024 × 1024 grids that inherit the unit distance of either 1/32 used in the expanding domain of Section 4.3 or 1/1024 used in the fixed domain of Section 4.4. In addition to the exponential kernel, we 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 of 1/1024, for which we add a nugget effect of 1e−2 to reduce singularity.

Fig. 6 shows the statistics of simulated Gaussian random fields, with the same tolerance level ϵ and upper limits for $k_{j_b}^{\mathbf{\Sigma}}$, $k_{j_b}^{\mathbf{L}}$, and $k_{j_b}^{\mathbf{D}}$ as used in the 1 million dimensional factorization of Section 4.4, under which the time cost for simulating one replicate with 8 threads is less than 5 s. The mean estimator has a zero expectation and a variance of $\frac{1}{n^2}\mathbf{1}_n^\top \mathbf{\Sigma} \mathbf{1}_n$, where $\mathbf{1}_n^\top = (1, \ldots, 1)_{1\times n}$, which is not asymptotically zero in the fixed domain. Since the Whittle kernel is bigger than the exponential kernel in $(0, \sqrt{2})$ when both have a range parameter of 0.1, the mean estimator under the former has a larger variance in the fixed domain. The expectation of the variance estimator is $1 - \mathrm{Var}(\bar{z})$, where $\bar{z}$ is the mean estimator and hence, smaller under the Whittle kernel. The empirical semivariogram is computed by randomly sampling 10^8 pairs of spatial variables from each 1 million dimensional replicate to control the overall computation cost. Only those of the Whittle kernel are presented as functional boxplots (Sun and Genton, 2011) because the functional boxplots for

both kernels share strong similarly. Their medians denoted by the solid black curves are well aligned with the expected semivariogram curves denoted by dashed green, supporting the validity of the simulated Gaussian random fields. We conclude that Algorithm 3 provides a scalable option for simulating Gaussian random fields for a wider range of covariance functions. Furthermore, it also provides one direction for other SKP-based matrix operations, facilitating high-dimensional applications in spatial and spatio-temporal statistics.

6. Conclusion and discussion

This paper introduced a novel sum of Kronecker products (SKP) representation for spatial covariance matrices from a 2D 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 variables can be simply along the grid edges without the need for a complex spatial partitioning scheme to guarantee separability, which potentially makes the SKP representation easily adaptable for grids in $\mathbb{R}^p$, $p \geq 3$. We also designed a Cholesky factorization algorithm for the SKP representation to facilitate its practical usage in statistical applications, whose overarching idea is that both the covariance matrix and its Cholesky factor can be closely approximated with the sum of a small number of Kronecker products, e.g., less than 30. The factorization of a one million-dimensional covariance matrix can be achieved within 600 s on a standard scientific workstation, based on which the simulation of large Gaussian random fields under smooth kernels, regarded challenging for other state-of-the-art methods, is used to demonstrate the advantages of the introduced algorithm.

One limitation of the SKP representation is that it depends on the regular grid assumption, without which the approximation error converges slowly to zero. Hence, it may be less general than hierarchical matrices (Hackbusch, 2015) or HSS matrices (Chandrasekaran et al., 2005). Like most low-rank Cholesky factorizations, the Cholesky factorization under the SKP representation can be affected by numerical singularity of the covariance matrix and fail at a certain density level of the spatial variables, for which we proposed a correction mechanism that aimed to fix the factorization at the block column where it became non-positive definite. However, it works mostly for stationary covariance kernels, rendering the robustness and accuracy of the correction mechanism a topic for future study. The Kronecker product is not directly compatible with matrix multiplication, leading to higher complexities 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 paper, but we point out that matrix multiplication within our framework is amenable to massive hybrid distributed-shared memory parallelism.

Acknowledgments

This publication is based upon work supported by the King Abdullah University of Science and Technology (KAUST), Saudi Arabia.

Appendix A

List of symbols containing the variables necessary for the pseudo-algorithm of the Cholesky factorization.

Notations	Descriptions
$\mathbf{s}_1, \ldots, \mathbf{s}_n$	Locations in a spatial domain
$\mathcal{K}(\cdot, \cdot)$	Spatial covariance kernel
$\boldsymbol{\Sigma}$	Spatial covariance matrix such that $\boldsymbol{\Sigma}(i, j) = \mathcal{K}(\mathbf{s}_i, \mathbf{s}_j)$
$\mathbf{L}, \mathbf{D}$	$\mathbf{L}$ is the lower Cholesky factor of $\boldsymbol{\Sigma}$ and $\mathbf{D} = \mathbf{L}^{-1}$
$\boldsymbol{\Sigma}_{i_b j_b}, \mathbf{L}_{i_b j_b}, \mathbf{D}_{i_b j_b}$	The (i_b, j_b)th (m_2, n_2) dimensional block of $\boldsymbol{\Sigma}$, $\mathbf{L}$, and $\mathbf{D}$
m_1, n_1, m_2, n_2	Dimensions of Kronecker factors, $\mathbf{U}_i \in \mathbb{R}^{m_1 \times n_1}$, $\mathbf{V}_i \in \mathbb{R}^{m_2 \times n_2}$
$\{\mathbf{U}_i\}_{i=1}^k$, $\{\mathbf{V}_i\}_{i=1}^k$	Used in general SKP representations, $\sum_{i=1}^k \mathbf{U}_i \otimes \mathbf{V}_i$
$\{\mathbf{U}_i^{\mathbf{M}}\}_{i=1}^{k^{\mathbf{M}}}$, $\{\mathbf{V}_i^{\mathbf{M}}\}_{i=1}^{k^{\mathbf{M}}}$	Used in the specific SKP representation for matrix $\mathbf{M}$
$\tilde{\mathbf{U}}_i, \tilde{\mathbf{V}}_i, \tilde{\boldsymbol{\Sigma}}$	$\tilde{\mathbf{U}}_i = \mathrm{vec}(\mathbf{U}_i)$, $\tilde{\mathbf{V}}_i = \mathrm{vec}(\mathbf{V}_i)$, and $\tilde{\boldsymbol{\Sigma}} \in \mathbb{R}^{m_1 n_1 \times m_2 n_2}$ is rearranged $\boldsymbol{\Sigma}$
$k_{j_b}^{\boldsymbol{\Sigma}}, k_{j_b}^{\mathbf{L}}, k_{j_b}^{\mathbf{D}}$	Numbers of basis blocks when computing the j_bth block column
$\mathcal{S}_{j_b}$	The set of $\{\mathbf{V}_{l_1}^{\boldsymbol{\Sigma}} \mathbf{V}_{l_2}^{\mathbf{D}\top}, l_1 = 1, \ldots, k_{j_b}^{\boldsymbol{\Sigma}}, l_2 = 1, \ldots, k_{j_b}^{\mathbf{D}}\}$
$\mathcal{H}_{j_b}$	The set of $\{\mathbf{V}_{l_1}^{\mathbf{L}} \mathbf{V}_{l_2}^{\mathbf{L}\top}, l_1 = 1, \ldots, k_{j_b}^{\mathbf{L}}, l_2 = 1, \ldots, k_{j_b}^{\mathbf{L}}\}$
$\mathbf{S}_{j_b}, \mathbf{H}_{j_b}$	Matrices corresponding to $\mathcal{S}_{j_b}$ and $\mathcal{H}_{j_b}$

Notations	Descriptions
$\mathbf{Q}_{j_b}^{\mathbf{S}}, \mathbf{Q}_{j_b}^{\mathbf{H}}$	Matrices used as column bases for $\mathbf{S}_{j_b}$ and $\mathbf{H}_{j_b}$, respectively
$\mathbf{R}_{j_b}^{\mathbf{S}}, \mathbf{R}_{j_b}^{\mathbf{H}}$	Coordinate matrices such that $\mathbf{S}_{j_b} = \mathbf{Q}_{j_b}^{\mathbf{S}}\mathbf{R}_{j_b}^{\mathbf{S}}$ and $\mathbf{H}_{j_b} = \mathbf{Q}_{j_b}^{\mathbf{H}}\mathbf{R}_{j_b}^{\mathbf{H}}$
$\mathbf{C}_{j_b}^{\mathbf{\Sigma}}$	Coordinate matrix such that $[\mathrm{vec}(\mathbf{V}_i^{\mathbf{\Sigma}})$ for $i \in 1 : k_{j_b}^{\mathbf{\Sigma}}] = \mathbf{Q}_{j_b}^{\mathbf{H}}\mathbf{C}_{j_b}^{\mathbf{\Sigma}}$
$\mathbf{C}_{j_b}^{\mathbf{L}}$	Coordinate matrix such that $[\mathrm{vec}(\mathbf{V}_i^{\mathbf{L}})$ for $i \in 1 : k_{j_b}^{\mathbf{L}}] = \mathbf{Q}_{j_b}^{\mathbf{S}}\mathbf{C}_{j_b}^{\mathbf{L}}$

Appendix B. Supplementary data

Supplementary material related to this article can be found online at https://doi.org/10.1016/j.csda.2020.107165. The source code for the Cholesky factorization is written in C++ and published in a Github repository: https://github.com/SamCao1991/SKP_CGKT.

References

Ackerson, G., Fu, K., 1970. On state estimation in switching environments. IEEE Trans. Automat. Control 15 (1), 10–17.

Akbudak, K., Ltaief, H., Mikhalev, A., Keyes, D., 2017. Tile low rank Cholesky factorization for climate/weather modeling applications on manycore architectures. In: International Supercomputing Conference. Springer, pp. 22–40.

Bebendorf, M., 2000. Approximation of boundary element matrices. Numer. Math. 86 (4), 565–589.

Björck, Å., 1994. Numerics of gram-Schmidt orthogonalization. Linear Algebra Appl. 197, 297–316.

Börm, S., Grasedyck, L., Hackbusch, W., 2003. Introduction to hierarchical matrices with applications. Eng. Anal. Bound. Elem. 27 (5), 405–422.

Cao, J., Genton, M.G., Keyes, D.E., Turkiyyah, G.M., 2019. Hierarchical-block conditioning approximations for high-dimensional multivariate normal probabilities. Stat. Comput. 29, 585–598.

Castruccio, S., Genton, M.G., 2018. Principles for statistical inference on big spatio-temporal data from climate models. Statist. Probab. Lett. 136, 92–96.

Chan, T.F., 1987. Rank revealing QR factorizations. Linear Algebra Appl. 88, 67–82.

Chandrasekaran, S., Gu, M., Lyons, W., 2005. A fast adaptive solver for hierarchically semiseparable representations. Calcolo 42 (3–4), 171–185.

Dietrich, C., Newsam, G., 1993. A fast and exact method for multidimensional Gaussian stochastic simulations. Water Resour. Res. 29 (8), 2861–2869.

Dietrich, C.R., Newsam, G.N., 1997. Fast and exact simulation of stationary Gaussian processes through circulant embedding of the covariance matrix. SIAM J. Sci. Comput. 18 (4), 1088–1107.

Faloutsos, C., 1986. Multiattribute hashing using gray codes. In: ACM Sigmod Record, Vol. 15. ACM, pp. 227–238.

Faloutsos, C., Roseman, S., 1989. Fractals for secondary key retrieval. In: Proceedings of the Eighth ACM Sigact-Sigmod-Sigarc Symposium on Principles of Database Systems. ACM, pp. 247–252.

Genton, M.G., Keyes, D.E., Turkiyyah, G., 2018. Hierarchical decompositions for the computation of high-dimensional multivariate normal probabilities. J. Comput. Graph. Statist. 27 (2), 268–277.

Gneiting, T., Ševčíková, H., Percival, D.B., Schlather, M., Jiang, Y., 2006. Fast and exact simulation of large Gaussian lattice systems in $\mathbb{R}^2$: Exploring the limits. J. Comput. Graph. Statist. 15 (3), 483–501.

Grasedyck, L., Kressner, D., Tobler, C., 2013. A literature survey of low-rank tensor approximation techniques. GAMM-Mitt. 36 (1), 53–78.

Grasedyck, L., Kriemann, R., Le Borne, S., 2008. Parallel black box $\mathcal{H}$-LU preconditioning for elliptic boundary value problems. Comput. Vis. Sci. 11 (4–6), 273–291.

Greenewald, K., Tsiligkaridis, T., Hero, A.O., 2013. Kronecker sum decompositions of space-time data. In: 2013 5th IEEE International Workshop on Computational Advances in Multi-SEnsor Adaptive Processing (CAMSAP). IEEE, pp. 65–68.

Hackbusch, W., 2015. Hierarchical Matrices: Algorithms and Analysis, Vol. 49. Springer.

Halko, N., Martinsson, P.-G., Tropp, J.A., 2011. Finding structure with randomness: probabilistic algorithms for constructing approximate matrix decompositions. SIAM Rev. 53 (2), 217–288.

Jagadish, H.V., 1990. Linear clustering of objects with multiple attributes. ACM Sigmod Rec. 19 (2), 332–342.

Kriemann, R., 2005. Parallel-matrix arithmetics on shared memory systems. Computing 74 (3), 273–297.

Litvinenko, A., Kriemann, R., Genton, M.G., Sun, Y., Keyes, D.E., 2020. HLIBCov: Parallel hierarchical matrix approximation of large covariance matrices and likelihoods with applications in parameter identification. MethodsX 7, 100600.

Martinsson, P.-G., 2011. A fast randomized algorithm for computing a hierarchically semiseparable representation of a matrix. SIAM J. Matrix Anal. Appl. 32 (4), 1251–1274.

Moon, B., Jagadish, H.V., Faloutsos, C., Saltz, J.H., 2001. Analysis of the clustering properties of the Hilbert space-filling curve. IEEE Trans. Knowl. Data Eng. 13 (1), 124–141.

North, G.R., Wang, J., Genton, M.G., 2011. Correlation models for temperature fields. J. Clim. 24 (22), 5850–5862.

Orenstein, J.A., 1986. Spatial query processing in an object-oriented database system. In: ACM Sigmod Record, Vol. 15. ACM, pp. 326–336.

Paciorek, C.J., Schervish, M.J., 2004. Nonstationary covariance functions for Gaussian process regression. In: Advances in Neural Information Processing Systems. pp. 273–280.

Pérez-Reche, F.J., Vives, E., 2004. Spanning avalanches in the three-dimensional Gaussian random-field ising model with metastable dynamics: field dependence and geometrical properties. Phys. Rev. B 70 (21), 214422.

Rue, H., Held, L., 2005. Gaussian Markov Random Fields: Theory and Applications. Chapman and Hall/CRC.

Sun, Y., Genton, M.G., 2011. Functional boxplots. J. Comput. Graph. Statist. 20 (2), 316–334.

Tsiligkaridis, T., Hero, A.O., 2013. Covariance estimation in high dimensions via kronecker product expansions. IEEE Trans. Signal Process. 61 (21), 5347–5360.

Van Loan, C.F., 2000. The ubiquitous Kronecker product. J. Comput. Appl. Math. 123 (1–2), 85–100.

Van Loan, C.F., Pitsianis, N., 1993. Approximation with Kronecker products. In: Linear Algebra for Large Scale and Real-TIme Applications. Springer, pp. 293–314.

Vecchia, A.V., 1988. Estimation and model identification for continuous spatial processes. J. R. Stat. Soc. Ser. B Stat. Methodol. 50 (2), 297–312.

Xia, J., Gu, M., 2010. Robust approximate cholesky factorization of rank-structured symmetric positive definite matrices. SIAM J. Matrix Anal. Appl. 31 (5), 2899–2920.

Zhao, K., Vouvakis, M.N., Lee, J.-F., 2005. The adaptive cross approximation algorithm for accelerated method of moments computations of EMC problems. IEEE Trans. Electromagn. Compat. 47 (4), 763–773.