# Linear-Cost Covariance Functions for Gaussian Random Fields

Jie Chen* Michael L. Stein†

April 15, 2021

**Abstract**

Gaussian random fields (GRF) are a fundamental stochastic model for spatiotemporal data analysis. An essential ingredient of GRF is the covariance function that characterizes the joint Gaussian distribution of the field. Commonly used covariance functions give rise to fully dense and unstructured covariance matrices, for which required calculations are notoriously expensive to carry out for large data. In this work, we propose a construction of covariance functions that result in matrices with a hierarchical structure. Empowered by matrix algorithms that scale linearly with the matrix dimension, the hierarchical structure is proved to be efficient for a variety of random field computations, including sampling, kriging, and likelihood evaluation. Specifically, with $n$ scattered sites, sampling and likelihood evaluation has an $O(n)$ cost and kriging has an $O(\log n)$ cost after preprocessing, particularly favorable for the kriging of an extremely large number of sites (e.g., predicting on more sites than observed). We demonstrate comprehensive numerical experiments to show the use of the constructed covariance functions and their appealing computation time. Numerical examples on a laptop include simulated data of size up to one million, as well as a climate data product with over two million observations.

## Keywords

Gaussian sampling; Kriging; Maximum likelihood estimation; Hierarchical matrix; Climate data

## 1 Introduction

A Gaussian random field (GRF) $Z(\boldsymbol{x}) : \mathbb{R}^d \to \mathbb{R}$ is a random field where all of its finite-dimensional distributions are Gaussian. Often termed as *Gaussian processes*, GRFs are widely adopted as a practical model in areas ranging from spatial statistics [Stein, 1999], geology [Chilès and Delfiner, 2012], computer experiments [Koehler and Owen, 1996], uncertainty quantification [Smith, 2013], to machine learning [Rasmussen and Williams, 2006]. Among the many reasons for its popularity, a computational advantage is that the Gaussian assumption enables many computations to be done with basic numerical linear algebra.

Although numerical linear algebra [Golub and Van Loan, 1996] is a mature discipline and decades of research efforts result in highly efficient and reliable software libraries (e.g., BLAS [Goto and Geijn, 2008] and LAPACK [Anderson et al., 1999])[1], the computation of GRF models cannot

*MIT-IBM Watson AI Lab, IBM Research. Email: `chenjie@us.ibm.com`

†Rutgers University. Emails: `ms2870@stat.rutgers.edu`

[1] These libraries are the elementary components of commonly used software such as R, Matlab, and python.

overcome a fundamental scalability barrier. For a collection of $n$ scattered sites $\boldsymbol{x}_1$, $\boldsymbol{x}_2$, ..., $\boldsymbol{x}_n$, the computation typically requires $O(n^2)$ storage and $O(n^2)$ to $O(n^3)$ arithmetic operations, which easily hit the capacity of modern computers when $n$ is large. In what follows, we review the basic notation and a few computational components that underlie this challenge.

Denote by $\mu(\boldsymbol{x}) : \mathbb{R}^d \to \mathbb{R}$ the mean function and $k(\boldsymbol{x}, \boldsymbol{x}') : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ the covariance function, which is (strictly) positive definite. Let $X = \{\boldsymbol{x}_i\}_{i=1}^n$ be a set of sampling sites and let $\boldsymbol{z} = [Z(\boldsymbol{x}_1), \ldots, Z(\boldsymbol{x}_n)]^T$ (column vector) be a realization of the random field at $X$. Additionally, denote by $\boldsymbol{\mu}$ the mean vector with elements $\mu_i = \mu(\boldsymbol{x}_i)$ and by $K$ the covariance matrix with elements $K_{ij} = k(\boldsymbol{x}_i, \boldsymbol{x}_j)$.

**Sampling** Realizing a GRF amounts to sampling the multivariate normal distribution $\mathcal{N}(\boldsymbol{\mu}, K)$. To this end, one performs a matrix factorization $K = GG^T$ (e.g., Cholesky), samples a vector $\boldsymbol{y}$ from the standard normal, and computes

$$\boldsymbol{z} = \boldsymbol{\mu} + G\boldsymbol{y}. \tag{1}$$

**Kriging** Kriging is the estimation of $Z(\boldsymbol{x}_0)$ at a new site $\boldsymbol{x}_0$. Other terminology includes *interpolation*, *regression*[2], and *prediction.* The random variable $Z(\boldsymbol{x}_0)$ conditioned on the observation $\boldsymbol{z}$ admits a normal distribution $\mathcal{N}(\mu_0, \sigma_0^2)$ with

$$\mu_0 = \mu(\boldsymbol{x}_0) + \boldsymbol{k}_0^T K^{-1}(\boldsymbol{z} - \boldsymbol{\mu}) \quad \text{and} \quad \sigma_0^2 = k(\boldsymbol{x}_0, \boldsymbol{x}_0) - \boldsymbol{k}_0^T K^{-1} \boldsymbol{k}_0, \tag{2}$$

where $\boldsymbol{k}_0$ is the column vector $[k(\boldsymbol{x}_1, \boldsymbol{x}_0), k(\boldsymbol{x}_2, \boldsymbol{x}_0), \ldots, k(\boldsymbol{x}_n, \boldsymbol{x}_0)]^T$.

**Log-likelihood** The log-likelihood function of a Gaussian distribution $\mathcal{N}(\boldsymbol{\mu}, K)$ is

$$\mathcal{L} = -\frac{1}{2}(\boldsymbol{z} - \boldsymbol{\mu})^T K^{-1}(\boldsymbol{z} - \boldsymbol{\mu}) - \frac{1}{2}\log\det K - \frac{n}{2}\log 2\pi. \tag{3}$$

The log-likelihood $\mathcal{L}$ is a function of $\boldsymbol{\theta} \in \mathbb{R}^p$ that parameterizes the mean function $\mu$ and the covariance function $k$. The evaluation of $\mathcal{L}$ is an essential ingredient in maximum likelihood estimation and Bayesian inference.

A common characteristic of these examples is the expensive numerical linear algebra computations: Cholesky-like factorization in (1), linear system solutions in (2) and (3), and determinant computation in (3). In general, the covariance matrix $K$ is dense and thus these computations have $O(n^2)$ memory cost and $O(n^3)$ arithmetic cost. Moreover, a subtlety occurs in the kriging of more than a few sites. In dense linear algebra, a preferred approach for solving linear systems is not to form the matrix inverse explicitly; rather, one factorizes the matrix as a product of two triangular matrices with $O(n^3)$ cost, followed by triangular solves whose costs are only $O(n^2)$. Then, if one wants to krige $m = O(n)$ sites, the formulas in (2), particularly the variance calculation, have a total cost of $O(n^2 m) = O(n^3)$. This cost indicates that speeding up matrix factorization alone is insufficient for kriging, because $m$ vectors $\boldsymbol{k}_0$ create another computational bottleneck.

[2] Regression often assumes a noise term that we omit here for simplicity. An alternative way to view the noise term is that the covariance function has a nugget.

## 1.1 Existing Approaches

Scaling up the computations for GRF models has been a topic of great interest in the statistics community for many years and has recently attracted the attention of the numerical linear algebra community. Whereas it is not the focus of this work to extensively survey the literature, we discuss a few representative approaches and their pros and cons.

A general idea for reducing the computations is to restrict oneself to covariance matrices $K$ that have an exploitable structure, e.g., sparse, low-rank, or block-diagonal. Covariance tapering [Furrer et al., 2006, Kaufman et al., 2008, Wang and Loh, 2011, Stein, 2013] approximates a covariance function $k$ by multiplying it with another one $k_{\mathrm{t}}$ that has a compact support. The resulting compactly supported function $kk_{\mathrm{t}}$ potentially introduces sparsity to the matrix. However, often the appropriate support for statistical purposes is not narrow, which undermines the use of sparse linear algebra to speed up computation. Low-rank approximations [Cressie and Johannesson, 2008, Eidsvik et al., 2012] generally approximate $K$ by using a low-rank matrix plus a diagonal matrix. In many applications, such an approximation is quite limited, especially when the diagonal component of $K$ does not dominate the small-scale variation of the random field [Stein, 2008, 2014]. In machine learning under the context of kernel methods, a number of randomized low-rank approximation techniques were proposed (e.g., Nyström approximation [Drineas and Mahoney, 2005] and random Fourier features [Rahimi and Recht, 2007]). In these methods, often the rank may need to be fairly large relative to $n$ for a good approximation, particularly in high dimensions [Huang et al., 2014]. Moreover, not every low-rank approximation can krige $m = O(n)$ sites efficiently. The block-diagonal approximation casts an artificial independence assumption across blocks, which is unappealing, although this simple approach can outperform covariance tapering and low-rank methods in many circumstances [Stein, 2008, 2014].

Additionally, a number of methods have been proposed through exploiting other computationally friendly structures on the Gaussian process. Notable examples include LatticeKrig [Nychka et al., 2015], predictive process [Finley et al., 2009], nearest neighbor Gaussian process [Datta et al., 2016a,b], stochastic PDE [Rue et al., 2009], periodic embedding [Guinness and Fuentes, 2017, Guinness, 2019], Metakriging [Minsker, 2015, Minsker et al., 2017], Gapfill [Gerber et al., 2018], and local approximate Gaussian process [Gramacy and Apley, 2015]. See the case study by Heaton et al. [2019] and references therein for a more complete list of computational methods and empirical comparisons.

There also exists a rich literature focusing on only the parameter estimation of $\boldsymbol{\theta}$. Among them, spectral methods [Whittle, 1954, Guyon, 1982, Dahlhaus and Künsch, 1987] deal with the data in the Fourier domain. These methods work less well for high dimensions [Stein, 1995] or when the data are ungridded [Fuentes, 2007]. Several methods focus on the approximation of the likelihood, wherein the log-determinant term (3) may be approximated by using Taylor expansions [Zhang, 2006] or Hutchinson approximations [Aune et al., 2014, Han et al., 2017, Dong et al., 2017, Ubaru et al., 2017]. The composite-likelihood approach [Vecchia, 1988, Stein et al., 2004, Caragea and Smith, 2007, Varin et al., 2011] partitions $X$ into subsets and expands the likelihood by using the law of successive conditioning. Then, the conditional likelihoods in the product chain are approximated by dropping the conditional dependence on faraway subsets. This approach is often competitive. Yet another approach is to solve unbiased estimating equations [Anitescu et al., 2012, Stein et al., 2013, Anitescu et al., 2017] instead of maximizing the log-likelihood $\mathcal{L}$. This approach rids the computation of the determinant term, but its effectiveness relies on fast matrix-vector multiplications [Chen et al., 2014] and effective preconditioning of the covariance matrix [Stein et al., 2012, Chen, 2013].

Recently, a multi-resolution approach [Katzfuss, 2017] based on successive conditioning was proposed, wherein the covariance structure is approximated in a hierarchical manner. The remainder of the approximation at the coarse level is filled by the finer level. This approach shares quite a few characteristics with our approach, which falls under the umbrella of "hierarchical matrices" in numerical linear algebra. Whereas the structure of Katzfuss [2017] is obtained in a coarse-to-fine fashion, our approach derives the structure in a fine-to-coarse manner, allowing translations to a type of hierarchical matrices that admit $O(n)$ cost without $\log n$ factors. Comparison of kriging and likelihood performance can be found in Section 8.7.

### 1.2 Proposed Approach

In this work, we take a holistic view and propose an approach applicable to the various computational components of GRF. The idea is to construct covariance functions that render a linear storage and arithmetic cost for (at least) the computations occurring in (1) to (3). Specifically, for any (strictly) positive definite function $k(\cdot,\cdot)$, which we call the "base function," we propose a recipe to construct (strictly) positive definite functions $k_{\mathrm{h}}(\cdot,\cdot)$ as alternatives. The base function $k$ is not necessarily stationary. The subscript "h" standards for "hierarchical," because the first step of the construction is a hierarchical partitioning of the computation domain. With the subscript "h", the storage of the corresponding covariance matrix $K_{\mathrm{h}}$, as well as the additional storage requirement incurred in matrix computations, is $O(n)$. Additionally,

1. the arithmetic costs of matrix construction $K_{\mathrm{h}}$, factorization $K_{\mathrm{h}} = G_{\mathrm{h}}G_{\mathrm{h}}^T$, explicit inversion $K_{\mathrm{h}}^{-1}$, and determinant calculation $\det(K_{\mathrm{h}})$ are $O(n)$;

2. for any dense vector $\boldsymbol{y}$ of matching dimension, the arithmetic costs of matrix-vector multiplications $G_{\mathrm{h}}\boldsymbol{y}$ and $K_{\mathrm{h}}^{-1}\boldsymbol{y}$ are $O(n)$; and

3. for any dense vector $\boldsymbol{w}$ of matching dimension, the arithmetic costs of the inner product $\boldsymbol{k}_{\mathrm{h},0}^T\boldsymbol{w}$ and the quadratic form $\boldsymbol{k}_{\mathrm{h},0}^T K_{\mathrm{h}}^{-1}\boldsymbol{k}_{\mathrm{h},0}$ are $O(\log n)$, provided that an $O(n)$ preprocessing is done independently of the new site $\boldsymbol{x}_0$.

The last property indicates that the overall cost of kriging $m = O(n)$ sites and estimating the uncertainties is $O(n \log n)$, which dominates the preprocessing $O(n)$.

The essence of this computationally attractive approach is a special covariance structure that we coin "recursively low-rank." Informally speaking, a matrix $A$ is recursively low-rank if it is a block-diagonal matrix plus a low-rank matrix, with such a structure re-occurring in each main diagonal block of the matrix. The "recursive" part mandates that the low-rank factors share the same subspace across levels. The matrix $K_{\mathrm{h}}$ resulting from the proposed covariance function $k_{\mathrm{h}}$ is a symmetric positive definite version of recursively low-rank matrices. Interesting properties of the recursively low-rank structure of $A$ include that $A^{-1}$ admits exactly the same structure, and that if $A$ is symmetric positive definite, it may be factorized as $GG^T$ where $G$ also admits the same structure, albeit not being symmetric. These are the essential properties that allow for the development of $O(n)$ algorithms throughout. Moreover, the recursively low-rank structure is carried out to the out-of-sample vector $\boldsymbol{k}_{\mathrm{h},0}$, which makes it possible to compute inner products $\boldsymbol{k}_{\mathrm{h},0}^T\boldsymbol{w}$ and quadratic forms $\boldsymbol{k}_{\mathrm{h},0}^T K_{\mathrm{h}}^{-1}\boldsymbol{k}_{\mathrm{h},0}$ in an $O(\log n)$ cost, asymptotically lower than $O(n)$.

This matrix structure is closely connected to the rich literature of fast kernel approximation methods in scientific computing, reflected through a similar hierarchical framework but fine distinctions in design choices. A holistic design that aims at fitting the many computational components of GRF simultaneously however narrows down the possible choices and rationalizes the one that we take. After the presentation of the technical details, we will discuss in depth the subtle distinctions with many related hierarchical matrix approaches in Section 6.

Note that although the proposal is based on approximations, the constructed covariance function $k_{\mathrm{h}}$ is valid for any "rank" and the involved linear algebra algorithms compute exact quantities (under infinite precision). The properties of $k_{\mathrm{h}}$ can be far from those of $k$ owing to the hierarchical nature. In practice, one should fix the rank and let the data size grow, subject to computational budget. Treat $k_{\mathrm{h}}$ as a covariance model by itself and perform model selection, rather than increasing the rank to chase approximation quality.

## 2 Recursively Low-Rank Covariance Function

Let $k : S \times S \to \mathbb{R}$ be positive definite for some domain $S$; that is, for any set of points $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n \in S$ and any set of coefficients $\alpha_1, \ldots, \alpha_n \in \mathbb{R}$, the quadratic form $\sum_{ij} \alpha_i \alpha_j k(\boldsymbol{x}_i, \boldsymbol{x}_j) \geq 0$. We say that $k$ is *strictly* positive definite if the quadratic form is strictly greater than 0 whenever the $\boldsymbol{x}$'s are distinct and not all of the $\alpha_i$'s are 0. Given any $k$ and $S$, in this section we propose a recipe for constructing functions $k_{\mathrm{h}}$ that are (strictly) positive definite if $k$ is so. We note the often confusing terminology that a strictly positive definite function always yields a positive definite covariance matrix for $n$ distinct observations, whereas, for a positive definite function, this matrix is only required to be positive semi-definite.

Some notations are necessary. Let $X$ be an ordered list of points in $S$. We will use $k(X, X)$ to denote the matrix with elements $k(\boldsymbol{x}, \boldsymbol{x}')$ for all pairs $\boldsymbol{x}, \boldsymbol{x}' \in X$. Similarly, we use $k(X, \boldsymbol{x})$ and $k(\boldsymbol{x}, X)$ to denote a column and a row vector, respectively, when one of the arguments passed to $k$ contains a singleton $\{\boldsymbol{x}\}$. These notations apply to any function $k$ (including the constructed $k_{\mathrm{h}}$ and the $\psi^{(i)}$ defined later) and any domain $S$ (including subdomains of it).

The construction of $k_{\mathrm{h}}$ is based on a hierarchical partitioning of $S$. For simplicity, let us first consider a partitioning with only one level. Let $S$ be partitioned into disjoint subdomains $S_1, \ldots, S_t$ such that $S = S_1 \cup \cdots \cup S_t$. Let $\underline{X}$ be a set of $r$ distinct points in $S$. If $k(\underline{X}, \underline{X})$ is invertible, define

$$k_{\mathrm{h}}(\boldsymbol{x}, \boldsymbol{x}') = \begin{cases} k(\boldsymbol{x}, \boldsymbol{x}'), & \text{if } \boldsymbol{x}, \boldsymbol{x}' \in S_j \text{ for some } j, \\ k(\boldsymbol{x}, \underline{X}) k(\underline{X}, \underline{X})^{-1} k(\underline{X}, \boldsymbol{x}'), & \text{otherwise.} \end{cases} \tag{4}$$

In words, (4) states that the covariance for a pair of sites $\boldsymbol{x}, \boldsymbol{x}'$ is equal to $k(\boldsymbol{x}, \boldsymbol{x}')$ if they are located in the same subdomain; otherwise, it is replaced by the Nyström approximation $k(\boldsymbol{x}, \underline{X}) k(\underline{X}, \underline{X})^{-1} k(\underline{X}, \boldsymbol{x}')$. The Nyström approximation is always no greater than $k(\boldsymbol{x}, \boldsymbol{x}')$ and when $k$ is strictly positive definite, it attains $k(\boldsymbol{x}, \boldsymbol{x}')$ only when either $\boldsymbol{x}$ or $\boldsymbol{x}'$ belongs to $\underline{X}$. Following convention, we call the $r$ points in $\underline{X}$ *landmark points.* Throughout this work, we will reserve underscores to indicate a list of landmark points. The term "low-rank" comes from the fact that a matrix generated from Nyström approximation generically has rank $r$ (when $n \geq r$), regardless of how large $n$ is.

The positive definiteness of $k_{\mathrm{h}}$ follows a simple Schur-complement split. Furthermore, we have a stronger result when $k$ is assumed to be strictly positive definite; in this case, $k_{\mathrm{h}}$ carries over the strictness. We summarize this property in the following theorem, whose proof is given in the appendix.

**Theorem 1.** *The function $k_{\mathrm{h}}$ defined in* (4) *is positive definite if $k$ is positive definite and $k(\underline{X}, \underline{X})$ is invertible. Moreover, $k_{\mathrm{h}}$ is strictly positive definite if $k$ is so.*

We now proceed to hierarchical partitioning. Such a partitioning of the domain $S$ may be represented by a partitioning tree $T$. We name the tree nodes by using lower case letters such as $j$ and let the subdomain it corresponds to be $S_j$. The root is always $j = 1$ and hence $S \equiv S_1$. We write $\mathrm{Ch}(j)$ to denote the set of all child nodes of $j$. Equivalently, this means that a (sub)domain $S_j$ is partitioned into disjoint subdomains $S_l$ for all $l \in \mathrm{Ch}(j)$. An example is illustrated in Figure 1, where $S_1 = S_2 \cup S_3 \cup S_4$, $S_2 = S_5 \cup S_6 \cup S_7$, and $S_4 = S_8 \cup S_9$.

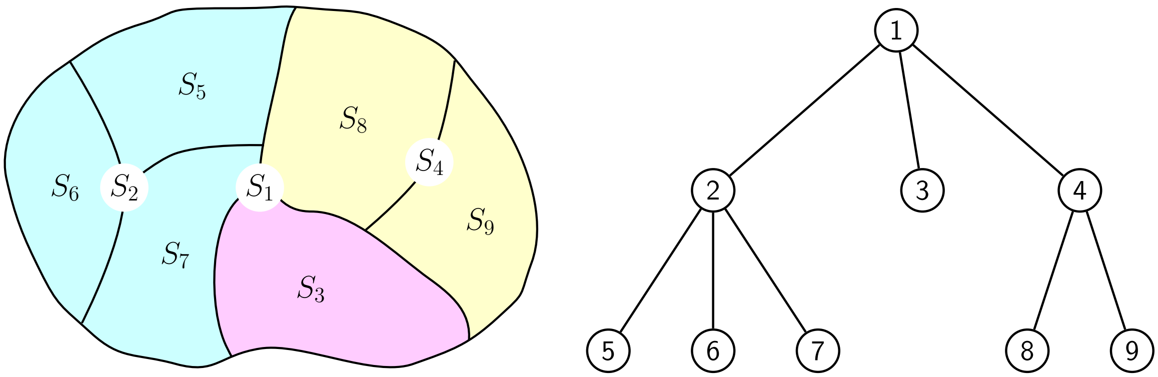


Figure 1: Domain $S$ and partitioning tree $T$.

We now define a covariance function $k_{\mathrm{h}}$ based on hierarchical partitioning. For each nonleaf node $i$, let $\underline{X}_i$ be a set of $r$ landmark points in $S_i$ and assume that $k(\underline{X}_i, \underline{X}_i)$ is invertible. The main idea is to cascade the definition of covariance to those of the child subdomains. Thus, we recursively define a function $k_{\mathrm{h}}^{(i)} : S_i \times S_i \to \mathbb{R}$ such that if $\boldsymbol{x}$ and $\boldsymbol{x}'$ belong to the same child subdomain $S_j$ of $S_i$, then $k_{\mathrm{h}}^{(i)}(\boldsymbol{x}, \boldsymbol{x}') = k_{\mathrm{h}}^{(j)}(\boldsymbol{x}, \boldsymbol{x}')$; otherwise, $k_{\mathrm{h}}^{(i)}(\boldsymbol{x}, \boldsymbol{x}')$ resembles a Nyström approximation. Formally, our covariance function

$$k_{\mathrm{h}} \equiv k_{\mathrm{h}}^{(1)}, \tag{5}$$

where for any tree node $i$,

$$k_{\mathrm{h}}^{(i)}(\boldsymbol{x}, \boldsymbol{x}') = \begin{cases} k(\boldsymbol{x}, \boldsymbol{x}'), & \text{if } i \text{ is leaf}, \\ k_{\mathrm{h}}^{(j)}(\boldsymbol{x}, \boldsymbol{x}'), & \text{if } \boldsymbol{x}, \boldsymbol{x}' \in S_j \text{ for some } j \in \mathrm{Ch}(i), \\ \psi^{(i)}(\boldsymbol{x}, \underline{X}_i) k(\underline{X}_i, \underline{X}_i)^{-1} \psi^{(i)}(\underline{X}_i, \boldsymbol{x}'), & \text{otherwise}. \end{cases} \tag{6}$$

The auxiliary function $\psi^{(i)}(\boldsymbol{x}, \underline{X}_i)$ cannot be the same as $k(\boldsymbol{x}, \underline{X}_i)$, because positive definiteness will be lost. Instead, we make the following recursive definition when $\boldsymbol{x} \in S_i$:

$$\psi^{(i)}(\boldsymbol{x}, \underline{X}_i) = \begin{cases} k(\boldsymbol{x}, \underline{X}_i), & \text{if } \boldsymbol{x} \in S_j \text{ for some } j \in \mathrm{Ch}(i) \text{ and } j \text{ is leaf}, \\ \psi^{(j)}(\boldsymbol{x}, \underline{X}_j) k(\underline{X}_j, \underline{X}_j)^{-1} k(\underline{X}_j, \underline{X}_i), & \text{if } \boldsymbol{x} \in S_j \text{ for some } j \in \mathrm{Ch}(i) \text{ but } j \text{ is not leaf}. \end{cases} \tag{7}$$

To understand the definition, we expand the recursive formulas (5)–(7) for a pair of points $\boldsymbol{x} \in S_j$ and $\boldsymbol{x}' \in S_l$, where $j$ and $l$ are two leaf nodes. If $j = l$, it is trivial that $k_{\mathrm{h}}(\boldsymbol{x},\boldsymbol{x}') = k(\boldsymbol{x},\boldsymbol{x}')$. Otherwise, they have a unique least common ancestor $p$. Then,

$$\begin{aligned} k_{\mathrm{h}}(\boldsymbol{x},\boldsymbol{x}') &= k_{\mathrm{h}}^{(p)}(\boldsymbol{x},\boldsymbol{x}') \\ &= \underbrace{k(\boldsymbol{x},\underline{X}_{j_1})k(\underline{X}_{j_1},\underline{X}_{j_1})^{-1}k(\underline{X}_{j_1},\underline{X}_{j_2})\cdots k(\underline{X}_{j_s},\underline{X}_{j_s})^{-1}k(\underline{X}_{j_s},\underline{X}_{p})}_{\psi^{(p)}(\boldsymbol{x},\underline{X}_p)}\, k(\underline{X}_p,\underline{X}_p)^{-1} \\ &\qquad \cdot \underbrace{k(\underline{X}_p,\underline{X}_{l_t})k(\underline{X}_{l_t},\underline{X}_{l_t})^{-1}\cdots k(\underline{X}_{l_2},\underline{X}_{l_1})k(\underline{X}_{l_1},\underline{X}_{l_1})^{-1}k(\underline{X}_{l_1},\boldsymbol{x}')}_{\psi^{(p)}(\underline{X}_p,\boldsymbol{x}')}, \end{aligned} \tag{8}$$

where $(j, j_1, j_2, \ldots, j_s, p)$ is the path in the tree connecting $j$ and $p$ and similarly $(l, l_1, l_2, \ldots, l_t, p)$ is the path connecting $l$ and $p$. The vectors $\psi^{(p)}(\boldsymbol{x},\underline{X}_p)$ and $\psi^{(p)}(\underline{X}_p,\boldsymbol{x}')$ on the two sides of $k(\underline{X}_p,\underline{X}_p)^{-1}$ come from recursively applying (7).

The definition (5)–(7) admits a covariance decomposition that progressively includes cross-covariances for larger and larger subdomains up the tree. Let us define a function $\xi^{(i)} : S \times S \to \mathbb{R}$ for each node $i$, which has a support on only $S_i \times S_i$; that is, $\xi^{(i)}(\boldsymbol{x},\boldsymbol{x}') = 0$ if either $\boldsymbol{x}$ or $\boldsymbol{x}' \notin S_i$. When both $\boldsymbol{x}$ and $\boldsymbol{x}'$ belong to $S_i$,

$$\xi^{(i)}(\boldsymbol{x},\boldsymbol{x}') = \begin{cases} k(\boldsymbol{x},\boldsymbol{x}') - k(\boldsymbol{x},\underline{X}_p)k(\underline{X}_p,\underline{X}_p)^{-1}k(\underline{X}_p,\boldsymbol{x}'), & \text{if } i \text{ is leaf,} \\ \psi^{(i)}(\boldsymbol{x},\underline{X}_i)k(\underline{X}_i,\underline{X}_i)^{-1}\Delta k(\underline{X}_i,\underline{X}_i)^{-1}\psi^{(i)}(\underline{X}_i,\boldsymbol{x}'), & \text{if } i \text{ is neither leaf nor root,} \\ \psi^{(i)}(\boldsymbol{x},\underline{X}_i)k(\underline{X}_i,\underline{X}_i)^{-1}\psi^{(i)}(\underline{X}_i,\boldsymbol{x}'), & \text{if } i \text{ is root,} \end{cases} \tag{9}$$

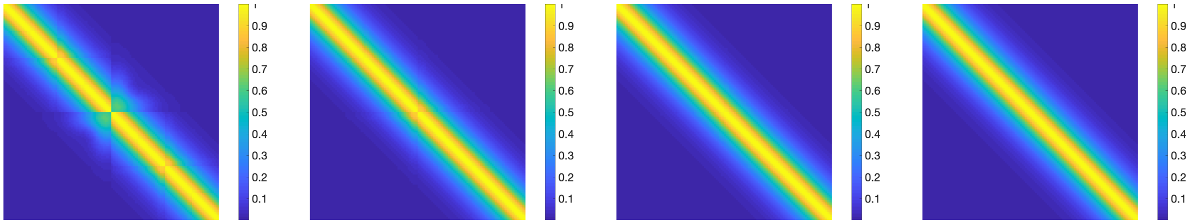
where $\Delta = k(\underline{X}_i,\underline{X}_i) - k(\underline{X}_i,\underline{X}_p)k(\underline{X}_p,\underline{X}_p)^{-1}k(\underline{X}_p,\underline{X}_i)$ and $p$ denotes the parent of $i$. Through telescoping, one sees that $k_{\mathrm{h}}$ is the sum of $\xi^{(i)}$ for all nodes $i$ in the tree: $k(\boldsymbol{x},\boldsymbol{x}') = \sum_{i \in T} \xi^{(i)}(\boldsymbol{x},\boldsymbol{x}')$. Intuitively, at a leaf node $i$, $\xi^{(i)}$ is the covariance of the posterior Gaussian conditioned on the landmark set $\underline{X}_p$. Moving up one level, $\xi^{(p)}$ defines not only the cross-covariance between subdomains of $S_p$, but also modifies the covariance inside each subdomain, say $i$, into $k(\boldsymbol{x},\boldsymbol{x}') - \psi^{(q)}(\boldsymbol{x},\underline{X}_q)k(\underline{X}_q,\underline{X}_q)^{-1}\psi^{(q)}(\underline{X}_q,\boldsymbol{x}')$, where $q$ is the parent of $p$, when $\xi^{(p)}$ is added to $\xi^{(i)}$. Iteratively adding the $\xi$'s from leaf to root, we have all the cross-covariances defined and subsequently modified, as well as the covariance inside each leaf node modified to $k(\boldsymbol{x},\boldsymbol{x}')$.

Similar to Theorem 1, the positive definiteness of $k$ follows from recursive Schur-complement splits across the hierarchy tree. Furthermore, we have that $k_{\mathrm{h}}$ is strictly positive definite if $k$ is so. We summarize the overall result in the following theorem, whose proof is given in the appendix.

**Theorem 2.** *The function $k_{\mathrm{h}}$ defined in* (5)–(7) *is positive definite if $k$ is positive definite and $k(\underline{X}_i,\underline{X}_i)$ is invertible for all nonleaf nodes $i$. Moreover, $k_{\mathrm{h}}$ is strictly positive definite if $k$ is so.*

In Figure 2, we show an example covariance function $k$ on $\mathbb{R}^1 \times \mathbb{R}^1$ together with the constructed $k_{\mathrm{h}}$'s with different number of landmark points, $r$. The base $k$ is the Matérn covariance function (see (11) for definition) with sill 1.0, range 0.2, smoothness 1.5, and nugget 0. The considered domain $S = [0, 1]$ is partitioned into equal halves recursively three times, resulting in eight leaf subdomains. Although $k$ is stationary, $k_{\mathrm{h}}$ is not and thus the visualization does not show a diagonally constant pattern.

The visual cues offered by plot (a) reveal a recursive blocking structure of $k_{\mathrm{h}}$, whereby the main diagonal blocks hold covariances inside the same subdomain and off-diagonal blocks across



(a) $k_{\mathrm{h}}$, $r = 8$ (b) $k_{\mathrm{h}}$, $r = 16$ (c) $k_{\mathrm{h}}$, $r = 32$ (d) $k$

Figure 2: An example Matérn covariance function $k(\cdot, \cdot)$ and the constructed $k_{\mathrm{h}}(\cdot, \cdot)$'s in $[0, 1] \times [0, 1]$.

subdomains. For a pair of points in the same leaf subdomain, their covariance retains the value $k$. When they belong to different subdomains, low-rank approximation takes effects; the higher level in the hierarchy tree, the more aggressive is the approximation (see (8)). Naturally, when $r$ is small, the aggressive approximation renders a noticeable departure from the value $k$, as evident in the off-diagonal blocks toward the center of the plot. As $r$ increases, such a discrepancy is mitigated. When $r = 32$, one sees barely any difference between plots (c) and (d).

It is important to note that the approximation does not depend on the number of sites, $n$. More importantly, $k_{\mathrm{h}}$ is a valid covariance function for any positive integer $r$. Rather than interpreting $k_{\mathrm{h}}$ as an approximation of $k$, one can treat $k_{\mathrm{h}}$ as a new covariance model and select models through comparing likelihoods. Given a fixed $r$, $k_{\mathrm{h}}$ can be applied to arbitrary data size $n$. The appealing $O(n)$ computational cost elucidated subsequently allows for efficient likelihood comparison.

## 3 Recursively Low-Rank Matrix $A$

An advantage of the proposed covariance function $k_{\mathrm{h}}$ is that when the number of landmark points in each subdomain is considered fixed, the covariance matrix $K_{\mathrm{h}} \equiv k_{\mathrm{h}}(X, X)$ for a set $X$ of $n$ points admits computational costs only linear in $n$. Such a desirable scaling comes from the fact that $K_{\mathrm{h}}$ is a special case of *recursively low-rank matrices* whose computational costs are linear in the matrix dimension. In this section, we discuss these matrices and their operations (such as factorization and inversion). Then, in the section that follows, we will show the specialization of $K_{\mathrm{h}}$ and discuss additional vector operations tied to $k_{\mathrm{h}}$.

Let us first introduce some notation. Let $I = \{1, \ldots, n\}$. The index set $I$ may be recursively (permuted and) partitioned, resulting in a hierarchical formation that resembles the second panel of Figure 1. Then, corresponding to a node $i$ is a subset $I_i \subset I$. Moreover, we have $I_i = \cup_{j \in \mathrm{Ch}(i)} I_j$ where the $I_j$'s under union are disjoint. For an $n \times n$ real matrix $A$, we use $A(I_j, I_l)$ to denote a submatrix whose rows correspond to the index set $I_j$ and columns to $I_l$. We also follow the Matlab convention and use : to mean all rows/columns when extracting submatrices. Further, we use $|I|$ to denote the cardinality of an index set $I$. We now define a recursively low-rank matrix.

**Definition 1.** A matrix $A \in \mathbb{R}^{n \times n}$ is said to be *recursively low-rank* with a partitioning tree $T$ and a positive integer $r$ if

1. for every pair of sibling nodes $i$ and $j$ with parent $p$, the block $A(I_i, I_j)$ admits a factorization

$$A(I_i, I_j) = U_i \Sigma_p V_j^T$$

for some $U_i \in \mathbb{R}^{|I_i| \times r}$, $\Sigma_p \in \mathbb{R}^{r \times r}$, and $V_j \in \mathbb{R}^{|I_j| \times r}$; and

2. for every pair of child node $i$ and parent node $p$ not being the root, the factors

$$U_p(I_i,:) = U_i W_p \quad \text{and} \quad V_p(I_i,:) = V_i Z_p$$

   for some $W_p, Z_p \in \mathbb{R}^{r\times r}$.

In Definition 1, the first item states that each off-diagonal block of $A$ is a rank-$r$ matrix. The middle factor $\Sigma_p$ is shared by all children of the same parent $p$, whereas the left factor $U_i$ and the right factor $V_j$ may be obtained through a change of basis from the corresponding factors in the child level, as detailed by the second item of the definition. As a consequence, if $\mathrm{Ch}(i) = \{i_1, \dots, i_s\}$ and $\mathrm{Ch}(j) = \{j_1, \dots, j_t\}$, then

$$A(I_i, I_j) = \underbrace{\begin{bmatrix} U_{i_1} \\ \vdots \\ U_{i_s} \end{bmatrix}}_{U_i} W_i \, \Sigma_p \, \underbrace{Z_j^T \begin{bmatrix} V_{j_1}^T & \cdots & V_{j_t}^T \end{bmatrix}}_{V_j^T}.$$

From now on, we use the shorthand notation $A_{ii}$ to denote a diagonal block $A(I_i, I_i)$ and $A_{ij}$ to denote an off-diagonal block $A(I_i, I_j)$. A pictorial illustration of $A$, which corresponds to the tree in Figure 1, is given in Figure 3. Then, $A$ is completely represented by the factors

$$\{A_{ii}, U_i, V_i, \Sigma_p, W_q, Z_q \mid i \text{ is leaf, } p \text{ is nonleaf, } q \text{ is neither leaf nor root}\}. \tag{10}$$

In computer implementation, we store these factors in the corresponding nodes of the tree. See Figure 4 for an extended example of Figure 1. Clearly, $A$ is symmetric when $A_{ii}$ and $\Sigma_p$ are symmetric, $U_i = V_i$, and $W_q = Z_q$ for all appropriate nodes $i$, $p$, and $q$. In this case, the computer storage can be reduced by approximately a factor of 1/3 through omitting the $V_i$'s and $Z_q$'s; meanwhile, matrix operations with $A$ often have a reduced cost, too.

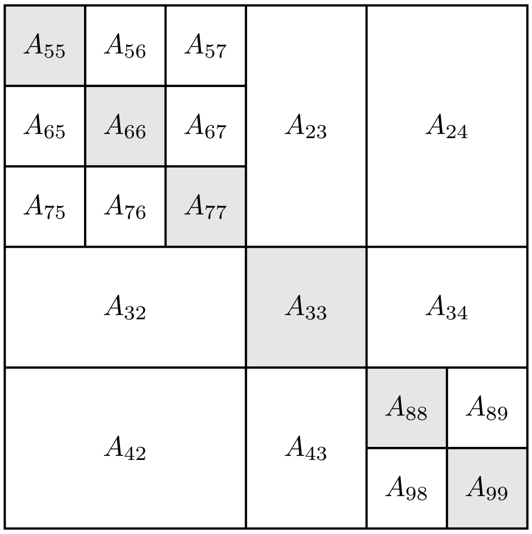


Figure 3: The matrix $A$ corresponding to the partitioning tree in Figure 1.

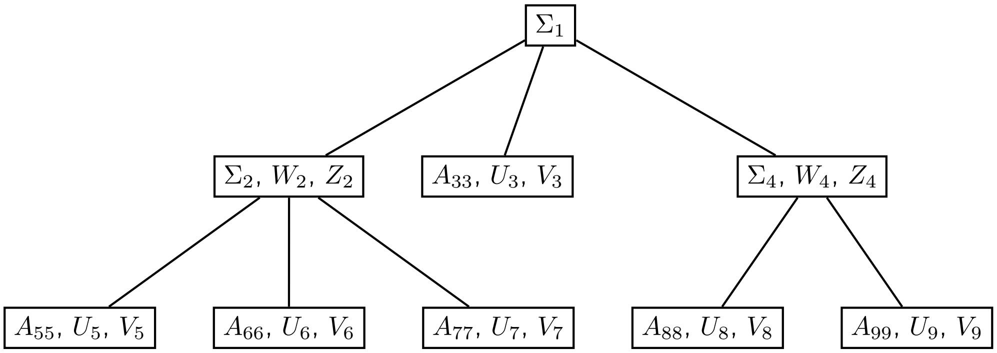


Figure 4: Data structure for storing $A$. The partitioning tree is the same as that in Figure 1.

It is useful to note that not all matrix computations concerned in this paper are done with a symmetric matrix, although the covariance matrix is always so. One instance with unsymmetric matrices is sampling, where the matrix is a Cholesky-like factor of the covariance matrix. Hence, in this section, general algorithms are derived whenever $A$ may be unsymmetric, but we note the simplification for the symmetric case as appropriate.

The four matrix operations under consideration are:

1. matrix-vector multiplication $\boldsymbol{y} = A\boldsymbol{b}$;
2. matrix inversion $\widetilde{A} = A^{-1}$;
3. determinant $\det(A)$; and
4. Cholesky-like factorization $A = GG^T$ (when $A$ is symmetric positive definite).

The detailed algorithms are presented in the appendix. Suffice it to mention here that interestingly, all algorithms are in the form of tree walks (e.g., preorder or postorder traversals) that heavily use the tree data structure illustrated in Figure 4. The inversion and Cholesky-like factorization rely on existence results summarized in the following. The proofs of these theorems are constructive, which simultaneously produce the algorithms. Hence, one may find the proofs inside the algorithms given in the appendix.

**Theorem 3.** *Let $A$ be recursively low-rank with a partitioning tree $T$ and a positive integer $r$. If $A$ is invertible and additionally, $A_{ii} - U_i \Sigma_p V_i^T$ is also invertible for all pairs of nonroot node $i$ and parent $p$, then there exists a recursively low-rank matrix $\widetilde{A}$ with the same partitioning tree $T$ and integer $r$, such that $\widetilde{A} = A^{-1}$. Following* (10)*, we denote the corresponding factors of $\widetilde{A}$ to be*

$$\{\widetilde{A}_{ii}, \widetilde{U}_i, \widetilde{V}_i, \widetilde{\Sigma}_p, \widetilde{W}_q, \widetilde{Z}_q \mid i \text{ is leaf, } p \text{ is nonleaf, } q \text{ is neither leaf nor root}\}.$$

**Theorem 4.** *Let $A$ be recursively low-rank with a partitioning tree $T$ and a positive integer $r$. If $A$ is symmetric, by convention let $A$ be represented by the factors*

$$\{A_{ii}, U_i, U_i, \Sigma_p, W_q, W_q \mid i \text{ is leaf, } p \text{ is nonleaf, } q \text{ is neither leaf nor root}\}.$$

*Furthermore, if $A$ is positive definite and additionally, $A_{ii} - U_i \Sigma_p U_i^T$ is also positive definite for all pairs of nonroot node $i$ and parent $p$, then there exists a recursively low-rank matrix $G$ with the same partitioning tree $T$ and integer $r$, and with factors*

$$\{G_{ii}, U_i, V_i, \Omega_p, W_q, Z_q \mid i \text{ is leaf, } p \text{ is nonleaf, } q \text{ is neither leaf nor root}\},$$

*such that $A = GG^T$.*

## 4 Covariance Matrix $K_\mathrm{h}$ as a Special Case of $A$ and Out-Of-Sample Extension

As noted at the beginning of the preceding section, the covariance matrix $K_\mathrm{h} = k_\mathrm{h}(X, X)$ is a special case of recursively low-rank matrices. This fact may be easily verified through populating the factors of $A$ defined in Definition 1. Specifically, let $X$ be a set of $n$ distinct points in $S$ and let $X_j = X \cap S_j$ for all (sub)domains $S_j$. To avoid degeneracy assume $X_j \neq \emptyset$ for all $j$. Assign a recursively low-rank matrix $A$ in the following manner:

1. for every leaf node $i$, let $A_{ii} = k(X_i, X_i)$;
2. for every nonleaf node $p$, let $\Sigma_p = k(\underline{X}_p, \underline{X}_p)$;
3. for every leaf node $i$, let $U_i = V_i = k(X_i, \underline{X}_p) k(\underline{X}_p, \underline{X}_p)^{-1}$ where $p$ is the parent of $i$; and
4. for every nonleaf node $p$ not being the root, let $W_p = Z_p = k(\underline{X}_p, \underline{X}_q) k(\underline{X}_q, \underline{X}_q)^{-1}$ where $q$ is the parent of $p$.

Then, one sees that $A = K_\mathrm{h}$. Clearly, $A$ is symmetric. Moreover, such a construction ensures that the preconditions of Theorems 3 and 4 be satisfied.

In this section, we consider two operations with the vector $\boldsymbol{v} = k_\mathrm{h}(X, \boldsymbol{x})$, where $\boldsymbol{x} \notin X$ is an out-of-sample (i.e., unobserved site). The quantities of interest are

1. the inner product $\boldsymbol{w}^T \boldsymbol{v}$ for a general length-$n$ vector $\boldsymbol{w}$; and
2. the quadratic form $\boldsymbol{v}^T \widetilde{A} \boldsymbol{v}$, where $\widetilde{A}$ is a *symmetric* recursively low-rank matrix with the same partitioning tree $T$ and integer $r$ as that used for constructing $k_\mathrm{h}$.

For the quadratic form, in practical use $\widetilde{A} = K_\mathrm{h}^{-1}$, but the algorithm we develop here applies to a general symmetric $\widetilde{A}$. The inner product is used to compute prediction (first equation of (2)) whereas the quadratic form is used to estimate standard error (second equation of (2)).

The detailed algorithms are presented in the appendix. Similar to those in the preceding section, they are organized as tree algorithms. The difference is that both algorithms in this section are split into a preprocessing computation independent of $\boldsymbol{x}$ and a separate $\boldsymbol{x}$-dependent computation. The preprocessing still consists of tree traversals that visit all nodes of the hierarchy tree, but the $\boldsymbol{x}$-dependent computation visits only one path that connects the root and the leaf node that $\boldsymbol{x}$ lies in. In all cases, one needs not explicitly construct the vector $\boldsymbol{v}$, which otherwise costs $O(n)$ storage.

# 5 Cost Analysis

All the recipes and algorithms developed in this work apply to a general partitioning of the domain $S$. As is usual, if the tree is arbitrary, cost analysis of many tree-based algorithms is unnecessarily complex. To convey informative results, here we assume that the partitioning tree $T$ is binary and perfect and the associated partitioning of the point set $X$ is balanced. That is, with some positive integer $n_0$, $|X_i| = n_0$ for all leaf nodes $i$. Then, with a partitioning tree of height $h$, the number of points is $|X| = n = n_0 2^h$. We assume that the number of landmark points, $r$, is equal to $n_0$ for simplicity.

Since the factors $A_{ii}$, $U_i$ and $V_i$ are stored in the leaf nodes $i$ and $\Sigma_p$, $W_p$, and $Z_p$ are stored in the nonleaf nodes $p$ (in fact, at the root there is no $W_p$ or $Z_p$), the storage is clearly

$$\underbrace{(2^h)(n_0^2)}_{\text{for } A_{ii}} + \underbrace{2(2^h)(n_0 r)}_{\text{for } U_i \text{ and } V_i} + \underbrace{(2^h - 1)(r^2)}_{\text{for } \Sigma_p} + \underbrace{2(2^h - 2)(r^2)}_{\text{for } W_p \text{ and } Z_p} = O(nr).$$

An alternative way to conclude this result is that the tree has $O(n/r)$ nodes, each of which contains an $O(1)$ number of matrices of size $r \times r$. Therefore, the storage is $O(n/r \times r^2) = O(nr)$. This viewpoint also applies to the additional storage needed when executing all the matrix algorithms, wherein temporary vectors and matrices are allocated. This additional storage is $O(r)$ or $O(r^2)$ per node, hence it does not affect the overall assessment $O(nr)$.

The analysis of the arithmetic cost of each matrix operation is presented in the appendix. In brief summary, matrix construction is $O(n \log n + nr^2)$, matrix-vector multiplication is $O(nr)$, matrix inversion and Cholesky-like factorization are $O(nr^2)$, determinant computation is $O(n/r)$, inner product is $O(r^2 \log_2(n/r))$ with $O(nr)$ preprocessing, and quadratic form is $O(r^2 \log_2(n/r))$ with $O(nr^2)$ preprocessing.

We informally say that the computational cost of the proposed work is $O(n)$, omitting the dependency on $r$. From the function point of view, the quality of $k_{\mathrm{h}}$ is independent of data. It is a valid covariance function for any positive integer $r$. Hence, one may use a fixed $r$ and apply $k_{\mathrm{h}}$ to increasingly more data (e.g., increasingly dense sampling within a fixed domain). It is in this sense that the matrix operations are linear in $n$, although we recognize that for some purposes, one may want to consider allowing $r$ to increase with $n$.

# 6 Connections and Distinctions to Hierarchical Matrices

The proposed recursively low-rank matrix structure builds on a number of previous efforts. For decades, researchers in scientific computing have been keenly developing fast methods for multiplying a dense matrix with a vector, $K\boldsymbol{y}$, where the matrix $K$ is defined based on a kernel function (e.g., Green's function) that resembles a covariance function. Notable methods include the tree code [Barnes and Hut, 1986], the fast multipole method (FMM) [Greengard and Rokhlin, 1987, Sun and Pitsianis, 2001], hierarchical matrices [Hackbusch, 1999, Hackbusch and Börm, 2002, Börm et al., 2003], and various extensions [Gimbutas and Rokhlin, 2002, Ying et al., 2004, Chandrasekaran et al., 2006a, Martinsson and Rokhlin, 2007, Fong and Darve, 2009, Ho and Ying, 2013, Ambikasaran and O'Neil, 2014, March et al., 2015]. These methods were either co-designed, or later generalized, for solving linear systems $K^{-1}\boldsymbol{y}$. They are all based on a hierarchical partitioning of the computation domain, or equivalently, a hierarchical block partitioning of the matrix. The diagonal blocks at the bottom level remain unchanged but (some of) the off-diagonal blocks are low-rank approximated. The differences, however, lie in the fine details, including whether all off-diagonal blocks are low-rank approximated or the ones immediately next to the diagonal blocks should remain unchanged; whether the low-rank factors across levels share bases; and how the low-rank approximations are computed.

The aim of this work is an approach applicable to as many computational components as possible of GRF. Hence, the aforementioned design details necessarily differ from those for other applications. Moreover, certain compromises may need to be made for a broad coverage; for example, a structure optimal for kriging is out of the question if not generalizable to likelihood calculation. The rationale of our design choice is best conveyed through comparing with related methods. Our work distinguishes from them in the following aspects.

**Function versus matrix.** We explicitly define the covariance function on $\mathbb{R}^d \times \mathbb{R}^d$, which is shown to be (strictly) positive definite. Whereas the related methods are all understood as matrix approximations, to the best of our knowledge, none of these works considers the underlying kernel function that corresponds to the approximate matrix. The knowledge of the underlying function is important for out-of-sample extensions, because, for example in kriging (2), one should approximate also the vector $\boldsymbol{k}_0$ in addition to the matrix $K$.

One may argue that if $K$ is well approximated (e.g., accurate to many digits), then it suffices to use the nonapproximate $\boldsymbol{k}_0$ for computation. It is important to note, however, that the matrix approximations are elementwise, which does not guarantee good spectral approximations. As a consequence, numerical error may be disastrously amplified through inversion, especially when there is no or a small nugget effect. Moreover, using the nonapproximate $\boldsymbol{k}_0$ for computation will incur a computational bottleneck if one needs to krige a large number of sites, because constructing the vector $\boldsymbol{k}_0$ alone incurs an $O(n)$ cost.

On the other hand, we start from the covariance function and hence one needs not interpret the proposed approach as an approximation. *All the linear algebra computations are exact in infinite precision, including inversion and factorization.* Additionally, positive definiteness is proved. Few methods under comparison hold such a guarantee.

**Positive definiteness.** A substantial flexibility in the design of methods under comparison is the low-rank approximation of the off-diagonal blocks. If the approximation is algebraic, the common objective is to minimize the approximation error balanced with computational efficiency (otherwise the standard truncated singular value decomposition suffices). Unfortunately, rarely does such a method maintain the positive definiteness of the matrix, which poses difficulty for Cholesky-like factorization and log-determinant computation. A common workaround is some form of compensation, either to the original blocks of the matrix [Bebendorf and Hackbusch, 2007] or to the Schur complements [Xia and Gu, 2010]. Our approach requires no compensation because of the guaranteed positive definiteness.

**Matrix structures and algorithms.** The fine distinctions in matrix structures lead to substantially different algorithms for matrix operations, if even possible. Our structure is almost the same as that of HSS matrices [Chandrasekaran et al., 2006a, Xia et al., 2010] and of $\mathrm{H}^2$ matrices with weak admissiblity [Hackbusch and Börm, 2002], but distant from that of tree code [Barnes and Hut, 1986], FMM [Greengard and Rokhlin, 1987], H matrices [Hackbusch, 1999], and HODLR matrices [Ambikasaran and O'Neil, 2014]. Whereas fast matrix-vector multiplications are a common capability of different matrix structures, the picture starts to diverge for solving linear systems: some structures (e.g., HSS) are amenable for direct factorizations [Chandrasekaran et al., 2006b, Xia and Gu, 2010, Li et al., 2012, Wang et al., 2013], while the others must apply preconditioned iterative methods. An additional complication is that direct factorizations may only be approximate, and thus if the approximation is not sufficiently accurate, it can serve only as a preconditioner but cannot be used in a direct method [Iske et al., 2017]. Then, it will be nearly impossible for these matrix structures to perform Cholesky-like factorizations accurately.

In this regard, our matrix structure is the most clean. Thanks to the property that the matrix inverse and the Cholesky-like factor admit the same structure as that of the original matrix, all the matrix operations considered in this work are exact. Moreover, the explicit covariance function also allows for the development of $O(\log n)$ algorithms for computing inner products and quadratic forms, which, to the best of our knowledge, has not been discussed in the literature for other matrix structures.

**Translation from function to matrix.** In the proposed approach, the factors are defined by exploiting the base covariance function, as opposed to HSS and $\mathrm{H}^2$ approaches where the factors are generally computed through algebraic factorization and approximation. The delicate definition of the factors ensures positive definiteness, which is lacked by the algebraic methods and even by the methods that exploit the base kernel (e.g., Fong and Darve [2009]). The guarantee of positive definiteness necessitates certain sacrifice in approximation accuracy. Thus, the proposed approach is well suited for GRF but for other applications, such as solving partial differential equations, where more specialized methods such as HSS and $\mathrm{H}^2$ are preferred.

**Computational costs.** Although most of the methods under this category enjoy an $O(n)$ or $O(n \log^p n)$ (for some small $p$) arithmetic cost, not every one does so. For example, the cost of skeletonization [Ho and Ying, 2013, Minden et al., 2016] is dimension dependent; in two dimensions it is approximately $O(n^{3/2})$ and in higher dimensions it will be even higher. In general, all these methods are considered matrix approximation methods, and hence there exists a likely tradeoff between approximation accuracy and computational cost. What confounds the approximation is that the low-rank phenomenon exhibited in the off-diagonal blocks fades as the dimension increases [Ambikasaran et al., 2016]. In this regard, it is beneficial to shift the focus from covariance matrices to covariance functions where approximation holds in a more meaningful sense. We conduct experiments to show that predictions and likelihoods are well preserved with the proposed approach.

# 7 Practical Considerations

So far, we have presented a hierarchical framework for constructing valid covariance functions and revealed their appealing computational consequences. The framework is general but there remain instantiations for specific use. In this section, we discuss details tailored to GRF, a low dimensional use case as opposed to the more general (often high-dimensional) case of reproducing kernel Hilbert space.

## 7.1 Partitioning of Domain

For GRF, the sampling sites often reside on a regular grid or a structured (e.g., triangular) mesh. Large spatial datasets with irregular locations commonly occur in remote sensing, although even in this setting, there is usually substantial regularity in the locations due to, for example, the periodicity in a polar-orbiting satellite. When the sites are on a regular grid, a natural choice of the partitioning is axis aligned and balanced. We recommend the following bounding box approach: Begin with the bounding box of the grid, select the longest dimension, cut it into equal halves, and repeat. If the number of grid points along the partitioning dimension in each partitioning is even, the procedure results in a perfect binary tree, whose leaf nodes have exactly the same bounding box volume and the same number of sites. If the number of grid points is odd in some occasion, one shifts the cutting point by half the grid spacing, so that the sampling sites in the middle are not cut.

This bounding box approach straightforwardly generalizes to the mesh or random configuration: Each time the longest dimension of the bounding box is selected and the box is cut into two halves, each of which contains approximately the same number of sampling sites. For random points without exploitable structures, the resulting partitioning tree is known as the k-d tree [Bentley, 1975].

## 7.2 Landmark Points

Assume that the partitioning tree is balanced. As explained in the cost analysis, we consolidate the two parameters, leaf size $n_0$ and the number of landmark points, $r$, into one for convenience. To achieve so, we set the tree height $h$ to be some integer such that the leaf size $n_0 = n/2^h$ is greater than or equal to $r$ but less than $2r$. Even if the partitioning is not balanced, the same effect can still be achieved: the recursive partitioning is terminated when each leaf size is $\geq r$ but $< 2r$.

The appropriate $r$ is case dependent. There exists a tradeoff between approximation accuracy and computational cost. The larger $r$, the closer $k_\text{h}$ is to $k$ but the more expensive is the computation (the cost of matrix-vector multiplication is linear in $r$, whereas those for inversion, Cholesky, inner product, and quadratic forms are all quadratic in $r$). Although there exists analysis (see, e.g., Drineas and Mahoney [2005]) on the approximation error of the covariance matrix under Nyström approximation (which is part of our one-level construction), extending it to the error analysis of kriging or likelihood is challenging, let alone to the analysis under the multilevel setting. For empirical evidence, we show later a computational example of the kriging error and the log-likelihood, as $r$ varies. We suggest that in practice, one sets $r$ through balancing the tolerable error (which may be estimated, for example, by using a hold out set) and the computational resources at hand.

The configuration of the landmark points is flexible. Because of the low dimension, a regular grid is feasible. One may set the number of grid points along each dimension to be approximately proportional to the size of the bounding box. An advantage of using regular grids is that the results are deterministic. An alternative is randomization. The landmark points may either be uniformly random within the bounding box, or uniformly sampled from the sampling sites. A later experiment indicates that the random choice yields a worse approximation on average, but the variance is nonnegligible such that sometimes a better approximation is obtained compared with the regular-grid choice.

# 8 Numerical Experiments

In this section, we show a comprehensive set of experiments to demonstrate the practical use of the proposed covariance function $k_{\mathrm{h}}$ for various GRF computations. These computations are centered around simulated data and data from test functions, based on a simple stationary covariance model $k$. In the next section we will demonstrate an application with real-life data and a more realistic nonstationary covariance model.

The base covariance function $k$ in this section is the Matérn model

$$k(\boldsymbol{x},\boldsymbol{x}') = \frac{10^{\alpha}}{2^{\nu-1}\Gamma(\nu)}\left(\frac{\sqrt{2\nu}\|\boldsymbol{r}\|}{\ell}\right)^{\nu}\mathsf{K}_{\nu}\left(\frac{\sqrt{2\nu}\|\boldsymbol{r}\|}{\ell}\right) + 10^{\tau}\cdot\mathbf{1}(\boldsymbol{r}=\mathbf{0}) \quad \text{with} \quad \boldsymbol{r}=\boldsymbol{x}-\boldsymbol{x}', \tag{11}$$

where $10^{\alpha}$ is the sill, $\ell$ is the range, $\nu$ is the smoothness, and $10^{\tau}$ is the nugget. In each experiment, the vector $\boldsymbol{\theta}$ of parameters include some of them depending on appropriate setting. We have reparameterized the sill and the nugget through a power of ten, because often the plausible search range is rather wide or narrow. Note that for the extremely smooth case (i.e., $\nu=\infty$), (11) becomes equivalently the squared-exponential model

$$k(\boldsymbol{x},\boldsymbol{x}') = 10^{\alpha}\exp\left(-\frac{\|\boldsymbol{r}\|^2}{2\ell^2}\right) + 10^{\tau}\cdot\mathbf{1}(\boldsymbol{r}=\mathbf{0}). \tag{12}$$

We will use this covariance function in one of the experiments. Throughout we assume zero mean for simplicity.

## 8.1 Small-Scale Example

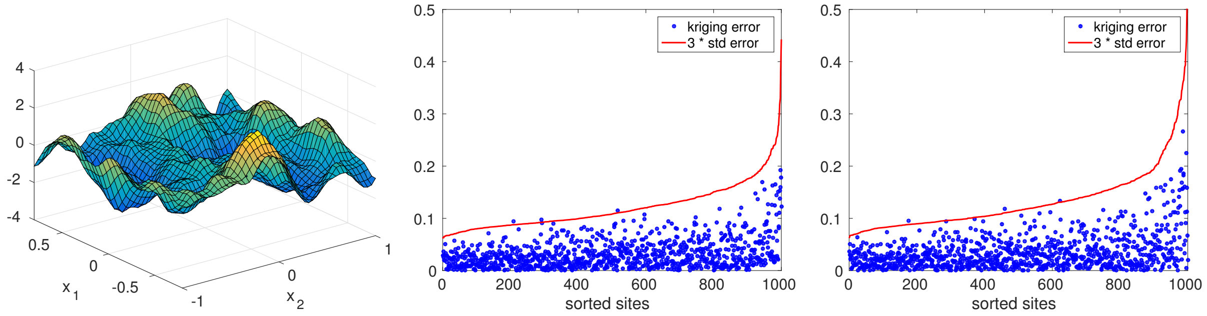
We first conduct a closed-loop experiment whereby data are simulated on a two-dimensional grid from some prescribed parameter vector $\boldsymbol{\theta}$. We discard (uniformly randomly) half the data and perform maximum likelihood estimation. The purpose is to verify that the estimated $\widehat{\boldsymbol{\theta}}$ is indeed close to the $\boldsymbol{\theta}$ that generates the data. Afterward, we perform kriging by using the estimated $\widehat{\boldsymbol{\theta}}$ to recover the discarded data. Because it is a closed-loop setting and there is no model misspecification, the kriging errors should align well with the square root of the variance of the conditional distribution (see (2)). We do not use a large $n$, since we will compare the results of the proposed method with those from the standard method that requires $O(n^3)$ expensive linear algebra computations.

The prescribed parameter vector $\boldsymbol{\theta}$ consists of three elements: $\alpha$, $\ell$, and $\nu$. We choose to use a zero nugget because in some real-life settings, measurements can be quite precise and it is unclear one always needs a nugget effect. This experiment covers such a scenario. Further, note that numerically accurate codes for evaluating the derivatives with respect to $\nu$ are unknown. Such a limitation poses constraints when choosing optimization methods.

Further details are as follows. We simulate data on a grid of size $40\times 50$ occupying a physical domain $[-0.8,0.8]\times[-1,1]$, by using prescribed parameters $\alpha=0$, $\ell=0.2$, and $\nu=2.5$. Half of the data are discarded, which results in $n=1000$ sites for estimation and $m=1000$ sites for kriging.

For the proposed method, we build the partitioning tree by using the bounding box approach elaborated in Section 7. We specify the number of landmark points, $r$, to be 125, and make the height of the partitioning tree $h=\lfloor\log_2(n/r)\rfloor$ such that the number of points in each leaf node is approximately $r$. The landmark points for each subdomain in the hierarchy are placed on a regular grid.

Figure 5(a) illustrates the random field simulated by using $k$. With this data, maximum likelihood estimation is performed, by using separately $k$ and $k_{\mathrm{h}}$. The parameter estimates and their standard errors are given in Table 1. The numbers between the two methods are both quite close to the truth. With the estimated parameters, kriging is performed, with the results shown in Figure 5(b) and (c). The kriging errors are sorted in the increasing order of the prediction variance. The red curves in the plots are three times the square root of the variance; not surprisingly almost all the errors are below this curve.



(a) Simulated random field

(b) Kriging error using $k$

(c) Kriging error using $k_{\mathrm{h}}$

Figure 5: Simulated random field and kriging errors.

Table 1: True parameters and estimates.

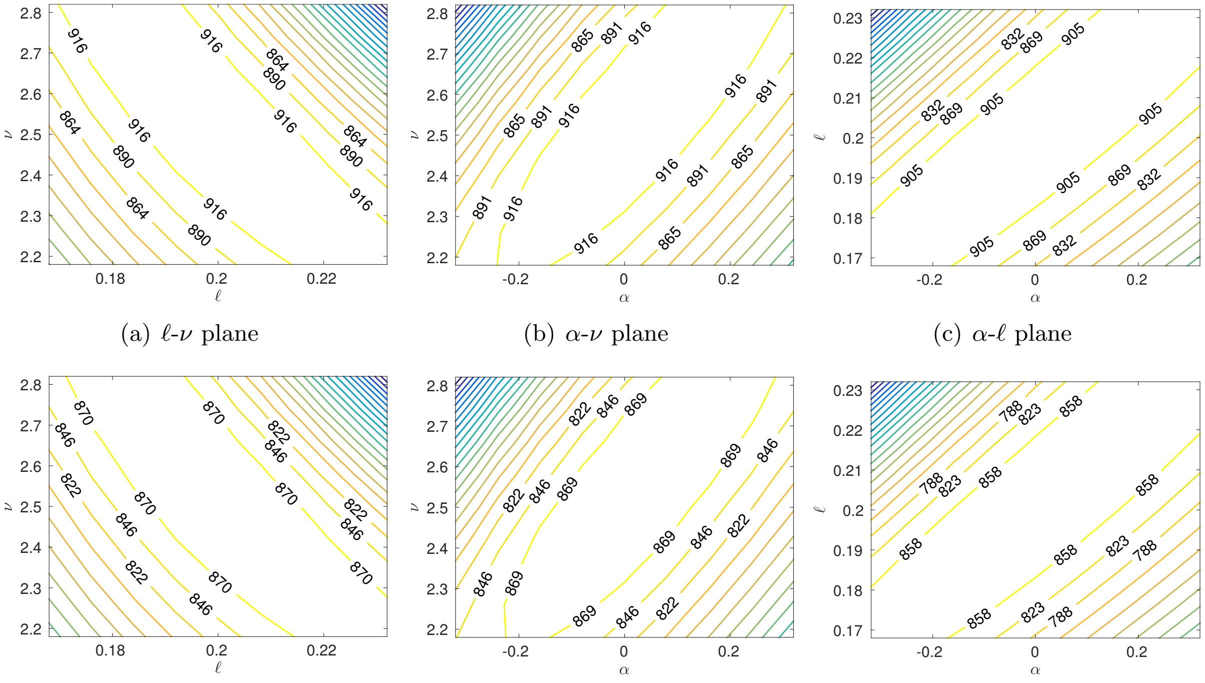
| | $\alpha$ | $\ell$ | $\nu$ |
|---|---|---|---|
| Truth | 0.000 | 0.200 | 2.50 |
| Estimated with $k$ | $-0.172$ (0.076) | 0.182 (0.012) | 2.56 (0.11) |
| Estimated with $k_{\mathrm{h}}$ | $-0.150$ (0.075) | 0.186 (0.012) | 2.53 (0.11) |

### 8.2 Comparison of Log-Likelihoods and Estimates

One should note that the base covariance function $k$ and the proposed $k_{\mathrm{h}}$ are not particularly close, because the number $r$ of landmarks for defining $k_{\mathrm{h}}$ is only 125 (compare this number with the number of observed sites, $n = 1000$). Hence, if one compares the covariance matrix $K$ with $K_{\mathrm{h}}$, they agree in only a limited number of digits. However, the reason why $k_{\mathrm{h}}$ is a good alternative of $k$ is that the shapes of the likelihoods are similar, as well as the locations of the optimum.

We graph in Figure 6 the cross sections of the log-likelihood centered at the truth $\boldsymbol{\theta}$. The top row corresponds to $k$ and the bottom row to $k_{\mathrm{h}}$. One sees that in both cases, the center (truth $\boldsymbol{\theta}$) is located within a reasonably concave neighborhood, whose contours are similar to each other.

In fact, the maxima of the log-likelihoods are rather close. We repeat the simulation ten times and report the statistics in Table 2. The quantities with a subscript "h" correspond to the proposed covariance function $k_{\mathrm{h}}$. One sees that for each parameter, the differences of the estimates are generally about 20% of the standard errors of the estimates. Furthermore, the difference of the true log-likelihoods at the two estimates is always substantially less than one unit. These results indicate that the proposed $k_{\mathrm{h}}$ produces highly comparable parameter estimates with the base covariance function $k$.



(a) $\ell$-$\nu$ plane

(b) $\alpha$-$\nu$ plane

(c) $\alpha$-$\ell$ plane

(d) $\ell$-$\nu$ plane

(e) $\alpha$-$\nu$ plane

(f) $\alpha$-$\ell$ plane

Figure 6: Cross sections of log-likelihood. Top row: base covariance function $k$; bottom row: proposed covariance function $k_{\mathrm{h}}$.

### 8.3 Landmark Points

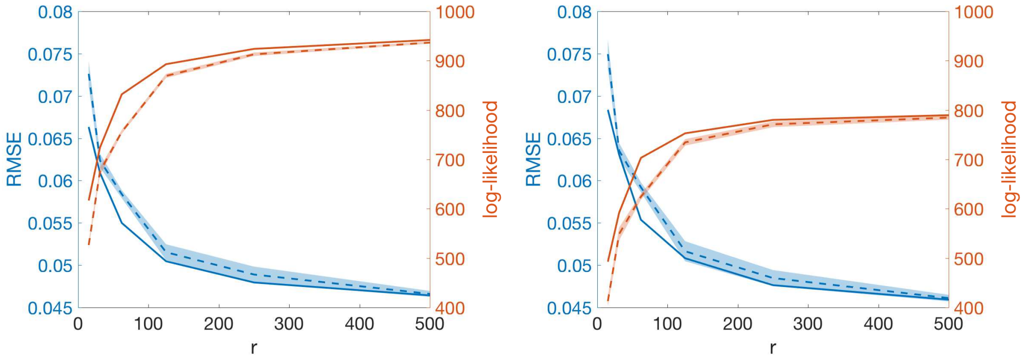
In the preceding two subsections, we fixed the number of landmark points, $r$, to be 125 and placed them on a regular grid within each subdomain. Here, we study the effect of $r$ and the locations.

In Figure 7, we show two plots on the kriging error and the log-likelihood, one obtained by using the ground truth parameters $[\alpha, \ell, \nu] = [0, 0.2, 2.5]$ and the other by using $[\alpha, \ell, \nu] = [0.2, 0.24, 2.7]$, which results in a noticeably different covariance function as judged from the likelihood surface exhibited in Figure 6. The experimented values of $r$ are 7, 15, 31, 62, 125, 250, and 500, geometrically progressing toward the number of observed sites, $n = 1000$. The solid curve corresponds to a regular grid of landmark points, whereas the dashed curve corresponds to the randomized choice, with one times standard deviation shown as a shaded region. "RMSE" denotes root mean squared error.

One sees that the error decreases monotonically as $r$ increases. There thus forms a tradeoff between error and time, since the computational cost is quadratic in $r$. In this particular case, it appears that 125 yields a significant decrease in RMSE while being reasonably small. The likelihood shows a similar trend of change as $r$ varies (except that it increases rather than decreases). Moreover, the randomized choice of landmark points is inferior to the regular-grid choice, considering the mean and standard deviation. However, one should note that if three times standard deviation is considered instead, the shaded region will cover the solid curve for large $r$, indicating that the advantage of regular grid diminishes as $r$ increases. Finally, an interesting observation is that the kriging error remains highly comparable when one uses less accurate covariance parameters, although in this case the reduction of likelihood is substantial.

Table 2: Difference of estimates and log-likelihoods under $k$ and $k_{\mathrm{h}}$. The unparenthesized number is the mean and the number with parenthesis is the standard deviation. For reference, the uncertainties (denoted as stderr) of the estimates are listed in the second part of the table.

| $\lvert\widehat{\alpha}-\widehat{\alpha}_{\mathrm{h}}\rvert$ | $\lvert\widehat{\ell}-\widehat{\ell}_{\mathrm{h}}\rvert$ | $\lvert\widehat{\nu}-\widehat{\nu}_{\mathrm{h}}\rvert$ | $\lvert\mathcal{L}_k(\widehat{\boldsymbol{\theta}})-\mathcal{L}_k(\widehat{\boldsymbol{\theta}}_{\mathrm{h}})\rvert$ |
|---|---|---|---|
| 0.0120 (0.0098) | 0.0018 (0.0018) | 0.0240 (0.0211) | 0.1151 (0.0880) |
| stderr($\widehat{\alpha}$) | stderr($\widehat{\ell}$) | stderr($\widehat{\nu}$) | |
| 0.0841 (0.0050) | 0.0137 (0.0016) | 0.1002 (0.0074) | |



(a) Using ground truth parameters

(b) Using a different set of parameters

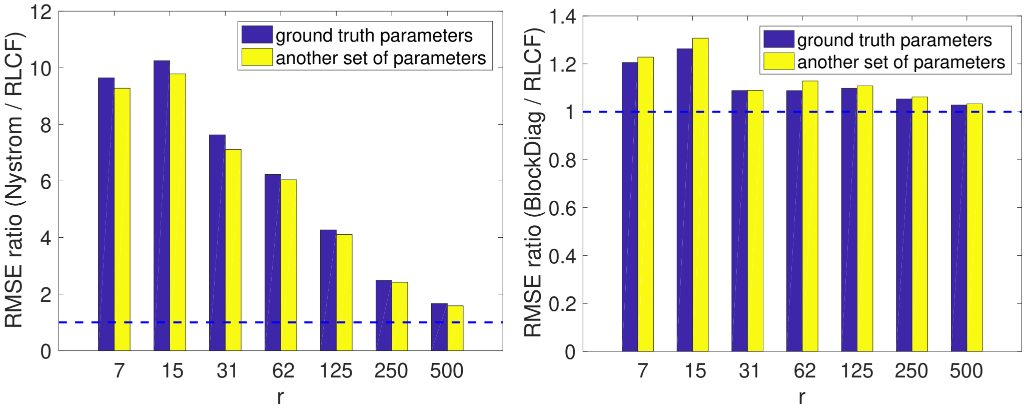
Figure 7: Kriging error and log-likelihood as $r$ varies. The solid curve corresponds to a regular grid configuration of landmark points, whereas the dashed curve with shaded region corresponds to randomized landmark points (repeated 30 times).

### 8.4 Comparison with Nyström and Block-Diagonal Approximation

In this subsection, we compare with two methods: Nyström and block-diagonal approximation. The former is a part of our one-level construction, whereas the latter performs kriging in each fine-level subdomain independently (equivalent to applying a block-diagonal approximation of the covariance matrix $K$). The experiment setting is the same as that of the preceding subsections.

Figure 8(a) shows the kriging error of Nyström normalized by that of the proposed method. First, all error ratios are greater than one, indicating that the hierarchical approach clearly strengthens the approximation with only one level as in Nyström. Moreover, this observation is consistent regardless of what covariance parameters are used. Interestingly, the ratio is slightly smaller when the used parameters are less accurate, suggesting that one-level approximation appears to suffer less when the parameters are not close to the ground truth. Finally, as $r$ increases, the error ratio generally decreases, which is expected since the number of levels that strengthen the approximation becomes fewer. Nyström performs disastrously in light of the fact that the error ratio is greater than 2 when $r < 500$.

Similarly, Figure 8(b) shows the kriging error of block-diagonal approximation, normalized. This method performs much better than Nyström, with the normalized errors only slightly greater than 1. Interestingly, contrary to Nyström, this method suffers more when the parameters are not close to the ground truth. Since the method performs essentially local kriging by ignoring the long-range correlation, this phenomenon is expected.



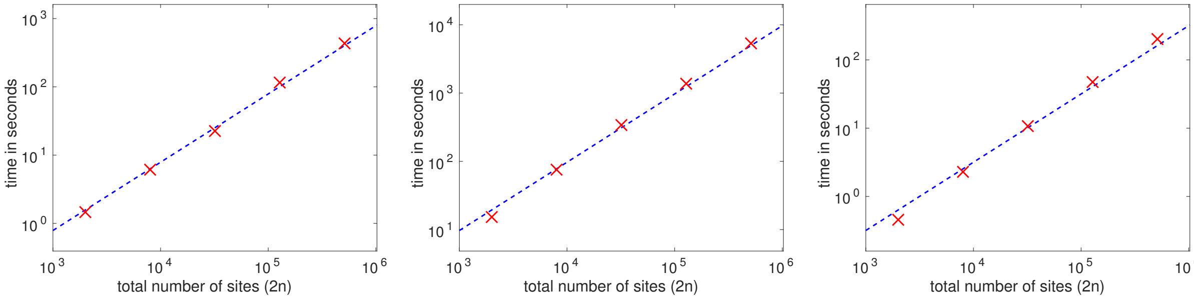
(a) Compared with Nyström (b) Compared with block-diagonal approx.

Figure 8: RMSE ratio between a compared method and the proposed method (RLCF). Ground truth parameters are $[\alpha, \ell, \nu] = [0, 0.2, 2.5]$ and the other set is $[\alpha, \ell, \nu] = [0.2, 0.24, 2.7]$.

## 8.5 Scaling

In this subsection, we verify that the linear algebra costs for the proposed method indeed agree with the theoretical analysis. Namely, random field simulation and log-likelihood evaluation are both $O(n)$, and the kriging of $m = n$ sites is $O(n \log n)$. Note that all these computations require the construction of the covariance matrix, which is $O(n \log n)$.

The experiment setting is the same as that of the preceding subsections, except that we restrict the number of log-likelihood evaluations to 125 to avoid excessive computation. We vary the grid size from $40 \times 50$ to $640 \times 800$ to observe the scaling. The random removal of sites has a minimal effect on the partitioning and hence on the overall time. The computation is carried out on a laptop with eight Intel cores (CPU frequency 2.8GHz) and 32GB memory. Six parallel threads are used.



(a) Random field simulation (b) 125 Log-likelihood evaluations (c) Kriging $n$ sites

Figure 9: Computation time. The dashed blue line is an $O(n)$ scaling.

Figure 9 plots the computation times, which indeed well agree with the theoretical scaling. As expected, log-likelihood evaluations are the most expensive, particularly when many evaluations are needed for optimization. The simulation of a random field follows, with kriging being the least expensive, even when a large number of sites are kriged.

## 8.6 Large-Scale Example Using Test Function

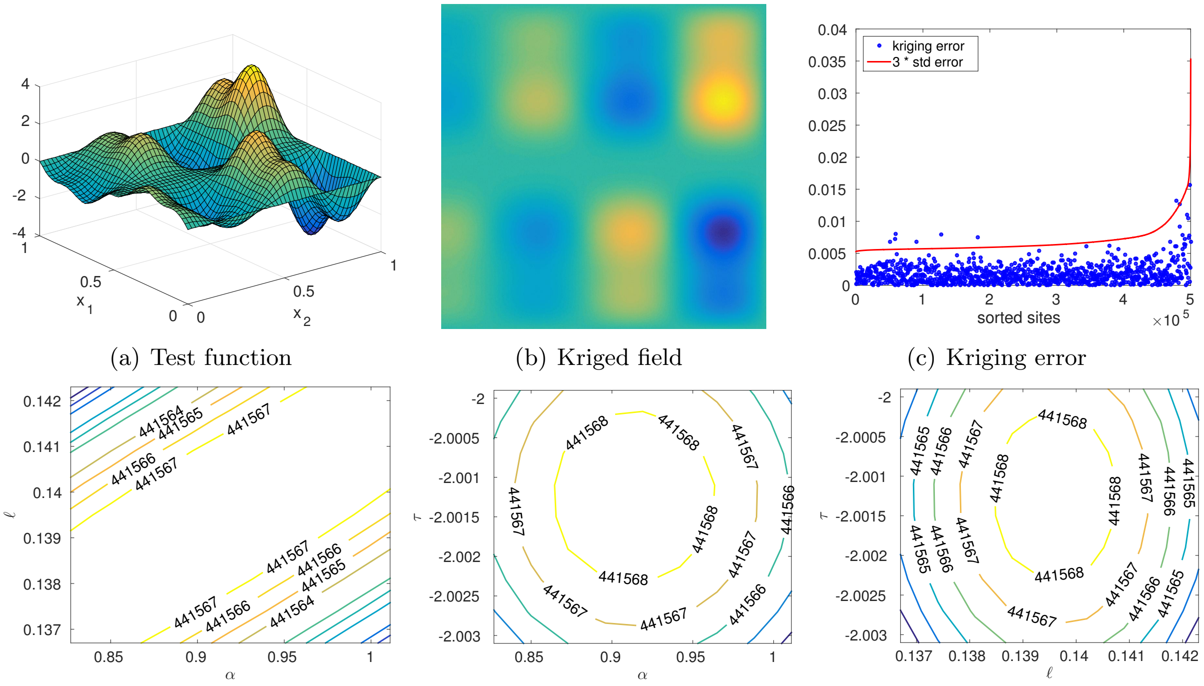
The above scaling results confirm that handling a large $n$ is feasible on a laptop. In this subsection, we perform an experiment with up to one million data sites. Different from the closed-loop setting that uses a known covariance model, here we generate data by using a test function. We estimate the covariance parameters and krige with the estimated model.

The test function is

$$Z(\boldsymbol{x}) = \exp(1.4x_1)\cos(3.5\pi x_1)[\sin(2\pi x_2) + 0.2\sin(8\pi x_2)] \tag{13}$$

on $[0,1]^2$. This function is rather smooth (see Figure 10(a) for an illustration). Hence, we use the squared-exponential model (12) for estimation. The high smoothness results in a too ill-conditioned matrix; therefore, a nugget is necessary. The vector of parameters is $\boldsymbol{\theta} = [\alpha, \ell, \tau]^T$. We inject independent Gaussian noise $\mathcal{N}(0, 0.1^2)$ to the data so that the nugget will not vanish. As before, we randomly select half of the sites for parameter estimation and the other half for kriging. The number of landmark points, $r$, remains 125.

Our strategy for large-scale estimation is to first perform a small-scale experiment with the base covariance function $k$ that quickly locates the optimum. The result serves as a reference for later use of the proposed $k_{\mathrm{h}}$ in the larger-scale setting. The results are shown in Figure 10 (for the largest grid) and Table 3.



(a) Test function

(b) Kriged field

(c) Kriging error

(d) $\alpha$-$\ell$ plane

(e) $\alpha$-$\tau$ plane

(f) $\ell$-$\tau$ plane

Figure 10: Top row: test function and kriging results; bottom row: log-likelihood. For plot (c), the blue dots are subsampled evenly so that they do not clutter the figure.

Each of the cross sections of the log-likelihood on the bottom row of Figure 10 is plotted by setting the unseen parameter at the estimated value. For example, the $\alpha$-$\ell$ plane is located at

Table 3: Estimated parameters.

| Grid | Est. w/ | $\widehat{\alpha}$ | $\widehat{\ell}$ | $\widehat{\tau}$ |
|---|---|---|---|---|
| $50 \times 50$ | $k$ | 0.313 (0.098) | 0.1199 (0.0035) | −2.0109 (0.0186) |
| $100 \times 100$ | $k_{\mathrm{h}}$ | 0.389 (0.095) | 0.1238 (0.0029) | −1.9923 (0.0089) |
| $1000 \times 1000$ | $k_{\mathrm{h}}$ | 0.919 (0.134) | 0.1395 (0.0031) | −2.0011 (0.0009) |

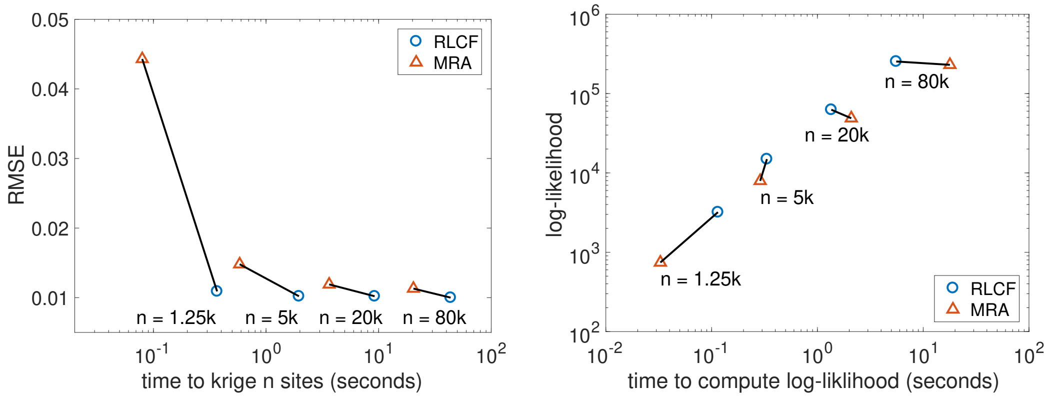
$\widehat{\tau} = -2.0011$. From these contour plots, we see that the estimated parameters are located at a local maximum with nicely concave contours in a neighborhood of this maximum. The estimated nugget ($\approx -2$) well agrees with $\log_{10}$ of the actual noise variance. The kriged field (plot (b)) is visually as smooth as the test function. The kriging errors for predicting the test function $Z(\cdot)$, again sorted by their estimated standard errors, are plotted in (c). As one would expect, nearly all of the errors are less than three times their estimated standard errors. Note that the kriging errors are counted without the perturbed noise; they are substantially lower than the noise level.

## 8.7 Comparison with MRA

We use the test function (13) in the preceding subsection to further compare the proposed method with MRA [Katzfuss, 2017] on kriging and maximum likelihood. Both methods perform a hierarchical decomposition. Our method defines the covariance structure in a bottom-up manner across the partitioning tree and translates it to a recursive low-rank compressed matrix that admits $O(n)$ complexity, while MRA decomposes the random field in a top-down fashion along the tree and yields $O(n \log^p n)$ computational costs for certain $p$'s, suppressing dependency on $r$. The terminology *knots* in MRA plays a similar role as *landmark points* in our method, but the resulting covariance structure is quite different.

We follow almost the same setting as in the preceding subsection, except to inject $\mathcal{N}(0, 0.01^2)$ noise for a smaller RMSE. The MRA code is the C++ implementation suggested by `https://github.com/katzfuss-group/MRA_JASA`, for fair comparison. The computing platform is the same as that in Section 8.5. We experiment with a few grid sizes, for each of which we first optimize the log-likelihood on half of the randomly sampled data to estimate covariance parameters, and then perform kriging on the rest of the data. In both methods, we fix $r = 125$ and let the tree height be $h = \lfloor \log_2(n/r) \rfloor$.

Figure 11 plots RMSE/log-likelihood versus time. A few observations follow. First, with the same hierarchical partitioning and $r$, our method yields lower RMSE (nearly the noise level) and higher log-likelihood. More appealingly, when $n$ grows, the absolute log-likelihood difference increases. Second, both methods obey the $O(n)$ trend (ignoring the logarithmic factor), because the time approximately follows an arithmetic progression under the logarithmic scale. Third, our method calculates log-likelihood faster at large $n$, whereas slower in other cases compared with MRA. For kriging, the consistently slower time is probably caused by a larger constant factor in the big-O complexity. For log-likelihood, one observes that the spacing in elapsed time is different across the two methods. MRA has a bigger spacing, due to an additional $r$ factor in the big-O complexity.



| $n$ | 1.25k | 5k | 20k | 80k |
|---|---|---|---|---|
| $\text{RMSE}_{\text{MRA}} - \text{RMSE}_{\text{RLCF}}$ | 0.0334 | 0.0046 | 0.0018 | 0.0013 |
| $(\text{RMSE}_{\text{MRA}} - 0.01)\ /\ (\text{RMSE}_{\text{RLCF}} - 0.01)$ | 37.04 | 19.40 | 11.37 | 108.21 |
| $\text{log-likelihood}_{\text{RLCF}} - \text{log-likelihood}_{\text{MRA}}$ | 2455 | 6983 | 13703 | 23478 |
| $\text{log-likelihood}_{\text{RLCF}}\ /\ \text{log-likelihood}_{\text{MRA}}$ | 4.3046 | 1.8800 | 1.2804 | 1.1020 |

Figure 11: Comparison with MRA. The quantity $n$ is both the number of observations and the number of kriging sites. For each $n$, the hierarchical partitioning yields the same tree and we use the same $r$. Covariance parameters for each case are individually estimated.

# 9 Analysis of Climate Data

In this section, we apply the proposed method to analyze a climate data product developed by the National Centers for Environmental Prediction (NCEP) of the National Oceanic and Atmospheric Administration (NOAA).[3] The Climate Forecast System Reanalysis (CFSR) data product [Saha et al., 2010] offers hourly time series as well as monthly means data with a resolution down to one-half of a degree (approximately 56 km) around the Earth, over a period of 32 years from 1979 to 2011. For illustration purpose, we extract the temperature variable at 500 mb height from the monthly means data and show a snapshot on the top of Figure 12. Temperatures at this pressure (generally around a height of 5 km) provide a good summary of large-scale weather patterns and should be more nearly stationary than surface temperatures. We will estimate a covariance model for every July over the 32-year period.

Through preliminary investigations, we find that the data appears fairly Gaussian after a subtraction of pixelwise mean across time. An illustration of the demeaned data for the same snapshot is given at the bottom of Figure 12. Moreover, the correlation between the different snapshots are so weak that we shall treat them as independent anomalies. Although temperatures have warmed during this period, the warming is modest compared to the interannual variation in temperatures at this spatial resolution, so we assume the anomalies have mean 0. We use $\boldsymbol{z}_i$ to denote the anomaly at time $i$. Then, the log-likelihood with $N = 32$ zero-mean independent anomalies $\boldsymbol{z}_i$ is

$$\mathcal{L} = -\sum_{i=1}^{N} \frac{1}{2} \boldsymbol{z}_i^T K^{-1} \boldsymbol{z}_i - \frac{N}{2} \log\det K - \frac{Nn}{2} \log 2\pi.$$

[3] `https://www.ncdc.noaa.gov/data-access/model-data/model-datasets/climate-forecast-system-version2-cfsv2`

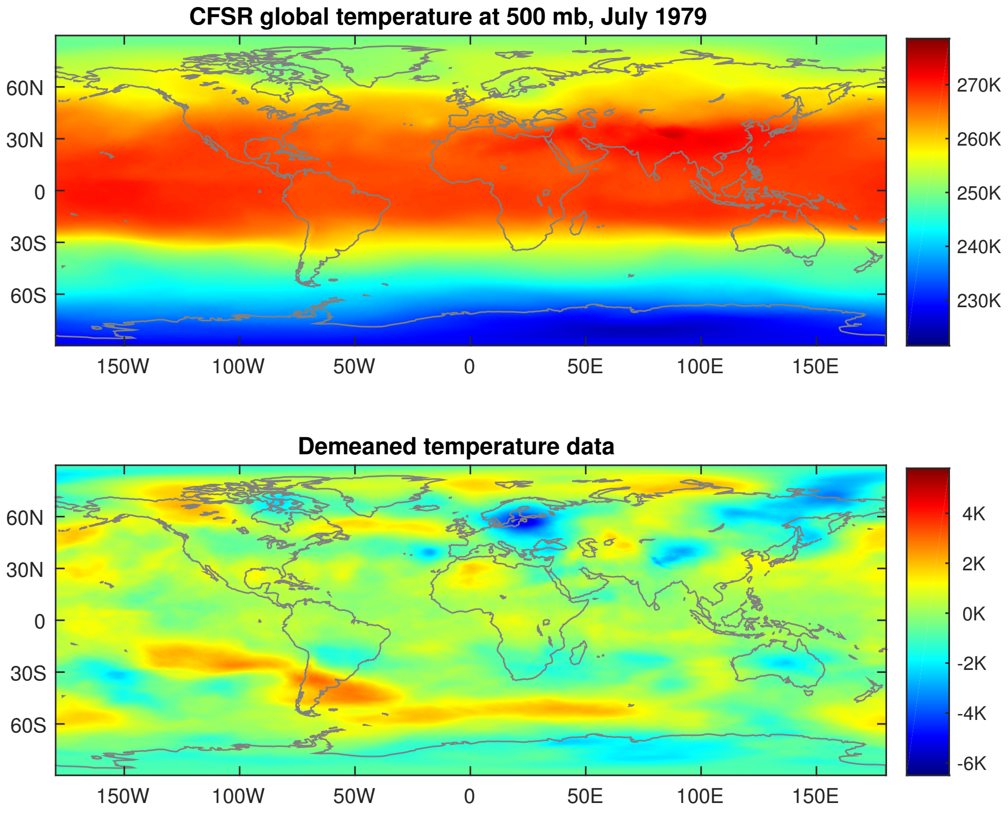


Figure 12: Snapshot of CFSR global temperature at 500 mb and the resulting data after subtraction of pixelwise mean for the same month over 32 years.

For random fields on a sphere, a reasonable covariance function for a pair of sites $\boldsymbol{x}$ and $\boldsymbol{x}'$ may be based on their great-circle distance, or equivalently the chordal distance, because of their monotone relationship. Specifically, let a site $\boldsymbol{x}$ be represented by latitude $\phi$ and longitude $\psi$. Then, the chordal distance between two sites $\boldsymbol{x}$ and $\boldsymbol{x}'$ is

$$r = 2\left[\sin^2\left(\frac{\phi-\phi'}{2}\right)+\cos\phi\cos\phi'\sin^2\left(\frac{\psi-\psi'}{2}\right)\right]^{1/2}. \tag{14}$$

Here, we assume that the radius of the sphere is 1 for simplicity, because it can always be absorbed into a range parameter later. We still use the Matérn model

$$k(\boldsymbol{x},\boldsymbol{x}') = \frac{10^\alpha}{2^{\nu-1}\Gamma(\nu)}\left(\frac{\sqrt{2\nu}r}{\ell}\right)^\nu \mathsf{K}_\nu\left(\frac{\sqrt{2\nu}r}{\ell}\right) + 10^\tau\cdot\mathbf{1}(r=0) \tag{15}$$

to define the covariance function, where $r$ is the chordal distance (14), so that the model is isotropic on the sphere. More sophisticated models based on the same Matérn function and the chordal distance $r$ are proposed in [Jun and Stein, 2008]. Note that this model depends on the longitudes for $\boldsymbol{x}$ and $\boldsymbol{x}'$ only through their differences modulo $2\pi$. Such a model is called *axially symmetric* [Jones, 1963].

A computational benefit of an axially symmetric model and gridded observations is that one may afford computations with $k$ even when the latitude-longitude grid is dense. The reason is that for any two fixed latitudes, the cross-covariance matrix between the observations is circulant and diagonalizing it requires only one discrete Fourier transform (DFT), which is efficient. Thus, diagonalizing the whole covariance matrix amounts to diagonalizing only the blocks with respect to each longitude, apart from the DFT's for each latitude.

Hence, we will perform computations with both the base covariance function $k$ and the proposed function $k_{\mathrm{h}}$ and compare the results. We subsample the grid with every other latitude and longitude for parameter estimation. We also remove the two grid lines 90N and 90S due to their degeneracy at the pole. Because of the half-degree resolution, this results in a coarse grid of size $180 \times 360$ for parameter estimation, for a total of $180 \times 360 \times 32 = 2{,}073{,}600$ observations. The rest of the grid points are used for kriging. As before, we set the number $r$ of landmark points to be 125.

Table 4: Optimization results for different $\nu$'s using the base covariance function $k$.

| $\nu$ | Initial guess $\boldsymbol{\theta}_0$ | Terminate at $\widehat{\boldsymbol{\theta}}$ | Log-likelihood |
|---|---|---|---|
| 0.5 | $(-0.285 \quad 0.156 \quad -4.935)$ | $(-0.794 \quad 1.446 \quad -7.165)$ | $3.938 \times 10^6$ |
| | $(-0.794 \quad 1.446 \quad -7.165)$ | diverge | |
| 1.0 | $(-0.285 \quad 0.156 \quad -4.935)$ | $(-0.279 \quad 0.411 \quad -5.133)$ | $4.696 \times 10^6$ |
| | $(-0.279 \quad 0.411 \quad -5.133)$ | $(\ \ 0.838 \quad 1.494 \quad -5.125)$ | $4.700 \times 10^6$ |
| 1.5 | $(\ \ 0.124 \quad 0.215 \quad -4.933)$ | $(-0.285 \quad 0.156 \quad -4.935)$ | $4.757 \times 10^6$ |
| | $(-0.285 \quad 0.156 \quad -4.935)$ | $(-0.285 \quad 0.156 \quad -4.935)$ | $4.757 \times 10^6$ |
| 2.0 | $(-0.285 \quad 0.156 \quad -4.935)$ | $(-0.279 \quad 0.094 \quad -4.933)$ | $4.643 \times 10^6$ |
| | $(-0.279 \quad 0.094 \quad -4.933)$ | $(-0.545 \quad 0.081 \quad -4.821)$ | $4.653 \times 10^6$ |

We set the parameter vector $\boldsymbol{\theta} = [\alpha, \ell, \tau]^T$, considering only several values for the smoothness parameter $\nu$ because of the difficulties of numerical optimization of the loglikelihood over $\nu$. To our experience, blackbox optimization solvers do not always find accurate optima. We show in Table 4 several results of the Matlab solver `fminunc` when one varies $\nu$. For each $\nu$, we start the solver at some initial guess $\boldsymbol{\theta}_0$ until it claims a local optimum $\widehat{\boldsymbol{\theta}}$. Then, we use this optimum as the initial guess to run the solver again. Ideally, the solver should terminate at $\widehat{\boldsymbol{\theta}}$ if it indeed is an optimum. However, reading Table 4, one finds that this is not always the case.

When $\nu = 0.5$, the second search diverges from the initial $\widehat{\boldsymbol{\theta}}$. The cross-section plots of the log-likelihood (not shown) indicate that $\widehat{\boldsymbol{\theta}}$ is far from the center of the contours. The solver terminates merely because the gradient is smaller than a threshold and the Hessian is positive-definite (recall that we *minimize* the negative log-likelihood). The diverging search starting from $\widehat{\boldsymbol{\theta}}$ (with $\alpha$ and $\ell$ continuously increasing) implies that the infimum of the negative log-likelihood may occur at infinity, as can sometimes happen in our experience.

When $\nu = 1.0$, although the search starting at $\widehat{\boldsymbol{\theta}}$ does not diverge, it terminates at a location quite different from $\widehat{\boldsymbol{\theta}}$, with the log-likelihood increased by about 4000, which is arguably a small amount given the number of observations. Such a phenomenon is often caused by the fact that the peak of the log-likelihood is flat (at least along some directions); hence, the exact optimizer is hard to locate. This phenomenon similarly occurs in the case $\nu = 2.0$. Only when $\nu = 1.5$ does restarting the optimization yield $\widehat{\boldsymbol{\theta}}$ that is essentially the same as the initial estimate. Incidentally, the log-likelihood in this case is also the largest. Hence, all subsequent results are produced for only $\nu = 1.5$.

Table 5: Estimation results ($\nu = 1.5$).

| Est. w/ | $\widehat{\alpha}$ | $\widehat{\ell}$ | $\widehat{\tau}$ |
|---|---|---|---|
| $k$ | $-0.2875$ (0.0047) | 0.15620 (0.00058) | $-4.9360$ (0.0014) |
| $k_{\mathrm{h}}$ | $-0.2275$ (0.0044) | 0.16640 (0.00058) | $-4.9300$ (0.0015) |

Table 6: Log-likelihood (left) and root mean squared prediction error (right).

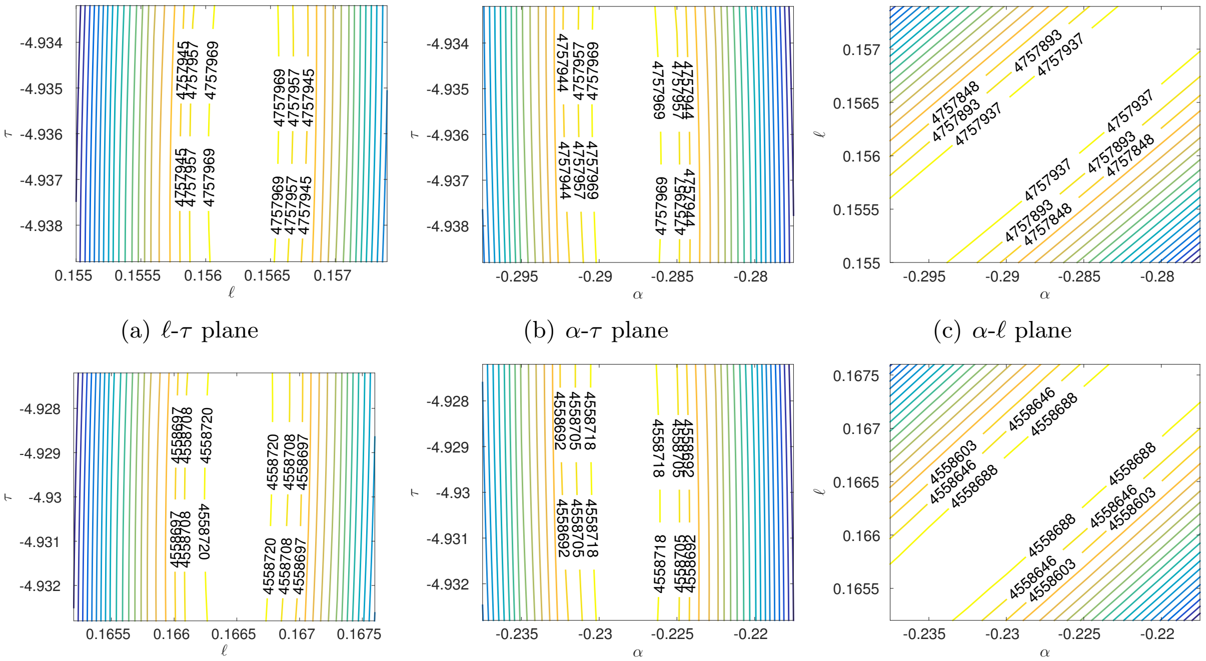
| | at $\widehat{\boldsymbol{\theta}}$ | at $\widehat{\boldsymbol{\theta}}_{\mathrm{h}}$ |
|---|---|---|
| Using $k$ | 4757982 | 4756981 |
| Using $k_{\mathrm{h}}$ | 4557568 | 4558731 |

| | at $\widehat{\boldsymbol{\theta}}$ | at $\widehat{\boldsymbol{\theta}}_{\mathrm{h}}$ |
|---|---|---|
| Using $k$ | 0.01394 | 0.01394 |
| Using $k_{\mathrm{h}}$ | 0.01556 | 0.01556 |

Near $\widehat{\boldsymbol{\theta}}$, we further perform a local grid search and obtain finer estimates, as shown in Table 5. One sees that the estimated parameters produced by $k$ and $k_{\mathrm{h}}$ are qualitatively similar, although their differences exceed the tiny standard errors. To distinguish the two estimates, we use $\widehat{\boldsymbol{\theta}}$ to denote the one resulting from $k$ and $\widehat{\boldsymbol{\theta}}_{\mathrm{h}}$ from $k_{\mathrm{h}}$. In Table 6, we list the log-likelihood values and the kriging errors when the covariance function is evaluated at both locations. One sees that the estimate $\widehat{\boldsymbol{\theta}}_{\mathrm{h}}$ is quite close to $\widehat{\boldsymbol{\theta}}$ in two important regards: first, the root mean squared prediction errors using $k$ are the same to four significant figures, and the log-likelihood under $k$ differs by 1000, which we would argue is a very small difference for more than 2 million observations. On the other hand, $k_{\mathrm{h}}$ does not provide a great approximation to the loglikelihood itself and the predictions using $k_{\mathrm{h}}$ are slightly inferior to those using $k$ no matter which estimate is used. Figure 13 plots the log-likelihoods centered around the respectively optimal estimates. The shapes are visually identical, which supports the use of $k_{\mathrm{h}}$ for parameter estimation. Since kriging with $k$ is often much easier than maximizing the log-likelihood, in this data example one could use $k_{\mathrm{h}}$ to estimate $\boldsymbol{\theta}$ and then $k$ to krige.

## 10 Conclusions

We have presented a computationally friendly approach that addresses the challenge of formidably expensive computations of Gaussian random fields in the large scale. Unlike many methods that focus on the approximation of the covariance matrix or of the likelihood, the proposed approach operates on the covariance function such that positive definiteness is maintained. The hierarchical structure and the nested bases in the proposed construction allow for organizing various computations in a tree format, achieving costs proportional to the tree size and hence to the data size $n$. These computations range from the simulation of random fields to kriging and likelihood evaluations. More desirably, kriging has an amortized cost of $O(\log n)$ and hence one may perform predictions for as many as $O(n)$ sites easily. Moreover, the efficient evaluation of the log-likelihoods paves the way for maximum likelihood estimation as well as Markov Chain Monte Carlo. Numerical experiments show that the proposed construction yields comparable prediction results and likelihood surfaces with those of the base covariance function, while being scalable to data of ever increasing size.



(a) $\ell$-$\tau$ plane

(b) $\alpha$-$\tau$ plane

(c) $\alpha$-$\ell$ plane

(d) $\ell$-$\tau$ plane

(e) $\alpha$-$\tau$ plane

(f) $\alpha$-$\ell$ plane

Figure 13: Log-likelihood centered around optimum. Top row: base covariance function $k$; bottom row: proposed covariance function $k_{\mathrm{h}}$.

## References

Sivaram Ambikasaran and Michael O'Neil. Fast symmetric factorization of hierarchical matrices with applications. arXiv preprint arXiv:1405.0223, 2014.

Sivaram Ambikasaran, Daniel Foreman-Mackey, Leslie Greengard, David W. Hogg, and Michael O'Neil. Fast direct methods for Gaussian processes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 38:252–265, 2016.

E. Anderson, Z. Bai, C. Bischof, L. S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen. *LAPACK Users' Guide*. Society for Industrial and Applied Mathematics, 1999.

Mihai Anitescu, Jie Chen, and Lei Wang. A matrix-free approach for solving the parametric Gaussian process maximum likelihood problem. *SIAM Journal on Scientific Computing*, 34(1): A240–A262, 2012.

Mihai Anitescu, Jie Chen, and Michael L. Stein. An inversion-free estimating equations approach for Gaussian process models. *Journal of Computational and Graphical Statistics*, 26(1):98–107, 2017.

W. F. III Arnold and A. J. Laub. Generalized eigenproblem algorithms and software for algebraic Riccati equations. *Proceedings of the IEEE*, 72(12):1746–1754, 1984.

Erlend Aune, Daniel P. Simpson, and Jo Eidsvik. Parameter estimation in high dimensional Gaussian distributions. *Statistics and Computing*, 24(2):247–263, 2014.

J. E. Barnes and P. Hut. A hierarchical $O(N \log N)$ force-calculation algorithm. *Nature*, 324: 446–449, 1986.

M. Bebendorf and W. Hackbusch. Stabilized rounded addition of hierarchical matrices. *Numer. Lin. Alg. Appl.*, 4(15):407–423, 2007.

Jon Louis Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 8(9):509–517, 1975.

Steffen Börm, Lars Grasedyck, and Wolfgang Hackbusch. Introduction to hierarchical matrices with applications. *Engineering Analysis with Boundary Elements*, 27(5):405–422, 2003.

P. C. Caragea and R. L. Smith. Asymptotic properties of computationally efficient alternative estimators for a class of multivariate normal models. *Journal of Multivariate Analysis*, 98:1417–1440, 2007.

S. Chandrasekaran, P. Dewilde, M. Gu, W. Lyons, and T. Pals. A fast solver for HSS representations via sparse matrices. *SIAM J. Matrix Anal. Appl.*, 29(1):67–81, 2006a.

S. Chandrasekaran, M. Gu, and T. Pals. A fast $ULV$ decomposition solver for hierarchically semiseparable representations. *SIAM J. Matrix Anal. Appl.*, 28(3):603–622, 2006b.

Jie Chen. On the use of discrete Laplace operator for preconditioning kernel matrices. *SIAM Journal on Scientific Computing*, 35(2):A577–A602, 2013.

Jie Chen, Lei Wang, and Mihai Anitescu. A fast summation tree code for Matérn kernel. *SIAM Journal on Scientific Computing*, 36(1):A289–A309, 2014.

J. P. Chilès and P. Delfiner. *Geostatistics: Modeling Spatial Uncertainty.* New York: Wiley-Interscience, 2nd edition, 2012.

N. Cressie and G. Johannesson. Fixed rank kriging for very large spatial data sets. *Journal of the Royal Statistical Society, Series B*, 70:209–226, 2008.

R. Dahlhaus and H. Künsch. Edge effects and efficient parameter estimation for stationary random fields. *Biometrika*, 74:877–882, 1987.

Abhirup Datta, Sudipto Banerjee, Andrew O. Finley, and Alan E. Gelfand. Hierarchical nearest-neighbor Gaussian process models for large geostatistical datasets. *Journal of the American Statistical Association*, 111(54):800–812, 2016a.

Abhirup Datta, Sudipto Banerjee, Andrew O. Finley, and Alan E. Gelfand. On nearest-neighbor Gaussian process models for massive spatial data. *WIREs Comput Stat*, 8:162–171, 2016b.

Kun Dong, David Eriksson, Hannes Nickisch, David Bindel, and Andrew Gordon Wilson. Scalable log determinants for Gaussian process kernel learning. In *Advances in Neural Information Processing Systems 30*, 2017.

Petros Drineas and Michael W. Mahoney. On the Nyström method for approximating a Gram matrix for improved kernel-based learning. *Journal of Machine Learning Research*, 6:2153–2175, 2005.

M. Eidsvik, A. O. Finley, S. Banerjee, and H. Rue. Approximate bayesian inference for large spatial datasets using predictive process models. *Computational Statistics and Data Analysis*, 56:1362–1380, 2012.

Andrew O. Finley, Huiyan Sang, Sudipto Banerjee, and Alan E. Gelfand. Improving the performance of predictive process modeling for large datasets. *Comput Stat Data Anal.*, 53(8): 2873–2884, 2009.

William Fong and Eric Darve. The black-box fast multipole method. *J. Comput. Phys.*, 228(23): 8712–8725, 2009.

M. Fuentes. Approximate likelihood for large irregularly spaced spatial data. *Journal of the American Statistical Association*, 102:321–331, 2007.

Reinhard Furrer, Marc G Genton, and Douglas Nychka. Covariance tapering for interpolation of large spatial datasets. *Journal of Computational and Graphical Statistics*, 15(3):502–523, 2006.

Florian Gerber, Rogier de Jong, Michael E. Schaepman, Gabriela Schaepman-Strub, and Reinhard Furrer. Predicting missing values in spatio-temporal remote sensing data. *IEEE Transactions on Geoscience and Remote Sensing*, 56(5):2841–2853, 2018.

Zydrunas Gimbutas and Vladimir Rokhlin. A generalized fast multipole method for nonoscillatory kernels. *SIAM Journal on Scientific Computing*, 24(3):796–817, 2002.

Gene H. Golub and Charles F. Van Loan. *Matrix Computations.* Johns Hopkins University Press, 3rd edition, 1996.

Kazushige Goto and Robert Van De Geijn. High-performance implementation of the level-3 BLAS. *ACM Transactions on Mathematical Software*, 35(1):4:1–4:14, 2008.

Robert B. Gramacy and Daniel W. Apley. Local Gaussian process approximation for large computer experiments. *Journal of Computational and Graphical Statistics*, 24(2):561–578, 2015.

L. Greengard and V. Rokhlin. A fast algorithm for particle simulations. *Journal of Computational Physics*, 73:325–348, 1987.

Joseph Guinness. Spectral density estimation for random fields via periodic embeddings. *Biometrika*, 106(2):267–286, 2019.

Joseph Guinness and Montserrat Fuentes. Circulant embedding of approximate covariances for inference from Gaussian data on large lattices. *Journal of Computational and Graphical Statistics*, 26(1):88–97, 2017.

X. Guyon. Parameter estimation for a stationary process on a $d$-dimensional lattice. *Biometrika*, 69:95–105, 1982.

W. Hackbusch. A sparse matrix arithmetic based on $\mathcal{H}$-matrices, part I: Introduction to $\mathcal{H}$-matrices. *Computing*, 62(2):89–108, 1999.

W. Hackbusch and S. Börm. Data-sparse approximation by adaptive $\mathcal{H}^2$-matrices. *Computing*, 69 (1):1–35, 2002.

Insu Han, Dmitry Malioutov, Haim Avron, and Jinwoo Shin. Approximating spectral sums of large-scale matrices using Chebyshev approximations. *SIAM Journal on Scientific Computing*, 39(4):A1558–A1585, 2017.

Matthew J. Heaton, Abhirup Datta, Andrew Finley, Reinhard Furrer, Rajarshi Guhaniyogi, Florian Gerber, Robert B. Gramacy, Dorit Hammerling, Matthias Katzfuss, Finn Lindgren, Douglas W. Nychka, Furong Sun, and Andrew Zammit-Mangion. A case study competition among methods for analyzing large spatial data. *Journal of Agricultural, Biological and Environmental Statistics*, 24:398–425, 2019.

Kenneth L Ho and Lexing Ying. Hierarchical interpolative factorization for elliptic operators: integral equations. arXiv preprint arXiv:1307.2666, 2013.

P. Huang, H. Avron, T. N. Sainath, V. Sindhwani, and B. Ramabhadran. Kernel methods match deep neural networks on TIMIT. In *IEEE International Conference on Acoustics, Speech and Signal Processing*, 2014.

A. Iske, S. Le Borne, and M. Wende. Hierarchical matrix approximation for kernel-based scattered data interpolation. *SIAM Journal on Scientific Computing*, 39(5):A2287–A2316, 2017.

Richard H. Jones. Stochastic processes on a sphere. *The Annals of Mathematical Statistics*, 34(1): 213–218, 1963.

Mikyoung Jun and Michael L. Stein. Nonstationary covariance models for global data. *The Annals of Applied Statistics*, 2(4):1271–1289, 2008.

Matthias Katzfuss. A multi-resolution approximation for massive spatial datasets. *Journal of the American Statistical Association*, 112(517):201–214, 2017.

Cari Kaufman, Mark Schervish, and Douglas Nychka. Covariance tapering for likelihood-based estimation in large spatial data sets. *Journal of the American Statistical Association*, 103:1545–1555, 2008.

J. R. Koehler and A. B. Owen. *Handbook of statistics*, volume 13, chapter 9 Computer Experiments, pages 261–308. Elsevier B.V., 1996.

Alan J. Laub. A Schur method for solving algebraic Riccati equations. *IEEE Transcation on Automatic Control*, AC-24(6):913–921, 1979.

S. Li, M. Gu, C. Wu, and J. Xia. New efficient and robust HSS Cholesky factorization of SPD matrices. *SIAM J. Matrix Anal. Appl.*, 33:886–904, 2012.

William B. March, Bo Xiao, and George Biros. ASKIT: Approximate skeletonization kernel-independent treecode in high dimensions. *SIAM Journal on Scientific Computing*, 37(2):A1089–A1110, 2015.

P. G. Martinsson and V. Rokhlin. An accelerated kernel-independent fast multipole method in one dimension. *SIAM Journal on Scientific Computing*, 29(3):1160–1178, 2007.

Victor Minden, Anil Damle, Kenneth L. Ho, and Lexing Ying. Fast spatial Gaussian process maximum likelihood estimation via skeletonization factorizations. arXiv Preprint arXiv:1603.08057, 2016.

Stanislav Minsker. Geometric median and robust estimation in banach spaces. *Bernoulli*, 21(4): 2308–2335, 2015.

Stanislav Minsker, Sanvesh Srivastava, Lizhen Lin, and David B. Dunson. Robust and scalable Bayes via a median of subset posterior measures. *Journal of Machine Learning Research*, 18 (124):1–40, 2017.

Douglas Nychka, Soutir Bandyopadhyay, Dorit Hammerling, Finn Lindgren, and Stephan Sain. A multiresolution Gaussian process model for the analysis of large spatial datasets. *Journal of Computational and Graphical Statistics*, 24(2):579–599, 2015.

Ali Rahimi and Ben Recht. Random features for large-scale kernel machines. In *Neural Infomration Processing Systems*, 2007.

C.E. Rasmussen and C. Williams. *Gaussian Processes for Machine Learning.* MIT Press, 2006.

Høavard Rue, Sara Martino, and Nicolas Chopin. Approximate Bayesian inference for latent Gaussian models by using integrated nested Laplace approximations. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 72(2):319–392, 2009.

Suranjana Saha, Shrinivas Moorthi, Hua-Lu Pan, and Coauthors. The NCEP climate forecast system reanalysis. *Bulletin of the American Meteorological Society*, 91:1015–1057, 2010.

Ralph C. Smith. *Uncertainty Quantification: Theory, Implementation, and Applications.* Society for Industrial and Applied Mathematics, 2013.

M. L. Stein. Fixed domain asymptotics for spatial periodograms. *Journal of the American Statistical Association*, 90:1277–1288, 1995.

M. L. Stein. *Interpolation of Spatial Data: Some Theory for Kriging.* New York: Springer, 1999.

M. L. Stein. A modeling approach for large spatial datasets. *Journal of the Korean Statistical Society*, 37:3–10, 2008.

M. L. Stein. Statistical properties of covariance tapers. *Journal of Computational and Graphical Statistics*, 22(4):866–885, 2013.

M. L. Stein, Z. Chi, and L. J. Welty. Approximating likelihoods for large spatial datasets. *Journal of the Royal Statistical Society, Series B*, 66:275–296, 2004.

M. L. Stein, J. Chen, and M. Anitescu. Difference filter preconditioning for large covariance matrices. *SIAM J. Matrix Anal. Appl.*, 33(1):52–72, 2012.

Michael L. Stein. Limitations on low rank approximations for covariance matrices of spatial data. *Spatial Statistics*, 8:1–19, 2014.

Michael L. Stein, Jie Chen, and Mihai Anitescu. Stochastic approximation of score functions for Gaussian processes. *Annals of Applied Statistics*, 7(2):1162–1191, 2013.

Xiaobai Sun and Nikos P. Pitsianis. A matrix version of the fast multipole method. *SIAM Review*, 43(2):289–300, 2001.

Shashanka Ubaru, Jie Chen, and Yousef Saad. Fast estimation of $\mathrm{tr}(f(A))$ via stochastic Lanczos quadrature. *SIAM J. Matrix Anal. Appl.*, 38(4):1075–1099, 2017.

C. Varin, N. Reid, and D. Firth. An overview of composite likelihood methods. *Statistica Sinica*, 21:5–42, 2011.

A. V. Vecchia. Estimation and model identification for continuous spatial processes. *Journal of the Royal Statistical Society, Series B*, 50:297–312, 1988.

D. Wang and W.-L. Loh. On fixed-domain asymptotics and covariance tapering in gaussian random field models. *Electronic Journal of Statistics*, 5:238–269, 2011.

S. Wang, X. S. Li, J. Xia, Y. Situ, and M. V. de Hoop. Efficient scalable algorithms for solving dense linear systems with hierarchically semiseparable structures. *SIAM Journal on Scientific Computing*, 35(6):C519–C544, 2013.

P. Whittle. On stationary processes in the plane. *Biometrika*, 41:434–449, 1954.

J. Xia, S. Chandrasekaran, M. Gu, and X. S. Li. Fast algorithms for hierarchically semiseparable matrices. *Numer. Lin. Alg. Appl.*, 17(6):953–976, 2010.

Jianlin Xia and Ming Gu. Robust approximate Cholesky factorization of rank-structured symmetric positive definite matrices. *SIAM J. Matrix Anal. Appl.*, 31(5):2899–2920, 2010.

Lexing Ying, George Biros, and Denis Zorin. A kernel-independent adaptive fast multipole algorithm in two and three dimensions. *J. Comput. Phys.*, 196(2):591–626, 2004.

Y. Zhang. Uniformly distributed seeds for randomized trace estimator on $O(N^2)$-operation log-det approximation in gaussian process regression. In *Proceedings of the 2006 IEEE International Conference on Networking, Sensing and Control*, pages 498–503, 2006.

# A Proof of Theorem 1

For a proof of positive definiteness, we write $k_{\mathrm{h}}$ as a sum of two functions $\xi^{(1)}$ and $\xi^{(2)}$, where

$$\xi^{(1)}(\boldsymbol{x}, \boldsymbol{x}') = k(\boldsymbol{x}, \underline{X}) k(\underline{X}, \underline{X})^{-1} k(\underline{X}, \boldsymbol{x}')$$

is the Nyström approximation in the whole domain $S$ and hence positive definite, and

$$\xi^{(2)}(\boldsymbol{x}, \boldsymbol{x}') = \begin{cases} k(\boldsymbol{x}, \boldsymbol{x}') - k(\boldsymbol{x}, \underline{X}) k(\underline{X}, \underline{X})^{-1} k(\underline{X}, \boldsymbol{x}'), & \text{if } \boldsymbol{x}, \boldsymbol{x}' \in S_j \text{ for some } j, \\ 0, & \text{otherwise,} \end{cases}$$

is a Schur complement in each subdomain $S_j$ and hence also positive definite. Then, the constructed $k_{\mathrm{h}}$ is positive definite.

To prove the strict positive definiteness, we need the following lemma.

**Lemma 5.** *Let $k$ be strictly positive definite. For any set of points $X = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\}$ such that $X \cap \underline{X} = \emptyset$ and for any set of coefficients $\alpha_1, \ldots, \alpha_n \in \mathbb{R}$ that are not all zero, we have*

$$\sum_{i,j=1}^{n} \alpha_i \alpha_j \left[ k(\boldsymbol{x}_i, \boldsymbol{x}_j) - k(\boldsymbol{x}_i, \underline{X}) k(\underline{X}, \underline{X})^{-1} k(\underline{X}, \boldsymbol{x}_j) \right] > 0.$$

*Proof.* The result is equivalent to saying that the matrix $k(X, X) - k(X, \underline{X}) k(\underline{X}, \underline{X})^{-1} k(\underline{X}, X)$ is positive definite. To see so, consider

$$k(X \cup \underline{X}, X \cup \underline{X}) = \begin{bmatrix} k(X, X) & k(X, \underline{X}) \\ k(\underline{X}, X) & k(\underline{X}, \underline{X}) \end{bmatrix}.$$

Because of the strict positive definiteness of the function $k$, the matrix $k(X \cup \underline{X}, X \cup \underline{X})$ is positive definite. Then, the Schur complement matrix $k(X, X) - k(X, \underline{X}) k(\underline{X}, \underline{X})^{-1} k(\underline{X}, X)$ is also positive definite. □

We now continue the proof of Theorem 1. For a set of coefficients $\alpha_1, \ldots, \alpha_n \in \mathbb{R}$,

$$\sum_{i,j=1}^{n} \alpha_i \alpha_j k_{\mathrm{h}}(\boldsymbol{x}_i, \boldsymbol{x}_j) = \underbrace{\sum_{i,j=1}^{n} \alpha_i \alpha_j \xi^{(1)}(\boldsymbol{x}_i, \boldsymbol{x}_j)}_{B_1} + \underbrace{\sum_{i,j=1}^{n} \alpha_i \alpha_j \xi^{(2)}(\boldsymbol{x}_i, \boldsymbol{x}_j)}_{B_2}. \tag{16}$$

If we want the left-hand side to be zero, $B_1$ and $B_2$ must be simultaneously zero. Because $\xi^{(2)}(\boldsymbol{x}, \boldsymbol{x}')$ is zero whenever $\boldsymbol{x}$ or $\boldsymbol{x}'$ belongs to $\underline{X}$, based on Lemma 5, $B_2 = 0$ implies that $\alpha_i = 0$ for all $\boldsymbol{x}_i \notin \underline{X}$. In such a case, $B_1$ is simplified to

$$B_1 = \sum_{\boldsymbol{x}_i, \boldsymbol{x}_j \in \underline{X}} \alpha_i \alpha_j \xi^{(1)}(\boldsymbol{x}_i, \boldsymbol{x}_j) = \underline{\boldsymbol{\alpha}}^T k(\underline{X}, \underline{X}) \underline{\boldsymbol{\alpha}},$$

where $\underline{\boldsymbol{\alpha}}$ is the column vector of $\alpha_i$'s for all $\boldsymbol{x}_i \in \underline{X}$. Then, because of the strict positive definiteness of $k$, $B_1 = 0$ implies that $\alpha_i = 0$ for all $\boldsymbol{x}_i \in \underline{X}$. Thus, all coefficients $\alpha_i$ must be zero for the left-hand side of (16) to be zero. This concludes that $k_{\mathrm{h}}$ is strictly positive definite.

# B Proof of Theorem 2

The positive definiteness of $k_{\mathrm{h}}$ straightforwardly follows from the discussion in the main text: $k_{\mathrm{h}}$ is the sum of all $\xi^{(i)}$'s, each of which is positive definite.

To prove strict positive definiteness, we first simplify notations. We write for the covariance function $k$:

$$k_{\boldsymbol{x},\boldsymbol{x}'} \equiv k(\boldsymbol{x},\boldsymbol{x}'), \quad k_{\boldsymbol{x},\underline{i}} \equiv k(\boldsymbol{x},\underline{X}_i), \quad k_{\underline{i},\underline{j}} \equiv k(\underline{X}_i,\underline{X}_j),$$

and similarly for the auxiliary function $\psi^{(i)}$. Then, (9) is simplified to

$$\xi^{(i)}(\boldsymbol{x},\boldsymbol{x}') = 0 \text{ if either } \boldsymbol{x} \text{ or } \boldsymbol{x}' \notin S_i\text{; otherwise:}$$

$$\xi^{(i)}(\boldsymbol{x},\boldsymbol{x}') = \begin{cases} k_{\boldsymbol{x},\boldsymbol{x}'} - k_{\boldsymbol{x},\underline{p}} k_{\underline{p},\underline{p}}^{-1} k_{\underline{p},\boldsymbol{x}'}, & \text{if } i \text{ is leaf,} \\ \psi^{(i)}_{\boldsymbol{x},\underline{i}} k_{\underline{i},\underline{i}}^{-1} \left( k_{\underline{i},\underline{i}} - k_{\underline{i},\underline{p}} k_{\underline{p},\underline{p}}^{-1} k_{\underline{p},\underline{i}} \right) k_{\underline{i},\underline{i}}^{-1} \psi^{(i)}_{\underline{i},\boldsymbol{x}'}, & \text{if } i \text{ is neither leaf nor root,} \\ \psi^{(i)}_{\boldsymbol{x},\underline{i}} k_{\underline{i},\underline{i}}^{-1} \psi^{(i)}_{\underline{i},\boldsymbol{x}'}, & \text{if } i \text{ is root.} \end{cases}$$

We need the following lemma.

**Lemma 6.** *Let $l$ be a leaf descendant of some nonleaf node $i$ and let $(l, l_1, l_2, \ldots, l_s, i)$ be the path connecting $l$ and $i$. Then,*

$$\psi^{(i)}_{\boldsymbol{x},\underline{i}} = k_{\boldsymbol{x},\underline{i}}$$

*if $\boldsymbol{x} \in \underline{X}_{l_1} \cap \underline{X}_{l_2} \cap \cdots \cap \underline{X}_{l_s}$.*

*Proof.* The result is a straightforward verification. For an array of distinct points which contains some point $\boldsymbol{x}$ at the $j$-th location, we use the notation $\boldsymbol{e}_{\boldsymbol{x}}$ to denote a column vector whose $j$-th element is 1 and otherwise 0. Then, for $\boldsymbol{x} \in S_l$ and also $\in \underline{X}_{l_1}$,

$$\psi^{(i)}_{\boldsymbol{x},\underline{i}} = k_{\boldsymbol{x},\underline{l_1}} k_{\underline{l_1},\underline{l_1}}^{-1} k_{\underline{l_1},\underline{l_2}} k_{\underline{l_2},\underline{l_2}}^{-1} \cdots k_{\underline{l_s},\underline{l_s}}^{-1} k_{\underline{l_s},\underline{i}} = \boldsymbol{e}_{\boldsymbol{x}}^T k_{\underline{l_1},\underline{l_2}} k_{\underline{l_2},\underline{l_2}}^{-1} \cdots k_{\underline{l_s},\underline{l_s}}^{-1} k_{\underline{l_s},\underline{i}} = k_{\boldsymbol{x},\underline{l_2}} k_{\underline{l_2},\underline{l_2}}^{-1} \cdots k_{\underline{l_s},\underline{l_s}}^{-1} k_{\underline{l_s},\underline{i}}.$$

Iteratively simplifying by noting that $\boldsymbol{x}$ also belongs to $\underline{X}_{l_1}, \ldots, \underline{X}_{l_s}$, we eventually reach

$$\psi^{(i)}_{\boldsymbol{x},\underline{i}} = k_{\boldsymbol{x},\underline{l_s}} k_{\underline{l_s},\underline{l_s}}^{-1} k_{\underline{l_s},\underline{i}} = \boldsymbol{e}_{\boldsymbol{x}}^T k_{\underline{l_s},\underline{i}} = k_{\boldsymbol{x},\underline{i}}.$$

□

We now continue the proof of Theorem 2. The strategy resembles induction. For a set of coefficients $\alpha_1, \ldots, \alpha_n \in \mathbb{R}$, write

$$\sum_{j,l=1}^{n} \alpha_j \alpha_l k_{\mathrm{h}}(\boldsymbol{x}_j, \boldsymbol{x}_l) = \sum_i \underbrace{\sum_{j,l=1}^{n} \alpha_j \alpha_l \xi^{(i)}(\boldsymbol{x}_j, \boldsymbol{x}_l)}_{B_i}. \tag{17}$$

If we want the left-hand side to be zero, all the $B_i$'s on the right must be simultaneously zero. When $i$ is a leaf node, $\xi^{(i)}(\boldsymbol{x},\boldsymbol{x}')$ is zero whenever $\boldsymbol{x}$ or $\boldsymbol{x}'$ belongs to $\underline{X}_p$, where $p$ is the parent of $i$. Then, $B_i = 0$ implies that $\alpha_j = 0$ for all $\boldsymbol{x}_j \in S_i \backslash \underline{X}_p$.

For any nonleaf node $p$, we use $Q_p$ to denote the union of the intersections of landmark points:

$$Q_p \equiv \bigcup_{l \text{ is leaf descendant of } p} \{ \underline{X}_{l_1} \cap \cdots \cap \underline{X}_{l_s} \cap \underline{X}_p \mid (l, l_1, \ldots, l_s, p) \text{ is a path connecting } l \text{ and } p \}.$$

Clearly, $Q_p \subset S_p$. As a special case, if all the children of $p$ are leaf nodes, $Q_p = \underline{X}_p$. We now have an induction hypothesis: for a nonroot node $i$ with parent $p$, there holds $\alpha_j = 0$ for all $\boldsymbol{x}_j \in S_i \backslash (Q_i \cap \underline{X}_p)$. Assume that the hypothesis is true for all child nodes of some node $p$, who has a parent $q$. Then, summarizing the results for all these child nodes, we have $\alpha_j = 0$ for all $\boldsymbol{x}_j \in S_p \backslash Q_p$. Furthermore, based on Lemma 6, $\xi^{(p)}(\boldsymbol{x}, \boldsymbol{x}')$ is zero whenever $\boldsymbol{x}$ or $\boldsymbol{x}'$ belongs to $Q_p \cap \underline{X}_q$. Then, $B_p = 0$ implies that $\alpha_j = 0$ for all $\boldsymbol{x}_j \in (S_p \backslash Q_p) \cup (Q_p \backslash \underline{X}_q) = S_p \backslash (Q_p \cap \underline{X}_q)$. This finishes the induction step.

At the end of the induction, we reach the root node $p$. Summarizing the results for all the child nodes of the root, we have $\alpha_j = 0$ for all $\boldsymbol{x}_j \in S_p \backslash Q_p$. Invoking Lemma 6 again, we have $\xi^{(p)}(\boldsymbol{x}, \boldsymbol{x}') = k_{\boldsymbol{x},\boldsymbol{x}'}$ whenever $\boldsymbol{x}$ or $\boldsymbol{x}'$ belongs to $Q_p$. Then, by the strict positive definiteness of $k$, $B_p = 0$ implies that $\alpha_j = 0$ for all $\boldsymbol{x}_j \in Q_p$. Thus, all coefficients $\alpha_i$ must be zero for the left-hand side of (17) to be zero. This concludes that $k_{\mathrm{h}}$ is strictly positive definite.

# C Algorithm for Matrix-vector Multiplication

The objective is to compute $\boldsymbol{y} = A\boldsymbol{b}$. We will use a shorthand notation $\boldsymbol{b}_i$ to denote a subvector of $\boldsymbol{b}$ that corresponds to the index set $I_i$; and similarly for the vector $\boldsymbol{y}$. In computer implementation, only the subvectors corresponding to leaf nodes are stored therein. On the other hand, we need auxiliary vectors $\boldsymbol{c}_j$ and $\boldsymbol{d}_j$, all of length $r$, to be stored in each nonroot node $j$. These auxiliary vectors are defined in the following context.

The vector $\boldsymbol{y}$ is the sum of two parts: the first part comes from $A_{ll}\boldsymbol{b}_l$ for every leaf node $l$ and the second part comes from $A_{ij}\boldsymbol{b}_j$ for every pair of sibling nodes $i$ and $j$. The first part is straightforward to calculate. The second part, however, needs an expansion through change of basis according to Definition 1. In particular, let $l$ be a leaf descendant of $i$. Then, the subvector of $A_{ij}\boldsymbol{b}_j$ corresponding to the index set $I_l$ is

$$U_l W_{l_1} W_{l_2} \cdots W_{l_s} W_i \Sigma_p Z_j^T \left( \sum_{\substack{q \text{ is leaf} \\ (q,q_1,q_2,\ldots,q_t,j) \text{ is path}}} Z_{q_t}^T \cdots Z_{q_2}^T Z_{q_1}^T V_q^T \boldsymbol{b}_q \right),$$

where $p$ is the parent of $i$ and $j$, $(l, l_1, l_2, \ldots, l_s, i)$ is the path connecting $l$ and $i$, and the bracketed expression to the right of $Z_j^T$ sums over all the contributions from any descendant leaf $q$ of $j$.

Many computations in the above summation are duplicated. For example, the term $V_q^T\boldsymbol{b}_q$ at a leaf node $q$ appears in all $A_{ij}\boldsymbol{b}_j$ whenever $q$ is a leaf descendant of $j$. Hence, we define two sets of auxiliary vectors

$$\boldsymbol{c}_i = \begin{cases} V_i^T \boldsymbol{b}_i, & \text{if } i \text{ is leaf,} \\ Z_i^T \displaystyle\sum_{j \in \mathrm{Ch}(i)} \boldsymbol{c}_j, & \text{otherwise,} \end{cases}$$

and

$$\boldsymbol{d}_j = W_i \boldsymbol{d}_i + \sum_{j' \in \mathrm{Ch}(i) \setminus \{j\}} \Sigma_i \boldsymbol{c}_{j'}, \quad \text{for } j \text{ being a child of } i; \quad W_i \boldsymbol{d}_i = 0 \text{ if } i \text{ is root,}$$

as temporary storage to avoid duplicate computation. It is not hard to see that for any leaf node $l$, the final output subvector is $\boldsymbol{y}_l = A_{ll}\boldsymbol{b}_l + U_l\boldsymbol{d}_l$.

By definition, the set of auxiliary vectors $\boldsymbol{c}_i$ may be recursively computed from children to parent, whereas the other set $\{\boldsymbol{d}_j\}$ may be computed in a reverse order, from parent to children. Then, the overall computation consists of two tree walks, one upward and the other downward. This computation is summarized in Algorithm 1. The blue texts highlight the modification of the algorithm when $A$ is symmetric. All subsequent algorithms similarly use blue texts to indicate modifications for symmetry.

**Algorithm 1** Computing $\boldsymbol{y} = A\boldsymbol{b}$

```
Initialize $\boldsymbol{d}_i \leftarrow \boldsymbol{0}$ for each nonroot node $i$ of the tree
UPWARD(root)
DOWNWARD(root)

function UPWARD($i$)
    if $i$ is leaf then
        $\boldsymbol{c}_i \leftarrow V_i^T \boldsymbol{b}_i$;   $\boldsymbol{y}_i \leftarrow A_{ii}\boldsymbol{b}_i$        ▷ if $A$ is symmetric, replace $V_i$ by $U_i$
    else
        for all children $j$ of $i$ do UPWARD($j$) end for
        $\boldsymbol{c}_i \leftarrow Z_i^T \left(\sum_{j \in \mathrm{Ch}(i)} \boldsymbol{c}_j\right)$ if $i$ is not root        ▷ if $A$ is symmetric, replace $Z_i$ by $W_i$
    end if
    if $i$ is not root then
        for all siblings $l$ of $i$ do $\boldsymbol{d}_l \leftarrow \boldsymbol{d}_l + \Sigma_p \boldsymbol{c}_i$ end for        ▷ $p$ is parent of $i$
    end if
end function

function DOWNWARD($i$)
    if $i$ is leaf then $\boldsymbol{y}_i \leftarrow \boldsymbol{y}_i + U_i \boldsymbol{d}_i$ and return end if
    for all children $j$ of $i$ do
        $\boldsymbol{d}_j \leftarrow \boldsymbol{d}_j + W_i \boldsymbol{d}_i$, if $i$ is not root
        DOWNWARD($j$)
    end for
end function
```

# D Algorithm for Matrix Inversion

The objective is to compute $A^{-1}$. We first note that $A^{-1}$ has exactly the same structure as that of $A$. We repeat this observation mentioned in the main paper:

**Theorem 7.** *Let $A$ be recursively low-rank with a partitioning tree $T$ and a positive integer $r$. If $A$ is invertible and additionally, $A_{ii} - U_i \Sigma_p V_i^T$ is also invertible for all pairs of nonroot node $i$ and parent $p$, then there exists a recursively low-rank matrix $\widetilde{A}$ with the same partitioning tree $T$ and integer $r$, such that $\widetilde{A} = A^{-1}$. We denote the corresponding factors of $\widetilde{A}$ to be*

$$\{\widetilde{A}_{ii}, \widetilde{U}_i, \widetilde{V}_i, \widetilde{\Sigma}_p, \widetilde{W}_q, \widetilde{Z}_q \mid i \text{ is leaf, } p \text{ is nonleaf, } q \text{ is neither leaf nor root}\}. \tag{18}$$

This theorem may be proved by construction, which simultaneously gives all the factors in (18). Consider a pair of child node $p$ and parent $q$ and let $p$ have children such as $i$ and $j$. By noting that a diagonal block of $A_{pp}$ is $A_{ii}$ and an off-diagonal block is $A_{ij} = U_i \Sigma_p V_j^T$, we may write $A_{pp} - U_p \Sigma_q V_p^T$ as a block diagonal matrix (with diagonal blocks equal to $A_{ii} - U_i \Sigma_p V_i^T$) plus a rank-$r$ term:

$$A_{pp} - U_p \Sigma_q V_p^T = \operatorname{diag}\left[A_{ii} - U_i \Sigma_p V_i^T\right]_{i \in \mathrm{Ch}(p)} + \begin{bmatrix} \vdots \\ U_i \\ \vdots \end{bmatrix} (\Sigma_p - W_p \Sigma_q Z_p^T) \begin{bmatrix} \cdots & V_i^T & \cdots \end{bmatrix}. \tag{19}$$

In fact, this equation also applies to $p = \text{root}$, in which case one treats $\Sigma_q, W_p, Z_p = 0$. Then, the Sherman–Morrison–Woodbury formula gives the inverse

$$(A_{pp} - U_p \Sigma_q V_p^T)^{-1} = \operatorname{diag}\left[(A_{ii} - U_i \Sigma_p V_i^T)^{-1}\right]_{i \in \mathrm{Ch}(p)} + \begin{bmatrix} \vdots \\ \widetilde{U}_i \\ \vdots \end{bmatrix} \widetilde{\Pi}_p \begin{bmatrix} \cdots & \widetilde{V}_i^T & \cdots \end{bmatrix}, \tag{20}$$

where the tilded factors are related to the non-tilded factors through

$$\widetilde{U}_i = (A_{ii} - U_i \Sigma_p V_i^T)^{-1} U_i, \qquad \widetilde{V}_i = (A_{ii} - U_i \Sigma_p V_i^T)^{-T} V_i, \tag{21}$$

and

$$\widetilde{\Pi}_p = -(I + \widetilde{\Lambda}_p \widetilde{\Xi}_p)^{-1} \widetilde{\Lambda}_p \quad \text{with} \quad \widetilde{\Lambda}_p = \Sigma_p - W_p \Sigma_q Z_p^T \quad \text{and} \quad \widetilde{\Xi}_p = \sum_{i \in \mathrm{Ch}(p)} V_i^T \widetilde{U}_i. \tag{22}$$

Equation (21) immediately gives the $\widetilde{U}_i$ and $\widetilde{V}_i$ factors of $\widetilde{A}$ for all leaf nodes $i$. Further, right-multiplying $U_p$ to both sides of (20) and similarly left-multiplying $V_p^T$ to both sides, we obtain

$$\widetilde{W}_p = (I + \widetilde{\Pi}_p \widetilde{\Xi}_p) W_p \quad \text{and} \quad \widetilde{Z}_p = (I + \widetilde{\Pi}_p^T \widetilde{\Xi}_p^T) Z_p,$$

which give the $\widetilde{W}_p$ and $\widetilde{Z}_p$ factors of $\widetilde{A}$ for all nonleaf and nonroot nodes $p$.

Additionally, (20) may be interpreted as relating the inverse of $A_{pp} - U_p \Sigma_r V_p^T$ at some parent level $p$, to that of $A_{ii} - U_i \Sigma_p V_i^T$ at the child level $i$ with a rank-$r$ correction. Then, let $i$ be a leaf node and $(i, i_1, i_2, \ldots, i_s, 1)$ be the path connecting $i$ and the root $= 1$. We expand the chain of corrections and obtain

$$\widetilde{A}(I_i, I_i) = (A_{ii} - U_i \Sigma_{i_1} V_i^T)^{-1} + \widetilde{U}_i \widetilde{\Pi}_{i_1} \widetilde{V}_i^T + \widetilde{U}_i \widetilde{W}_{i_1} \widetilde{\Pi}_{i_2} \widetilde{Z}_{i_1}^T \widetilde{V}_i^T + \cdots + (\widetilde{U}_i \widetilde{W}_{i_1} \cdots \widetilde{W}_{i_s} \widetilde{\Pi}_1 \widetilde{Z}_{i_s}^T \cdots \widetilde{Z}_{i_1}^T \widetilde{V}_i^T). \tag{23}$$

Meanwhile, for any nonleaf node $p$, the factor $\widetilde{\Sigma}_p$ admits a similar chain of corrections:

$$\widetilde{\Sigma}_p = \widetilde{\Pi}_p + \widetilde{W}_p \widetilde{\Pi}_{p_1} \widetilde{Z}_p^T + \widetilde{W}_p \widetilde{W}_{p_1} \widetilde{\Pi}_{p_2} \widetilde{Z}_{p_1}^T \widetilde{Z}_p^T + \cdots + (\widetilde{W}_p \widetilde{W}_{p_1} \cdots \widetilde{W}_{p_t} \widetilde{\Pi}_1 \widetilde{Z}_{p_t}^T \cdots \widetilde{Z}_{p_1}^T \widetilde{Z}_p^T), \tag{24}$$

where $(p, p_1, p_2, \ldots, p_t, 1)$ is the path connecting $p$ and the root $= 1$. The above two formulas give the $\widetilde{A}_{ii}$ and $\widetilde{\Sigma}_p$ factors of $\widetilde{A}$ for all leaf nodes $i$ and nonleaf nodes $p$.

Hence, the computation of $\widetilde{A}$ consists of two tree walks, one upward and the other downward. In the upward phase, $\widetilde{U}_i$, $\widetilde{V}_i$, $\widetilde{W}_p$, and $\widetilde{Z}_p$ are computed. This phase also computes $(A_{ii} - U_i \Sigma_{i_1} V_i^T)^{-1}$ and $\widetilde{\Pi}_p$ as the starting point of corrections. Then, in the downward phase, a chain of corrections as detailed by (23) and (24) is performed from parent to children, which eventually yields the correct $\widetilde{A}_{ii}$ and $\widetilde{\Sigma}_p$. The overall computation is summarized in Algorithm 2. The algorithm also includes straightforward modifications for the case of symmetric $A$.

**Algorithm 2** Computing $\widetilde{A} = A^{-1}$

```
Upward(root)
Downward(root)

function Upward($i$)
    if $i$ is leaf then
        $\widetilde{A}_{ii} \leftarrow (A_{ii} - U_i \Sigma_p V_i^T)^{-1}$    ▷ $p$ is parent of $i$
                                                  ▷ if $A$ is symmetric, replace $V_i$ by $U_i$
        $\widetilde{U}_i \leftarrow \widetilde{A}_{ii} U_i$
        $\widetilde{V}_i \leftarrow \widetilde{A}_{ii}^T V_i$    ▷ if $A$ is symmetric, no need for this step
        $\widetilde{\Theta}_i \leftarrow V_i^T \widetilde{U}_i$    ▷ if $A$ is symmetric, replace $V_i$ by $U_i$
        return
    end if
    for all children $j$ of $i$ do
        Upward($j$)
        $\widetilde{W}_j \leftarrow (I + \widetilde{\Sigma}_j \widetilde{\Xi}_j) W_j$ if $j$ is not leaf
        $\widetilde{Z}_j \leftarrow (I + \widetilde{\Sigma}_j^T \widetilde{\Xi}_j^T) Z_j$ if $j$ is not leaf    ▷ if $A$ is symmetric, no need for this step
        $\widetilde{\Theta}_j \leftarrow Z_j^T \widetilde{\Xi}_j \widetilde{W}_j$ if $j$ is not leaf    ▷ if $A$ is symmetric, replace $Z_j$ by $W_j$
    end for
    $\widetilde{\Xi}_i \leftarrow \sum_{j \in \mathrm{Ch}(i)} \widetilde{\Theta}_j$
    if $i$ is not root then $\widetilde{\Lambda}_i \leftarrow \Sigma_i - W_i \Sigma_p Z_i^T$ else $\widetilde{\Lambda}_i \leftarrow \Sigma_i$ end if    ▷ $p$ is parent of $i$
                                                  ▷ if $A$ is symmetric, replace $Z_i$ by $W_i$
    $\widetilde{\Sigma}_i \leftarrow -(I + \widetilde{\Lambda}_i \widetilde{\Xi}_i)^{-1} \widetilde{\Lambda}_i$
    for all children $j$ of $i$ do
        $\widetilde{E}_j \leftarrow \widetilde{W}_j \widetilde{\Sigma}_i \widetilde{Z}_j^T$ if $j$ is not leaf    ▷ if $A$ is symmetric, replace $\widetilde{Z}_j$ by $\widetilde{W}_j$
    end for
    $\widetilde{E}_i \leftarrow 0$ if $i$ is root
end function

function Downward($i$)
    if $i$ is leaf then
        $\widetilde{A}_{ii} \leftarrow \widetilde{A}_{ii} + \widetilde{U}_i \widetilde{\Sigma}_p \widetilde{V}_i^T$ if $i$ is not root    ▷ $p$ is parent of $i$
                                                  ▷ if $A$ is symmetric, replace $\widetilde{V}_i$ by $\widetilde{U}_i$
    else
        $\widetilde{E}_i \leftarrow \widetilde{E}_i + \widetilde{W}_i \widetilde{E}_p \widetilde{Z}_i^T$ if $i$ is not root    ▷ $p$ is parent of $i$
                                                  ▷ if $A$ is symmetric, replace $\widetilde{Z}_i$ by $\widetilde{W}_i$
        $\widetilde{\Sigma}_i \leftarrow \widetilde{\Sigma}_i + \widetilde{E}_i$
        for all children $j$ of $i$ do Downward($j$) end for
    end if
end function
```

# E Algorithm for Determinant Computation

The computation of the determinant $\delta = \det(A)$ is rather simple if done simultaneously with the inversion of $A$. The key idea is that one may apply Sylvester's determinant theorem on (19) to obtain

$$\det(A_{pp} - U_p \Sigma_q V_p^T) = \det(I + \widetilde{\Lambda}_p \widetilde{\widetilde{\Xi}}_p) \prod_{i \in \mathrm{Ch}(p)} \det(A_{ii} - U_i \Sigma_p V_i^T), \tag{25}$$

where $\widetilde{\Lambda}_p$ and $\widetilde{\widetilde{\Xi}}_p$ are given in (22). In fact, $I + \widetilde{\Lambda}_p \widetilde{\widetilde{\Xi}}_p$ must have been factorized in order to compute $\widetilde{\Pi}_p$ in (22); hence its determinant is trivial to obtain. Then, the determinant of $A_{pp} - U_p \Sigma_q V_p^T$ at the parent $p$ is the product of those at the children $i$, multiplied by $\det(I + \widetilde{\Lambda}_p \widetilde{\widetilde{\Xi}}_p)$. A simple recursion suffices for obtaining the determinant at the root. The procedure is summarized as Algorithm 3. It is organized as an upward tree walk.

Note that the determinant easily overflows or underflows in finite precision arithmetics. A common treatment is to compute the log-determinant instead, in which case the multiplications in (25) becomes summation. However, the log-determinant may be complex if $\det(A)$ is negative. Hence, if one wants to avoid complex arithmetic, as we do in Algorithm 3, one may use two quantities, the log-absolute-determinant $\log|\delta|$ and the sign $\mathrm{sgn}(\delta)$, to uniquely represent $\delta$.

**Algorithm 3** Computing $\delta = \det(A)$

```
Patch Algorithm 2:
    Line 5: Store $\log|\delta_i|$ and $\mathrm{sgn}(\delta_i)$, where $\delta_i = \det(A_{ii} - U_i \Sigma_p V_i^T)$
    Line 19: Store $\log|\delta_i|$ and $\mathrm{sgn}(\delta_i)$, where $\delta_i = \det(I + \widetilde{\Lambda}_i \widetilde{\widetilde{\Xi}}_i)$
UPWARD(root)

function UPWARD($i$)
    $\log|\delta| \leftarrow \log|\delta_i|$;  $\mathrm{sgn}(\delta) \leftarrow \mathrm{sgn}(\delta_i)$
    if $i$ is not leaf then
        for all children $j$ of $i$ do
            UPWARD($j$)
            $\log|\delta| \leftarrow \log|\delta| + \log|\delta_j|$;  $\mathrm{sgn}(\delta) \leftarrow \mathrm{sgn}(\delta) \cdot \mathrm{sgn}(\delta_j)$
        end for
    end if
    return $\log|\delta|$ and $\mathrm{sgn}(\delta)$
end function
```

## F Algorithm for Cholesky-like Factorization

The objective is to compute a factorization $A = GG^T$ when $A$ is symmetric positive definite. This factorization is not Cholesky in the traditional sense, because $G$ is not triangular. Rather, we would like to compute a $G$ that has the same structure as $A$, so that we can reuse the matrix-vector multiplication developed in Section C on $G$. We repeat the existence theorem of $G$ mentioned in the main paper:

**Theorem 8.** *Let $A$ be recursively low-rank with a partitioning tree $T$ and a positive integer $r$. If $A$ is symmetric, by convention let $A$ be represented by the factors*

$$\{A_{ii}, U_i, U_i, \Sigma_p, W_q, W_q \mid i \text{ is leaf, } p \text{ is nonleaf, } q \text{ is neither leaf nor root}\}.$$

*Furthermore, if $A$ is positive definite and additionally, $A_{ii} - U_i\Sigma_p U_i^T$ is also positive definite for all pairs of nonroot node $i$ and parent $p$, then there exists a recursively low-rank matrix $G$ with the same partitioning tree $T$ and integer $r$, and with factors*

$$\{G_{ii}, U_i, V_i, \Omega_p, W_q, Z_q \mid i \text{ is leaf, } p \text{ is nonleaf, } q \text{ is neither leaf nor root}\},$$

*such that $A = GG^T$.*

Note that in the theorem, $G$ and $A$ share factors $U_i$ and $W_q$. In other words, only the factors $G_{ii}$, $V_i$, $\Omega_p$, and $Z_q$ are to be determined. Similar to matrix inversion, we will prove this theorem through constructing these factors. Consider a pair of child node $p$ and parent $q$ and let $p$ have children such as $i$ and $j$. We repeat (19) for the symmetric case in the following

$$\underbrace{A_{pp} - U_p\Sigma_q U_p^T}_{B_{pp}} = \operatorname{diag}\left[\underbrace{A_{ii} - U_i\Sigma_p U_i^T}_{B_{ii}}\right]_{i\in\operatorname{Ch}(p)} + \begin{bmatrix} \vdots \\ U_i \\ \vdots \end{bmatrix} \underbrace{(\Sigma_p - W_p\Sigma_r W_p^T)}_{\Lambda_p} \begin{bmatrix} \cdots & U_i^T & \cdots \end{bmatrix}, \tag{26}$$

and also write

$$\underbrace{G_{pp} - U_p\Omega_q V_p^T}_{C_{pp}} = \operatorname{diag}\left[\underbrace{G_{ii} - U_i\Omega_p V_i^T}_{C_{ii}}\right]_{i\in\operatorname{Ch}(p)} + \begin{bmatrix} \vdots \\ U_i \\ \vdots \end{bmatrix} D_p \begin{bmatrix} \cdots & V_i^T & \cdots \end{bmatrix} \tag{27}$$

for some $D_p$. Suppose we have computed $B_{ii} = C_{ii}C_{ii}^T$ for all $i \in \operatorname{Ch}(p)$, then equating $B_{pp} = C_{pp}C_{pp}^T$ we obtain

$$C_{ii}V_i = U_i \tag{28}$$

and

$$\Lambda_p = D_p^T + D_p + D_p\Xi_p D_p^T \quad \text{where} \quad \Xi_p = \sum_{i\in\operatorname{Ch}(p)} V_i^T V_i. \tag{29}$$

When $i$ is a leaf node, we let $C_{ii}$ be the Cholesky factor of $B_{ii} = A_{ii} - U_i\Sigma_p U_i^T$. Then, (28) gives the factors $V_i$ of $G$ for all leaf nodes $i$: $V_i = C_{ii}^{-1}U_i$. Further, right-multiplying $V_p$ to both sides of (27) and substituting (28), we have $W_p = (I + D_p\Xi_p)Z_p$, which gives the factors $Z_p$ of $G$ for all nonleaf and nonroot nodes $p$, provided that $D_p$ and $\Xi_p$ are known. The term $\Xi_p$ enjoys a simple recurrence relation that we omit here to avoid tediousness. On the other hand, the term $D_p$ is solved from (29). Equation (29) is a continuous-time algebraic Riccati equation and it admits a symmetric solution $D_p$ when all the eigenvalues of $I + \Xi_p \Lambda_p$ are positive. It is not hard to see that the eigenvalues of $I + \Xi_p \Lambda_p$ are positive if and only if $B_{pp}$ is symmetric positive definite, which is satisfied based on the assumptions of the theorem. The solution $D_p$ may be computed by using the well-known Schur method [Laub, 1979, Arnold and Laub, 1984].

Additionally, (27) may be interpreted as relating the Cholesky-like factor of $B_{pp}$ at some parent level $p$, to that of $B_{ii}$ at the child level $i$ with a rank-$r$ correction. Then, let $i$ be a leaf node and $(i, i_1, i_2, \ldots, i_s, 1)$ be the path connecting $i$ and the root $= 1$. We expand the chain of corrections and obtain

$$G_{ii} = C_{ii} + U_i D_{i_1} V_i^T + U_i W_{i_1} D_{i_2} Z_{i_1}^T V_i^T + \cdots + (U_i W_{i_1} \cdots W_{i_s} D_1 Z_{i_s}^T \cdots Z_{i_1}^T V_i^T). \tag{30}$$

Meanwhile, for any nonleaf node $p$, the factor $\Omega_p$ admits a similar chain of corrections:

$$\Omega_p = D_p + W_p D_{p_1} Z_p^T + W_p W_{p_1} D_{p_2} Z_{p_1}^T Z_p^T + \cdots + (W_p W_{p_1} \cdots W_{p_t} D_1 Z_{p_t}^T \cdots Z_{p_1}^T Z_p^T), \tag{31}$$

where $(p, p_1, p_2, \ldots, p_t, 1)$ is the path connecting $p$ and the root $= 1$. The above two formulas give the $G_{ii}$ and $\Omega_p$ factors of $G$ for all leaf nodes $i$ and nonleaf nodes $p$.

Hence, the computation of $G$ consists of two tree walks, one upward and the other downward. In the upward phase, $V_i$ and $Z_p$ are computed. This phase also computes $C_{ii}$ and $D_p$ as the starting point of corrections. Then, in the downward phase, a chain of corrections as detailed by (30) and (31) are performed from parent to children, which eventually yields the correct $G_{ii}$ and $\Omega_p$. The overall computation is summarized in Algorithm 4.

**Algorithm 4** Cholesky-like factorization $A = GG^T$ (for symmetric positive definite $A$)

```
Copy all factors $U_i$ and $W_i$ from $A$ to $G$
UPWARD(root)
DOWNWARD(root)

function UPWARD($i$)
    if $i$ is leaf then
        Factorize $G_{ii}G_{ii}^T \leftarrow A_{ii} - U_i\Sigma_p U_i^T$;   $V_i \leftarrow G_{ii}^{-1}U_i$;   $\Theta_i \leftarrow V_i^T V_i$        ▷ $p$ is parent of $i$
        return
    end if
    for all children $j$ of $i$ do
        UPWARD($j$)
        $Z_j \leftarrow (I + \Omega_j\Xi_j)^{-1}W_j$ if $j$ is not leaf
        $\Theta_j \leftarrow Z_j^T\Xi_j Z_j$ if $j$ is not leaf
    end for
    $\Xi_i \leftarrow \sum_{j\in\mathrm{Ch}(i)} \Theta_j$
    if $i$ is not root then $\Lambda_i \leftarrow \Sigma_i - W_i\Sigma_p W_i^T$ else $\Lambda_i \leftarrow \Sigma_i$ end if        ▷ $p$ is parent of $i$
    Solve $\Lambda_i = \Omega_i^T + \Omega_i + \Omega_i\Xi_i\Omega_i^T$ for $\Omega_i$
    for all children $j$ of $i$ do
        $E_j \leftarrow W_j\Omega_i Z_j^T$ if $j$ is not leaf
    end for
    $E_i \leftarrow 0$ if $i$ is root
end function

function DOWNWARD($i$)
    if $i$ is leaf then
        $G_{ii} \leftarrow G_{ii} + U_i\Omega_p V_i^T$ if $i$ is not root        ▷ $p$ is parent of $i$
    else
        $E_i \leftarrow E_i + W_i E_p Z_i^T$ if $i$ is not root        ▷ $p$ is parent of $i$
        $\Omega_i \leftarrow \Omega_i + E_i$
        for all children $j$ of $i$ do DOWNWARD($j$) end for
    end if
end function
```

# G Algorithm for Constructing $K_{\mathrm{h}}$

The computation is summarized in Algorithm 5. See Section 4 of the main paper.

**Algorithm 5** Constructing $A = k_{\mathrm{h}}(X, X)$

```
Construct a partitioning tree and for every nonleaf node $i$, find landmark points $\underline{X}_i$
DOWNWARD(root)

function DOWNWARD($i$)
    if $i$ is leaf then
        $A_{ii} \leftarrow k(X_i, X_i)$;   $U_i \leftarrow k(X_i, \underline{X}_p) k(\underline{X}_p, \underline{X}_p)^{-1}$        ▷ $p$ is parent of $i$
        $V_i \leftarrow$ empty matrix
        return
    end if
    $\Sigma_i \leftarrow k(\underline{X}_i, \underline{X}_i)$;
    $W_i \leftarrow k(\underline{X}_i, \underline{X}_p) k(\underline{X}_p, \underline{X}_p)^{-1}$ if $i$ is not root        ▷ $p$ is parent of $i$
    $Z_i \leftarrow$ empty matrix
    for all children $j$ of $i$ do DOWNWARD($j$) end for
end function
```

# H Algorithm for Computing $\boldsymbol{w}^T\boldsymbol{v}$ with $\boldsymbol{v} = k_{\mathrm{h}}(X, \boldsymbol{x})$

To begin with, note that $\boldsymbol{x}$ must lie in one of the subdomains $S_j$ for some leaf node $j$. We will abuse language and say that "$\boldsymbol{x}$ lies in the leaf node $j$" for simplicity. In such a case, the subvector $\boldsymbol{v}_j = k(X_j, \boldsymbol{x})$ and for any leaf node $l \neq j$, the subvector

$$\boldsymbol{v}_l = U_l W_{l_1} W_{l_2} \cdots W_{l_s} \Sigma_p W_{j_t}^T \cdots W_{j_2}^T W_{j_1}^T k(\underline{X}_{j_1}, \underline{X}_{j_1})^{-1} k(\underline{X}_{j_1}, \boldsymbol{x}),$$

where $p$ is the least common ancestor of $j$ and $l$, $(l, l_1, l_2, \ldots, l_s, p)$ is the path connecting $l$ and $p$, and $(j, j_1, j_2, \ldots, j_t, p)$ is the path connecting $j$ and $p$. Then, the inner product

$$\boldsymbol{w}^T\boldsymbol{v} = \boldsymbol{w}_j^T k(X_j, \boldsymbol{x}) + \sum_{l \neq j,\ l \text{ is leaf}} \boldsymbol{w}_l^T U_l W_{l_1} W_{l_2} \cdots W_{l_s} \Sigma_p W_{j_t}^T \cdots W_{j_2}^T W_{j_1}^T k(\underline{X}_{j_1}, \underline{X}_{j_1})^{-1} k(\underline{X}_{j_1}, \boldsymbol{x}).$$

Similar to matrix-vector multiplications, we may define a few sets of auxiliary vectors to avoid duplicate computations. Specifically, define $\boldsymbol{x}$-independent vectors

$$\boldsymbol{e}_i = \begin{cases} U_i^T \boldsymbol{w}_i, & \text{if } i \text{ is leaf,} \\ W_i^T \displaystyle\sum_{j \in \mathrm{Ch}(i)} \boldsymbol{e}_j, & \text{otherwise,} \end{cases}$$

and

$$\boldsymbol{c}_l = \Sigma_p^T \boldsymbol{e}_i \quad \text{for } i \text{ and } l \text{ being siblings with parent } p,$$

and $\boldsymbol{x}$-dependent vectors

$$\boldsymbol{d}_p = W_p^T \boldsymbol{d}_i \quad \text{for } p \text{ being the parent of } i; \qquad \boldsymbol{d}_j = k(\underline{X}_{j_1}, \underline{X}_{j_1})^{-1} k(\underline{X}_{j_1}, \boldsymbol{x}) \quad \text{for } \boldsymbol{x} \text{ lying in } j.$$

Then, the inner product is simplified as

$$\boldsymbol{w}^T\boldsymbol{v} = \boldsymbol{w}_j^T k(X_j, \boldsymbol{x}) + \sum_{j_t \,\in \text{ path connecting } j \text{ and root}} \boldsymbol{c}_{j_t}^T \boldsymbol{d}_{j_t}.$$

Hence, the computation of $\boldsymbol{w}^T\boldsymbol{v}$ consists of a full tree walk and a partial one, both upward. The first upward phase computes $\boldsymbol{e}_i$ from children to parent and simultaneously $\boldsymbol{c}_l$ by crossing sibling nodes from $i$ to $l$. This computation is independent of $\boldsymbol{x}$ and hence is considered preprocessing. The second upward phase computes $\boldsymbol{d}_{j_t}$ for all $j_t$ along the path connecting $j$ and the root. This phase visits only one path but not the whole tree, which is the reason why it costs less than $O(n)$. We summarize the detailed procedure in Algorithm 6.

**Algorithm 6** Computing $z = \boldsymbol{w}^T \boldsymbol{v}$, where $\boldsymbol{v} = k_{\text{h}}(X, \boldsymbol{x})$, for $\boldsymbol{x} \notin X$

```
Common-Upward(root)
▷ The above step is independent of $\boldsymbol{x}$ and is treated as preprocessing. In computer implementation, the intermediate results $\boldsymbol{c}_i$ are carried over to the next step Second-Upward, whereas the contents in $\boldsymbol{d}_i$ are discarded and the allocated memory is reused.
Second-Upward(root)

function Common-Upward($i$)
    if $i$ is leaf then
        $\boldsymbol{d}_i \leftarrow U_i^T \boldsymbol{w}_i$
    else
        for all children $j$ of $i$ do Common-Upward($j$) end for
        $\boldsymbol{d}_i \leftarrow W_i^T \left( \sum_{j \in \text{Ch}(i)} \boldsymbol{d}_j \right)$ if $i$ is not root
    end if
    if $i$ is not root then
        for all siblings $l$ of $i$ do $\boldsymbol{c}_l \leftarrow \Sigma_p^T \boldsymbol{d}_i$ end for        ▷ $p$ is parent of $i$
    end if
end function

function Second-Upward($i$)
    if $i$ is leaf then
        $\boldsymbol{d}_i \leftarrow k(\underline{X}_p, \underline{X}_p)^{-1} k(\underline{X}_p, \boldsymbol{x})$        ▷ $p$ is parent of $i$
        $z \leftarrow \boldsymbol{w}_i^T k(X_i, \boldsymbol{x})$
    else
        Find the child $j$ (among all children of $i$) where $\boldsymbol{x}$ lies in
        Second-Upward($j$)
        $\boldsymbol{d}_i \leftarrow W_i^T \boldsymbol{d}_j$ if $i$ is not root
    end if
    $z \leftarrow z + \boldsymbol{c}_i^T \boldsymbol{d}_i$ if $i$ is not root
end function
```

# I Algorithm for Computing $\boldsymbol{v}^T\widetilde{A}\boldsymbol{v}$ with $\boldsymbol{v} = k_{\mathrm{h}}(X, \boldsymbol{x})$ for Symmetric $\widetilde{A}$

We consider the general case where $\widetilde{A}$ is not necessarily related to the covariance function $k_{\mathrm{h}}$; what is assumed is only symmetry. We recall that $\widetilde{A}$ is represented by the factors

$$\{\widetilde{A}_{ii}, \widetilde{U}_i, \widetilde{U}_i, \widetilde{\Sigma}_p, \widetilde{W}_q, \widetilde{W}_q \mid i \text{ is leaf, } p \text{ is nonleaf, } q \text{ is neither leaf nor root}\}.$$

The derivation of the algorithm is more involved than that of the previous ones; hence, we need to introduce further notations. Let $\mathrm{p}(i)$ denote the parent of a node $i$ and similarly $\mathrm{p}(i,j)$ denote the common parent of $i$ and $j$. Let $(l, l_1, l_2, \ldots, l_t, p)$ be a path connecting nodes $l$ and $p$, where $l$ is a descendant of $p$. Denote this path as $\mathrm{path}(l,p)$ for short. We will use subscripts $l \to p$ and $p \leftarrow l$ to simplify the notation of the product chain of the $W$ factors:

$$W_{l\to p} \equiv W_{l_1}W_{l_2}\cdots W_{l_t} \quad \text{and} \quad W_{p\leftarrow l}^T \equiv W_{l_t}^T \cdots W_{l_2}^T W_{l_1}^T.$$

Note that the two ends of the path (i.e., $l$ and $p$) are not included in the product chain. If $l$ is a leaf and $p$ is the root, then every node $i \in \mathrm{path}(l,p)$, except the root, has the parent also in the path, but its siblings are not. We collect all these sibling nodes to form a set $\mathrm{B}(l)$. It is not hard to see that $\mathrm{B}(l) \cup \{l\}$ is a disjoint partitioning of whole index set. Moreover, any two nodes from the set $\mathrm{B}(l) \cup \{l\}$ must have a least common ancestor belonging to $\mathrm{path}(l, \mathrm{root})$; and this ancestor is the parent of (at least) one of the two nodes. If $\boldsymbol{x}$ lies in a leaf node $l$, $i$ is some node $\in \mathrm{path}(l, \mathrm{root})$, and $j \in \mathrm{B}(l)$ is a sibling of $i$, then by reusing the $\boldsymbol{d}$ vectors defined in the preceding subsection, we have

$$\boldsymbol{v}_l = k(X_l, \boldsymbol{x}) \quad \text{and} \quad \boldsymbol{v}_j = U_j\Sigma_{\mathrm{p}(j)}W_{\mathrm{p}(j)\leftarrow l}^T k(\underline{X}_{\mathrm{p}(l)}, \underline{X}_{\mathrm{p}(l)})^{-1}k(\underline{X}_{\mathrm{p}(l)}, \boldsymbol{x}) = U_j\Sigma_{\mathrm{p}(j,i)}\boldsymbol{d}_i. \tag{32}$$

Because $\mathrm{B}(l) \cup \{l\}$ forms a disjoint partitioning of whole index set, the quadratic form $\boldsymbol{v}^T\widetilde{A}\boldsymbol{v}$ consists of three parts:

$$\boldsymbol{v}^T\widetilde{A}\boldsymbol{v} = \boldsymbol{v}_l^T\widetilde{A}_{ll}\boldsymbol{v}_l + \sum_{i\in\mathrm{B}(l)} \boldsymbol{v}_i^T\widetilde{A}_{ii}\boldsymbol{v}_i + \sum_{\substack{i,j\in\mathrm{B}(l)\\ i\neq j}} \boldsymbol{v}_i^T\widetilde{A}_{ij}\boldsymbol{v}_j.$$

The first part involving the leaf node $l$ is straightforward. For the second part, we expand $\boldsymbol{v}_i$ by using (32) and define two quantities therein:

$$\boldsymbol{v}_i^T\widetilde{A}_{ii}\boldsymbol{v}_i = \Big(\boldsymbol{d}_t^T \underbrace{\Sigma_{\mathrm{p}(i,t)}^T \overbrace{U_i^T\Big)\widetilde{A}_{ii}\Big(U_i}^{\Xi_i} \Sigma_{\mathrm{p}(i,t)}}_{\widetilde{\Xi}_i} \boldsymbol{d}_t\Big),$$

where $t$ as a sibling of $i$ belongs to $\mathrm{path}(l, \mathrm{root})$. For the third part, we similarly expand each individual term and define additionally two quantities:

$$\boldsymbol{v}_i^T\widetilde{A}_{ij}\boldsymbol{v}_j = \Big(\boldsymbol{d}_s^T \underbrace{\Sigma_{\mathrm{p}(i,s)}^T \overbrace{U_i^T\Big)\Big(\widetilde{U}_i}^{\Theta_i^T}}_{\widetilde{\Theta}_i^T} \widetilde{W}_{i\to q}\widetilde{\Sigma}_q\widetilde{W}_{q\leftarrow j}^T \underbrace{\overbrace{\widetilde{U}_j^T\Big)\Big(U_j}^{\Theta_j} \Sigma_{\mathrm{p}(j,t)}}_{\widetilde{\Theta}_j} \boldsymbol{d}_t\Big),$$

where $s$ as a sibling of $i$ belongs to $\mathrm{path}(l, \mathrm{root})$, $t$ as a sibling of $j$ belongs to the same path, and $q$ is the least common ancestor of $i$ and $j$. The four newly introduced quantities $\Xi_i$, $\widetilde{\Xi}_i$, $\Theta_i$, and $\widetilde{\Theta}_i$ are independent of $\boldsymbol{x}$ and may be computed in preprocessing, in a recursive manner from children to parent. We omit the simple recurrence relation here to avoid tediousness. Then, the quadratic form $\boldsymbol{v}^T\widetilde{A}\boldsymbol{v}$ admits the following expression:

$$\boldsymbol{v}^T\widetilde{A}\boldsymbol{v} = \boldsymbol{v}_l^T\widetilde{A}_{ll}\boldsymbol{v}_l + \sum_{i\in\mathrm{B}(l)} \boldsymbol{d}_t^T\widetilde{\Xi}_i\boldsymbol{d}_t + \sum_{\substack{i,j\in\mathrm{B}(l)\\ i\neq j}} \boldsymbol{d}_s^T\widetilde{\Theta}_i^T\widetilde{W}_{i\to q}\widetilde{\Sigma}_q\widetilde{W}_{q\leftarrow j}^T\widetilde{\Theta}_j\boldsymbol{d}_t.$$

We may further simplify the summation in the last term of this equation to avoid duplicate computation. As mentioned, any two nodes in $\mathrm{B}(l)$ have a least common ancestor that happens to be the parent of one of them. Assume that this node is $i$. Then, we write

$$\sum_{\substack{i,j\in\mathrm{B}(l)\\ i\neq j}} \boldsymbol{d}_s^T\widetilde{\Theta}_i^T\widetilde{W}_{i\to q}\widetilde{\Sigma}_q\widetilde{W}_{q\leftarrow j}^T\widetilde{\Theta}_j\boldsymbol{d}_t = 2\sum_{i\in\mathrm{B}(l)} \boldsymbol{d}_s^T\widetilde{\Theta}_i^T\widetilde{\Sigma}_{\mathrm{p}(i)} \sum_{\substack{j\in\mathrm{B}(l),\ j\neq i\\ j \text{ is descendant of } \mathrm{p}(i)}} \widetilde{W}_{\mathrm{p}(i)\leftarrow j}^T\widetilde{\Theta}_j\boldsymbol{d}_t.$$

Note the inner summation on the right-hand side of this equality. This quantity iteratively accumulates as $i$ moves up the tree. Therefore, we define

$$\boldsymbol{c}_i = \begin{cases} \widetilde{\Theta}_i\boldsymbol{d}_s, & \text{if } i\in\mathrm{B}(l),\\ \widetilde{W}_i^T\displaystyle\sum_{j\in\mathrm{Ch}(i)}\boldsymbol{c}_j, & \text{if } i\in\mathrm{path}(l,\mathrm{root}), \end{cases}$$

where recall that $s$ as a sibling of $i$ belongs to $\mathrm{path}(l, \mathrm{root})$. Then, the inner summation becomes $\boldsymbol{c}_s$. In other words,

$$\sum_{\substack{i,j\in\mathrm{B}(l)\\ i\neq j}} \boldsymbol{d}_s^T\widetilde{\Theta}_i^T\widetilde{W}_{i\to q}\widetilde{\Sigma}_q\widetilde{W}_{q\leftarrow j}^T\widetilde{\Theta}_j\boldsymbol{d}_t = 2\sum_{i\in\mathrm{B}(l)} \boldsymbol{c}_i^T\widetilde{\Sigma}_{\mathrm{p}(i,s)}\boldsymbol{c}_s.$$

To summarize, the computation of $\boldsymbol{v}^T\widetilde{A}\boldsymbol{v}$ consists of a full tree walk and a partial one, both upward. The first upward phase computes $\Xi_i$, $\widetilde{\Xi}_i$, $\Theta_i$, and $\widetilde{\Theta}_i$ recursively from children to parent. This computation is independent of $\boldsymbol{x}$ and hence is considered preprocessing. The second upward phase computes $\boldsymbol{d}_s$ and $\boldsymbol{c}_s$ for all $s$ along the path connecting $l$ and the root (assuming $\boldsymbol{x}\in S_l$), as well as all $\boldsymbol{c}_i$ for $i$ being sibling nodes of $s$. This phase visits only one path but not the whole tree, which is the reason why it costs less than $O(n)$. The detailed procedure is given in Algorithm 7.

**Algorithm 7** Computing $z = \boldsymbol{v}^T \widetilde{A} \boldsymbol{v}$, where $\widetilde{A}$ is symmetric and $\boldsymbol{v} = k_{\mathrm{h}}(X, \boldsymbol{x})$, for $\boldsymbol{x} \notin X$

```
Common-Upward(root)
▷ The above step is independent of $\boldsymbol{x}$ and is treated as preprocessing.
Second-Upward(root)

function Common-Upward($i$)
    if $i$ is leaf then
        $\Theta_i \leftarrow \widetilde{U}_i^T U_i$;   $\widetilde{\Theta}_i \leftarrow \Theta_i \Sigma_p$                ▷ $p$ is parent of $i$
        $\Xi_i \leftarrow U_i^T \widetilde{A}_i U_i$;   $\widetilde{\Xi}_i \leftarrow \Sigma_p^T \Xi_i \Sigma_p$                ▷ $p$ is parent of $i$
        return
    end if
    for all children $j$ of $i$ do Common-Upward($j$) end for
    if $i$ is not root then
        $\Theta_i \leftarrow \widetilde{W}_i^T \left( \sum_{j \in \mathrm{Ch}(i)} \Theta_j \right) W_i$;   $\widetilde{\Theta}_i \leftarrow \Theta_i \Sigma_p$                ▷ $p$ is parent of $i$
        $\Xi_i \leftarrow W_i^T \left( \sum_{j \in \mathrm{Ch}(i)} \Xi_j + \sum_{\substack{j,k \in \mathrm{Ch}(i) \\ j \neq k}} \Theta_j^T \widetilde{\Sigma}_i \Theta_k \right) W_i$;   $\widetilde{\Xi}_i \leftarrow \Sigma_p^T \Xi_i \Sigma_p$                ▷ $p$ is parent of $i$
    end if
end function

function Second-Upward($i$)
    if $i$ is leaf then
        $\boldsymbol{d}_i \leftarrow k(\underline{X}_p, \underline{X}_p)^{-1} k(\underline{X}_p, \boldsymbol{x})$                ▷ $p$ is parent of $i$
        $\boldsymbol{c}_i \leftarrow \widetilde{U}_i^T k(X_i, \boldsymbol{x})$
        $z \leftarrow k(\boldsymbol{x}, X_i) \widetilde{A}_i k(X_i, \boldsymbol{x})$
    else
        Find the child $j$ (among all children of $i$) where $\boldsymbol{x}$ lies in
        Second-Upward($j$)
        $\boldsymbol{d}_i \leftarrow W_i^T \boldsymbol{d}_j$ if $i$ is not root
    end if
    if $i$ is not root then
        for all siblings $l$ of $i$ do
            $\boldsymbol{c}_l \leftarrow \widetilde{\Theta}_l \boldsymbol{d}_i$
            $z \leftarrow z + \boldsymbol{d}_i^T \widetilde{\Xi}_l \boldsymbol{d}_i + 2 \boldsymbol{c}_l^T \widetilde{\Sigma}_p \boldsymbol{c}_i$                ▷ $p$ is parent of $i$
        end for
        $\boldsymbol{c}_p \leftarrow \widetilde{W}_p^T \left( \sum_{j \in \mathrm{Ch}(p)} \boldsymbol{c}_j \right)$ if $p$ is not root                ▷ $p$ is parent of $i$
    end if
end function
```

# J Cost Analysis

The storage cost has been analyzed in the main paper. In what follows is the analysis of arithmetic costs.

## J.1 Arithmetic Cost of Matrix-Vector Multiplication (Algorithm 1)

The algorithm consists of two tree walks, each of which visits all the $O(n/r)$ nodes. Inside each tree node, the computation is dominated by $O(1)$ matrix-vector multiplications with $r \times r$ matrices; hence the per-node cost is $O(r^2)$. Then, the overall cost is $O(n/r \times r^2) = O(nr)$.

## J.2 Arithmetic Cost of Matrix Inversion (Algorithm 2)

The algorithm consists of two tree walks, each of which visits all the $O(n/r)$ nodes. Inside each tree node, the computation is dominated by $O(1)$ matrix operations (matrix-matrix multiplications and inversions) with $r \times r$ matrices; hence the per-node cost is $O(r^3)$. Then, the overall cost is $O(n/r \times r^3) = O(nr^2)$.

## J.3 Arithmetic Cost of Determinant Computation (Algorithm 3)

The algorithm requires patching Algorithm 2 with additional computations that do not affect the $O(nr^2)$ cost of Algorithm 2. Omitting the patching, Algorithm 3 visits every tree node once and the computation per node is $O(1)$. Hence, the cost of this algorithm is only $O(n/r)$.

In practice, we indeed implement the patching inside Algorithm 2.

## J.4 Arithmetic Cost of Cholesky-like Factorization (Algorithm 4)

The cost analysis of this algorithm is almost the same as that of Algorithm 2, except that the dominating per-node computation also includes Cholesky factorization of $r \times r$ matrices and the solving of continuous-time algebraic Riccati equation of size $r \times r$. Both costs are $O(r^3)$, the same as that of matrix-matrix multiplications and inversions. Hence, the overall cost of this algorithm is $O(nr^2)$.

## J.5 Arithmetic Cost of Constructing $K_{\mathrm{h}}$ (Algorithm 5)

The algorithm consists of three parts: (i) hierarchical partitioning of the domain; (ii) finding landmark points; and (iii) instantiating the factors of a symmetric recursively low-rank matrix.

For part (i), much flexibility exists. In practice, partitioning is data driven, which ensures that the number of points is balanced in all leaf nodes. If we assume that the cost of partitioning a set of $n$ points is $O(n)$, then the overall partitioning cost counting recursion is $O(n \log n)$.

Similarly, part (ii) depends on the specific method used for choosing the landmark points. In general, we may assume that choosing $r$ landmark points costs $O(r)$. Then, because each of the $O(n/r)$ nonleaf nodes has a set of landmark points, the cost is $O(n/r \times r) = O(n)$.

Part (iii) is a tree walk that visits each of the $O(n/r)$ nodes once. The per-node computation is dominated by constructing one or a few $r \times r$ covariance matrices and performing matrix-matrix multiplications and inversions. We assume that constructing an $r \times r$ covariance matrix costs $O(r^2)$, which is less expensive than the $O(r^3)$ cost of matrix-matrix multiplications and inversions. Then, the overall cost for instantiating the overall matrix is $O(n/r \times r^3) = O(nr^2)$.

### J.6 Arithmetic Cost of Computing $\boldsymbol{w}^T\boldsymbol{v}$ (Algorithm 6)

The algorithm consists of two tree walks (one full and one partial): the first one is $\boldsymbol{x}$-independent preprocessing and the second one is $\boldsymbol{x}$-dependent.

For preprocessing, the tree walk visits all the $O(n/r)$ nodes. Inside each tree node, the computation is dominated by $O(1)$ matrix-vector multiplications with $r \times r$ matrices; hence the per-node cost is $O(r^2)$. Then, the overall preprocessing cost is $O(n/r \times r^2) = O(nr)$.

For the $\boldsymbol{x}$-dependent computation, only $O(h) = O(\log_2(n/r))$ tree nodes are visited. Inside each visited node, the computation is dominated by $O(1)$ matrix-vector multiplications with $r \times r$ matrices; hence the per-node cost is $O(r^2)$. Here, we assume that finding the child node where $\boldsymbol{x}$ lies in has $O(1)$ cost. Note also that although the computation of the $\boldsymbol{d}$ vectors requires a matrix inverse, the matrix in fact has been prefactorized when constructing $K_{\mathrm{h}}$ (that is, inside Algorithm 5). Hence, the per-node cost is not $O(r^3)$. To conclude, the $\boldsymbol{x}$-dependent cost is $O(r^2 \log_2(n/r))$.

### J.7 Arithmetic Cost of Computing $\boldsymbol{v}^T\widetilde{A}\boldsymbol{v}$ (Algorithm 7)

The cost analysis of this algorithm is almost the same as that of Algorithm 6, except that in the preprocessing phase, the dominant per-node computation is $O(1)$ matrix-matrix multiplications with $r \times r$ matrices. Hence, the preprocessing cost is $O(n/r \times r^3) = O(nr^2)$ whereas the $\boldsymbol{x}$-dependent cost is still $O(r^2 \log_2(n/r))$.