Fast Spatial Gaussian Process Maximum Likelihood Estimation via Skeletonization Factorizations

Victor Minden VMINDEN@STANFORD.EDU
Anil Damle DAMLE@STANFORD.EDU
Institute for Computational and Mathematical Engineering
Stanford University
Stanford, CA 94305, USA

Kenneth L. Ho KLHO@ALUMNI.CALTECH.EDU
Department of Mathematics
Stanford University
Stanford, CA 94305, USA

Lexing Ying LEXING@MATH.STANFORD.EDU
Department of Mathematics and Institute for Computational and Mathematical Engineering
Stanford University
Stanford, CA 94305, USA

Editor: X

Abstract

Maximum likelihood estimation for parameter-fitting given observations from a kernelized Gaussian process in space is a computationally-demanding task that restricts the use of such methods to moderately-sized datasets. We present a framework for unstructured observations in two spatial dimensions that allows for evaluation of the log-likelihood and its gradient (*i.e.*, the score equations) in $\tilde{O}(n^{3/2})$ time under certain assumptions, where n is the number of observations. Our method relies on the skeletonization procedure described by Martinsson & Rokhlin (2005) in the form of the recursive skeletonization factorization of Ho & Ying (2015). Combining this with an adaptation of the matrix peeling algorithm of Lin, Lu & Ying (2011) for constructing $\mathcal{H}$-matrix representations of black-box operators, our method can be used in the context of any first-order optimization routine to quickly and accurately compute maximum-likelihood estimates.

Keywords: spatial Gaussian processes, Kriging, hierarchical matrices, maximum likelihood estimation, fast algorithms

1. Introduction

Kernelized Gaussian processes are commonly used in the applied sciences to model observations with an underlying spatial structure. In such applications, each observation $z_i \in \mathbb{R}$ is associated with a corresponding location $x_i \in \mathbb{R}^d$ with $d = 2$ or 3 and the vector of observations $z = [z_1, \ldots, z_n]$ is assumed to be randomly distributed as a multivariate Gaussian random field $z \sim N(0, \Sigma(\theta))$. The covariance matrix $\Sigma(\theta)$ is assumed to be specified up to some parameter $\theta \in \mathbb{R}^p$ by selecting an appropriate covariance kernel $k(x, y)$ that decays smoothly as $\|x-y\| \to \infty$ and letting $[\Sigma(\theta)]_{ij} = k(x_i, x_j; \theta)$. Here, the mean of the process is assumed to be 0 for simplicity, though this is not strictly necessary.

Typically, the parameter vector θ is unknown and must be estimated from the data. For example, given a parameterized family of kernels and a set of observations, we might want to use the observed field values to infer θ for later use in estimating the value of the field at other spatial locations as in Kriging (see Stein, 1999). As such, in this paper we consider the general Gaussian process maximum likelihood estimation (MLE) problem for θ: given an observation vector $z \in \mathbb{R}^n$, find $\hat{\theta}_{\mathrm{MLE}}$ maximizing the Gaussian process log-likelihood

$$\ell(\theta) = -\frac{1}{2} z^T \Sigma^{-1} z - \frac{1}{2}\log|\Sigma| - \frac{n}{2}\log 2\pi, \tag{1}$$

where we have dropped the explicit dependence of Σ on θ.

Taking the gradient of (1) with respect to θ, we obtain components

$$g_i = \frac{\partial \ell(\theta)}{\partial \theta_i} = \frac{1}{2} z^T \Sigma^{-1} \Sigma_i \Sigma^{-1} z - \frac{1}{2}\operatorname{Tr}(\Sigma^{-1}\Sigma_i), \quad i = 1, \ldots, p, \tag{2}$$

where $\Sigma_i = \frac{\partial \Sigma}{\partial \theta_i}$. If θ is unconstrained, $\hat{\theta}_{\mathrm{MLE}}$ is given by maximizing (1) over all of $\mathbb{R}^p$, in which case under certain assumptions it is possible to obtain the maximum likelihood estimate by solving the score equations $g_i = 0$ for $i = 1, \ldots, p$ without evaluating $\ell(\theta)$ as is done, *e.g.*, by Anitescu et al. (2012) and Stein et al. (2013). In this paper we consider the use of first-order methods for nonlinear optimization that, at each iteration, use both gradient and log-likelihood evaluations to find a local optimum. This allows for the treatment of constraints if desired, though we note the methods here are equally applicable to the unconstrained case.

The log-likelihood $\ell(\theta)$ and the components of its gradient each consist of a number of terms that are traditionally computationally-intensive to evaluate. For example, using the Cholesky decomposition of the covariance matrix Σ admits computation of the quadratic forms $z^T\Sigma^{-1}z$ and $z^T\Sigma^{-1}\Sigma_i\Sigma^{-1}$ as well as the log-determinant $\log|\Sigma|$ and trace $\operatorname{Tr}(\Sigma^{-1}\Sigma_i)$, but the asymptotic computation and storage costs are $O(n^3)$ and $O(n^2)$ respectively. Thus, for datasets with a large number of observations the time and memory requirements become prohibitively expensive, which necessitates alternative approaches.

1.1 Our Method

The contribution of this paper is a framework for efficiently finding $\hat{\theta}_{\mathrm{MLE}}$ using hierarchical matrix representations developed in the scientific computing and numerical linear algebra communities. In particular, we propose a two-part scheme wherein Σ and Σ_i are first factored in a matrix-free fashion to obtain fast hierarchical factorizations that can be used to evaluate $\ell(\theta)$ and the first term of g_i for any θ. This overlaps with recent work by Ambikasaran et al. (2016) that addresses the use of hierarchical factorizations to solve with and compute determinants of kernelized covariance matrices (see also Khoromskij et al. (2008) and Börm and Garcke (2007) for earlier work on $\mathcal{H}$-matrix techniques for fast computation of Σx in a Gaussian process context). While this piece of the framework alone gives sufficient machinery to quickly evaluate the log-likelihood and thus perform black-box optimization using numerical derivatives, such techniques depend strongly on the local smoothness of the objective about the current iterate and can suffer from conditioning issues (see Section 5.4 for a relatively benign example). Therefore, central to our framework is the computation of the full gradient of the log-likelihood, which requires treatment of the second term of g_i involving the trace of $\Sigma^{-1}\Sigma_i$. Combining hierarchical factorizations of Σ and Σ_i to obtain an operator that can be applied quickly to a vector, we leverage an algorithm for explicitly constructing a hierarchical matrix representation of a black-box linear operator from which the trace can then be extracted. Combining these two tools and given a set of observations and appropriate kernel function, our framework allows for fast evaluation of the log-likelihood and its gradient for use in any black-box optimization package (*e.g.*, MATLAB's `fminunc` or R's `optim`).

There are many different hierarchical matrix formats and efficient algorithms for their construction. For all such formats the key broad assumption is that, given a hierarchical spatial partitioning of the domain and a matrix A whose indices correspond to points in the domain, most of the off-diagonal blocks of A as induced by the partitioning are numerically low-rank and thus compressible. For example, due to the assumption that the covariance kernel k is smoothly decaying, off-diagonal blocks of Σ and Σ_i corresponding to observations from distinct subdomains have many small singular values, with exact bounds on the numerical rank for different subdomains depending on the assumed structure of the kernel. In the simplest form that we detail in this paper, our framework employs the *recursive skeletonization factorization* as described by Ho and Ying (2015) as the scheme of choice for obtaining approximate factorizations of Σ and Σ_i accurate in operator norm to specified tolerance ϵ. For the trace terms, we feed these factorizations to a simplified variant of the matrix peeling algorithm of Lin et al. (2011) based on weakly-admissible hierarchical rank structure. Through this, we obtain an efficient method for evaluating (1) and (2)—and, ultimately, finding $\hat{\theta}_{\mathrm{MLE}}$—with high and controllable accuracy.

While the same techniques used in this paper apply to observations in d-dimensions for general d, the complexity analysis is different due to the different properties of kernel matrices in different dimensions. For example, applying these methods in the case $d = 1$ is essentially optimal in the sense that the ranks of low-rank blocks do not grow significantly with the number of observations and a scheme with essentially linear complexity in the number of observations is attained. We direct the reader to Ambikasaran et al. (2016) for extensive numerical examples of factorizing kernel matries in this case. In contrast, the observed rank-growth for $d = 3$ is in general much larger and leads to greater asymptotic complexities. In the remainder of this paper, we focus on the case $d = 2$.

1.2 Alternative Approaches

A number of methods exist in the literature for working with the kernel matrices involved in evaluating the log-likelihood and its gradient. To decrease apply, solve, and storage costs, the covariance matrix can be replaced with a sparser "tapered" approximant as described by Furrer et al. (2006), wherein the desired covariance kernel is windowed with a compactly-supported tapering function to attain sparsity. Of course, the computational benefit of tapering depends on the sparsity of the resulting approximant, which is limited by the desired accuracy if the correlation length of the kernel is not naturally small.

If the covariance matrix decomposes naturally into the sum of a diagonal and a numerically low-rank matrix then such a decomposition can be quite efficient for computation (see Cressie and Johannesson, 2008), but this representation is too simple for the applications and kernel functions we consider. Notably, Sang and Huang (2012) show that combining a low-rank model with covariance tapering can sometimes perform better than using either method on its own.

For cases where the underlying process is stationary and the observations lie on a regular grid it is possible to directly approximate the log-likelihood using spectral approximations due to Whittle (1954) or to quickly apply the covariance matrix in Fourier space to solve linear systems with an iterative method. Further, in such cases these systems can be preconditioned using the method of L. et al. (2012) yielding efficient methods for many important problem classes. For irregularly spaced data such as we consider in this paper, however, these approaches do not apply directly. More recently, Castrillón-Candás et al. (2015) demonstrate a combination of multi-level preconditioning and tapering for fast derivative-free restricted maximum likelihood estimation, though gradient computation is not discussed.

An alternative approach to approximating the Gaussian process log-likelihood directly is to explicitly construct and solve a different set of estimating equations that is less computationally cumbersome. For example, the Hutchinson-like sample average approximation (SAA) estimator introduced by Anitescu et al. (2012) and further analyzed by Stein et al. (2013) falls into this category, as do the composite likelihood methods described by, *e.g.*, Vecchia (1988) and Stein et al. (2004). In practice, these methods perform quite well. In our approach, however, we restrict our attention to the true MLE.

1.3 Outline

The remainder of the paper is organized as follows. In Section 2 we review hierarchical matrix structure and outline the recursive skeletonization factorization as discussed by Ho and Ying (2015), which is our hierarchical matrix format of choice for fast MLE computation. In Section 3, we discuss a modification of the matrix peeling algorithm by Lin et al. (2011), which we use to admit fast evaluation of the gradient of the log-likelihood. In Section 5 we present numerical results on a number of test problems and demonstrate the scaling of our approach. Finally, in Section 6, we discuss possible extensions of the current work.

2. Factorization of the Covariance Matrix

To begin, we consider the kernelized covariance matrix $\Sigma = \Sigma(\theta) \in \mathbb{R}^{n\times n}$ with entries $[\Sigma(\theta)]_{ij} = k(x_i, x_j; \theta)$, where the choice of kernel function k typically represents smoothness assumptions on the Gaussian process. Neglecting for a moment the parameter vector θ, common choices of k include the squared-exponential kernel

$$k(x,y) = \exp\left(-\|x-y\|^2\right) \tag{3}$$

and the Matérn family of kernels

$$k(x,y) = \frac{1}{\Gamma(\nu)2^{\nu-1}}\left(\sqrt{2\nu}\cdot\|x-y\|\right)^{\nu} K_{\nu}\left(\sqrt{2\nu}\cdot\|x-y\|\right), \tag{4}$$

where K_ν is the modified second-kind Bessel function and $\Gamma(\nu)$ is the gamma function. To parameterize k, we might for example choose to introduce an unknown variance σ^2 or unknown scale parameter s leading to, in the case of the squared-exponential kernel,

$$k(x, y; \theta) = \sigma^2 \exp\left(-s\|x - y\|^2\right),$$

where $\theta = [\sigma^2, s]$. Because the problem is non-convex, it is difficult to determine the nearness of this point to the true maximizer of the log-likelihood

A key feature of these kernels is that $k(x, y)$ decays relatively smoothly as $\|x - y\| \to \infty$, which gives the covariance matrix Σ a sense of locality. Consider partitioning the domain Ω into the four subdomains Ω_i, $i = 1, ..., 4,$. It has been observed that for our kernels of interest such a partitioning induces the rank structure in Figure 1, wherein certain off-diagonal blocks of Σ corresponding to covariances between observations in distinct subdomains tend to be numerically low-rank and thus compressible. We use the term "low-rank" here to refer to any block that is numerically rank-deficient, and not just those where the rank is bounded essentially independently of the number of observations.

It is important to note that this rank structure includes low-rank blocks at multiple spatial scales. For a concrete example, of a hierarchical matrix format taking advantage of this rank structure, define the set $[n] = \{1, 2, \ldots, n\}$ and let the subdomain $\Omega_{1;i}$ contain degrees of freedom (DOFs) indexed by $\mathcal{I}_{1;i} \subset [n]$ for each $i = 1, \ldots, 4$. Further subdividing $\Omega_{1;1}$ into four subdomains $\Omega_{2;i}$ with corresponding DOFs $\mathcal{I}_{2;i}$ for $i = 1, \ldots, 4$ gives the partitioning of Figure 1. At one spatial scale, we observe that the off-diagonal block $\Sigma(\mathcal{I}_{1;2}, \mathcal{I}_{1;3})$ is numerically low-rank. Similarly, at the next level of the hierarchy, the smaller off-diagonal block $\Sigma(\mathcal{I}_{2;1}, \mathcal{I}_{2;2})$ is also numerically low-rank. Explicitly representing each of these blocks in low-rank form leads to the so-called hierarchical off-diagonal low-rank (HODLR) matrix format that has been used by Ambikasaran et al. (2016) for compression of various families of covariance kernels k for application to Gaussian processes.

The HODLR format is only one of many hierarchical matrix formats from the scientific computing and numerical linear algebra communities, appearing as a simple special case of the $\mathcal{H}$- and $\mathcal{H}^2$-matrices of Hackbusch et al. (see Hackbusch, 1999; Hackbusch and Khoromskij, 2000; Hackbusch and Börm, 2002). This format is particularly simple as at each level it expresses in low-rank form *all* off-diagonal blocks that have not been compressed at previous levels of the hierarchy. Matrices compressible in this way are referred to as *weakly admissible* as coined by Hackbusch et al. (2004), in contrast to *strongly admissible* matrices for which only a subset of off-diagonal blocks at each level are low-rank. Closely related literature includes work on hierarchically semiseparable (HSS) matrices by Xia et al. (2010), Chandrasekaran et al. (2006), and Chandrasekaran et al. (2007), which offers simplified representations with improved runtime, as well as the the similar hierarchically block separable (HBS) format described by Martinsson and Rokhlin (2005) and Gillman et al. (2012).

Remark 1 *The complexity of our framework will depend on the hierarchical matrix format and factorization algorithm used, and the choice of which format to use will depend on the kernel function k. In this paper, we will not review all the above-mentioned formats in detail but will describe which hierarchical rank structure we are assuming at the time we do so.*

For kernelized matrices such as Σ and Σ_i where the entries are explicitly generated by an underlying kernel function, specific factorization algorithms have been developed to exploit

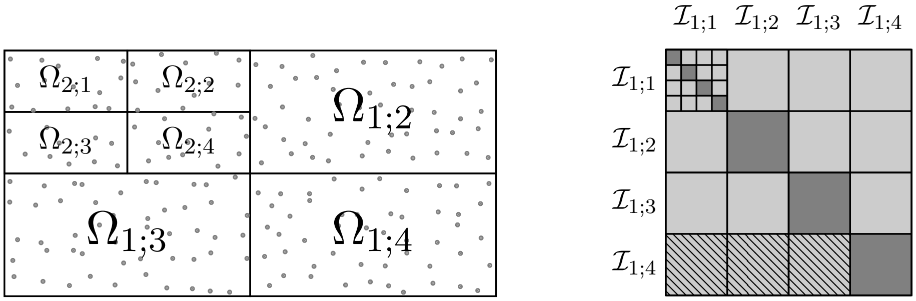

Figure 1: To expose low-rank structure in the matrix Σ, consider the partitioning $\Omega = \cup_{i=1}^{4} \Omega_{1;i}$ (left) where each $\Omega_{1;i}$ contains points indexed by index set $\mathcal{I}_{1;i}$ and is further subdivided in the same manner. Because the kernel function k decays smoothly, the blocks of Σ are rank-structured (right), where dark gray blocks on the diagonal are full-rank and light gray off-diagonal blocks are numerically low-rank. Using a separate low-rank representation for each off-diagonal block leads to the HODLR format. In contrast, the HBS format uses only a single low-rank representation per subdomain at each level to represent off-diagonal blocks. For example, the shaded off-diagonal blocks corresponding to subdomain $\Omega_{1;4}$ are aggregated and compressed together in the HBS format.

this extra structure for added efficiency (see Greengard et al., 2009; Gillman et al., 2012; Ho and Greengard, 2012; Ho and Ying, 2015; Corona et al., 2015). These "skeletonization-based" algorithms, based on the framework introduced by Martinsson and Rokhlin (2005) stemming from observations by Starr and Rokhlin (1994) and Greengard and Rokhlin (1991), construct low-rank representations of certain off-diagonal blocks using the skeletonization process described by Cheng et al. (2005). The *recursive skeletonization factorization* that we use in this paper was first introduced by Ho and Ying (2015) as a multiplicative variant of the method described by Ho and Greengard (2012) and Martinsson and Rokhlin (2005). In the remainder of this section we provide a brief review of the algorithm, but point the reader to Ho and Ying (2015) for a more detailed treatment.

2.1 Block Compression Through Skeletonization

To begin, we build a quadtree on Ω with levels labeled $\ell = 0$ through $\ell = L$ to recursively partition the domain into four subdomains until each leaf-level subdomain contains a constant number of observations independent of n, the total number of observations. The basic intuition of the method is to first compress all blocks of Σ corresponding to covariances between observations in distinct subdomains at the leaf-level, and then to recurse for each higher level of the quadtree.

Consider first a single leaf-level subdomain containing observations indexed by $\mathcal{I} \subset [n]$ and define the complement DOF set $\mathcal{I}^c = [n] \backslash \mathcal{I}$. The algorithm proceeds by compressing the off-diagonal blocks $\Sigma(\mathcal{I}, \mathcal{I}^c)$ and $\Sigma(\mathcal{I}^c, \mathcal{I})$ through the use of an *interpolative decomposition* (ID) (see Cheng et al., 2005).

Definition 2 *Given a matrix $A \in \mathbb{R}^{m \times |\mathcal{I}|}$ with columns indexed by $\mathcal{I}$ and a tolerance $\epsilon > 0$, an ϵ-accurate interpolative decomposition of A is a partitioning of $\mathcal{I}$ into DOF sets associated* *with so-called skeleton columns $\mathcal{S} \subset \mathcal{I}$ and redundant columns $\mathcal{R} = \mathcal{I} \backslash \mathcal{S}$ and a corresponding interpolation matrix $T_{\mathcal{I}}$ such that*

$$A(:, \mathcal{R}) = A(:, \mathcal{S}) T_{\mathcal{I}} + E,$$

where $\|E\|_2 \leq \epsilon \|A\|_2$. In other words, the redundant columns are approximated as a linear combination of the skeleton columns to within the prescribed relative accuracy.

Remark 3 *In contrast to the low-rank blocks used in the HODLR format, note that each block $\Sigma(\mathcal{I}, \mathcal{I}^c)$ corresponds to a larger block row of the matrix Σ, obtained by aggregating the shaded blocks in Figure 1, which can lead to slightly larger ranks. This is the rank structure exploited in the HBS or HSS formats.*

Given an ID of $\Sigma(\mathcal{I}^c, \mathcal{I})$ such that $\Sigma(\mathcal{I}^c, \mathcal{R}) \approx \Sigma(\mathcal{I}^c, \mathcal{S}) T_{\mathcal{I}}$, we have by symmetry that Σ can be written (up to a permutation) as

$$\begin{aligned}
\Sigma &= \left[\begin{array}{cc|c} \Sigma(\mathcal{I}^c, \mathcal{I}^c) & \Sigma(\mathcal{I}^c, \mathcal{S}) & \Sigma(\mathcal{I}^c, \mathcal{R}) \\ \Sigma(\mathcal{S}, \mathcal{I}^c) & \Sigma(\mathcal{S}, \mathcal{S}) & \Sigma(\mathcal{S}, \mathcal{R}) \\ \hline \Sigma(\mathcal{R}, \mathcal{I}^c) & \Sigma(\mathcal{R}, \mathcal{S}) & \Sigma(\mathcal{R}, \mathcal{R}) \end{array} \right] \\
&\approx \left[\begin{array}{cc|c} \Sigma(\mathcal{I}^c, \mathcal{I}^c) & \Sigma(\mathcal{I}^c, \mathcal{S}) & \Sigma(\mathcal{I}^c, \mathcal{S}) T_{\mathcal{I}} \\ \Sigma(\mathcal{S}, \mathcal{I}^c) & \Sigma(\mathcal{S}, \mathcal{S}) & \Sigma(\mathcal{S}, \mathcal{R}) \\ \hline T_{\mathcal{I}}^T \Sigma(\mathcal{S}, \mathcal{I}^c) & \Sigma(\mathcal{R}, \mathcal{S}) & \Sigma(\mathcal{R}, \mathcal{R}) \end{array} \right].
\end{aligned}$$

Using a sequence of block row and column operations, we first eliminate the blocks $\Sigma(\mathcal{I}^c, \mathcal{S}) T_{\mathcal{I}}$ and $T_{\mathcal{I}}^T \Sigma(\mathcal{S}, \mathcal{I}^c)$ and then decouple the bottom-right block through a Schur complement step to obtain

$$\begin{aligned}
\Sigma &\approx L_{\mathcal{I}} \left[\begin{array}{cc|c} \Sigma(\mathcal{I}^c, \mathcal{I}^c) & \Sigma(\mathcal{I}^c, \mathcal{S}) & \\ \Sigma(\mathcal{S}, \mathcal{I}^c) & \Sigma(\mathcal{S}, \mathcal{S}) & D_{\mathcal{S}, \mathcal{R}} \\ \hline & D_{\mathcal{R}, \mathcal{S}} & D_{\mathcal{R}, \mathcal{R}} \end{array} \right] L_{\mathcal{I}}^T \\
&= L_{\mathcal{I}} U_{\mathcal{I}} \left[\begin{array}{cc|c} \Sigma(\mathcal{I}^c, \mathcal{I}^c) & \Sigma(\mathcal{I}^c, \mathcal{S}) & \\ \Sigma(\mathcal{S}, \mathcal{I}^c) & D_{\mathcal{S}, \mathcal{S}} & \\ \hline & & D_{\mathcal{R}, \mathcal{R}} \end{array} \right] U_{\mathcal{I}}^T L_{\mathcal{I}}^T,
\end{aligned}$$

where $L_{\mathcal{I}}$ and $U_{\mathcal{I}}$ are block unit-triangular matrices and the D subblocks are linear combinations of the Σ subblocks.

2.2 The Recursive Skeletonization Factorization

Denoting by $\mathscr{L}_L$ the collection of DOF sets corresponding to subdomains at the leaf-level of the quadtree, we use the skeletonization process of Section 2.1 to compress the corresponding blocks of Σ for each $\mathcal{I} \in \mathscr{L}_L$, yielding

$$\Sigma \approx \left(\prod_{\mathcal{I} \in \mathscr{L}_L} L_{\mathcal{I}} U_{\mathcal{I}} \right) \tilde{\Sigma}_L \left(\prod_{\mathcal{I} \in \mathscr{L}_L} U_{\mathcal{I}}^T L_{\mathcal{I}}^T \right),$$

where the matrix $\tilde{\Sigma}_L$ is such that, for each $\mathcal{I} \in \mathscr{L}_L$ with corresponding skeleton DOFs $\mathcal{S}$ and redundant DOFs $\mathcal{R}$,

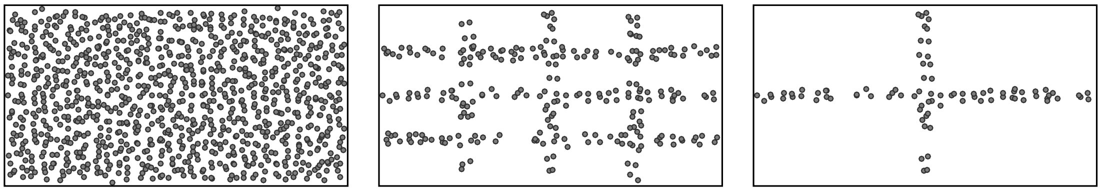

Figure 2: Shown here are the DOFs to be compressed at each level of the recursive skeletonization factorization. At the leaf level (left), all DOFs are involved in skeletonization. At each subsequent level (center, right), only skeleton DOFs from the previous level are involved in further skeletonization. We see that the skeleton DOFs tend to line the boundaries of their corresponding subdomains.

- the block $\tilde{\Sigma}_L(\mathcal{S},\mathcal{S}) = D_{\mathcal{S},\mathcal{S}}$ has been modified,
- the block $\tilde{\Sigma}_L(\mathcal{R},\mathcal{R}) = D_{\mathcal{R},\mathcal{R}}$ has been modified and decoupled from the rest of $\tilde{\Sigma}_L$, and,
- the blocks $\tilde{\Sigma}_L(\mathcal{S},\mathcal{I}^c) = \Sigma(\mathcal{S},\mathcal{I}^c)$ and $\tilde{\Sigma}_L(\mathcal{I}^c,\mathcal{S}) = \Sigma(\mathcal{I}^c,\mathcal{S})$ remain unmodified.

In other words, we have identified and decoupled all redundant DOFs at the leaf level while leaving the blocks of Σ between skeleton DOFs in distinct leaf-level subdomains unchanged.

The recursive skeletonization factorization of Σ is given by repeating this process at each higher level of the quadtree. For example, at level $\ell = L-1$ of the quadtree, each subdomain contains skeleton DOFs corresponding to its four distinct child subdomains in the tree. We thus define the DOFs $\mathcal{I}$ for a subdomain at this level to be the collection of skeleton DOFs of its children and note that, for two distinct subdomains at this level with corresponding DOFs $\mathcal{I}_i$ and $\mathcal{I}_j$, $\tilde{\Sigma}_L(\mathcal{I}_i,\mathcal{I}_j) = \Sigma(\mathcal{I}_i,\mathcal{I}_j)$ and $\tilde{\Sigma}_L(\mathcal{I}_j,\mathcal{I}_i) = \Sigma(\mathcal{I}_j,\mathcal{I}_i)$ are original blocks of the the covariance matrix. Due to the hierarchical block low-rank structure of Σ, these are again compressible through skeletonization, yielding

$$\tilde{\Sigma}_L \approx P_L \left(\prod_{\mathcal{I}\in\mathscr{L}_{L-1}} L_{\mathcal{I}} U_{\mathcal{I}} \right) \tilde{\Sigma}_{L-1} \left(\prod_{\mathcal{I}\in\mathscr{L}_{L-1}} U_{\mathcal{I}}^T L_{\mathcal{I}}^T \right) P_L^T,$$

where P_1 is a global permutation matrix regrouping DOFs to their corresponding subdomains on this level. Proceeding in this fashion level-by-level, we obtain the full recursive skeletonization factorization F of Σ,

$$\begin{aligned} \Sigma &\approx \left[\prod_{\ell=1}^{L} \left(\prod_{\mathcal{I}\in\mathscr{L}_\ell} L_{\mathcal{I}} U_{\mathcal{I}} \right) P_\ell \right] \tilde{\Sigma}_0 \left[\prod_{\ell=1}^{L} \left(\prod_{\mathcal{I}\in\mathscr{L}_\ell} L_{\mathcal{I}} U_{\mathcal{I}} \right) P_\ell \right]^T \\ &= \left[\prod_{\ell=1}^{L} \left(\prod_{\mathcal{I}\in\mathscr{L}_\ell} L_{\mathcal{I}} U_{\mathcal{I}} \right) P_\ell \right] CC^T \left[\prod_{\ell=1}^{L} \left(\prod_{\mathcal{I}\in\mathscr{L}_\ell} L_{\mathcal{I}} U_{\mathcal{I}} \right) P_\ell \right]^T = F, \end{aligned} \tag{5}$$

where $\tilde{\Sigma}_0 \succeq 0$ is block-diagonal and contains all the diagonal blocks corresponding to redundant DOFs at each level. Here, we use the Cholesky decomposition $\tilde{\Sigma}_0 = CC^T$ to factorize the positive definite central matrix $\tilde{\Sigma}_0$, though one could also use, *e.g.*, the LDL^T-decomposition should Σ be indefinite.

2.3 Computational Complexity

To determine the computational cost of the recursive skeletonization factorization, we begin with the cost of an ID. The typical ID algorithm is based on a (strong) rank-revealing QR factorization, meaning that the $m \times |\mathcal{I}|$ ID of Definition 2 has complexity $O(m|\mathcal{I}|^2)$ (see Cheng et al., 2005). This implies that computing an ID of $\Sigma(\mathcal{I}^c, \mathcal{I})$ has complexity $O(n|\mathcal{I}|^2)$, in principle. Using a variant of the so-called "proxy trick" described by Martinsson and Rokhlin (2005), however, this cost can be reduced.

To see this, let $\mathcal{I}_i$ be an index set corresponding to points in subdomain Ω_i (Figure 3). The key to reducing the cost of computing an ID using the proxy trick is noting that, for analytic kernel functions that decay smoothly, it is not necessary to look at all rows to form an ID of $\Sigma(\mathcal{I}_i^c, \mathcal{I}_i)$. Instead, as discussed in detail by Ho and Ying (2015), we can partition $\mathcal{I}_i^c$ into points near Ω_i and points far from Ω_i, denoted $\mathcal{N}_i$ and $\mathcal{F}_i$, respectively. Further, we introduce a small number p of artificial observation points discretizing a region Γ around Ω_i and define the matrix Σ_{prox} corresponding to evaluations of the covariance kernel k between points in Ω_i and points discretizing Γ. With this partitioning, the proxy trick relies on the existence of a well-conditioned matrix B such that

$$\begin{bmatrix} \Sigma(\mathcal{N}_i, \mathcal{I}_i) \\ \Sigma(\mathcal{F}_i, \mathcal{I}_i) \end{bmatrix} \approx B \begin{bmatrix} \Sigma(\mathcal{N}_i, \mathcal{I}_i) \\ \Sigma_{\text{prox}} \end{bmatrix}. \tag{6}$$

Given an ϵ-accurate ID of the right-most matrix above, we see that the same interpolation matrix $T_{\mathcal{I}_i}$ and skeleton DOFs $\mathcal{S}$ give an ID of $\Sigma(\mathcal{I}_i^c, \mathcal{I}_i)$ that is accurate to roughly the same tolerance by our assumption on B. With this trick, we can compute only the smaller decomposition, which brings the ID complexity down from $O(n|\mathcal{I}_i|^2)$ to $O(|\mathcal{I}_i|^3 + |\mathcal{I}_i|^2 p)$. This is an essential acceleration to achieve the desired asymptotic complexity. Note that p here must be taken significantly large such that the row-space of the right-most matrix in (6) contains the row-space of the left-hand side, at least numerically to the specified tolerance. This implies that p need be on the order of the $\epsilon-$rank of $\Sigma(\mathcal{F}_i, \mathcal{I}_i)$, which is typically $O(1)$ or $O(\log n)$ for smooth kernel functions. We will proceed by assuming p is constant with respect to n, which is how we treat p in practice.

Using the proxy trick, we see that the cost of the recursive skeletonization factorization is essentially determined by the number of DOFs interior to each skeletonized subdomain. As seen in Figure 2, the skeleton DOFs tend to line the boundaries of their corresponding subdomains. This can be intuitively understood by noting that, since these DOFs are closest to the neighboring subdomains, they are the most correlated with DOFs in the other subdomains and form a well-conditioned basis for representing the blocks $\Sigma(\mathcal{I}, \mathcal{I}^c)$ and $\Sigma(\mathcal{I}^c, \mathcal{I})$. Letting $|s_\ell|$ denote the average number of skeleton DOFs per subdomain at level ℓ, *i.e.*,

$$|s_\ell| = \frac{1}{|\mathscr{L}_\ell|} \sum_{\substack{\mathcal{I} \in \mathscr{L}_\ell \\ \mathcal{I} = \mathcal{S} \cup \mathcal{R}}} |\mathcal{S}|,$$

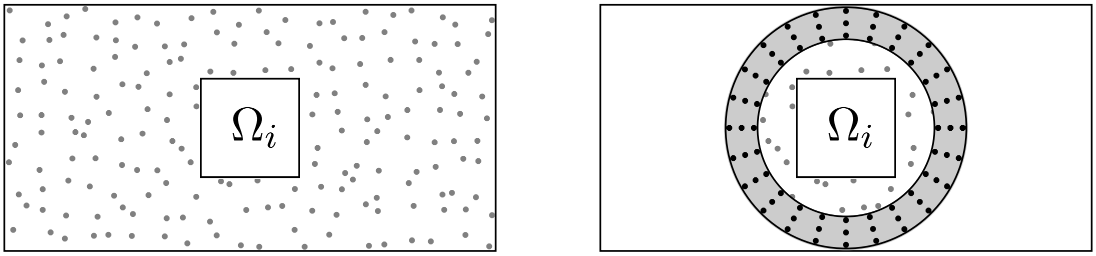

Figure 3: Becuase of the existence of an underlying kernel, computing an interpolative decomposition of the submatrix $\Sigma(\mathcal{I}_i^c, \mathcal{I}_i)$ can be accelerated, where $\mathcal{I}_i \subset [n]$ indexes observations inside the subdomain Ω_i (left). Rather than considering all of $\mathcal{I}_i^c$, the *proxy trick* involves neglecting rows of $\Sigma(\mathcal{I}_i^c, \mathcal{I}_i)$ corresponding to points far from Ω_i, forming an interpolative decomposition by considering only points near Ω_i and additionally a small number of so-called "proxy points" discretizing an annulus around Ω_i.

we have by a geometric observation that $|s_\ell| \sim 2^{-\ell}$ in 2D, *i.e.*, the number of skeleton DOFs per box grows by roughly a factor of two each time we step up a level in the tree. The relatively weak assumptions that lead to this rank growth bound are described in more detail by Ho and Greengard (2012); we do not go into them here. Assuming this bound on rank growth of off-diagonal blocks as we proceed, the computational complexity of the factorization is dominated by the cost of factoring the top level of the quadtree where the number of remaining skeleton DOFs is $|s_0| = O(\sqrt{n})$. From this, it can be shown that the computational complexity of constructing the recursive skeletonization factorization under these assumptions is $O(n^{3/2})$, whereas the storage cost is $O(n \log n)$.

From (5) we see that the application of the factorization F to a vector $x \in \mathbb{R}^n$ simply requires application of the block unit-triangular matrices $L_\mathcal{I}$ and $U_\mathcal{I}$ corresponding to each subdomain at each level as well as the block-diagonal Cholesky factor C of $\tilde{\Sigma}_0$. Further, the inverse of F can be applied by noting that

$$F^{-1} = \left[\prod_{\ell=L}^{1} \left(\prod_{\mathcal{I} \in \mathscr{L}_\ell} P_\ell^T U_\mathcal{I}^{-1} L_\mathcal{I}^{-1} \right) \right]^T C^{-T} C^{-1} \left[\prod_{\ell=L}^{1} \left(\prod_{\mathcal{I} \in \mathscr{L}_\ell} P_\ell^T U_\mathcal{I}^{-1} L_\mathcal{I}^{-1} \right) \right].$$

Additionally, a generalized square root $F^{1/2}$ such that $F = F^{1/2} F^{T/2}$ can be applied (as can its transpose or inverse) by taking

$$F^{1/2} = \left[\prod_{\ell=1}^{L} \left(\prod_{\mathcal{I} \in \mathscr{L}_\ell} L_\mathcal{I} U_\mathcal{I} \right) P_\ell \right] C.$$

Finally, the log-determinant of Σ is given by

$$\log |\Sigma| \approx \log |F| = 2 \log |C|.$$

Table 1: Complexity of the 2D recursive skeletonization factorization

Operation	Complexity
Construct $F \approx \Sigma$	$O\left(n^{3/2}\right)$
Apply F or F^{-1} to a vector	$O(n \log n)$
Apply $F^{1/2}$ or $F^{-1/2}$ to a vector	$O(n \log n)$
Compute $\log \lvert F \rvert$ from F	$O(n)$

The computational complexities for these operations are summarized in Table 1.

By constructing the recursive skeletonization factorizations F of Σ and F_i of Σ_i for $i = 1, \dots, p$, we see that after the $O(n^{3/2})$ initial factorization cost each term in the evaluation of the log-likelihood or its gradient can be computed in linear time except for the product traces $\mathrm{Tr}(\Sigma^{-1}\Sigma_i)$ for $i = 1, \dots, p$. Further, through the approximate generalized square root $F^{1/2}$ of Σ we can quickly sample from the distribution $N(0, \Sigma)$. In the next section, we will describe our approach for fast evaluation of the trace terms.

Remark 4 *While the recursive skeletonization factorization described here exploits the most well-justified rank assumptions on the covariance kernel k, it has been observed numerically that many kernels exhibit additional low-rank structure that can also be systematically employed. In particular, each recursive skeletonization factorization in our framework can be replaced by a closely-related* hierarchical interpolative factorization *(HIF) as introduced by Ho and Ying (2015), which is observed in practice to exhibit better scaling properties and also admits simple log-determinant computation.*

3. Computing the Trace Terms

There are a number of methods for approximating the trace terms $\mathrm{Tr}(\Sigma^{-1}\Sigma_i)$ involved in the computation of each gradient component g_i for $i = 1, \dots, p$. Employing the recursive skeletonization factorizations F of Σ and F_i of Σ_i, we see that the product $\Sigma^{-1}\Sigma_i \approx F^{-1}F_i$ or a symmetrized form with the same trace can be applied to a vector with complexity $O(n \log n)$. Using G to denote this black-box operator, the naïve approach to beat consists of forming $e_i^T G e_i$ for $i = 1, \dots, n$ to extract each diagonal entry and then summing the result. This leaves us with the problem of calculating the trace of a matrix that is only available as a fast black-box operator using fewer that $O(n)$ applications of the operator.

The classical statistical approach is the randomized estimator of Hutchinson (1990), which draws vectors $u_i \in \{-1, 1\}^n$ for $i = 1, \dots, m$ with i.i.d. Rademacher components and computes

$$\mathrm{Tr}(G) \approx \frac{1}{m} \sum_{i=1}^{m} u_i^T G u_i, \tag{7}$$

which is unbiased with variance decaying as $1/m$. For low-accuracy estimates of the trace, the Hutchinson estimator is simple and computationally-efficient, but for higher accuracy it proves computationally infeasible to use the Hutchinson approach because of the slow rate of convergence in m, see Section 5.

Due to the fact that Σ and Σ_i have hierarchical block low-rank structure, it is natural to wonder if the product $\Sigma^{-1}\Sigma_i$ might also have this structure. For example, if Σ is strongly-admissible with hierarchical block low-rank structure then so is Σ^{-1}. Now, if we consider the block decomposition

$$\Sigma^{-1}\Sigma_i = \begin{bmatrix} \Sigma^{-1}(\mathcal{I},\mathcal{I}) & \Sigma^{-1}(\mathcal{I},\mathcal{I}^c) \\ \Sigma^{-1}(\mathcal{I}^c,\mathcal{I}) & \Sigma^{-1}(\mathcal{I}^c,\mathcal{I}^c) \end{bmatrix} \begin{bmatrix} \Sigma_i(\mathcal{I},\mathcal{I}) & \Sigma_i(\mathcal{I},\mathcal{I}^c) \\ \Sigma_i(\mathcal{I}^c,\mathcal{I}) & \Sigma_i(\mathcal{I}^c,\mathcal{I}^c) \end{bmatrix},$$

we see that the top-right block of $\Sigma^{-1}\Sigma_i$ is $\Sigma^{-1}(\mathcal{I},\mathcal{I})\Sigma_i(\mathcal{I},\mathcal{I}^c)+\Sigma^{-1}(\mathcal{I},\mathcal{I}^c)\Sigma_i(\mathcal{I}^c,\mathcal{I}^c)$, which has rank no greater than the sum of the ranks of $\Sigma_i(\mathcal{I},\mathcal{I}^c)$ and $\Sigma^{-1}(\mathcal{I},\mathcal{I}^c)$. This simple example shows that, at least for the HBS format, low-rank off-diagonal blocks of Σ^{-1} and Σ_i lead to low-rank off-diagonal blocks of the product. This encourages us to again follow an approach based on exploiting hierarchical structure.

Remark 5 *The example above shows that the product of two HBS (HSS) matrices is again similarly rank-structured, and can be simply extended to the HODLR case. This is a consequence of the fact that these formats are all based on the weak admissibility criterion: the block $G(\mathcal{I}_i,\mathcal{I}_j)$ of a weakly-admissible matrix G is low-rank for distinct DOF sets $\mathcal{I}_i$ and $\mathcal{I}_j$ corresponding to boxes on the same level of the spatial hierarchy. For strongly-admissible matrices wherein only a subset of the blocks $G(\mathcal{I}_i,\mathcal{I}_j)$ are low-rank a similar result on certain off-diagonal blocks of the product holds, though the number of full-rank blocks is in general greater for the product than for the factors.*

To take advantage of the hierarchical structure of the product $\Sigma^{-1}\Sigma_i$, we use the matrix peeling algorithm of Lin et al. (2011) for constructing an explicit $\mathcal{H}$-matric representation of a fast black-box operator G. At a high-level, the method proceeds by applying the operator G to random vectors drawn with a specific sparsity structure to construct an approximate representation of the off-diagonal blocks at each level. We recursively perform low-rank compression level-by-level, following the same quadtree hierarchy as in the recursive skeletonization factorization, albeit in a top-down traversal rather than bottom-up. Finally, at the bottom level of the tree, the diagonal blocks corresponding to leaf-level subdomains can be extracted and their traces computed. While the full algorithm is applicable to both the strongly-admissible and weakly-admissible setting, the version of the algorithm we detail here is efficient for the simple weakly-admissible case. We point the reader to Lin et al. (2011) for more details related to the modifications required for strong admissibility.

The use of a randomized method for computing low-rank representations of matrices, which we review below, is integral to the peeling algorithm.

3.1 Randomized Low-Rank Approximations

To begin, suppose that matrix $A \in \mathbb{R}^{m\times m}$ has (numerical) rank r and that we wish to construct an explicit rank-r approximation

$$A \approx U_1 M U_2^T$$

with $U_1 \in \mathbb{R}^{m\times r}$, $U_2 \in \mathbb{R}^{m\times r}$, and $M \in \mathbb{R}^{r\times r}$. In the context of approximating the trace terms, for example, A will be an off-diagonal block of $\Sigma^{-1}\Sigma_i$ or perhaps of a related symmetrized form with the same trace. Here we provide an overview of an algorithm that accomplishes this goal.

We begin by constructing approximations to the column space and row space of the matrix A. Let c be a small integer and suppose $W_1 \in \mathbb{R}^{m\times(r+c)}$ and $W_2 \in \mathbb{R}^{m\times(r+c)}$ are appropriately chosen random matrices, the distribution of which we will discuss later. Following Halko et al. (2011), let U_1 be a well-conditioned basis for the column space of AW_1 and U_2 be a well-conditioned basis for the column space of A^TW_2 constructed using, *e.g.*, column-pivoted QR factorizations. Using the Moore-Penrose pseudoinverse, we obtain a low-rank approximation according to the approach summarized by Lin et al. (2011) via

$$A \approx U_1 \left[(W_2^TU_1)^\dagger(W_2^TAW_1)(U_2^TW_1)^\dagger\right] U_2^T = U_1MU_2^T. \tag{8}$$

Perhaps surprisingly, with an appropriate choice of W_1 and W_2 it is the case that with high probability this approximation is *near-optimal*, in the sense that

$$\|A - U_1MU_2^T\|_2 \leq \alpha(n,c)\sigma_{r+1}(A),$$

where $\alpha(n,c)$ is a small factor dependent on n and c. Further, the approximation process can be monitored and controlled adaptively to ensure a target desired accuracy (see Halko et al., 2011).

It remains to discuss the choice of distribution for W_1 and W_2. The most common and straightforward choice is for both to have i.i.d. $N(0,1)$ entries, which guarantees the strongest analytical error bounds and highest success probability. Under this choice, one can show that the algorithm as stated takes $O(T_{\text{apply}}r + nr^2 + r^3)$, where T_{apply} is the complexity of applying A to a vector. From a performance standpoint, however, it can be desirable to choose other random matrices such as the subsampled randomized discrete Fourier transform (see Rokhlin and Tygert, 2008; Halko et al., 2011) or the subsampled randomized Hadamard transform (see Tropp, 2011). In our examples, we find that the randomized low-rank decompositions do not bottleneck our algorithm, so we restrict our attention to i.i.d. Gaussian entries for simplicity.

3.2 Matrix Peeling for Weakly-Admissible Matrices

In what follows, we assume that G is symmetric, since if the trace of nonsymmetric G is required we can always instead consider a symmetrized form with the same trace such as $\frac{1}{2}(G + G^T)$. Further, we will assume that the numerical ranks of the off-diagonal blocks to a specified tolerance ϵ_{peel} are known *a priori* at each level, such that off-diagonal blocks of G at level ℓ have rank at most r_ℓ. In practice, an adaptive procedure is used to find the ranks. Finally, we will describe the algorithm as though the quadtree representing the hierarchical partitioning of space is *perfect*, *i.e.*, every subdomain is subdivided into four child subdomains at every level, though this is only for simplicity of exposition.

To begin, at the top level, $\ell = 1$, the domain is partitioned into four subdomains $\Omega_i^{(1)}$ with corresponding index sets $\mathcal{I}_{1;i}$, $i = 1,\ldots,4$ as in the left side of Figure 4. To simplify notation, we follow the style of Lin et al. (2011) and write the block $G\left(\mathcal{I}_{1;i}, \mathcal{I}_{1;j}\right)$ as $G_{1;ij}$. In order to construct randomized low-rank approximations of the off-diagonal blocks at this level, $G_{1;ij}$ for $i \neq j$, we need to find the action of these off-diagonal blocks on random matrices as described in Section 3.1. Ordering the entries of G accordingly, we see that applying G to the block-sparse matrix $[W_{1;1}, 0, 0, 0]^T$ where $W_{1;1}$ is a random matrix of

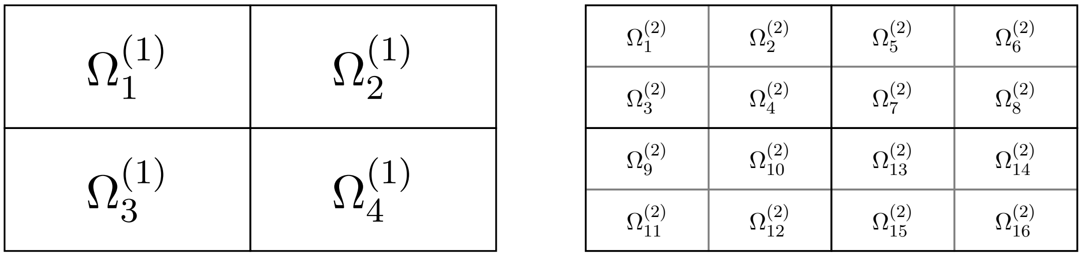

Figure 4: Subdivision of each subdomain into four child subdomains at each level using the labeling scheme of our method leads to subdomain $\Omega_k^{(\ell-1)}$ at level $\ell-1$ dividing into child subdomains $\Omega_{4(k-1)+i}^{(\ell)}$ at level ℓ for $i=1,\ldots,4$.

dimension $|\mathcal{I}_{1;1}| \times (r_1+c)$ gives

$$\begin{bmatrix} G_{1;11} & G_{1;12} & G_{1;13} & G_{1;14} \\ G_{1;21} & G_{1;22} & G_{1;23} & G_{1;24} \\ G_{1;31} & G_{1;32} & G_{1;33} & G_{1;34} \\ G_{1;41} & G_{1;42} & G_{1;43} & G_{1;44} \end{bmatrix} \begin{bmatrix} W_{1;1} \\ \\ \\ \\ \end{bmatrix} = \begin{bmatrix} G_{1;11}W_{1;1} \\ G_{1;21}W_{1;1} \\ G_{1;31}W_{1;1} \\ G_{1;41}W_{1;1} \end{bmatrix}.$$

While the top block of the result is unused as it involves a diagonal block of G, the remaining blocks are exactly the matrices $G_{1;i1}W_{1;1}$ for $i \neq 1$ as required by the randomized low-rank approximation scheme. Repeating the above process by applying G to matrices of the form $[0, W_{1;2}, 0, 0]^T$, $[0, 0, W_{1;3}, 0]^T$, and $[0, 0, 0, W_{1;4}]^T$, we obtain random samples of the column space and row space of each of the blocks $G_{1;ij}$ with $i \neq j$ using our assumption of symmetry of G.

Using the randomized algorithm to construct rank-r_1 approximations of each of these blocks, we write the approximation of $G_{1;ij}$ as

$$G_{1;ij} \approx \tilde{G}_{1;ij} = U_{1;ij} M_{1;ij} U_{1;ji}^T,$$

where the approximation is accurate to the specified tolerance ϵ with high probability. Collecting all of these approximations, we obtain the larger matrix $G^{(1)}$ where we note that

$$G - G^{(1)} = G - \begin{bmatrix} & \tilde{G}_{1;12} & \tilde{G}_{1;13} & \tilde{G}_{1;14} \\ \tilde{G}_{1;21} & & \tilde{G}_{1;23} & \tilde{G}_{1;24} \\ \tilde{G}_{1;31} & \tilde{G}_{1;32} & & \tilde{G}_{1;34} \\ \tilde{G}_{1;41} & \tilde{G}_{1;42} & \tilde{G}_{1;43} & \end{bmatrix} \approx \begin{bmatrix} G_{1;11} & & & \\ & G_{1;22} & & \\ & & G_{1;33} & \\ & & & G_{1;44} \end{bmatrix}.$$

In other words, we have approximated the off-diagonal blocks at this level to high-accuracy and used the result to obtain a fast operator $G - G^{(1)}$ such that the subdomains Ω_i, $i = 1, \ldots, 4$ are decoupled. Naturally, the next step of the peeling algorithm is to recurse on the diagonal subblocks $G_{1;ii}$. Partitioning each subdomain Ω_i at level $\ell = 1$ into four child subdomains at level $\ell = 2$ using the quadtree structure and renumbering blocks accordingly, we write

$$G_{1;11} = \begin{bmatrix} G_{2;11} & G_{2;12} & G_{2;13} & G_{2;14} \\ G_{2;21} & G_{2;22} & G_{2;23} & G_{2;24} \\ G_{2;31} & G_{2;32} & G_{2;33} & G_{2;34} \\ G_{2;41} & G_{2;42} & G_{2;43} & G_{2;44} \end{bmatrix}, G_{1;22} = \begin{bmatrix} G_{2;55} & G_{2;56} & G_{2;57} & G_{2;58} \\ G_{2;65} & G_{2;66} & G_{2;67} & G_{2;68} \\ G_{2;75} & G_{2;76} & G_{2;77} & G_{2;78} \\ G_{2;85} & G_{2;86} & G_{2;87} & G_{2;88} \end{bmatrix},$$

and so on for $G_{1;33}$ and $G_{1;44}$. The peeling process can then be repeated for each of these subblocks. We formalize the algorithm in the next few subsections.

3.2.1 Level 1

At the first level, we break the domain Ω into four subdomains $\Omega_i^{(1)}$, $i = 1, \ldots, 4$, as discussed above. For each $i = 1, \ldots, 4$, we construct a block matrix $R_{1;i}$ such that $R_{1;i}(\mathcal{I}_{1;i}, :) = W_{1;i}$, a random matrix of size $|\mathcal{I}_{1;i}| \times (r_1 + c)$, and $R_{1;i} = 0$ otherwise. We apply G to $R_{1;i}$ and extract the subblocks $G_{1;ji}W_{1;i}$ for $j \neq i$.

For each off-diagonal subblock $G_{1;ij}$ with $i \neq j$ we use the randomized low-rank approximation algorithm to construct the rank-r_1 approximant

$$\tilde{G}_{1;ij} = U_{1;ij} M_{1;ij} U_{1;ji}^T.$$

We define the matrix of off-diagonal approximants at this level to be $G^{(1)}$, which has blocks of zeros on the diagonal, *i.e.*,

$$G^{(1)}\left(\mathcal{I}_{1;i}, \mathcal{I}_{1;j}\right) = \begin{cases} \tilde{G}_{1;ij} & i \neq j, \\ 0 & \text{else.} \end{cases}$$

We note that this matrix need not be explicitly constructed.

3.2.2 Level $\ell > 1$

At level $\ell > 1$, we take each subdomain $\Omega_k^{(\ell-1)}$ from level $\ell - 1$ and divide it into four child subdomains $\Omega_{4(k-1)+i}^{(\ell)}$, $i = 1, \ldots, 4$, as in Figure 4. Note that, under this numbering, two subdomains $\Omega_i^{(\ell)}$ and $\Omega_j^{(\ell)}$ share a parent subdomain if and only if $\lfloor (i-1)/4 \rfloor = \lfloor (j-1)/4 \rfloor$. For each $i = 1, \ldots, 4$, we construct a block matrix $R_{\ell;i}$ defined such that

$$R_{\ell;i}(\mathcal{I}_{\ell;j}, :) = \begin{cases} W_{\ell;j} & j \equiv i \pmod 4, \\ 0 & \text{else,} \end{cases}$$

that is, $R_{\ell;i}$ has $4^{\ell-1}$ non-zero blocks each corresponding to a subdomain at level ℓ with a unique parent at level $\ell - 1$. Here, $W_{\ell;j}$ is a random matrix of size $|\mathcal{I}_{\ell;j}| \times (r_\ell + c)$.

We apply the operator $G - \sum_{k=1}^{\ell-1} G^{(k)}$ to $R_{\ell;i}$ and extract the subblocks $G_{\ell;kj}W_{\ell;j}$ for each k and j such that $k \neq j$ and $\lfloor (k-1)/4 \rfloor = \lfloor (j-1)/4 \rfloor$. Repeating this for each $i = 1, \ldots, 4$ gives us randomized samples of the row and column spaces of each off-diagonal subblock at this level.

For each off-diagonal subblock $G_{\ell;ij}$ such that $i \neq j$ and $\lfloor (i-1)/4 \rfloor = \lfloor (j-1)/4 \rfloor$, we use the randomized low-rank approximation algorithm to construct the rank-r_ℓ approximant

$$\tilde{G}_{\ell;ij} = U_{\ell;ij} M_{\ell;ij} U_{\ell;ji}^T.$$

We define the matrix of off-diagonal approximants at this level to be $G^{(\ell)}$, which has blocks of zeros on the diagonal, *i.e.*,

$$G^{(\ell)}\left(\mathcal{I}_{\ell;i}, \mathcal{I}_{\ell;j}\right) = \begin{cases} \tilde{G}_{\ell;ij} & i \neq j \text{ and } \lfloor (i-1)/4 \rfloor = \lfloor (j-1)/4 \rfloor, \\ 0 & \text{else.} \end{cases}$$

Again, this matrix need not be explicitly constructed.

3.2.3 Extracting the Diagonal

At the bottom level of the quadtree, the operator $G - \sum_{k=1}^{L} G^{(k)}$ is (approximately) block-diagonal and each diagonal block is of a constant size independent of n as discussed in Section 2.1. Let the maximum number of observations in a subdomain at this level $\Omega_i^{(L)}$ be n_L, *i.e.*, with $n_{L;i} = |\mathcal{I}_{L;i}|$, define

$$n_L = \max_i n_{L;i}.$$

We construct a block matrix E with size $n \times n_L$ such that, with $I_{n_{L;i} \times n_{L;i}}$ as the identity matrix,

$$E\left(\mathcal{I}_{L;i}, [n_{L;i}]\right) = I_{n_{L;i} \times n_{L;i}}$$

for each i. Applying the operator $G - \sum_{k=1}^{L} G^{(k)}$ to E and denoting the result by $H \in \mathbb{R}^{n \times n_L}$, we find that

$$H\left(\mathcal{I}_{L;i}, [n_{L;i}]\right) \approx G_{L;ii}$$

for each $i = 1, \ldots, 4^L$. We can then obtain the trace of G using the relation

$$\mathrm{Tr}(G) = \sum_{i=1}^{4^L} \mathrm{Tr}(G_{L;ii}),$$

where we can approximate the trace of $\mathrm{Tr}(G_{L;ii})$ by the trace of the corresponding sub-block of H with approximation error on the order of the tolerance used for the low-rank approximation at all levels of the algorithm.

Remark 6 *When using the peeling algorithm to construct an approximate trace of the operator G, it is important to note that we do not have direct control of the relative error of the trace approximation when the blocks of G are only numerically low-rank and not truly low-rank. This is because cancellation of terms can lead to a matrix with diagonal entries with large absolute value having a small trace. In practice, however, our numerical results in Section 5 show excellent agreement between the approximate trace and true trace.*

3.3 Computational Complexity

In the algorithm described above we find that there are two key steps at each level: applying the operator $G - \sum_{k=1}^{\ell-1} G^{(k)}$ and forming the low-rank factorizations $\tilde{G}_{\ell;ij}$. We analyze the algorithm overall by determining the complexity level-by-level.

Let the cost of applying G to a vector be T_{apply} and assume that the number of observations in a box of the quadtree at level ℓ is $O(4^{-\ell}n)$, *i.e.*, that the observations are pseudo-uniformly distributed in 2D space. At the first level, we see that applying G to the four block-sparse matrices costs $O(T_{\text{apply}}r_1)$ and each randomized factorization costs $O(4^{-1}nr_1^2)$, leading to an overall cost for level 1 of $O(T_{\text{apply}}r_1 + nr_1^2)$.

At level $\ell > 1$, we again must apply G to four block-sparse matrices $R_{\ell;i}$ for $i = 1, \ldots, 4$ with cost $O(T_{\text{apply}}r_\ell)$. Further, the partial $\mathcal{H}-$matrix $\sum_{k=1}^{\ell-1} G^{(k)}$ must be applied to these matrices as well, with complexity bounded by $O(nr_1 r_\ell \log n)$ assuming that the largest rank of off-diagonal blocks is r_1. Finally, each randomized factorization costs $O(4^{-\ell}nr_\ell^2)$ and there are $O(4^\ell)$ blocks to compress at this level, so the overall cost for level ℓ is $O(T_{\text{apply}}r_\ell + nr_1 r_\ell \log n + nr_\ell^2)$. Note that at level $\ell = L$, we must additionally extract the diagonal blocks, but by the assumption that the quadtree is subdivided until $n_L = O(1)$ this does not increase the asymptotic cost.

Summing the complexity for each level, we see that the cost of extracting the trace using the weak form of the peeling algorithm is bounded by

$$T_{\text{peel}} = \sum_{\ell=1}^{L} O(T_{\text{apply}}r_\ell + nr_1 r_\ell \log n). \tag{9}$$

When the underlying matrix is truly weakly-admissible we have $r_\ell = O(1)$ for all ℓ, making the computational complexity of weak peeling $\tilde{O}(T_{\text{apply}} + n)$, where we use the so-called "soft-O" notation from theoretical computer science to suppress logarithmic dependence on n. The squared-exponential kernel of (3) is an example kernel with exceedingly nice rank growth of all off-diagonal blocks of Σ (see Baxter and Roussos, 2002). In this case, peeling Σ itself using its recursive skeletonization factorization results in $\tilde{O}(n)$ complexity for peeling both in terms of time and memory. While we are not interested in peeling with $G = \Sigma$ but rather with $G = \Sigma^{-1}\Sigma_i$, we observe in practice that the same complexity holds for the product as well, *i.e.*, the product $\Sigma^{-1}\Sigma_i$ seems to also obey weak admissibility.

Unfortunately, many real matrices G of interest decidedly do not exhibit true weak admissibility in the sense that not all off-diagonal blocks have rank $O(1)$. For example, our experiments with the Matèrn kernel of (4) show that a constant number of off-diagonal blocks at each level of the hierarchy exhibit ranks bounded only as $r_\ell = O(2^{-\ell}\sqrt{n})$, which coincides with the argument for rank growth in the recursive skeletonization factorization in Section 2.3. Since the rank of off-diagonal blocks of the product $G = \Sigma^{-1}\Sigma_i$ cannot be expected to be lower than the rank of off-diagonal blocks of the factors, we see that for such kernels our theoretical asymptotic time complexity is $\tilde{O}(n^2)$ and the storage complexity is $\tilde{O}(n^{3/2})$. In practice, the standard form of the peeling algorithm as described by Lin et al. (2011) that uses the full generality of strong admissibility can be employed to remedy such rank growth by explicitly avoiding compression of off-diagonal blocks that are not sufficiently low-rank. However, strong peeling exhibits a larger constant factor. Using the modifications described in that paper, the complexity of peeling follows the same bound as (9) but with the ranks r_ℓ referring to a bound on the ranks of only those blocks that are compressed in the strongly-admissible hierarchical format.

We summarize our complexity results in Table 2. Note that, while these results were derived on the assumption that $n_\ell = O(4^{-\ell}n)$, *i.e.*, a quasi-uniform distribution of obser-

Table 2: Complexity of the the peeling algorithm using the recursive skeletonization factorization

Rank r_ℓ of off-diagonal blocks of G at level ℓ	Time	Storage
$O(1) - O(\log n_\ell)$	$\tilde{O}(n)$	$\tilde{O}(n)$
$O(\sqrt{n_\ell})$	$\tilde{O}\left(n^2\right)$	$\tilde{O}(n^{3/2})$

vations and a perfect quadtree decomposition of space, in practice observations that are distributed in a different fashion exhibit similar behavior.

4. Summary of MLE Framework

In Algorithm 4.1 we briefly summarize our complete framework for computing the log-likelihood and gradient, which can be used inside of any first-order optimization routine for Gaussian process maximum likelihood estimation. As mentioned previously, the approach is flexible and does not rely on the specific hierarchical factorization used (*e.g.*, the recursive skeletonization factorization or the hierarchical interpolative factorization) or the form of peeling used (*i.e.*, the peeling based on weak admissibility described in Sectrion 3 or the form by Lin et al. (2011) based on strong admissibility). Rather, the exact components of the framework should be decided on a case-by-case basis depending on the rank properties of the kernel family.

Algorithm 4.1 Computing the Gaussian process log-likelihood and gradient

```
Given: observation vector $z \in \mathbb{R}^n$, parameter vector $\theta \in \mathbb{R}^p$, peel tolerance $\epsilon_{\text{peel}}$, factorization tolerance $\epsilon_{\text{fact}} < \epsilon_{\text{peel}}$ and covariance kernel family $k(\cdot,\cdot;\theta)$
// Factor $\Sigma$ with hierarchial factorization
$F \leftarrow$ Recursive skeletonization factorization of $\Sigma$ with tolerance $\epsilon_{\text{fact}}$
// Use fast hierarchical solve and log-determinant
$\hat{\ell}(\theta) \leftarrow -\frac{1}{2} z^T F^{-1} z - \frac{1}{2}\log|F| - \frac{1}{2}\log 2\pi \approx \ell(\theta)$
for $i = 1, \ldots, p$ do
    // Factor $\Sigma_i$ with hierarchical factorization
    $F_i \leftarrow$ Recursive skeletonization factorization of $\Sigma_i$ with tolerance $\epsilon_{\text{fact}}$)
    // Compute trace of $\Sigma^{-1}\Sigma_i$ with peeling algorithm
    $t_i \leftarrow$ Trace of peeled operator $\frac{1}{2}(F^{-1}F_i + F_i F^{-1})$ via peeling algorithm with tolerance $\epsilon_{\text{peel}}$
    // Use fast hierarchical apply and solve
    $\hat{g}_i \leftarrow \frac{1}{2} z^T F^{-1} F_i F^{-1} z - \frac{1}{2} t_i \approx g_i$
end for
Output: $\hat{\ell}(\theta)$ and $\hat{g}$
```

5. Numerical Results

To demonstrate the effectiveness of our approach to Gaussian process maximum likelihood estimation, we first test the accuracy and runtime of the peeling-based technique for approximating the trace and then test our full method on two examples using synthetic datasets and one example using a dataset of measurements of ocean surface temperatures.

In our tests we use the FLAM library at `https://github.com/klho/FLAM/` for the recursive skeletonization factorization and a custom implementation of matrix peeling as described in Section 3.2. All numerical results shown were run in MATLAB R2015a on a quad-socket Intel Xeon E5-4640 processor clocked at 2.4 GHz using 1.5 TB of RAM.

5.1 Runtime Scaling of the Peeling Algorithm

To begin, we investigate the numerical performance of the peeling algorithm on synthetic examples. We take the observation locations $\{x_i\}_{i=1}^n$ to be a $\sqrt{n}\times\sqrt{n}$ grid of points uniformly discretizing the square $[0,100]^2 \subset \mathbb{R}^2$. We let the parameter vector $\theta = [\theta_1, \theta_2]$ parameterize the correlation length scale of the process in each coordinate direction, defining the scaled distance

$$\|x-y\|_\theta^2 = \frac{(x_1-y_1)^2}{\theta_1^2} + \frac{(x_2-y_2)^2}{\theta_2^2},$$

where here x_i and y_i are used to denote components of vectors x and y. Using this parameterization and incorporating an additive noise term, the two kernels we test are the squared-exponential kernel of (3),

$$k_{SE}(x,y;\theta) = \exp(-\|x-y\|_\theta^2) + \sigma^2 I, \tag{10}$$

and the Matérn kernel of (3) with parameter $\nu = 3/2$,

$$k_M(x,y;\theta) = (1+\sqrt{3}\|x-y\|_\theta)\exp(-\sqrt{3}\|x-y\|_\theta) + \sigma^2 I. \tag{11}$$

Remark 7 *In both* (10) *and* (11) *the additional term $\sigma^2 I$ can be interpreted as modeling additive white noise with variance σ^2 on top of the base Gaussian process model. In practice, this so-called "nugget effect" is frequently incorporated to account for measurement error or small-scale variation from other sources (see Matheron, 1963) and, further, is numerically necessary for many choices of parameter θ due to exceedingly poor conditioning of many kernel matrices.*

Using a quad-tree decomposition of space defined such that leaf nodes contain only a maximum of $n_{\text{occ}} = 64$ points each, we compute high-accuracy recursive skeletonization factorizations of the matrices Σ, and Σ_1. Combining these we obtain the fast black-box operator

$$G = \frac{1}{2}(\Sigma^{-1}\Sigma_1 + \Sigma_1\Sigma^{-1}) \tag{12}$$

for input to the peeling algorithm to compute the trace to specified tolerance $\epsilon_{\text{peel}} = 1\times10^{-6}$. We choose the parameter vector $\theta = [10,7]$ for these examples as in Figure 5 (left), and set the noise parameter at $\sigma^2 = 1\times10^{-4}$.

Beginning with the squared-exponential kernel, in Table 3 we give runtime results for both the simplified peeling algorithm described in Section 3.2 ("weak peeling") as well as

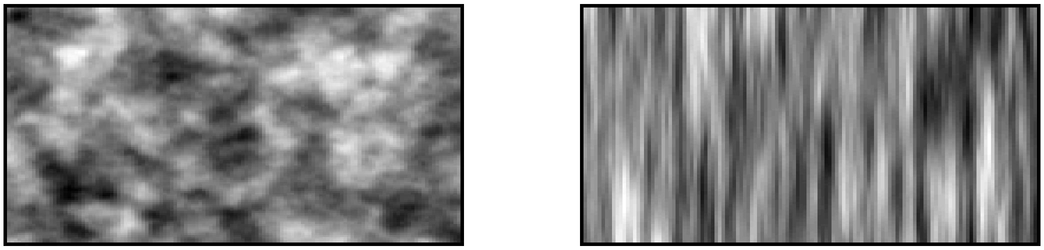

Figure 5: Two different realizations on the domain $[0, 100]^2$ of the Matérn kernel Gaussian process with covariance seen in (11) and noise parameter $\sigma^2 = 0$. In the left figure the parameter vector is $\theta = [10, 7]$ corresponding to a kernel that is relatively close to isotropic. In contrast, in the right figure the parameter vector $\theta = [3, 30]$ exhibits much greater anisotropy.

Table 3: Runtime t_{peel} of the the peeling algorithm with the squared-exponential kernel. Note that we omit the relative error e_{peel} in the estimated trace for our largest example, as the operator was too large to determine the true trace using the naïve approach.

n	$t_{\text{peel,weak}}$ (s)	$e_{\text{peel,weak}}$	$t_{\text{peel,strong}}$ (s)	$e_{\text{peel,strong}}$
64^2	1.5260×10^1	3.1746×10^{-6}	5.0961×10^1	3.9944×10^{-7}
128^2	5.4249×10^1	6.0112×10^{-5}	4.4187×10^2	6.9313×10^{-7}
256^2	2.4327×10^2	2.0725×10^{-5}	2.1141×10^3	3.0438×10^{-6}
512^2	1.0847×10^3	-	9.5325×10^3	-

the full strong admissibility based peeling algorithm of Lin et al. (2011) ("strong peeling"). As can be seen in Figure 6 (left), the runtime of the peeling algorithm with the kernel (10) seems to scale linearly with the number of observations n, regardless of whether weak or strong peeling is used. Further, the relative error in the trace approximation, e_{peel} seems to be well-controlled by the specified tolerance ϵ_{peel}, though the tolerance does not serve as a hard upper bound. Note that we omit the relative error for our largest example, as the operator was too large to determine the true trace using the naïve approach.

In contrast, the results in Table 4 for the Matérn kernel in (11) show different scaling behavior for weak and strong peeling. In Figure 6 (right), we see that the runtime for weak peeling seems to be close to quadratic in the number of observations, which agrees with our analysis from Section 3. Using strong peeling, however, the complexity of peeling scales considerably better, ultimately following the $O(n^{3/2})$ trend line. We see again that the relative trace error is well-controlled by ϵ_{peel} in both cases.

Though the observed scaling behavior of strong peeling is as good or better than that for weak peeling for both kernels, in practice we see that for problems with up to a quarter of a million observations weak peeling has a smaller time-to-solution. As such, in the remainder of our examples we show results using only weak peeling.

Table 4: Runtime t_{peel} of the the peeling algorithm with the Matérn kernel. Note that we omit the relative error e_{peel} in the estimated trace for our largest example, as the operator was too large to determine the true trace using the naïve approach.

n	$t_{\text{peel,weak}}$ (s)	$e_{\text{peel,weak}}$	$t_{\text{peel,strong}}$ (s)	$e_{\text{peel,strong}}$
64^2	6.0313×10^0	5.7294×10^{-8}	2.0632×10^1	4.7802×10^{-10}
128^2	5.3001×10^1	2.4621×10^{-7}	2.2933×10^2	3.3632×10^{-10}
256^2	5.3684×10^2	4.2817×10^{-6}	1.6239×10^3	8.1432×10^{-10}
512^2	7.0688×10^3	-	1.0013×10^4	-

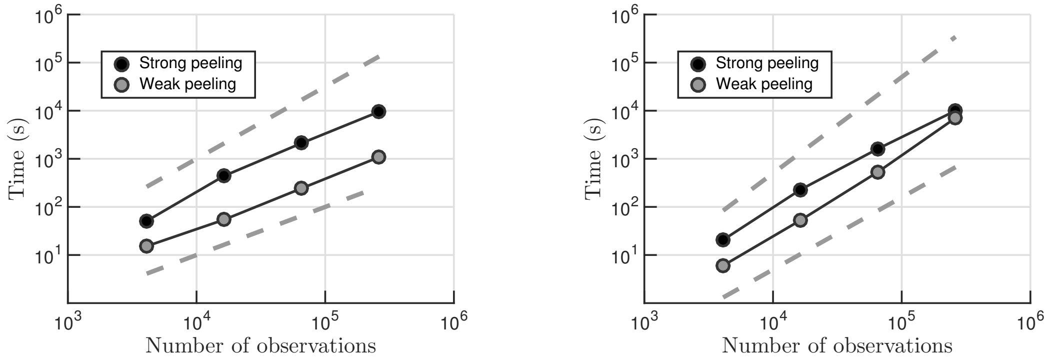

Figure 6: Plotting the runtime of the peeling algorithm as seen in Tables 3 and 4, we see that the squared-exponential kernel and the Matérn kernel exhibit very different behaviors. On the left the runtime of peeling for the squared-exponential kernel is plotted along with a $O(n^{3/2})$ trend line (top) and a $O(n)$ trend line (bottom), exhibiting close to $O(n)$ scaling regardless of whether strong peeling or weak peeling is used. In contrast, on the right the runtime of peeling for the Matérn kernel is plotted along with a $O(n^2)$ trend line (top) and a $O(n^{3/2})$ trend line (bottom). We see that using weak peeling with the Matérn kernel seems to ultimately exhibit $O(n^2)$ scaling, whereas using strong peeling seems to exhibit slightly better than $O(n^{3/2})$ scaling.

5.2 Relative Efficiency of Peeling versus the Hutchinson Estimator

As discussed in Section 3, a common alternative statistical approach for approximating the trace of a matrix G is the estimator of Hutchinson (1990) seen in (7). The aim of this section is to show that for matrices with hierarchical low-rank structure our peeling-based algorithm can be much more efficient when a high-accuracy trace approximation is desired.

As in Section 5.1, we take our observations to be a regular grid discretizing $[0, 100]^2 \subset \mathbb{R}^2$ using the Matérn kernel of (11) with noise $\sigma^2 = 1 \times 10^{-4}$ and parameter vector $\theta = [10, 7]$. We fix the number of observations at $n = 64^2$ and consider how the accuracy of the trace approximation varies with the number of applications of the black-box operator for both weak peeling and the Hutchinson estimator.

Using a high-tolerance recursive skeletonization factorization to construct the black-box operator in (12), we vary the tolerance ϵ_{peel} in the peeling algorithm and plot in Figure 7 the relative error in the trace approximation as a function of both the number of black-box applies and total peeling runtime. Additionally, for the Hutchinson estimator we use the same factorizations to construct the unsymmetric operator $G' = \Sigma^{-1}\Sigma_1$. We plot the same quantities for a given instantiation of the estimator for comparison.

For low-accuracy approximations with relative error on the order of 1×10^{-1} to 1×10^{-3}, we see that the Hutchinson estimator is a competitive alternative to the peeling algorithm for finding the trace. When increased accuracy is desired, however, it is clear that in our examples that the peeling algorithm is the more attractive option. While the Hutchinson estimator has a simple form and is easy to compute, the relatively slow inverse square root convergence means that M in (7) must be taken to be exceedingly large to drive the variance down to reasonable levels, whereas the peeling algorithm is observed to make more economical use of its black-box matrix-vector products. It is worth noting that, for this choice of n, only 4096 applies are needed to explicitly construct all diagonal entries of the operator via application to the identity, though this is not feasible for larger n.

5.3 Synthetic Data Example, Matérn Kernel

With the efficiency of the peeling portion of Algorithm 4.1 verified, we move on to profiling a full objective function and gradient evaluation for the MLE problem for θ. As before, we consider the Matérn kernel of (11) with noise $\sigma^2 = 1 \times 10^{-4}$.

We set the parameter vector at $\theta = [10, 7]$ and again take the observation locations to be a regular $\sqrt{n} \times \sqrt{n}$ grid discretizing the square $[0, 100]^2$. Evaluating $\ell(\theta)$ and g_i for $i = 1, \ldots, 2$ then requires three skeletonization factorizations and two different trace approximations. We investigate the algorithm's performance for two different peeling tolerances ϵ_{peel}, and in each case take the factorization tolerance to be $\epsilon_{\text{fact}} = \frac{1}{1000}\epsilon_{\text{peel}}$. For varying n between 64^2 and 512^2, we measured the runtime of both the factorization portion and peeling portion of Algorithm 4.1. We note that, given the factorizations and peeled trace estimates, the remaininig pieces of Algorithm 4.1 are several orders of magnitude less costly in terms of runtime.

In Figure 8 (left), we plot the total runtime for a single objective function and gradient evaluation for the uniform grid of observations (corresponding data in Table 5). We see from the figure that the runtime seems to scale as roughly $O(n^{3/2})$ with the number of observations; a least-squares fit of the data gives $O(n^{1.6})$. As can be seen in the table, the

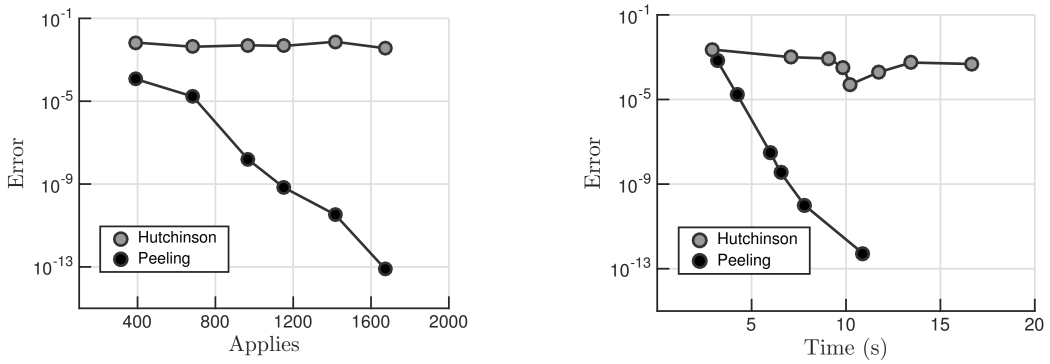

Figure 7: Plotting the relative error in the trace approximation versus the number of applications of the black-box operator, we see in the left figure that the Hutchinson estimator exhibits characteristic inverse square root convergence as dictated by the central limit theorem. In contrast, using the peeling algorithm described in Section 3.2, we see that the same number of black-box applies yields a much improved accuracy, though the rate of convergence depends on the spectra of off-diagonal blocks of the operator. In the right figure, we plot the error of each method versus wall-clock time to establish that the same scaling behavior holds when error is viewed as a function of time-to-solution as well.

Table 5: Runtime for one objective function and gradient evaluation (*i.e.*, the work for a single iteration) on a uniform grid of observations.

ϵ_{peel}	n	t_{fact} (s)	$t_{\text{peel,weak}}$ (s)	t_{total} (s)
1×10^{-6}	64^2	7.3437×10^0	1.2250×10^1	1.9594×10^1
	128^2	4.8615×10^1	1.1957×10^2	1.6818×10^2
	256^2	2.7345×10^2	1.2173×10^3	1.4907×10^3
	512^2	1.4052×10^3	1.3904×10^4	1.5309×10^4
1×10^{-8}	64^2	9.7004×10^0	1.5533×10^1	2.5233×10^1
	128^2	7.0788×10^1	1.6809×10^2	2.3888×10^2
	256^2	4.2785×10^2	1.7640×10^3	2.2918×10^3
	512^2	2.6402×10^3	1.5396×10^4	1.8036×10^4

amount of time spent in calculating the recursive skeletonization factorizations is roughly an order of magnitude less than the time spent in the peeling trace approximation, and, further, scales slightly better than peeling for this example.

5.4 Ocean Data Example, Matérn Kernel

While all examples thus far have used a regular grid of observations, our framework does not rely on this assumption. To complement the examples on gridded observations, we repeat

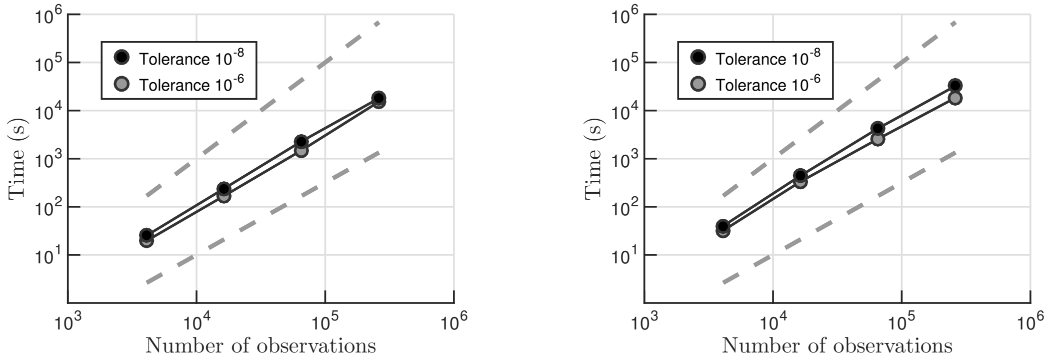

Figure 8: On the left we plot the total runtime of evaluating a single objective function and gradient for the uniform grid example of Section 5.3 as a function of the total number of observations for two different tolerances. The top trend line shows $O(n^2)$ scaling and the bottom shows $O(n^{3/2})$ scaling. On the right we plot the corresponding results for the scattered data of Section 5.4 with the same trend lines. We observe that the scaling in all cases looks like $O(n^{3/2})$.

the same experiment from the previous section with real-world observation locations coming from release 2.5 of the International Comprehensive Ocean-Atmosphere Data Set (ICOADS) obtained from the National Center for Atmospheric Research at `http://rda.ucar.edu/datasets/ds540.0/` (see Woodruff et al., 2011). We subselect from ICOADS a set of sea surface temperatures measured at varying locations in the North Atlantic ocean between the years 2008 and 2014. Restricting the data to observations made in the month of July across all years, we use an artificial mask to simulate occlusion and obtain roughly 300,000 unique observation locations and corresponding sea surface temperature measurements. We use a Mercator projection centered on the observations to project the data to two spatial dimensions.

To perform scaling tests on the cost of an objective function and gradient evaluation according to Algorithm 4.1, we subselect from our full dataset of unique observation locations by drawing observations uniformly at random without replacement. Figure 8 (right) shows the runtime scaling results as a function of the number of observations, with corresponding data in Table 6. We see that the runtime scaling for the scattered observations follows essentially the same scaling behavior as the gridded observations from Section 5.3, with observed complexity between $O(n^{1.5})$ and $O(n^{1.6})$. Again, the skeletonization factorizations take considerably less time than the trace estimation.

As an illustrative example of the full power of Algorithm 4.1 in context, we take a subset of $n = 2^{16}$ scattered observations and realize an instance of a Gaussian process at those locations with true parameter vector $\theta^* = [10, 7]$ to generate the observation vector z. Setting the peel tolerance to $\epsilon_{\text{peel}} = 1 \times 10^{-6}$ and the factorization tolerance to $\epsilon_{\text{fact}} = 1 \times 10^{-9}$, we plugged our approximate log-likelihood and gradient routines into MATLAB's `fminunc` using the quasi-Newton option. Starting from an initial guess of $\theta_0 = [3, 30]$, we found that after 13 iterations (14 calls to Algorithm 4.1) the first-order optimality as measured by the ℓ_∞-norm of the gradient had been reduced by three orders of magnitude,

Table 6: Runtime for one objective function and gradient evaluation (*i.e.*, the work for a single iteration) on scattered observations with locations from ICOADS.

ϵ_{peel}	n	t_{fact} (s)	$t_{\text{peel,weak}}$ (s)	t_{total} (s)
1×10^{-6}	2^{12}	7.8242×10^{0}	2.4086×10^{1}	3.1910×10^{1}
	2^{14}	4.2508×10^{1}	2.9136×10^{2}	3.3387×10^{2}
	2^{16}	8.6008×10^{1}	2.4859×10^{3}	2.5719×10^{3}
	2^{18}	4.8378×10^{2}	1.7947×10^{4}	1.8431×10^{4}
1×10^{-8}	2^{12}	1.0553×10^{1}	2.9230×10^{1}	3.9783×10^{1}
	2^{14}	5.9568×10^{1}	3.8345×10^{2}	4.4302×10^{2}
	2^{16}	2.7216×10^{2}	3.9830×10^{3}	4.2552×10^{3}
	2^{18}	1.0291×10^{3}	3.1516×10^{4}	3.2542×10^{4}

yielding an estimate of $\hat{\theta} = [10.0487, 7.0496]$ after approximately 4.8638×10^4 seconds. While a large percentage of this runtime was spent in the peeling algorithm, we find it worthwhile to note that in this example the use of our gradient approximation proved essential—using finite difference approximations to the gradient led to stagnation at the first iteration, even with a factorization tolerance $\epsilon_{\text{fact}} = 1 \times 10^{-15}$, *i.e.*, at the limits of machine precision. Because the number of iterations to convergence depends on many factors (*e.g.*, the choice of optimization algorithm, how well the data can be modeled by a Gaussian process, and many convergence tolerances depending on the chosen algorithm), we do not find it useful to attempt to profile the full minimization algorithm more extensively than this, but direct the reader instead to the single-iteration results.

To give a sense of the qualitative performance of our method in practice, we mask our set of observations with an artificial occlusion along a fixed longitude band yielding the approximately 150,000 observations in Figure 9 (left). Note that the choice of axis scaling in the Mercator projection is arbitrary; in our convention the horizontal axis spans 90 units and the vertical axis spans 70 units. Choosing the Matérn kernel of (11) with noise paramater $\sigma^2 = 1 \times 10^{-2}$, we use the MATLAB optimization routine `fminunc` to determine the length scale parameters θ_1 and θ_2 for Gaussian process regression on the temperature observations. Because the problem is non-convex, it is difficult to determine a true maximizer of the log-likelihood, but through this process we find the essentially optimal parameter vector $\theta^* = [20.7036, 19.4316]$, at which $\|g\|_\infty / \ell(\theta) \approx 1 \times 10^{-7}$ (*i.e.*, the gradient has infinity norm approximately seven orders of magnitude smaller than the scale of the objective function). Letting the index set $\mathcal{O}$ correspond to the observation locations and $z(\mathcal{O})$ be the observed temperatures, we use this parameter vector to approximate the temperature field $z(\mathcal{A})$ on a fine-grained discretization of this region of the Atlantic ocean via the conditional mean relation

$$\hat{z}(\mathcal{A}) = \bar{z}(\mathcal{O}) + \Sigma(\mathcal{A}, \mathcal{O})[\Sigma(\mathcal{O}, \mathcal{O})]^{-1}(z(\mathcal{O}) - \bar{z}(\mathcal{O})), \tag{13}$$

where $\bar{z}(\mathcal{O}) = \sum_{i \in \mathcal{O}} z(i)$ is the sample mean across all observed temperatures. The estimate (13) can be computed via recursive skeletonization factorizations to a specified tolerance,

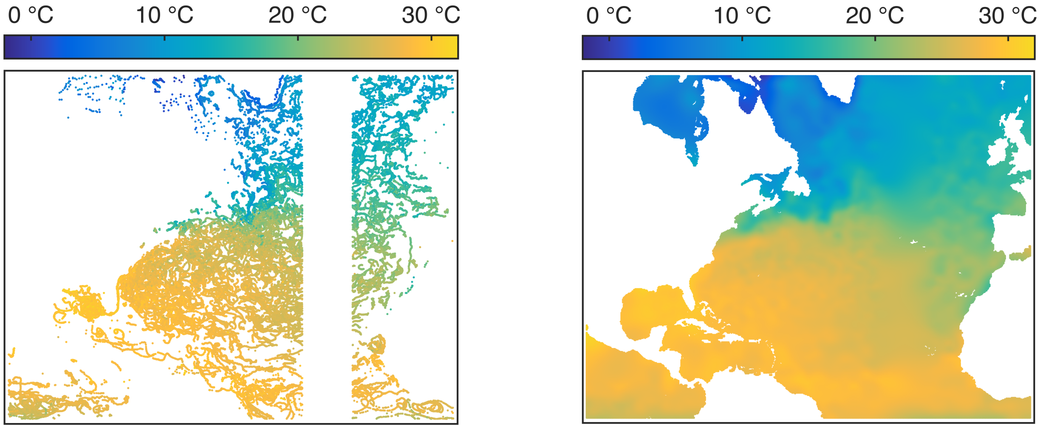

Figure 9: On the left, we show a subselection of approximately 150,000 Atlantic ocean surface temperature measurements from ICOADS, projected to a 2D plane through Mercator projection and then scattered on top of a white background for visualization. Learning the characteristic length scales of a Gaussian process model with the Matérn covariance kernel in (11), we use the learned parameters to find the conditional mean temperatures throughout this region of the Atlantic given the imposed model. The results can be seen in the right visualization, where the estimates are again scattered on a white background and colored by sea surface temperature.

either by adapting the machinery of Section 2 to the unsymmetric case or by factorizing the larger covariance matrix $\Sigma(\mathcal{A} \cup \mathcal{O}, \mathcal{A} \cup \mathcal{O})$ and applying it to an appropriate block-sparse vector. The results of this estimation are shown in Figure 9 (right).

While modeling sea surface temperature as a kernelized Gaussian process is a very crude approximation, we see that the estimated values across the ocean in the right image agree qualitatively with the observed temperatures in the left image. Due to the inclusion of the noise parameter σ, however, the estimates in high-variance regions are slightly smoothed. Overall, this example, while simple, exhibits the clear utility of hierarchical matrices when the data are non-uniformly spaced, which we view as an important advantage of this framework.

6. Discussion

The framework for Gaussian process maximum likelihood estimation presented in this paper and summarized in Algorithm 4.1 provides a straight-forward method of leveraging hierarchical matrix representations from the scientific computing literature for fast computations with kernelized covariance matrices arising in spatial statistics. The general linear algebraic approach to approximating off-diagonal blocks of the covariance matrix to a specified error tolerance by adaptively determining their ranks gives a flexible way of attaining high-precision approximations with reasonable runtime. A further merit to this approach is that it does not rely on having gridded observations or a translation-invariant covariance kernel.

While in this paper we have focused on point estimation for kernelized Gaussian processes, these methods are equally viable for, *e.g.*, sampling from the posterior in a Bayesian setting.

Our numerical results in Section 5 show that our framework scales favorably when applied to our two test cases of the squared-exponential kernel and a Matérn family kernel, leading to runtimes scaling approximately as $O(n^{3/2})$ with n the number of observations. Further, we see that the tolerance parameter ϵ_{peel} controlling the rank of off-diagonal block approximations in the peeling algorithm serves as a good estimate of the order of the error in the ultimate trace approximation as well. In practice, the tolerances ϵ_{peel} and ϵ_{fact} can be dynamically modified during the course of the maximum likelihood process for performance, *e.g.*, one could use relatively low-accuracy approximations during initial iterations of the obtimization routine and slowly decrease the tolerance as the optimization progresses.

While the methods and complexity estimates discussed in this paper relate to the case of two spatial dimensions, they trivially extend to one-dimensional (time-series) data or quasi-two-dimensional data, *i.e.*, observations in three spatial dimensions where the extent in one dimension is of a much smaller scale than the other two. While the same methods apply in principle to truly three-dimensional data, the corresponding computational complexity appears to be bottlenecked by the cost of getting a high-accuracy trace estimate of the matrices $\Sigma^{-1}\Sigma_i$ for $i = 1, \ldots, p$ due to increased rank growth. In fact, even in the two-dimensional case it is clear from Tables 5 and 6 that the most expensive piece of of our framework in practice is determining these traces. One solution is to instead use the hierarchical matrix representations inside of an estimator such as that of Stein et al. (2013), which obviates the need for the trace. For the true MLE, however, future work on efficiently computing this trace to high accuracy is necessary.

Acknowledgments

V.M. is supported by a U.S. Department of Energy Computational Science Graduate Fellowship under grant number DE-FG02-97ER25308. A.D. is partially supported by a National Science Foundation Graduate Research Fellowship under grant number DGE-1147470 and a Simons Graduate Research Assistantship. K.H. is supported by a National Science Foundation Mathematical Sciences Postdoctoral Research Fellowship under grant number DMS-1203554. L.Y. is supported by the National Science Foundation under award DMS-1328230 and DMS-1521830 and the U.S. Department of Energy's Advanced Scientific Computing Research program under award DE-FC02-13ER26134/DE-SC0009409.

The authors thank Matthias Cremon, Eileen Martin, and Sven Schmit for useful discussion on Gaussian process regression, as well as Stanford University and the Stanford Research Computing Center for providing computational resources and support that have contributed to these research results.

References

S. Ambikasaran, D. Foreman-Mackey, L. Greengard, D. W. Hogg, and M. O'Neil. Fast direct methods for Gaussian processes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 38(2):252–265, Feb 2016.

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

B. J. C. Baxter and G. Roussos. A new error estimate of the fast Gauss transform. *SIAM Journal on Scientific Computing*, 24(1):257–259, January 2002.

S. Börm and J. Garcke. Approximating Gaussian processes with $\mathcal{H}^2$-matrices. In *Proceedings of the 18th European Conference on Machine Learning*, pages 42–53. Springer, 2007.

J. E. Castrillón-Candás, M. G. Genton, and R. Yokota. Multi-level restricted maximum likelihood covariance estimation and Kriging for large non-gridded spatial datasets. *Spatial Statistics*, 2015.

S. Chandrasekaran, M. Gu, and T. Pals. A fast ULV decomposition solver for hierarchically semiseparable representations. *SIAM Journal on Matrix Analysis and Applications*, 28 (3):603–622, August 2006.

S. Chandrasekaran, P. Dewilde, M. Gu, W. Lyons, and T. Pals. A fast solver for HSS representations via sparse matrices. *SIAM Journal on Matrix Analysis and Applications*, 29(1):67–81, 2007.

H. Cheng, Z. Gimbutas, P. G. Martinsson, and V. Rokhlin. On the compression of low rank matrices. *SIAM Journal on Scientific Computing*, 26(4):1389–1404, 2005.

E. Corona, P. G. Martinsson, and D. Zorin. An $O(N)$ direct solver for integral equations on the plane. *Applied and Computational Harmonic Analysis*, 38(2):284 – 317, 2015.

N. Cressie and G. Johannesson. Fixed rank Kriging for very large spatial data sets. *Journal of the Royal Statistical Society, Series B*, 70(1):209–226, 2008.

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

A. Gillman, P. M. Young, and P. G. Martinsson. A direct solver with $O(N)$ complexity for integral equations on one-dimensional domains. *Frontiers of Mathematics in China*, 7(2): 217–247, 2012.

L. Greengard and V. Rokhlin. On the numerical solution of two-point boundary value problems. *Communications on Pure and Applied Mathematics*, 44(4):419–452, 1991.

L. Greengard, D. Gueyffier, P. G. Martinsson, and V. Rokhlin. Fast direct solvers for integral equations in complex three-dimensional domains. *Acta Numerica*, 18:243–275, 5 2009.

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.

W. Hackbusch and B.N. Khoromskij. A sparse $\mathcal{H}$-matrix arithmetic. Part II: Application to multi-dimensional problems. *Computing*, 64(1):21–47, Jan. 2000.

W. Hackbusch, N. B. Khoromskij, and R. Kriemann. Hierarchical matrices based on a weak admissibility criterion. *Computing*, 73(3):207–243, 2004.

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

K. L. Ho and L. Greengard. A fast direct solver for structured linear systems by recursive skeletonization. *SIAM Journal on Scientific Computing*, 34(5):A2507–A2532, 2012.

K. L. Ho and L. Ying. Hierarchical interpolative factorization for elliptic operators: Integral equations. *Communications on Pure and Applied Mathematics*, 2015.

M. F. Hutchinson. A stochastic estimator of the trace of the influence matrix for Laplacian smoothing splines. *Communications in Statistics - Simulation and Computation*, 19(2): 433–450, 1990.

B. N. Khoromskij, A. Litvinenko, and H. G. Matthies. Application of hierarchical matrices for computing the Karhunen–Loève expansion. *Computing*, 84(1):49–67, 2008.

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

L. Lin, J. Lu, and L. Ying. Fast construction of hierarchical matrix representation from matrix-vector multiplication. *Journal of Computational Physics*, 230(10):4071–4087, 2011.

P. G. Martinsson and V. Rokhlin. A fast direct solver for boundary integral equations in two dimensions. *Journal of Computational Physics*, 205(1):1–23, May 2005.

G. Matheron. Principles of geostatistics. *Economic geology*, 58(8):1246–1266, 1963.

V. Rokhlin and M. Tygert. A fast randomized algorithm for overdetermined linear least-squares regression. *Proceedings of the National Academy of Sciences*, 105(36):13212–13217, 2008.

H. Sang and J. Z. Huang. A full-scale approximation of covariance functions for large spatial data sets. *Journal of the Royal Statistical Society, Series B*, 74(1):111–132, 2012.

P. Starr and V. Rokhlin. On the numerical solution of two-point boundary value problems II. *Communications on Pure and Applied Mathematics*, 47(8):1117–1159, 1994.

M. L. Stein. *Interpolation of Spatial Data: Some Theory for Kriging.* Springer Series in Statistics. Springer New York, 1999. ISBN 9780387986296.

M. L. Stein, Z. Chi, and L. J. Welty. Approximating likelihoods for large spatial data sets. *Journal of the Royal Statistical Society, Series B*, 66(2):275–296, 2004.

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

J. A. Tropp. Improved analysis of the subsampled randomized Hadamard transform. *Advances in Adaptive Data Analysis*, 03:115–126, 2011.

A. V. Vecchia. Estimation and model identification for continuous spatial processes. *Journal of the Royal Statistical Society, Series B*, 50(2):pp. 297–312, 1988.

P. Whittle. On stationary processes in the plane. *Biometrika*, 41(3/4):pp. 434–449, 1954.

S. D. Woodruff, S. J. Worley, S. J. Lubker, Z. Ji, J. E. Freeman, D. I. Berry, P. Brohan, E. C. Kent, R. W. Reynolds, S. R. Smith, and C. Wilkinson. ICOADS release 2.5: extensions and enhancements to the surface marine meteorological archive. *International Journal of Climatology*, 31(7):951–967, 2011.

J. Xia, S. Chandrasekaran, M. Gu, and X. S. Li. Fast algorithms for hierarchically semiseparable matrices. *Numerical Linear Algebra With Applications*, 17(6):953–976, Dec 2010.