HLIBCov: Parallel Hierarchical Matrix Approximation of Large Covariance Matrices and Likelihoods with Applications in Parameter Identification

Alexander Litvinenko

King Abdullah University of Science and Technology, Thuwal 23955-6900, Saudi Arabia

Abstract. In this article, we demonstrate how to use the very fast and very efficient parallel hierarchical-matrix (H-matrix) library HLIBpro, which can be applied to many tasks that appear in spatial statistics, geostatistics, and kriging. The data structures and algorithms in HLIBpro allow us to improve statistical models by considering many more locations, finer meshes, and more parameters.

We also describe the HLIBCov package, an extension of the HLIBpro library. HLIBCov allows us to approximate large covariance matrices, to maximize likelihood functions, and to compute the matrix inverse, the Cholesky and LU factorizations, and different matrix norms. We show that an approximate Cholesky factorization of a dense $2M \times 2M$ can be computed on a modern multi-core desktop in just a few minutes. Further, we use HLIBCov to simultaneously estimate the unknown parameters of a Matern covariance function, such as the covariance length, variance, nugget, and smoothness parameter, by maximizing the joint Gaussian log-likelihood function. The computational bottleneck here is the expensive linear algebra arithmetic caused by large and dense covariance matrices. Therefore, we approximate the covariance matrices in the $\mathcal{H}$-matrix format with a computational cost of $\mathcal{O}(k^2 n \log^2 n)$ and a storage cost of $\mathcal{O}(kn\log n)$, where the rank k is a small integer (typically $k < 25$), and n is the number of locations on a fairly general mesh. We present an application of HLIBpro and HLIBCov to a synthetic example, where the true values of the unknown parameters are known.

Key words: Computational statistics; parallel hierarchical matrices; large datasets; Matérn covariance; random fields; spatial statistics; HLIB; HLIBpro; Cholesky; matrix determinant; call C++ from R; parameter identification

HLIBCov software library

Program title: HLIBCov

Nature of problem: To approximate large covariance matrices. To perform efficient linear algebra with large covariance matrices on a non-tensor grid. To estimate the unknown parameters (variance, smoothness parameter, and covariance length) of a covariance function by maximizing the joint Gaussian log-likelihood function with a log-linear computational cost and storage.

Software license: HLIBCov (no license), HLIBpro (GPL 2.0)

CiCP scientific software URL:

Distribution format: *.cc and R files via github

Programming language(s): C++ and R

Computer platform: 32 or 64

Operating system: Unix-like operating systems

Compilers: clang

RAM: 4 GB and more (depending on the matrix size)

External routines/libraries: HLIBCov requires HLIBpro and GNU Scientific Library (`https://www.gnu.org/software/gsl/`).

Running time: $\mathcal{O}(k^2 n \log^2 n)/p$ with p number of cores

Restrictions: For multiple-core architectures with shared memory systems

Supplementary material and references: `www.HLIBpro.com` and references therein.

Additional Comments: HLIBpro is a software library that implements parallel algorithms for hierarchical matrices. It is freely available in binary form for academic purposes. HLIBpro algorithms are designed for one, two, and three - dimensional problems.

1 Introduction

HLIBpro is a very fast and efficient parallel $\mathcal{H}$-matrices library. In our previous paper [52], we used the gradient-free optimization method to estimate the unknown parameters of a covariance function using HLIB and HLIBpro. Here, we present new implementations with HLIBPro, such as calling $\mathcal{H}$-matrix routines from an R environment and implementing a gradient-based optimization method, as well as more numerical results regarding the efficiency.

Novelty of this work. In this paper we use parallel hierarchical ($\mathcal{H}$-) matrices to approximate the joint Gaussian log-likelihood function with a computational complexity of $\mathcal{O}(k^2 n \log^2 n)$, where n is the number of measurements, and $k \ll n$ is the maximal $\mathcal{H}$-matrix rank, which defines the quality of the approximation; we compute the maximum likelihood estimates and estimate the unknown parameters of a covariance function; and we provide some examples of calling efficient HLIBPro C++ routines from the R environment. Additionally, we highlight and explain technical details and best practices which are crucial for an efficient parallel implementation.

Parameter estimation and problem settings. We let n be the number of spatial measurements $\boldsymbol{Z}$ located irregularly across a given geographical region at locations $\mathbf{s} := \{\mathbf{s}_1,\ldots,\mathbf{s}_n\} \in \mathbb{R}^d$, $d \geq 1$. We model these data as a realization from a stationary Gaussian spatial random field. Specifically, we let $\boldsymbol{Z} = \{Z(\mathbf{s}_1),\ldots,Z(\mathbf{s}_n)\}^\top$, where $Z(\mathbf{s})$ is a Gaussian random field. Then, we assume that $\boldsymbol{Z}$ has mean zero and a stationary parametric covariance function $C(\mathbf{h};\boldsymbol{\theta}) = \mathrm{cov}\{Z(\mathbf{s}),Z(\mathbf{s}+\mathbf{h})\}$, where $\mathbf{h} \in \mathbb{R}^d$ is a spatial lag vector and $\boldsymbol{\theta} \in \mathbb{R}^q$ is the unknown parameter vector of interest. To infer the unknown parameters $\boldsymbol{\theta}$, we maximize the Gaussian log-likelihood function,

$$\mathcal{L}(\boldsymbol{\theta}) = -\frac{n}{2}\log(2\pi) - \frac{1}{2}\log|\boldsymbol{C}(\boldsymbol{\theta})| - \frac{1}{2}\boldsymbol{Z}^\top \boldsymbol{C}(\boldsymbol{\theta})^{-1}\boldsymbol{Z}, \tag{1.1}$$

where $\boldsymbol{C}(\boldsymbol{\theta})_{ij} = C(\mathbf{s}_i - \mathbf{s}_j;\boldsymbol{\theta})$, $i,j = 1,\ldots,n$. The maximum likelihood estimator of $\boldsymbol{\theta}$ is the value $\widehat{\boldsymbol{\theta}}$ that maximizes (1.1). When the sample size n is large, the evaluation of (1.1) becomes challenging, due to the computation of the quadratic form and the log-determinant of the n-by-n dense covariance matrix $\boldsymbol{C}(\boldsymbol{\theta})$. Indeed, this operation requires $\mathcal{O}(n^2)$ memory and $\mathcal{O}(n^3)$ computational steps. Hence, scalable and efficient methods that can process larger sample sizes are needed.

Definition 1.1. An $\mathcal{H}$-matrix approximation with the maximal rank k of the exact log-likelihood $\mathcal{L}(\boldsymbol{\theta})$ is defined by $\tilde{\mathcal{L}}(\boldsymbol{\theta};k)$:

$$\tilde{\mathcal{L}}(\boldsymbol{\theta};k) = -\frac{n}{2}\log(2\pi) - \sum_{i=1}^{n}\log\{\tilde{\boldsymbol{L}}_{ii}(\boldsymbol{\theta};k)\} - \frac{1}{2}\mathbf{v}(\boldsymbol{\theta})^\top \mathbf{v}(\boldsymbol{\theta}), \tag{1.2}$$

where $\tilde{\boldsymbol{L}}(\boldsymbol{\theta};k)$ is an $\mathcal{H}$-matrix approximation of the Cholesky factor $\boldsymbol{L}(\boldsymbol{\theta})$ with the maximal rank k in the sub-blocks, $\boldsymbol{C}(\boldsymbol{\theta}) = \boldsymbol{L}(\boldsymbol{\theta})\boldsymbol{L}(\boldsymbol{\theta})^\top$, and $\mathbf{v}(\boldsymbol{\theta})$ is the solution of the linear system $\tilde{\boldsymbol{L}}(\boldsymbol{\theta};k)\mathbf{v}(\boldsymbol{\theta}) = \boldsymbol{Z}$.

The argument k in $\tilde{\mathcal{L}}(\boldsymbol{\theta};k)$ indicates that the fixed-rank strategy is used; in other words, the covariance matrix in each sub-block has a rank equal to or smaller than k.

In this case, we cannot say anything about the accuracy in each sub-block (see more in Remark 2.3).

To maximize $\tilde{\mathcal{L}}(\boldsymbol{\theta};k)$ in (1.2), we use the Brent-Dekker method [15, 60] both with and without derivatives. We note that maximization of the log-likelihood function is an ill-posed problem, since even very small perturbations in the covariance matrix $\boldsymbol{C}(\boldsymbol{\theta})$ may result in large perturbations in the log-determinant and the log-likelihood. A possible remedy, which may or may not help, is to take a higher rank k.

Features of the $\mathcal{H}$-matrix approximation. Other advantages of applying the $\mathcal{H}$-matrix technique are the following:

1. The $\mathcal{H}$-matrix class is large, including low-rank and sparse matrix classes;

2. $\boldsymbol{C}(\boldsymbol{\theta})^{-1}$, $\boldsymbol{C}(\boldsymbol{\theta})^{1/2}$, $\det\boldsymbol{C}(\boldsymbol{\theta})$, Cholesky decomposition, the Schur complement, and many others can be computed in the $\mathcal{H}$-matrix format [30];

3. Since the $\mathcal{H}$-matrix technique has been well studied, there are many examples, multiple sequential and parallel implementations and a solid theory already available. Therefore, no specific MPI or OpenMP knowledge is needed;

4. The $\mathcal{H}$-matrix accuracy can be controlled by the rank, k. The full rank gives an exact representation;

5. In contrast to sparse matrices, etc., the $\mathcal{H}$-matrix format preserves the block-structure of the matrix after the Cholesky decomposition and the inverse have been computed, although possibly with a larger rank.

Related work. Stationary covariance functions that have block Toeplitz or block circulant structure can be resolved by the Fast Fourier Transform (FFT) with a computing cost of $\mathcal{O}(n\log n)$ (e.g., [27, 73] and references therein). However, this approach either does not work for data measured at irregularly spaced locations or requires expensive, non-trivial modifications.

Recently, a large amount of research has been devoted to approximating large covariance matrices through, for example, covariance tapering [23, 38, 68], likelihood approximations in both the spatial [72] and spectral [21] domains, latent processes such as Gaussian predictive processes [6] and fixed-rank kriging [17], and Gaussian Markov random-field approximations [22, 45, 65, 66]; see [74] for a review. A matrix-free approach has also been introduced [4]. Each of these methods has strengths and weaknesses [70, 71, 75].

Some other ideas include nearest-neighbor Gaussian process models [18], multiresolution Gaussian process models [58], equivalent kriging [41], multi-level restricted Gaussian maximum likelihood estimators [16], and hierarchical low-rank approximations [37]. Bayesian approaches could be also applied to identify unknown or uncertain parameters [51,55,55,56,59,62–64].

Previous results [30] show that the $\mathcal{H}$-matrix technique is very stable when approximating the covariance matrix itself [3,5,12,34,39,48,67], its inverse [3,10], its Cholesky decomposition [8,9], and the conditional covariance matrix [30,46,47].

Recently, a maximum likelihood estimator for parameter-fitting Gaussian observations with a Matérn covariance matrix was computed via a framework for unstructured observations in two spatial dimensions [35,53]. This allowed the evaluation of the log-likelihood and its gradient to have a computational complexity of $\mathcal{O}(n^{3/2})$, where n was the number of observations. However, the consequences of the approximation on the maximum likelihood estimator were not studied.

In [12], the authors computed the exact solution of a Gaussian process regression by replacing the kernel matrix with a data-sparse approximation, which they called the $\mathcal{H}^2$-matrix technique. The $\mathcal{H}^2$-matrix approximation had a computational complexity and storage cost of $\mathcal{O}(kn)$, where k is the parameter controlling the accuracy of the approximation.

The authors of [69] observed that the structure of shift-invariant kernels changed from low-rank to block-diagonal (without any low-rank structure) when they varied the scale parameter. Based on this observation, the authors proposed a new kernel approximation algorithm, which they called the Memory-Efficient Kernel Approximation. That approximation considered both the low-rank and the clustering structure of the kernel matrix. They also showed that the resulting algorithm outperformed state-of-the-art low-rank kernel approximation methods in terms of speed, approximation error, and memory usage.

In [5], the authors estimated the covariance matrix of a set of normally distributed random vectors. To overcome large numerical issues in the high-dimensional regime, they computed the (dense) inverses of sparse covariance matrices using $\mathcal{H}$-matrices.

In [34], the authors offered methods for the approximation of random fields that rapidly computed separable expansions. In particular, they suggested the pivoted Cholesky method and provided an a posteriori error estimate in the trace norm. Low-rank tensor techniques in kriging were introduced in [49,57], low-rank tensor train data format for the representation of random fields in [20]. The BigQUIC method for a sparse inverse

covariance estimation of a million variables was introduced in [36].

This method could solve ℓ_1-regularized Gaussian Maximum Likelihood Estimation (MLE-) problems with dimensions of one-million. In [67], the authors applied $\mathcal{H}$-matrices to linear inverse problems in large-scale geostatistics. An $\mathcal{H}^2$-matrix technique for solving large-scale stochastic linear inverse problems with applications in subsurface modeling was demonstrated in [3]. From [2], we learned that typical covariance functions can be hierarchically factored into a product of block low-rank updates of the identity matrix, yielding an $\mathcal{O}(n\log n)$ algorithm for inversion. Additionally, the authors demonstrated that this factorization enabled evaluation of the determinant.

Structure of the paper. In Section 2, we introduce the methodology and algorithms. We review the $\mathcal{H}$-matrix technique, and the $\mathcal{H}$-matrix approximations of Matérn covariance functions and Gaussian likelihood functions. In Section 3, we estimate the memory storage and computing costs. In Section 4, we describe the software installation details, procedures of the HLIBCov code, and the algorithm for parameter estimation. The estimation of unknown parameters is reported in Section 5. Best practices are listed in Section 6. We end the paper with a conclusion in Section 7. The auxiliary $\mathcal{H}$-matrix details are provided in the Appendix A.

2 Methodology and algorithms

2.1 Matérn covariance functions

Matérn covariance functions [28] are very widely used in spatial statistics, geostatistics, machine learning, image analysis, and other areas. The Matérn form of spatial correlations was introduced into statistics as a flexible parametric class [54], with one parameter regulating the smoothness of the underlying spatial random field [33].

Matérn functions with different values for the covariance length (top), and smoothness ν (bottom) are shown in Fig. 1. The singularities and the decay of singular values can be determined from the shapes of the Matérn functions. For instance, the singularity is located on the diagonal $x=y$ in Fig. 1. Additionally, with larger values of ν, the covariance function is smoother and the singular values decay faster (convergence graphics are presented in Section 3).

For any two spatial locations $\mathbf{s}$ and $\mathbf{s}'$, where $\mathbf{h}:=\|\mathbf{s}-\mathbf{s}'\|$, the Matérn class of covariance functions is defined as

$$C(h;\boldsymbol{\theta})=\frac{\sigma^2}{2^{\nu-1}\Gamma(\nu)}\left(\frac{\mathbf{h}}{\ell}\right)^{\nu}\mathcal{K}_{\nu}\left(\frac{\mathbf{h}}{\ell}\right), \tag{2.1}$$

where $\boldsymbol{\theta} = (\sigma^2, \ell, \nu)^\top$; σ^2 is the variance; $\nu > 0$ controls the smoothness of the random field, with larger values of ν corresponding to smoother fields; and $\ell > 0$ is a spatial range parameter. Here, $\mathcal{K}_\nu$ denotes a modified Bessel function of the second kind of order ν, and $\Gamma(\cdot)$ denotes the Gamma function. When $\nu = 1/2$, the Matérn covariance function reduces to the exponential covariance model and describes a rough field. The value $\nu = \infty$ corresponds to a Gaussian covariance model that describes a very smooth, infinitely differentiable field. Random fields with a Matérn covariance function are $\lfloor \nu - 1 \rfloor$ times mean square differentiable. The Matérn covariance functions become simple [61]

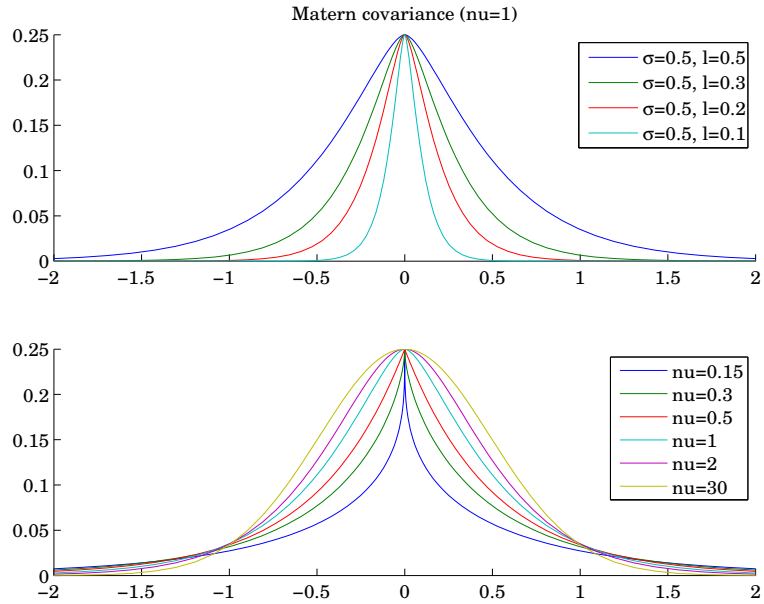

Figure 1: Matérn function for different parameters (computed in sglib [76]).

when $\nu = p + 1/2$, where p is a non-negative integer. In this case, the covariance function is the product of an exponential and a polynomial of order p [1]:

$$C_{\nu=3/2}(\mathbf{h}) = \left(1 + \frac{\sqrt{3}\mathbf{h}}{\ell}\right) \exp\left(-\frac{\sqrt{3}\mathbf{h}}{\ell}\right), \quad C_{\nu=5/2}(\mathbf{h}) = \left(1 + \frac{\sqrt{5}\mathbf{h}}{\ell} + \frac{5\mathbf{h}^2}{3\ell^2}\right) \exp\left(-\frac{\sqrt{5}\mathbf{h}}{\ell}\right).$$

2.2 Hierarchical matrices

Detailed descriptions of hierarchical matrices [25, 29–32, 46] and applications of the $\mathcal{H}$-matrix technique to covariance matrices can be found elsewhere [2, 3, 12, 34, 39, 40, 50].

The $\mathcal{H}$-matrix technique was originally introduced by W. Hackbusch (1999) for the

approximation of stiffness matrices and their inverses coming from partial differential and integral equations [14,25,29]. Briefly, the key idea of the $\mathcal{H}$-matrix technique is to divide the initial matrix into sub-blocks in a specific way, and approximate the sub-blocks, which are far away from the singularity of low-rank matrices. The admissibility conditions criteria are used to divide a given matrix into these sub-blocks (Fig. 4) and to define which sub-blocks can be approximated well by low-rank matrices. There are many different admissibility criteria (see Fig. 2). The typical criteria are the strong, weak, and based on domain decomposition [30]. The user can also develop his own admissibility criteria, which may depend, for instance, on the covariance length.

Figure 2 shows examples of $\mathcal{H}$-matrices with three different admissibility criteria: (left) strong, (middle) domain-decomposition-based, and (right) weak. The numbers inside the sub-blocks show the matrix ranks. The "stairs" inside the sub-blocks demonstrate the decay of the eigenvalues in a log-scale. The dark (or red) blocks indicate fully-populated blocks. The maximal size of the dark blocks (i.e., how deep the hierarchical subdivision into sub-blocks is) is regulated by the parameter "n_{min}", which usually depends on the available DRAM memory, architecture, and the problem size, and typically takes values from the set $\{32,64,128\}$.

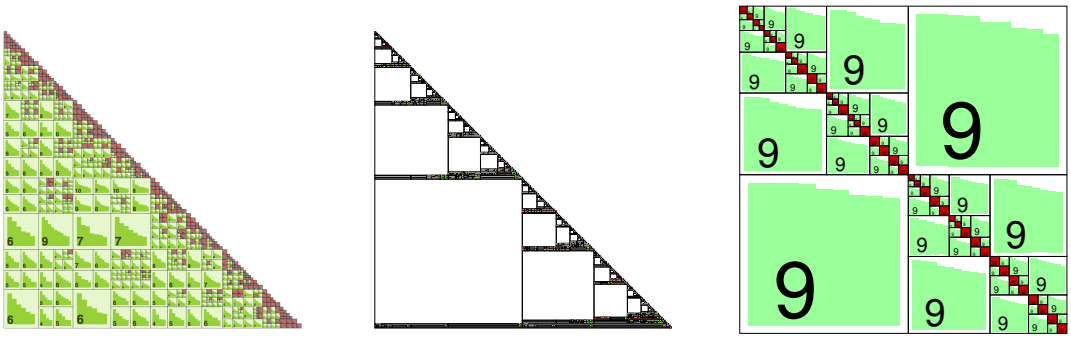

Figure 2: Examples of $\mathcal{H}$-matrices with three different admissibility criteria: (left) standard, (middle) domain-decomposition-based, and (right) weak. Matrices are taken from different applications and illustrate only the diversity of block partitioning.

Remark 2.1. Visualizations of $\mathcal{H}$-matrices larger than $n=10{,}000$ are seldom seen because the eps/pdf/ps files for $n>10{,}000$ are either very large or have bad resolution.

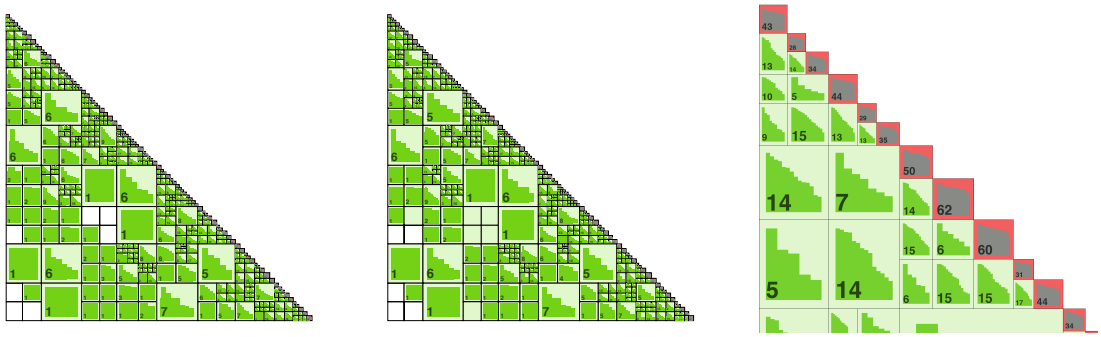

Figure 3: Examples of $\mathcal{H}$-matrix approximations of the exponential covariance matrix (left), its hierarchical Cholesky factor $\tilde{\boldsymbol{L}}$ (middle), and the zoomed upper-left corner of the matrix (right), $n = 4000$, $\ell = 0.09$, $\nu = 0.5$, $\sigma^2 = 1$. The adaptive-rank arithmetic is used, the relative accuracy in each sub-block is $1e-5$. Green blocks indicate low-rank matrices, the number inside of each block indicates the maximal rank in this block. Very small dark-red blocks are dense blocks, in this example they are located only on the diagonal. The "stairs" inside blocks indicate decay of singular values in log-scale.

In general, covariance matrices are dense and, therefore, require $\mathcal{O}(n^2)$ units of memory for storage and $\mathcal{O}(n^2)$ FLOPs for a matrix-vector product. The $\mathcal{H}$-matrix technique is defined as a hierarchical partitioning of a given matrix into sub-blocks (Fig. 4, right), followed by the further approximation of the majority of these sub-block by low-rank matrices (Fig. 3). Figure 3 shows an $\mathcal{H}$-matrix approximation of the exponential covariance matrix (left), its hierarchical Cholesky factor $\tilde{\boldsymbol{L}}$ (middle), and the zoomed upper-left corner of the matrix (right). The matrix size is 4000×4000, the covariance length $\ell = 0.09$, smoothness $\nu = 0.5$, and the variance $\sigma^2 = 1$. The standard admissibility condition, the relative accuracy $1e-5$, and adaptive-rank arithmetic are used in each sub-block. The green blocks indicate low-rank matrices and the number inside of each block indicates the maximal rank used in that block. The very small, dark red blocks are dense blocks, which are computed without approximation. In this example, the dense blocks are located only along the diagonal. The maximal size of the dense blocks is the parameter $n_{\min}$, which is 64. The "stairs" inside the green blocks indicate the decay of the singular values in log-scale. The faster the decay the better, because a smaller rank can be used

in the block, resulting in a smaller approximation error. Completely white (i.e., empty) blocks are "zero" blocks, which contain zero entries.

After the matrix has been decomposed into sub-blocks, the low-rank approximations (green blocks) should be computed. For this purpose, the Adaptive Cross Approximation algorithm (or similar techniques like ACA+ or HACA) [7, 11, 13, 14, 24] is used, which performs the approximations with a linear complexity $\mathcal{O}(kn)$, where n is the size of each sub-block.

Remark 2.2. The $\mathcal{H}$-matrix approximation error may destroy the symmetry; therefore, we perform the trick $\boldsymbol{C} := \frac{1}{2}(\boldsymbol{C} + \boldsymbol{C}^\top)$.

Definition 2.1. We let I be an index set. The hierarchical decomposition of I is described by the cluster tree T_I (see Fig. 4). The index set I is the root of the tree. Usually I, is decomposed into two or three additional sub-index sets. Each of these index sets is decomposed again into two or three sub-index sets. A hierarchical division of the index sets product $(I \times I)$ into sub-blocks (Fig. 4, right) is called a block cluster tree $(T_{I\times I})$. The set of $\mathcal{H}$-matrices is

$$\mathcal{H}(T_{I\times I}, k) := \{\boldsymbol{C} \in \mathbb{R}^{I\times I} \,|\, \operatorname{rank}(\boldsymbol{C}|_b) \leq k \text{ for all admissible blocks } b \text{ of } T_{I\times I}\},$$

where k is the maximum rank. Here, $\boldsymbol{C}|_b = (c_{ij})_{(i,j)\in b}$ denotes the matrix block of $\boldsymbol{C} = (c_{ij})_{i,j\in I}$ corresponding to sub-block $b \in T_{I\times I}$.

Blocks that satisfy the admissibility condition can be approximated by low-rank matrices; see [29]. The $\mathcal{H}$-matrix approximation of $\boldsymbol{C}$ is denoted by $\tilde{\boldsymbol{C}}$. Figure 4 demonstrates

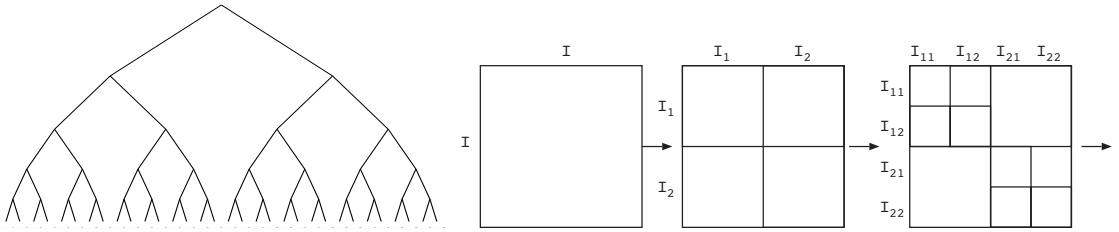

Figure 4: Examples of a cluster tree T_I (left) and a block cluster tree $T_{I\times I}$ (right). The decomposition of the matrix into sub-blocks is defined by $T_{I\times I}$ and the admissibility condition.

the constructing of the cluster and block-cluster trees. The first step (left) is the hierarchical decomposition of the index set I into sub-sets. The maximal size of the smallest

sub-index set is regulated by the parameter $n_{\min}$. The construction of the block-cluster tree $T_{I\times J}$ (in this example $I=J$) is illustrated on the right. $T_{I\times J}$ is the Cartesian product of two index sets, I and J. Additionally, the admissibility condition decides wether or not a certain block will be divided further. For instance, the sub-block $I_1\times I_1$ is divided further and sub-block $I_1\times I_2$ is not.

Remark 2.3. (Fixed and adaptive-rank strategies) For each sub-block, we either fix the accuracy ε or the maximal approximative rank k. In the first case, we speak about the *adaptive-rank strategy*, i.e., the accuracy in the spectral norm for each sub-block is ε. In the second variant, we speak about the *fixed-rank strategy*, i.e., each sub-block has the maximal rank k. The fixed-rank strategy does not provide us with a reliable estimate, but does make it easier to estimate the total computing cost and memory storage. We write $\mathcal{H}(T_{I\times I};\varepsilon)$ and $\tilde{\mathcal{L}}(\boldsymbol{\theta};\varepsilon)$ for the adaptive ranks, and $\mathcal{H}(T_{I\times I};k)$ and $\tilde{\mathcal{L}}(\boldsymbol{\theta};k)$ for the fixed ranks. The fixed-rank strategy is useful for a priori evaluations of the computational resources and storage memory. The adaptive-rank strategy is preferable for practical approximations and is useful when the accuracy in each sub-block is crucial.

Boxplots for different ranks $k\in\{3,7,9,12\}$ are shown in Fig. 5 (left). In the experiment, we run 100 replicates. The boxplots for $k\in\{7,9,12\}$ do not change; therefore, there is no reason to increase the rank, i.e., rank 7 is sufficient. Fig. 5 shows that the estimation of the unknown parameter is better when there are more observations.

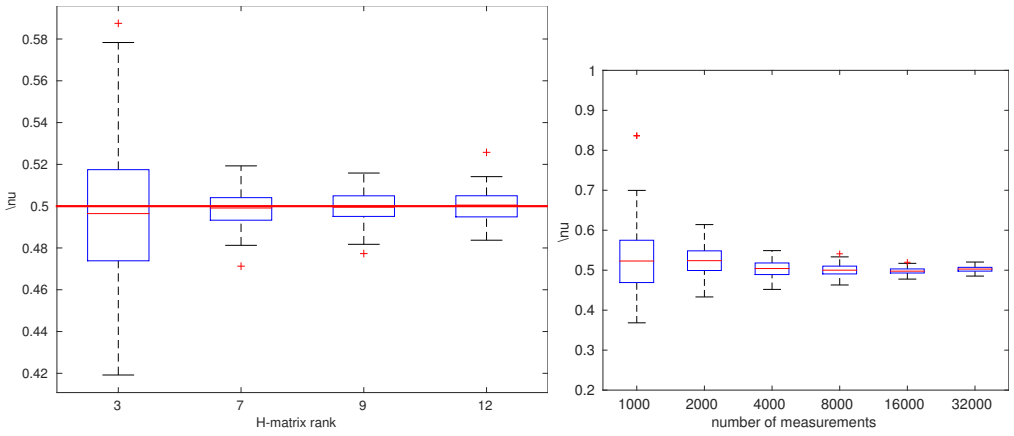

Figure 5: (left) Dependence of the boxplots for ν on the $\mathcal{H}$-matrix rank, when $n=16{,}000$; (right) Convergence of the boxplots for ν with increasing n; 100 replicates.

In Fig. 6 (left), we illustrate the shape of a negative log-likelihood, quadratic form, and log-determinant $\log(|C|)$. The true value of the parameter $\theta^*=\ell=12$. In Fig. 6 (right),

the three log-likelihood functions - the exact one and two others computed with different ranks 7 and 17 - are similar.

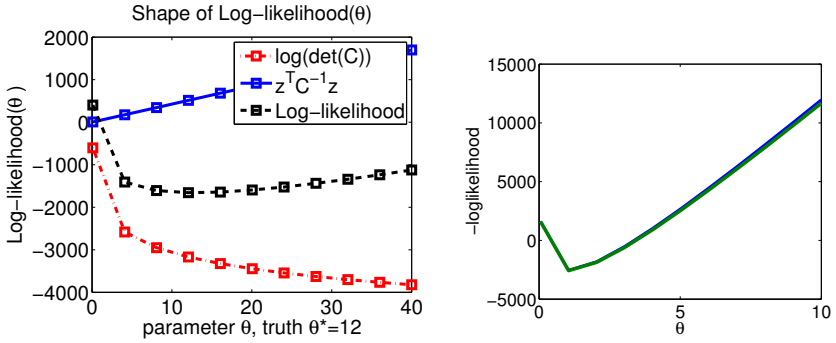

Figure 6: (left) Shape of the log-likelihood function; (right) Three negative log-likelihood functions: exact and computed with $\mathcal{H}$-matrix ranks 7 and 17. Even with rank 7, very accurate results can be achieved.

2.3 Parallel hierarchical-matrix technique

We used the parallel $\mathcal{H}$-matrix library HLIBpro [26, 42–44] to approximate the Matérn covariance matrix, to compute a Cholesky factorization, to solve a linear system, and to calculate the determinant and a quadratic form. HLIBpro is fast, robust, and efficient; see the theoretical parallel complexity in Table 1. Here, $|V(T)|$ denotes the number of vertices, $|L(T)|$ is the number of leaves in the block-cluster tree $T = T_{I\times I}$, and $n_{\min}$ is the size of a block when we stop dividing the sub-blocks (see Section 2.2). Usually, $n_{\min} = 32$ or 64; a deeper hierarchy slows down computations.

3 Memory storage, computing time, and convergence

The Kullback-Leibler divergence (KLD) $D_{KL}(P\|Q)$ is a measure of information lost when a distribution Q is used to approximate P. For the multivariate normal distributions $(\boldsymbol{\mu}_0, \boldsymbol{C})$ and $(\boldsymbol{\mu}_1, \tilde{\boldsymbol{C}})$, it is defined as follows:

$$D_{KL}(\boldsymbol{C}, \tilde{\boldsymbol{C}}) = 0.5\left(\mathrm{tr}(\tilde{\boldsymbol{C}}^{-1}\boldsymbol{C}) + (\boldsymbol{\mu}_1 - \boldsymbol{\mu}_0)^\top \tilde{\boldsymbol{C}}^{-1}(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_0) - n - \ln\left(\frac{|\boldsymbol{C}|}{|\tilde{\boldsymbol{C}}|}\right)\right)$$

In Tables 2 and 3, we show the dependence of KLD and two matrix errors on the $\mathcal{H}$-matrix rank k for the Matérn covariance function with parameters $\ell = \{0.25, 0.75\}$ and

Table 1: Parallel complexity of the main linear operations in HLIBpro on p cores.

Operation	Parallel Complexity [42] (Shared Memory)
build $\tilde{\boldsymbol{C}}$	$\frac{\mathcal{O}(n\log n)}{p} + \mathcal{O}(\lvert V(T)\backslash L(T)\rvert)$
store $\tilde{\boldsymbol{C}}$	$\mathcal{O}(kn\log n)$
$\tilde{\boldsymbol{C}}\cdot\boldsymbol{z}$	$\frac{\mathcal{O}(kn\log n)}{p} + \frac{n}{\sqrt{p}}$
$\alpha\tilde{\mathbf{A}}\oplus\beta\tilde{\mathbf{B}}$	$\frac{\mathcal{O}(n\log n)}{p}$
$\alpha\tilde{\mathbf{A}}\odot\tilde{\mathbf{B}}\oplus\beta\tilde{\boldsymbol{C}}$	$\frac{\mathcal{O}(n\log n)}{p} + \mathcal{O}(C_{sp}(T)\lvert V(T)\rvert)$
$\tilde{\boldsymbol{C}}^{-1}$	$\frac{\mathcal{O}(n\log n)}{p} + \mathcal{O}(nn_{\min}^2)$, $n_{\min}\approx 64$
$\mathcal{H}$-Cholesky $\tilde{\boldsymbol{L}}$	$\frac{\mathcal{O}(n\log n)}{p} + \mathcal{O}(\frac{k^2 n\log^2 n}{n^{1/d}})$, $d=1,2,3$
determinant $\lvert\tilde{\boldsymbol{C}}\rvert$	$\frac{\mathcal{O}(n\log n)}{p} + \mathcal{O}(\frac{k^2 n\log^2 n}{n^{1/d}})$, $d=1,2,3$

$\nu=\{0.5,1.5\}$, computed on the domain $\mathcal{G}=[0,1]^2$. All errors are under control, except for the last column. The ranks $k=10,12$ are not sufficient to approximate the inverse, and the resulting error $\|\boldsymbol{C}(\tilde{\boldsymbol{C}})^{-1}-\boldsymbol{I}\|_2$ is large. One remedy could be to increase the rank, but it may not help (depending on the conditional number). Relatively often, the $\mathcal{H}$-matrix procedure, which computes the $\mathcal{H}$-Cholesky factor $\tilde{\boldsymbol{L}}$ or the $\mathcal{H}$-inverse, produces "NaN" (not a number) and terminates. One possible cause is that some of the diagonal elements can be very close to zero, and their inverse is not defined. This may happen when two locations are very close to each other and, as a result, two columns (rows) are linear dependent. To avoid such cases, the available data should be preprocessed to remove duplicate locations. Very often, the nugget $\tau^2\boldsymbol{I}$ is added to the main diagonal to stabilize numerical calculations (see more in Section 5.2), i.e., $\tilde{\boldsymbol{C}}:=\tilde{\boldsymbol{C}}+\tau^2\boldsymbol{I}$. By adding a nugget, we "push" all the singular values away from zero. In Tables 2 and 3, the nugget is equal to zero.

k	KLD		$\|\boldsymbol{C}-\tilde{\boldsymbol{C}}\|_2$		$\|\boldsymbol{C}(\tilde{\boldsymbol{C}})^{-1}-\boldsymbol{I}\|_2$	
	$\ell=0.25$	$\ell=0.75$	$\ell=0.25$	$\ell=0.75$	$\ell=0.25$	$\ell=0.75$
10	2.6e-3	0.2	7.7e-4	7.0e-4	6.0e-2	3.1
12	5.0e-4	2e-2	9.7e-5	5.6e-5	1.6e-2	0.5
15	1.0e-5	9e-4	2.0e-5	1.1e-5	8.0e-4	0.02
20	4.5e-7	4.8e-5	6.5e-7	2.8e-7	2.1e-5	1.2e-3
50	3.4e-13	5e-12	2.0e-13	2.4e-13	4e-11	2.7e-9

Table 2: Convergence of the $\mathcal{H}$-matrix approximation error vs. the $\mathcal{H}$-matrix rank k of a Matérn covariance function with parameters $\ell=\{0.25,0.75\}$, $\nu=0.5$, domain $\mathcal{G}=[0,1]^2$, and $\|C_{(\ell=0.25,0.75)}\|_2=\{212,568\}$.

k	KLD		$\|\boldsymbol{C}-\tilde{\boldsymbol{C}}\|_2$		$\|\boldsymbol{C}(\tilde{\boldsymbol{C}})^{-1}-\boldsymbol{I}\|_2$	
	$\ell=0.25$	$\ell=0.75$	$\ell=0.25$	$\ell=0.75$	$\ell=0.25$	$\ell=0.75$
20	0.12	2.7	5.3e-7	2e-7	4.5	72
30	3.2e-5	0.4	1.3e-9	5e-10	4.8e-3	20
40	6.5e-8	1e-2	1.5e-11	8e-12	7.4e-6	0.5
50	8.3e-10	3e-3	2.0e-13	1.5e-13	1.5e-7	0.1

Table 3: Convergence of the $\mathcal{H}$-matrix approximation error vs. the $\mathcal{H}$-matrix rank k of a Matérn covariance function with parameters $\ell=\{0.25,0.75\}$, $\nu=1.5$, domain $\mathcal{G}=[0,1]^2$, and $\|C_{(\ell=0.25,0.75)}\|_2=\{720,1068\}$.

Figure 7 shows that the $\mathcal{H}$-matrix storage cost remains almost the same for the different parameters $\ell=\{0.15,...,2.2\}$ (left) and $\nu=\{0.3,...,1.3\}$ (right). The computational domain is $[32.4,43.4]\times[-84.8,-72.9]$ with $n=2{,}000$.

In Figure 8, we plot the convergence of $\|\boldsymbol{C}-\tilde{\boldsymbol{C}}\|$ in the Frobenius and spectral norms vs. the rank k for different covariance lengths. The smoothness parameter is equal to 1 (left), and 0.5 (right). In Figure 9, we plot $\|\boldsymbol{C}-\tilde{\boldsymbol{C}}\|_2$ vs. the rank k for different smoothness parameters. The covariance length is equal to 0.1(left), and 0.5 (right). The computational domain in both cases was a unit square $[0,1]^2$.

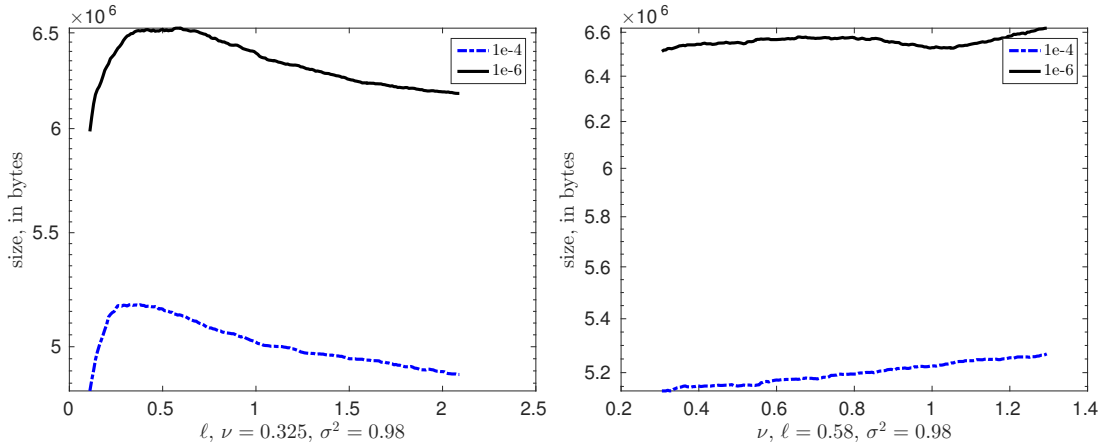

Figure 7: (left) Dependence of the matrix size on (left) the covariance length ℓ, and (right) the smoothness ν for two different accuracies in the $\mathcal{H}$-matrix sub-blocks $\varepsilon = \{1e-4, 1e-6\}$, for $n = 2{,}000$ locations in the domain $[32.4, 43.4] \times [-84.8, -72.9]$.

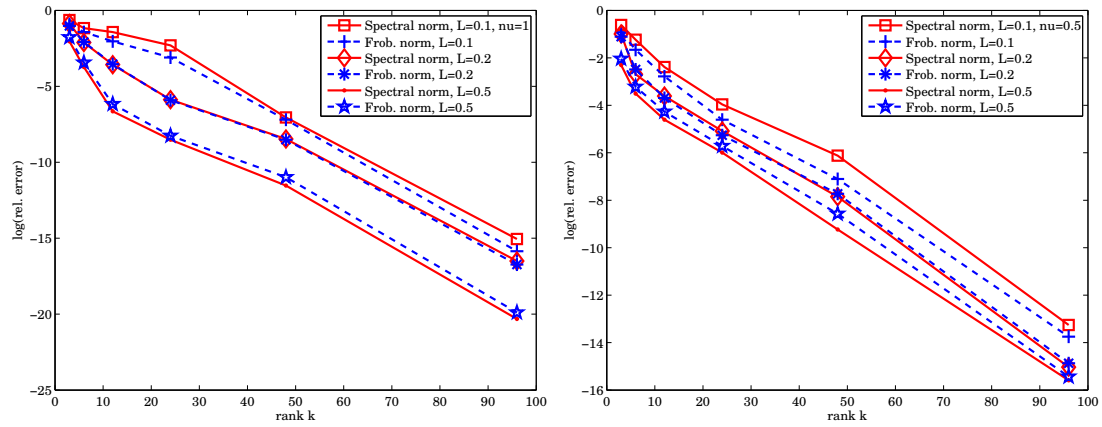

Figure 8: Convergence of the $\mathcal{H}$-matrix approximation errors for covariance lengths $\{0.1, 0.2, 0.5\}$; (left) $\nu = 1$ and (right) $\nu = 0.5$, computational domain $[0,1]^2$.

4 Software installation and numerical examples

4.1 Introduction to HLIBCov and HLIBpro

This section contains a summary of the information provided at `www.hlib.com`. The software library HLIBpro, developed by Ronald Kriemann, is a robust parallel C++ implementation of the $\mathcal{H}$-matrix technique and $\mathcal{H}$-matrix arithmetics [42]. HLIBpro supports

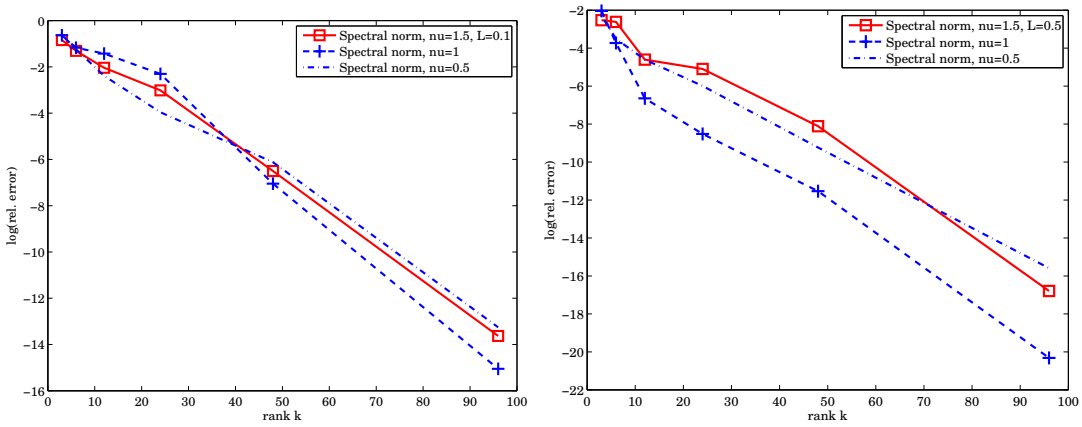

Figure 9: Convergence of the $\mathcal{H}$-matrix approximation errors for $\nu = \{0.5,1,1.5\}$; (left) covariance length 0.1 and (right) covariance length 0.5, computational domain $[0,1]^2$.

both the shared and distributed memory architectures, though in this work we use only the shared memory version. Intel's Threading Building Blocks (TBB) are used for parallelization on the shared memory systems. HLIBpro is free for academic purposes, and is distributed as a compiled code (no source code is available). Originally, HLIBpro was developed for solving FEM and BEM problems [26, 44]. In this work, we extend the applicability of HLIBpro to dense covariance matrices and log-likelihood functions.

Installation: HLIBCov uses the functionality of HLIBpro; therefore, HLIBpro must be installed first. Since HLIBcov is just a few C++ procedures (contained in one file, loglikelihood.cc), the applicability of HLIBCov is limited only by the applicability of HLIBpro. To install HLIBpro on MacOS and Windows, we refer the reader to `www.HLIBpro.com` for further details. Additionally, HLIBCov requires the GNU Scientific Library (GSL). We use GSL-1.16; the installation steps are described at `https://www.gnu.org/software/gsl/`. The GSL library provides maximization algorithms and Bessel functions. The reader can easily replace GSL with his own optimization library. The Bessel functions are also available in other packages.

Hardware. All of the numerical experiments herein are performed on a Dell workstation with 20 processors (40 cores) and total 128 GB RAM.

Adding HLIBCov to HLIBpro. For use with the default settings/paths, there is no need to change the *SConstruct* file. To compile HLIBCov, we add the following line to */examples/SConscript*;

```
1 $ examples.append(cxxenv.Program('loglikelihood.cc'))
```

```
$ cd ..
$ scons
$ cd examples/
$ ./loglikelihood
```

Input of HLIBCov

The input contained in the first line is the total number of locations N. Lines 2,...,$N+1$ contain the coordinates x_i, y_i, and the measurement value. An example is provided below;

```
3
0.1  0.2  88.1
0.1  0.3  87.2
0.2  0.4  86.0
```

HLIBpro requires neither a list of finite elements nor a list of edges. We provide several examples of few input files of different size on the github.

Output of HLIBCov

The main output is the three identified parameter values $\boldsymbol{\theta} = (\ell, \nu, \sigma^2)$. The auxiliary output may include many details: $\mathcal{H}$-matrix details (the maximal rank k, the maximal accuracy in each sub-block, and the Frobenius and spectral norms of $\tilde{\boldsymbol{C}}$, $\tilde{\boldsymbol{L}}$, $\tilde{\boldsymbol{L}}^{-1}$, $\|\boldsymbol{I} - \tilde{\boldsymbol{L}}\tilde{\boldsymbol{L}}^{\top -1}\|$). Additionally, iterations of the maximization algorithm can also be printed out. The example of an output file provided below contains two iterations: the index, ν, ℓ, σ^2, $\tilde{\mathcal{L}}$, and the residual TOL of the iterative method:

```
1 0.27   2.4   1.30  L = 1762.1 TOL= 0.007
2 0.276 2.41 1.29 L = 1757.2 TOL= 0.009
```

If the iterative process is converging, then the last row will contain the solution $\boldsymbol{\theta}^* = (\ell^*, \nu^*, \sigma^{*2})$. When computing error boxes, the output file will contain M solutions (n, ℓ^*, ν^*, σ^{*2}), where M is the number of replicants:

```
4000 5.4e-1 8.2e-2 1.01
4000 5.3e-1 8.3e-2 1.02
4000 5.5e-1 8.1e-2 1.02
```

4.2 Preprocessing the C++ function

Below we provide the C++ code, which reads the data and initializes all the required C++ objects (*loglikelihood.cc*):

```
    vector< double * > vertices;
    TScalarVector     rhs;
    INIT();
    in >> N;
```

```
    cout << " reading " << N << " coordinates" << endl;
    vertices.resize( N );
    rhs.set_size( N );
    double x, y, v=0.0;
    for ( int i = 0; i < N; ++i ){
           in >> x >> y >> v;
           vertices[i] = new double[ dim ];
           vertices[i][0] = x; vertices[i][1] = y;
           rhs.set_entry( i, v ); }// for
    TCoordinate coord( vertices, dim );
    TAutoBSPPartStrat part_strat( adaptive_split_axis );
    TBSPCTBuilder     ct_builder( & part_strat );
    auto              ct = ct_builder.build( & coord );
    TStdGeomAdmCond   adm_cond( 2.0, use_min_diam );
    TBCBuilder        bct_builder( std::log2( 16 ) );
    auto              bct = bct_builder.build( ct.get(), ct.get(), & adm_cond );
    // bring RHS into H-ordering
    ct->perm_e2i()->permute( & rhs );
    double output[3];
    call_compute_max_likelihood(rhs, nu, length, sigma2, bct.get(), ct.get(), vertices, output);
    DONE();
```

HLIBpro License

To receive the license file **HLIBpro.lic**, send a request via email to Ronald Kriemann (rok@mis.mpg.de). In your request, include your user login name and the name of the machine on which you plan to run the code.The machine name can be received with the terminal command *hostname*. The license file, once received, must be placed either in the directory where the final program will be executed, e.g., in the "examples" subdirectory, or in the directory indicated by the environment variable, HLIBpro_LIC_PATH. You can add this variable to the system paths as follows:

```
$ export hlibpro_LIC_PATH=<path_to_HLIBpro.lic>
```

5 Numerical experiments with synthetic data

Here we perform numerical experiments with simulated data to recover the true values of the parameters of the Matérn covariance matrix, which are known to be $(\ell^*,\nu^*,\sigma^*) = (1.0,0.5,1.0)$.

5.1 Generation of the synthetic data

To build M various data-sets (M replicates) with $n \in \{128,64,...,4,2\} \times 1000$ locations, we generate a large vector $\boldsymbol{Z}_0$ with $n_0 = 2 \cdot 10^6$ locations, and randomly sample n points from it. We note that if the locations are very close to each other, then the covariance matrix may be singular or the Cholesky factorization will be very difficult to compute.

To generate the random data $\boldsymbol{Z}_0 \in \mathbb{R}^{n_0}$, we compute the $\mathcal{H}$-Cholesky factorization of $\boldsymbol{C}(1.0,0.5,1.0) = \tilde{\boldsymbol{L}}\tilde{\boldsymbol{L}}^\top$. Then, we evaluate $\boldsymbol{Z}_0 = \tilde{\boldsymbol{L}}\boldsymbol{\xi}$, where $\boldsymbol{\xi} \in \mathbb{R}^{n_0}$ is a normal vector with zero mean and unit variance. We generate $\boldsymbol{Z}_0$ only once. Next, we run our optimization algorithm and try to identify (recover) the "unknown" parameters (ℓ^*,ν^*,σ^*). The resulting boxplots for ℓ and ν^2 over $M = 100$ replicates are illustrated in Fig. 10. We see that the variance (or uncertainty) decreases with increasing n. The green line indicates the true values.

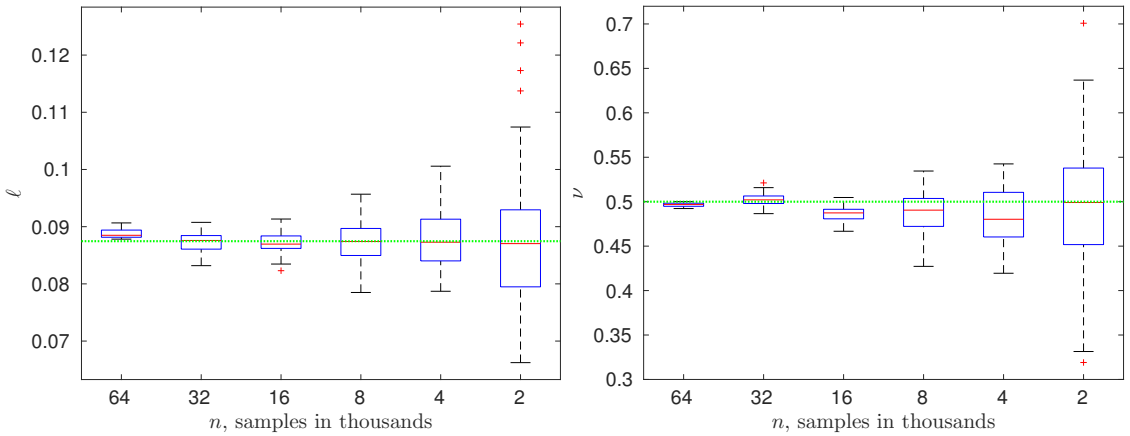

Figure 10: Synthetic data with known parameters $(\ell^*,\nu^*,\sigma^{2^*}) = (0.5,1,0.5)$. Boxplots for ℓ and ν for $n = 1{,}000 \times \{64,32,...,4,2\}$; 100 replicates.

To identify all three parameters simultaneously, we solve a three-dimensional optimization problem. The maximal number of iterations is set to 200, and the residual is 10^{-6}. The behavior and accuracy of the boxplots depend on the $\mathcal{H}$-matrix rank, the maximum number of iterations to achieve a certain threshold, the threshold (or residual) itself, the initial guess, the step size in each parameter of the maximization algorithm, and the maximization algorithm. All replicates of $\boldsymbol{Z}$ are sampled from the same generated vector of size $n_0 = 2 \cdot 10^6$.

In Table 4, we present the almost-linear storage cost (columns 3 and 6) and the computing time (columns 2 and 5).

The shape of the log-likelihood function and its components are illustrated in Fig. 11. This helps us to understand the behavior of the iterative optimization method, and the contributions of the log-determinant and the quadratic functional. We see that the log-likelihood is almost flat, and that may be necessary to perform many iterations in order

Table 4: Computing time and storage cost for parallel $\mathcal{H}$-matrix approximation; number of cores is 40, $\hat{\nu}=0.33$, $\hat{\ell}=0.65$, $\hat{\sigma^2}=1.0$. $\mathcal{H}$-matrix accuracy in each sub-block for both $\tilde{\boldsymbol{C}}$ and $\tilde{\boldsymbol{L}}$ is 10^{-5}.

n	$\tilde{\boldsymbol{C}}$			$\tilde{\boldsymbol{L}}\tilde{\boldsymbol{L}}^\top$		
	comp. time sec.	size MB	kB/dof	comp. time sec.	size MB	$\|\boldsymbol{I}-(\tilde{\boldsymbol{L}}\tilde{\boldsymbol{L}}^\top)^{-1}\tilde{\boldsymbol{C}}\|_2$
32,000	3.3	162	5.1	2.4	172.7	$2.4\cdot10^{-3}$
128,000	13.3	776	6.1	13.9	881.2	$1.1\cdot10^{-2}$
512,000	52.8	3420	6.7	77.6	4150	$3.5\cdot10^{-2}$
2,000,000	229	14790	7.4	473	18970	$1.4\cdot10^{-1}$

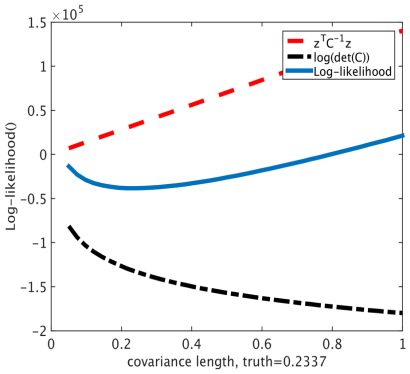

Figure 11: Dependence of the negative log-likelihood and its ingredients on parameter ℓ, where parameter $\nu=0.5$ is fixed and the true value of the covariance length is 0.2337. Exponential covariance, $n=66049$, rank $k=16$, $\sigma^2=1$.

to find the minimum.

Table 5: Comparison of three log-likelihood functions computed with three different $\mathcal{H}$-matrix accuracies $\{10^{-7},10^{-9},10^{-11}\}$. Exponential covariance function discretized in the domain $[32.4,43.4]\times[-84.8,-72.9]$, $n=32{,}000$ locations. Columns correspond to different covariance lengths $\{0.001,...,0.1\}$.

ℓ	0.001	0.005	0.01	0.02	0.03	0.05	0.07	0.1
$-\tilde{\mathcal{L}}(\ell;10^{-7})$	44657	36157	36427	40522	45398	68450	70467	90649
$-\tilde{\mathcal{L}}(\ell;10^{-9})$	44585	36352	36113	41748	47443	60286	70688	90615
$-\tilde{\mathcal{L}}(\ell;10^{-11})$	44529	37655	36390	42020	47954	60371	72785	90639

5.2 Nugget τ^2

When the diagonal values of $\tilde{\boldsymbol{C}}$ are very close to zero, then the numerical algorithms cannot compute the $\mathcal{H}$-Cholesky factor $\tilde{\boldsymbol{L}}$ or the inverse. They produce "NaN" or the error message "division by zero". By adding a diagonal matrix with small positive numbers, all the singular values are increased and moved away from zero. Of course, by adding a nugget, we redefine the original matrix as $\tilde{\boldsymbol{C}} := \tilde{\boldsymbol{C}} + \tau^2 \boldsymbol{I}$. Below, by adding a nugget, we analyze how the loglikelihood function, as well as its maximum are changing.

We assume $|\tilde{\boldsymbol{C}}| \neq 0$. For a small perturbation matrix $\mathbf{E}$ [19], it holds that

$$\frac{\|(\tilde{\boldsymbol{C}}+\mathbf{E})^{-1} - (\tilde{\boldsymbol{C}})^{-1}\|}{\|\tilde{\boldsymbol{C}}^{-1}\|} \leq \kappa(\boldsymbol{C}) \cdot \frac{\|\mathbf{E}\|}{\|\tilde{\boldsymbol{C}}\|} = \frac{\kappa(\tilde{\boldsymbol{C}})\tau^2}{\|\tilde{\boldsymbol{C}}\|},$$

where $\kappa(\tilde{\boldsymbol{C}})$ is the condition number of $\tilde{\boldsymbol{C}}$, and $\mathbf{E} = \tau^2 \boldsymbol{I}$. Alternatively, by substituting $\kappa(\tilde{\boldsymbol{C}}) := \|\tilde{\boldsymbol{C}}\| \cdot \|\tilde{\boldsymbol{C}}^{-1}\|$, we obtain

$$\frac{\|(\tilde{\boldsymbol{C}}+\tau^2 \boldsymbol{I})^{-1} - (\tilde{\boldsymbol{C}})^{-1}\|}{\|\tilde{\boldsymbol{C}}^{-1}\|} \leq \tau^2 \|\tilde{\boldsymbol{C}}^{-1}\|. \tag{5.1}$$

From Eq. 5.1, we see that the relative error on the left-hand side of Eq. 5.1 depends on the norm $\|\tilde{\boldsymbol{C}}^{-1}\|$, i.e., the relative error is inversely proportional to the smallest singular value of $\boldsymbol{C}$. This may explain a possible failing of approximating matrices, where the smallest singular values tend towards zero. The estimates for the $\mathcal{H}$-Cholesky and the Schur complement for general sparse positive-definite matrices are given in [9]). The approximation errors are proportional to the cond$(\boldsymbol{C})$, i.e., matrices with a very large conditional number may require a very large $\mathcal{H}$-matrix rank. Figure 12 (left) demonstrates three negative log-likelihood functions omputed with the nuggets 0.01, 0.005, and 0.001. For this particular example, the behavior of likelihood is preserved, and the minimum does not change (or changes very slightly). Figure 12 (right) is just a zoomed in version of the picture on the left.

6 Best practices (HLIBpro)

In this section, we list our recommendations and warnings.

1. Initialize the variables and the license with `INIT()`.

2. For practical computations, use adaptive-rank arithmetic.

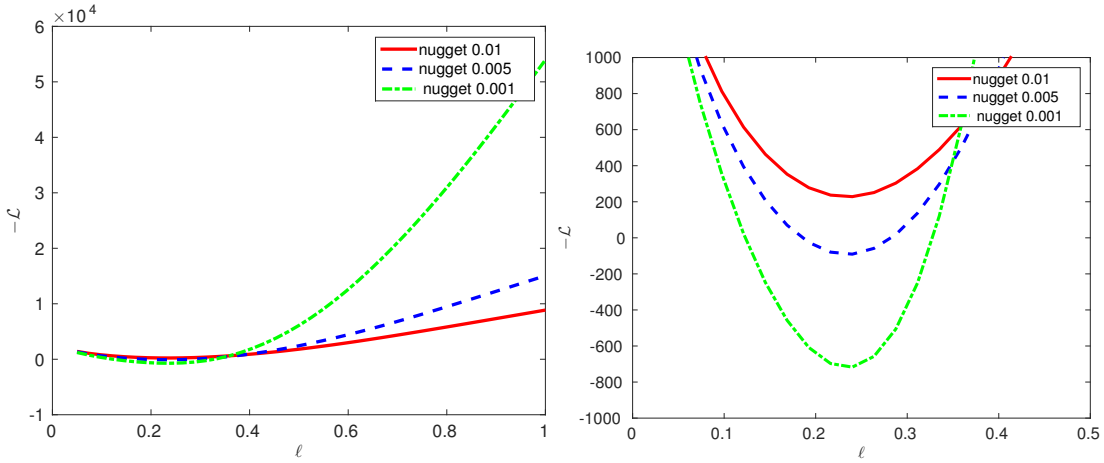

Figure 12: (left) Dependence of the log-likelihood on parameter ℓ with nuggets ($\{0.01, 0.005, 0.001\}$) for Gaussian covariance. (right) Zoom of the left figure near minimum; $n = 2000$ random points from moisture example, rank $k = 14$, $\sigma^2 = 1$.

3. For the input, it is sufficient to define a file by three columns: both location coordinates (x, y) and the observed value; no triangles or edges are required.

4. If two locations coincide or are very close to each other, then the matrix will be close to singular or singular. As a result, it will be hard to compute the Cholesky factorization. Our suggested remedy is to improve the quality of the locations by preprocessing the data.

5. Construction of $\mathcal{H}$-matrices permutes the indices of the degrees of freedom. Therefore, there are two permutation mappings: `ct->perm_e2i()` and `ct->perm_i2e()`. For example, to prepare the observation vector $\boldsymbol{Z}$ (computed on a grid) for multiplication with the hierarchical matrix $\tilde{\boldsymbol{L}}^{-1}$ (or $\tilde{\boldsymbol{C}}^{-1}$), use `ct->perm_e2i()->permute(&Z)`. To find which grid node corresponds to the i-th row (column), use `ct->perm_i2e()`.

6. HLIBpro uses smart pointers, e.g., `std::unique_ptr`. If the user leaves a code block, those smart pointers will automatically delete the object they manage. This simplifies the programming and makes it much more secure.

7. By default, the $\mathcal{H}$-Cholesky or $\mathcal{H}-$LU factorizations are always directed acyclic graph (DAG)- based. This can be turned off by setting `HLIB::CFG::Arith::use_dag = false`.

8. By default, HLIBpro uses all available computing cores. To perform computations on 16 cores, use `HLIB::CFG::set_nthreads(16)` at the beginning of the program (after command `INIT()`).

9. Since HLIBpro is working for 1D, 2D and 3D domains, only very minor changes are required to move from 1D locations to 2D or 3D in HLIBCov. Replace dim=2 with dim=3 in
   `TCoordinate coord(vertices, dim);`
   then add `>> z` to `in >> x >> y>> z >> v;`

10. Finalize all computations with `DONE();`

The $\mathcal{H}$-matrix data format is a rather complicated data structure (class) in HLIBpro. Therefore, the $\mathcal{H}$-matrix objects (or the pointers on them) are neither the input nor the output parameters. Instead, the input parameters for the HLIBpro C++ routines are: a vector (array) of locations, a vector of observations $\boldsymbol{Z}$, etc. The triangulation (a list of triangles/edges) is not needed. The output parameters are either scalar values or a vector; for example, the determinant, the trace, a norm, the result of the matrix-vector product, an approximation error, etc.

7 Conclusion

We extended functionality of the parallel $\mathcal{H}$-matrix library HLIBpro to infer unknown parameters for applications in spatial statistics. This new extension allows us to work with large covariance and precision matrices. We approximated the joint multivariate Gaussian likelihood function and found its maxima in the $\mathcal{H}$-matrix format. These maxima were used to estimate the unknown parameters (ℓ, ν, and σ^2) of a covariance model. The new code is parallel, highly efficient, and written in C++ language. With the $\mathcal{H}$-matrix technique, we reduced the storage cost and the computing cost (Tables 3, 4) of the log-likelihood function dramatically, from cubic to almost linear. We demonstrated these advantages in a synthetic example, where we were able to identify the true parameters of the covariance model. We were also able to compute the log-likelihood function for 2,000,000 locations in just a few minutes on a powerful desktop machine (Table 4). The $\mathcal{H}$-matrix technique allowed us to increase the spatial resolution, handle more measurements, consider larger regions, and identify more parameters simultaneously.

Acknowledgments

The author thanks Ronald Kriemann from the Max Planck Institute for Mathematics in the Sciences in Leipzig for his assistance with the HLIBpro code. This work would be impossible without assistance from Marc Genton and Ying Sun. The research reported in this publication was supported by funding from the Extreme Computing Research Center (ECRC) at King Abdullah University of Science and Technology (KAUST).

References

[1] M. Abramowitz and I. A. Stegun. *Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables*. Number 55 in National Bureau of Standards Applied Mathematics Series. Superintendent of Documents, U.S. Government Printing Office, Washington, DC, 1964.

[2] S. Ambikasaran, D. Foreman-Mackey, L. Greengard, D. W. Hogg, and M. OŃeil. Fast direct methods for Gaussian processes and the analysis of NASA Kepler mission data. *arXiv preprint arXiv:1403.6015*, 2014.

[3] S. Ambikasaran, J. Y. Li, P. K. Kitanidis, and E. Darve. Large-scale stochastic linear inversion using hierarchical matrices. *Computational Geosciences*, 17(6):913–927, 2013.

[4] 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):240–262, Feb. 2012.

[5] J. Ballani and D. Kressner. Sparse inverse covariance estimation with hierarchical matrices. *http://sma.epfl.ch/∼anchpcommon/publications/quic_ballani_kressner_2014.pdf*, 2015.

[6] S. Banerjee, A. E. Gelfand, A. O. Finley, and H. Sang. Gaussian predictive process models for large spatial data sets. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 70(4):825–848, 2008.

[7] M. Bebendorf. Approximation of boundary element matrices. *Numerical Mathematics*, 86(4):565–589, 2000.

[8] M. Bebendorf. Why approximate LU decompositions of finite element discretizations of elliptic operators can be computed with almost linear complexity. *SIAM J. Numerical Analysis*, 45:1472–1494, 2007.

[9] M. Bebendorf and T. Fischer. On the purely algebraic data-sparse approximation of the inverse and the triangular factors of sparse matrices. *Numerical Linear Algebra with Applications*, 18(1):105–122, 2011.

[10] M. Bebendorf and W. Hackbusch. Existence of H-matrix approximants to the inverse FE-matrix of elliptic operators with L^∞-coefficients. *Numerische Mathematik*, 95(1):1–28, 2003.

[11] M. Bebendorf and S. Rjasanow. Adaptive low-rank approximation of collocation matrices. *Computing*, 70(1):1–24, 2003.

[12] S. Börm and J. Garcke. Approximating Gaussian processes with H^2-matrices. In J. N. Kok, J. Koronacki, R. L. de Mantaras, S. Matwin, D. Mladen, and A. Skowron, editors, *Proceedings of 18th European Conference on Machine Learning, Warsaw, Poland, September 17-21, 2007. ECML 2007*, volume 4701, pages 42–53, 2007.

[13] S. Börm and L. Grasedyck. Hybrid cross approximation of integral operators. *Numer. Math.*, 101(2):221–249, 2005.

[14] S. Börm, L. Grasedyck, and W. Hackbusch. *Hierarchical Matrices*, volume 21 of *Lecture Note*. Max-Planck Institute for Mathematics, Leipzig, 2003. www.mis.mpg.de.

[15] R. P. Brent. Chapter 4: An algorithm with guaranteed convergence for finding a zero of a function, algorithms for minimization without derivatives. *Englewood Cliffs, NJ: Prentice-Hall*, 1973.

[16] J. E. Castrillon, M. G. Genton, and R. Yokota. Multi-Level Restricted Maximum Likelihood Covariance Estimation and Kriging for Large Non-Gridded Spatial Datasets. *Spatial Statistics*, 18:105–124, 2016.

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

[18] A. Datta, S. Banerjee, A. O. Finley, and A. E. Gelfand. Hierarchical nearest-neighbor Gaussian process models for large geostatistical datasets. *Journal of the American Statistical Association*, 0(ja):00–00, 2015.

[19] J. Demmel. The componentwise distance to the nearest singular matrix. *SIAM Journal on Matrix Analysis and Applications*, 13(1):10–19, 1992.

[20] S. Dolgov, B. N. Khoromskij, A. Litvinenko, and H. Matthies. Polynomial chaos expansion of random coefficients and the solution of stochastic partial differential equations in the tensor train format. *SIAM/ASA Journal on Uncertainty Quantification*, 3(1):1109–1135, 2015.

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

[22] G.-A. Fuglstad, D. Simpson, F. Lindgren, and H. Rue. Does non-stationary spatial data always require non-stationary random fields? *Spatial Statistics*, 14:505–531, 2015.

[23] 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.

[24] S. A. Goreinov, E. E. Tyrtyshnikov, and N. L. Zamarashkin. A theory of pseudoskeleton approximations. *Linear Algebra Appl.*, 261:1–21, 1997.

[25] L. Grasedyck and W. Hackbusch. Construction and arithmetics of $\mathcal{H}$-matrices. *Computing*, 70(4):295–334, 2003.

[26] L. Grasedyck, R. Kriemann, and S. LeBorne. Parallel black box H-LU preconditioning for elliptic boundary value problems. *Computing and visualization in science*, 11(4-6):273–291, 2008.

[27] J. Guinness and M. 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.

[28] P. Guttorp and T. Gneiting. Studies in the history of probability and statistics XLIX: On the Matérn correlation family. *Biometrika*, 93:989–995, 2006.

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

[30] W. Hackbusch. *Hierarchical matrices: Algorithms and Analysis*, volume 49 of *Springer Series in Comp. Math.* Springer, 2015.

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

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

[33] M. S. Handcock and M. L. Stein. A Bayesian analysis of kriging. *Technometrics*, 35:403–410, 1993.

[34] H. Harbrecht, M. Peters, and M. Siebenmorgen. Efficient approximation of random fields for numerical applications. *Numerical Linear Algebra with Applications*, 22(4):596–617, 2015.

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

[36] C.-J. Hsieh, M. A. Sustik, I. S. Dhillon, P. K. Ravikumar, and R. Poldrack. Big & QUIC: Sparse inverse covariance estimation for a million variables. In C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Weinberger, editors, *Advances in Neural Information Processing Systems 26*, pages 3165–3173. Curran Associates, Inc., 2013.

[37] H. Huang and Y. Sun. Hierarchical low rank approximation of likelihoods for large spatial datasets. *Journal of Computational and Graphical Statistics, to appear. ArXiv e-prints 1605.08898*, 2017.

[38] C. G. Kaufman, M. J. Schervish, and D. W. Nychka. Covariance tapering for likelihood-based estimation in large spatial datasets. *Journal of the American Statistical Association*, 103(484):1545–1555, 2008.

[39] B. N. Khoromskij and A. Litvinenko. Data sparse computation of the Karhunen-Loève expansion. *AIP Conference Proceedings*, 1048(1):311, 2008.

[40] B. N. Khoromskij, A. Litvinenko, and H. G. Matthies. Application of hierarchical matrices for computing the Karhunen–Loève expansion. *Computing*, 84(1-2):49–67, 2009.

[41] W. Kleiber and D. W. Nychka. Equivalent kriging. *Spatial Statistics*, 12:31–49, 2015.

[42] R. Kriemann. Parallel H-matrix arithmetics on shared memory systems. *Computing*, 74(3):273–297, 2005.

[43] R. Kriemann. HLIBpro user manual. Technical report, Max Planck Institute for Mathematics in the Sciences, 2008.

[44] R. Kriemann. H-LU factorization on many-core systems. *Computing and Visualization in Science*, 16(3):105–117, Jun 2013.

[45] F. Lindgren, H. Rue, and J. Lindström. An explicit link between Gaussian fields and Gaus-

sian Markov random fields: the stochastic partial differential equation approach. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 73(4):423–498, 2011.

[46] A. Litvinenko. Application of hierarchical matrices for solving multiscale problems. *PhD Dissertation, Leipzig University*, 2006.

[47] A. Litvinenko. Partial inversion of elliptic operator to speed up computation of likelihood in bayesian inference. *arXiv preprint arXiv:1708.02207*, 2017.

[48] A. Litvinenko, M. Genton, Y. Sun, and D. Keyes. H-matrix techniques for approximating large covariance matrices and estimating its parameters. *PAMM*, 16(1):731–732, 2016.

[49] A. Litvinenko, D. Keyes, V. Khoromskaia, B. N. Khoromskij, and H. G. Matthies. Tucker Tensor analysis of Matern functions in spatial statistics. *ArXiv e-prints*, Nov. 2017.

[50] A. Litvinenko and H. G. Matthies. Sparse data representation of random fields. *PAMM*, 9(1):587–588, 2009.

[51] A. Litvinenko and H. G. Matthies. Inverse problems and uncertainty quantification. *arXiv preprint arXiv:1312.5048, 2013*, 2013.

[52] A. Litvinenko, Y. Sun, M. G. Genton, and D. Keyes. Likelihood approximation with hierarchical matrices for large spatial datasets. *preprint arXiv:1709.04419, submitted to J. Computational Statistics and Data Analysis, Elsevier*, 2017.

[53] 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, 2005.

[54] B. Matérn. *Spatial Variation*, volume 36 of *Lecture Notes in Statistics*. Springer-Verlag, Berlin; New York, second edition edition, 1986.

[55] H. Matthies, A. Litvinenko, O. Pajonk, B. V. Rosić, and E. Zander. Parametric and uncertainty computations with tensor product representations. In A. M. Dienstfrey and R. F. Boisvert, editors, *Uncertainty Quantification in Scientific Computing*, volume 377 of *IFIP Advances in Information and Communication Technology*, pages 139–150. Springer Berlin Heidelberg, 2012.

[56] H. G. Matthies, E. Zander, O. Pajonk, B. V. Rosić, and A. Litvinenko. Inverse problems in a Bayesian setting. In *Computational Methods for Solids and Fluids Multiscale Analysis, Probability Aspects and Model Reduction Editors: Ibrahimbegovic, Adnan (Ed.), ISSN: 1871-3033*, pages 245–286. Springer, 2016.

[57] W. Nowak and A. Litvinenko. Kriging and spatial design accelerated by orders of magnitude: combining low-rank covariance approximations with FFT-techniques. *Mathematical Geosciences*, 45(4):411–435, 2013.

[58] D. Nychka, S. Bandyopadhyay, D. Hammerling, F. Lindgren, and S. Sain. A multiresolution Gaussian process model for the analysis of large spatial datasets. *Journal of Computational and Graphical Statistics*, 24(2):579–599, 2015.

[59] O. Pajonk, B. V. Rosić, A. Litvinenko, and H. G. Matthies. A deterministic filter for non-Gaussian Bayesian estimation — applications to dynamical system estimation with noisy measurements. *Physica D: Nonlinear Phenomena*, 241(7):775–788, 2012.

[60] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. *Section 9.3. Van Wijngaarden-*

Dekker-Brent Method. Numerical Recipes: The Art of Scientific Computing, volume 3rd ed. New York: Cambridge University Press., 2007.

[61] C. E. Rasmussen and C. K. I. Williams. *Gaussian Processes for Machine Learning: www.gaussianprocess.org/gpml/chapters/*, volume 497. MIT Press, 2006.

[62] B. V. Rosić, A. Kučerová, J. Sỳkora, O. Pajonk, A. Litvinenko, and H. G. Matthies. Parameter identification in a probabilistic setting. *Engineering Structures*, 50:179–196, 2013.

[63] B. V. Rosic, A. Litvinenko, O. Pajonk, and H. G. Matthies. Direct bayesian update of polynomial chaos representations. *Informatik-Berichte der Technischen Universität Braunschweig*, 2011-02.

[64] B. V. Rosić, A. Litvinenko, O. Pajonk, and H. G. Matthies. Sampling-free linear bayesian update of polynomial chaos representations. *Journal of Computational Physics*, 231(17):5761–5787, 2012.

[65] H. Rue and L. Held. Gaussian Markov random fields: theory and applications, 2005.

[66] H. Rue and H. Tjelmeland. Fitting Gaussian Markov random fields to Gaussian fields. *Scandinavian Journal of Statistics*, 29(1):31–49, 2002.

[67] A. Saibaba, S. Ambikasaran, J. Yue Li, P. Kitanidis, and E. Darve. Application of hierarchical matrices to linear inverse problems in geostatistics. *Oil & Gas Science and Technology–Rev. IFP Energies Nouvelles*, 67(5):857–875, 2012.

[68] 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 (Statistical Methodology)*, 74(1):111–132, 2012.

[69] S. Si, C.-J. Hsieh, and I. S. Dhillon. Memory efficient kernel approximation. In *International Conference on Machine Learning (ICML)*, jun 2014.

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

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

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

[73] J. R. Stroud, M. L. Stein, and S. Lysen. Bayesian and maximum likelihood estimation for gaussian processes on an incomplete lattice. *Journal of Computational and Graphical Statistics*, 26(1):108–120, 2017.

[74] Y. Sun, B. Li, and M. G. Genton. Geostatistics for large datasets. In M. Porcu, J. M. Montero, and M. Schlather, editors, *Space-Time Processes and Challenges Related to Environmental Problems*, pages 55–77. Springer, 2012.

[75] Y. Sun and M. L. Stein. Statistically and computationally efficient estimating equations for large spatial datasets. *Journal of Computational and Graphical Statistics*, 25(1):187–208, 2016.

[76] E. Zander. SGLib - A Matlab Toolbox for Stochastic Galerkin Methods, Feb. 2010.

A Maximum of the log-likelihood function

The C++ code for computing the maximum of the log-likelihood function (*loglikelihood.cc*):

```
double call_compute_max_likelihood(TScalarVector Z, double nu, double covlength, double sigma2, TBlockClusterTree* bct,
      TClusterTree* ct, std::vector <double*> vertices, double output[3])
{ gsl_function F;
  int status; iter = 0, max_iter = 200; smy_f_params params ;
  FILE* f1; double size;
  const gsl_multimin_fminimizer_type *T = gsl_multimin_fminimizer_nmsimplex2;
  gsl_multimin_fminimizer *s = NULL; gsl_vector *ss, *x;
  gsl_multimin_function minex_func;
  params.bct = bct; params.ct = ct; params.Z = Z; params.nu = nu;
  params.covlength=covlength; params.sigma2=sigma2; params.vertices=vertices;
  /* Starting point */
  x = gsl_vector_alloc(3); gsl_vector_set (x, 0, nu);
  gsl_vector_set (x, 1, covlength); gsl_vector_set (x, 2, sigma2);
  /* Set initial step sizes to 0.1 */
  ss = gsl_vector_alloc (3);
  gsl_vector_set (ss, 0, 0.02); //for nu
  gsl_vector_set (ss, 1, 0.04); //for theta
  gsl_vector_set (ss, 2, 0.01); //for sigma2
  /* Initialize method and iterate */
  minex_func.n = 3; //dimension
  minex_func.f = &eval_logli;
  minex_func.params = &params;
  s = gsl_multimin_fminimizer_alloc (T, 3); /* optimize in 3-dim space */
  gsl_multimin_fminimizer_set (s, &minex_func, x, ss);
  do{ iter++;
      status = gsl_multimin_fminimizer_iterate(s);
      if (status) break;
      size = gsl_multimin_fminimizer_size (s); //for stopping criteria
      status = gsl_multimin_test_size (size, 1e-5);
      if (status == GSL_SUCCESS) printf ("converged to minimum at \n");}}
  while (status == GSL_CONTINUE && iter < max_iter);
  output[0]= gsl_vector_get(s->x, 0); //nu
  output[1]= gsl_vector_get(s->x, 1); //theta
  output[2]= gsl_vector_get(s->x, 2); //sigma2
  gsl_vector_free(x); gsl_vector_free(ss); gsl_multimin_fminimizer_free (s);
  return status; }
```

Below we list the C++ code, which computes the value of the log-likelihood for given parameters (*loglikelihood.cc*):

```
double eval_logli (const gsl_vector *sol, void* p)
{   pmy_f_params params ;
    double nu = gsl_vector_get(sol, 0);
    double length = gsl_vector_get(sol, 1);
    double sigma2 = gsl_vector_get(sol, 2);
    unique_ptr< TProgressBar > progress( verbose(2) ? new TConsoleProgressBar : nullptr );
    params = (pmy_f_params)p;
    TScalarVector rhs= (params->Z);
    TBlockClusterTree* bct = (params->bct); TClusterTree* ct = (params->ct);
    vector< double * > vertices= (params->vertices);
    double err2=0.0, nugget = 1.0e-4, s = 0.0;
    auto              acc = fixed_prec( 1e-5 ); int dim = 2, N = 0;
    TCovCoeffFn coefffn(length,nu,sigma2,nugget,vertices,ct->perm_i2e(),ct->perm_i2e());
    TACAPlus< real_t >       aca( & coefffn );
    TDenseMatBuilder< real_t > h_builder( & coefffn, & aca );
    // enable coarsening during construction
    h_builder.set_coarsening( false );
    auto A = h_builder.build( bct, acc, progress.get() );
    N=A->cols();
```

```
    auto A_copy = A->copy();
    auto options = fac_options_t( progress.get() );
    options.eval = point_wise; //! Extreme important
    auto A_inv = ldl_inv( A_copy.get(), acc, options );
    for ( int i = 0; i < N; ++i ) {
          const auto v = A_copy->entry( i, i );
          s = s + log(v);}// for
     TStopCriterion sstop( 150, 1e-6, 0.0 );
     TCG           solver( sstop );
     TSolverInfo sinfo( false, verbose( 4 ) );
     auto          solu = A->row_vector();
     solver.solve( A.get(), solu.get(), & rhs, A_inv.get(), & sinfo );
     auto dotp = re( rhs.dot( solu.get() ) );
     auto LL = 0.5*N*log(2*Math::pi<double>())+0.5*s+0.5*dotp;}
```

Rank-k Adaptive Cross Approximation (ACA): An $\mathcal{H}$-matrix contains many sub-blocks, which can be well approximated by low-rank matrices. These low-rank approximations can be computed accurately by truncated singular value decomposition (SVD), but it is very slow. HLIBpro uses the Adaptive Cross Approximation method (ACA) [24] and its improved modifications such as ACA+ and HACA [7,11,13].

Remark A.1. Further optimization of the ACA algorithm can be achieved by truncated SVD. If we suppose that a factorization of the matrix $\boldsymbol{R}=\boldsymbol{A}\boldsymbol{B}^\top$, $\boldsymbol{A}\in\mathbb{R}^{p\times K}$, $\boldsymbol{B}\in\mathbb{R}^{q\times K}$, is found by ACA and that the rank of $\boldsymbol{R}$ is k, $k<K$. Then we can apply the truncated SVD algorithm to compute $\boldsymbol{R}=\boldsymbol{U}\Sigma\boldsymbol{V}^\top$ with $\mathcal{O}((p+q)K^2+K^3)$ FLOPs.

B Multidimensional optimization

There are two ways to compute the MLE estimate $\hat{\boldsymbol{\theta}}=\operatorname{argmax}_\theta \tilde{\mathcal{L}}(\boldsymbol{\theta})$: gradient-based or gradient-free algorithms. Usually, the gradient-free algorithms work well in low-dimensional cases, but they may fail or work very slowly for high-dimensional problems. The advantage of gradient-based algorithms over gradient-free algorithms typically grows for higher-dimensional problems. On the other hand, it might be tricky to compute the gradient when the function is very complicated. In [61],

$$\frac{\partial\mathcal{L}(\theta_i)}{\partial\theta_i}=\frac{1}{2}\operatorname{tr}\left(\boldsymbol{C}^{-1}\frac{\partial\boldsymbol{C}}{\partial\theta_i}\right)-\frac{1}{2}\boldsymbol{Z}^\top\boldsymbol{C}^{-1}\frac{\partial\boldsymbol{C}}{\partial\theta_i}\boldsymbol{C}^{-1}\boldsymbol{Z}, \tag{B.1}$$

and in Eq. 5.9 from [61],

$$\frac{\partial\mathcal{L}(\theta_i)}{\partial\theta_i}=\frac{1}{2}\operatorname{tr}\left(\boldsymbol{C}^{-1}\frac{\partial\boldsymbol{C}}{\partial\theta_i}\right)-\frac{1}{2}\mathbf{Y}^\top\frac{\partial\boldsymbol{C}}{\partial\theta_i}\mathbf{Y}, \tag{B.2}$$

where $\mathbf{Y}$ and $\mathbf{Y}^\top$ are the solutions of $\boldsymbol{C}\mathbf{Y} = \boldsymbol{Z}$, and $\mathbf{Y}^\top \boldsymbol{C} = \boldsymbol{Z}^\top$. For an exponential covariance matrix, we have $\boldsymbol{\theta} = (\theta_1, \theta_2, \theta_3) = (\nu, \ell, \sigma^2) = (0.5, \ell, \sigma^2)$:

$$\frac{\partial \boldsymbol{C}(\boldsymbol{\theta})}{\partial \theta_2} = \frac{\partial \boldsymbol{C}(\boldsymbol{\theta})}{\partial \ell} = \frac{\partial}{\partial \ell} \sigma^2 \exp\left(\frac{-r}{\ell}\right) = \frac{\sigma^2 r}{\ell^2} \exp\left(\frac{-r}{\ell}\right), \tag{B.3}$$

$$\frac{\partial \boldsymbol{C}(\boldsymbol{\theta})}{\partial \theta_3} = \frac{\partial \boldsymbol{C}(\boldsymbol{\theta})}{\partial \sigma^2} = \frac{\partial}{\partial \sigma^2} \sigma^2 \exp\left(\frac{-r}{\ell}\right) = \exp\left(\frac{-r}{\ell}\right). \tag{B.4}$$

Finally, Equations B.1-B.4 should be substituted into the GSL optimization procedures.