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

Alexander Litvinenko and to be defined

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

Abstract. The main goal of this article is to demonstrate how to use very fast and very efficient parallel hierarchical matrix library HLIBpro for solving many tasks which appear, for instance, in spatial statistics, geostatistics, and in kriging. We describe the HLIBCov package, which is an extension of the HLIBpro library for approximating large covariance matrices, computing the matrix inverse, the Cholesky and LU factorisations, different matrix norms, and maximizing likelihood functions. We show that an approximate Cholesky factorization of a dense matrix of size $2M \times 2M$ can be computed on a modern multi-core desktop in few minutes. Further, HLIBCov is used for estimating the unknown parameters such as the covariance length, variance and smoothness parameter of a Matérn covariance function by maximizing the joint Gaussian log-likelihood function. The computational bottleneck here is expensive linear algebra arithmetics due to large and dense covariance matrices. Therefore covariance matrices are approximated in the hierarchical ($\mathcal{H}$-) matrix format with computational cost $\mathcal{O}(k^2 n \log^2 n/p)$ and storage $\mathcal{O}(kn\log n)$, where the rank k is a small integer (typically $k < 25$), p the number of cores and n the number of locations on a fairly general mesh. We demonstrate a synthetic example, where the true values of known parameters are known. For reproducibility we provide the C++ code, the R-interface for calling C++ procedures from R, and the documentation.

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

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: 4GB 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 implementing parallel algorithms for Hierarchical matrices. It is freely available in binary form for academic purposes. HLIBpro algorithms design for 1, 2 and 3 - dimensional problems.

1 Introduction

This paper is complementary to the paper [49].

Novelty of this work. In this paper we a) use parallel hierarchical ($\mathcal{H}$-) matrices for approximating the joint Gaussian log-likelihood function with computational complexity $\mathcal{O}(k^2 n \log^2 n)$, where n is the number of measurements; $k \ll n$ is the maximal $\mathcal{H}$-matrix rank, which defines the quality of the approximation; b) compute maximum likelihood

estimates ow with other words estimate unknown parameters of the covariance function; c) provide some examples how to call efficient HLIBPro C++ routines from R.

Parameter estimation. Problem settings. Let n be the number of spatial measurements $\mathbf{Z}$ located irregularly across a given geographical region at locations $\mathbf{s} := \{\mathbf{s}_1,\ldots,\mathbf{s}_n\} \in \mathbb{R}^d$, $d \geq 1$. These data are frequently modeled as a realization from a stationary Gaussian spatial random field. Specifically, we let $\mathbf{Z} = \{Z(\mathbf{s}_1),\ldots,Z(\mathbf{s}_n)\}^\top$, where $Z(\mathbf{s})$ is a Gaussian random field. Then, we assume that $\mathbf{Z}$ 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 unknown parameters $\boldsymbol{\theta}$ we maximize the Gaussian log-likelihood function:

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

where $\mathbf{C}(\boldsymbol{\theta})_{ij} = C(\mathbf{s}_i - \mathbf{s}_j;\boldsymbol{\theta})$, $i,j = 1,\ldots,n$. 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 log-determinant of the n-by-n dense covariance matrix $\mathbf{C}(\boldsymbol{\theta})$. Indeed, this 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 $\tilde{\mathcal{L}}(\boldsymbol{\theta};k)$:

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

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

The argument k in $\tilde{\mathcal{L}}(\boldsymbol{\theta};k)$ indicates that the fixed rank strategy, i.e., each sub-block in the covariance matrix has a rank equal to k or smaller, is used. 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 Brent-Dekker method [14,57]. We note that maximization of the log-likelihood function is an ill-posed problem, since even very small perturbations in the covariance matrix $\mathbf{C}(\boldsymbol{\theta})$ may result in large perturbations in the log-determinant and in the log-likelihood. A possible remedy is to take a higher rank k.

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

1. The class of $\mathcal{H}$-matrices is wide, for instance, it includes the classes of low-rank and sparse matrices;
2. $\mathbf{C}(\boldsymbol{\theta})^{-1}$, $\mathbf{C}(\boldsymbol{\theta})^{1/2}$, $\det\mathbf{C}(\boldsymbol{\theta})$, Cholesky decomposition, the Schur complement and, many others can be computed in the $\mathcal{H}$-matrix format [28];
3. The $\mathcal{H}$-matrix technique is well-studied and has solid theory, many examples, and multiple sequential and parallel implementations, no specific MPI or OpenMP knowledge is needed;
4. The $\mathcal{H}$-matrix accuracy is controllable by the rank, k. The full rank gives an exact representation;
5. The $\mathcal{H}$-matrix format preserves the structure (in contrast, for instance, to sparse matrices) after the Cholesky decomposition and the inverse have been computed, although with possibly a larger rank.

Related work. Stationary covariance functions, which have block Toeplitz or block circulant structure can be resolved with the Fast Fourier Transform (FFT) with the computing cost $\mathcal{O}(n\log n)$ [22]. 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: for example, covariance tapering [21, 36, 65], likelihood approximations in both the spatial [69] and spectral [19] domains, latent processes such as Gaussian predictive processes [5] and fixed-rank kriging [16], and Gaussian Markov random-field approximations [20, 43, 62, 63]; see [70] for a review. Each of these methods has its strengths and drawbacks [67, 68, 71]. Some other ideas include the nearest-neighbor Gaussian process models [17], multiresolution Gaussian process models [55], equivalent kriging [39], multi-level restricted Gaussian maximum likelihood estimators [15], and hierarchical low-rank approximations [35]. Bayesian approaches to identify unknown or uncertain parameters could be also applied [48, 52, 52, 53, 56, 59–61].

Previous results [28] show that the $\mathcal{H}$-matrix technique is very stable when approximating the covariance matrix itself [3, 4, 11, 32, 37, 46, 64], its inverse [3, 9], its Cholesky decomposition [7, 8], and the conditional covariance matrix [28, 44, 45].

Recently, the 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, which allowed the evaluation of the log-likelihood and its gradient with computational complexity $\mathcal{O}(n^{3/2})$, where n was the

number of observations; the method relied on the recursive skeletonization factorization procedure [33, 50]. However, the consequences of the approximation on the maximum likelihood estimator were not studied.

In [11] the authors computed the exact solution of Gaussian process regression via replacing the kernel matrix by a data-sparse approximation, called an $\mathcal{H}^2$-matrix technique. An $\mathcal{H}^2$-matrix approximation has $\mathcal{O}(kn)$ computational complexity and storage cost, where k is a parameter controlling the accuracy of the approximation.

Memory efficient kernel approximation was offered by [66]. The authors observe that the structure of shift-invariant kernels changes from low-rank to block-diagonal (without any low-rank structure) when varying the scale parameter. Based on this observation, the authors propose a new kernel approximation algorithm, which they call Memory Efficient Kernel Approximation. That approximation considers both low-rank and clustering structure of the kernel matrix. They also show that the resulting algorithm outperforms state-of-the-art low-rank kernel approximation methods regarding speed, approximation error, and memory usage.

In [4] the authors estimated the covariance matrix of a set of normally distributed random vectors. To overcome large numerical issues in high-dimensional regime, they computed (dense) inverses of sparse covariance matrices by the usage of $\mathcal{H}$-matrices.

In [32] the authors offered methods for rapid computation of separable expansions for the approximation of random fields. In particular, they suggested the pivoted Cholesky method and provided a-posteriori error estimate in the trace norm. Low-rank tensor techniques in kriging are introduces in [54]. BigQUIC method for sparse inverse covariance estimation for a million variables was introduced in [34]. This method can solve 1 million dimensional ℓ_1 regularized Gaussian MLE problems. In [64] the authors applied $\mathcal{H}$-matrices to linear inverse problems in large-scale geostatistics. The $\mathcal{H}^2$ matrix technique for solving large-scale stochastic linear inverse problems with applications in the 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 enables the evaluation of the determinant.

The structure of the paper. In Section 2 we introduce the methodology and algorithms. We review the $\mathcal{H}$-matrix technique, 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 software installation details, procedures of the HLIBCov code and the algorithm for parameter estimation. Estimation of

unknown parameters is reported in Section 5. The best practices are listed in Section 6. We end the paper with a conclusion in Section 8. The auxiliary $\mathcal{H}$-matrix details are provided in the Appendix B.

2 Methodology and algorithms

2.1 Matérn covariance functions

The class of Matérn covariance functions [26] is 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 [51], with one parameter regulating the smoothness of the underlying spatial random field [31]. Matérn functions for different values of parameters: variance (on the top) σ^2, covariance length and ν (on the bottom) are shown in Fig. 1. By watching on the shapes of the Matérn functions we can judge about singularities and the decay of singular values. For instance, one can see that singularity is located on the diagonal $x=y$. Additionally, one sees that the larger ν is, the smoother is the covariance function and the faster is decay of singular values (the reader will see convergence graphics later in Section 3).

Let $\mathbf{s}$ and $\mathbf{s}'$ are any two spatial locations, and $\mathbf{h}:=\|\mathbf{s}-\mathbf{s}'\|$. The Matérn class of covariance functions is defined as

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

with $\boldsymbol{\theta}=(\sigma^2,\ell,\nu)^{\top}$; where 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 the modified Bessel function of the second kind of order ν. 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 [58] for $\nu=p+1/2$, where p is a non-negative integer. In this case the covariance function is a product of an exponential and a polynomial of order p [1]:

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

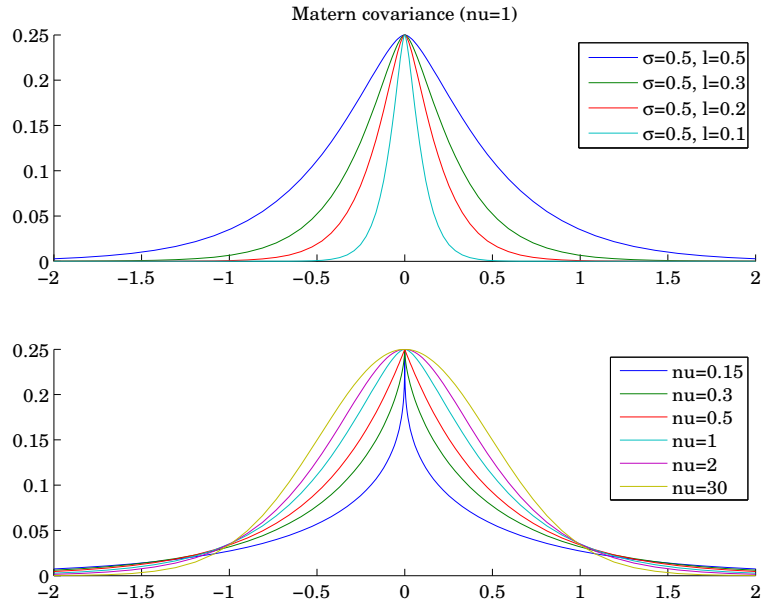

Figure 1: Matérn function for different parameters (computed in sglib [72]).

2.2 Hierarchical matrices

Hierarchical matrices have been described in detail [24, 27–30, 44]. Applications of the $\mathcal{H}$-matrix technique to the covariance matrices can be found in [2, 3, 11, 32, 37, 38, 47].

Originally the $\mathcal{H}$-matrix technique was introduced by W. Hackbusch in 1999 for the approximation of stiffness matrices and their inverses coming from partial differential and integral equations [13, 24, 27]. The key idea of the $\mathcal{H}$-matrix technique is to divide the initial matrix into sub-blocks in a clever way and then to approximate sub-blocks, which are far away from the singularity by low-rank matrices. The admissibility condition (criteria) is used to divide a given matrix into sub-blocks (Fig. 4) and define which sub-blocks can be approximated well by low-rank matrices and which not. There are many different admissibility criteria (see Fig. 2). The typical criteria are: the strong, weak and domain decomposition-based [28]. The user can also develop his own admissibility criteria, which may depend, for instance, on the covariance length.

Figure 2 shows three examples of $\mathcal{H}$-matrices with three different admissibility criteria: (left) standard, (middle) domain-decomposition based and (right) weak. The numbers inside sub-blocks show the matrix ranks. The "stairs" inside sub-blocks demonstrate the decay of eigenvalues in a log-scale. Dark (or red) blocks indicate fully-populated blocks. The maximal size of dark blocks is regulated by the parameter nmin, which usu-

ally depends on the available DRAM and cache memory, architecture and the problem size. Parameter nmin regulates how deep the hierarchical subdivision into sub-blocks is and takes typically values from the set $\{32,64,128\}$.

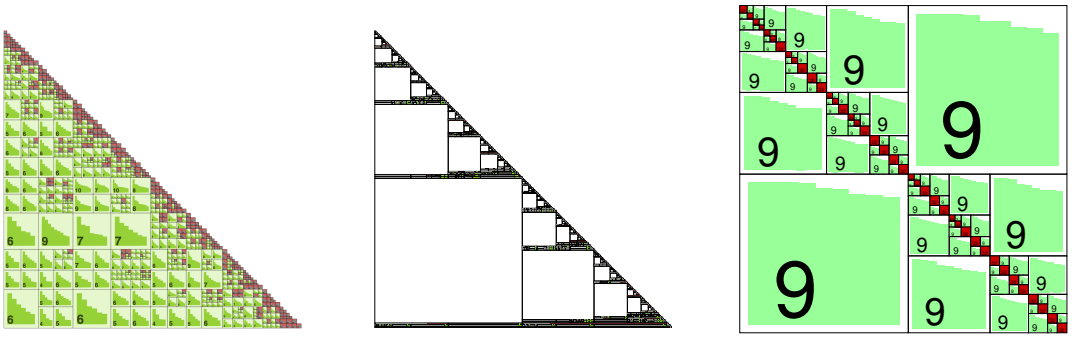

Figure 2: Examples of $\mathcal{H}$-matrices with three different admissibility criteria: (left) standard, (middle) domain-decomposition based and (right) weak. Matrices are taken from different applications and illustrate only the diversity of block partitioning.

Remark 2.1. It is seldom that one can see a visualization of an $\mathcal{H}$-matrix of the size larger than $n = 10{,}000$, although in the computations n can be few millions. The reason is that the eps/pdf/ps files for $n > 10{,}000$ will be very large or of a bad resolution.

In general, covariance matrices are dense and, therefore, require $\mathcal{O}(n^2)$ units of memory for the 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 (on the left), its hierarchical Cholesky factor $\tilde{\mathbf{L}}$ (in the middle), and zoomed left-upper cornel of the left matrix (on the 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 and the adaptive rank arithmetics are used with the relative accuracy in each sub-block $1e-5$. The green blocks indicate low-rank matrices and the number inside of each block indicates the maximal rank used in this block. Very small dark-red blocks are dense blocks, they are computed without approximation. In this example dense blocks are located only on the diagonal. The maximal size of dense blocks

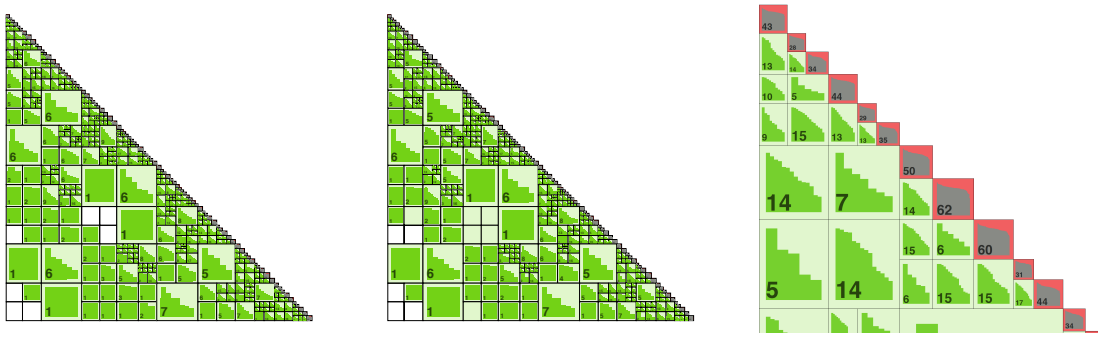

Figure 3: Examples of $\mathcal{H}$-matrix approximations of the exponential covariance matrix (left), its hierarchical Cholesky factor $\tilde{\mathbf{L}}$ (middle), and zoomed left-upper cornel of the left matrix (right), $n=4000$, $\ell=0.09$, $\nu=0.5$, $\sigma^2=1$. The adaptive rank arithmetic 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.

is a parameter nmin, which is taken 64. The "stairs" inside green blocks indicate decay of singular values in log-scale. The faster is this decay the better, i.e., the smaller rank can be used in this block and the smaller the approximation error. Completely white (or empty) blocks are zero blocks. They contain only zero entries.

After the decomposition of the matrix into sub-blocks is done, one should compute the low-rank approximations (green blocks). For this purpose the Adaptive Cross Approximation (or similar techniques like ACA+, HACA) algorithm [6,10,12,13,23] is used, which performs the approximations with a linear complexity $\mathcal{O}(kn)$, where n is the size of the sub-block.

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

Definition 2.1. We let I be an index set. A hierarchical decomposition of I we will call the cluster tree T_I (see Fig. 4 and in Appendix). A hierarchical division of the index set product, $I\times I$, into sub-blocks (Fig. 4, right) is called block cluster tree $T_{I\times I}$. The set of

$\mathcal{H}$-matrices is

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

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

Blocks that satisfy the admissibility condition can be approximated by low-rank matrices; see [27]. An $\mathcal{H}$-matrix approximation of $\mathbf{C}$ is denoted by $\tilde{\mathbf{C}}$. Figure 4 demonstrates

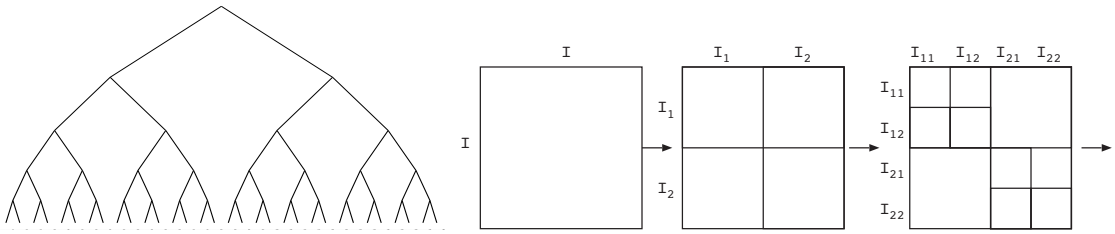

Figure 4: (left) A cluster tree T_I and (right) a block cluster tree $T_{I\times I}$ (matrix decomposition into sub-blocks).

two important steps in building an $\mathcal{H}$-matrix, namely, constructing the cluster and block-cluster trees. The first step (in the left) is hierarchical decomposition of the index set I into sub-sets. The maximal size of the smallest sub-index set is regulated by parameter nmin. The picture on the right demonstrates constructing the block-cluster tree $T_{I\times J}$ (in this example $I = J$). $T_{I\times J}$ is the cartesian product of two index sets I and I. Additionally, the admissibility condition decides should a certain block be divided further or not. For instance, sub-block $I_1\times I_1$ is divided further and sub-block $I_1\times I_2$ not.

Remark 2.3. (Fixed and adaptive rank strategies) For each sub-block we can either fix the accuracy ε or the maximal approximative rank k. In the first case we speak about the *adaptive rank strategy*, i.e., where 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 maximal rank k. The last variant does not provide us with the reliable estimate, but makes 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. 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.

In Fig. 5 (left) we demonstrate boxplots for different ranks $k \in \{3,7,9,12\}$. In the experiment we run 100 replicates. One can see that boxplots for $k \in \{7,9,12\}$ are not changing, therefore there is no point to increase the rank, i.e. rank 7 is sufficient. Fig. 5 shows that the larger is the number of observations the better is estimation of the unknown parameter.

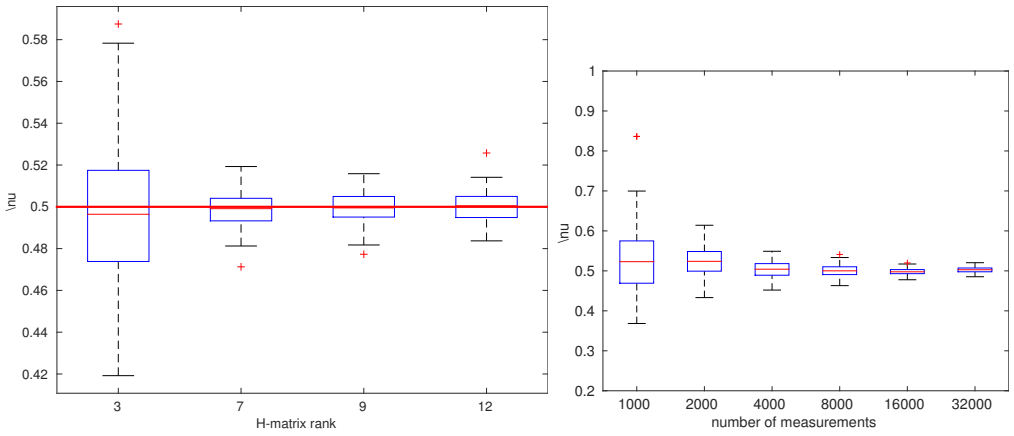

Figure 5: (left) Dependence of the boxplots for ν on the $\mathcal{H}$-matrix rank, $n = 16{,}000$; (right) Convergence of boxplots for ν with increasing n, 100 replicates.

In Fig. 6 (left) we demonstrate the shape of the negative log-likelihood, of the quadratic form and of the log-determinant $\log(|C|)$. The true value of the parameter $\theta^* = \ell = 12$. In Fig. 6 (right) one can see three log-likelihood functions - the exact one and two others computed with different ranks 7 and 17. One can see, that all three plots look very similar.

2.3 Parallel hierarchical matrix technique

We used the parallel $\mathcal{H}$-matrix library HLIBpro [25,40–42] to approximate the Matérn covariance matrix, to compute a Cholesky factorization, solve a linear system, calculate the determinant and a quadratic form. HLIBpro is fast, robust and efficient; see theoretical parallel complexity in Table 6. Here $|V(T)|$ denotes the number of vertices, $|L(T)|$ the number of leaves in the block-cluster tree $T = T_{I \times I}$, and n_{min} the size of a block when we stop further division into sub-blocks (see Section 2.2). Usually $n_{min} = 32$ or 64, since a very deep hierarchy slows down computations.

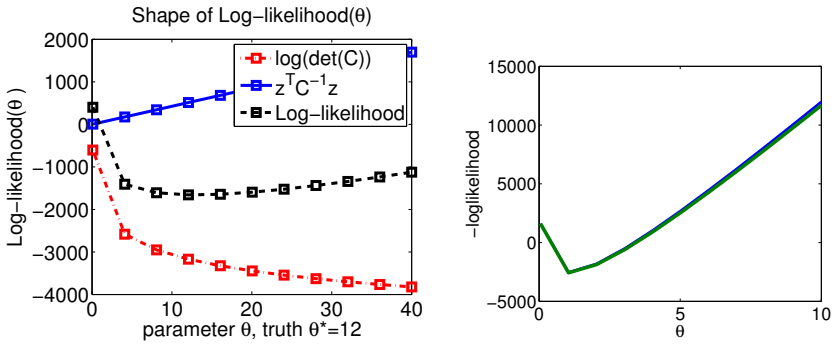

Figure 6: (left) Shape of the log-likelihood function; (right) Three negative log-likelihood functions: the exact, and computed with $\mathcal{H}$-matrix ranks 7 and 17. One can see that even with rank 7 one can achieve very accurate results.

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

Operation	Parallel Complexity [40] (Shared Memory)
build $\tilde{\mathbf{C}}$	$\frac{\mathcal{O}(n\log n)}{p}+\mathcal{O}(\lvert V(T)\backslash L(T)\rvert)$
store $\tilde{\mathbf{C}}$	$\mathcal{O}(kn\log n)$
$\tilde{\mathbf{C}}\cdot\mathbf{z}$	$\frac{\mathcal{O}(kn\log n)}{p}+\frac{n}{\sqrt{p}}$
$\alpha\tilde{\mathbf{A}}\oplus\beta\tilde{\mathbf{B}}$	$\frac{\mathcal{O}(n\log n)}{p}$
$\alpha\tilde{\mathbf{A}}\odot\tilde{\mathbf{B}}\oplus\beta\tilde{\mathbf{C}}$	$\frac{\mathcal{O}(n\log n)}{p}+\mathcal{O}(C_{sp}(T)\lvert V(T)\rvert)$
$\tilde{\mathbf{C}}^{-1}$	$\frac{\mathcal{O}(n\log n)}{p}+\mathcal{O}(nn_{min}^2)$, $n_{min}\approx 64$
$\mathcal{H}$-Cholesky $\tilde{\mathbf{L}}$	$\frac{\mathcal{O}(n\log n)}{p}+\mathcal{O}(\frac{k^2n\log^2 n}{n^{1/d}})$, $d=1,2,3$
determinant $\lvert\tilde{\mathbf{C}}\rvert$	$\frac{\mathcal{O}(n\log n)}{p}+\mathcal{O}(\frac{k^2n\log^2 n}{n^{1/d}})$, $d=1,2,3$

3 Memory storage, computing time and convergence

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

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

In Tables 2 and 3 we show dependence of KLD and two matrix errors on the $\mathcal{H}$-matrix rank k for Matérn covariance function with parameters $\ell=\{0.25,0.75\}$ and $\nu=\{0.5,1.5\}$,

computed on the domain $\mathcal{G}=[0,1]^2$. All errors are under control, besides the last column. The ranks $k=10,12$ are not sufficient to approximate the inverse, the error $\|\boldsymbol{C}(\tilde{\boldsymbol{C}})^{-1}-\mathbf{I}\|_2$ is large. A 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 $\tilde{\mathbf{L}}$ or the $\mathcal{H}$-inverse produces NaN (not a number) and terminates. One of the possible reasons is that some diagonal elements can become very close to zero and their inverse is not defined. This may happen when, for instance, two locations are very close to each other, and, as a result, two columns (rows) are linear dependent. To avoid such cases, one should do some preprocessing with the available data (remove the same locations). Very often to stabilize numerical calculations the nugget $\tau^2\mathbf{I}$ is added to the main diagonal (see more in Section 5.2), i.e., $\tilde{\mathbf{C}}:=\tilde{\mathbf{C}}+\tau^2\mathbf{I}$. Adding a nugget, we "push" all singular values away from zero. In Tables 2 and 3 the nugget is equal to zero.

k	KLD		$\|\boldsymbol{C}-\tilde{\boldsymbol{C}}\|_2$		$\|\boldsymbol{C}(\tilde{\boldsymbol{C}})^{-1}-\mathbf{I}\|_2$	
	$\ell=0.25$	$\ell=0.75$	$\ell=0.25$	$\ell=0.75$	$\ell=0.25$	$\ell=0.75$
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, Matérn covariance with parameters $\ell=\{0.25,0.75\}$ and $\nu=0.5$, domain $\mathcal{G}=[0,1]^2$, $\|C_{(\ell=0.25,0.75)}\|_2=\{212,568\}$.

k	KLD		$\|\boldsymbol{C}-\tilde{\boldsymbol{C}}\|_2$		$\|\boldsymbol{C}(\tilde{\boldsymbol{C}})^{-1}-\mathbf{I}\|_2$	
	$\ell=0.25$	$\ell=0.75$	$\ell=0.25$	$\ell=0.75$	$\ell=0.25$	$\ell=0.75$
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, Matérn covariance with parameters $\ell=\{0.25,0.75\}$ and $\nu=1.5$, domain $\mathcal{G}=[0,1]^2$, $\|C_{(\ell=0.25,0.75)}\|_2=\{720,1068\}$.

Figure 7 shows that the $\mathcal{H}$-matrix storage cost is almost not changing for different parameters $\ell = \{0.15,...,2.2\}$ (on the left) and $\nu = \{0.3,...,1.3\}$ on the right. The computational domain is $[32.4,43.4] \times [-84.8 - 72.9]$ with $n = 2{,}000$.

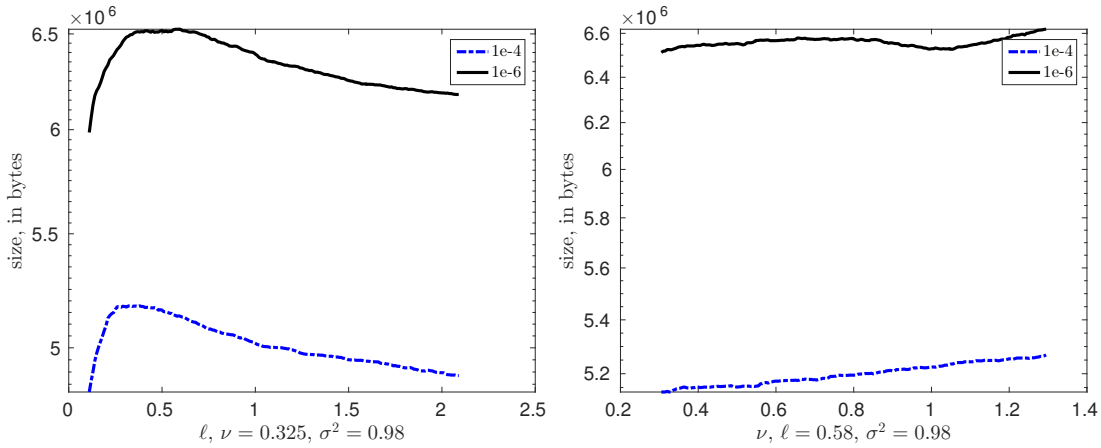

Figure 7: (left) Dependence of the matrix size on (left) the covariance length ℓ; (right) the smoothness ν for two different accuracies in $\mathcal{H}$-matrix sub-blocks $\varepsilon = \{1e-4, 1e-6\}$, $n = 2{,}000$ locations in the domain $[32.4,43.4] \times [-84.8 - 72.9]$.

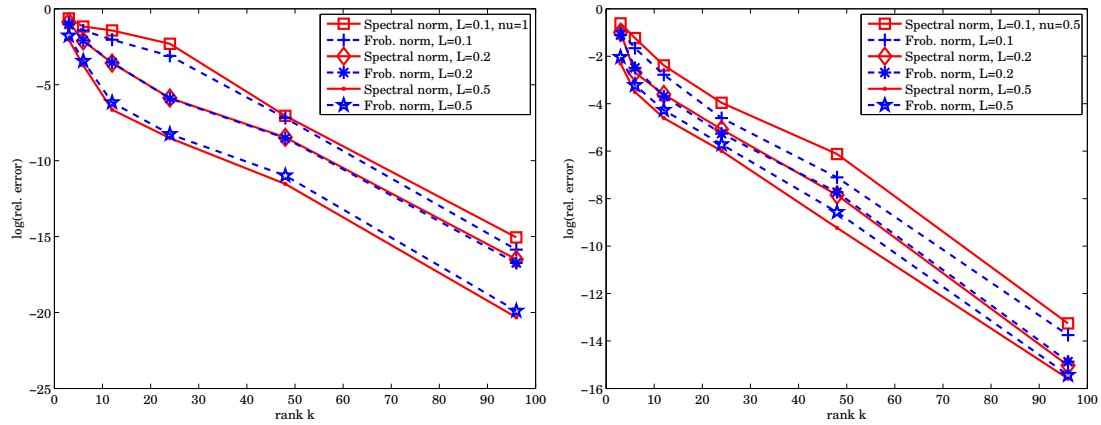

Figure 8: Convergence of the $\mathcal{H}$-matrix approximation errors for covariance lengths $\{0.1,0.2,0.5\}$; (left) $\nu = 1$ and (right) $\nu = 0.5$.

In Fig. 8 we plotted convergence of $\|\mathbf{C} - \tilde{\mathbf{C}}\|$ in the Frobenius and spectral norms vs. the rank k for different smoothness parameters and covariance lengths. In Fig. 9 we plotted $\|\mathbf{C} - \tilde{\mathbf{C}}\|_2$ vs. rank k for different ν and covariance lengths.

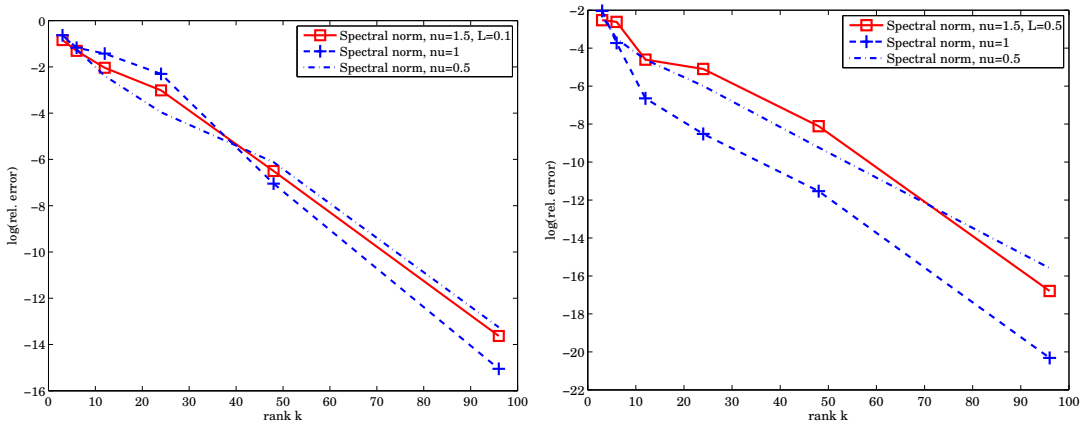

Figure 9: Convergence of the $\mathcal{H}$-matrix approximation errors for $\nu = \{0.5, 1, 1.5\}$; (left) covariance length 0.1 and (right) covariance length 0.5.

4 Software Installation and Numerical Examples

4.1 Introduction to HLIBCov and HLIBpro

This Section contains shortened information taken from `www.hlib.com`. The software library HLIBpro is a robust parallel C++ implementation of the $\mathcal{H}$-matrix technique and $\mathcal{H}$-matrix arithmetics, developed by Ronald Kriemann [40]. HLIBpro supports both the shared and distributed memory architectures, but in this work we use only the shared memory version. Intel's Threading Building Blocks (TBB) are used for parallelisation on shared memory systems. HLIBpro is free for academic purposes, is distributed as a compiled code (no source code is available). Originally HLIBpro was developed for solving FEM and BEM problems [25, 42]. 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, first, the reader needs to install HLIBpro. Since HLIBcov is just few C++ procedures (only one file loglikelihood.cc), the applicability of HLIBCov is limited only by the applicability of HLIBpro. To install HLIBpro on MacOS and on Windows we refer to `www.HLIBpro.com` for further details. Additionally, the HLIBCov requires the GNU Scientific Library (GSL). We use GSL-1.16, the installation steps are described on `https://www.gnu.org/software/gsl/`. The GSL library provides maximization algorithms and Bessel functions. The reader can easily replace GSL on his own optimization library. The Bessel functions are also avail-

able 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 default settings/paths, there is no need to change *SConstruct* file. To compile HLIBCov, add the following line to */examples/SConscript*

```
$ examples.append(cxxenv.Program('loglikelihood.cc'))
$ cd ..
$ scons
$ cd examples/
$ ./loglikelihood
```

Input of the HLIBCov

The input contains in the first line the total number of locations N. Lines 2,...,N+1 contain the coordinates x_i, y_i, and the measurement value. An example

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

The HLIBpro requires neither a list of finite elements nor a list of edges. On the github we provide few input files examples of different size.

Output of the HLIBCov

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

Below we list the C++ code, which reads the data and initializes all 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 an email with your user login name and the machine name, on which you plan to run the code, to Ronald Kriemann rok@mis.mpg.de and request a license file. The machine name can be received with the terminal command *hostname*. The received license file must be placed either in the directory where the final program will be executed, e.g. in the examples/ subdirectory, or in the directory pointed to by the environment variable HLIBpro_LIC_PATH. You can add this variable to the system paths

```
$ export hlibpro$\_$LIC_PATH=<path_to_HLIBpro.lic>
```

5 Numerical experiments with synthetic data

We performed numerical experiments with simulated data to recover the true values of the parameters of the Matérn covariance matrix, 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 $\mathbf{Z}_0$ with $n_0 = 2 \cdot 10^6$ locations and randomly sample n points from it. Note, that if locations are very close to each other, then the covariance matrix could be singular or it will be difficult to compute the Cholesky factorization.

To generate the random data $\mathbf{Z}_0 \in \mathbb{R}^{n_0}$ we compute the $\mathcal{H}$-Cholesky factorization of $\mathbf{C}(1.0,0.5,1.0) = \tilde{\mathbf{L}}\tilde{\mathbf{L}}^\top$. Then we evaluate $\mathbf{Z}_0 = \tilde{\mathbf{L}}\boldsymbol{\xi}$, where $\boldsymbol{\xi} \in \mathbb{R}^{n_0}$ is a normal vector with zero mean and unit variance. We generate $\mathbf{Z}_0$ only once. After that 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. One can see that the variance (or uncertainty) is decreasing with increasing of n. The green line indicates the true values.

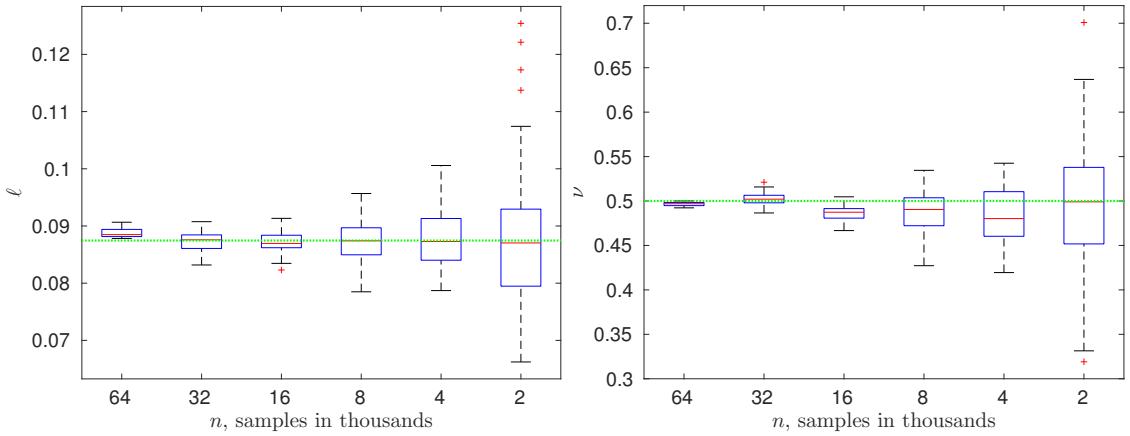

Figure 10: Synthetic data with known parameters $(\ell^*,\nu^*,\sigma^{2^*}) = (0.5,1,0.5)$. Boxplots for ℓ and ν for $n = 1{,}000 \times \{64,32,...,4,2\}$, 100 replicates.

To identify all three parameters simultaneously a three-dimensional optimization problem is solved. The maximal number of iterations is set to 200, the residual to 10^{-6}. The behavior and accuracy of the boxplots depend on the hierarchical 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 in the maximization algorithm, and the maximization algorithm. All replicates of $\mathbf{Z}$ are sampled from the same generated vector of size $n_0 = 2 \cdot 10^6$.

In Table 4 we demonstrate the almost linear storage cost (columns 3 and 6) and com-

puting time (columns 2 and 5).

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 $\tilde{\mathbf{C}}$ and $\tilde{\mathbf{L}}$ is 10^{-5}.

n	$\tilde{\mathbf{C}}$			$\tilde{\mathbf{L}}\tilde{\mathbf{L}}^\top$		
	comp. time sec.	size MB	kB/dof	comp. time sec.	size MB	$\|\mathbf{I} - (\tilde{\mathbf{L}}\tilde{\mathbf{L}}^\top)^{-1}\tilde{\mathbf{C}}\|_2$
32,000	3.3	162	5.1	2.4	172.7	$2.4 \cdot 10^{-3}$
128,000	13.3	776	6.1	13.9	881.2	$1.1 \cdot 10^{-2}$
512,000	52.8	3420	6.7	77.6	4150	$3.5 \cdot 10^{-2}$
2,000,000	229	14790	7.4	473	18970	$1.4 \cdot 10^{-1}$

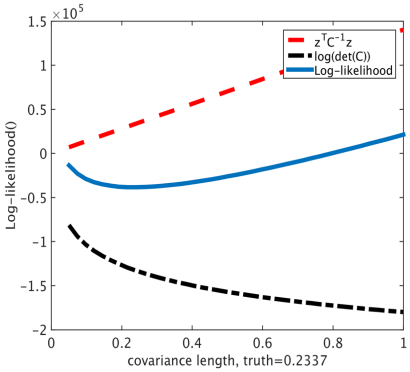

Figure 11: Dependence of the negative log-likelihood and its ingredients on parameter ℓ, parameter $\nu = 0.5$ is fixed, the true value of covariance length is 0.2337. Exponential covariance, $n = 66049$, rank $k = 16$, $\sigma^2 = 1$.

To get an idea about the shape of the log-likelihood function, and about its components we draw Fig. 11. This picture helps us to understand the behavior of the iterative optimization method, contribution of the log-determinant, of the quadratic functional. One can see that the log-likelihood is almost flat and it may be necessary a lot of iteration steps 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 corresponds 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 diagonal values of $\tilde{\mathbf{C}}$ are very close to zero, then numerical algorithms can not compute the $\mathcal{H}$-Cholesky factor $\tilde{\mathbf{L}}$ or the inverse. They produce `nan` (means "not a number") or the error message "division by zero". A possible remedy is to add a diagonal matrix with small positive numbers. This trick will increase all singular values and move them away from zero. Of course, adding a nugget, we change the original matrix $\tilde{\mathbf{C}}:=\tilde{\mathbf{C}}+\tau^2\mathbf{I}$. Below we analyze how the loglikelihood function, as well as its maximum are changing, when a nugget is added.

Assume $|\tilde{\mathbf{C}}|\neq 0$. It is known [18], that for a small perturbation matrix $\mathbf{E}$ it is hold

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

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

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

From the last equation one can see that if $a\leq\|\tilde{\mathbf{C}}^{-1}\|\leq b$, where a, b are some positive constants, then the relative error will be of order τ^2. Figure 12 (left) demonstrates three negative log-likelihood functions. One is computed with the nugget 0.01, another with 0.005 and the third with 0.001. As one can see, for this particular example, the behavior of likelihood is preserved, and the minimum is not changing (or changing very slightly). Figure 12 (right) is just a zoomed version of the left picture.

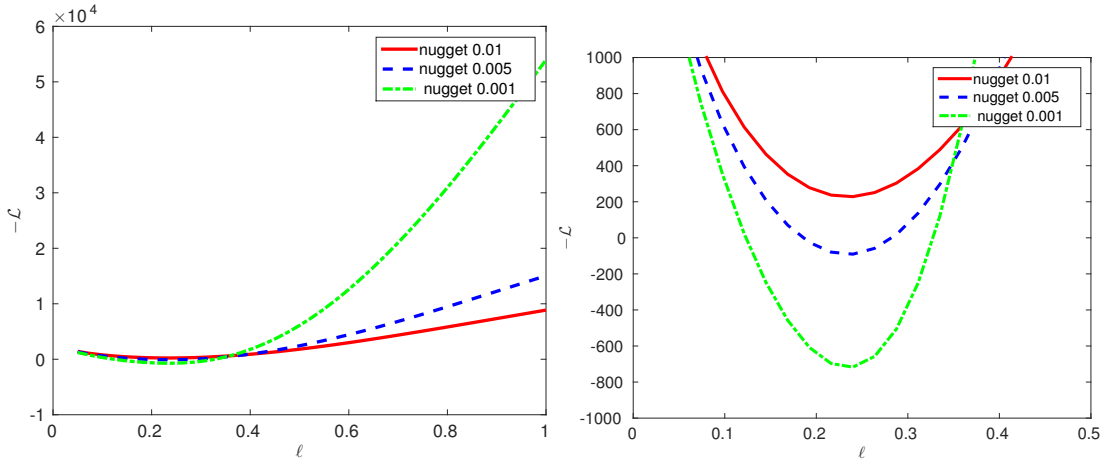

Figure 12: (left) Dependence of the log-likelihood on parameter ℓ with nuggets ($\{0.01, 0.005, 0.001\}$) for Gaussian covariance. (right) Zoom of the left figure near minimum; $n = 2000$ random points from moisture example, rank $k = 14$, $\sigma^2 = 1$.

6 Best practices (HLIBpro)

In this section we list our recommendations and warnings.

1. Initialize variables and the license with `INIT()`;
2. For practical computations use the adaptive rank arithmetics.
3. For the input it is enough to define a file with 3 columns: location coordinates (x, y) and the observed value, no triangles, no edges are required;
4. If two locations are coincide or 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. A remedy is to improve the quality of locations.
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 $\mathbf{Z}$, computed on a grid, for multiplication with the hierarchical matrix $\tilde{\mathbf{L}}^{-1}$ (or $\tilde{\mathbf{C}}^{-1}$) use `ct->perm_e2i()->permute(&Z)` To find to which grid node corresponds 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 programming and makes it much more secure.

7. By default, the $\mathcal{H}$-Cholesky or $\mathcal{H}-\mathbf{LU}$ factorizations are always DAG based. You can turn this 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 2D and 3D, there are only very minor changes in HLIBCov to move from 2D locations to 3D. Replace dim=2 to dim=3 in
   `TCoordinate coord(vertices, dim);`
   Then add `>> z` to `in >> x >> y>> z >> v;`

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

7 Call C++ routines from R language

In this section we demonstrate how to call very efficient and fast HLIBpro C++ procedures form the R environment. We implement a list of R-functions, which build an interface to the corresponding HLIBpro functions. Everything what is needed is to install HLIBpro and compile it as a shared object (.so) library. This shared object file can be opened in the R-environment and the required C++ functions can be called from R. The $\mathcal{H}$-matrix data format is rather complicated data structure (class) in HLIBpro. Therefore $\mathcal{H}$-matrix objects (or pointers on them) is neither an input, not the output parameters. Instead, we decided that input parameters for the HLIBpro C++ routines will be: a vector (array) of locations, a vector of observations $\mathbf{Z}$, etc. The triangulation (list of triangles/edges) is not needed. The output parameters will be scalar values or a vector, for example, the determinant, the trace, a norm, the result of the matrix-vector product, an approximation error etc. $\mathcal{H}$-matrix objects will exist only in the C++ environment.

8 Conclusion

We extended functionality of the parallel $\mathcal{H}$-matrix library HLIBpro for application in spatial statistics and in parameters inference. This new extension allows us to work with large covariance and precision matrices. The first novelty is that we approximated the

Table 6: List of R functions, which call C++ routines from HLIBpro

Name	Functionality
HM_Matern_approx()	compute $\mathcal{H}$-matrix approximation of a Matern cov. matrix
HM_exp_approx()	compute $\mathcal{H}$-matrix approximation of a exp. cov. matrix
HM_Gaussian_approx()	compute $\mathcal{H}$-matrix approximation of a Gaussain cov. matrix
HM_chol()	compute $\mathcal{H}$-Cholesky factorisation, $\tilde{\mathbf{C}} = \tilde{\mathbf{L}}\tilde{\mathbf{L}}^T$
HM_inverse()	compute $\mathcal{H}$-matrix inverse, $\tilde{\mathbf{C}}^{-1}$
HM_norm2()	compute spectral norm of matrix $\tilde{\mathbf{C}}$
HM_fro()	compute Frobenius norm of matrix $\tilde{\mathbf{C}}$
HM_KLD()	compute KL divergence for Gaussian case
HM_MM()	compute matrix-matrix product, $\mathbf{C} := \beta\mathbf{C} + \alpha\mathbf{AB}$
HM_scale()	scale a matrix, $\mathbf{B} := \beta\mathbf{B}$
HM_nugget()	add a nugget, $\mathbf{B} := \mathbf{B} + \tau^2\mathbf{I}$
HM_add()	compute sum of two matrices, $\mathbf{C} := \beta\mathbf{B} + \alpha\mathbf{A}$
HM_MV()	compute matrix-vector product, $\mathbf{w} := \beta\mathbf{w} + \alpha\mathbf{Av}$
HM_loglikelihood()	compute joint Gaussian loglikelihood
HM_determinant()	compute matrix determinant
HM_error()	compute $\mathcal{H}$-matrix approximation error, $\|\mathbf{C} - \tilde{\mathbf{C}}\|$
HM_inverror()	compute $\mathcal{H}$-matrix approximation error, $\|\mathbf{I} - \tilde{\mathbf{C}}\tilde{\mathbf{C}}^{-1}\|$
HM_Cholerror()	compute $\mathcal{H}$-Cholesky error $\|\mathbf{I} - \tilde{\mathbf{C}}(\tilde{\mathbf{L}}\tilde{\mathbf{L}}^T)^{-1}\|$

joint multivariate Gaussian likelihood function and have found its maxima in the $\mathcal{H}$-matrix format. This maxima estimated unknown parameters (ℓ, ν, and σ^2) of the covariance model. The second novelty is that 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 a synthetic example, where we were able to identify the true parameters of the covariance model. With the suggested method we were able to compute the log-likelihood function for 2,000,000 locations just in a few minutes on a powerful desktop machine (Table 4). The $\mathcal{H}$-matrix technique allows us to increase the spatial resolution, to handle more measurements, consider larger regions and identify more parameters simultaneously. We also demonstrated how to speed up typical statistical routines (Table **??**) by using R-interface to HLIBpro.

Acknowledgments

The author would like to thank Ronald Kriemann from the Max Planck Institute for Mathematics in the Sciences in Leipzig for his assistance in the HLIBpro code. This work would be impossible without assistance from Marc Genton, Ying Sun and David Keyes. The research reported in this publication was supported by funding from the Extreme Computing Research Center (ECRC) at the 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. O'Neil. 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] J. Ballani and D. Kressner. Sparse inverse covariance estimation with hierarchical matrices. *http://sma.epfl.ch/∼anchpcommon/publications/quic_ballani_kressner_2014.pdf*, 2015.

[5] 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.

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

[7] 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.

[8] 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.

[9] 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.

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

[11] 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.

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

[13] 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.

[14] 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.

[15] 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.

[16] 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.

[17] 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.

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

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

[20] 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.

[21] 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.

[22] G. H. Golub and C. F. Van Loan. Matrix computations, 2012.

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

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

[25] 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.

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

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

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

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

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

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

[32] 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.

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

[34] 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.

[35] 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.

[36] 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.

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

[38] 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.

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

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

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

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

[43] F. Lindgren, H. Rue, and J. Lindström. An explicit link between Gaussian fields and Gaussian Markov random fields: the stochastic partial differential equation approach. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 73(4):423–498, 2011.

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

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

[46] 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.

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

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

[49] 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.

[50] 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.

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

[52] 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.

[53] 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.

[54] 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.

[55] 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.

[56] 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.

[57] 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.

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

[59] 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.

[60] 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.

[61] 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.

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

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

[64] 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.

[65] 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.

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

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

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

[69] 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.

[70] 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.

[71] 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.

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

A Maximum of the log-likelihood function

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

```
1  double call_compute_max_likelihood(TScalarVector Z, double nu, double covlength, double sigma2, TBlockClusterTree* bct,
       TClusterTree* ct, std::vector <double*> vertices, double output[3])
2  { gsl_function F;
3    int status; iter = 0, max_iter = 200; smy_f_params params ;
4    FILE* f1; double size;
5    const gsl_multimin_fminimizer_type *T = gsl_multimin_fminimizer_nmsimplex2;
6    gsl_multimin_fminimizer *s = NULL; gsl_vector *ss, *x;
7    gsl_multimin_function minex_func;
8    params.bct = bct; params.ct = ct; params.Z = Z; params.nu = nu;
9    params.covlength=covlength; params.sigma2=sigma2; params.vertices=vertices;
10   /* Starting point */
11   x = gsl_vector_alloc(3); gsl_vector_set (x, 0, nu);
12   gsl_vector_set (x, 1, covlength); gsl_vector_set (x, 2, sigma2);
13   /* 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;}
```

B (Block) Cluster trees and admissibility condition

Let I be an index set of all degrees of freedom. Denote for each index $i \in I$ corresponding to a basis function b_i the support $\mathcal{G}_i := \operatorname{supp} b_i \subset \mathbb{R}^d$. Now we define two trees which are necessary for the definition of hierarchical matrices. These trees are labeled trees where the label of a vertex t is denoted by $\hat{t}$.

Definition B.1. (Cluster Tree $T_{I\times I}$)
A finite tree T_I is a cluster tree over the index set I if the following conditions hold:

- I is the root of T_I and a subset $\hat{t} \subseteq I$ holds for all $t \in T_I$.
- If $t \in T_I$ is not a leaf, then the set of sons $\operatorname{sons}(t)$ contains disjoint subsets of I and the subset $\hat{t}$ is the disjoint union of its sons, $\hat{t} = \bigcup_{s \in \operatorname{sons}(t)} \hat{s}$.
- If $t \in T_I$ is a leaf, then $|\hat{t}| \leq n_{min}$ for a fixed number n_{min}.

Definition B.2. (Block Cluster Tree $T_{I\times I}$) Let T_I be a cluster tree over the index set I. A finite tree $T_{I\times I}$ is a block cluster tree based on T_I if the following conditions hold:

- $\operatorname{root}(T_{I\times I}) = I \times I$.
- Each vertex b of $T_{I\times I}$ has the form $b = (\tau,\sigma)$ with clusters $\tau,\sigma \in T_I$.
- For each vertex (τ,σ) with $\operatorname{sons}(\tau,\sigma) \neq \varnothing$, we have

$$\operatorname{sons}(\tau,\sigma) = \begin{cases} (\tau,\sigma') : \sigma' \in \operatorname{sons}(\sigma), & \text{if } \operatorname{sons}(\tau) = \varnothing \wedge \operatorname{sons}(\sigma) \neq \varnothing \\ (\tau',\sigma) : \tau' \in \operatorname{sons}(\tau), & \text{if } \operatorname{sons}(\tau) \neq \varnothing \wedge \operatorname{sons}(\sigma) = \varnothing \\ (\tau',\sigma') : \tau' \in \operatorname{sons}(\tau), \sigma' \in \operatorname{sons}(\sigma), & \text{otherwise} \end{cases}$$

- The label of a vertex (τ,σ) is given by $\widehat{(\tau,\sigma)} = \hat{\tau} \times \hat{\sigma} \subseteq I \times I$.

We can see that $\widehat{root(T_{I\times I})} = I \times I$. This implies that the set of leaves of $T_{I\times I}$ is a partition of $I \times I$.

C Rank-k Adaptive Cross Approximation (ACA)

An $\mathcal{H}$-matrix contains many sub-blocks, which can be well approximated by low-rank matrices. How to compute these low-rank approximations? The truncated singular value decomposition is accurate, but very slow. HLIBpro uses the Adaptive Cross Approximation method (ACA) [23] and its improved modifications such as ACA+ and HACA [6,10,12].

Definition C.1. Let $\boldsymbol{R} \in \mathbb{R}^{p\times q}$, then factorization

$$\boldsymbol{R} = \boldsymbol{A}\boldsymbol{B}^{\top}, \quad \text{where } \boldsymbol{A} \in \mathbb{R}^{p\times k}, \quad \boldsymbol{B} \in \mathbb{R}^{q\times k}, \quad k \ll \min\{p,q\} \in \mathbb{N}. \tag{C.1}$$

we will call a *low-rank representation*.

Note that any matrix of rank k can be represented in the form (C.1).
Suppose that b is a sub-block in the block-cluster tree, and $\boldsymbol{R} := \mathbf{C}|_b$. Suppose it is known that $\boldsymbol{R}$ could be approximated by a rank-k matrix. We explain below how to compute $\boldsymbol{R}$ in the form (C.1). ACA is especially effective for assembling low-rank matrices. It requires to compute only k columns and k rows of the matrix under consideration and, thus, has the computational cost $k(p+q)$. In [23] it is proved that if there exists a sufficiently good low-rank approximation, then there also exists a cross approximation with almost the same accuracy in the sense of the spectral norm. The ACA algorithm computes vectors a_ℓ and b_ℓ which form $\tilde{\boldsymbol{R}} = \sum_{\ell=1}^{k} a_\ell b_\ell^{\top}$ such that $\|\boldsymbol{R} - \tilde{\boldsymbol{R}}\| \le \varepsilon$, where ε is the desired accuracy [10,13]. In [12] the reader can also find different counterexamples when the standard ACA algorithm does not work. Here we present the standard version of the ACA algorithm.

Algorithm C.1. ACA algorithm

begin
  /* input is a required accuracy ε and a function to compute $\boldsymbol{R}_{ij}$ */;
  /* output is matrix $\tilde{\boldsymbol{R}}$ */;
  $k = 0$; $\tilde{\boldsymbol{R}} = 0$;
  $S = \emptyset$; $T = \emptyset$; /* sets of row and column indices */
  do
    Take a row $i^* \notin S$;
    Subtract $\boldsymbol{R}_{i^*j} := \boldsymbol{R}_{i^*j} - \tilde{\boldsymbol{R}}_{i^*j}$, $j = 1..q$;
    Find $\max_j |a_{i^*j}| \neq 0$, $j < q$. Suppose it lies in column j^*;
    Compute all elements b_{ij^*} in column j^*, $i < p$;

Subtract $\boldsymbol{R}_{ij^*} := \boldsymbol{R}_{ij^*} - \tilde{\boldsymbol{R}}_{ij^*}$, $i = 1..p$;
$k := k+1$; $S := S \cup \{i^*\}$; $T := T \cup \{j^*\}$;
Compute $\tilde{\boldsymbol{R}} = \tilde{\boldsymbol{R}} + a_{i^*} \cdot b_{j^*}^\top$; /* it is rank k approximation*/
if($\|a_{i^*} \cdot b_{j^*}^\top\|_2 \leq \varepsilon \cdot \|a_1 \cdot b_1^\top\|_2$) return $\tilde{\boldsymbol{R}}$;
Find $\max_i |b_{ij^*}|$, $i < p$, $i \neq i^*$. The row where it lies is a new row i^*;
until($k < k_{\max}$)
return $\tilde{\boldsymbol{R}}$;
end;

Note that the algorithm does not compute the whole matrix $\boldsymbol{R}$. The subtraction is done only from the elements under consideration, i.e. row a_ℓ and column b_ℓ, $\ell = 1,...,k$.

Remark C.1. Further optimisation of the ACA algorithm can be done by the truncated SVD. Suppose that a factorisation of 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. Suppose also that the rank of $\boldsymbol{R}$ is k, $k < K$. Then one can apply the truncated SVD algorithm to compute $\boldsymbol{R} = \boldsymbol{U}\Sigma\boldsymbol{V}^\top$ requiring $\mathcal{O}((p+q)K^2 + K^3)$ FLOPs.

Definition C.2. The Kullback-Leibler divergence (KLD) $D_{KL}(P\|Q)$ is a measure of the information lost when distribution Q is used to approximate P:

$$D_{KL}(P\|Q) = \sum_i P(i) \ln \frac{P(i)}{Q(i)}, D_{KL}(P\|Q) = \int_{-\infty}^{\infty} p(x) \ln \frac{p(x)}{q(x)} dx,$$

where p, q densities of P and Q. For multivariate normal distributions $(\boldsymbol{\mu}_0, \Sigma_0)$ and $(\boldsymbol{\mu}_1, \Sigma_1)$

$$D_{KL}(\mathcal{N}_0\|\mathcal{N}_1) = 0.5 \left(tr(\Sigma_1^{-1}\Sigma_0) + (\boldsymbol{\mu}_1 - \boldsymbol{\mu}_0)^\top \Sigma_1^{-1} (\boldsymbol{\mu}_1 - \boldsymbol{\mu}_0) - k - \ln \left(\frac{\det \Sigma_0}{\det \Sigma_1} \right) \right)$$