MethodsX 7 (2020) 100600

Contents lists available at ScienceDirect

MethodsX

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

Method Article

HLIBCov: Parallel hierarchical matrix approximation of large covariance matrices and likelihoods with applications in parameter identification

Alexander Litvinenko[a,*], Ronald Kriemann[b], Marc G. Genton[c], Ying Sun[c], David E. Keyes[c]

[a] *RWTH Aachen, Kackertstr. 9C, 52072 Aachen, Germany*
[b] *Max-Planck-Institut für Mathematik in den Naturwissenschaften, Inselstr 22, 04103 Leipzig, Germany*
[c] *King Abdullah University of Science and Technology, 23955-6900, Thuwal, Saudi Arabia*

A B S T R A C T

We provide more technical details about the HLIBCov package, which is using parallel hierarchical (H-) matrices to:

- Approximate large dense inhomogeneous covariance matrices with a log-linear computational cost and storage requirement.
- Compute matrix-vector product, Cholesky factorization and inverse with a log-linear complexity.
- Identify unknown parameters of the covariance function (variance, smoothness, and covariance length).

These unknown parameters are estimated by maximizing the joint Gaussian log-likelihood function. To demonstrate the numerical performance, we identify three unknown parameters in an example with 2,000,000 locations on a PC-desktop.

A R T I C L E I N F O

Keywords: Parallel, Hierarchical matrices, Large datasets, Matérn covariance, Random fields, HLIBCov, HLIBpro, Cholesky, Matrix determinant, Parameter identification
Article history: Received 30 January 2019; Accepted 1 July 2019; Available online 11 July 2019

* Corresponding author at: RWTH Aachen, Kackertstr. 9C, 52072 Aachen, Germany.
E-mail address: litvinenko@uq.rwth-aachen.de (A. Litvinenko).

https://doi.org/10.1016/j.mex.2019.07.001

Subject Area	Mathematics
More specific subject area	Applied mathematics, computational statistics, data analysis
Method name	HLIBCov (parallel hierarchical matrices for approximating large covariance matrices, likelihood functions and MLE estimations)
Name and reference of original method	A. Litvinenko, Y. Sun, M. G. Genton, and D. E. Keyes. Likelihood approximation with hierarchical matrices for large spatial datasets. Computational Statistics & Data Analysis, 137:115–132, (2019). W. Hackbusch. A sparse matrix arithmetic based on H-matrices. Introduction to H-matrices. Computing, 62(2):89-108, 1999. ISSN0010-485X. R. Kriemann. Parallel H-matrix arithmetics on shared memory systems. Computing, 74 (3):273-297, 2005. ISSN 0010-485x. doi: 10.1007/s00607-004-0102-2. URL: http://www.mis.mpg.de/de/publications/preprints/2004/prepr2004-29.html
Resource availability	A. Litvinenko. HLIBCov: Log-likelihood approximation with hierarchical matrices, 2017. URL https://github.com/litvinen/HLIBCov.git

1 Technical details

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 (GPL 2.0), HLIBpro (proprietary)
CiCP scientific software URL:	Distribution format:
Distribution format:	*.cc files via github
Programming language(s):	C++
Computer platform:	any
Operating system:	Linux, MacOSX and MS Windows
Compilers:	standard C++ compilers
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 CPU cores
Restrictions:	None (similar limitations as HLIBpro)
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.

2 Introduction

HLIBpro is a very fast and efficient parallel $\mathcal{H}$-matrices library. This is an auxiliary technical paper, which contains technical details to our previous paper [31]. In [31] we used the gradient-free optimization method to estimate the unknown parameters of a covariance function using HLIB and HLIBpro.

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 $\mathbf{s} := \{\mathbf{s}_1, \dots, \mathbf{s}_n\} \in \mathbb{R}^d$, $d \geq 1$. We also let $\boldsymbol{Z} = \{Z(\mathbf{s}_1), \dots, Z(\mathbf{s}_n)\}^\top$, where $Z(\mathbf{s})$ is a stationary 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 distance and vector $\boldsymbol{\theta} \in \mathbb{R}^q$ denotes q unknown parameters. To infer $\boldsymbol{\theta}$, we maximize the joint Gaussian log-likelihood function,

$$\mathcal{L}(\boldsymbol{\theta}) = -\frac{n}{2}\log(2\pi) - \frac{1}{2}\log|\mathbf{C}(\boldsymbol{\theta})| - \frac{1}{2}\mathbf{Z}^\top \mathbf{C}(\boldsymbol{\theta})^{-1}\mathbf{Z}, \tag{2.1}$$

where $\mathbf{C}(\boldsymbol{\theta})_{ij} = C(\mathbf{s}_i - \mathbf{s}_j;\boldsymbol{\theta})$, $i,j = 1, \ldots, n$. Let us assume that $\hat{\boldsymbol{\theta}}$ maximizes (2.1). When the sample size n is large, the evaluation of (2.1) becomes challenging, due to $O(n^3)$ computational cost of the Cholesky factorization. Hence, scalable and efficient methods that can process larger n are needed.

For this, the hierarchical matrices ($\mathcal{H}$-matrix) technique is used, which approximates sub-blocks of the dense matrix by a low-rank representation of either a given rank k or a given accuracy $\varepsilon > 0$ (see Section 3.2).

Definition 2.1. An $\mathcal{H}$-matrix approximation with maximal rank k of the exact log-likelihood $\mathcal{L}(\boldsymbol{\theta})$ is defined by $\tilde{\mathcal{L}}(\boldsymbol{\theta};k)$:

$$\tilde{\mathcal{L}}(\boldsymbol{\theta};k) = -\frac{n}{2}\log(2\pi) - \sum_{i=1}^{n}\log\{\tilde{\mathbf{L}}_{ii}(\boldsymbol{\theta};k)\} - \frac{1}{2}\mathbf{v}(\boldsymbol{\theta})^\top\mathbf{v}(\boldsymbol{\theta}), \tag{2.2}$$

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

To maximize $\tilde{\mathcal{L}}(\boldsymbol{\theta};k)$ in (2.2), we use the Brent-Dekker method [9,33]. It could be used with or without derivatives.

An additional difficulty is the ill-posedness of the optimization problem. Even a small perturbation in the covariance matrix $\mathbf{C}(\boldsymbol{\theta})$ 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. $\mathbf{C}(\boldsymbol{\theta})^{-1}$, $\mathbf{C}(\boldsymbol{\theta})^{1/2}$, $|\mathbf{C}(\boldsymbol{\theta})|$, Cholesky decomposition, the Schur complement, and many others can be computed in the $\mathcal{H}$-matrix format [16];
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 cost and accuracy is controlled by k;
5. The $\mathcal{H}$-Cholesky factor and the $\mathcal{H}$-inverse have often moderate ranks.

Structure of the paper. After introduction and problem setting in Section 2, we explain the $\mathcal{H}$-matrix approximation of Matérn covariance matrices in Section 3. In Section 4, we describe the software installation details, the input, and output of the HLIBCov code. In Section 5, we provide dependence of the storage and computing costs vs. the $\mathcal{H}$-matrix rank k. We visualize the shapes of the log-likelihood functions for different n. Finally, we demonstrate how to estimate three unknown parameters $\boldsymbol{\theta} = (\ell, \nu, \sigma^2)^\top$ of $\mathbf{C}(\boldsymbol{\theta})$. Best practices are listed in Section 6. We end the paper with a conclusion in Section 7. The auxiliary $\mathcal{H}$-matrix and log-likelihood details are provided in the Appendices A and B.

3 Methodology and algorithms

3.1 Matérn covariance functions

Matérn covariance functions [32] are a very widely used class of functions [14,20].

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

$$C(\mathbf{h};\boldsymbol{\theta}) = \frac{\sigma^2}{2^{\nu-1}\Gamma(\nu)}\left(\frac{\mathbf{h}}{\ell}\right)^{\nu} K_\nu\left(\frac{\mathbf{h}}{\ell}\right), \tag{3.1}$$

where $\boldsymbol{\theta} = (\ell, \nu, \sigma^2)^\top$; $\ell > 0$ is a spatial range parameter; $\nu > 0$ is the smoothness, with larger values of ν corresponding to smoother random fields; and σ^2 is the variance. Here, K_ν denotes a modified Bessel function of the second kind of order ν, and $\Gamma(\cdot)$ denotes the Gamma function. The values $\nu = 1/2$ and $\nu = \infty$ correspond to the exponential and Gaussian covariance functions respectively.

3.2 Introduction to hierarchical matrices

Detailed descriptions of hierarchical matrices [12,15–18,29] and their applications can be found elsewhere [1,2,6,21,23,24,30].

The $\mathcal{H}$-matrix technique was originally introduced by Hackbusch (1999) for the approximation of stiffness matrices and their inverses coming from partial differential and integral equations [8,12,15]. Briefly, the key idea of the $\mathcal{H}$-matrix technique is to divide the initial matrix into sub-blocks in a specific way, identify those sub-blocks which can be approximated by low-rank matrices and compute the corresponding low-rank approximations.

The partitioning of the matrix into sub-blocks starts by recursively dividing the rows and columns into disjoint sub-sets, e.g., splitting the set of all rows into two (equal sized) sub-sets, which are again divided. This yields a *cluster tree* where each sub-set of rows/columns is called a *cluster*. By multiplying the cluster trees for the rows and the columns, a hierarchical partitioning of the matrix index set is obtained, the so called *block cluster tree* or $\mathcal{H}$. Within this block cluster tree, low-rank approximable blocks are identified using an *admissibility condition*. Such *admissible* blocks are not further refined into sub-blocks, i.e., the corresponding sub-tree is not computed or stored. For all admissible blocks a low-rank approximation of the initial matrix is computed, either with a given rank k (fixed-rank strategy) or an accuracy $\varepsilon > 0$ (fixed-accuracy strategy). The result of this computation is called an $\mathcal{H}$-matrix. This process is also shown in Fig. 1.

Definition 3.1. Let I be an index set (representing the rows/columns) and T_I be a cluster tree based on I. Furthermore, let $T_{I\times I}$ be a block cluster tree based on T_I and an admissibility condition adm : $T_{I\times I} \to \{true, false\}$. Then the set of $\mathcal{H}$-matrices with maximal rank k is defined as

$$\mathcal{H}(T_{I\times I}, k) := \{\mathbf{C} \in \mathbb{R}^{I\times I} |\, \mathrm{rank}(\mathbf{C}|_{t\times s}) \le k \text{ for all } (t,s) \text{ of } T_{I\times I} \text{ with } \mathrm{adm}(t,s) = \mathrm{true}\}.$$

Here the block cluster (t, s) is an element (a vertex) of the block cluster tree $T_{I\times I}$.

Various partitioning strategies for the rows and columns of the matrix and admissibility conditions have been developed to approximate different types of matrices. Typical admissibility conditions are *strong* (also called *standard*), *weak* and based on domain decomposition [16], for which examples are shown in Fig. 2. The red blocks indicate dense or in-admissible blocks whereas green blocks are identified as admissible. The maximal size of the dense blocks (i.e., how deep the hierarchical subdivision into sub-blocks is) is regulated by the parameter "$n_{\min}$", whose value affects the storage

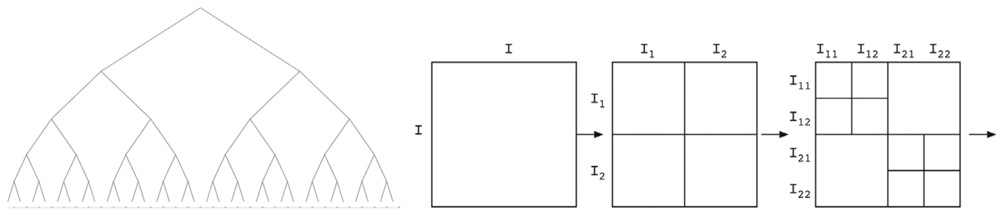

Fig. 1. Examples of a cluster tree T_I (left) and a block cluster tree $T_{I\times I}$ (right). The decomposition of the matrix into sub-blocks is defined by $T_{I\times I}$ and the admissibility condition.

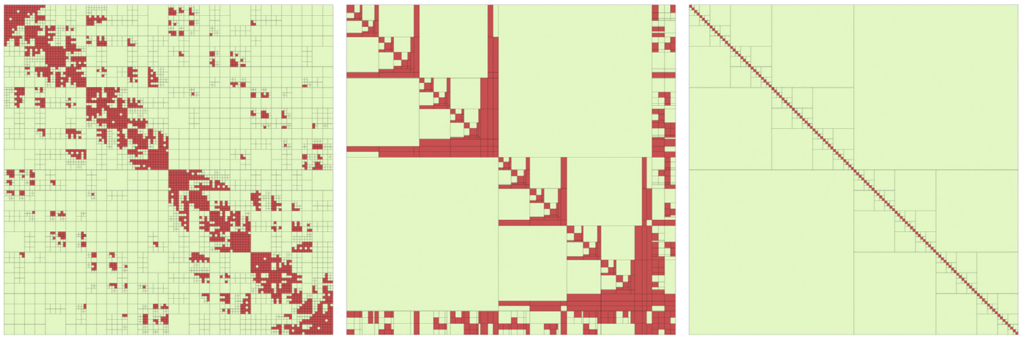

Fig. 2. Examples of three different block partitioning, generated with three different admissibility criteria: (left) strong, (middle) domain-decomposition-based, and (right) weak.

size and the runtime of the $\mathcal{H}$-matrix arithmetic, e.g., a smaller value leads to less storage but is often in-efficient with respect to CPU performance. Typically values of $n_{\min}$ are in the range 20 to 150.

Remark 1. With a more appropriate choice of the admissibility condition (criteria), one can influence the depth of the hierarchy, the size of the sub-blocks, their number, the number of empty sub-blocks. The choice of the admissibility criteria is not crucial here; the $\mathcal{H}$-matrix rank (or accuracy) in each sub-block is much more crucial. If the $\mathcal{H}$-matrix rank in each block is sufficiently large, then any admissibility criteria will work. In our numerical tests, we used the standard admissibility criteria.

For the computation of the low-rank approximation for admissible sub-blocks many different methods are available, e.g., adaptive cross approximation (ACA), hybrid cross approximation (HCA), rank-revealing QR, randomized SVD [3,5,7,8,11,19,22]. For the fixed-rank strategy, the resulting low-rank matrix is of rank at most k. In case of the fixed-accuracy strategy with a given $\varepsilon > 0$, the low-rank approximation $\tilde{\boldsymbol{M}}$ of the sub-block $\boldsymbol{M}$ is computed such that $\|\boldsymbol{M} - \tilde{\boldsymbol{M}}\| \leq \varepsilon\|\boldsymbol{M}\|$. The storage size of the resulting $\mathcal{H}$-matrix is of order $\mathcal{O}(kn\log n)$ [12].

In Fig. 3 (left), an example of an $\mathcal{H}$-matrix approximation to $\mathbf{C}(\boldsymbol{\theta})$ can be found. There, the local ranks and the decay of singular values in the admissible blocks (green) in logarithmic scale are shown.

In addition to efficient matrix approximation, $\mathcal{H}$-matrices also permit full matrix arithmetic, e.g., matrix addition, matrix multiplication, inversion or factorization. However, similar to matrix compression, $\mathcal{H}$-matrix arithmetic is approximate to maintain log-linear complexity. The approximation during arithmetic is again either of a fixed-rank or a fixed-accuracy [12]. In this work, we make use of the $\mathcal{H}$-Cholesky factorization of $\mathbf{C}(\boldsymbol{\theta})$ (see Fig. 3).

For $\mathbf{C}(\boldsymbol{\theta})$, the predefined rank (or accuracy ε) defines the accuracy of the $\mathcal{H}$-matrix approximation, for the initial approximation of $\mathbf{C}(\boldsymbol{\theta})$ as well as for the Cholesky factorization for $\mathbf{C}(\boldsymbol{\theta})^{-1}$.

3.3 Parallel hierarchical-matrix technique

We used the parallel $\mathcal{H}$-matrix library HLIBpro [13,25,27,28], which implements $\mathcal{H}$-matrix approximation and arithmetic functions using a task-based approach to make use of todays many-core architectures. For this, the mathematical operation is decomposed into small atomic *tasks* with corresponding incoming and outgoing data dependencies. This set of tasks and dependencies forms a directed acyclic graph (DAG), which is used for scheduling the tasks to the CPU cores, e.g., if all incoming data dependencies are met, the corresponding task is executed on the next free CPU core available.

The computational complexity of the different $\mathcal{H}$-matrix operations is shown in Table 1. Here, $|V(T)|$ denotes the number of vertices, $|L(T)|$ is the number of leaves in the block-cluster tree $T = T_{I\times I}$. The sequential terms in those estimates are typically due to the sequential behaviour of the corresponding

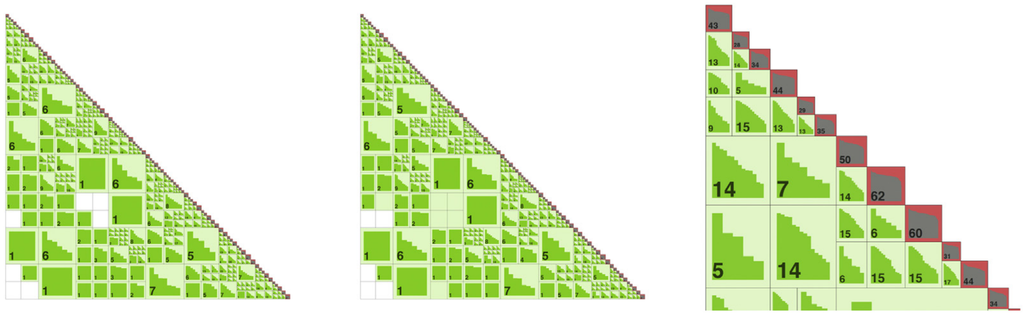

Fig. 3. Examples of $\mathcal{H}$-matrix approximations of the exponential covariance matrix (left), its hierarchical Cholesky factor $\tilde{\boldsymbol{L}}$ (middle), and the zoomed upper-left corner of the matrix (right), $n = 4000$, $\ell = 0.09$, $\nu = 0.5$, $\sigma^2 = 1$. Approximation and arithmetic performed with a fixed-accuracy of 10^{-5}. The number inside a sub-block indicates the maximal rank, while the "stairs" represent its singular values in logarithmic scaling.

Table 1
Parallel complexity of the main linear operations in HLIBpro on p cores. Truncated multiplication and addition are denoted by $\odot$ and $\oplus$.

Operation	Parallel Complexity [26] (Shared Memory)
build $\tilde{\boldsymbol{C}}$	$\frac{\mathcal{O}(n\log n)}{p} + \mathcal{O}(\lvert V(T)\backslash L(T)\rvert)$
store $\tilde{\boldsymbol{C}}$	$\mathcal{O}(kn\log n)$
$\tilde{\boldsymbol{C}}\cdot\boldsymbol{z}$	$\frac{\mathcal{O}(kn\log n)}{p}$
$\alpha\tilde{\boldsymbol{A}} \oplus \beta\tilde{\boldsymbol{B}}$	$\frac{\mathcal{O}(n\log n)}{p}$
$\alpha\tilde{\boldsymbol{A}} \odot \tilde{\boldsymbol{B}} \oplus \beta\tilde{\boldsymbol{C}}$	$\frac{\mathcal{O}(n\log n)}{p} + \mathcal{O}(\lvert V(T)\rvert)$
$\tilde{\boldsymbol{C}}^{-1}$	$\frac{\mathcal{O}(n\log n)}{p} + \mathcal{O}(nn_{\min}^2)$
$\mathcal{H}$-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 $\lvert\tilde{\boldsymbol{C}}\rvert$	$\frac{\mathcal{O}(n\log n)}{p} + \mathcal{O}(\frac{k^2 n\log^2 n}{n^{1/d}}), d = 1,2,3$

algorithm, e.g., strictly following the diagonal during Cholesky factorization, but usually do not show in practical applications since the majority of the computation work is parallelized.

4 HLIBCov and HLIBpro installation

This section contains a summary of the information provided at https://www.hlibpro.com and https://github.com/litvinen/HLIBCov.git. HLIBpro supports both shared and distributed memory architectures, though in this work we only use the shared memory version. For the implementation of the task-parallel approach, Intel's Threading Building Blocks (TBB) is used, Table 2. HLIBpro is free for academic purposes, and is distributed in a pre-compiled form (no source code available). Originally, HLIBpro was developed for solving FEM and BEM problems [13,28]. 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. All functionality implemented by HLIBCov is based on HLIBpro, i.e., no extra software is needed in addition to the libraries needed by HLIBpro. This also holds for the Matérn kernel, which uses Bessel functions and maximization algorithms, both being provided by the GNU Scientific Library (GSL) and also used by HLIBpro. The reader can easily replace GSL with his own optimization library. The Bessel functions are also available in other packages.

To install HLIBpro on MacOS and Windows, we refer the reader to www.HLIBpro.com for further details.

Table 2
Version of Software used for Experiments.

Software	Version
HLIBCov	1.0
HLIBpro	2.6
GSL	1.16
TBB	4.3

Hardware. All of the numerical experiments herein are performed on a Dell workstation with two Intel(R) Xeon(R) E5-2680 v2 CPUs (2.80GHz, 10 cores/20 threads) and 128 GB main memory.

Adding HLIBCov to HLIBpro. The easiest form of compiling HLIBCov is by using the compilation system of HLIBpro. For this, the source code file of HLIBCov is placed in the *examples* directory of HLIBpro and an entry is added to the file *examples/SConscript*:

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

Afterwards, the make process of HLIBpro is run to compile also HLIBCov (see HLIBpro installation instructions at www.hlibpro.com).

Input of HLIBCov. The input contained in the first line is the total number of locations n. Lines $2, \ldots, n+1$ 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 open-access file hosting service GitHub (https://github.com/litvinen/HLIBCov.git). We added two data sets to GitHub: `data.tar.gz` and `moisture_data.zip`. Both examples contain multiple data sets of different sizes.

Output of HLIBCov. The main output is the three identified parameter values $\boldsymbol{\theta} = (\ell, \nu, \sigma^2)^\top$. The auxiliary output may include $\mathcal{H}$-matrix details: the maximal rank k, the maximal accuracy in each sub-block, and the Frobenius and spectral norms of $\tilde{\mathbf{C}}$, $\tilde{\mathbf{L}}$, $\tilde{\mathbf{L}}^{-1}$, $\left\| I - \tilde{\mathbf{L}}\tilde{\mathbf{L}}^{\top -1} \right\|$. 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 $\boldsymbol{\theta}^* = (\ell^*, \nu^*, \sigma^{*2})^\top$. When computing error boxes, the output file will contain M solutions (n, ℓ^*, ν^*, σ^{*2}), where M is the number of replicates:

```
4000  0.54  0.082  1.01
4000  0.53  0.083  1.02
4000  0.55  0.081  1.02
```

The name of the output file can be found in the main() procedure in loglikelihood.cc.

5 Numerical experiments

We perform several numerical tests. First, we investigate how the approximation errors and the Kullback-Leibler divergence depend on the maximal rank k. Second, we show how the memory requirement for the matrix $\tilde{\mathbf{C}}$ depends on ℓ and ν. Third, we compute the variances of ℓ and ν vs. k and n. Fourth, we estimate the unknown parameters.

5.1 Convergence errors and memory requirement

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

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

In Tables 3 and 4, we show the dependence of KLD and two matrix errors on the $\mathcal{H}$-matrix rank k for the Matérn covariance function with parameters $\ell = \{0.25,\ 0.75\}$, $\nu = \{0.5,\ 1.5\}$, and $\sigma^2 = \{1.0,\ 1.0\}$, computed on the domain $\mathcal{G} = [0,1]^2$. All errors are under control, except for the last column. The ranks $k = \{10, 12\}$ or $k = \{10, 20\}$ are too small to approximate the inverse, and, therefore, the resulting error $\|\mathbf{C}(\tilde{\mathbf{C}})^{-1} - \mathbf{I}\|_2$ is large. 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 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 linearly dependent. To avoid such cases, the available data should be preprocessed to remove duplicate locations. Very often, the nugget $\tau^2\mathbf{I}$ is added to the main diagonal to stabilize numerical calculations (see more in Section 5.5), i.e., $\tilde{\mathbf{C}} := \tilde{\mathbf{C}} + \tau^2\mathbf{I}$. In Tables 3 and 4, the nugget is equal to zero.

Fig. 4 shows that the $\mathcal{H}$-matrix storage cost remains almost the same for the different parameters $\ell = \{0.15,\ \ldots, 2.2\}$ (left) and $\nu = \{0.3,\ \ldots, 1.3\}$ (right). The Matérn covariance function is discretized in the domain $[32.4,\ 43.4] \times [-84.8,\ -72.9]$ with $n = 2,000$ mesh points.

Table 3
KLD and $\mathcal{H}$-matrix approximation errors vs. the $\mathcal{H}$-matrix rank k for Matérn covariance function, $\ell = \{0.25,\ 0.75\}$, $\nu = 0.5$, $\sigma^2 = 1$, domain $\mathcal{G} = [0,1]^2$, and $\|\mathbf{C}_{(\ell=0.25,0.75)}\|_2 = \{212,\ 568\}$.

k	KLD		$\|\mathbf{C} - \tilde{\mathbf{C}}\|_2$		$\|\mathbf{C}(\tilde{\mathbf{C}})^{-1} - \mathbf{I}\|_2$	
	$\ell = 0.25$	$\ell = 0.75$	$\ell = 0.25$	$\ell = 0.75$	$\ell = 0.25$	$\ell = 0.75$
10	2.6×10^{-3}	2.0×10^{-1}	7.7×10^{-4}	7.0×10^{-4}	6.0×10^{-2}	3.1×10^{0}
12	5.0×10^{-4}	2.2×10^{-2}	9.7×10^{-5}	5.6×10^{-5}	1.6×10^{-2}	5.0×10^{-1}
15	1.0×10^{-5}	9.0×10^{-4}	2.0×10^{-5}	1.1×10^{-5}	8.0×10^{-4}	2.0×10^{-2}
20	4.5×10^{-7}	4.8×10^{-5}	6.5×10^{-7}	2.8×10^{-7}	2.1×10^{-5}	1.2×10^{-3}
50	3.4×10^{-13}	5.0×10^{-12}	2.0×10^{-13}	2.4×10^{-13}	4.0×10^{-11}	2.7×10^{-9}

Table 4
KLD and $\mathcal{H}$-matrix approximation error vs. the $\mathcal{H}$-matrix rank k for Matérn covariance function, $\ell = \{0.25,\ 0.75\}$, $\nu = 1.5$, $\sigma^2 = 1$, domain $\mathcal{G} = [0,1]^2$, and $\|\mathbf{C}_{(\ell=0.25,0.75)}\|_2 = \{720,\ 1068\}$.

k	KLD		$\|\mathbf{C} - \tilde{\mathbf{C}}\|_2$		$\|\mathbf{C}(\tilde{\mathbf{C}})^{-1} - \mathbf{I}\|_2$	
	$\ell = 0.25$	$\ell = 0.75$	$\ell = 0.25$	$\ell = 0.75$	$\ell = 0.25$	$\ell = 0.75$
20	1.2×10^{-1}	2.7×10^{0}	5.3×10^{-7}	2.3×10^{-7}	4.5×10^{0}	7.2×10^{1}
30	3.2×10^{-5}	4.0×10^{-1}	1.3×10^{-9}	5.0×10^{-10}	4.8×10^{-3}	2.0×10^{1}
40	6.5×10^{-8}	1.0×10^{-2}	1.5×10^{-11}	8.0×10^{-12}	7.4×10^{-6}	5.0×10^{-1}
50	8.3×10^{-10}	3.0×10^{-3}	2.0×10^{-13}	1.5×10^{-13}	1.5×10^{-7}	1.0×10^{-1}

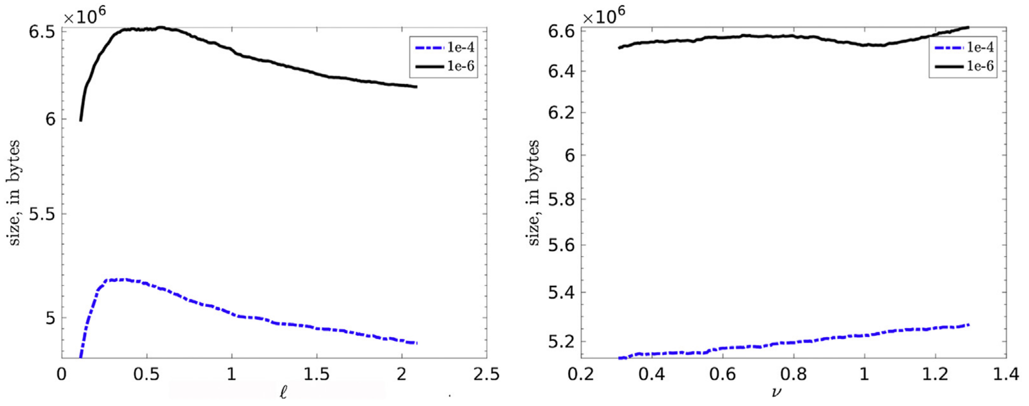

Fig. 4. Dependence of the matrix size on (left) the covariance length ℓ (other two parameters are fixed $\nu = 0.325$, $\sigma^2 = 0.98$), and (right) the smoothness ν (other two parameters are fixed $\ell = 0.58$, $\sigma^2 = 0.98$) for two different accuracies in the $\mathcal{H}$-matrix sub-blocks $\varepsilon = \{10^{-4}, 10^{-6}\}$, for $n = 2, 000$ locations in the domain $[32.4, 43.4] \times [-84.8, -72.9]$.

5.2 Uncertainty in parameters vs. k *and* n

In Fig. 5 (left), the results for computing ℓ with a different rank in the $\mathcal{H}$-matrix approximation for 100 replicates are shown. On each box, the central red line indicates the median. The lower edge of the box indicates the 25% percentile, and the top edge indicates the 75% percentile. The outliers are marked by the red symbol '+'. The bold long red line denotes the true value of the parameter $\ell = 0.0334$. With a larger rank and hence, with a better approximation, the variance of ℓ decreases.

The dependence of ν on the problem size, e.g., the number of measurements is also tested with the results shown in Fig. 5 (right). As the results demonstrate, with a larger number n the estimation of the parameter ν is getting better. This experiment is also showing that the estimations, obtained on a smaller data set, say with $n = 2000$ measurement, could be used as a starting value in the next experiment with a larger data set, say $n = 4000$.

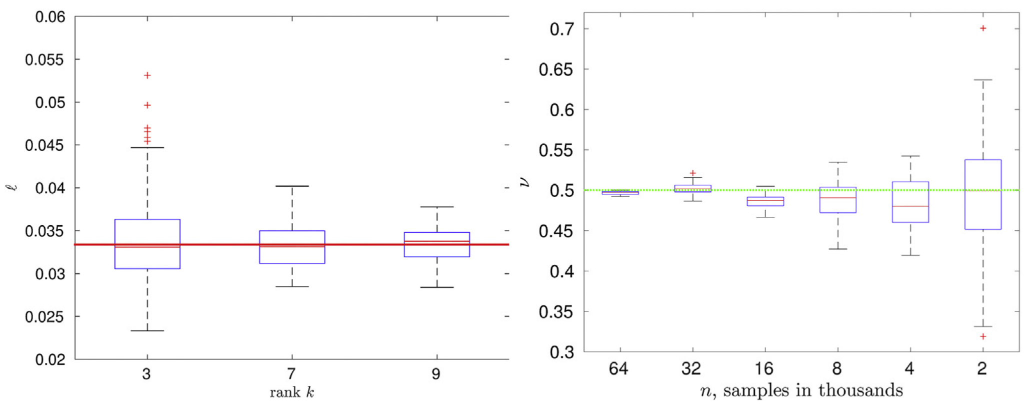

Fig. 5. (left) Dependence of the boxplots for ℓ on the $\mathcal{H}$-matrix rank k, when $n = 16, 000$; (right) Convergence of the boxplots for ν with increasing n (read this plot from right to the left); 100 replicates.

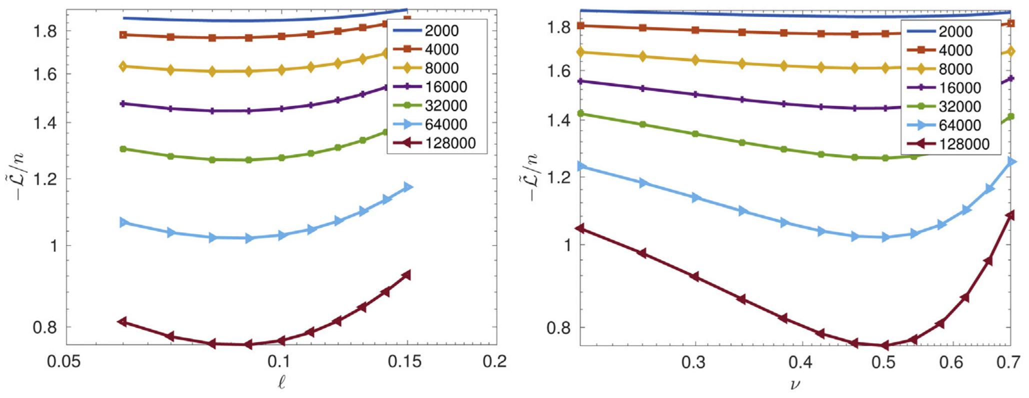

Fig. 6. (left) Shape of the scaled log-likelihood function, $-\tilde{\mathcal{L}}/n$, vs. ℓ for different sample sizes n. (right) Shape of the scaled log-likelihood function, $-\tilde{\mathcal{L}}/n$, vs. ν for different sample sizes n.

5.3 Log-likelihood vs. ℓ and ν for different n

In Fig. 6, we illustrate the dependence of $-\tilde{\mathcal{L}}/n$ on the parameters ℓ (left, with $\nu = 0.5$, $\sigma^2 = 1$), and ν (right, with $\ell = 0.0864$ and $\sigma^2 = 1$). Both figures demonstrate the smooth dependence of $\tilde{\mathcal{L}}/n$ on ℓ and ν. It also illustrates the locations of the minima for different numbers $n = \{2000, 4000, \ldots, 128000\}$.

5.4 Identification of unknown parameters

We generate a "synthetic" data set with parameters $(\ell^*, \nu^*, \sigma^{*2}) = (0.0864, 0.5, 1.0)$ and then try to infer these parameters. To build M various datasets (M replicates) with $n \in \{64, \ldots, 4, 2\} \times 1000$ locations, we generate a large vector $\mathbf{Z}_0$ with $n_0 = 2 \times 10^6$ 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 $\mathbf{Z}_0 \in \mathbb{R}^{n_0}$, we compute the $\mathcal{H}$-Cholesky factorization of $\tilde{\mathbf{C}}(0.086, 0.5, 1.0) = \tilde{\mathbf{L}}\tilde{\mathbf{L}}^\top$. 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. Next, we run our optimization algorithm and try to

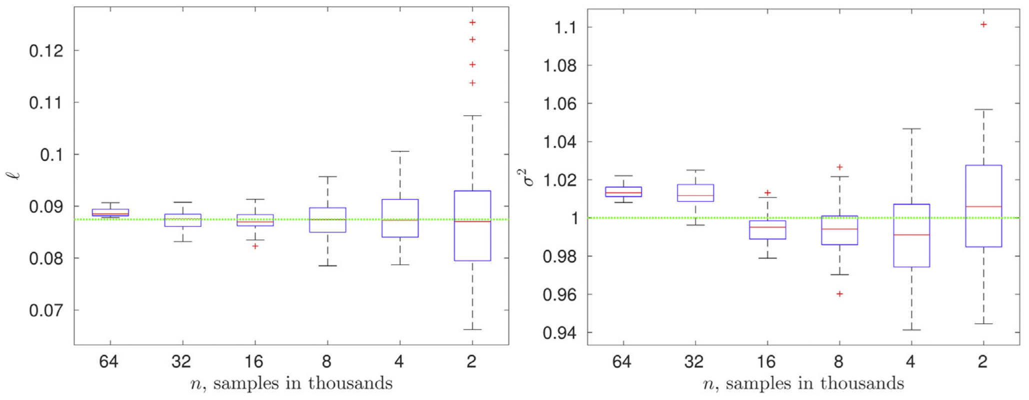

Fig. 7. Synthetic data with known parameters $(\ell^*, \nu^*, \sigma^{*2}) = (0.0864, 0.5, 1.0)$. Boxplots for ℓ and σ^2 for $n = 1,000 \times \{64, 32, \ldots, 4, 2\}$; 100 replicates.

Table 5
Computing time and storage vs n. The number of parallel computing cores is 40, $\hat{\nu} = 0.33, \hat{} = 0.65$, $\hat{\sigma}^2 = 1.0$. $\mathcal{H}$-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	size	kB/dof	comp. time	size	$\lVert \boldsymbol{I} - (\tilde{\boldsymbol{L}}\tilde{\boldsymbol{L}}^\top)^{-1}\tilde{\boldsymbol{C}} \rVert_2$
	sec.	MB		sec.	MB	
32,000	3.3	162	5.1	2.4	172.7	2.4×10^{-3}
128,000	13.3	776	6.1	13.9	881.2	1.1×10^{-2}
512,000	52.8	3420	6.7	77.6	4150	3.5×10^{-2}
2,000,000	229	14790	7.4	473	18970	1.4×10^{-1}

identify (recover) the "unknown" parameters $(\ell, \nu, \sigma^2)^\top$. The resulting boxplots for ℓ and σ^2 over $M = 100$ replicates are illustrated in Fig. 7. We see that the variance (or uncertainty) decreases with increasing n. The green line indicates the true values.

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 $n_0 = 2 \times 10^6$.

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

The shape of the negative log-likelihood function and its components are illustrated in Fig. 8. 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 it may be necessary to perform many iterations in order to find the minimum.

To research how close to each other the log-likelihood functions are, computed with different $\mathcal{H}$-matrix accuracy, we demonstrate Table 6. It contains the values of three log-likelihood functions computed with three different $\mathcal{H}$-matrix accuracies $\{10^{-7}, 10^{-9}, 10^{-11}\}$. The covariance function is exponential (i.e., $\nu = 0.5$) and is discretized in the domain $[32.4, 43.4] \times [-84.8, -72.9]$ with $n = 32,000$ mesh points (locations). The columns correspond to different covariance lengths {0.001, . . . ,0.1}.

5.5 Adding nugget τ^2

When diagonal values of $\tilde{\boldsymbol{C}}$ are very close to zero, the $\mathcal{H}$-Cholesky procedure becomes unstable. It produces negative entries on the diagonal during computation. By adding a diagonal matrix with small positive numbers, all the singular values become larger and move away from zero. However, by adding

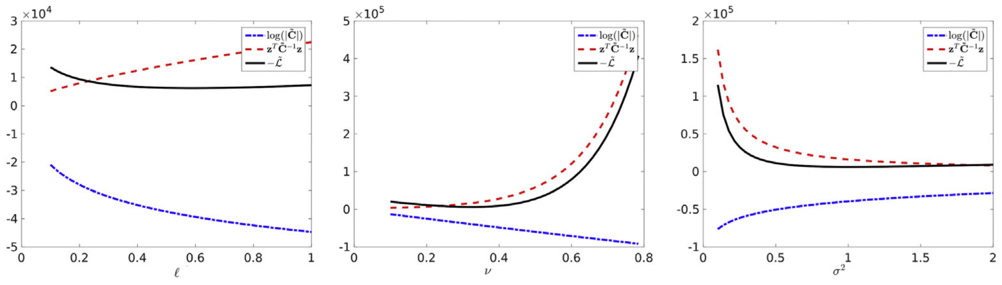

Fig. 8. Dependence of the negative log-likelihood and its ingredients on parameters ℓ (on the left); ν (in the middle); and σ^2 (on the right). In each experiment the other two parameters are always fixed: ($\nu = 0.325$, $\sigma^2 = 0.98$) on the left, ($\ell = 0.62$, $\sigma^2 = 0.98$) in the middle; ($\ell = 0.62$, $\nu = 0.325$) on the right; $n = 64,000$.

Table 6
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.

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

a nugget, we redefine the original matrix as $\tilde{\mathbf{C}} := \tilde{\mathbf{C}} + \tau^2 \mathbf{I}$. Below, we analyze how the loglikelihood function, as well as its maximum are changing by this.

We assume $\|\tilde{\mathbf{C}}\| \neq 0$. For a small perturbation matrix $\mathbf{E}$ [10], it holds that

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

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

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

From (5.1), we see that the relative error on the left-hand side of (5.1) depends on the norm $\|\tilde{\mathbf{C}}^{-1}\|$, i.e., the relative error is inversely proportional to the smallest singular value of $\tilde{\mathbf{C}}$. 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 [4]. The approximation errors are proportional to the $\kappa(\tilde{\mathbf{C}})$, i.e., matrices with a very large condition number may require a very large $\mathcal{H}$-matrix rank.

Fig. 9 (left) demonstrates three negative log-likelihood functions computed with the nuggets 0.01, 0.005, and 0.001. For this particular example, the behavior of the likelihood is preserved, and the

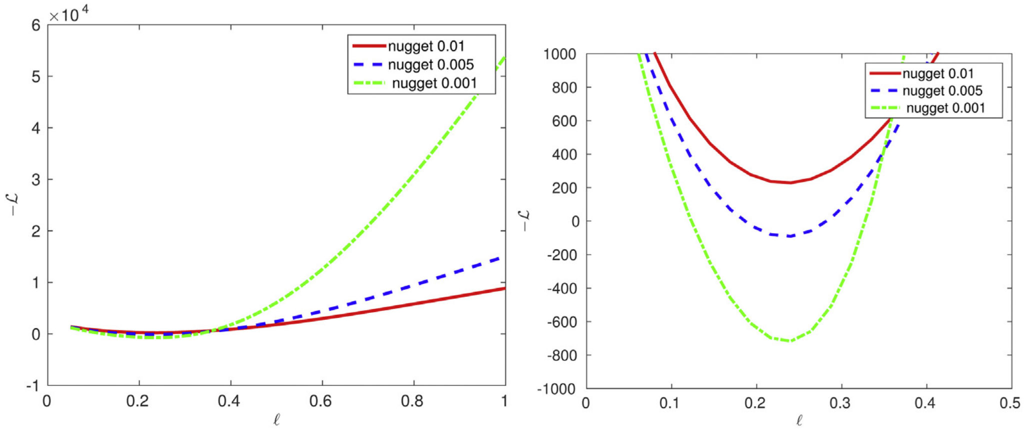

Fig. 9. (left) Dependence of the log-likelihood on parameter ℓ with nuggets ($\{0.01, 0.005, 0.001\}$) for Gaussian covariance. (right) Zoom of the left figure near minimum; $n = 2000$ random locations, rank $k = 14$, $\sigma^2 = 1$.

minimum does not change (or changes very slightly). Fig. 9 (right) is just a zoomed version of the picture on the left.

6 Best practices (HLIBCov)

In this section, we list our recommendations and warnings.

For practical computations, use adaptive-rank arithmetic since it produces smaller matrices and faster runtime.

For the input, it is sufficient to define a file by three columns in 2D and four columns in 3D: location coordinates (x, y, z) and the observed value; no triangles or edges are required.

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.

By default, the $\mathcal{H}$-Cholesky or $\mathcal{H}-$LU factorizations use a task-based approach employing a DAG (directed acyclic graph). For sequential computations this can be turned off to revert to a slightly faster recursive implementation by setting

```
HLIB::CFG::Arith::use_dag = false
```

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()`).

Since HLIBpro is working in $d = 1, 2, 3, \ldots$-dimensional case, only very minor changes are required to move from 1D locations to 2D or 3D (or higher). Replace dim= 2 with dim= 3 in

```
TCoordinate coord(vertices, dim);
```

then add “> >z” to

```
in > >x > >y > >z > >v;
```

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 and a vector of observations $\mathbf{Z}$. 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, and an approximation error.

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 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 4, 5) 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 desktop machine (Table 5). 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 research reported in this publication was supported by funding from the Alexander von Humboldt foundation (chair of Mathematics for Uncertainty Quantification at RWTH Aachen) and Extreme Computing Research Center (ECRC) at King Abdullah University of Science and Technology (KAUST).

Appendix A. Admissibility condition

Here we give an example of the admissibility criteria [15,12,8]. Let

$$\mathrm{cov}(x,y) := \log|x-y|, \quad x,y \in \mathbb{R}^d, \tag{A.1}$$

with singularity at $x = y$. We will introduce a condition, which divides all sub-blocks into admissible and inadmissible. Admissible blocks will be approximated by low-rank matrices.

Definition A.1. Let I be an index set of all degrees of freedom, i.e. $I = \{1, 2, \ldots, n\}$. Denote for each index $i \in I$ corresponding to a basis function b_i the support $\mathcal{G}_i := \mathrm{supp} b_i \subset \mathbb{R}^d$.

Let $\tau, \delta \in T_I$ be two clusters (elements of the cluster tree T_I). Clusters τ, δ are subsets of I, i.e. $\tau, \delta \subseteq I$. We generalise $\mathcal{G}_i$ to clusters $\tau \in T_I$ by setting $\mathcal{G}_\tau := \bigcup_{i\in\tau} \mathcal{G}_i$, i.e., $\mathcal{G}_\tau$ is the minimal subset of $\mathbb{R}^d$ that contains the supports of all basis functions b_i with $i \in \tau$.

Suppose that $\mathcal{G}_\tau \subset \mathbb{R}^d$ and $\mathcal{G}_\delta \subset \mathbb{R}^d$ are compact and $\chi(x, y)$ is defined for $(x,y) \in \mathcal{G}_\tau \times \mathcal{G}_\delta$ with $x \neq y$. The standard assumption on the kernel function in the $\mathcal{H}$-matrix theory is asymptotic smoothness of $\chi(x,y) \in C^\infty(\mathcal{G}_\tau \times \mathcal{G}_\delta)$, i.e, that

$$|\partial_x^\alpha \partial_y^\beta \chi(x,y)| \leq C_1 |\alpha+\beta|! C_0^{|\alpha+\beta|} \|x-y\|^{-|\alpha+\beta|-\gamma}, \quad \alpha, \beta \in \mathbb{N},$$

holds for constants C_1, C_0 and $\gamma \in \mathbb{R}$. This estimation is used to control the error ε_q from the Taylor expansion

$$\chi(x,y) = \sum_{\alpha\in\mathbb{N}_0^d, |\alpha|\leq q} (x-x_0)^\alpha \frac{1}{\alpha!} \partial_x^\alpha \chi(x_0,y) + \varepsilon_q.$$

Suppose that $\chi_k(x, y)$ is an approximation of χ in $\mathcal{G}_\tau \times \mathcal{G}_\delta$ of the separate form (e.g., Taylor or Lagrange polynomials):

$$\chi_k(x,y) = \sum_{\nu=1}^{k} \varphi_\nu(x)\psi_\nu(y), \tag{A.2}$$

where k is the rank of separation. We are aiming at an approximation of the form (A.2) such that exponential convergence

$$\|\chi - \chi_k\|_{\infty, \mathcal{G}_\tau \times \mathcal{G}_\delta} \leq \mathcal{O}(\eta^k) \tag{A.3}$$

holds.

Let $B_\tau, B_\delta \subset \mathbb{R}^d$ be axis-parallel bounding boxes of the clusters τ and δ such that $\mathcal{G}_\tau \subset B_\tau$ and $\mathcal{G}_\delta \subset B_\delta$.

Definition A.2. The *standard admissibility* condition (Adm_η), shown in Fig. 2 on the left, for two clusters τ and δ is

$$\min\{\mathrm{diam}(B_\tau), \mathrm{diam}(B_\delta)\} \leq \eta \mathrm{dist}(B_\tau, B_\delta). \tag{A.4}$$

Another example is

$$\max\{\mathrm{diam}(B_\tau), \mathrm{diam}(B_\delta)\} \leq \eta \mathrm{dist}(B_\tau, B_\delta),$$

where η is some positive number.

Definition A.3. We will say that a pair (τ, δ) of clusters τ and $\delta \in T_I$ is admissible if the condition (A.4) is satisfied. The blocks for which condition (A.4) is true are called admissible blocks.

The admissibility condition indicates blocks that allow rank-k approximation and those that do not. Admissible blocks are either very small (and computed exactly) or are approximated by rank-k matrices. All other (inadmissible) blocks are computed as usual.

In order to get a simpler partitioning (see an example in Fig. 2, right), we define the *weak admissibility condition* Adm_W for a pair (τ, δ):

$$\text{Block } b = \tau \times \delta \in T_{I\times I} \quad \text{is weak admissible} \Leftrightarrow ((\text{b is a leaf}) \text{ or } \delta \neq \tau), \tag{A.5}$$

where τ, δ are assumed to belong to the same level of $T_{I\times I}$.

See more details about derivation of admissibility condition for covariance matrices in [24].

Appendix B. 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) [11] and its improved modifications such as ACA+ and HACA [3,5,7].

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

References

[1] S. Ambikasaran, D. Foreman-Mackey, L. Greengard, D.W. Hogg, M. O'Neil, Fast direct methods for Gaussian processes and the analysis of NASA Kepler mission data, (2014) arXiv preprint arXiv:1403.6015.
[2] S. Ambikasaran, J.Y. Li, P.K. Kitanidis, E. Darve, Large-scale stochastic linear inversion using hierarchical matrices, Comput. Geosci. 17 (6) (2013) 913–927.
[3] M. Bebendorf, Approximation of boundary element matrices, Numer. Math. 86 (4) (2000) 565–589.
[4] M. Bebendorf, T. Fischer, On the purely algebraic data-sparse approximation of the inverse and the triangular factors of sparse matrices, Numer. Linear Algebra Appl. 18 (1) (2011) 105–122.
[5] M. Bebendorf, S. Rjasanow, Adaptive low-rank approximation of collocation matrices, Computing 70 (1) (2003) 1–24.
[6] S. Börm, J. Garcke, Approximating Gaussian processes with H^2-matrices, in: J.N. Kok, J. Koronacki, R.L. de Mantaras, S. Matwin, D. Mladen, A. Skowron (Eds.), Proceedings of 18th European Conference on Machine Learning, Warsaw, Poland, September 17-21, 2007. ECML 2007, volume 4701, 2007, pp. 42–53.
[7] S. Börm, L. Grasedyck, Hybrid cross approximation of integral operators, Numer. Math. 101 (2) (2005) 221–249.
[8] S. Börm, L. Grasedyck, W. Hackbusch, Hierarchical Matrices, volume 21 of Lecture Note, Max-Planck Institute for Mathematics, Leipzig, 2003 www.mis.mpg.de.
[9] R.P. Brent, Chapter 4: An algorithm with Guaranteed Convergence for Finding a Zero of a Function, Algorithms for Minimization without Derivatives, Prentice-Hall, Englewood Cliffs, NJ, 1973.
[10] J. Demmel, The componentwise distance to the nearest singular matrix, SIAM J. Matrix Anal. Appl. 13 (1) (1992) 10–19.

[11] S.A. Goreinov, E.E. Tyrtyshnikov, N.L. Zamarashkin, A theory of pseudoskeleton approximations, Linear Algebra Appl. 261 (1997) 1–21.
[12] L. Grasedyck, W. Hackbusch, Construction and arithmetics of $\mathcal{H}$-matrices, Computing 70 (4) (2003) 295–334.
[13] L. Grasedyck, R. Kriemann, S. LeBorne, Parallel black box H-LU preconditioning for elliptic boundary value problems, Comput. Visualization Sci. 11 (4-6) (2008) 273–291.
[14] P. Guttorp, T. Gneiting, Studies in the history of probability and statistics XLIX: On the Matérn correlation family, Biometrika 93 (2006) 989–995.
[15] W. Hackbusch, A sparse matrix arithmetic based on $\mathcal{H}$-matrices. I. Introduction to $\mathcal{H}$-matrices, Computing 62 (2) (1999) 89–108.
[16] W. Hackbusch, Hierarchical matrices: Algorithms and Analysis, volume 49 of Springer Series in Comp. Math, Springer, 2015.
[17] W. Hackbusch, B.N. Khoromskij, A sparse $\mathcal{H}$-matrix arithmetic. II. Application to multi-dimensional problems, Computing 64 (1) (2000) 21–47.
[18] W. Hackbusch, B.N. Khoromskij, R. Kriemann, Hierarchical matrices based on a weak admissibility criterion, Computing 73 (3) (2004) 207–243.
[19] N. Halko, P.G. Martinsson, J.A. Tropp, Finding Structure with Randomness: Probabilistic Algorithms for Constructing Approximate Matrix Decompositions, SIAM Rev. 53 (2) (2011) 217–288.
[20] M.S. Handcock, M.L. Stein, A Bayesian analysis of kriging, Technometrics 35 (1993) 403–410.
[21] H. Harbrecht, M. Peters, M. Siebenmorgen, Efficient approximation of random fields for numerical applications, Numer. Linear Algebra Appl. 22 (4) (2015) 596–617.
[22] Y.P. Hong, C.-T. Pan, Rank-revealing qr factorizations and the singular value decomposition, Math. Comput. 58 (197) (1992) 213–232.
[23] B.N. Khoromskij, A. Litvinenko, Data sparse computation of the Karhunen-Loève expansion, AIP Conference Proceedings 1048 (1) (2008) 311.
[24] B.N. Khoromskij, A. Litvinenko, H.G. Matthies, Application of hierarchical matrices for computing the Karhunen-Loève expansion, Computing 84 (1-2) (2009) 49–67.
[25] R. Kriemann, Parallel H-matrix arithmetics on shared memory systems, Computing 74 (3) (2005) 273–297.
[26] R. Kriemann, Parallele Algorithmen für $\mathcal{H}$-Matrizen, University of Kiel, 2005 PhD thesis.
[27] R. Kriemann, HLIBpro user manual, Max Planck Institute for Mathematics in the Sciences, 2008 Technical report.
[28] R. Kriemann, H-LU factorization on many-core systems, Comput. Visualization Sci. 16 (3) (2013 Jun) 105–117.
[29] A. Litvinenko, Application of hierarchical matrices for solving multiscale problems, Leipzig University, 2006 PhD Dissertation.
[30] A. Litvinenko, H.G. Matthies, Sparse data representation of random fields, PAMM 9 (1) (2009) 587–588.
[31] A. Litvinenko, Y. Sun, M.G. Genton, D.E. Keyes, Likelihood approximation with hierarchical matrices for large spatial datasets, Comput. Stat. Data Anal. 137 (2019) 115–132.
[32] B. Matérn, Spatial Variation, volume 36 of Lecture Notes in Statistics, second edition, Springer-Verlag, Berlin; New York, 1986.
[33] W.H. Press, S.A. Teukolsky, W.T. Vetterling, B.P. Flannery, Section 9.3. Van Wijngaarden-Dekker-Brent Method. Numerical Recipes: The Art of Scientific Computing, volume 3rd ed, Cambridge University Press, New York, 2007.