Peter Benner Thomas Mach

Computing all or some Eigenvalues of symmetric $\mathcal{H}_\ell$-Matrices

Max Planck Institute Magdeburg
Preprints

MPIMD/10-01 November 17, 2010

Impressum:

Max Planck Institute for Dynamics of Complex Technical Systems, Magdeburg

Publisher:
Max Planck Institute for Dynamics of Complex Technical Systems

Address:
Max Planck Institute for Dynamics of Complex Technical Systems
Sandtorstr. 1
39106 Magdeburg

www.mpi-magdeburg.mpg.de/preprints

COMPUTING ALL OR SOME EIGENVALUES OF SYMMETRIC $\mathcal{H}_\ell$-MATRICES

PETER BENNER AND THOMAS MACH

Abstract. We use a bisection method, [Par80, p. 51], to compute the eigenvalues of a symmetric $\mathcal{H}_\ell$-matrix M. The number of negative eigenvalues of $M-\mu I$ is computed via the LDL^T factorisation of $M-\mu I$. For dense matrices, the LDL^T factorisation is too expensive to yield an efficient eigenvalue algorithm in general, but not for $\mathcal{H}_\ell$-matrices. In the special structure of $\mathcal{H}_\ell$-matrices there is an LDL^T factorisation with linear-polylogarithmic complexity. The bisection method requires only matrix-size independent many iterations to find an eigenvalue up to the desired accuracy, so that an eigenvalue can be found in linear-polylogarithmic time. For all n eigenvalues, $\mathcal{O}\left(n^2 (\log n)^4 \log\left(\|M\|_2/\epsilon_{\text{ev}}\right)\right)$ flops are needed to compute all eigenvalues with an accuracy ϵ_{ev}. It is also possible to compute only eigenvalues in a specific interval or the j-th smallest one. Numerical experiments demonstrate the efficiency of the algorithm, in particular for the case where some interior eigenvalues are required.

Keywords: symmetric hierarchical matrices, eigenvalues, $\mathcal{H}_\ell$-matrices, hierachically semiseparable matrices, HSS matrices, slicing the spectrum

Mathematics Subject Classification: 65F15, 65F50, 15A18

1. Introduction

In "The Symmetric Eigenvalue Problem", Beresford N. Parlett describes a bisection method to find the eigenvalues of a symmetric matrix $M \in \mathbb{R}^{n\times n}$ [Par80, p. 51]. He calls this process "slicing the spectrum". The spectrum Λ of a real, symmetric matrix is contained in $\mathbb{R}$ and so the following question is well posed: How many eigenvalues $\lambda_i \in \Lambda$ are smaller than μ? We will call this number $\nu(\mu)$ or $\nu(M-\mu I)$. Obviously, ν is a function $\mathbb{R} \to \{0,\dots,n\} \subset \mathbb{N}_0$. If the function $\nu(\cdot)$ is known, one can find the m-th eigenvalue as the limit of the following process:

- **a:** Start with an interval $[a,b]$ with $\nu(a) < m \leq \nu(b)$.
- **b:** Determine $\nu_m := \nu(\frac{a+b}{2})$. If $\nu_m > m$, then continue with the interval $[a, \frac{a+b}{2}]$, else with $[\frac{a+b}{2}, b]$.
- **c:** Repeat the bisection (step b) until the interval is small enough.

The function $\nu(\cdot)$ can be evaluated using the LDL^T factorisation of $M-\mu I$, since Sylvester's inertia law implies that the number of negative eigenvalues is invariant under congruence transformations. For dense matrices the evaluation of ν is expensive. So this method is not recommended if no special structure, like tridiagonality, is available.

Here we consider $\mathcal{H}_\ell$-matrices, which have such a special structure. $\mathcal{H}_\ell$-matrices can be regarded as the simplest form of $\mathcal{H}$-matrices [Hac99]. They include, among others, tridiagonal and numerous finite-element matrices. We will see in the next section that the LDL^T factorisation for $\mathcal{H}_\ell$-matrices (for all shifts) can be computed in linear-polylogarithmic complexity. We will further see that

$$\mathcal{O}\left(n^2 k^2 (\log n)^4 \log\left(\|M\|_2/\epsilon_{\text{ev}}\right)\right) \text{ flops} \tag{1}$$

are sufficient to find all eigenvalues with an accuracy of ϵ_{ev}, where k is the maximal rank of the admissible submatrices.

There are other eigenvalue algorithms for symmetric $\mathcal{H}_\ell$-matrices. In [Gör09], an eigenvalue algorithm for $\mathcal{H}_\ell(1)$-matrices based on Divide-and-Conquer is described. This algorithm, if combined with an efficient strategy based on an efficient solver for H_ℓ-matrices like proposed in [BG10],

Date: November 17, 2010.

has a total complexity of $\mathcal{O}(n^2 (\log n)^\beta)$. Further in [DFV09], a transformation of $\mathcal{H}_\ell$- and the related HSS-matrices into semi-separable matrices is presented, symmetry is not needed. For semi-separable matrices there is a QR-algorithm [VVM05]. Both steps have quadratic or quadratic-polylogarithmic complexity.

The complexity of the LDL^T slicing algorithm is competitive with the existing ones if we are interested in all eigenvalues. If we are interested only in some (interior) eigenvalues, the algorithm will be superior, since the two others mentioned in the previous paragraph have to compute all eigenvalues. The LDL^T slicing algorithm is fundamentally different from the two other algorithms. The computational complexity depends logarithmically on the wanted accuracy, so that it is really cheap to get a sketch of the eigenvalue distribution. In contrast, the algorithm can compute one eigenvalue, e.g., the smallest, second smallest, or 42nd smallest, without computing any other eigenvalue in almost linear complexity.

In the next subsection, we will cite some definitions. Especially the definitions of $\mathcal{H}_\ell$- and $\mathcal{H}$-matrices will be used in the following sections. Further, we will make a small change in the definition of $\mathcal{H}_\ell$-matrices, which increases the computational efficiency. We allow the matrices on the lowest level to be of size $n_{\min} \times n_{\min}$ and not only of size 1×1.

1.1. **Definitions.**

Hierarchical ($\mathcal{H}$-) matrices were introduced by W. Hackbusch in 1998 [Hac99]. In that paper the $\mathcal{H}_\ell$-matrices are mentioned in Section 2.2.2, too. The $\mathcal{H}_\ell$-matrices can be regarded as the simplest form of $\mathcal{H}$-matrices.

The following definition of $\mathcal{H}_\ell$-matrices is given in [Hac09, p. 43] and [Gör09].

Definition 1.1. *($\mathcal{H}_\ell$-matrix)*
Let $I = \{1, \ldots, n\}$ be an index set and $n = 2^\ell$ with $\ell \in \mathbb{N}$. A matrix $M \in \mathbb{R}^{I \times I}$ is called an $\mathcal{H}_\ell$-matrix of block-wise rank k, short $M \in \mathcal{H}_\ell(k)$, if it fulfils the following recursive conditions:

(1) *$n_0 = 1$ / $\ell = 0$: $M \in \mathcal{H}_0(k)$ if $M \in \mathbb{R}^{1\times 1}$ and*
(2) *$n_\ell = 2^\ell$: M is partitioned in*

$$M = \begin{bmatrix} M_{11} & M_{12} \\ M_{21} & M_{22} \end{bmatrix}$$

with $M_{11}, M_{22} \in \mathcal{H}_{\ell-1}(k)$, $M_{12} = A_1 B_1^T$ and $M_{21} = B_2 A_2^T$, where $A_i, B_i \in \mathbb{R}^{n_{\ell-1} \times k'}$, with $k' \leq k$.

We are interested only in symmetric $\mathcal{H}_\ell$-matrices, so we have $M_{12} = M_{21}^T$, $A_1 = A_2$ and $B_1 = B_2$. A symmetric $\mathcal{H}_3$-matrix is depicted in Figure 1.

Further we will need the concept of $\mathcal{H}$-matrices. We will give a short definition of $\mathcal{H}$-matrices here, for details see [Hac09] or [GH03]. We define some necessary terms first. A hierarchical tree, short $\mathcal{H}$-tree, T_I of an index set I is a tree with the special conditions:

- the index set I is the root of T_I and
- a vertex $r \in T_I$ is either the disjoint union of its sons $s \in S(r)$ or a leaf of T_I.

The set of sons of a vertex $r \in T_I$ is called $S(r)$. We denote the set of leaves (vertices without sons $S(\cdot) = \emptyset$) of the $\mathcal{H}$-tree T_I by $\mathcal{L}(T_I)$. The $\mathcal{H}$-tree T has a depth ℓ, which is the maximum length of the paths from the root to each leave. If cardinality or geometrically balanced clustering is used the depth of the tree is in $\mathcal{O}(\log n)$ [GH03, p. 320ff].

A hierarchical product tree, short $\mathcal{H}_\times$-tree, $T_{I\times I}$ is a special $\mathcal{H}$-tree over the index set $I \times I$ and can be regarded as the product $T_I \times T_I$. Every vertex of $T_{I\times I}$ is the product of two vertices of the same level of the $\mathcal{H}$-tree T_I.

Now we are able to define the set of $\mathcal{H}$-matrices based on the $\mathcal{H}_\times$-tree $T_{I\times I}$ with maximum block-wise rank k and the minimum block size $n_{\min}$ by

$$\mathcal{H}(T_{I\times I}, k) := \left\{ M \in \mathbb{R}^{I\times I} \,\middle|\, \begin{array}{c} \forall r \times s \in \mathcal{L}(T_{I\times I}) : \operatorname{rank}(M_{r\times s}) \leq k \\ \text{or } \#r \leq n_{\min} \text{ or } \#s \leq n_{\min} \end{array} \right\}.$$

The low rank matrices $M_{r\times s}$ are stored in factored form AB^T. There are a lot of arithmetic operations for $\mathcal{H}$-matrices with linear-polylogarithmic complexity [Hac09, Beb08, Gra01].

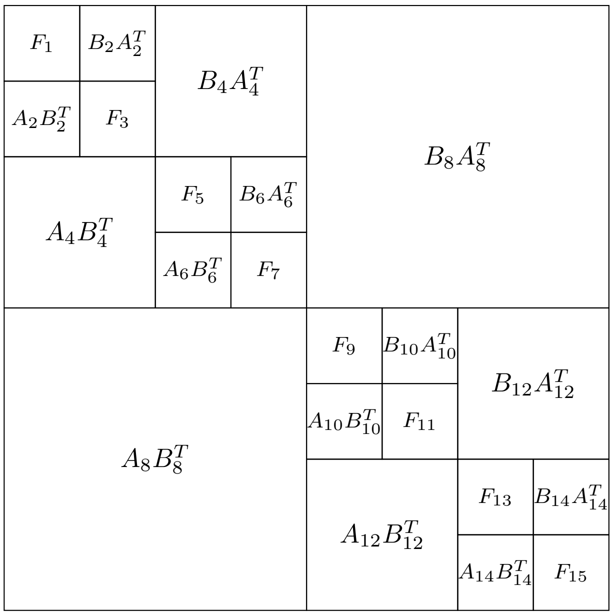

FIGURE 1. Structure of an $\mathcal{H}_3$-matrix.

We divide the set of leaves in admissible leaves $\mathcal{L}^+(T)$ and inadmissible leaves $\mathcal{L}^-(T)$. The submatrices corresponding to admissible leaves have at most rank k and will be stored as so called $\mathbb{R}^k$-matrices AB^T, with $k = \operatorname{rank}\left(AB^T\right)$. The submatrices corresponding to inadmissible leaves will be stored in the standard way as dense matrices without any approximation.

Lemma 1.2. *If $M \in \mathcal{H}_\ell(k)$, then M is a hierarchical matrix of block-wise rank k, too.*

Proof. The minimum block-size $n_{\min}$ is 1. The $\mathcal{H}$-tree T_I is a binary tree, which divides each node $r = \{i_1, \ldots, i_m\}$ into $r_1 = \left\{i_1, \ldots, i_{m/2}\right\}$ and $r_2 = \left\{i_{m/2+1}, \ldots, i_m\right\}$ on the next level. In the $\mathcal{H}_\times$-tree only nodes of the type $r \times r$ are subdivided. The other nodes $r \times s$, with $r \cap s = \emptyset$, correspond to blocks M_{12} or M_{21}, which have at most rank k. $\square$

In the format of hierarchical matrices, blocks of size lower than $n_{\min}$ are stored in the dense matrix format. The hierarchical structure is not efficient for small matrices, since the overhead costs are too large. We will do the same for $\mathcal{H}_\ell$-matrices. We change in Definition 1.1 condition (1) to:

(1') $\ell = 0$: $n_0 \leq n_{\min}$ and $M \in \mathcal{H}_0(k)$ if $M \in \mathbb{R}^{n_0 \times n_0}$.

So the size of a matrix $M \in \mathcal{H}_\ell$ is increased to $n = 2^\ell n_0$. Lemma 1.2 holds for matrices fulfilling the new definition, too.

Each tridiagonal matrix $T \in \mathbb{R}^{2^\ell \times 2^\ell}$ is an $\mathcal{H}_\ell$-matrix. Due to that inclusion we should not expect to find faster eigenvalue algorithms for $\mathcal{H}_\ell$-matrices than for tridiagonal matrices. The best known eigenvalue algorithms for symmetric tridiagonal matrices have quadratic complexity.

The hierarchically semiseparable matrices (HSS-matrices) form an important subset of the $\mathcal{H}_\ell$-matrices. There are a lot of publications [CGL05, CGP06, XCGL09, CG01, DC06] from Chandrasekaran et al. giving all similar definitions based on hierarchically semiseparable representation. We need a slight different definition more emphasising the relationship to $\mathcal{H}_\ell$-matrices, but we will use the same index-system like is usual for HSS-matrices: if $|r - s| = 1$, then $M|_{(r-1)2^{\ell-j}n_{\min}+1:r2^{\ell-j}n_{\min},(s-1)2^{\ell-j}n_{\min}+1:s2^{\ell-j}n_{\min}} = M_{j;r,s} = A_{j;r,s}B_{j;r,s}^T$ is an off-diagonal block.

Definition 1.3. *(HSS-matrices)*
Let $M \in \mathcal{H}_\ell(k)$. We will call M a hierarchically semiseparable matrix of (HSS) rank k, or short $M \in HSS(k)$, if M fulfils the following conditions:

- $\forall i \in \{1,\ldots,2^\ell\} : \exists U_{\ell;i} \in \mathbb{R}^{n_{\min}\times k}$ *and* $\exists U_{j;r} \in \mathbb{R}^{2^{\ell-j}n_{\min}\times k}, j = 1,\ldots,\ell-1, r = 1,\ldots,2^j$ *with* $\operatorname{range}(U_{j;r}) \subset \operatorname{span}\left(\begin{bmatrix} U_{j+1;2r-1} \\ 0 \end{bmatrix}\right) \oplus \operatorname{span}\left(\begin{bmatrix} 0 \\ U_{j+1;2r} \end{bmatrix}\right)$ *and* $\operatorname{range}(A_{j;r,s}) \subset \operatorname{span}(U_{j;r})$*;*
- $\forall i \in \{1,\ldots,2^\ell\} : \exists V_{\ell;i} \in \mathbb{R}^{n_{\min}\times k}$ *and* $\exists V_{j;s} \in \mathbb{R}^{2^{\ell-j}n_{\min}\times k}, j = 1,\ldots,\ell-1, s = 1,\ldots,2^j$ *with* $\operatorname{range}(V_{j;s}) \subset \operatorname{span}\left(\begin{bmatrix} V_{j+1;2s-1} \\ 0 \end{bmatrix}\right) \oplus \operatorname{span}\left(\begin{bmatrix} 0 \\ U_{V+1;2s} \end{bmatrix}\right)$ *and* $\operatorname{range}(B_{j;r,s}) \subset \operatorname{span}(V_{j;s})$.

Example 1.4. *Figure 1 shows an example of a symmetric $\mathcal{H}_3$-matrix. For this matrix the conditions of Definition 1.3 mean, that there are matrices $U_1,\ldots,U_8 \in \mathbb{R}^{n_{\min}\times k}$, with*

$$\begin{aligned}
&\operatorname{range}(B_2) \subset \operatorname{span}(U_1)\\
&\operatorname{range}(A_2) \subset \operatorname{span}(U_2)\\
&\operatorname{range}(B_4) \subset \operatorname{span}\left(\begin{bmatrix} U_1 \\ 0 \end{bmatrix}\right) \oplus \operatorname{span}\left(\begin{bmatrix} 0 \\ U_2 \end{bmatrix}\right)\\
&\operatorname{range}(B_6) \subset \operatorname{span}(U_3)\\
&\operatorname{range}(A_6) \subset \operatorname{span}(U_4)\\
&\operatorname{range}(A_4) \subset \operatorname{span}\left(\begin{bmatrix} U_3 \\ 0 \end{bmatrix}\right) \oplus \operatorname{span}\left(\begin{bmatrix} 0 \\ U_4 \end{bmatrix}\right)\\
&\operatorname{range}(B_8) \subset \operatorname{span}\left(\begin{bmatrix} U_1 \\ 0 \\ 0 \\ 0 \end{bmatrix}\right) \oplus \operatorname{span}\left(\begin{bmatrix} 0 \\ U_2 \\ 0 \\ 0 \end{bmatrix}\right) \oplus \operatorname{span}\left(\begin{bmatrix} 0 \\ 0 \\ U_3 \\ 0 \end{bmatrix}\right) \oplus \operatorname{span}\left(\begin{bmatrix} 0 \\ 0 \\ 0 \\ U_4 \end{bmatrix}\right)\\
&\qquad\vdots
\end{aligned}$$

One can show, that each $\mathcal{H}_\ell(k)$-matrix is an HSS($k\ell$)-matrix, too. In the next chapter we will explain detailed how to compute all or some eigenvalues of symmetric $\mathcal{H}_\ell$-matrices..

2. Slicing the Spectrum by LDL^T Factorisation

In this section the details of the slicing algorithm, mentioned in the first section, will be explained. Essentially we use a bisection method halving the intervals $[a_i, b_i]$, which contain the searched eigenvalue λ_i, in each step. This process is stopped if the interval is small enough.

We will employ Algorithm 1. If the function ν is computed exactly, the algorithm will choose the part of the interval containing λ_i. The algorithm needs $\mathcal{O}(\log_2((b-a)/\epsilon_{\text{ev}}))$ iterations to reduce the interval to size ϵ_{ev}. We know $\lambda_i \in [a_i, b_i]$, $b_i - a_i < \epsilon_{\text{ev}}$ and $\hat{\lambda}_i = (b_i - a_i)/2$. And so it holds, that

$$\left|\lambda_i - \hat{\lambda}_i\right| < \frac{1}{2}\epsilon_{\text{ev}}. \tag{2}$$

The evaluation of the function $\nu(\cdot)$ is the topic of the next subsection.

2.1. The Function $\nu(M - \mu I)$.

We recall some basic linear algebra facts.

Definition 2.1. *Two square matrices M and N are* congruent *if there exists an invertible matrix P such that*

$$P^T M P = N. \tag{3}$$

Further we will use Sylvester's inertia law:

Theorem 2.2. *(Sylvester's inertia law, e.g.,* [Par80, p. 11]*)*
Each square matrix M is congruent to a matrix $\operatorname{diag}(-I_\nu, 0_\xi, I_{n-\nu-\xi})$, where ν is the number of negative eigenvalues, ξ the number of zero eigenvalues, and $n-\nu-\xi$ is the number of positive eigenvalues. The triple $(\nu, \xi, n-\nu-\xi)$ is called M's inertia.

Algorithm 1: Slicing the spectrum [Par80, p. 50ff]

```
Input: $M \in \mathcal{H}(T_{I\times I})$, with $|I| = n$ and $a, b \in \mathbb{R}$, so that $\Lambda(M) \subset [a, b]$;
Output: $\left\{\hat{\lambda}_1, \dots, \hat{\lambda}_n\right\} \approx \Lambda(M)$;
1  for $i = 1, \dots, n$ do
2      $b_i := b$;   $a_i := a$;
3      while $b_i - a_i \geq \epsilon_{ev}$ do
4          $\mu := (b_i - a_i)/2$;
5          $[L, D] :=$ LDL$^T$ factorisation$(M - \mu I)$;
6          $\nu(M - \mu I) := |\{j | D_{jj} < 0\}|$;
7          if $\nu(M - \mu I) \geq i$ then $b_i := \mu$ else $a_i := \mu$ ;
8      end
9      $\hat{\lambda}_i := (b_i - a_i)/2$;
10 end
```

If $M - \mu I$ has an LDL^T factorisation $M - \mu I = LDL^T$ with L invertible, then D and $M - \mu I$ are congruent. Since D is diagonal we can count easily the number of positive or negative eigenvalues. Sylvester's inertia law tells us, that the number of negative diagonal entries in D is equal to the number of negative eigenvalues of $M - \mu I$, $\nu(D) = \nu(\mu)$.

If a diagonal entry of D is zero, we have shifted with an eigenvalue. In this case one of the leading principal submatrices of $M - \mu I$ is rank deficient and the LDL^T factorisation may fail, which in this case is a welcome event as an eigenvalue has been found.

We investigate the LDL^T factorisation of $\mathcal{H}_\ell$-matrices in the next subsection.

2.2. LDL^T Factorisation of $\mathcal{H}_\ell$-Matrices.

Definition 2.3. *(LDL^T factorisation* [GV96]*)*
If $M \in \mathbb{R}^{n\times n}$ is a symmetric matrix and all the leading principal submatrices of M are invertible, then there exists a unit lower triangular matrix L and a diagonal matrix $D = \operatorname{diag}(d_1, \dots, d_n)$ such that $M = LDL^T$. We will call this factorisation LDL^T factorisation *or* LDL^T decomposition*.*

There is an algorithm to compute LDL^T factorisations for hierarchical matrices, first described in [Lin02, p. 70]. The $\mathcal{H}$-LDL^T factorisation is block recursive, see Algorithm 2. For a hierarchical matrix $M \in \mathcal{H}(T, k)$ this factorisation has a complexity of

$$\mathcal{O}(nk^2 (\log n)^2) \tag{4}$$

in fixed rank $\mathcal{H}$-arithmetic. We note that the LDL^T factorisation for $\mathcal{H}$-matrices is much cheaper than for dense matrices, where $\mathcal{O}(n^3)$ flops are needed. In standard arithmetic, the stability of the factorisation is improved by e.g. Bunch-Kaufmann pivoting [BK77]. Since pivoting would destroy the hierarchical structure, pivoting can not be used here. Many practical problems lead to diagonal dominant matrices and at least for them pivoting is not necessary for good results. Here we need only an exact evaluation of $\nu(\mu)$, and for this we not necessarily require a high accurate LDL^T factorisation.

We will use Algorithm 2 for $\mathcal{H}_\ell$-matrices, too. In this case the solution of the equation

$$L_{21} D_1 L_{11}^T = M_{21}$$

is simplified, since $M_{21} = AB^T$. If D_1 has a zero entry the solution will fail. But in this case we know, that zero is an eigenvalue of M. After the computation of L_{21} an update is performed. This update increases in general the rank of the submatrix M_{22}. In fixed rank $\mathcal{H}$-arithmetic the update is followed by a truncation step, which reduces the rank again to k. For $\mathcal{H}_\ell$-matrices we will omit the truncation, since the growth of the block-wise ranks is bounded. The next lemma gives this bound, which will be used for the complexity analysis of Algorithm 2.

Algorithm 2: $\mathcal{H}$-LDL^T-factorisation $M = LDL^T$

```
1 $\mathcal{H}$-LDL$^T$-factorisation($M$);
  Input: $M \in \mathcal{H}(T)$
  Output: $L \in \mathcal{H}(T)$, $D = \operatorname{diag}(d_1, \ldots, d_n)$ with $LDL^T = M$ and $L$ lower triangular
2 if $M = \left[\begin{smallmatrix} M_{11} & M_{12} \\ M_{21} & M_{22} \end{smallmatrix}\right] \notin \mathcal{L}(T)$ then
3     $[L_{11}, D_1] :=$ $\mathcal{H}$-LDL$^T$-factorisation($M_{11}$);
4     Compute the solution $L_{21}$ of $L_{21} D_1 L_{11}^T = M_{21}$;
5     $[L_{22}, D_2] :=$ $\mathcal{H}$-LDL$^T$-factorisation($M_{22} - L_{21} D_1 L_{21}^T$);
6 else
7     Compute the dense LDL$^T$-factorisation $LDL^T = M$, since inadmissible diagonal blocks
      are stored as dense matrices.
8 end
9 return $L, D$;
```

Lemma 2.4. *Let $M \in \mathcal{H}_\ell(k)$. If the assumptions of Definition 2.3 are fulfilled, then the triangular matrix L of the LDL^T factorisation is an $\mathcal{H}_\ell(k\ell)$-matrix. Further the complexity of the computation of L and D by Algorithm 2 is*

$$\mathcal{O}(nk^2 (\log n)^4). \tag{5}$$

Proof. We will first proof the statement on the block-wise ranks and will then use this for the complexity estimation. We number the blocks of M like in Figure 1. Each block has a number out of the index set $S = \{1, \ldots, 2^{l+1} - 1\}$. First we will define some functions on S. The function $m(i)$ is defined by

$$m(i) = \begin{cases} 0, & \text{if } i \text{ is odd,} \\ 1 + m(i/2), & \text{if } i \text{ is even.} \end{cases}$$

The size of block i is $n_0 2^{\max\{m(i)-1,0\}} \times n_0 2^{\max\{m(i)-1,0\}}$. The next function $t(i)$ gives us the indices of blocks on the left hand side of block i:

$$t(i) = \begin{cases} \emptyset, & \text{if } i - 2^{m(i)} = 0, \\ t(i - 2^{m(i)}) \cup \{i - 2^{m(i)}\}, & \text{else.} \end{cases}$$

Finally we will need

$$u(i) = |t(i)|.$$

Algorithm 2 processes the blocks in the order of their numbering. Most operations of Algorithm 2 do not change the block-wise ranks, only the update in line 5 increases the rank of some blocks. We are interested in the final rank of block i. Obviously only updates from blocks $j \in t(i)$ act on i. We assume that $t(i) = \{j_1, j_2, \ldots, j_{u(i)}\}$. Let the smallest index in $t(i)$ be j_1. We compute the solution of $L_{11} D_1 L_{21}^T = M_{21}^T$ for block j_1. The matrix L_{21} is low rank and $L_{21} = A_{j_1} B_{L_{j_1}}$, with $B_{L_{j_1}}$ being the solution of $L_{11} D_1 B_{L_{j_1}} = B_{j_1}$. So the update $L_{21} D_1 L_{21}^T$ has the form $A_{j_1} X^T$ and is of rank k. After the update, block j_2 has the form $[A'_{j_1}, A_{j_2}][X', B_{j_2}]^T$, where $A'_{j(1)}$ and X' are the suitable parts of $A_{j(1)}$ and X. This means that the next update is of rank $2k$ and has the form $[A'_{j_1}, A_{j_2}]Y^T$. This second update increases the rank of block j_3 only by k since the block has the form $[A''_{j_1}, A_{j_3}]Z$ before the update. Finally block i has rank $k(u(i) + 1)$.

The maximum

$$\max_{i \in S} u(i) = \ell$$

is attained only for odd i. The rank of the inadmissible diagonal blocks is not of interest, since they are stored in dense matrix format. So the maximum rank of an admissible block is bounded by $k\ell$.

The $\mathcal{H}$-LDL^T factorisation does not change the hierarchical structure. So we have $L \in \mathcal{H}_\ell(k\ell)$. With Lemma 1.2, Equation (4) and $\ell = \mathcal{O}(\log n)$ we conclude, that the complexity of Algorithm 2 is in

$$\mathcal{O}(nk^2\,(\log n)^4).$$

□

The main difference between the LDL^T factorisation for $\mathcal{H}_\ell$-matrices and $\mathcal{H}$-matrices is that the factorisation for $\mathcal{H}_\ell$-matrices can be done without truncation and so exact up to round off respecting IEEE-double precision arithmetic.

If $k\ell$ is too large, we can use truncation like in the $\mathcal{H}$-LDL^T factorisation of general symmetric $\mathcal{H}$-matrices. The computed LDL^T factorisation is then only approximative, especially the computed $\hat{D}$ may differ from the exact D. So possibly for the computed $\hat{\nu}(M-\mu I) \neq \nu(M-\mu I)$. In this case we will continue the search for λ_i in an interval $[a_i, b_i]$, which does not contain λ_i. For example, let $\lambda_i > b_i$. If there is no other wrong decision, the algorithm will give us $\hat{\lambda}_i = b_i - \hat{\epsilon}$, with $\hat{\epsilon} < \epsilon_{\text{ev}}$. The difference between λ_i and $\hat{\lambda}_i$ is then bounded by the sum of ϵ_{ev} and $\lambda_i - b_i$.

Remark 2.5. *If the matrix $M \in \mathcal{H}_\ell(k)$ fulfils the following conditions (here exemplary for $\mathcal{H}_3(k)$ with the same notation as in Figure 1):*

$$\begin{aligned}
\operatorname{range}(A_4) &\subset \operatorname{span}\left(\begin{bmatrix} F_5 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ A_6 \end{bmatrix}\right),\\
\operatorname{range}(A_8) &\subset \operatorname{span}\left(\begin{bmatrix} F_9 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ A_{10} \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ \\ A_{12} \end{bmatrix}\right) \quad \textit{and}\\
\operatorname{range}(A_{12}) &\subset \operatorname{span}\left(\begin{bmatrix} F_{13} \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ A_{14} \end{bmatrix}\right),
\end{aligned}$$

or an analogue generalisation, then $L \in \mathcal{H}_\ell(k)$. E.g., tridiagonal matrices, generator representable semi-separable matrices, diagonal plus semi-separable matrices [VVM08] *and hierarchically semiseparable matrices are of this structure.*

2.3. **Start-interval** $[a, b]$**.** The interval $[a, b]$ must contain the whole spectrum. This is the case for $a := -\|M\|_2$ and $b := \|M\|_2$. The spectral norm $\|M\|_2$ can be approximated using the power iteration [Gra01]. We should test $\nu(a) = 0$ and $\nu(b) = n$. If $[a, b]$ does not contain all eigenvalues, we increase it until both test conditions are satisfied. The two tests cost only two additional $\mathcal{H}$-LDL^T factorisations of linear-polylogarithmic complexity.

2.4. **Complexity.** For each eigenvalue λ_i we have to do several $\mathcal{H}$-LDL^T factorisations to reduce the length of the interval $[a_i, b_i]$. Each factorisation halves the interval since we use a bisection method. So we need $\mathcal{O}(\log(\|M\|_2/\epsilon_{\text{ev}}))$ $\mathcal{H}$-LDL^T factorisations per eigenvalue. One $\mathcal{H}$-LDL^T has a complexity of $\mathcal{O}(nk^2(\log n)^4)$. Multiplying both complexities gives us the complexity per eigenvalue $\mathcal{O}(nk^2(\log n)^4 \log(\|M\|_2/\epsilon_{\text{ev}}))$ and the total complexity for all n eigenvalues:

$$\mathcal{O}(n^2k^2(\log n)^4 \log(\epsilon_{\text{ev}}\|M\|_2)). \tag{6}$$

3. Numerical Results

We have implemented Algorithm 1 with the LDL^T factorisation for $\mathcal{H}$-matrices, see Algorithm 2, using the $\mathcal{H}$lib [HLi09]. The $\mathcal{H}$lib can handle $\mathcal{H}_\ell$-matrices, too, since $\mathcal{H}_\ell$-matrices are a subset of $\mathcal{H}$-matrices. We use the fixed rank arithmetic of the $\mathcal{H}$lib, with the known maximal block-wise rank $k\ell$ for $\mathcal{H}_\ell(k)$-matrices. Further we choose a minimum block-size of $n_{\min} = 32$. The computations were done on an Intel® Core™i7 CPU 920 with 12 GB RAM.

To test the algorithm we use five different example series. The matrices have block-wise rank 1 or 2 in the admissible submatrices. The size of the matrices is varied from 64 to 1,048,576. The first two series are sets of randomly generated $\mathcal{H}_\ell$-matrices of block-wise rank 1 resp. 2. The next series are slightly different, since we use randomly generated HSS-matrices, fulfilling the

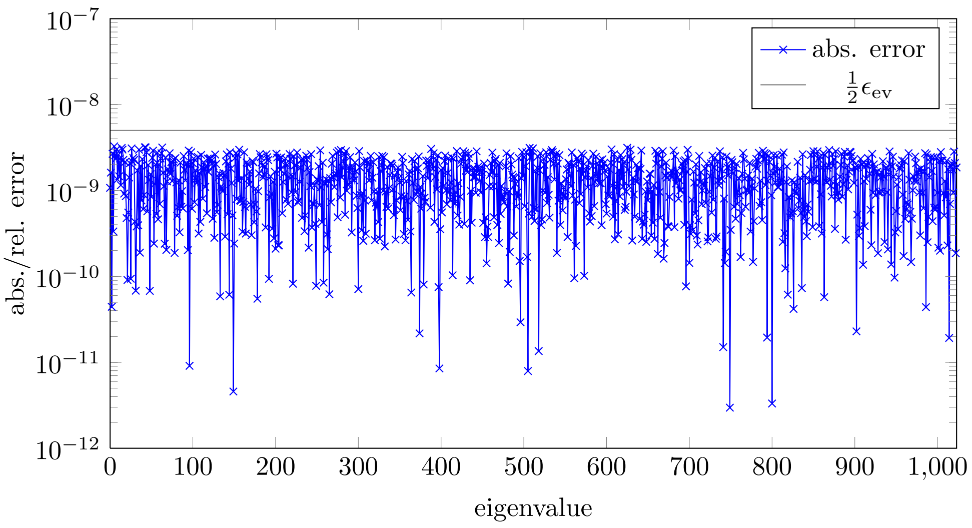

FIGURE 2. Absolute error $\left|\lambda_i - \hat{\lambda}_i\right|$ for a 1024×1024 matrix (H5 r1), $\epsilon_{\text{ev}} = 10^{-8}$.

additional conditions from Definition 1.3. The last series is a set of tridiagonal matrices with 2 on the diagonal and -1 on the subdiagonals, represent the discrete 1D Laplace operator. Except the tridiagonal matrices we normalize the matrices to $\|M\|_2 = 1$, since $\|M\|_2$ is part of the complexity estimate.

For the matrices up to dimension $32,768$ we compute their corresponding dense matrix and use the LAPACK-function `dsyev` [ABB+99] to compute the eigenvalues. The difference between the results of `dsyev` and the results from our new LDL^T slicing algorithm are the errors in the tables and figures. The time `dsyev` needs is given in the table, too. Note: there are faster algorithms for tridiagonal matrices but not for $\mathcal{H}_\ell$- and HSS-matrices in LAPACK.

Figure 2 shows the absolute errors of the computed eigenvalues of the matrix H5 r1 $\in \mathcal{H}_5(1)$ of size 1024. All the errors are below the expected bound.

Table 1 and Table 2 show the computation times and the errors for the five example series, if we compute only the 10 eigenvalues $\lambda_{n/4+5}, \ldots, \lambda_{n/4+14}$. (Similar results will be obtained when choosing other subsets of the spectrum.) The growth in the expected costs

$$\frac{N_i}{N_{i-1}} = \frac{n_i(\log n_i)^4}{n_{i-1}(\log n_{i-1})^4}$$

is given in the last column, k and $\|M\|_2$ are constant here. The computation times growth slower than expected until the fast storage is filled. This confirms the estimated computational complexity from Equation (5) and shows, that there is probably a tighter bound. The absolute resp. relative errors in the tables are computed by:

$$e_{\text{abs}} = \left\|\lambda_i - \hat{\lambda}_i\right\|_2 \qquad \text{resp.} \qquad e_{\text{rel}} = \left\|\frac{\lambda_i - \hat{\lambda}_i}{\lambda_i}\right\|_2 .$$

Tables 3 and 4 show the same as Tables 1 / 2 but for computing all eigenvalues, with $N_i = n_i^2(\log n_i)^4$. Table 3 shows that the computation of all eigenvalues with the LDL^T slicing algorithm is more expensive then using LAPACK. But since the transformation into a dense matrix requires n^2 storage, we are able to solve much larger problems by using the LDL^T slicing algorithm
.

Figure 3 and 4 compare the computation times with $\mathcal{O}(n\,(\log n)^\beta)$, $\beta = 0, 1, 2, 3, 4$. There we see that the β in the examples is rather 2 than 4.

[1]More than 12 GB RAM were required for the matrix H15 r2 and swapping parts of the RAM to disk slows down the program significantly.

Name	n	t_{LAPACK} in s	abs. Error	rel. Error	t in s	t_i/t_{i-1}	N_i/N_{i-1}
H1 r1	64	<0.01	5.36E-009	8.53E-009	0.01		
H2 r1	128	<0.01	5.91E-009	1.06E-008	0.03	3.00	3.71
H3 r1	256	0.14	6.52E-009	1.22E-008	0.08	2.67	3.41
H4 r1	512	0.10	5.32E-009	1.04E-008	0.18	2.25	3.20
H5 r1	1024	0.73	5.74E-009	1.14E-008	0.45	2.50	3.05
H6 r1	2048	5.87	5.79E-009	1.15E-008	1.08	2.40	2.93
H7 r1	4096	45.65	4.49E-009	8.95E-009	2.32	2.15	2.83
H8 r1	8192	371.48	3.92E-009	7.83E-009	5.54	2.39	2.75
H9 r1	16384	3180.03	5.72E-009	1.14E-008	12.40	2.24	2.69
H10 r1	32768	23849.26	4.88E-009	9.77E-009	23.22	1.87	2.64
H11 r1	65536	—	—	—	42.36	1.82	2.59
H12 r1	131072	—	—	—	96.52	2.28	2.55
H13 r1	262144	—	—	—	210.85	2.18	2.51
H14 r1	524288	—	—	—	427.05	2.03	2.48
H15 r1	1048576	—	—	—	1035.25	2.42	2.46
H1 r2	64	<0.01	5.35E-009	8.79E-009	0.01		
H2 r2	128	<0.01	4.58E-009	8.41E-009	0.04	*4.00*	3.71
H3 r2	256	0.02	5.35E-009	1.04E-008	0.11	2.75	3.41
H4 r2	512	0.10	5.24E-009	1.03E-008	0.30	2.73	3.20
H5 r2	1024	0.81	5.06E-009	1.00E-008	0.85	2.83	3.05
H6 r2	2048	6.15	6.94E-009	1.38E-008	2.42	2.85	2.93
H7 r2	4096	55.22	4.96E-009	9.90E-009	5.49	2.27	2.83
H8 r2	8192	432.61	3.77E-009	7.50E-009	13.43	2.45	2.75
H9 r2	16384	3232.65	4.72E-009	9.42E-009	30.07	2.24	2.69
H10 r2	32768	24043.06	5.08E-009	1.02E-008	57.71	1.92	2.64
H11 r2	65536	—	—	—	112.63	1.95	2.59
H12 r2	131072	—	—	—	270.13	2.40	2.55
H13 r2	262144	—	—	—	558.07	2.07	2.51
H14 r2	524288	—	—	—	1660.12	*2.97*	2.48
H15 r2	1048576	—	—	—	46664.91	*28.11*[1]	2.46
tri1	64	<0.01	5.26E-009	4.08E-009	<0.01		
tri2	128	<0.01	2.40E-009	2.10E-009	<0.01		3.71
tri3	256	0.01	3.58E-009	5.19E-009	<0.01		3.41
tri4	512	0.05	1.34E-011	1.94E-011	0.02		3.20
tri5	1024	0.36	5.07E-011	7.35E-011	0.03	1.50	3.05
tri6	2048	2.94	1.23E-011	1.78E-011	0.07	2.33	2.93
tri7	4096	23.48	1.89E-011	2.74E-011	0.12	1.71	2.83
tri8	8192	187.89	5.83E-011	8.45E-011	0.26	2.17	2.75
tri9	16384	1572.58	4.06E-011	5.88E-011	0.55	2.12	2.69
tri10	32768	12903.42	5.21E-011	7.55E-011	1.22	2.22	2.64
tri11	65536	—	—	—	2.45	2.01	2.59
tri12	131072	—	—	—	5.41	2.21	2.55
tri13	262144	—	—	—	12.89	2.38	2.51
tri14	524288	—	—	—	24.06	1.87	2.48
tri15	1048576	—	—	—	46.51	1.93	2.46

TABLE 1. Comparison of errors and computation times for the $\mathcal{H}_\ell$ and the tridiagonal example series computing only 10 eigenvalues ($n/4+5, \ldots, n/4+14$); *italic entries* are larger than expected.

Name	n	t_{LAPACK} in s	abs. Error	rel. Error	t in s	t_i/t_{i-1}	N_i/N_{i-1}
HSS1 r1	64	<0.01	3.67E-009	5.89E-009	<0.01		
HSS2 r1	128	<0.01	3.99E-009	5.95E-009	0.03		3.71
HSS3 r1	256	0.02	4.54E-009	7.65E-009	0.07	2.33	3.41
HSS4 r1	512	0.10	5.41E-009	1.12E-008	0.16	2.29	3.20
HSS5 r1	1024	0.76	6.20E-009	1.34E-008	0.34	2.13	3.05
HSS6 r1	2048	5.82	4.19E-009	7.65E-009	0.86	2.53	2.93
HSS7 r1	4096	45.78	4.16E-009	7.99E-009	2.01	2.34	2.83
HSS8 r1	8192	368.82	4.68E-009	8.50E-009	4.10	2.04	2.75
HSS9 r1	16384	3109.88	5.23E-009	9.88E-009	9.21	2.25	2.69
HSS10 r1	32768	23641.64	4.89E-009	9.61E-009	19.19	2.08	2.64
HSS11 r1	65536	—	—	—	36.81	1.92	2.59
HSS12 r1	131072	—	—	—	79.21	2.15	2.55
HSS13 r1	262144	—	—	—	185.65	2.34	2.51
HSS14 r1	524288	—	—	—	326.87	1.76	2.48
HSS15 r1	1048576	—	—	—	680.48	2.08	2.46
HSS1 r2	64	<0.01	5.48E-009	8.48E-009	0.01		
HSS2 r2	128	<0.01	6.07E-009	1.18E-008	0.04	4.00	3.71
HSS3 r2	256	0.02	4.59E-009	7.98E-009	0.08	2.00	3.41
HSS4 r2	512	0.10	6.32E-009	1.40E-008	0.21	2.63	3.20
HSS5 r2	1024	0.74	5.42E-009	1.04E-008	0.48	2.29	3.05
HSS6 r2	2048	5.85	4.32E-009	7.28E-009	1.17	2.44	2.93
HSS7 r2	4096	45.78	2.58E-009	4.28E-009	2.39	2.04	2.83
HSS8 r2	8192	354.75	3.64E-009	8.84E-009	5.19	2.17	2.75
HSS9 r2	16384	3037.22	6.18E-009	1.12E-008	11.15	2.15	2.69
HSS10 r2	32768	24160.20	4.55E-009	8.63E-009	24.97	2.24	2.64
HSS11 r2	65536	—	—	—	46.23	1.85	2.59
HSS12 r2	131072	—	—	—	94.67	2.05	2.55
HSS13 r2	262144	—	—	—	194.95	2.06	2.51
HSS14 r2	524288	—	—	—	317.33	1.63	2.48
HSS15 r2	1048576	—	—	—	622.37	1.96	2.46

Table 2. Comparison of errors and computation times for the HSS example series computing only 10 eigenvalues ($n/4+5,\dots,n/4+14$).

4. Possible Extensions

In the last two sections we have described an algorithm to compute the eigenvalues of $\mathcal{H}_\ell$-matrices. In this section we will discuss, what happens if we apply this algorithm to other, related hierarchically structured matrices. Further we will describe, how one can improve the LDL^T slicing algorithm for $\mathcal{H}_\ell$-matrices.

4.1. **LDL^T slicing algorithm for HSS-matrices.**

In [XCGL09], Xia and Chandrasekaran et al. present an algorithm to compute the Cholesky factorisation of a symmetric, positive definite hierarchical semi-separable matrix. Their ideas can be used to construct a similar LDL^T factorisation for HSS-matrices. Such an algorithm would use the structure of HSS-matrices much better, so that the complexity of the LDL^T factorisation would be reduced to $\mathcal{O}(nk^2)$.

In Algorithm 3 we take [XCGL09, Algorithm 2] and do some small changes, so that the algorithm now computes the LDL^T factorisation. The comments are the corresponding lines from

[1]More than 12 GB RAM were required for the matrix H15 r2 and swapping parts of the RAM to disk slows down the program significantly.

Name	n	t_{LAPACK} in s	abs. Error	rel. Error	t in s	t_i/t_{i-1}	N_i/N_{i-1}
H1 r1	64	0.00	1.33E-008	2.29E-008	0.07		
H2 r1	128	0.00	1.76E-008	3.18E-008	0.38	5.43	7.41
H3 r1	256	0.14	2.37E-008	4.60E-008	1.78	4.68	6.82
H4 r1	512	0.10	3.69E-008	8.83E-008	8.88	4.99	6.41
H5 r1	1024	0.73	5.22E-008	1.09E-007	42.68	4.81	6.10
H6 r1	2048	5.87	7.18E-008	2.73E-007	195.83	4.59	5.86
H7 r1	4096	45.65	9.86E-008	4.11E-007	889.63	4.54	5.67
H8 r1	8192	371.48	1.87E-007	6.13E-007	3436.77	3.86	5.51
H9 r1	16384	3180.03	2.48E-007	5.06E-007	14162.68	4.12	5.38
H10 r1	32768	23849.26	3.63E-007	4.20E-006	44102.81	3.11	5.27
H11 r1	65536	—	—	—	123026.60	2.79	5.18
H1 r2	64	0.01	1.18E-008	1.87E-008	0.07		
H2 r2	128	0.01	1.71E-008	3.24E-008	0.46	6.57	7.41
H3 r2	256	0.02	2.43E-008	5.00E-008	2.68	5.83	6.82
H4 r2	512	0.10	3.50E-008	7.58E-008	15.29	5.71	6.41
H5 r2	1024	0.81	5.35E-008	1.12E-007	83.06	5.43	6.10
H6 r2	2048	6.15	6.86E-008	2.29E-007	440.86	5.31	5.86
H7 r2	4096	55.22	1.16E-007	3.66E-007	2072.20	4.70	5.67
H8 r2	8192	432.61	1.82E-007	8.67E-007	8542.46	4.12	5.51
H9 r2	16384	3232.65	2.41E-007	2.28E-006	36054.62	4.22	5.38
H10 r2	32768	24043.06	3.77E-007	2.46E-003	119185.40	3.31	5.27
H11 r2	65536	—	—	—	370772.30	3.11	5.18
tri1	64	0.00	1.11E-008	1.40E-007	0.04		
tri2	128	0.00	1.16E-008	2.51E-007	0.11	2.75	7.41
tri3	256	0.01	1.21E-008	1.02E-006	0.18	1.64	6.82
tri4	512	0.05	4.03E-009	7.89E-008	0.36	2.00	6.41
tri5	1024	0.36	1.83E-008	1.92E-006	0.78	2.17	6.10
tri6	2048	2.94	2.29E-014	2.51E-013	3.18	4.08	5.86
tri7	4096	23.48	3.23E-014	3.57E-013	12.68	3.99	5.67
tri8	8192	187.89	4.58E-014	5.05E-013	53.34	4.21	5.51
tri9	16384	1572.58	6.41E-013	7.45E-013	232.55	4.36	5.38
tri10	32768	12903.42	9.76E-014	1.01E-012	1036.22	4.46	5.27
tri11	65536	—	—	—	4272.12	4.12	5.18
tri12	131072	—	—	—	17762.08	4.16	5.10
tri13	262144	—	—	—	73983.06	4.17	5.03

TABLE 3. Comparison of errors and computation times for the $\mathcal{H}_\ell$ and the tridiagonal example series computing all eigenvalues.

the originally algorithm. Only three small changes are needed. We are currently working on an implementation in our numerical code. Numerical results will be reported in the future.

4.2. **LDL^T slicing algorithm for $\mathcal{H}$-Matrices.** Does Algorithm 1 works for $\mathcal{H}$-matrices, too? The answer is yes, if we have $\mathcal{O}(n^2)$ storage and $\mathcal{O}(n^3)$ time. The answer is no, if linear-polylogarithmic complexity per eigenvalue has to be reached.

We have to use the $\mathcal{H}$-LDL^T factorisation for $\mathcal{H}$-matrices. Two additional problems appear: First, in general there is no exact $\mathcal{H}$-LDL^T factorisation like in the $\mathcal{H}_\ell$ case. So truncation is required to keep the block-wise ranks at a reasonable size. But then the results are much less accurate and so we encounter more wrong decisions, resulting in wrong $\nu(\mu)$ leading to intervals not containing the searched eigenvalue. Second, if we use fixed accuracy $\mathcal{H}$-arithmetic we get

Name	n	t_{LAPACK} in s	abs. Error	rel. Error	t in s	t_i/t_{i-1}	N_i/N_{i-1}
HSS1 r1	64	0.01	1.39E-008	2.31E-008	0.09		
HSS2 r1	128	0.00	1.74E-008	2.68E-008	0.38	4.22	7.41
HSS3 r1	256	0.02	2.51E-008	4.12E-008	1.63	4.29	6.82
HSS4 r1	512	0.10	3.74E-008	7.13E-008	7.63	4.68	6.41
HSS5 r1	1024	0.76	5.00E-008	1.03E-007	35.55	4.66	6.10
HSS6 r1	2048	5.82	9.24E-008	1.55E-007	164.99	4.64	5.86
HSS7 r1	4096	45.78	9.64E-008	1.74E-007	749.00	4.54	5.67
HSS8 r1	8192	368.82	1.66E-007	2.80E-007	3097.23	4.14	5.51
HSS9 r1	16384	3109.88	2.66E-007	4.75E-007	12930.91	4.17	5.38
HSS10 r1	32768	23641.63	3.85E-007	7.16E-007	52557.52	4.06	5.27
HSS11 r1	65536	—	—	—	209961.80	3.99	5.18
HSS1 r2	64	0.00	1.32E-008	2.09E-008	0.08		
HSS2 r2	128	0.00	1.89E-008	3.60E-008	0.42	5.25	7.41
HSS3 r2	256	0.02	2.66E-008	4.73E-008	2.11	5.02	6.82
HSS4 r2	512	0.10	3.51E-008	8.51E-008	9.73	4.61	6.41
HSS5 r2	1024	0.74	5.26E-008	1.01E-007	47.53	4.88	6.10
HSS6 r2	2048	5.85	9.57E-008	1.60E-007	210.06	4.42	5.86
HSS7 r2	4096	45.78	1.35E-007	2.19E-007	941.32	4.48	5.67
HSS8 r2	8192	354.75	1.44E-007	3.55E-007	3814.07	4.05	5.51
HSS9 r2	16384	3037.22	2.80E-007	5.20E-007	15706.36	4.12	5.38
HSS10 r2	32768	24160.20	3.87E-007	7.39E-007	61013.11	3.88	5.27
HSS11 r2	65536	—	—	—	223563.10	3.66	5.18

TABLE 4. Comparison of errors and computation times for the HSS example series computing all eigenvalues.

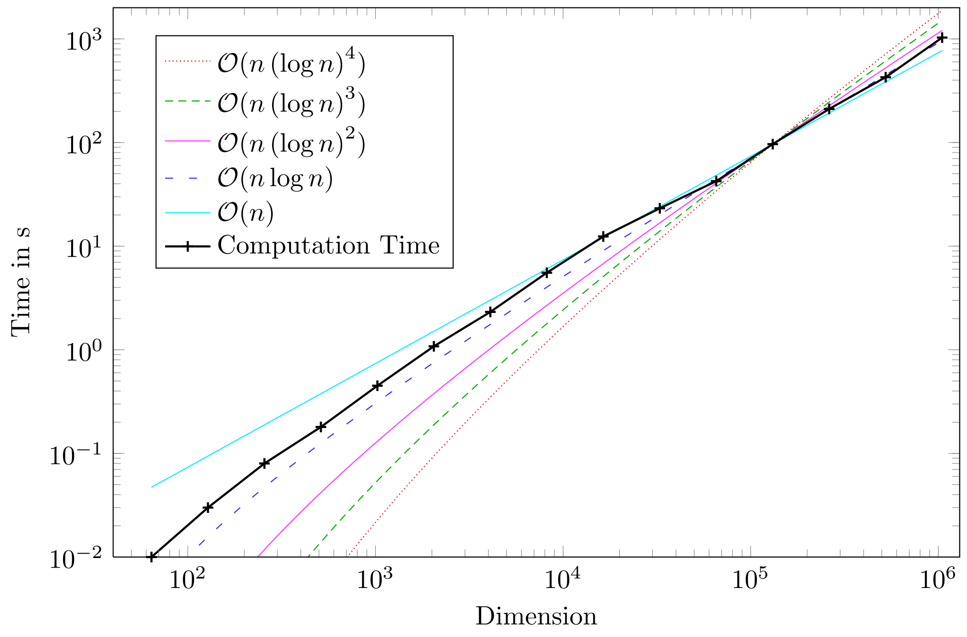

FIGURE 3. Computation times for **10** eigenvalues of Hℓ r1 matrices ($\ell = 1, \ldots, 15$).

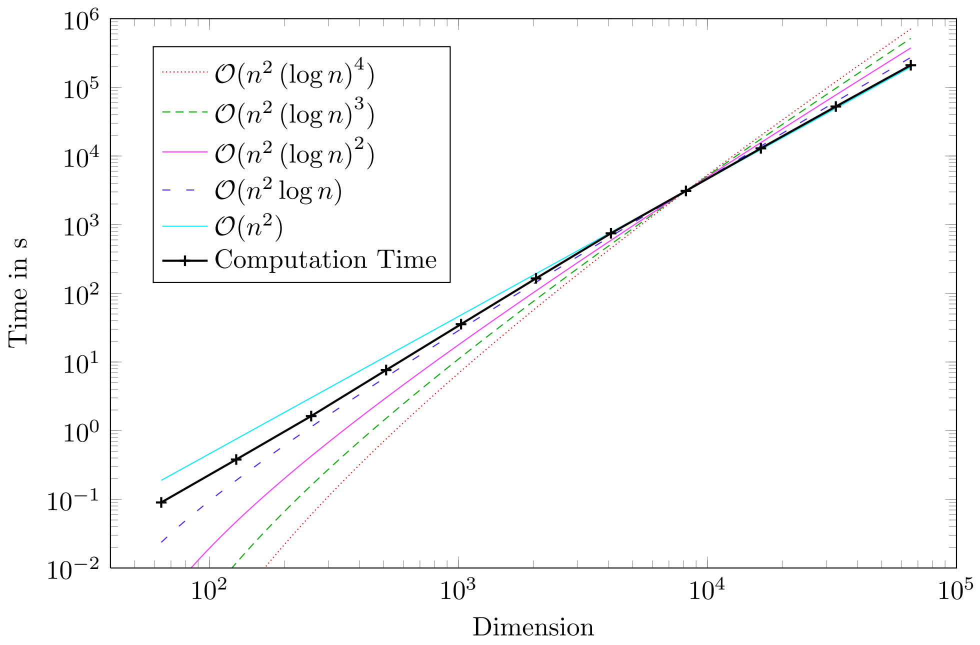

FIGURE 4. Computation times for **all** eigenvalues of HSSℓ r1 matrices ($\ell = 1, \ldots, 10$).

admissible blocks of large rank for some shifts. This will increase the computational as well as the storage complexity.

We have done some example computations to show the rank growth for different shifts. We use FEM discretisations of the 2D-Laplacian as example matrices, called FEMX, where X is the number of discretisation points in each direction. This matrices are generated using an example of the $\mathcal{H}$lib [HLi09]. Table 5 shows the maximal block-wise rank of the factors after the LDL^T factorisation of FEMX matrices for different shifts. If the shifted matrix is positive definite, the ranks will stay small. For shifts near, e.g., the eigenvalue 4 we get large ranks for large matrices. We observe, that the maximal block-wise rank is doubled from one column to the next. This contradicts the first statement from Lemma 2.4 since the rank grows faster than ℓk for large matrices. It follows that the complexity is not in

$$\mathcal{O}\left(nk^2 (\log n)^4\right),$$

and so for large matrices the computation time grows faster than the expected costs N_i

$$N_i = |EV|\, C_{sp} C_{id} n_i (\log n_i)^4,$$

like we see in Table 6. Still the computation time is much better than using LAPACK and we can solve eigenvalue problems not solvable otherwise.

4.3. **Parallelisation.** If we search more than one eigenvalue, we can use some of the computed $\nu(\mu)$ again. For example, since a_i and b_i are the same for all i, at the beginning the first μ_i will be the same, too. After the first LDL^T factorisation and the computation of $\nu(M - \mu I) = \nu$, we have two intervals. The interval $[a, \mu]$ contains ν eigenvalues and the interval $[\mu, b]$ contains $n - \nu$ eigenvalues. The computations on the two intervals are independent. We can use two cores, one for each interval and continue the computations independently. If we have more cores, we can increase the number of working cores on the next level to four and so on.

Since there is nearly no communication this will lead to a very good parallelisation. All we need is enough storage for the two $\mathcal{H}_\ell$-matrices, M and L, for each core. Since only linear-poly-logarithmic storage is required, that should be often the case.

Algorithm 3: Generalised LDL^T factorisation based on a generalised Cholesky factorisation for HSS-matrices, see Algorithm 2 [XCGL09]

```
   Input: HSS-matrix $H$ with $n$ nodes in the HSS tree, $H = H^T$
   Output: HSS-matrix containing $L$ and $D$, with $H = LDL^T$
 1 Allocate space for a stack.
 2 for each node $i = 1, \dots, n-1$ do
 3     if $i$ is a non-leaf node then
 4         Pop four matrices $\tilde{E}_{c_2}, \tilde{U}_{c_2}, \tilde{E}_{c_1}, \tilde{U}_{c_1}$ from the stack, where $c_1$, $c_2$ are the children of $i$.
 5         Obtain $E_i$ and $U_i$ by
               $$E_i = \begin{bmatrix} \tilde{E}_{c_1} & \tilde{U}_{c_1} B_{c_1} \tilde{U}_{c_2}^T \\ \tilde{U}_{c_2} B_{c_1} \tilde{U}_{c_1}^T & \tilde{E}_{c_2} \end{bmatrix}, U_i = \begin{bmatrix} \tilde{U}_{c_1} R_{c_1} \\ \tilde{U}_{c_2} R_{c_2} \end{bmatrix}.$$
 6     end
 7     Compress $U_i$ by the QL factorisation $\hat{U}_i = Q_i U_i = \begin{bmatrix} 0 \\ \tilde{U}_i \end{bmatrix}$ and push $\tilde{U}_i$ onto the stack
 8     Update $E_i$ with $\tilde{E}_i = Q_i^T E_i Q_i$. Factorise $\tilde{E}_i$ with
           $$\hat{E}_i = \begin{bmatrix} L_i & 0 \\ E_{i;2,1} L_i^{-T} D_i^{-1} & I \end{bmatrix} \begin{bmatrix} D_i & 0 \\ 0 & \tilde{E}_i \end{bmatrix} \begin{bmatrix} L_i^T & D_i^{-1} L_i^{-1} E_{i;1,2} \\ 0 & I \end{bmatrix},$$
       and obtain the Schur complement $\tilde{E}_i$ as
           $$\tilde{E}_i = E_{i;2,2} - E_{i;2,1} L_i^{-T} D_i^{-1} L_i^{-1} E_{i;1,2}.$$
       Push $\tilde{E}_i$ onto the stack.
       If $D_i$ is singular, stop the algorithm and return error message $0 \in \Lambda(H)$.
       /* Update $E_i$ with $\tilde{E}_i = Q_i^T E_i Q_i$. Factorise $\tilde{E}_i$ with
           $$\hat{E}_i = \begin{bmatrix} L_i & 0 \\ E_{i;2,1} L_i^{-T} & I \end{bmatrix} \begin{bmatrix} L_i^T & L_i^{-1} E_{i;1,2} \\ 0 & \tilde{E}_i \end{bmatrix},$$
          and obtain the Schur complement $\tilde{E}_i$ as
           $$\tilde{E}_i = E_{i;2,2} - E_{i;2,1} L_i^{-T} L_i^{-1} E_{i;1,2}.$$
          Push $\tilde{E}_i$ onto the stack.                                  */
 9     If $D_i = 0$, then stop and note that 0 is an eigenvalue of $H$.
10 end
11 For root $n$, compute the Cholesky factorisation $E_n = L_n D_n L_n^T$
   /* For root $n$, compute the Cholesky factorisation $E_n = L_n L_n^T$    */
```

Shift	FEM8	FEM16	FEM32	FEM64	FEM128	FEM256	FEM512
0	8	10	11	11	11	11	11
2	8	11	13	21	37	69	101
4.1	8	11	16	32	61	126	173
4.01	8	11	16	32	64	127	180
4.001	8	11	16	32	64	128	190
4.0001	8	11	16	32	64	128	183
ℓ	1	3	5	7	9	11	13
n	64	256	1024	4096	16384	65536	262144

TABLE 5. Maximal block-wise rank after LDL^T factorisation for different shifts and different FEM matrices ($\epsilon = 10^{-5}$, block-wise rank 8 before LDL^T factorisation).

Name	n	t_{LAPACK} in s	abs. Error	rel. Error	t in s	t_i/t_{i-1}	N_i/N_{i-1}
FEM8	64	<0.01	3.26E-005	9.54E-006	0.01		
FEM16	256	0.02	5.72E-005	1.98E-005	0.12	12.00	189.63
FEM32	1024	0.64	5.84E-005	2.19E-005	1.29	10.75	42.32
FEM64	4096	47.88	3.65E-005	1.41E-005	10.06	7.80	11.61
FEM128	16384	4404.31	5.21E-005	2.03E-005	69.83	6.94	7.41
FEM256	65536	—	—	—	810.62	*11.61*	6.82
FEM512	262144	—	—	—	5558.06	6.86	8.24

Table 6. Example of finding the 10 eigenvalues $n/4+5,\dots,n/4+14$ of FEMX, $\epsilon = 10^{-5}, \epsilon_{\text{ev}} = 10^{-4}$; *italic entries* are larger than expected.

Name	Dimension	$t_{1\text{ core}}$	$t_{2\text{ cores}}$	t_1/t_2	$t_{4\text{ cores}}$	t_1/t_4
H2 r1	128	0.38	0.19	2.00	0.12	3.17
H4 r1	512	8.88	4.49	1.98	2.40	3.70
H6 r1	2048	195.83	100.70	1.94	54.85	3.57
H8 r1	8192	3436.77	1832.07	1.88	986.94	3.48
H10 r1	32768	44102.81	23120.85	1.91	13143.76	3.36
tri2	128	0.11	0.05	2.20	0.05	2.20
tri4	512	0.36	0.20	1.80	0.12	3.00
tri6	2048	3.18	1.62	1.96	0.81	3.93
tri8	8192	53.34	29.09	1.83	16.51	3.23
tri10	32768	1036.22	560.68	1.85	304.16	3.41
tri12	131072	17762.08	9738.99	1.82	5378.70	3.30
HSS2 r2	128	0.42	0.20	2.10	0.13	3.23
HSS4 r2	512	9.73	4.93	1.97	2.65	3.67
HSS6 r2	2048	210.06	105.75	1.99	57.64	3.64
HSS8 r2	8192	3814.07	1947.86	1.96	1053.48	3.62
HSS10 r2	32768	61013.11	32318.87	1.89	17542.21	3.48

Table 7. Parallelisation speedup for different matrices.

We have used OpenMP [Ope10] to parallelise the program code used for the numerical examples. This leads to a simple parallelisation, that is probably improvable. Table 7 shows the timing results again on Intel®Core™i7 CPU 920, but now we use all four cores. The speedup for the use of four cores instead of one is about 3.3. This is a good value compared with the parallelisation of other algorithms, e.g. the parallelisation of the LAPACK function `dlahqr` (QR algorithm for unsymmetric eigenvalue problems) has a speedup of 2.5 [HV96].

4.4. **Eigenvectors.**

Often, the eigenvectors of some eigenvalues are of interested, too. The LDL^T slicing algorithm does not compute the eigenvectors. The last LDL^T factorisation can be used as a preconditioner for a shifted preconditioned inverse iteration [BMGS06]. In this case the shift is near the eigenvalue, maximal ϵ_{ev} away, so the convergence properties should be well. Besides the eigenvector this will give us an improved approximation to the eigenvalue.

If the eigenvectors are clustered, we can compute the corresponding invariant subspace by a subspace version of preconditioned inverse iteration.

5. Conclusions

We have discussed the application of the old and nearly forgotten slicing-the-spectrum algorithm for computing selected eigenvalues of symmetric matrices to the class of $\mathcal{H}_\ell$-matrices. The LDL^T slicing algorithm uses the special structure of symmetric $\mathcal{H}_\ell$-matrices which makes the repeated computation of LDL^T-factorizations (which is the obstacle to its use for general dense matrices) a computational feasible task. In particular, the LDL^T slicing algorithm enables us to compute an interior eigenvalue of a symmetric $\mathcal{H}_\ell$-matrix in linear-polylogarithmic complexity. Numerical results confirm this. For the computation of a single or a few interior eigenvalues the algorithm is superior to existing ones. It is less efficient for computing all eigenvalues of symmetric $\mathcal{H}_\ell$-matrices, but due to the efficient use of memory, it allows to solve much larger dense eigenproblems within the considered class of matrices than simply applying the methods available in LAPACK.

We may also use the LDL^T slicing algorithm for computing the eigenvalues of general, symmetric $\mathcal{H}$-matrices. But then the algorithm is no longer of linear-polylogarithmic complexity. Nevertheless, again the computation of a few interior eigenvalues is possible for problem sizes that by far exceed the capabilities of standard Numerical Linear Algebra algorithms for symmetric matrices.

For special classes within the set of $\mathcal{H}_\ell$-matrices, like hierarchical semi-separable (HSS) matrices, there might be even more efficient variants if the special structure is exploited in the LDL^T-factorization. For HSS matrices, this is work in progress at this writing.

Finally we have seen that multi-core architectures can be exploited easily and lead to fairly good speed-up and parallel efficiency.

Acknowledgments

We are grateful to the Faculty of Mathematics of Chemnitz University of Technology for providing access to its compute servers for performing the numerical experiments. We would also like to express our thanks to Jessica Gördes and Steffen Börm, both with the Christian-Albrechts-Universität zu Kiel, as this work was inspired by discussions with them about computing eigenvalues of $\mathcal{H}_\ell$-matrices.

References

[ABB$^+$99] E. Anderson, Z. Bai, C. Bischof, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen. *LAPACK Users' Guide*. SIAM, Philadelphia, PA, third edition, 1999.

[Beb08] M. Bebendorf. *Hierarchical Matrices: A Means to Efficiently Solve Elliptic Boundary Value Problems*, volume 63 of *Lecture Notes in Computational Science and Engineering (LNCSE)*. Springer Verlag, Berlin Heidelberg, 2008.

[BG10] S. Börm and J. Gördes. An exact solver for simple H-matrix systems. Preprint, Christian Albrechts Univsität zu Kiel, February 2010.

[BK77] J.R. Bunch and L. Kaufman. Some stable methods for calculating inertia and solving symmetric linear systems. *Math. Comp.*, 31(137):163–179, 1977.

[BMGS06] J. Berns-Müller, I.G. Graham, and A. Spence. Inexact inverse iteration for symmetric matrices. *Linear Algebra and its Applications*, 416(2–3):389–413, 2006.

[CG01] S. Chandrasekaran and M. Gu. A fast and stable solver for recursively semi-separable systems of linear equations. In *Structured Matrices in Mathematics, Computer Science, and Engineering: Proceedings of an AMS-IMS-SIAM Joint Summer Research Conference, University of Colorado, Boulder, June 27-July 1, 1999.* American Mathematical Society, 2001.

[CGL05] S. Chandrasekaran, M. Gu, and W. Lyons. A fast adaptive solver for hierarchically semiseparable representations. *Calcolo*, 42:171–185, 2005.

[CGP06] S. Chandrasekaran, M. Gu, and T. Pals. A fast ULV decomposition solver for hierarchically semiseparable representation. *SIAM J. Matrix Anal. Appl.*, 28(3):603–622, 2006.

[DC06] P. Dewilde and S. Chandrasekaran. A hierarchical semi-separable Moore-Penrose equation solver. In *Wavelets, Multiscale Systems and Hypercomplex Analysis*, volume 167 of *Oper. Theory Adv. Appl.*, pages 69–85. Springer, 2006.

[DFV09] S. Delvaux, K. Frederix, and M. Van Barel. Transforming a hierarchical into a unitary-weight representation. *Electr. Trans. Num. Anal.*, 33:163–188, 2009.

[GH03] L. Grasedyck and W. Hackbusch. Construction and arithmetics of $\mathcal{H}$-matrices. *Computing*, 70(4):295–334, 2003.

[Gör09] J. Gördes. Eigenwertproblem von hierarchischen Matrizen mit lokalem Rang 1. Diplomarbeit, Mathematisch-Naturwissenschaftlichen Fakultät der Christian-Albrechts-Universität zu Kiel, May 2009.

[Gra01] L. Grasedyck. *Theorie und Anwendungen Hierarchischer Matrizen.* Dissertation, Mathematisch-Naturwissenschaftliche Fakultät der Christian-Albrechts-Universität zu Kiel, July 2001.

[GV96] G.H. Golub and C.F. Van Loan. *Matrix Computations.* Johns Hopkins University Press, Baltimore, third edition, 1996.

[Hac99] W. Hackbusch. A Sparse Matrix Arithmetic Based on $\mathcal{H}$-Matrices. Part I: Introduction to $\mathcal{H}$-Matrices. *Computing*, 62(2):89–108, 1999.

[Hac09] W. Hackbusch. *Hierarchische Matrizen. Algorithmen und Analysis.* Springer-Verlag, Berlin, 2009.

[HLi09] $\mathcal{H}$lib 1.3. `http://www.hlib.org`, 1999–2009.

[HV96] G. Henry and R. Van de Geijn. Parallelizing the qr algorithm for the unsymmetric algebraic eigenvalue problem: myths and reality. *SIAM J. Sci. Comput.*, 17(4):870–883, 1996.

[Lin02] M. Lintner. *Lösung der 2D Wellengleichung mittels hierarchischer Matrizen.* Dissertation, Fakultät für Mathematik, TU München, `http://tumb1.biblio.tu-muenchen.de/publ/diss/ma/2002/lintner.pdf`, June 2002.

[Ope10] OpenMP. `http://openmp.org`, 1997–2010.

[Par80] B.N. Parlett. *The Symmetric Eigenvalue Problem.* Prentice-Hall, Englewood Cliffs, first edition, 1980.

[VVM05] R. Vandebril, M. Van Barel, and N. Mastronardi. An implicit QR algorithm for symmetric semiseparable matrices. *Numer. Lin. Alg. Appl.*, 12(7):625–658, 2005.

[VVM08] R. Vandebril, M. Van Barel, and N. Mastronardi. *Matrix Computations and Semiseparable Matrices. Vol. I+II.* Johns Hopkins University Press, Baltimore, MD, 2008.

[XCGL09] J. Xia, S. Chandrasekaran, M. Gu, and X.S. Li. Fast algorithms for hierachically semiseparable matrices. *Numer. Lin. Alg. Appl.*, 2009. availible online in advance of print, doi: 10.1002/nla.691.