SIAM J. SCI. COMPUT.
Vol. 34, No. 1, pp. A485–A496

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

PETER BENNER† AND THOMAS MACH‡

Abstract. We use a bisection method [B. Parlett, *The Symmetric Eigenvalue Problem*, Prentice-Hall, Englewood Cliffs, NJ, 1980, 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 factorization of $M - \mu I$. For dense matrices, the LDL^T factorization is too expensive to yield an efficient eigenvalue algorithm in general, but not so for $\mathcal{H}_\ell$-matrices. In the special structure of $\mathcal{H}_\ell$-matrices there is an LDL^T factorization 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, $\mathcal{O}(n^2(\log n)^4 \log(\|M\|_2/\epsilon_{\mathrm{ev}}))$ 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 jth smallest one. Numerical experiments demonstrate the efficiency of the algorithm, in particular for the case when some interior eigenvalues are required.

Key words. symmetric hierarchical matrices, eigenvalues, $\mathcal{H}_\ell$-matrices, slicing the spectrum

AMS subject classifications. 65F15, 65F50, 15A18

DOI. 10.1137/100815323

1. Introduction.

In *The Symmetric Eigenvalue Problem*, Beresford N. Parlett describes a bisection method to find the eigenvalues of a symmetric matrix $M \in \mathbb{R}^{n\times n}$ [19, 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 mth eigenvalue as the limit of the following process:

i. Start with an interval $[a, b]$ for which $\nu(a) < m \le \nu(b)$ holds.
ii. Determine $\nu_m := \nu(\frac{a+b}{2})$. If $\nu_m > m$, then continue with the interval $[a, \frac{a+b}{2}]$, else with $[\frac{a+b}{2}, b]$.
iii. Repeat the bisection (step ii) until the interval is small enough.

The function $\nu(\cdot)$ can be evaluated using the LDL^T factorization 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 [13]. They include, among others, tridiagonal and numerous finite-element matrices. We will see in the next section that the LDL^T factorization for $\mathcal{H}_\ell$-matrices (for all shifts) can be computed in linear-poly-

*Submitted to the journal's Methods and Algorithms for Scientific Computing section November 18, 2010; accepted for publication (in revised form) October 26, 2011; published electronically February 14, 2012.
http://www.siam.org/journals/sisc/34-1/81532.html
†Max Planck Institute for Dynamics of Complex Technical Systems, 39106 Magdeburg, Germany (benner@mpi-magdeburg.mpg.de) and Chemnitz University of Technology, Department of Mathematics, 09107 Chemnitz, Germany.
‡Max Planck Institute for Dynamics of Complex Technical Systems, 39106 Magdeburg, Germany (thomas.mach@googlemail.com).

logarithmic complexity. We will further see that

$$\mathcal{O}\left(n^2k^2\left(\log n\right)^4\log\left(\|M\|_2/\epsilon_{\mathrm{ev}}\right)\right)\ \text{flops} \tag{1.1}$$

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 [9], 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 [5], has a total complexity of $\mathcal{O}(n^2\left(\log n\right)^\beta)$. Further, in [7] a transformation of $\mathcal{H}_\ell$- and the related hierarchical semiseparable (HSS) matrices into semiseparable matrices is presented, symmetry is not needed. For semiseparable matrices there is a QR algorithm [20]. Both steps have quadratic or quadratic-polylogarithmic complexity.

The complexity of the LDL^Tslicing 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^Tslicing 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 [13]. 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 $\mathcal{H}_\ell$-matrices are investigated, among others, in [12].

The following definition of $\mathcal{H}_\ell$-matrices is given in [14, p. 43] and [9].

DEFINITION 1.1 ($\mathcal{H}_\ell$-matrix). *Let $I=\{1,\dots,n\}$ 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 blockwise rank k, short $M\in\mathcal{H}_\ell(k)$, if it fulfills 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 $M_{11},M_{22}\in\mathcal{H}_{\ell-1}(k)$, $M_{12}=A_1B_1^T$, and $M_{21}=B_2A_2^T$, where $A_i,B_i\in\mathbb{R}^{n_{\ell-1}\times k'}$, with $k'\le k$.

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 1.1.

We will need the concept of $\mathcal{H}$-matrices. We will give a short definition of $\mathcal{H}$-matrices here; for details see [14] or [10]. We define some necessary terms first. A hierarchical tree, short $\mathcal{H}$-tree, T_I of an index set I is a tree with special conditions:

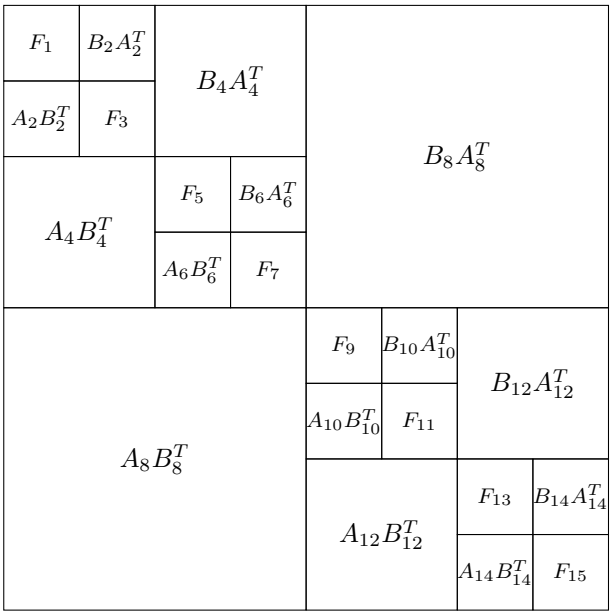

FIG. 1.1. *Structure of an $\mathcal{H}_3$-matrix.*

- 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 $\mathcal{O}(\log n)$ [10, 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 *hierarchical matrices* based on the $\mathcal{H}_\times$-tree $T_{I\times I}$ with maximum blockwise rank k and the minimum block-size $n_{\min}$ by

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

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 [14, 2, 11].

We divide the set of leaves into 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 $k = \operatorname{rank} AB^T$. 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 blockwise 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 $r_1 = \{i_1, \ldots, i_{m/2}\}$ and $r_2 = \{i_{m/2+1}, \ldots, i_m\}$ 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. ∎

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 condition 1 in Definition 1.1 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. In the next section we will detail how to compute all or some eigenvalues of symmetric $\mathcal{H}_\ell$-matrices.

2. Slicing the spectrum by LDL^T factorization. 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 $\hat{\lambda}_i = (b_i - a_i)/2$. So it holds that

$$\left|\lambda_i - \hat{\lambda}_i\right| < \frac{1}{2}\epsilon_{\text{ev}}. \tag{2.1}$$

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

ALGORITHM 1. *Slicing the spectrum* [19, 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: $\left\{\hat{\lambda}_1, \ldots, \hat{\lambda}_n\right\} \approx \Lambda(M)$;

```
for i = 1, ..., n do
    b_i := b;   a_i := a;
    while b_i − a_i ≥ ε_ev do
        μ := (b_i − a_i)/2;
        [L, D] := LDL^T factorization(M − μI);
        ν(M − μI) := |{j|D_jj < 0}|;
        if ν(M − μI) ≥ i then b_i := μ else a_i := μ ;
    end
    λ̂_i := (b_i − a_i)/2;
end
```

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*

$$P^T M P = N. \tag{2.2}$$

Further, we will use Sylvester's inertia law.

THEOREM 2.2 (Sylvester's inertia law, e.g., [19, 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 $n - \nu - \xi$ the number of positive eigenvalues. The triple $(\nu, \xi, n - \nu - \xi)$ is called M's inertia.*

If $M - \mu I$ has an LDL^T factorization $M - \mu I = LDL^T$ 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 $M - \mu I$, that is, $\nu(D) = \nu(\mu)$.

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 factorization may fail, which in this case is a welcome event as an eigenvalue has been found.

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

2.2. LDL^T factorization of $\mathcal{H}_\ell$-matrices.

DEFINITION 2.3 (LDL^T factorization [8]). *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 $D = \operatorname{diag}(d_1, \ldots, d_n)$ such that $M = LDL^T$. We will call this factorization* LDL^T factorization *or* LDL^T decomposition.

There is an algorithm to compute LDL^T factorizations for hierarchical matrices, first described in [17]. The $\mathcal{H}$-LDL^T factorization is block recursive; see Algorithm 2. For a hierarchical matrix $M \in \mathcal{H}(T, k)$ this factorization has a complexity of

$$\mathcal{O}\left(nk^2 (\log n)^2\right) \tag{2.3}$$

in fixed rank $\mathcal{H}$-arithmetic. We note that the LDL^T factorization 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 factorization is improved by, e.g., Bunch–Kaufmann pivoting [6]. Since pivoting would destroy the hierarchical structure, pivoting cannot 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 highly accurate LDL^T factorization.

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 in general increases 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 blockwise ranks is bounded. The next lemma provides this bound, which will be used for the complexity analysis of Algorithm 2.

ALGORITHM 2. *$\mathcal{H}$-LDL^T-factorization* $M = \mathrm{LDL}^T$.

```
1 H-LDL^T-factorization(M);
  Input: M ∈ H(T)
  Output: L ∈ H(T), D = diag(d_1, ..., d_n) with LDL^T = M and L lower
          triangular
2 if M = [M_11 M_12; M_21 M_22] ∉ L(T) then
3 |  [L_11, D_1] := H-LDL^T-factorization(M_11);
4 |  Compute the solution L_21 of L_21 D_1 L_11^T = M_21;
5 |  [L_22, D_2] := H-LDL^T-factorization(M_22 − L_21 D_1 L_21^T);
6 else
7 |  Compute the dense LDL^T-factorization 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 factorization is an $\mathcal{H}_\ell(k\ell)$-matrix. Further, the complexity of the computation of L and D by Algorithm* 2 *is*

$$\mathcal{O}(nk^2 (\log n)^4). \tag{2.4}$$

Proof. A matrix M belongs to the set $\mathcal{M}_{k,\tau}$ if

$$\operatorname{rank}\left(M|_{\tau' \times \tau}\right) = k$$

with $\tau' = I \setminus \tau$; see [12, Definition 4.1].

Analogous to [12, Lemma 4.2], we get for the LDL^T factorization of $M = LDL^T$ that $L \in \mathcal{M}_{k,\tau}$ if $M \in \mathcal{M}_{k,\tau}$. Together with Remark 4.4 and Lemma 4.5 of [12], we get $L \in \mathcal{H}(k\ell)$.

With Lemma 1.2, (2.3), and $\ell = \mathcal{O}(\log n)$ we conclude that the complexity of Algorithm 2 is in $\mathcal{O}(nk^2(\log n)^4)$. ∎

Remark 2.5. One can further show that the rank of a block in the lower triangular of L is equal to the number of blocks on the left-hand side of this block times k. Since this does not lead to an improved complexity estimate, we do not show the proof here.

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

Remark 2.6. In the case of stronger conditions on M, one can reduce the bound on the blockwise rank from $k\ell$ to k. For instance, if the matrix $M \in \mathcal{H}_\ell(k)$ fulfills the following conditions (here exemplary for $\mathcal{H}_3(k)$ with the same notation as in

Figure 1.1),

$$\operatorname{range}(A_4) \subset \operatorname{span}\left(\begin{bmatrix} F_5 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ A_6 \end{bmatrix}\right),$$

$$\operatorname{range}(A_8) \subset \operatorname{span}\left(\begin{bmatrix} F_9 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ A_{10} \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ \\ A_{12} \end{bmatrix}\right), \text{ and}$$

$$\operatorname{range}(A_{12}) \subset \operatorname{span}\left(\begin{bmatrix} F_{13} \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ A_{14} \end{bmatrix}\right),$$

or an analogue generalization, then $L \in \mathcal{H}_\ell(k)$. Tridiagonal matrices, generator representable semiseparable matrices, diagonal plus semiseparable matrices [21], and HSS matrices are of this structure. For all these special structures there exist good eigenvalue algorithms.

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 from below using the power iteration [11]. Multiplying the approximation by a small factor $1 + \delta$ will give an upper bound for $\|M\|_2$.

2.4. Complexity. For each eigenvalue λ_i we have to do several $\mathcal{H}$-LDL^T factorizations to reduce the length of the interval $[a_i, b_i]$. Each factorization halves the interval since we use a bisection method. So we need $\mathcal{O}(\log(\|M\|_2/\epsilon_{\text{ev}}))$ $\mathcal{H}$-LDL^T factorizations 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:

$$\mathcal{O}(n^2k^2(\log n)^4 \log(\|M\|_2/\epsilon_{\text{ev}})). \tag{2.5}$$

3. Numerical results. We have implemented Algorithm 1 with the LDL^T factorization for $\mathcal{H}$-matrices (see Algorithm 2) using the $\mathcal{H}$lib [16]. 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 blockwise 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 two Intel Xeon Westmere X5650 with 2.66 GHz and 48 GB DDR3 RAM, but we used only one core.

To test the algorithm we use a randomly generated series of $\mathcal{H}_\ell$-matrices of blockwise rank 1. The size of the matrices is varied from 64 to 1,048,576. 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` [1] 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 Table 3.1 and Figure 3.1. The time `dsyev` needs is given in the table, too.

Figure 3.1 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 3.1 shows the computation times and the errors for the example series if we compute all eigenvalues or 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 computation times grow more slowly than expected. Figure 3.2 compares the computation

TABLE 3.1
Comparison of errors and computation times for the $\mathcal{H}_\ell$ example series computing only 10 eigenvalues $(n/4+5,\ldots,n/4+14)$ and all eigenvalues.

Name	n	$t_{\text{LAP.}}$ in s (all ev.)	10 eigenvalues abs. err.	10 eigenvalues t in s	All eigenvalues abs. err.	All eigenvalues t in s
H1 r1	64	<0.01	5.36E-09	0.01	1.33E-08	0.06
H2 r1	128	<0.01	5.91E-09	0.03	1.76E-08	0.32
H3 r1	256	0.01	6.52E-09	0.07	2.37E-08	1.78
H4 r1	512	0.09	5.32E-09	0.19	3.69E-08	9.40
H5 r1	1 024	0.65	5.74E-09	0.50	5.22E-08	46.44
H6 r1	2 048	4.96	5.79E-09	1.20	7.18E-08	219.13
H7 r1	4 096	40.04	4.49E-09	2.58	9.86E-08	991.74
H8 r1	8 192	318.41	3.92E-09	6.39	1.87E-07	4 001.82
H9 r1	16 384	2 578.40	5.72E-09	13.76	2.48E-07	15 727.87
H10 r1	32 768	21 544.30	4.89E-09	26.06	3.63E-07	48 878.33
H11 r1	65 536	—	—	47.18	—	139 384.10
H12 r1	131 072	—	—	104.80	—	—
H13 r1	262 144	—	—	237.39	—	—
H14 r1	524 288	—	—	485.45	—	—
H15 r1	1 048 576	—	—	1 167.69	—	—
H9 r1	16 384	2 578.40	5.72E-09	13.76	2.48E-07	15 520.59
H9 r2	16 384	2 623.36	4.72E-09	36.26	2.41E-07	43 553.35
H9 r3	16 384	2 813.56	7.46E-09	68.73	2.05E-07	90 788.42
H9 r4	16 384	2 569.13	5.55E-09	108.63	2.62E-07	141 035.10
H9 r8	16 384	2 622.41	5.67E-09	345.52	2.60E-07	459 240.80
H9 r16	16 384	2 574.00	5.23E-09	998.93	2.70E-07	1 375 369.00

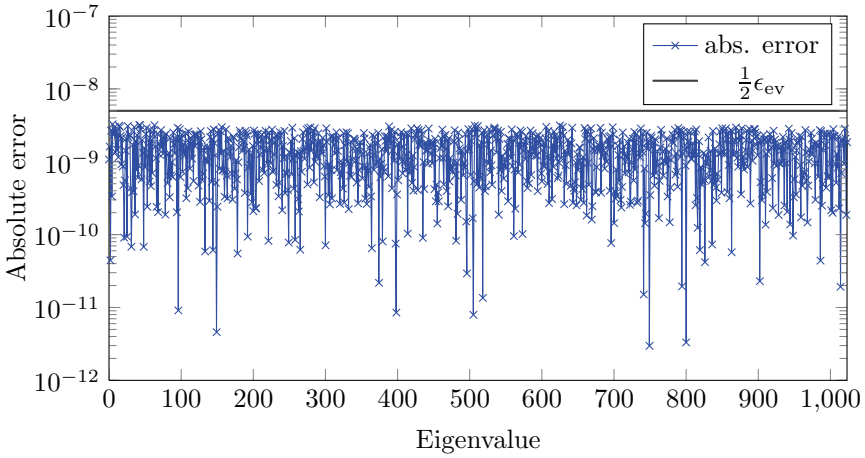

FIG. 3.1. *Absolute error* $\left|\lambda_i - \hat{\lambda}_i\right|$ *for a* 1024×1024 *matrix* ($H5r1$), $\epsilon_{ev} = 10^{-8}$.

times with $\mathcal{O}(n\,(\log n)^\beta)$, $\beta = 0, 1, 2, 3, 4$, too. There we see that the β in the example series is rather 2 than 4 like in (2.4). This confirms the estimated computational complexity from (2.4) and shows that there is probably a tighter bound. The absolute errors in the tables are computed as the maximum values of

$$e_{\text{abs}} = \left\|\lambda_i - \hat{\lambda}_i\right\|_2$$

taken over all computed $\hat{\lambda}_i$.

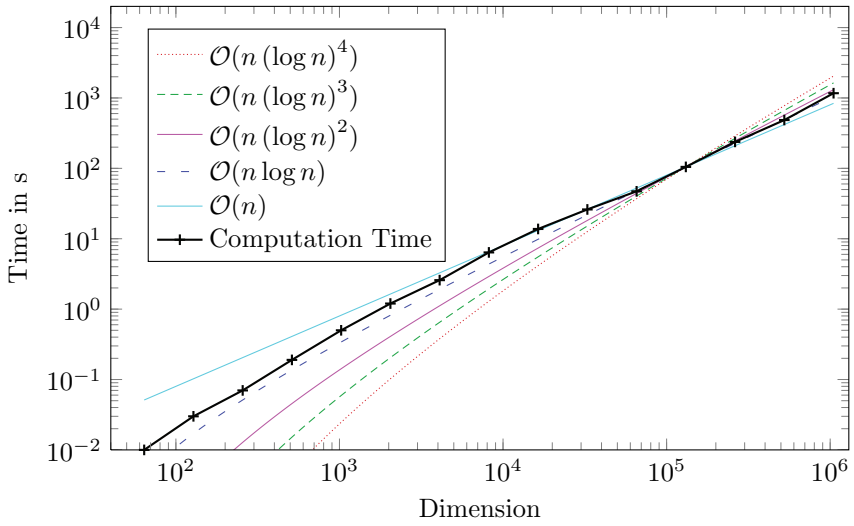

FIG. 3.2. *Computation times for* 10 *eigenvalues of H*ℓ *r*1 *matrices* ($\ell = 1, \ldots, 15$).

The table 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. For higher blockwise rank one gets similar results, which we present with further examples in the preprint [3] and in the second part of Table 3.1.

4. Possible extensions. In the last two sections we 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 hierarchical 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 $\mathcal{H}$-matrices. Does Algorithm 1 work for $\mathcal{H}$-matrices, too? Yes, the algorithm will converge for $\mathcal{H}$-matrices, too. But we are not able to prove linear-polylogarithmic complexity per eigenvalue. We have no accuracy estimation.

We have to use the $\mathcal{H}$-LDL^T factorization for $\mathcal{H}$-matrices. In general there is no exact $\mathcal{H}$-LDL^T factorization like in the $\mathcal{H}_\ell$ case. So truncation is required to keep the blockwise ranks at a reasonable size. But still we get admissible blocks of large rank for some shifts if we use fixed accuracy $\mathcal{H}$-arithmetic. This will increase the computational as well as the storage complexity.

We have done some example computations to illustrate the rank growth for different shifts. We use FEM discretizations of the two-dimensional Laplacian as example matrices, called FEMX, where X is the number of discretization points in each direction. These matrices are generated using an example of the $\mathcal{H}$lib [16]. Table 4.1 shows the maximal blockwise rank of the factors after the LDL^{T}factorization of FEMX matrices for different shifts. If the shifted matrix is positive definite, the ranks will stay small. For shifts near, e.g., eigenvalue 4, we get large ranks for large matrices. We observe that the maximal blockwise 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

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

TABLE 4.1

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

Shift	FEM8	FEM16	FEM32	FEM64	FEM128	FEM256	FEM512
0	8	10	11	11	11	11	11
4.1	8	11	16	32	61	126	173
4.001	8	11	16	32	64	128	190
ℓ	1	3	5	7	9	11	13
n	64	256	1024	4096	16384	65536	262144

TABLE 4.2

Example of finding the 10 eigenvalues $\hat{\lambda}_i$, $i = n/4+5, \ldots, n/4+14$, of FEMX, $\epsilon = 10^{-5}, \epsilon_{ev} = 10^{-4}$; italic entry is larger than expected.

Name	n	$t_{\text{LAP.}}$ in s	abs. err.	rel. err.	t in s	$\frac{t_i}{t_{i-1}}$	$\frac{N_i}{N_{i-1}}$
FEM8	64	<0.01	3.26E-005	9.54E-006	<0.01		
FEM16	256	0.01	5.72E-005	1.98E-005	0.13	—	189.63
FEM32	1 024	0.61	5.84E-005	2.19E-005	1.68	12.92	42.32
FEM64	4 096	39.64	3.65E-005	1.41E-005	12.52	7.45	11.61
FEM128	16 384	2 566.10	5.21E-005	2.03E-005	77.88	6.22	7.41
FEM256	65 536	—	—	—	774.52	*9.95*	6.82
FEM512	262 144	—	—	—	4 473.07	5.78	8.24

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

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

like we see in Table 4.2. Still, the computation time is much better than using LAPACK and we can solve eigenvalue problems not solvable otherwise.

But there is a second problem: The inexact computation of the LDL^T factorization leads to a perturbed matrix $\tilde{D} = D + E$. If the perturbation E is large enough, then the sign of one or more diagonal entries in D may change and so we get a wrong value $\tilde{\nu}(M - \mu I)$. This wrong $\tilde{\nu}(\mu)$ can cause wrong decisions, so we continue the search for eigenvalue i in an interval not containing λ_i.

To overcome this problem one can use an estimate ρ on $\|E\|_2$ to give lower and upper bounds on ν,

$$\left|\left\{i \middle| \tilde{D}_{ii} < -\rho\right\}\right| \le \nu \le \left|\left\{i \middle| \tilde{D}_{ii} < \rho\right\}\right|.$$

If i resides outside this interval, then the decision will be correct, independent of the exact ν. If i is contained in the interval, then we have to recompute ν with higher accuracy.

The problem here is the estimation of $\|E\|_2$, since the error depends on the condition number of $M - \mu I$, on the $\mathcal{H}$-matrix accuracy ϵ, and on the growth factor of the shifted matrix. A detailed investigation of this problem is for future research and exceeds the scope of this paper. At least in the example in Table 4.2 we see that the error in the computed spectrum is in $\mathcal{O}(\epsilon)$.

4.2. Parallelization. If we search more than one eigenvalue, then we can parallelize the algorithm. After the first LDL^T factorization 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. This is an advantage of the simple structure of the bisection method

TABLE 4.3
Parallelization speedup.

Name	n	$t_{1\text{ core}}$	$t_{2\text{ core}}$	t_{1c}/t_{2c}	$t_{4\text{ core}}$	t_{1c}/t_{4c}	$t_{8\text{ core}}$	t_{1c}/t_{8c}
H2 r1	128	0.33	0.18	1.83	0.10	3.30	0.06	5.50
H4 r1	512	9.44	4.87	1.94	2.57	3.67	1.43	6.60
H6 r1	2 048	219.28	114.69	1.91	60.27	3.64	33.88	6.47
H8 r1	8 192	4 022.8	2 151.6	1.87	1 170.6	3.44	676.57	5.95
H10 r1	32 768	49 012	25 376	1.93	15 402	3.18	10 007	4.90

compared with other eigenvalue algorithms like the QR algorithm. If we search eigenvalues in both intervals, then 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.

We used OpenMP [18] to parallelize the program code used for the numerical examples. This leads to a simple parallelization which is probably improvable. Table 4.3 shows the timing results on an Intel Xeon Westmere X5650, but now we use up to eight cores. The speedup for the use of four cores instead of one is about 3.3. This is a good value compared with the parallelization of other algorithms; e.g., the parallelization of the LAPACK function `dlahqr` (QR algorithm for unsymmetric eigenvalue problems) has a speedup of 2.5 [15]. The only drawback of this simple parallelization is that each core requires the same storage as the single-core variant of the program.

4.3. Eigenvectors. Often, the eigenvectors of some eigenvalues are of interest, too. The LDL^Tslicing algorithm does not compute the eigenvectors. In [4] the application of preconditioned inverse iteration to hierarchical matrices is investigated. A method for the computation of inner eigenvalues also is given there. Since we have a good approximation to the eigenvalue, we expect fast convergence.

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

Besides the eigenvector, this will provide an improved approximation to the eigenvalue. This may be used to detect and remedy the wrong decisions in the case of approximative arithmetic.

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^Tslicing 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 computationally feasible task. In particular, the LDL^Tslicing 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 us 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^Tslicing 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 HSS matrices [22], there might be even more efficient variants if the special structure is exploited in the LDL^T-factorization. Such a variant is described in [3].

Finally, we have seen that multicore architectures can be exploited easily and lead to fairly good speedup and parallel efficiency.

Acknowledgments. We would like to express our thanks to Jessica Gördes and Steffen Börm, both with Christian-Albrechts-Universität zu Kiel, as this work was inspired by discussions with them about computing eigenvalues of $\mathcal{H}_\ell$-matrices.

REFERENCES

[1] 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*, 3rd ed., SIAM, Philadelphia, 1999.

[2] M. Bebendorf, *Hierarchical Matrices: A Means to Efficiently Solve Elliptic Boundary Value Problems*, Lecture Notes in Comput. Sci. and Engrg. 63, Springer-Verlag, Berlin, 2008.

[3] P. Benner and T. Mach, *Computing All or Some Eigenvalues of Symmetric $\mathcal{H}_\ell$-Matrices*, Max Planck Institute Magdeburg Preprint MPIMD/10-01, http://www.mpi-magdeburg.mpg.de/preprints/ (2010).

[4] P. Benner and T. Mach, *The Preconditioned Inverse Iteration for Hierarchical Matrices*, Numer. Linear Algebra, Appl., to appear; also available as Max Planck Institute Preprint MPIMD/11-01 from http://www.mpi-magdeburg.mpg.de/preprints/ (2012).

[5] S. Börm and J. Gördes, *An Exact Solver for Simple H-matrix Systems*, preprint, Christian-Albrechts-Universität zu Kiel, 2010.

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

[7] S. Delvaux, K. Frederix, and M. Van Barel, *Transforming a hierarchical into a unitary-weight representation*, Electr. Trans. Numer. Anal., 33 (2009), pp. 163–188.

[8] G. Golub and C. Van Loan, *Matrix Computations*, 3rd ed., Johns Hopkins University Press, Baltimore, MD, 1996.

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

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

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

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

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

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

[15] 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 (1996), pp. 870–883.

[16] *$\mathcal{H}$lib* 1.3, http://www.hlib.org (1999–2009).

[17] M. Lintner, *The eigenvalue problem for the 2D Laplacian in $\mathcal{H}$-matrix arithmetic and application to the heat and wave equation*, Computing, 72 (2004), pp. 293–323.

[18] *OpenMP*, http://openmp.org (1997–2010).

[19] B. Parlett, *The Symmetric Eigenvalue Problem*, Prentice-Hall, Englewood Cliffs, NJ, 1980.

[20] R. Vandebril, M. Van Barel, and N. Mastronardi, *An implicit QR algorithm for symmetric semiseparable matrices*, Numer. Linear Algebra Appl., 12 (2005), pp. 625–658.

[21] R. Vandebril, M. Van Barel, and N. Mastronardi, *Matrix Computations and Semiseparable Matrices*, vol. 1, 2, Johns Hopkins University Press, Baltimore, MD, 2008.

[22] J. Xia, S. Chandrasekaran, M. Gu, and X. Li, *Fast algorithms for hierachically semiseparable matrices*, Numer. Linear Algebra Appl., 17 (2010), pp. 953–976.

SIAM J. SCI. COMPUT.
Vol. 34, No. 1, pp. A485–A496

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

PETER BENNER† AND THOMAS MACH‡

Abstract. We use a bisection method [B. Parlett, *The Symmetric Eigenvalue Problem*, Prentice-Hall, Englewood Cliffs, NJ, 1980, 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 factorization of $M - \mu I$. For dense matrices, the LDL^T factorization is too expensive to yield an efficient eigenvalue algorithm in general, but not so for $\mathcal{H}_\ell$-matrices. In the special structure of $\mathcal{H}_\ell$-matrices there is an LDL^T factorization 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, $\mathcal{O}(n^2(\log n)^4 \log(\|M\|_2/\epsilon_{\mathrm{ev}}))$ 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 jth smallest one. Numerical experiments demonstrate the efficiency of the algorithm, in particular for the case when some interior eigenvalues are required.

Key words. symmetric hierarchical matrices, eigenvalues, $\mathcal{H}_\ell$-matrices, slicing the spectrum

AMS subject classifications. 65F15, 65F50, 15A18

DOI. 10.1137/100815323

1. Introduction.

In *The Symmetric Eigenvalue Problem*, Beresford N. Parlett describes a bisection method to find the eigenvalues of a symmetric matrix $M \in \mathbb{R}^{n\times n}$ [19, 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 mth eigenvalue as the limit of the following process:

i. Start with an interval $[a, b]$ for which $\nu(a) < m \le \nu(b)$ holds.
ii. Determine $\nu_m := \nu(\frac{a+b}{2})$. If $\nu_m > m$, then continue with the interval $[a, \frac{a+b}{2}]$, else with $[\frac{a+b}{2}, b]$.
iii. Repeat the bisection (step ii) until the interval is small enough.

The function $\nu(\cdot)$ can be evaluated using the LDL^T factorization 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 [13]. They include, among others, tridiagonal and numerous finite-element matrices. We will see in the next section that the LDL^T factorization for $\mathcal{H}_\ell$-matrices (for all shifts) can be computed in linear-poly-

*Submitted to the journal's Methods and Algorithms for Scientific Computing section November 18, 2010; accepted for publication (in revised form) October 26, 2011; published electronically February 14, 2012.
http://www.siam.org/journals/sisc/34-1/81532.html
†Max Planck Institute for Dynamics of Complex Technical Systems, 39106 Magdeburg, Germany (benner@mpi-magdeburg.mpg.de) and Chemnitz University of Technology, Department of Mathematics, 09107 Chemnitz, Germany.
‡Max Planck Institute for Dynamics of Complex Technical Systems, 39106 Magdeburg, Germany (thomas.mach@googlemail.com).

logarithmic complexity. We will further see that

$$\mathcal{O}\left(n^2k^2\left(\log n\right)^4\log\left(\|M\|_2/\epsilon_{\mathrm{ev}}\right)\right)\ \text{flops} \tag{1.1}$$

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 [9], 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 [5], has a total complexity of $\mathcal{O}(n^2\left(\log n\right)^\beta)$. Further, in [7] a transformation of $\mathcal{H}_\ell$- and the related hierarchical semiseparable (HSS) matrices into semiseparable matrices is presented, symmetry is not needed. For semiseparable matrices there is a QR algorithm [20]. Both steps have quadratic or quadratic-polylogarithmic complexity.

The complexity of the LDL^Tslicing 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^Tslicing 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 [13]. 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 $\mathcal{H}_\ell$-matrices are investigated, among others, in [12].

The following definition of $\mathcal{H}_\ell$-matrices is given in [14, p. 43] and [9].

DEFINITION 1.1 ($\mathcal{H}_\ell$-matrix). *Let $I=\{1,\dots,n\}$ 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 blockwise rank k, short $M\in\mathcal{H}_\ell(k)$, if it fulfills 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 $M_{11},M_{22}\in\mathcal{H}_{\ell-1}(k)$, $M_{12}=A_1B_1^T$, and $M_{21}=B_2A_2^T$, where $A_i,B_i\in\mathbb{R}^{n_{\ell-1}\times k'}$, with $k'\le k$.

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 1.1.

We will need the concept of $\mathcal{H}$-matrices. We will give a short definition of $\mathcal{H}$-matrices here; for details see [14] or [10]. We define some necessary terms first. A hierarchical tree, short $\mathcal{H}$-tree, T_I of an index set I is a tree with special conditions:

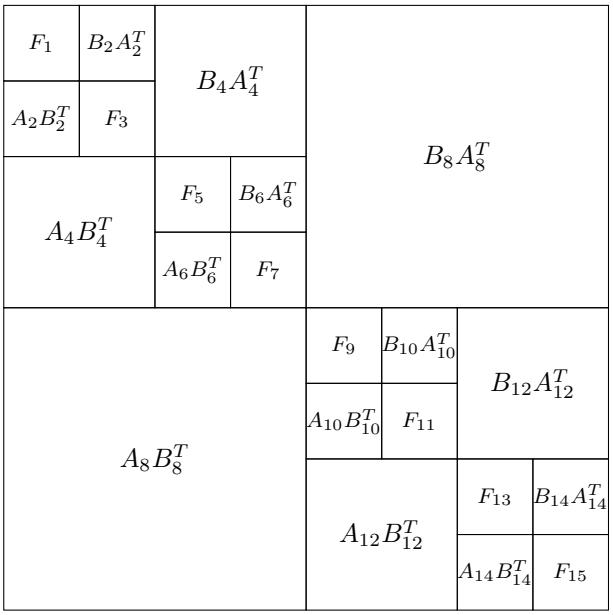

FIG. 1.1. *Structure of an $\mathcal{H}_3$-matrix.*

- 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 $\mathcal{O}(\log n)$ [10, 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 *hierarchical matrices* based on the $\mathcal{H}_\times$-tree $T_{I\times I}$ with maximum blockwise rank k and the minimum block-size $n_{\min}$ by

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

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 [14, 2, 11].

We divide the set of leaves into 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 $k = \operatorname{rank} AB^T$. 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 blockwise 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 $r_1 = \{i_1, \ldots, i_{m/2}\}$ and $r_2 = \{i_{m/2+1}, \ldots, i_m\}$ 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. ∎

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 condition 1 in Definition 1.1 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. In the next section we will detail how to compute all or some eigenvalues of symmetric $\mathcal{H}_\ell$-matrices.

2. Slicing the spectrum by LDL^T factorization. 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 $\hat{\lambda}_i = (b_i - a_i)/2$. So it holds that

$$\left|\lambda_i - \hat{\lambda}_i\right| < \frac{1}{2}\epsilon_{\text{ev}}. \tag{2.1}$$

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

ALGORITHM 1. *Slicing the spectrum* [19, 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: $\left\{\hat{\lambda}_1, \ldots, \hat{\lambda}_n\right\} \approx \Lambda(M)$;

```
for i = 1, ..., n do
    b_i := b;   a_i := a;
    while b_i − a_i ≥ ε_ev do
        μ := (b_i − a_i)/2;
        [L, D] := LDL^T factorization(M − μI);
        ν(M − μI) := |{j|D_jj < 0}|;
        if ν(M − μI) ≥ i then b_i := μ else a_i := μ ;
    end
    λ̂_i := (b_i − a_i)/2;
end
```

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*

$$P^T M P = N. \tag{2.2}$$

Further, we will use Sylvester's inertia law.

THEOREM 2.2 (Sylvester's inertia law, e.g., [19, 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 $n - \nu - \xi$ the number of positive eigenvalues. The triple $(\nu, \xi, n - \nu - \xi)$ is called M's inertia.*

If $M - \mu I$ has an LDL^T factorization $M - \mu I = LDL^T$ 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 $M - \mu I$, that is, $\nu(D) = \nu(\mu)$.

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 factorization may fail, which in this case is a welcome event as an eigenvalue has been found.

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

2.2. LDL^T factorization of $\mathcal{H}_\ell$-matrices.

DEFINITION 2.3 (LDL^T factorization [8]). *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 $D = \operatorname{diag}(d_1, \ldots, d_n)$ such that $M = LDL^T$. We will call this factorization* LDL^T factorization *or* LDL^T decomposition.

There is an algorithm to compute LDL^T factorizations for hierarchical matrices, first described in [17]. The $\mathcal{H}$-LDL^T factorization is block recursive; see Algorithm 2. For a hierarchical matrix $M \in \mathcal{H}(T, k)$ this factorization has a complexity of

$$\mathcal{O}\left(nk^2 (\log n)^2\right) \tag{2.3}$$

in fixed rank $\mathcal{H}$-arithmetic. We note that the LDL^T factorization 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 factorization is improved by, e.g., Bunch–Kaufmann pivoting [6]. Since pivoting would destroy the hierarchical structure, pivoting cannot 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 highly accurate LDL^T factorization.

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 in general increases 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 blockwise ranks is bounded. The next lemma provides this bound, which will be used for the complexity analysis of Algorithm 2.

ALGORITHM 2. *$\mathcal{H}$-LDL^T-factorization* $M = \mathrm{LDL}^T$.

```
1 H-LDL^T-factorization(M);
  Input: M ∈ H(T)
  Output: L ∈ H(T), D = diag(d_1, ..., d_n) with LDL^T = M and L lower
          triangular
2 if M = [M_11 M_12; M_21 M_22] ∉ L(T) then
3 |  [L_11, D_1] := H-LDL^T-factorization(M_11);
4 |  Compute the solution L_21 of L_21 D_1 L_11^T = M_21;
5 |  [L_22, D_2] := H-LDL^T-factorization(M_22 − L_21 D_1 L_21^T);
6 else
7 |  Compute the dense LDL^T-factorization 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 factorization is an $\mathcal{H}_\ell(k\ell)$-matrix. Further, the complexity of the computation of L and D by Algorithm* 2 *is*

$$\mathcal{O}(nk^2 (\log n)^4). \tag{2.4}$$

Proof. A matrix M belongs to the set $\mathcal{M}_{k,\tau}$ if

$$\operatorname{rank}\left(M|_{\tau' \times \tau}\right) = k$$

with $\tau' = I \setminus \tau$; see [12, Definition 4.1].

Analogous to [12, Lemma 4.2], we get for the LDL^T factorization of $M = LDL^T$ that $L \in \mathcal{M}_{k,\tau}$ if $M \in \mathcal{M}_{k,\tau}$. Together with Remark 4.4 and Lemma 4.5 of [12], we get $L \in \mathcal{H}(k\ell)$.

With Lemma 1.2, (2.3), and $\ell = \mathcal{O}(\log n)$ we conclude that the complexity of Algorithm 2 is in $\mathcal{O}(nk^2(\log n)^4)$. ∎

Remark 2.5. One can further show that the rank of a block in the lower triangular of L is equal to the number of blocks on the left-hand side of this block times k. Since this does not lead to an improved complexity estimate, we do not show the proof here.

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

Remark 2.6. In the case of stronger conditions on M, one can reduce the bound on the blockwise rank from $k\ell$ to k. For instance, if the matrix $M \in \mathcal{H}_\ell(k)$ fulfills the following conditions (here exemplary for $\mathcal{H}_3(k)$ with the same notation as in

Figure 1.1),

$$\operatorname{range}(A_4) \subset \operatorname{span}\left(\begin{bmatrix} F_5 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ A_6 \end{bmatrix}\right),$$

$$\operatorname{range}(A_8) \subset \operatorname{span}\left(\begin{bmatrix} F_9 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ A_{10} \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ \\ A_{12} \end{bmatrix}\right), \text{ and}$$

$$\operatorname{range}(A_{12}) \subset \operatorname{span}\left(\begin{bmatrix} F_{13} \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ A_{14} \end{bmatrix}\right),$$

or an analogue generalization, then $L \in \mathcal{H}_\ell(k)$. Tridiagonal matrices, generator representable semiseparable matrices, diagonal plus semiseparable matrices [21], and HSS matrices are of this structure. For all these special structures there exist good eigenvalue algorithms.

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 from below using the power iteration [11]. Multiplying the approximation by a small factor $1 + \delta$ will give an upper bound for $\|M\|_2$.

2.4. Complexity. For each eigenvalue λ_i we have to do several $\mathcal{H}$-LDL^T factorizations to reduce the length of the interval $[a_i, b_i]$. Each factorization halves the interval since we use a bisection method. So we need $\mathcal{O}(\log(\|M\|_2/\epsilon_{\text{ev}}))$ $\mathcal{H}$-LDL^T factorizations 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:

$$\mathcal{O}(n^2k^2(\log n)^4 \log(\|M\|_2/\epsilon_{\text{ev}})). \tag{2.5}$$

3. Numerical results. We have implemented Algorithm 1 with the LDL^T factorization for $\mathcal{H}$-matrices (see Algorithm 2) using the $\mathcal{H}$lib [16]. 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 blockwise 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 two Intel Xeon Westmere X5650 with 2.66 GHz and 48 GB DDR3 RAM, but we used only one core.

To test the algorithm we use a randomly generated series of $\mathcal{H}_\ell$-matrices of blockwise rank 1. The size of the matrices is varied from 64 to 1,048,576. 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` [1] 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 Table 3.1 and Figure 3.1. The time `dsyev` needs is given in the table, too.

Figure 3.1 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 3.1 shows the computation times and the errors for the example series if we compute all eigenvalues or 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 computation times grow more slowly than expected. Figure 3.2 compares the computation

TABLE 3.1
Comparison of errors and computation times for the $\mathcal{H}_\ell$ example series computing only 10 eigenvalues $(n/4+5,\ldots,n/4+14)$ and all eigenvalues.

Name	n	$t_{\text{LAP.}}$ in s (all ev.)	10 eigenvalues abs. err.	10 eigenvalues t in s	All eigenvalues abs. err.	All eigenvalues t in s
H1 r1	64	<0.01	5.36E-09	0.01	1.33E-08	0.06
H2 r1	128	<0.01	5.91E-09	0.03	1.76E-08	0.32
H3 r1	256	0.01	6.52E-09	0.07	2.37E-08	1.78
H4 r1	512	0.09	5.32E-09	0.19	3.69E-08	9.40
H5 r1	1 024	0.65	5.74E-09	0.50	5.22E-08	46.44
H6 r1	2 048	4.96	5.79E-09	1.20	7.18E-08	219.13
H7 r1	4 096	40.04	4.49E-09	2.58	9.86E-08	991.74
H8 r1	8 192	318.41	3.92E-09	6.39	1.87E-07	4 001.82
H9 r1	16 384	2 578.40	5.72E-09	13.76	2.48E-07	15 727.87
H10 r1	32 768	21 544.30	4.89E-09	26.06	3.63E-07	48 878.33
H11 r1	65 536	—	—	47.18	—	139 384.10
H12 r1	131 072	—	—	104.80	—	—
H13 r1	262 144	—	—	237.39	—	—
H14 r1	524 288	—	—	485.45	—	—
H15 r1	1 048 576	—	—	1 167.69	—	—
H9 r1	16 384	2 578.40	5.72E-09	13.76	2.48E-07	15 520.59
H9 r2	16 384	2 623.36	4.72E-09	36.26	2.41E-07	43 553.35
H9 r3	16 384	2 813.56	7.46E-09	68.73	2.05E-07	90 788.42
H9 r4	16 384	2 569.13	5.55E-09	108.63	2.62E-07	141 035.10
H9 r8	16 384	2 622.41	5.67E-09	345.52	2.60E-07	459 240.80
H9 r16	16 384	2 574.00	5.23E-09	998.93	2.70E-07	1 375 369.00

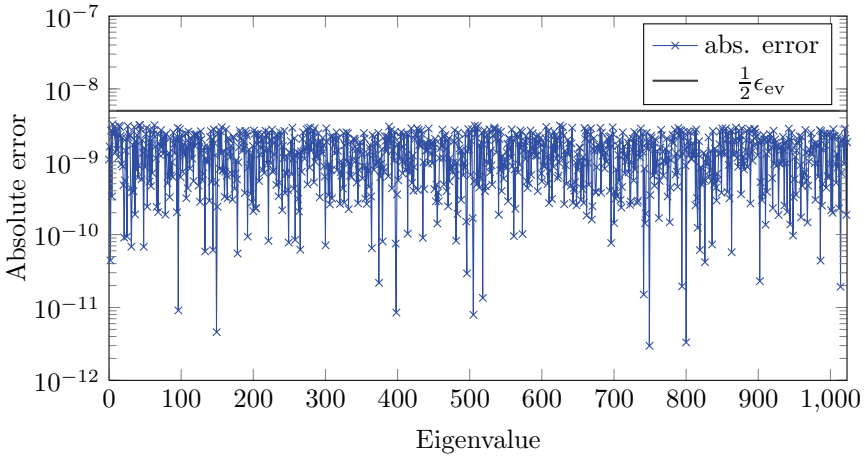

FIG. 3.1. *Absolute error* $\left|\lambda_i - \hat{\lambda}_i\right|$ *for a* 1024×1024 *matrix* ($H5r1$), $\epsilon_{ev} = 10^{-8}$.

times with $\mathcal{O}(n\,(\log n)^\beta)$, $\beta = 0, 1, 2, 3, 4$, too. There we see that the β in the example series is rather 2 than 4 like in (2.4). This confirms the estimated computational complexity from (2.4) and shows that there is probably a tighter bound. The absolute errors in the tables are computed as the maximum values of

$$e_{\text{abs}} = \left\|\lambda_i - \hat{\lambda}_i\right\|_2$$

taken over all computed $\hat{\lambda}_i$.

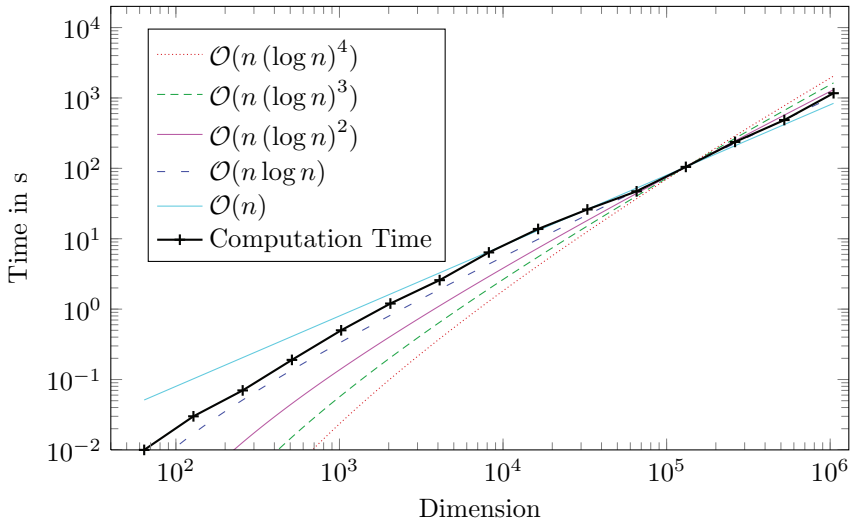

FIG. 3.2. *Computation times for* 10 *eigenvalues of H*ℓ *r*1 *matrices* ($\ell = 1, \ldots, 15$).

The table 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. For higher blockwise rank one gets similar results, which we present with further examples in the preprint [3] and in the second part of Table 3.1.

4. Possible extensions. In the last two sections we 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 hierarchical 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 $\mathcal{H}$-matrices. Does Algorithm 1 work for $\mathcal{H}$-matrices, too? Yes, the algorithm will converge for $\mathcal{H}$-matrices, too. But we are not able to prove linear-polylogarithmic complexity per eigenvalue. We have no accuracy estimation.

We have to use the $\mathcal{H}$-LDL^T factorization for $\mathcal{H}$-matrices. In general there is no exact $\mathcal{H}$-LDL^T factorization like in the $\mathcal{H}_\ell$ case. So truncation is required to keep the blockwise ranks at a reasonable size. But still we get admissible blocks of large rank for some shifts if we use fixed accuracy $\mathcal{H}$-arithmetic. This will increase the computational as well as the storage complexity.

We have done some example computations to illustrate the rank growth for different shifts. We use FEM discretizations of the two-dimensional Laplacian as example matrices, called FEMX, where X is the number of discretization points in each direction. These matrices are generated using an example of the $\mathcal{H}$lib [16]. Table 4.1 shows the maximal blockwise rank of the factors after the LDL^{T}factorization of FEMX matrices for different shifts. If the shifted matrix is positive definite, the ranks will stay small. For shifts near, e.g., eigenvalue 4, we get large ranks for large matrices. We observe that the maximal blockwise 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

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

TABLE 4.1

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

Shift	FEM8	FEM16	FEM32	FEM64	FEM128	FEM256	FEM512
0	8	10	11	11	11	11	11
4.1	8	11	16	32	61	126	173
4.001	8	11	16	32	64	128	190
ℓ	1	3	5	7	9	11	13
n	64	256	1024	4096	16384	65536	262144

TABLE 4.2

Example of finding the 10 eigenvalues $\hat{\lambda}_i$, $i = n/4+5, \ldots, n/4+14$, of FEMX, $\epsilon = 10^{-5}, \epsilon_{ev} = 10^{-4}$; italic entry is larger than expected.

Name	n	$t_{\text{LAP.}}$ in s	abs. err.	rel. err.	t in s	$\frac{t_i}{t_{i-1}}$	$\frac{N_i}{N_{i-1}}$
FEM8	64	<0.01	3.26E-005	9.54E-006	<0.01		
FEM16	256	0.01	5.72E-005	1.98E-005	0.13	—	189.63
FEM32	1 024	0.61	5.84E-005	2.19E-005	1.68	12.92	42.32
FEM64	4 096	39.64	3.65E-005	1.41E-005	12.52	7.45	11.61
FEM128	16 384	2 566.10	5.21E-005	2.03E-005	77.88	6.22	7.41
FEM256	65 536	—	—	—	774.52	*9.95*	6.82
FEM512	262 144	—	—	—	4 473.07	5.78	8.24

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

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

like we see in Table 4.2. Still, the computation time is much better than using LAPACK and we can solve eigenvalue problems not solvable otherwise.

But there is a second problem: The inexact computation of the LDL^T factorization leads to a perturbed matrix $\tilde{D} = D + E$. If the perturbation E is large enough, then the sign of one or more diagonal entries in D may change and so we get a wrong value $\tilde{\nu}(M - \mu I)$. This wrong $\tilde{\nu}(\mu)$ can cause wrong decisions, so we continue the search for eigenvalue i in an interval not containing λ_i.

To overcome this problem one can use an estimate ρ on $\|E\|_2$ to give lower and upper bounds on ν,

$$\left|\left\{i \middle| \tilde{D}_{ii} < -\rho\right\}\right| \le \nu \le \left|\left\{i \middle| \tilde{D}_{ii} < \rho\right\}\right|.$$

If i resides outside this interval, then the decision will be correct, independent of the exact ν. If i is contained in the interval, then we have to recompute ν with higher accuracy.

The problem here is the estimation of $\|E\|_2$, since the error depends on the condition number of $M - \mu I$, on the $\mathcal{H}$-matrix accuracy ϵ, and on the growth factor of the shifted matrix. A detailed investigation of this problem is for future research and exceeds the scope of this paper. At least in the example in Table 4.2 we see that the error in the computed spectrum is in $\mathcal{O}(\epsilon)$.

4.2. Parallelization. If we search more than one eigenvalue, then we can parallelize the algorithm. After the first LDL^T factorization 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. This is an advantage of the simple structure of the bisection method

TABLE 4.3
Parallelization speedup.

Name	n	$t_{1\text{ core}}$	$t_{2\text{ core}}$	t_{1c}/t_{2c}	$t_{4\text{ core}}$	t_{1c}/t_{4c}	$t_{8\text{ core}}$	t_{1c}/t_{8c}
H2 r1	128	0.33	0.18	1.83	0.10	3.30	0.06	5.50
H4 r1	512	9.44	4.87	1.94	2.57	3.67	1.43	6.60
H6 r1	2 048	219.28	114.69	1.91	60.27	3.64	33.88	6.47
H8 r1	8 192	4 022.8	2 151.6	1.87	1 170.6	3.44	676.57	5.95
H10 r1	32 768	49 012	25 376	1.93	15 402	3.18	10 007	4.90

compared with other eigenvalue algorithms like the QR algorithm. If we search eigenvalues in both intervals, then 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.

We used OpenMP [18] to parallelize the program code used for the numerical examples. This leads to a simple parallelization which is probably improvable. Table 4.3 shows the timing results on an Intel Xeon Westmere X5650, but now we use up to eight cores. The speedup for the use of four cores instead of one is about 3.3. This is a good value compared with the parallelization of other algorithms; e.g., the parallelization of the LAPACK function `dlahqr` (QR algorithm for unsymmetric eigenvalue problems) has a speedup of 2.5 [15]. The only drawback of this simple parallelization is that each core requires the same storage as the single-core variant of the program.

4.3. Eigenvectors. Often, the eigenvectors of some eigenvalues are of interest, too. The LDL^Tslicing algorithm does not compute the eigenvectors. In [4] the application of preconditioned inverse iteration to hierarchical matrices is investigated. A method for the computation of inner eigenvalues also is given there. Since we have a good approximation to the eigenvalue, we expect fast convergence.

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

Besides the eigenvector, this will provide an improved approximation to the eigenvalue. This may be used to detect and remedy the wrong decisions in the case of approximative arithmetic.

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^Tslicing 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 computationally feasible task. In particular, the LDL^Tslicing 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 us 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^Tslicing 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 HSS matrices [22], there might be even more efficient variants if the special structure is exploited in the LDL^T-factorization. Such a variant is described in [3].

Finally, we have seen that multicore architectures can be exploited easily and lead to fairly good speedup and parallel efficiency.

Acknowledgments. We would like to express our thanks to Jessica Gördes and Steffen Börm, both with Christian-Albrechts-Universität zu Kiel, as this work was inspired by discussions with them about computing eigenvalues of $\mathcal{H}_\ell$-matrices.

REFERENCES

[1] 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*, 3rd ed., SIAM, Philadelphia, 1999.

[2] M. Bebendorf, *Hierarchical Matrices: A Means to Efficiently Solve Elliptic Boundary Value Problems*, Lecture Notes in Comput. Sci. and Engrg. 63, Springer-Verlag, Berlin, 2008.

[3] P. Benner and T. Mach, *Computing All or Some Eigenvalues of Symmetric $\mathcal{H}_\ell$-Matrices*, Max Planck Institute Magdeburg Preprint MPIMD/10-01, http://www.mpi-magdeburg.mpg.de/preprints/ (2010).

[4] P. Benner and T. Mach, *The Preconditioned Inverse Iteration for Hierarchical Matrices*, Numer. Linear Algebra, Appl., to appear; also available as Max Planck Institute Preprint MPIMD/11-01 from http://www.mpi-magdeburg.mpg.de/preprints/ (2012).

[5] S. Börm and J. Gördes, *An Exact Solver for Simple H-matrix Systems*, preprint, Christian-Albrechts-Universität zu Kiel, 2010.

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

[7] S. Delvaux, K. Frederix, and M. Van Barel, *Transforming a hierarchical into a unitary-weight representation*, Electr. Trans. Numer. Anal., 33 (2009), pp. 163–188.

[8] G. Golub and C. Van Loan, *Matrix Computations*, 3rd ed., Johns Hopkins University Press, Baltimore, MD, 1996.

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

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

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

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

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

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

[15] 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 (1996), pp. 870–883.

[16] *$\mathcal{H}$lib* 1.3, http://www.hlib.org (1999–2009).

[17] M. Lintner, *The eigenvalue problem for the 2D Laplacian in $\mathcal{H}$-matrix arithmetic and application to the heat and wave equation*, Computing, 72 (2004), pp. 293–323.

[18] *OpenMP*, http://openmp.org (1997–2010).

[19] B. Parlett, *The Symmetric Eigenvalue Problem*, Prentice-Hall, Englewood Cliffs, NJ, 1980.

[20] R. Vandebril, M. Van Barel, and N. Mastronardi, *An implicit QR algorithm for symmetric semiseparable matrices*, Numer. Linear Algebra Appl., 12 (2005), pp. 625–658.

[21] R. Vandebril, M. Van Barel, and N. Mastronardi, *Matrix Computations and Semiseparable Matrices*, vol. 1, 2, Johns Hopkins University Press, Baltimore, MD, 2008.

[22] J. Xia, S. Chandrasekaran, M. Gu, and X. Li, *Fast algorithms for hierachically semiseparable matrices*, Numer. Linear Algebra Appl., 17 (2010), pp. 953–976.