A FAST RANDOMIZED EIGENSOLVER WITH STRUCTURED LDL FACTORIZATION UPDATE

YUANZHE XI*, JIANLIN XIA*, AND RAYMOND CHAN†

Abstract. In this paper, we propose a structured bisection method with adaptive randomized sampling for finding selected or all of the eigenvalues of certain real symmetric matrices A. For A with a low-rank property, we construct a hierarchically semiseparable (HSS) approximation and show how to quickly evaluate and update its inertia in the bisection method. Unlike some existing randomized HSS constructions, the methods here do not require the knowledge of the off-diagonal (numerical) ranks in advance. Moreover, for A with a weak rank property or slowly decaying off-diagonal singular values, we show an idea of aggressive low-rank inertia evaluation, which means that a compact HSS approximation can preserve the inertia for certain shifts. This is analytically justified for a special case, and numerically shown for more general ones. A generalized LDL factorization of the HSS approximation is then designed for the fast evaluation of the inertia. A significant advantage over standard LDL factorizations is that the HSS LDL factorization (and thus the inertia) of $A - sI$ can be quickly updated with multiple shifts s in bisection. The factorization with each new shift can reuse about 60% of the work. As an important application, the structured eigensolver can be applied to symmetric Toeplitz matrices, and the cost to find one eigenvalue is nearly linear in the order of the matrix. The numerical examples demonstrate the efficiency and the accuracy of our methods, especially the benefit of low-rank inertia evaluations. The ideas and methods can be potentially adapted to other HSS computations where shifts are involved and to more problems without a significant low-rank property.

Key words. eigensolver, aggressive low-rank inertia evaluation, HSS matrix, adaptive randomized sampling, matrix-free HSS construction, inertia/LDL update for varying shifts

AMS subject classifications. 65F05, 65F15, 65F30, 15A18

1. Introduction.

Large eigenvalue problems are frequently encountered in scientific computing and numerical simulations. In this paper, we consider eigenvalue problems of the form

$$Ax = \lambda x, \tag{1.1}$$

where A is an $n \times n$ real and symmetric matrix. A typical approach to solve for λ is to first reduce A to a tridiagonal form by orthogonal transformations and then apply QR iterations [15, 28], the divide-and-conquer algorithm [16], the bisection method [28], etc. The tridiagonal reduction step usually requires $O(n^3)$ floating point operations (flops) for a general symmetric A.

This work focuses on A with certain rank structures. That is, its off-diagonal blocks have decaying singular values. If the decay is fast, A has small off-diagonal ranks or numerical ranks and is often said to have a *low-rank property*. A can then be approximated by rank structures such as quasiseparable, hierarchically semiseparable (HSS), or hierarchical matrices [4, 6, 7, 13, 37]. Rank structured techniques, and even more generally, the fast multipole method [17] have been used in QR iterations [13], bisection [1, 2], and divide-and-conquer [8] to solve certain classes of eigenvalue problems, especially companion [3, 9, 31] and block diagonal plus semiseparable matrices [8]. Note that for these two special cases, the off-diagonal blocks have ranks at most 1.

*Department of Mathematics, Purdue University, West Lafayette, IN 47907, USA (yxi@math.purdue.edu, xiaj@math.purdue.edu). The research of Jianlin Xia was supported in part by an NSF CAREER Award DMS-1255416 and NSF grants DMS-1115572 and CHE-0957024.

†Department of Mathematics, Chinese University of Hong Kong, Hong Kong (rchan@math.cuhk.edu.hk).

1.1. Main results. We intend to handle symmetric A with the low-rank property (where the off-diagonal ranks or numerical ranks may be greater than 1), as well as A with a *weak rank property* (where the off-diagonal singular values decay slowly). We first approximate A with a compact HSS form using some recent randomized techniques, and then give a fast structured bisection method to compute some or all of the eigenvalues of A. In particular, we exploit the following three ideas.

1. *Adaptive and matrix-free* randomized HSS approximation methods. By combining the adaptive randomized compression techniques in [20] with the HSS construction methods in [23, 24, 39], we can quickly compute an HSS approximation to A to a given accuracy, without the requirement of knowing the actual off-diagonal numerical ranks or their overestimates as in [23, 24, 39]. The construction schemes make our eigensolver also applicable to problems where only matrix-vector products (instead of the explicit matrix A) are available. Various efficiency and flexibility improvements are made over existing randomized compression and HSS construction methods.

2. *Aggressive low-rank inertia evaluation* for matrices *with the weak low-rank property* or with high off-diagonal numerical ranks. Since only the inertia (numbers of positive, zero, and negative eigenvalues) is needed in bisection, we study the potential of using a compact or low-accuracy HSS approximation, even if the off-diagonal blocks may have high numerical ranks. The effectiveness can be proven for a special case, and is discussed and shown numerically for more general ones. In fact, our tests indicate that we can often use a much lower accuracy in the HSS construction for our structured eigenvalue solution of some matrices than for their linear system solutions in [10, 39]. Such an idea helps extend the applicability of rank structured techniques to potentially more general problems with or without the low-rank property.

3. *Fast inertia evaluation and update for varying shifts* in bisection via a generalized LDL factorization of the HSS approximation. In standard LDL factorizations of A, when the diagonal is changed due to a shift s in bisection, the factorization of $A - sI$ usually needs to be recomputed. Here, we first apply a fast generalized HSS LDL factorization (modified from [37]) to the HSS approximation $\tilde{A}$ of A in a precomputation. The generalized LDL factorization can be used to quickly evaluate the inertia, as justified in Theorem 4.2. Then we show that we can quickly update such a factorization (and thus the inertia) to get that of $\tilde{A} - sI$. In fact, about 60% of the work is performed on the off-diagonal blocks of $\tilde{A}$ and can be reused for multiple shifts s.

Assume r is the maximum off-diagonal rank after truncating the off-diagonal singular values of A. Then it usually costs about $O(r^2n)$ flops to find one eigenvalue of A, with the overall memory requirement of $O(rn)$. The low-rank property or the aggressive low-rank inertia evaluation idea suggests that often a small r can be used. An important application of our structured eigensolver is to find the eigenvalues of symmetric Toeplitz matrices, since they have the low-rank property in Fourier space [10, 25, 27, 39]. The eigensolver is tested on some important classes of Toeplitz matrices, as well as some more general problems. We show both the efficiency and the accuracy for finding selected or all of the eigenvalues. In particular, we demonstrate that, unlike the case of HSS direct solutions in [37, 39], lower accuracies or smaller r in HSS constructions are usually sufficient for the inertia evaluation. The methods here can be conveniently extended to complex Hermitian matrices.

1.2. Outline and notation. The remaining sections are organized as follows. In Section 2, adaptive randomized sampling is reviewed and our adaptive matrix-free HSS construction is introduced. We present the idea of aggressive low-rank inertia evaluation in Section 3. Section 4 is devoted to the fast inertia evaluation for an HSS matrix via the generalized HSS LDL factorization, as well as the fast LDL/inertia update for varying shifts in bisection. The numerical results are shown in Section 5, and some concluding remarks are given in Section 6. Throughout the paper, we use the following notation and terminology:

- $\mathcal{T}$ denotes a postordered full binary tree with its nodes labeled as $1, 2, \ldots, k$, where $k \equiv \mathrm{root}(\mathcal{T})$ is the root of $\mathcal{T}$, so that each node i is either a leaf or has two children c_1 and c_2 ordered as $c_1 < c_2 < i$;
- for each node $i \neq \mathrm{root}(\mathcal{T})$ of $\mathcal{T}$, $\mathrm{par}(i)$ and $\mathrm{sib}(i)$ denote its parent and sibling, respectively;
- for a matrix A and two index sets $\mathbf{I}$ and $\mathbf{J}$, $A|_{\mathbf{I}}$ denotes a submatrix of A formed by the rows corresponding to the row index set $\mathbf{I}$, and $A|_{\mathbf{I}\times\mathbf{J}}$ denotes a submatrix of A corresponding to the row index set $\mathbf{I}$ and the column index set $\mathbf{J}$;
- $\mathrm{randn}(n, k)$ represents an $n \times k$ matrix with independent and identically distributed (i.i.d.) standard Gaussian random entries;
- $\mathrm{diag}()$ represents a diagonal or block diagonal matrix formed by the subsequent numbers or matrices in the parentheses.

2. Randomized HSS construction: adaptive and matrix-free schemes.

The HSS structure provides an efficient way of handling matrices with small off-diagonal (numerical) ranks. (Later, we use ranks to also mean numerical ranks.) A formal definition of an HSS matrix can be found in [37, 38]. As a special case, a symmetric HSS matrix A with an associated HSS tree $\mathcal{T}$ can be defined as follows:

- $\mathcal{T}$ is a postordered full binary tree with nodes $i = 1, 2, \ldots, k$, and $k \equiv \mathrm{root}(\mathcal{T})$ is at level 0;
- there is a contiguous index set $\mathbf{I}_i$ associated with each node i such that $\mathbf{I}_k = \mathcal{N} \equiv \{1, \ldots, n\}$, and for any non-leaf node i with children c_1 and c_2,

$$\mathbf{I}_{c_1} \cup \mathbf{I}_{c_2} = \mathbf{I}_i, \quad \mathbf{I}_{c_1} \cap \mathbf{I}_{c_2} = \emptyset;$$

- there are matrices D_i, U_i, R_i, and B_i, called *HSS generators* associated with each node i, which satisfy the following recursive relation:

$$D_i = D_i^T \equiv A|_{\mathbf{I}_i \times \mathbf{I}_i} = \begin{pmatrix} D_{c_1} & U_{c_1} B_{c_1} U_{c_2}^T \\ U_{c_2} B_{c_1}^T U_{c_1}^T & D_{c_2} \end{pmatrix}, \quad U_i = \begin{pmatrix} U_{c_1} R_{c_1} \\ U_{c_2} R_{c_2} \end{pmatrix}.$$

Figure 2.1 demonstrates an example. We refer to $A_i^- \equiv A|_{\mathbf{I}_i \times (\mathcal{N}\setminus\mathbf{I}_i)}$ and $A_i^| \equiv A|_{(\mathcal{N}\setminus\mathbf{I}_i)\times\mathbf{I}_i}$ as the ith *HSS block row* and *column*, respectively, and the maximal rank of all the HSS blocks as the *HSS rank*. In fact, the columns of U_i form a column basis for A_i^-. Once an HSS approximation to A is constructed, the well-established fast HSS factorization and solution algorithms in [7, 34, 37] can be applied.

The standard HSS construction algorithms [35, 37] for an $n \times n$ dense matrix A require A to be explicitly available so as to compress its off-diagonal blocks directly. Those algorithms have a complexity of $O(rn^2)$, where r is the HSS rank of A. Recently, randomized algorithms are utilized to speed up and facilitate the compression process [23, 24]. Here, we discuss randomized and matrix-free HSS construction schemes.

2.1. Adaptive randomized sampling.

Suppose $\Phi \in \mathbb{R}^{M\times N}$ has a small numerical rank r. The randomized algorithm in [20, 24] finds an approximate column basis matrix Q for Φ in the following way:

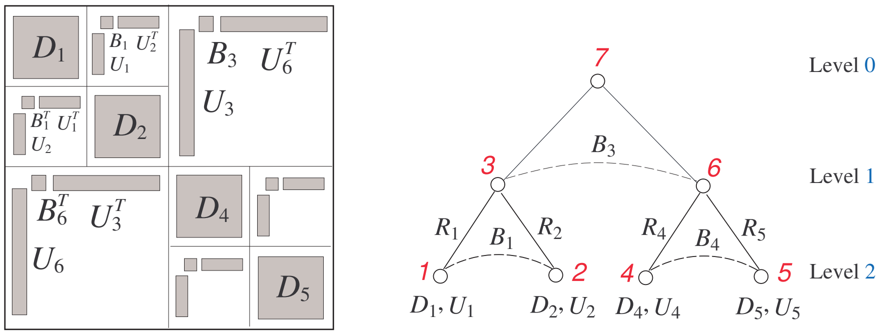


FIG. 2.1. *A symmetric HSS matrix and its HSS tree.*

1. pick a small integer α, called the oversampling size;
2. form $X = \text{randn}(N, \tilde{r})$, where $\tilde{r} = r + \alpha$ is called the sampling size;
3. compute the product $Y = \Phi X$;
4. compute an economy (or rank-revealing) QR factorization $Y = QS$, which, for convenience, is denoted by

$$Q = \mathtt{QR}(Y) \quad \text{or} \quad [Q,\ S] = \mathtt{QR}(Y). \tag{2.1}$$

Q is an approximate column basis matrix for Φ and it satisfies the following accuracy bound with the probability at least $1 - 6\alpha^{-\alpha}$ [20]:

$$||\Phi - Q(Q^T\Phi)||_2 \le (1 + 11\sqrt{r+\alpha}\sqrt{\min(M,N)})\sigma_{r+1},$$

where σ_{r+1} is the $(r+1)$st singular value of Φ.

On the other hand, if the numerical rank r is not known in advance, an adaptive procedure [20] as follows can be used to find r approximately. That is, the sampling size $\tilde{r}$ or the column size of X is increased until the following bound is satisfied for a given tolerance τ:

$$||\Phi - Q^{(r)}[(Q^{(r)})^T\Phi]||_2 \le \tau, \tag{2.2}$$

where $Q^{(r)}$ is an $M \times r$ matrix obtained as in (2.1). However, the direct computation of (2.2) for detecting the approximation error is quite expensive for each increment of $\tilde{r}$. A more efficient stopping criterion to determine r is given in [20, 33] as follows.

LEMMA 2.1. *Suppose $\Phi \in \mathbb{R}^{M\times N}$, d is an integer, and $x^{(i)} \in \mathbb{R}^N$, $i = 1, 2, \ldots, d$ are i.i.d. random vectors. Then the following bound holds for any real number η with the probability $1 - \eta^{-d}$:*

$$||\Phi||_2 \le \eta\sqrt{\frac{2}{\pi}} \max_{i=1,2,\ldots,d} ||\Phi x^{(i)}||_2.$$

According to Lemma 2.1, the approximation error $||\Phi - Q^{(\beta)}((Q^{(\beta)})^T\Phi)||_2$ for an $N \times \beta$ computed basis matrix $Q^{(\beta)}$ satisfies the following bound with high probability:

$$\begin{aligned} ||\Phi - Q^{(\beta)}[(Q^{(\beta)})^T\Phi]||_2 &= ||[I - Q^{(\beta)}(Q^{(\beta)})^T]\Phi||_2 \\ &\le \eta\sqrt{\frac{2}{\pi}} \max_{i=\beta-d+2,\ldots,\beta+1} ||[I - Q^{(\beta)}(Q^{(\beta)})^T]\Phi x^{(i)}||_2. \end{aligned} \tag{2.3}$$

That is, as long as

$$\max_{i=\beta-d+2,\ldots,\beta+1} ||(I - Q^{(\beta)}(Q^{(\beta)})^T)\Phi x^{(i)}||_2 \leq \frac{\tau}{\eta\sqrt{2/\pi}}, \tag{2.4}$$

the error bound in (2.2) will automatically hold with the probability at least $1-\eta^{-d}$.

If (2.4) does not hold, let $q = [I - Q^{(\beta)}(Q^{(\beta)})^T]\Phi x^{(\beta+1)}$, and set

$$Q^{(\beta+1)} = \begin{pmatrix} Q^{(\beta)} & \frac{q}{||q||_2} \end{pmatrix}.$$

Then repeat until (2.4) is satisfied. More details on this procedure can be found in [20]. For convenience, we also write

$$X^{(\beta+1)} = \begin{pmatrix} X^{(\beta)} & x^{(\beta+1)} \end{pmatrix} = \begin{pmatrix} x^{(1)} & \cdots & x^{(\beta)} & x^{(\beta+1)} \end{pmatrix}.$$

The above procedure uses matrix-vector multiplications in (2.3) instead of matrix-matrix multiplications in (2.2). A new basis vector q is conveniently computed based on the existing ones and $\Phi x^{(\beta+1)}$.

Remark 2.1. Under certain special circumstances, a matrix may have a small (numerical) rank, and only its largest eigenvalues are desired. Then the method in this section can be directly used to find the eigenvectors corresponding to those eigenvalues. More details will appear in future work.

2.2. Adaptive and matrix-free HSS constructions.

The adaptive randomized sampling method in Section 2.1 can be combined with some existing HSS construction schemes. For matrices which are explicitly available, we can directly use the methods in [24, 39]. These methods traverse the HSS tree $\mathcal{T}$ in a bottom-up order, and only require the matrix to be multiplied with $O(r)$ random vectors. We skip the details and denote such an HSS construction with adaptive randomized sampling by `AHSS`.

Our adaptive matrix-free HSS construction algorithm (denoted by `AMFHSS`) employs the HSS construction framework in [23] which needs only matrix-vector products instead of the matrix itself. However, we give various efficiency and flexibility improvements over the scheme in [23]:

- we use (rank-revealing) QR factorizations instead of SVD (e.g., (2.6) and (2.8) below);
- the R factors in the QR factorizations naturally help with the computation of certain generators (e.g., (2.7) and (2.9) below), thus avoiding various extra matrix multiplications in a peeling step in [23];
- fewer pseudo-inverses are needed (e.g., (2.7));
- moreover, we do not need to know the HSS rank or its overestimate in advance, and instead, we use adaptive randomized sampling so that the algorithm can dynamically detect the sampling size at each level of the HSS tree according to a pre-specified approximation tolerance τ.

These are explained as follows. The HSS construction for the symmetric matrix A traverses $\mathcal{T}$ from the top level (level 0) to the leaf level. Assume each node i is associated with an index set $\mathbf{I}_i$ as in Section 2, and define

$$\tilde{\mathbf{I}}^l = \bigcup\{\mathbf{I}_i | i\text{: all the left nodes (with } i < \text{sib}(i)\text{) at level } l\},$$
$$\hat{\mathbf{I}}^l = \bigcup\{\mathbf{I}_i | i\text{: all the right nodes (with } i > \text{sib}(i)\text{) at level } l\}.$$

The algorithm starts with $X^{(\beta)} \equiv \text{randn}(n, \beta)$, where β is a small integer which is an initial estimate of r. At level $l = 1$, choose two matrices $\tilde{X}_l^{(\beta)}$ and $\hat{X}_l^{(\beta)}$ which have the same sizes as $X^{(\beta)}$ and satisfy

$$\tilde{X}_l^{(\beta)}|_{\tilde{\mathbf{I}}^l} = X^{(\beta)}|_{\tilde{\mathbf{I}}^l}, \quad \tilde{X}_l^{(\beta)}|_{\hat{\mathbf{I}}^l} = 0, \quad \hat{X}_l^{(\beta)}|_{\hat{\mathbf{I}}^l} = X^{(\beta)}|_{\hat{\mathbf{I}}^l}, \quad \hat{X}_l^{(\beta)}|_{\tilde{\mathbf{I}}^l} = 0. \tag{2.5}$$

Compute

$$\tilde{Y}_l^{(\beta)} = A\tilde{X}_l^{(\beta)}, \quad \hat{Y}_l^{(\beta)} = A\hat{X}_l^{(\beta)}.$$

Then treat $\tilde{Y}_l^{(\beta)}$ and $\hat{Y}_l^{(\beta)}$ as Y in Section 2.1. Repeat the adaptive randomized sampling procedure until (2.4) is satisfied. For each left node i and $j = \text{sib}(i)$ at level 1, compute

$$[\hat{U}_i,\ \hat{S}_i] = \texttt{QR}(\hat{Y}_l^{(\beta)}|_{\mathbf{I}_i}), \quad [\hat{U}_j,\ \hat{S}_j] = \texttt{QR}(\tilde{Y}_l^{(\beta)}|_{\mathbf{I}_j}). \tag{2.6}$$

Set $U_i \equiv \hat{U}_i$ and $U_j \equiv \hat{U}_j$, which are the approximate column and row basis matrices for $A|_{\mathbf{I}_i \times \mathbf{I}_j}$, respectively. Also, let

$$\hat{B}_i = \hat{S}_i (U_j^T X^{(\beta)}|_{\mathbf{I}_j})^\dagger, \tag{2.7}$$

where we assume the pseudo-inverse $(U_j^T X^{(\beta)}|_{\mathbf{I}_j})^\dagger$ exists, and set $B_i \equiv \hat{B}_i$. (In [23], two pseudo-inverses and five matrix multiplications are needed.)

At level $l \geq 2$, choose $\tilde{X}_l^{(\beta)}$ and $\hat{X}_l^{(\beta)}$ as in (2.5). For a left node i at level l and $j = \text{sib}(i)$, we try to find $A|_{\mathbf{I}_i \times \mathbf{I}_j} \hat{X}_l^{(\beta)}|_{\mathbf{I}_j}$ (and $A|_{\mathbf{I}_j \times \mathbf{I}_i} \tilde{X}_l^{(\beta)}|_{\mathbf{I}_i}$). This can be done by subtracting the product of some upper level HSS blocks of A with appropriate pieces of $\hat{X}_l^{(\beta)}$ from $A|_{\mathbf{I}_i} \hat{X}_l^{(\beta)}$. Since the upper level HSS blocks of A are already in low-rank forms (part of the HSS approximation to A), this involves the multiplication of such low-rank blocks with a vector x. This is just a partial computation of the HSS multiplication [7], so we skip the details and denote it by

$$y = \texttt{HSSpmv}(A, x, l-1).$$

Compute

$$\hat{Y}_l^{(\beta)} = A\hat{X}_l^{(\beta)} - \texttt{HSSpmv}(A, \hat{X}_l^{(\beta)}, l-1), \quad \tilde{Y}_l^{(\beta)} = A\tilde{X}_l^{(\beta)} - \texttt{HSSpmv}(A, \tilde{X}_l^{(\beta)}, l-1).$$

For a left node i and $j = \text{sib}(i)$ at level l, treat $\hat{Y}_l^{(\beta)}|_{\mathbf{I}_i}$ as Y in the adaptive randomized sampling procedure until (2.4) is satisfied. Similarly, treat $\tilde{Y}_l^{(\beta)}|_{\mathbf{I}_j}$ as Y and repeat. Then compute approximate row and column basis matrices $\hat{U}_i$ and $\hat{U}_j$ for $A|_{\mathbf{I}_i \times \mathbf{I}_j}$, respectively, as in (2.6), and compute $\hat{B}_i$ as in (2.7).

At this point, let $p = \text{par}(i)$ and partition U_p as

$$U_p = \begin{pmatrix} U_p^1 \\ U_p^2 \end{pmatrix},$$

where U_p^1 and U_p^2 have the same row sizes as U_i and U_j, respectively. In [23], the matrices $\begin{pmatrix} \hat{U}_i & U_p^1 \end{pmatrix}$ and $\begin{pmatrix} \hat{U}_j & U_p^2 \end{pmatrix}$ are compressed with SVDs to obtain the left singular vectors as U_i and U_j, respectively, and are then multiplied by U_i and U_j on the left to get R_i and R_j, respectively. Here instead, we use QR factorizations

$$[U_i, \begin{pmatrix} S_i & R_i \end{pmatrix}] = \mathtt{QR}\begin{pmatrix} \hat{U}_i & U_p^1 \end{pmatrix}, \quad [U_j, \begin{pmatrix} S_j & R_j \end{pmatrix}] = \mathtt{QR}\begin{pmatrix} \hat{U}_j & U_p^2 \end{pmatrix}. \tag{2.8}$$

Then $U_p, \hat{U}_i, \hat{U}_j$ can be discarded. Also, set B_i as

$$B_i = S_i \hat{B}_i S_j^T. \tag{2.9}$$

Finally, for all the leaves i of $\mathcal{T}$, we also compute the following in order to obtain D_i:

$$\tilde{Y}_l = A \begin{pmatrix} I \\ \vdots \\ I \end{pmatrix} - \mathtt{HSSpmv}(A, \begin{pmatrix} I \\ \vdots \\ I \end{pmatrix}, l-1), \tag{2.10}$$

where each I is an identity matrix with size equal to the leaf level diagonal block size. (Here, the leaf level D_i blocks are all assumed to have the same size. Otherwise, we can place appropriate zero columns beside the identity blocks in (2.10).) Then set

$$D_i = \tilde{Y}_l|_{\mathbf{I}_i}.$$

The complexity of `AMFHSS` can be counted as follows. There are $O(r)$ matrix-vector multiplications at each level of the HSS tree, which has $O(\log n)$ levels. Also, for each node i, it costs $O(r^3)$ flops to compute the QR factorizations and the matrix multiplications. Typically, the number of nodes is $O(\frac{n}{r})$. Thus, the total cost of `AMFHSS` is $O(r\xi \log n) + O(r^2 n)$, where ξ is the cost of multiplying A with a vector. ξ is $O(n^2)$ for general dense A. But for some special matrices, ξ may be much smaller. For example, if A is a Cauchy-like matrix corresponding to a Toeplitz matrix in Fourier space [10, 25], then $\xi = O(n \log n)$ (such as when n is the product of small prime numbers).

3. Aggressive low-rank inertia evaluation.

Our eigensolver is a structured bisection scheme and it computes the inertia of the matrix A with shifts. The inertia of a symmetric matrix is defined as follows [12, 15].

DEFINITION 3.1. *For a symmetric matrix A, its inertia is the following triple of integers:*

$$\text{Inertia}(A) \equiv (\mathbf{n}_-(A), \mathbf{n}_0(A), \mathbf{n}_+(A)),$$

where $\mathbf{n}_-(A)$, $\mathbf{n}_0(A)$, and $\mathbf{n}_+(A)$ denote the numbers of negative, zero, and positive eigenvalues of A, respectively.

In the bisection scheme [28], Inertia$(A - sI)$ with different shifts s is evaluated, so as to detect the inertia change or find the number of eigenvalues in an interval.

Here, we seek to quickly evaluate Inertia$(A - sI)$ with low-rank structures. We illustrate an idea of aggressive low-rank approximation when only few eigenvalues are desired. That is, it is possible to discard most of the off-diagonal singular values of A to get a compact HSS approximation $\tilde{A}$ with Inertia$(A - sI)$ = Inertia$(\tilde{A} - sI)$, even if A does not have small off-diagonal numerical ranks. This can be analytically verified for a special case and numerically shown for more general ones.

Partition $A - sI$ into a block 2×2 form

$$A - sI = \begin{pmatrix} A_{11} & A_{12} \\ A_{12}^T & A_{22} \end{pmatrix} - sI,$$

Compute orthogonal diagonalizations

$$A_{11} = Q_1 \Lambda_1 Q_1^T, \quad A_{22} = Q_2 \Lambda_2 Q_2^T.$$

Then

$$A - sI = \begin{pmatrix} Q_1 & \\ & Q_2 \end{pmatrix} \left[\begin{pmatrix} \Lambda_1 & \Phi \\ \Phi^T & \Lambda_2 \end{pmatrix} - sI \right] \begin{pmatrix} Q_1^T & \\ & Q_2^T \end{pmatrix},$$

where

$$\Phi = Q_1^T A_{12} Q_2. \tag{3.1}$$

Thus, $A - sI$ has the same eigenvalues (and inertia) as

$$\hat{A} - sI \equiv \begin{pmatrix} \Lambda_1 & \Phi \\ \Phi^T & \Lambda_2 \end{pmatrix} - sI. \tag{3.2}$$

The idea of applying aggressive off-diagonal compression to $A - sI$ with its inertia preserved can be first illustrated with a special case:

$$\hat{A} - sI \equiv \begin{pmatrix} I & \Phi \\ \Phi^T & I \end{pmatrix} - sI. \tag{3.3}$$

(Such a special form is also useful in HSS preconditioning [38].) The following theorem shows how the inertia can be preserved after the compression of Φ when only the largest eigenvalues are desired.

Theorem 3.2. *Assume $\sigma_1, \sigma_2, \ldots, \sigma_N$ are the nonzero singular values of Φ in (3.3) ordered from the largest to the smallest, and r is a positive integer with $r \leq N$. Then for any shift s satisfying*

$$s > 1 + \sigma_{r+1}, \tag{3.4}$$

we have

$$\text{Inertia}(\hat{A} - sI) = \text{Inertia}(\tilde{A} - sI),$$

where $\tilde{A}$ is obtained from $\hat{A}$ in (3.3) with all the singular values $\sigma_{r+1}, \ldots, \sigma_N$ of Φ truncated. In particular, if

$$1 + \sigma_r > s \geq 1 + \sigma_{r+1},$$

then the r largest eigenvalues of $\hat{A} - sI$ and $\tilde{A} - sI$ are the same.

Proof. Let $\Phi = U \Sigma V^T$ be the SVD of Φ and $\Sigma = \text{diag}(\sigma_1, \ldots, \sigma_N)$. Then

$$\hat{A} = \begin{pmatrix} U & \\ & V \end{pmatrix} \begin{pmatrix} I & \Sigma \\ \Sigma & I \end{pmatrix} \begin{pmatrix} U^T & \\ & V^T \end{pmatrix},$$

which indicates that the eigenvalues of $\hat{A} - sI$ are either $1 - s$ or

$$1 \pm \sigma_i - s, \quad i = 1, \ldots, N.$$

The assumption for $\tilde{A}$ indicates

$$\tilde{A} = \begin{pmatrix} U & \\ & V \end{pmatrix} \begin{pmatrix} I & \Sigma_r \\ \Sigma_r & I \end{pmatrix} \begin{pmatrix} U^T & \\ & V^T \end{pmatrix},$$

where $\Sigma_r = \operatorname{diag}(\sigma_1, \ldots, \sigma_r, 0, \ldots, 0)$. Similarly, the eigenvalues of $\tilde{A} - sI$ are either $1 - s$ or

$$1 \pm \sigma_i - s, \quad i = 1, \ldots, r.$$

According to (3.4), if $s > 1 + \sigma_1$, then all the eigenvalues of $\hat{A} - sI$ and $\tilde{A} - sI$ are negative. If for certain integer j satisfying $1 \le j \le r$,

$$1 + \sigma_j > s > 1 + \sigma_{j+1}, \tag{3.5}$$

then both $\hat{A} - sI$ and $\tilde{A} - sI$ have exactly j positive eigenvalues

$$1 + \sigma_i - s, \quad i = 1, \ldots, j.$$

All of their remaining eigenvalues are negative. If $s = 1 + \sigma_j$ in (3.5) for certain $j \le r$, then both $\hat{A} - sI$ and $\tilde{A} - sI$ have $j - 1$ same positive eigenvalues and one zero eigenvalue. All of their remaining eigenvalues are negative.

Therefore, the results in the theorem hold. $\square$

Theorem 3.2 can be understood as follows. If we need to find the r largest eigenvalues of $\hat{A}$ and the shift s is larger than the $(r+1)$st eigenvalue as in (3.4), then we can aggressively truncate the singular values $\sigma_{r+1}, \ldots, \sigma_N$ of Φ. That is, with $s - 1$ as the threshold, all of the singular values smaller than it can be truncated, even if the numerical rank of Φ may be larger or if the singular values of Φ do not decay fast enough. After the truncation, the approximate matrix $\tilde{A} - sI$ is in a compact HSS form and has the same inertia as $\hat{A} - sI$. Inertia$(\tilde{A} - sI)$ can be computed quickly, as shown in the next section. Then we can apply the bisection method to $\tilde{A}$ instead of $\hat{A}$ to compute the r largest eigenvalues of $\hat{A}$. See Figure 3.1 for an example, which shows how the inertia can be preserved for a certain shift s.

One potential way of studying the connection of Theorem 3.2 to the general form (3.2) (which has the same eigenvalues as the general symmetric matrix A) is as follows. Write the SVD of Φ in (3.2) as

$$\Phi = U \Sigma V^T \equiv \begin{pmatrix} U_r & U_c \end{pmatrix} \begin{pmatrix} \Sigma_r & \\ & \Sigma_c \end{pmatrix} \begin{pmatrix} V_r^T \\ V_c^T \end{pmatrix}.$$

By truncating the trailing singular values in Σ_c, we get an approximation $\tilde{A}$ to $\hat{A}$. That is,

$$\begin{aligned} \hat{A} &= \tilde{A} + \begin{pmatrix} & U_c \Sigma_c V_c^T \\ V_c \Sigma_c U_c^T & \end{pmatrix} \\ &= \tilde{A} + \begin{pmatrix} & U_c \\ V_c & \end{pmatrix} \begin{pmatrix} \Sigma_c & \\ & \Sigma_c \end{pmatrix} \begin{pmatrix} & U_c \\ V_c & \end{pmatrix}^T. \end{aligned} \tag{3.6}$$

Assume the eigenvalues of A (and $\hat{A}$) are $\lambda_1 \ge \lambda_2 \ge \cdots \ge \lambda_n$, and those of $\tilde{A}$ are $\tilde{\lambda}_1 \ge \tilde{\lambda}_2 \ge \cdots \ge \tilde{\lambda}_n$. According to (3.6), $\hat{A}$ is equal to $\tilde{A}$ plus a symmetric low-rank update. Then the interlacing property of the eigenvalues (e.g., [15, p. 443]) means

$$\lambda_i \ge \tilde{\lambda}_i, \quad i = 1, 2, \ldots, n.$$

In fact, the removal of one singular value of Φ has the effect of shifting each λ_i toward λ_{i+1}. If the eigenvalues of A_{11} and A_{22} do not significantly vary, then (3.2) can be

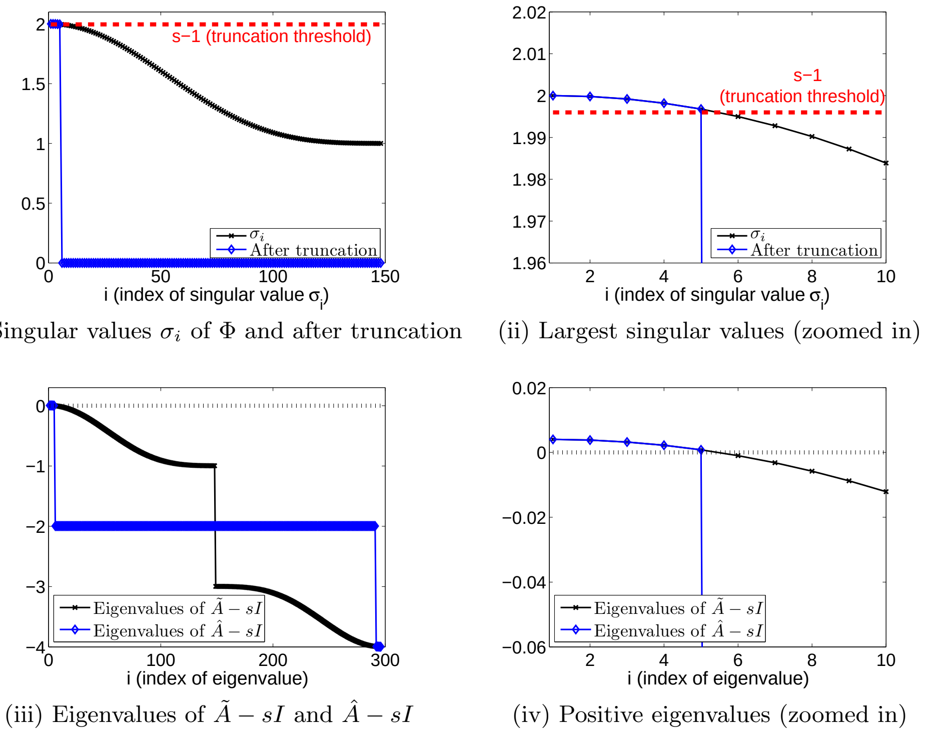


FIG. 3.1. *Truncation of the singular values σ_i of Φ and the eigenvalues of $\tilde{A}-sI$ and $\hat{A}-sI$, which indicate that $\tilde{A}-sI$ and $\hat{A}-sI$ have the same inertia.*

approximately related to the form in (3.3) after scaling. Thus, it is possible for such an aggressive low-rank truncation to roughly hold (in a weaker form) for more general cases.

In Figures 3.2–3.3, we use an example to demonstrate this possibility, where A has order $n = 1000$ and is the product of a random matrix with its transpose. The eigenvalues of A are shown in Figure 3.2(i). The singular values of Φ are shown in Figure 3.2(ii) and do not decay quickly.

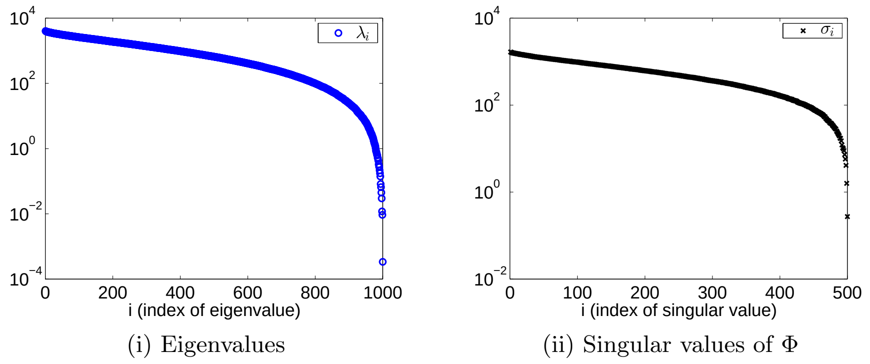


FIG. 3.2. *Eigenvalues of A and singular values of Φ for an example, where A is the product of a random matrix with its transpose.*

If we truncate most of the singular values of Φ and keep only the leading r of them, the impact on the leading r eigenvalues of A is not very significant. In Figure 3.3, for $r = 2, 4, 8, 16$, we can observe that the leading r eigenvalues of $\tilde{A}$ are quite close to those of A. That is, the shifting of the leading r eigenvalues is much smaller than that of the remaining ones. When $r = 16$, the two leading eigenvalues have more than four digits of accuracy. Then, if only a small number of the largest eigenvalues are desired, we can potentially truncate the off-diagonal singular values aggressively. In this case, if the shift s is not too far from the largest eigenvalues, we can accurately evaluate the inertia of $A - sI$ via that of $\tilde{A} - sI$. In fact, Figure 3.3 shows a significant gap between the leading r eigenvalues of $\tilde{A}$ and the remaining ones, which indicates a wide range for the choice of s.

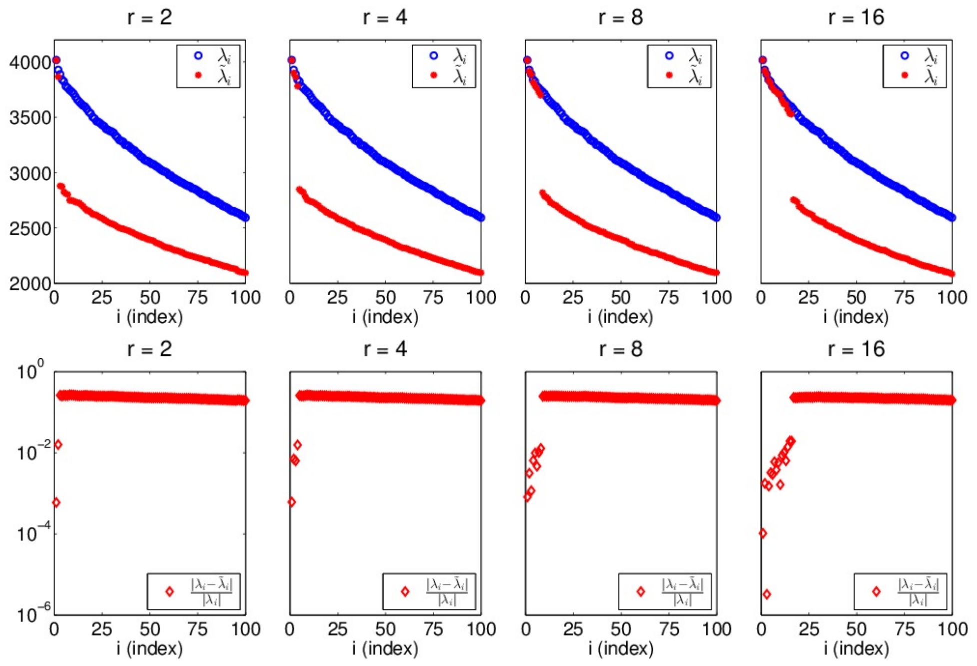


Fig. 3.3. *The eigenvalues λ_i of A in Figure 3.2 and the eigenvalues $\tilde{\lambda}_i$ of $\tilde{A}$ with only the leading r singular values of Φ kept in the compression of Φ.*

Although a full analytical justification of such an effect for general A is not yet available, we expect the results here to serve as a first effort in the study of low-rank inertia evaluations, and to provide a possible future direction for applying low-rank structures (which were previously applied often to problems with the low-rank property). We expect to extend the analysis in future work, probably in a weaker form. The experiments in Section 5 give further numerical support for this potential, including the performance for problems without significant low-rank properties.

Remark 3.1. Using a smaller HSS rank in the HSS construction makes both the construction and the factorization in the next section faster. Moreover, a smaller HSS rank also means that we can use more levels in the HSS tree to further improve the performance [35].

4. Fast inertia/LDL update for varying shifts in bisection.

Our eigensolver is based on the standard bisection scheme [12, 28]. For example, to find all of the eigenvalues within an interval $[a, b)$, we just need to compute $\mathbf{n}_-(A-bI)-\mathbf{n}_-(A-aI)$. Then we bisect the interval and repeat, until the interval size is small enough. In the scheme, a sequence of shifts s is used in evaluating $\mathbf{n}_-(A-sI)$.

To evaluate the inertia, Sylvester's inertia theorem [29] is often used, which states that the inertia is invariant under congruence transformations.

THEOREM 4.1. *(Sylvester's inertia theorem) For a symmetric matrix A and any invertible matrix S,*

$$\text{Inertia}(A) = \text{Inertia}(S^T AS).$$

Thus, if an LDL factorization of A as follows is computed:

$$A = \mathcal{L}\mathcal{D}\mathcal{L}^T, \tag{4.1}$$

then Theorem 4.1 implies that

$$\text{Inertia}(A) = \text{Inertia}(\mathcal{D}). \tag{4.2}$$

Here, we focus on the fast evaluation of the inertia with a generalized LDL factorization of the HSS approximation to $A - sI$, and especially the update of the factorization (and thus the inertia) when the shift s varies. For notational convenience, we use A to also represent its HSS approximation.

4.1. Fast inertia evaluation with HSS LDL factorization.

For the purpose of illustrating how to quickly evaluate Inertia(A) and how to update it with varying shifts s in the next subsection, we first briefly show the generalized LDL factorization of A. The factorization is a direct modification of the generalized Cholesky factorization in [37]. A similar version is also mentioned in [1] without an actual implementation. Other variants for $\mathcal{H}_l$-matrices can be found in [1, 2]. (Here, we are primarily interested in updating the factorization.) This generalized LDL factorization is not a standard LDL factorization, but can be written as (4.1) with $\mathcal{L}$ given by the product of a sequence of small orthogonal and triangular matrices.

The scheme traverses the nodes i of the HSS tree $\mathcal{T}$ following its postordering. If i is a leaf of $\mathcal{T}$, compute a QL factorization of U_i:

$$U_i = \bar{Q}_i \begin{pmatrix} 0 \\ \bar{U}_i \end{pmatrix}. \tag{4.3}$$

(This step is included for a general HSS form as in [7]. In our case, U_i already has orthonormal columns.) Then let

$$\bar{D}_i = \bar{Q}_i^T D_i \bar{Q}_i \equiv \begin{pmatrix} \bar{D}_{i;1,1} & \bar{D}_{i;1,2} \\ \bar{D}_{i;2,1} & \bar{D}_{i;2,2} \end{pmatrix}, \tag{4.4}$$

where $\bar{D}_i$ is partitioned so that $\bar{D}_{i;2,2}$ is a square matrix and has the same row size as $\bar{U}_i$. Compute an LDL factorization

$$\bar{D}_{i;1,1} = \hat{L}_{i;1,1}\hat{D}_i\hat{L}_{i;1,1}^T. \tag{4.5}$$

This yields

$$\begin{pmatrix} \bar{D}_{i;1,1} & \bar{D}_{i;1,2} \\ \bar{D}_{i;2,1} & \bar{D}_{i;2,2} \end{pmatrix} = \begin{pmatrix} \hat{L}_{i;1,1} & \\ \hat{L}_{i;2,1} & I \end{pmatrix} \begin{pmatrix} \hat{D}_i & \\ & S_i \end{pmatrix} \begin{pmatrix} \hat{L}_{i;1,1}^T & \hat{L}_{i;2,1}^T \\ & I \end{pmatrix}, \tag{4.6}$$

where

$$S_i = \bar{D}_{i;2,2} - \hat{L}_{i;2,1}\hat{D}_i\hat{L}_{i;2,1}^T, \quad \hat{L}_{i;2,1} = \bar{D}_{i;2,1}(\hat{D}_i\hat{L}_{i;1,1}^T)^{-1}.$$

If i is a non-leaf node with children c_1 and c_2, let

$$\tilde{D}_i = \begin{pmatrix} S_{c_1} & \bar{U}_{c_1}B_{c_1}\bar{U}_{c_2}^T \\ \bar{U}_{c_2}B_{c_1}^T\bar{U}_{c_1}^T & S_{c_2} \end{pmatrix}, \quad \tilde{U}_i = \begin{pmatrix} \bar{U}_{c_1}R_{c_1} \\ \bar{U}_{c_2}R_{c_2} \end{pmatrix}. \tag{4.7}$$

We can then remove c_1 and c_2 from $\mathcal{T}$. By induction, i becomes a new leaf with the associated generators $\tilde{D}_i, \tilde{U}_i, R_i, B_i$. Repeat the above process on node i. When $i = \mathrm{root}\,(\mathcal{T})$ is reached, compute an LDL factorization

$$\tilde{D}_i = \hat{L}_i\hat{D}_i\hat{L}_i^T. \tag{4.8}$$

See Figure 4.1 for a demonstration.

This factorization can be represented in the form of (4.1), where $\mathcal{L}$ is given by $\bar{Q}_i$, $\hat{L}_{i;1,1}$, etc., and $\mathcal{D}$ is a diagonal matrix with the diagonal given by the diagonal matrices $\hat{D}_i$ for all nodes i. Thus, (4.2) still holds. Unlike in [37], here we do not need to store $\mathcal{L}$. The cost of the scheme is $O(r^2n)$, with r the HSS rank of A.

Next, the inertia is counted with the $\hat{D}_i$ matrices as follows.

THEOREM 4.2. *After the generalized HSS LDL factorization, we have*

$$\mathrm{Inertia}(A) = \sum_{i=1}^{k}\mathrm{Inertia}(\hat{D}_i),$$

where $k = \mathrm{root}(\mathcal{T})$ and $\hat{D}_i$ is given in (4.6) or (4.8).

Proof. The matrix resulting from the removal of a node in the factorization process is a called a reduced (HSS) matrix in [36] (Figure 4.1(v)). Let

- $H^{(l_{\max})} \equiv A$ be the initial HSS form, where $\mathcal{T}$ has $l_{\max}$ levels;
- $H^{(l)}$ be the reduced matrix resulting from the partial elimination of $H^{(l+1)}$ due to the removal of all the nodes at level $l+1$ of $\mathcal{T}$;
- $P^{(l)}$ be an appropriate permutation matrix that merges the remaining blocks (from the partial elimination of $H^{(l+1)}$) corresponding to all the siblings at level $l+1$ to form $H^{(l)}$;
- $Q^{(l)}$ be a block diagonal matrix consisting of part of the factors:

$$Q^{(l)} = \mathrm{diag}\left(\bar{Q}_{i_1}^T\begin{pmatrix} \hat{L}_{i_1;1,1} & \\ \hat{L}_{i_1;2,1} & I \end{pmatrix}, \ldots, \bar{Q}_{i_\mu}^T\begin{pmatrix} \hat{L}_{i_\mu;1,1} & \\ \hat{L}_{i_\mu;2,1} & I \end{pmatrix}\right),$$

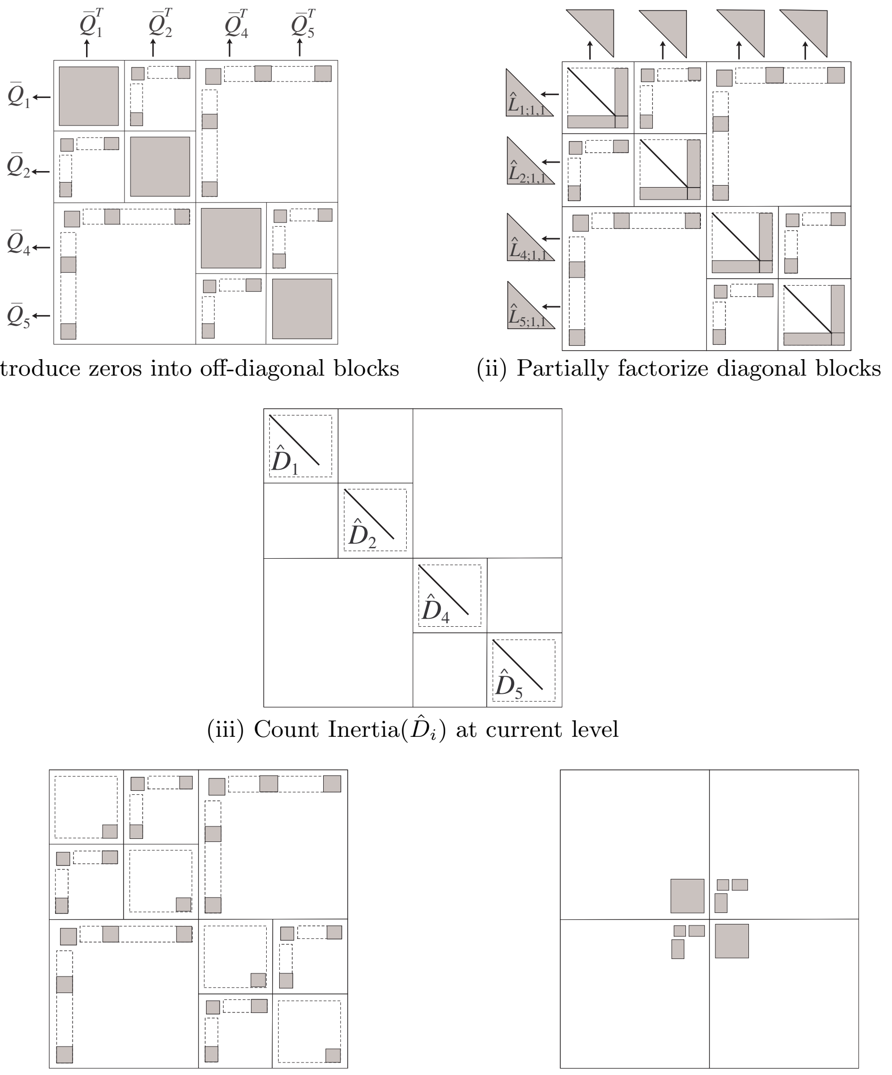
  where $i_1, \ldots, i_\mu$ are the nodes at level $l+1$ of $\mathcal{T}$.

With a mechanism similar to that in [36], it can be shown that the generalized HSS LDL factorization can be represented recursively as

$$(P^{(l)}Q^{(l)})H^{(l+1)}(P^{(l)}Q^{(l)})^T = \begin{pmatrix} \mathrm{diag}(\hat{D}_{i_1}, \ldots, \hat{D}_{i_\mu}) & \\ & H^{(l)} \end{pmatrix}.$$

According to Theorem 4.1, we then have

$$\begin{aligned}\mathrm{Inertia}(H^{(l+1)}) &= \mathrm{Inertia}\begin{pmatrix} \mathrm{diag}(\hat{D}_{i_1}, \ldots, \hat{D}_{i_\mu}) & \\ & H^{(l)} \end{pmatrix} \\ &= \mathrm{Inertia}(H^{(l)}) + \sum_{i\text{: all nodes at level } l}\mathrm{Inertia}(\hat{D}_i).\end{aligned}$$



(i) Introduce zeros into off-diagonal blocks (ii) Partially factorize diagonal blocks

(iii) Count Inertia($\hat{D}_i$) at current level

(iv) Blocks remaining to be factorized (v) Form a reduced HSS matrix

FIG. 4.1. *Illustration of the generalized LDL factorization scheme based on [37] and the inertia count for an HSS matrix.*

Thus,

$$\begin{aligned}\text{Inertia}(A) &= \text{Inertia}(H^{(l_{\max})}) \\ &= \sum_{l=l_{\max}}^{0}\left(\sum_{i:\text{ all nodes at level } l} \text{Inertia}(\hat{D}_i)\right) \\ &= \sum_{i=1}^{k} \text{Inertia}(\hat{D}_i).\end{aligned}$$

□

REMARK 4.1. In the generalized HSS LDL factorization, the orthogonal transformations (4.3)–(4.4) for introducing zeros into the off-diagonal blocks are stable. The regular LDL factorization is applied to only the small diagonal blocks $\bar{D}_{i;1,1}$ as in (4.5). In our algorithm, for the stability purpose, we incorporate the Bunch-Kaufman pivoting [5] in the LDL factorization of $\bar{D}_{i;1,1}$. The LDL factorization then actually looks like $P_i^T \bar{D}_{i;1,1} P_i = \bar{L}_i \hat{D}_i \bar{L}_{i;}^T$, where P_i is a permutation matrix. Thus, in (4.5), we eventually have $\hat{L}_{i;1,1} \equiv P_i \bar{L}_i$. In addition, it is shown in [34] that such type of HSS factorizations usually have much better stability than standard dense factorizations (for the same matrix) due to the hierarchical structure and the use of orthogonal off-diagonal operations.

4.2. Fast inertia/LDL update for varying shifts. Next, we show how to quickly update the generalized HSS LDL factorization of A in order to get that of $A-sI$ in bisection. This enables us to update the inertia evaluation for different shifts s. We make the following essential observations for the major steps when multiple shifts are involved:

1. For all the nodes i of $\mathcal{T}$, the zero introduction step (4.3) is identical.
2. For all the leaves i of $\mathcal{T}$, the diagonal update step (4.4) is identical up to a diagonal update $-sI$ to $\bar{D}_i$, since

$$\bar{Q}_i^T (D_i - sI)\bar{Q}_i = \bar{D}_i - sI. \tag{4.9}$$

   (For all the non-leaf nodes i of $\mathcal{T}$, it is also possible to save, since $\bar{U}_{c_1} B_{c_1} \bar{U}_{c_2}^T$ in (4.7) remains the same. We do not yet take advantage of this feature.)
3. For all the nodes i of $\mathcal{T}$, the merging step (4.7) is identical (in terms of the multiplications).

Thus, the results in steps (4.3) and (4.7) only need to be computed once and then remain the same for multiple s. The result in step (4.4) is also computed once (for half of the nodes) and can then be updated for multiple s with little additional work. That is, we can first store the appropriate results from the generalized LDL factorization of A in a precomputation. The results are then used to quickly update the factorization in order to get that of $A - sI$. See Algorithms 1 and 2. Notice that some steps in Algorithm 1 are totally omitted in Algorithm 2. Also in practice, the precomputation is done for the first shift s.

Algorithm 1 Precomputation (generalized LDL factorization of A)

```
1:  procedure (Input: HSS generators for $A$; output: stored results)
2:      for node $i$ from 1 to $k-1$ do                          ▷ $k \equiv \mathrm{root}(\mathcal{T})$
3:          Compute (4.3)–(4.6)          ▷ Some results are not needed by Algorithm 2
4:          if $i$ is a non-leaf node then
5:              $D_i \leftarrow \tilde{D}_i$, $U_i \leftarrow \tilde{U}_i$ for $\tilde{D}_i$ and $\tilde{U}_i$ in (4.7)
6:              Store $\bar{Q}_i$ and $F_i = \bar{U}_{c_1} B_{c_1} \bar{U}_{c_2}^T$
7:          else
8:              Store $\bar{D}_i$
9:          end if
10:     end for
11: end procedure
```

The difference between the precomputation and the additional factorizations can also be clearly seen from Table 4.1, where we list the costs of the major operations in the factorization. The saving in the cost of the LDL factorization for each s can be

Algorithm 2 Fast evaluation of Inertia$(A - sI)$

```
 1: procedure (Input: stored results and a shift $s$; output: $\mathcal{I} \equiv$ Inertia$(A - sI)$)
 2:     $\mathcal{I} \leftarrow 0$
 3:     for node $i$ from 1 to $k-1$ do                          ▷ $k \equiv$ root$(\mathcal{T})$
 4:         if $i$ is a leaf then          ▷ Reusing stored result from line 8 of Algorithm 1
 5:             $\bar{D}_i \leftarrow \bar{D}_i - sI$
 6:         else                           ▷ Reusing stored result from line 6 of Algorithm 1
 7:             $\bar{D}_i \leftarrow \bar{Q}_i^T \begin{pmatrix} S_{c_1} & F_i \\ F_i^T & S_{c_2} \end{pmatrix} \bar{Q}_i$
 8:         end if
 9:         Compute (4.5)–(4.6)
10:         $\mathcal{I} \leftarrow \mathcal{I} + $ Inertia$(\hat{D}_i)$
11:     end for
12:     $\bar{D}_k = \hat{L}_k \hat{D}_k \hat{L}_k^T$
13:     $\mathcal{I} \leftarrow \mathcal{I} + $ Inertia$(\hat{D}_k)$
14: end procedure
```

counted accordingly. In fact, the precomputation and each additional factorization cost about $\xi_1 = \frac{68}{3}kr^3 = \frac{68}{3}r^2n$ and $\xi_2 = \frac{28}{3}kr^3 = \frac{28}{3}r^2n$ flops, respectively, where the low order terms have been dropped. Thus,

$$\frac{\xi_2}{\xi_1} = \frac{\frac{28}{3}r^2n}{\frac{68}{3}r^2n} \approx 41\%.$$

That is, after the precomputation, the factorization for each s can save nearly 60% of the work. In fact, the saving may be even bigger, due to the statement below (4.9).

TABLE 4.1
Flops (leading terms only) of the major steps in the HSS LDL factorization, where we assume that the leaf level D_i generators have sizes $2r$ and the HSS rank is r.

Operation	Flops	Number of times used	
		Precomputation	Each s
(4.3)	$\frac{10r^3}{3}$	$\times k$	0
(4.4)	$12r^3$	$\times k$	$\times \frac{k}{2}$
(4.5)	$\frac{r^3}{3}$	$\times k$	$\times k$
(4.6)	$3r^3$	$\times k$	$\times k$
(4.7)	$8r^3$	$\times \frac{k}{2}$	0

The complexity of finding one eigenvalue is $O(r^2n)$, where we assume that the number of bisection steps for each eigenvalue is bounded.

5. Numerical experiments.

We use some examples to demonstrate the complexity and the accuracy of our structured eigensolver. The following notation is used in the tests:

- n: the order of the matrix A under consideration;
- m: the size of the leaf level diagonal blocks in an HSS approximation;
- $\tilde{r}$: the initial sampling size or rank estimate in the off-diagonal compression in adaptive randomized HSS constructions;
- τ: the relative tolerance for off-diagonal compression;

- δ: the minimal interval length in bisection;
- λ_i or $\lambda_i(A)$: the eigenvalues of A ordered from the largest (λ_1) to the smallest (λ_n) (here, we use the results from the Matlab function `eig` as these exact eigenvalues);
- $\tilde{\lambda}_i$: the computed numerical eigenvalue with our eigensolver;
- x: the vector of all the (or selected) eigenvalues λ_i;
- $\tilde{x}$: the vector of all the (or selected) computed eigenvalues $\tilde{\lambda}_i$;
- γ: the relative error $\frac{||x-\tilde{x}||_2}{||x||_2}$;
- `AHSS`: the adaptive HSS construction mentioned at the beginning of Section 2.2;
- `AMFHSS`: the adaptive matrix-free HSS construction in the remaining part of Section 2.2;
- `NEW`: our new eigensolver with either `AHSS` or `AMFHSS` for the HSS construction.

Example 1. We start with an illustration of the feasibility of using a compact HSS approximation for the inertia evaluation of a matrix A (with shifts), when the off-diagonal numerical ranks of A are not very small. Consider the following Helmholtz equation defined in a 3D cube Ω:

$$(-\Delta - \frac{\omega^2}{v(x)^2})u(x,\omega) = f(x,\omega),$$
$$u = 0 \text{ on } \delta\Omega,$$

where $v(x)$ is the velocity and $\omega = 10$Hz is the angular frequency. The matrix A under consideration is the last Schur complement in the factorization of the discretized matrix on a $30\times30\times30$ mesh after the nested dissection ordering. A has size $n = 900$ and is symmetric and indefinite.

It is known that the off-diagonal singular values of A usually do not decay fast enough in these 3D cases, such that the off-diagonal numerical ranks (with a small tolerance τ) are not small [32]. Figure 5.1(i) shows the first 100 singular values of $\Phi = A|_{(1:450)\times(451:900)}$. For $\tau = 10^{-4}$, the numerical rank of Φ is already about 94.

Even so, we can still construct a compact HSS approximation $\tilde{A}$ to A so as to accurately evaluate the inertia of $A - sI$ for some shifts s. For example, when $\mathcal{T}$ has two levels and $s \in (\lambda_{i+1}(A), \lambda_i(A))$ for $i = 10$ and 100, we plot the difference $|\mathbf{n}_-(A-sI) - \mathbf{n}_-(\tilde{A}-sI)|$ in Figure 5.1(ii–iii). Clearly, if we manually set a numerical rank r for Φ to be about 35, we can already get $\mathbf{n}_-(\tilde{A} - sI) = \mathbf{n}_-(A - sI)$. Even the choices of $r = 1$ in Figure 5.1(ii) and $r = 8$ in Figure 5.1(iii) give quite close estimates to $\mathbf{n}_-(A - sI)$, where $|\mathbf{n}_-(A - sI) - \mathbf{n}_-(\tilde{A} - sI)| = 2$.

Example 2. Next, we look at a symmetric Toeplitz matrix T with its first column given by

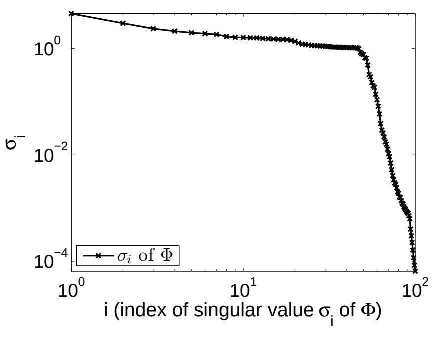
$$t_{0:n-1} = \text{randn}(n, 1).$$

To find the eigenvalues of T, we first convert it into the following matrix (called *Cauchy-like* matrix) by an orthogonal transformation:

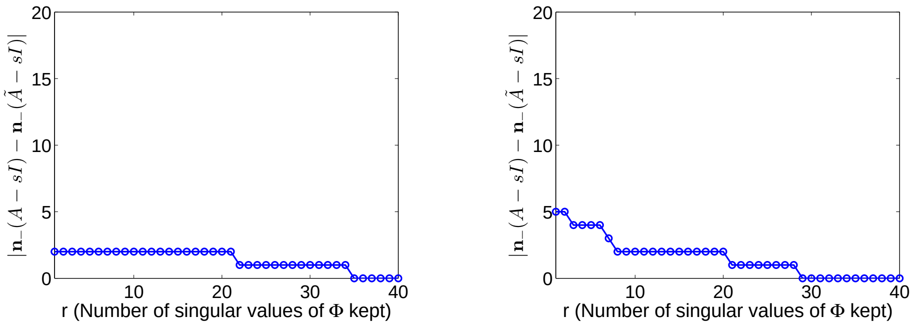
$$\mathcal{C} = \mathcal{F}_n T \mathcal{F}_n^*, \tag{5.1}$$

where $\mathcal{F}_n$ is the order-n normalized inverse discrete Fourier transform matrix

$$\mathcal{F}_n = \frac{1}{\sqrt{n}}(\omega_n^{2(i-1)(j-1)+i})_{1\le i,j\le n}, \quad \text{with } \omega_n = e^{\frac{\pi \mathbf{i}}{n}}, \quad \mathbf{i} = \sqrt{-1}.$$



(i) The first 100 singular values σ_i of Φ



(ii) $|\mathbf{n}_-(A-sI)-\mathbf{n}_-(\tilde{A}-sI)|$ with $s\in(\lambda_{11},\lambda_{10})$ (iii) $|\mathbf{n}_-(A-sI)-\mathbf{n}_-(\tilde{A}-sI)|$ with $s\in(\lambda_{101},\lambda_{100})$

FIG. 5.1. *Example 1: Accuracy in the evaluation of* Inertia$(A-sI)$ *by keeping only few singular values of* $\Phi = A|_{(1:450)\times(451:900)}$.

This choice of $\mathcal{F}_n$ enables $\mathcal{C}$ to remain real and symmetric [25, Lemma 2(iii)], and the entries of $\mathcal{C}$ can be conveniently found with displacement structures [14, 18, 19, 21, 22, 26]. In fact, each entry of $\mathcal{C}$ can be computed in about $O(\log n)$ flops on average.

It is known that $\mathcal{C}$ has small off-diagonal numerical ranks. In fact, the off-diagonal numerical ranks r (with a given tolerance) satisfy $r = O(\log n)$ [10, 25, 27, 39]. This rank property together with the fast matrix-vector multiplication based on (5.1) enable us to quickly approximate $\mathcal{C}$ by randomized HSS construction algorithms. In fact, the cost is either $O(n\log^2 n)$ with `AHSS`, or $O(n\log^3 n)$ with `AMFHSS`. Note that the idea of a rank pattern [35] makes the actual performance of the methods better than the estimates, similarly to [39].

We first demonstrate the performance of the adaptive randomized HSS constructions for $\mathcal{C}$. We compare `AHSS` and `AMFHSS`. See Table 5.1 and Figure 5.2. Two reference lines corresponding to $\hat{c}n\log^2 n$ and $\tilde{c}n\log^3 n$ with appropriate constants $\hat{c}$ and $\tilde{c}$ are plotted and marked as $O(n\log^2 n)$ and $O(n\log^3 n)$ in Figure 5.2. These two lines are used as references to compare with the slopes of the lines for the flops of `AHSS` and `AMFHSS`. The performance of both methods follows the prediction, and in fact, these complexity bounds overestimate the costs.

`AHSS` is faster, but requires selected entries of $\mathcal{C}$. Here, since the entries of $\mathcal{C}$ can be quickly computed, we use `AHSS` in our Toeplitz eigenvalue solutions.

Note that $\tau = 10^{-4}$ is used in the HSS construction, so that the accuracy of the HSS construction is lower than that used in Toeplitz solutions [39]. This accuracy

TABLE 5.1
Example 2: Flops of the adaptive HSS construction method `AHSS` *and the adaptive matrix-free version* `AMFHSS`*, where $\tau = 10^{-4}$, $\delta = 10^{-8}$, $m = 40$, and $\tilde{r} = 30$.*

n	80	160	320	640	1,280	2,560	5,120	10,240	20,480
`AMFHSS`	$2.03e6$	$9.48e6$	$3.40e7$	$1.02e8$	$2.99e8$	$8.90e8$	$2.40e9$	$6.18e9$	$1.62e10$
`AHSS`	$8.14e5$	$2.10e6$	$4.74e6$	$1.07e7$	$2.20e7$	$4.65e7$	$9.58e7$	$1.99e8$	$4.06e8$

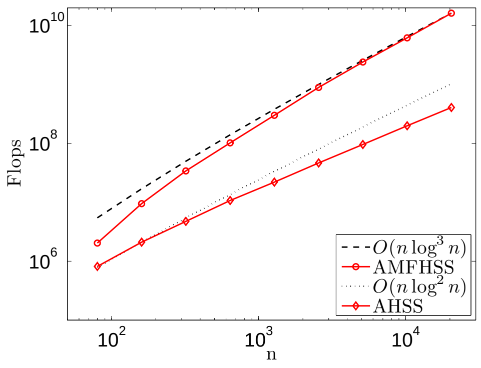


FIG. 5.2. *Example 2: Costs of the HSS construction methods as in Table 5.1, together with the complexity bounds.*

is sufficient for our purpose of inertia evaluations here. In fact, Table 5.2 shows the performance of `NEW` for finding some interior eigenvalues of T with different τ in the HSS construction. Using $\tau = 10^{-4}$ costs less than with a higher accuracy such as 10^{-8}, but gives comparable accuracies in the eigenvalue solutions (with the same δ). This is also the case for the tests below.

TABLE 5.2
Example 2: Performance of our eigensolver to compute 10 eigenvalues λ_i for $i = 325, 326, \ldots, 334$ of the random Toeplitz matrix of order $n = 1,280$, where $m = 40$, $\tilde{r} = 30$, and $\delta = 10^{-8}$.

τ		1e−1	1e−2	1e−3	1e−4	1e−5	1e−6	1e−7	1e−8
Flops	HSS construction	$1.63e7$	$1.80e7$	$1.94e7$	$2.21e7$	$2.31e7$	$2.42e7$	$2.47e7$	$2.51e7$
	Precomputation	$6.52e6$	$8.39e6$	$1.18e7$	$1.19e7$	$1.62e7$	$2.04e7$	$2.41e7$	$2.72e7$
	LDL per shift	$1.68e6$	$2.68e6$	$4.49e6$	$4.24e6$	$6.49e6$	$8.77e6$	$1.08e7$	$1.25e7$
	Total (all steps)	$3.72e8$	$6.43e8$	$9.43e8$	$1.09e9$	$1.65e9$	$2.05e9$	$2.52e9$	$2.92e9$
Time		$6.68e0$	$7.89e0$	$7.16e0$	$8.80e1$	$8.96e0$	$8.77e1$	$8.86e1$	$8.93e1$
γ		$3.91e{-3}$	$5.37e{-4}$	$4.42e{-3}$	$7.53e{-6}$	$4.72e{-6}$	$1.00e{-6}$	$6.90e{-7}$	$1.14e{-6}$

We would like to mention that a smaller τ may be needed for other problems, especially those with clustered eigenvalues. For such cases, the fast bisection method here may be used to provide an initial guess for Newton's method as in [11]. We expect to study how the choice of τ affects the accuracy of clustered eigenvalues in future work.

EXAMPLE 3. We now consider the Kac-Murdock-Szegö (KMS) Toeplitz matrix T as in [30], with its first column given by

$$t_{i-j} = \rho^{|i-j|}, \quad \rho = 0.5, \quad i = 1, \ldots, n, \quad j = 1, \ldots, n.$$

We first show the performance of NEW for finding all of the eigenvalues. The size of the matrix ranges from 80 to 10, 240. The results (flops, time, and errors) are given in Table 5.3. The cost of NEW scales roughly quadratically. This can also be seen from Figure 5.3, where a reference line for $O(n^2 \log^2 n)$ is included. Figure 5.3 also gives the comparison of the scaling between NEW and the Matlab function eig, which costs $O(n^3)$. NEW is slower since n is not large enough. We expect NEW to be faster for $n > 4 \times 10^4$. In fact, when n grows beyond 10, 240, eig runs out of memory which scales as $O(n^2)$. On the other hand, NEW needs only about $O(n)$ memory.

TABLE 5.3
Example 3: Flops, time (in seconds), and relative errors of NEW for finding all the eigenvalues of the KMS matrix of size n, where $m = 40$, $\tilde{r} = 10$, and $\tau = 10^{-4}$.

n	80	160	320	640	1,280	2,560	5,120	10,240
Flops	$9.08e7$	$3.17e8$	$1.19e9$	$4.72e9$	$1.82e10$	$7.14e10$	$2.82e11$	$1.12e12$
Time (s)	$2.06e0$	$7.45e0$	$2.93e1$	$1.13e2$	$4.47e2$	$1.75e3$	$6.88e3$	$2.74e4$
γ	$1.27e{-9}$	$1.28e{-9}$	$1.33e{-9}$	$1.42e{-9}$	$1.51e{-9}$	$1.62e{-9}$	$1.65e{-9}$	$1.64e{-9}$

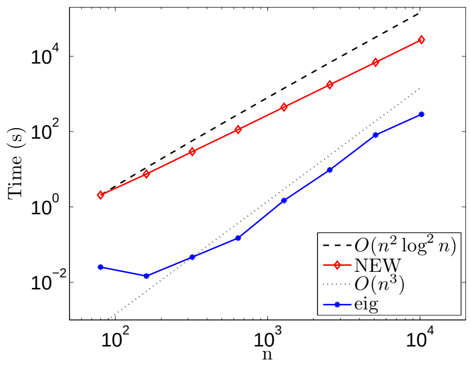


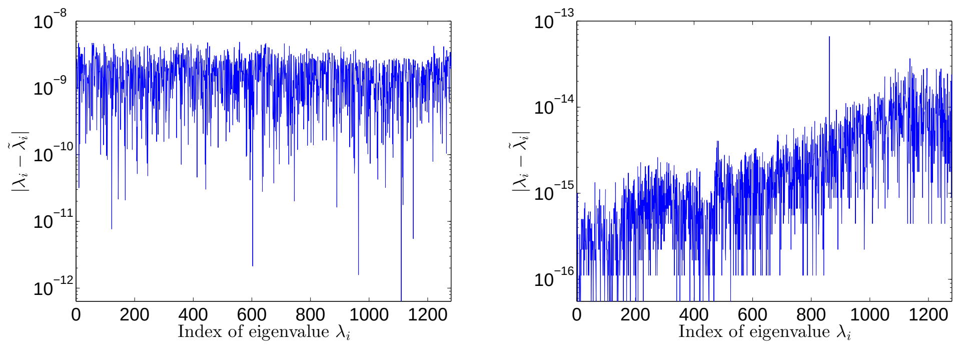
FIG. 5.3. *Example 3: Comparison of how the computation time of NEW and eig scales for finding all the eigenvalues of the KMS matrix of size n, where the details for NEW are given in Table 5.3.*

We also plot the absolute difference $|\lambda_i - \tilde{\lambda}_i|$ for each eigenvalue in Figure 5.4. With $\delta = 10^{-8}$ or $\delta = 10^{-15}$, we can get the desired accuracy. Note that $\tau = 10^{-4}$ is still sufficient.

Table 5.4 shows the performance of NEW for the computation of 10 interior eigenvalues clustered around 0.49. Clearly, the cost of NEW scales roughly linearly, while that of the Matlab function eigs scales nearly quadratically, as can be seen from Figure 5.5.

6. Conclusions.

This paper studies the structured eigenvalue solution of symmetric matrices with the low-rank property or even the weak rank property. We specifically present an adaptive matrix-free HSS construction, and justify theoretically (for certain cases) or numerically the effectiveness of using aggressive low-rank



(i) $\delta = 10^{-8}$ (ii) $\delta = 10^{-15}$

FIG. 5.4. *Example 3:* $|\lambda_i - \tilde{\lambda}_i|$ *for a* $1,280 \times 1,280$ *KMS matrix in Example 3, where* $\hat{n} = 40$, $\tilde{r} = 10$, *and* $\tau = 10^{-4}$ *in* `NEW`.

TABLE 5.4
Example 3: Flops, time (in seconds), and relative error of `NEW` *for finding* 10 *eigenvalues near* 0.49 *of the KMS matrix, where* $m = 40$, $\tilde{r} = 10$, *and* $\tau = 10^{-4}$.

n	80	160	320	640	1,280	2,560	5,120	10,240	20,480
Flops	1.04e7	1.91e7	3.88e7	7.76e7	1.48e8	2.82e8	5.39e8	1.02e8	1.93e9
Time (s)	2.94e–1	9.31e–1	1.30e0	2.60e0	5.12e0	1.04e1	2.14e1	4.70e1	9.31e1
γ	4.18e–9	4.62e–9	3.51e–9	3.36e–9	4.05e–9	2.81e–9	3.15e–9	3.09e–9	3.79e–9

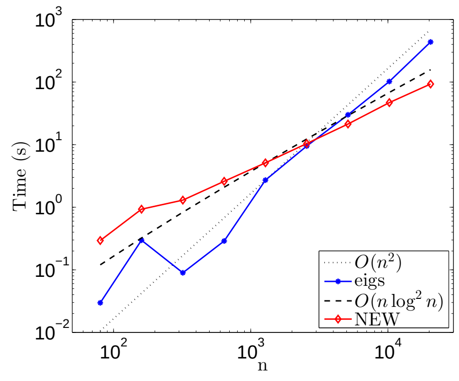


FIG. 5.5. *Example 3: Comparison of how the computation time of* `NEW` *and* `eigs` *scales for finding* 10 *eigenvalues near* 0.49 *of the KMS matrix, where the details for* `NEW` *are given in Table 5.4.*

off-diagonal approximations for inertia evaluations. The fast inertia evaluation with the generalized HSS LDL factorization is shown. A feature of quickly updating the generalized LDL factorization is very useful for updating the inertia evaluation with multiple shifts in the bisection method. The cost is about $O(n)$ for finding one eigenvalue. A useful application is the eigenvalue solution of symmetric Toeplitz problems.

On the other hand, to find all the eigenvalues, it may be possible to further improve the efficiency by combining HSS techniques with the divide-and-conquer method. Just like the way the fast multipole method is used to accelerate the divide-and-conquer method for certain symmetric matrices with off-diagonal ranks at most 1 [8], HSS methods can be used for higher off-diagonal ranks. The systematic and convenient HSS operations can lead to enhanced flexibility. The overall cost for all the eigenvalues may be potentially reduced to less than $O(n^2)$.

Acknowledgements. The authors are very grateful to the editor and the three anonymous referees for their valuable suggestions. Also thank James Vogel for helping with the revision.

REFERENCES

[1] P. Benner and T. Mach, *Computing all or some eigenvalues of symmetric $\mathcal{H}_l$-matrices*, Max Planck Institute Magdeburg Preprint MPIMD/10-01, http://www.mpi-magdeburg.mpg.de/preprints, (2010).

[2] P. Benner and T. Mach, *Computing all or some eigenvalues of symmetric $\mathcal{H}_l$-matrices*, SIAM J. Sci. Comput., 34-1 (2012), pp. A485–A496.

[3] D. A. Bini, L. Gemignani, and V. Y. Pan, *Fast and stable QR eigenvalue algorithms for generalized companion matrices and secular equations*, Numer. Math. 100 (2005), pp. 373–408.

[4] S. Börm, L. Grasedyck, and W. Hackbusch, *Introduction to hierarchical matrices with applications*, Eng. Anal. Bound. Elem, 27 (2003), pp. 405–422.

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

[6] S. Chandrasekaran, P. Dewilde, M. Gu, T. Pals, X. Sun, A.-J. van der Veen, and D. White, *Some fast algorithms for sequentially semiseparable representations*, SIAM J. Matrix Anal. Appl., 27 (2005), pp. 341–364.

[7] S. Chandrasekaran, P. Dewilde, M. Gu, and T. Pals, *A fast ULV decomposition solver for hierarchically semiseparable representations*, SIAM J. Matrix Anal. Appl., 28 (2006), pp. 603–622.

[8] S. Chandrasekaran and M. Gu, *A divide-and-conquer algorithm for the eigendecomposition of symmetric block-diagonal plus semiseparable matrices*, Numer. Math., 96 (2004), pp. 723–731.

[9] S. Chandrasekaran, M. Gu, J. Xia, and J. Zhu, *A fast QR algorithm for companion matrices*, Oper. Theory Adv. Appl., Birkhauser Basel, 179 (2007), pp. 111–143.

[10] S. Chandrasekaran, M. Gu, X. Sun, J. Xia, and J. Zhu, *A superfast algorithm for Toeplitz systems of linear equations*, SIAM J. Matrix Anal. Appl., 29 (2007), pp. 1247–1266.

[11] G. Cybenko and C. Van Loan, *Computing the minimum eigenvalue of a symmetric positive definite Toeplitz matrix*, SIAM J. Sci. Stat. Comput., 7 (1986), pp. 123–131.

[12] J. W. Demmel, *Applied Numerical Linear Algebra*, SIAM, Philadelphia, PA, 1997.

[13] Y. Eidelman, I. Gohberg and V. Olshevsky, *The QR iteration method for Hermitian quasiseparable matrices of an arbitrary order*, Linear Alg. Appl., 404 (2005), pp. 305–324.

[14] I. Gohberg, T. Kailath, and V. Olshevsky, *Fast Gaussian elimination with partial pivoting for matrices with displacement structure*, Math. Comp., 64 (1995), pp. 1557–1576.

[15] G. H. Golub and C. V. Loan, *Matrix Computations*, The Johns Hopkins University Press, Baltimore, MD, 4rd edition, 2013.

[16] M. Gu and S. C. Eisenstat, *A divide-and-conquer algorithm for the symmetric tridiagonal eigenproblem*, SIAM J. Matrix Anal. Appl., 16 (1995), pp. 79–92.

[17] L. Greengard and V. Rokhlin, *A fast algorithm for particle simulations*, J. Comp. Phys., 73 (1987), pp. 325–348.

[18] M. Gu, *Stable and efficient algorithms for structured systems of linear equations*, SIAM J.

Matrix Anal. Appl., 19 (1998), pp. 279–306.

[19] M. Gu, *New fast algorithms for structured linear least squares problems*, SIAM J. Matrix Anal. Appl., 19 (1998), pp. 279–306.

[20] N. Halko, P. G. Martinsson, and J. A. Tropp, *Finding structure with randomness probabilistic algorithms for constructing approximate matrix decompositions*, SIAM Rev., 53 (2011), pp. 217–288.

[21] G. Heinig, *Inversion of generalized Cauchy matrices and other classes of structured matrices*, in Linear Algebra for Signal Processing, IMA Vol. Math. Appl. 69, Springer, New York, 1995, pp. 63–81.

[22] T. Kailath, S. Kung, and M. Morf, *Displacement ranks of matrices and linear equations*, J. Math. Anal. Appl., 68 (1979), pp. 395–407.

[23] L. Lin, J. Lu, and L. Ying, *Fast construction of hierarchical matrix representation from matrix-vector multiplication*, J. Comput. Phys., 230 (2011), pp. 4071–4087.

[24] P. G. Martinsson, *A fast randomized algorithm for computing a hierarchically semiseparable representation of a matrix*, SIAM. J. Matrix Anal. Appl., 32 (2011), pp. 1251–1274.

[25] P. G. Martinsson, V. Rokhlin, and M. Tygert, *A fast algorithm for the inversion of general Toeplitz matrices*, Comput. Math. Appl. 50 (2005), pp. 741–752.

[26] V. Y. Pan, *On computations with dense structured matrices*, Math. Comp., 55 (1990), pp. 179–190.

[27] V. Y. Pan, *Transformations of matrix structures work again*, arXiv:1303.0353, (2013).

[28] B. N. Parlett, *The Symmetric Eigenvalue Problem*, SIAM, PA, 1998.

[29] J. J. Sylvester, *A demonstration of the theorem that every homogeneous quadratic polynomial is reducible by real orthogonal substitutions to the form of a sum of positive and negative squares*, Philosophical Magazine, IV (1852), pp. 138–142. doi:10.1080/14786445208647087.

[30] W. F. Trench, *Numerical solution of the eigenvalue problem for Hermitian Toeplitz matrices*, SIAM J. Matrix Anal. Appl., 10 (1989), pp. 135-156.

[31] M. Van Barel, R. Vandebril, P. Van Dooren, and K. Frederix, *Implicit double shift QR-algorithm for companion matrices*, Numer. Math., 116 (2010), pp. 177–212.

[32] S. Wang, M. V. de Hoop, J. Xia, and X. S. Li, *Massively parallel structured multifrontal solver for time-harmonic elastic waves in 3D anisotropic media*, Geophys. J. Int., 191 (2012), pp. 346–366.

[33] F. Woolfe, E. Liberty, V. Rokhlin, and M. Tygert, *A fast randomized algorithm for the approximation of matrices*, Appl. Comput. Harmon. Anal., 25 (2008), pp.335–366.

[34] Y. Xi, J. Xia, S. Cauley, and V. Balakrishnan, *Superfast and stable structured solvers for Toeplitz least squares via randomized sampling*, SIAM J. Matrix Anal. Appl., revised, (2013).

[35] J. Xia, *On the complexity of some hierarchical structured matrix algorithms*, SIAM J. Matrix Anal. Appl., 33 (2012), pp. 388–410.

[36] J. Xia, *Efficient structured multifrontal factorization for general large sparse matrices*, SIAM J. Sci. Comput., 35 (2013), pp. A832–A860.

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

[38] J. Xia and M. Gu, *Robust approximate Cholesky factorization of rank-structured symmetric positive definite matrices*, SIAM J. Matrix Anal. Appl., 31 (2010), pp. 2899–2920.

[39] J. Xia, Y. Xi, and M. Gu, *A superfast structured solver for Toeplitz linear systems via randomized sampling*, SIAM J. Matrix Anal. Appl., 33 (2012), pp. 837–858.