SUPERFAST DIVIDE-AND-CONQUER METHOD AND PERTURBATION ANALYSIS FOR STRUCTURED EIGENVALUE SOLUTIONS

JAMES VOGEL*, JIANLIN XIA*, STEPHEN CAULEY†, AND VENKATARAMANAN BALAKRISHNAN‡

Abstract. We present a *superfast divide-and-conquer* method for finding all the eigenvalues as well as all the eigenvectors (in a structured form) of a class of symmetric matrices with off-diagonal ranks or numerical ranks bounded by r, as well as the approximation accuracy of the eigenvalues due to off-diagonal compression. More specifically, the complexity is $O(r^2 n \log n) + O(rn \log^2 n)$, where n is the order of the matrix. Such matrices are often encountered in practical computations with banded matrices, Toeplitz matrices (in Fourier space), and certain discretized problems. They can be represented or approximated by hierarchically semiseparable (HSS) matrices. We show how to preserve the HSS structure throughout the dividing process that involves recursive updates, and how to quickly perform stable eigendecompositions of the structured forms. Various other numerical issues are discussed, such as computation reuse and deflation. The structure of the eigenvector matrix is also shown. We further analyze the structured perturbation, i.e., how compression of the off-diagonal blocks impacts the accuracy of the eigenvalues. They show that rank structured methods can serve as an effective and efficient tool for approximate eigenvalue solutions with controllable accuracy. The algorithm and analysis are very useful for finding the eigendecomposition of matrices arising from some important applications, and can be modified to find SVDs of nonsymmetric matrices. The efficiency and accuracy are illustrated in terms of Toeplitz and discretized matrices. Our method requires significantly fewer operations than a recent structured eigensolver, by nearly an order of magnitude.

Key words. superfast divide-and-conquer, eigenvalue decomposition, structured perturbation analysis, linear complexity, rank structure, compression

AMS subject classifications. 65F15, 65F30, 15A18, 15A42

1. Introduction.

In this paper, we consider the eigenvalue decomposition of an $n \times n$ Hermitian matrix A:

$$A = Q \Lambda Q^*, \tag{1.1}$$

where A is rank structured, i.e., A has off-diagonal ranks or numerical ranks bounded by r, Q is the eigenmatrix or matrix of eigenvectors, and Λ is a diagonal matrix for the eigenvalues λ_i. Here, r may be a constant, or may even depend on n, e.g., in the form of a low order power of $\log n$. Such matrices A are frequently encountered in structured matrix computations in recent years. Examples include banded matrices, certain discretized kernel functions, Schur complements in the direct factorizations of some discretized PDEs, Toeplitz matrices in Fourier space [13, 33], and some other types of structured matrices (Toeplitz-like, Hankel, and Hankel-like) after transformations of structures via displacement equations [18, 22, 26, 30, 35]. Effective structured representations have been proposed for such problems, such as quasiseparable, sequentially semiseparable, hierarchically semiseparable (HSS), and hierarchical

*Department of Mathematics, Purdue University, West Lafayette, IN 47907, USA (vogel13@purdue.edu, xiaj@math.purdue.edu). The research of Jianlin Xia was supported in part by an NSF CAREER Award DMS-1255416 and an NSF grant DMS-1115572.

†Athinoula A. Martinos Center for Biomedical Imaging, Department of Radiology, Massachusetts General Hospital, Harvard University, Charlestown, MA 02129, USA (stcauley@nmr.mgh.harvard.edu).

‡School of Electrical and Computer Engineering, Purdue University, West Lafayette, IN 47907, USA (ragu@ecn.purdue.edu).

matrices [5, 9, 12, 17, 24, 45]. Here for convenience, assume A is real, and is in an HSS form or can be approximated by one.

Existing work on such structured matrices is usually concerned with the fast solutions of linear systems. In particular, many fast direct solvers have been developed. For eigenvalue problems, most work focuses on iterative solutions. Structured QR iterations for special matrices such as companion forms are studied by many researchers. For symmetric HSS forms, fast iterative methods based on bisection have been designed recently [3, 41]. They cost about $O(n^2)$ flops to find all the eigenvalues. Additional costs are needed to extract the eigenvectors. Moreover, it is not clear previously how the HSS approximation affects the accuracy of the eigenvalues.

Here, we focus on the direct eigendecomposition of symmetric HSS matrices using the divide-and-conquer (DC) method. We also study the impact of the off-diagonal compression on the accuracy of the eigenvalues when a matrix is approximated by an HSS form.

1.1. Brief review of the tridiagonal DC method. In dense symmetric eigenvalue solutions, a typical approach is to first reduce a matrix to a tridiagonal form through orthogonal transformations. DC is a very efficient scheme for finding all the eigenvalues of a symmetric tridiagonal form. The scheme is first introduced by Cuppen in [14], built upon several previous results for a rank-one update to the symmetric eigenvalue problem [6, 19, 39]. The basic idea is to recursively divide a tridiagonal matrix into a block diagonal form plus a rank-1 update as follows:

$$
\begin{aligned}
T = \begin{pmatrix} a_1 & b_1 & & \\ b_1 & \ddots & \ddots & \\ & \ddots & a_{n-1} & b_{n-1} \\ & & b_{n-1} & a_n \end{pmatrix} &= \left(\begin{array}{ccc:ccc} & & & & & \\ & \tilde{T}_1 & & & & \\ & & & & & \\ \hdashline & & & & & \\ & & & & \tilde{T}_2 & \\ & & & & & \end{array}\right) + \left(\begin{array}{ccc:ccc} & & & & & \\ & & 0 & & & \\ & & & \beta & \beta & \\ \hdashline & & \beta & & & \\ & & \beta & & & \\ & & & & 0 & \end{array}\right) \\
&\equiv \begin{pmatrix} \tilde{T}_1 & \\ & \tilde{T}_2 \end{pmatrix} + \beta z z^T.
\end{aligned}
$$

Suppose $\tilde{T}_j = \tilde{Q}_j \tilde{\Lambda}_j \tilde{Q}_j^T$ is the eigendecomposition of $\tilde{T}_j$ for $j = 1, 2$, as obtained by recursion. Then

$$
\begin{aligned}
T &= \begin{pmatrix} \tilde{Q}_1 & \\ & \tilde{Q}_2 \end{pmatrix} \begin{pmatrix} \tilde{\Lambda}_1 & \\ & \tilde{\Lambda}_2 \end{pmatrix} \begin{pmatrix} \tilde{Q}_1^T & \\ & \tilde{Q}_2^T \end{pmatrix} + \beta z z^T \\
&= \operatorname{diag}(\tilde{Q}_1, \tilde{Q}_2)(\tilde{\Lambda} + \beta v v^T) \operatorname{diag}(\tilde{Q}_1^T, \tilde{Q}_2^T),
\end{aligned}
$$

where $\tilde{\Lambda} = \operatorname{diag}(\tilde{\Lambda}_1, \tilde{\Lambda}_2)$ denotes a block diagonal matrix with diagonal blocks $\tilde{\Lambda}_1, \tilde{\Lambda}_2$ and with diagonal entries $\tilde{\lambda}_i,\ i = 1, \ldots, n$, and $v = \operatorname{diag}(\tilde{Q}_1^T, \tilde{Q}_2^T) z$.

It is known that solving for all the eigenvalues λ_i of $\tilde{\Lambda} + \beta v v^T$ is equivalent to finding all the zeros of the secular equation $f(\lambda) = 1 - \beta \sum_{j=1}^{n} \frac{v_j^2}{\tilde{\lambda}_j - \lambda} = 0$ [19]. The eigenvectors q_i of $\tilde{\Lambda} + \beta v v^T$ also take a simple form: $q_i = (\tilde{\Lambda} - \lambda_i I)^{-1} v$. Computing all the eigenvectors explicitly is an $O(n^2)$ complexity operation, so the eigenmatrix is seldom formed explicitly in efficient implementations of the DC algorithm.

Cuppen's algorithm is later shown to suffer from instability [37], and while faster than many previous methods, it still has $O(n^2)$ complexity for finding all the eigenvalues and $O(n^3)$ for all the eigenvectors. It is nonetheless of great theoretical and historical importance. A more efficient and stable DC scheme is proposed by Gu

and Eisenstat in [23]. They resolve the stability issue by solving for the eigenvectors of a perturbed eigenvalue problem. The overall complexity for finding all the eigenvalues and eigenvectors is reduced to $O(n^2)$, with the potential to be accelerated to $O(n\log^p n)$ by the fast multipole method (FMM) [8, 21], where p is a small integer. It should be noted that the algorithm in [4] can already extract all the eigenvalues of T in nearly linear complexity and optimal parallel performance without using FMM.

More recently, Gu and Eisenstat's algorithm is extended to symmetric block-diagonal plus semiseparable matrices (with off-diagonal rank 1) in [10]. For this case, the efficiency relies entirely on rank structures instead of sparsity patterns. The method also costs $O(n^2)$ for finding all the eigenvalues and eigenvectors, similarly with the potential to be accelerated by FMM.

1.2. Main contributions. This work focuses on two major aspects:

1. The design of a structured DC algorithm for a more *general class of problems*, i.e., symmetric and possibly dense matrices A with off-diagonal ranks or numerical ranks bounded by r, and the achievement of $O(r^2n\log n) + O(rn\log^2 n)$ complexity for finding all the eigenvalues and all the eigenvectors (in a structured form).
2. The *structured perturbation analysis*, i.e., the study of the approximation accuracy of the eigenvalues when hierarchical rank structures are used to approximate the original matrix, and the justification of the effectiveness and reliability of such structured methods for fast eigenvalue solution.

The first contribution is to generalize the algorithms in [23] and [10] to problems that can be represented or approximated by HSS forms. The matrices in [23] and [10] can be considered as special cases of ours with $r = 1$. To our knowledge, even for such special cases, the FMM acceleration is not actually implemented or verified in [10, 23].

To apply DC to a symmetric HSS matrix A, there are some major differences from the tridiagonal case. For the tridiagonal case, the dividing stage is straightforward due to the sparsity. For the HSS case, which is usually dense, an obvious strategy would result in a block diagonal form plus a rank-$2r$ update. Here instead, we design a scheme to write A as a block diagonal matrix plus a rank-r update. The diagonal blocks are updated recursively. Special care is taken to preserve the HSS structures of the diagonal blocks throughout the recursive division. Strategies for reusing computations are shown. In the conquering stage, a strategy similar to the tridiagonal case is used, but by multiple times. Moreover, we use FMM to accelerate all the major computations. They include the solution of the secular equations for the eigenvalues, the stable computation of the eigenvectors, the normalization of the eigenvectors, and the multiplication of the intermediate eigenmatrices and vectors. Furthermore, deflation is also incorporated into the structured algorithm.

After the DC algorithm, we obtain all the eigenvalues, as well as the eigenmatrix Q in (1.1) represented by a sequence of structured intermediate eigenmatrices. Such eigenmatrices appear as block-diagonal forms with Cauchy-like and/or Householder diagonal blocks. Each intermediate eigenmatrix is thus defined by few vectors that can be conveniently used in a tree scheme to quickly compute the product of Q and vectors. The rank structure of Q is also mentioned. In fact, Q has off-diagonal numerical ranks at most $O(r\log^2 n)$.

The algorithm is applicable to general symmetric HSS problems, and further has $O(r^2n\log n) + O(rn\log^2 n)$ complexity for finding the structured eigendecomposition (1.1). This is significantly lower than those of the hierarchical/HSS eigensolvers in

[3, 41], by almost an order of magnitude. The cost to apply the eigenmatrix Q to a vector is $O(rn \log n)$. The algorithm is thus said to be *superfast*, following the terminology in [15, Section 5.3]. The storage for Q is $O(rn \log n)$.

Our second contribution is to further analyze the structured perturbation or approximation accuracy of the eigenvalues due to off-diagonal compression. We show that the impact of off-diagonal compression on the accuracy of the eigenvalues can be well controlled, even if such compression is hierarchical as needed in HSS and other hierarchical structured methods. The results can be viewed as structured perturbation analysis that extends the traditional studies. The tightness of an HSS approximation error bound is shown. For some cases, we also discuss the potential to accurately compute some eigenvalues even if the off-diagonal approximation accuracy is not very high.

All the analysis confirms that our structured DC method can indeed serve as an effective tool for finding the eigenvalues of problems with small off-diagonal ranks or numerical ranks. Its significance lies in both the efficiency and the reliability. Thus, it is also natural to approximate more general matrices (with high off-diagonal ranks) by low-accuracy HSS forms, so as to use our method to roughly estimate the eigenvalues and their distribution. This is very useful in preconditioning. The method and analysis can also be modified for the computation of SVDs of nonsymmetric HSS matrices.

We show the complexity, storage, and accuracy for some useful applications, including Toeplitz matrices and discretized problems. As compared with the symmetric HSS eigensolver in [41], our DC method needs significantly fewer operations for even relatively small n. In a test for a discretized matrix with $n = 4000$ (Table 5.4), the cost of the new method is already over 23 times lower than that of the eigensolver in [41]. Satisfactory eigenvalue accuracy and eigenvector orthogonality are also achieved. As expected, the accuracy is controlled by the approximation tolerance.

The remaining sections are organized as follows. Section 2 presents the superfast DC method, including the major steps and how they generalize from the tridiagonal case. The algorithm, its complete complexity analysis, and some applications are discussed in Section 3. The approximation error analysis for the eigenvalues due to off-diagonal compression is presented in Section 4, followed by tests for the efficiency and accuracy in Section 5. We give some concluding remarks in Section 6.

Throughout the paper, we use the following notation:

- for a matrix A and two index sets $\mathbf{I}$ and $\mathbf{J}$, we use $A|_{\mathbf{I}\times\mathbf{J}}$ to denote a submatrix of A selected by the row index set $\mathbf{I}$ and the column index set $\mathbf{J}$;
- $\mathrm{diag}(a_i|_{i=1}^n)$ or $\mathrm{diag}(a_i, i = 1, 2, \dots,)$ represents a diagonal matrix with diagonal entries $a_1, a_2, \dots$, and it represents a block diagonal matrix if a_i's are matrices;
- for a postordered full binary tree $\mathcal{T}$, we label its nodes as

$$\mathbf{i} = 1, 2, \dots, \mathbf{k} \equiv \mathrm{root}(\mathcal{T}), \tag{1.2}$$

  where $\mathrm{root}(\mathcal{T})$ represents the root;
- for each node $\mathbf{i} \neq \mathrm{root}(\mathcal{T})$ of $\mathcal{T}$, $\mathrm{par}(\mathbf{i})$ denotes its parent and $\mathrm{sib}(\mathbf{i})$ denotes its sibling.

2. Superfast divide-and-conquer method. In this section, we detail our algorithm for computing the eigendecomposition (1.1). In the algorithm presentation, we suppose A is already in an HSS form. How such an HSS form is obtained will be discussed in Section 3.2, and the impact of the approximation on the accuracy of the

eigenvalues will be shown in Section 4.

Before reviewing the formal definition of an HSS matrix, we give a simple example of a block 4×4 symmetric HSS matrix that corresponds to a tree (called an HSS tree) with 7 nodes (see Figure 2.1):

(2.1)
$$A \equiv D_7 = \begin{pmatrix} D_3 & U_3 B_3 U_6^T \\ U_6 B_3^T U_3^T & D_6 \end{pmatrix}, \quad U_3 = \begin{pmatrix} U_1 R_1 \\ U_2 R_2 \end{pmatrix}, \quad U_6 = \begin{pmatrix} U_4 R_4 \\ U_5 R_5 \end{pmatrix},$$

(2.2)
$$D_3 = \begin{pmatrix} D_1 & U_1 B_1 U_2^T \\ U_2 B_1^T U_1^T & D_2 \end{pmatrix}, \quad D_6 = \begin{pmatrix} D_4 & U_4 B_4 U_5^T \\ U_5 B_4^T U_4^T & D_5 \end{pmatrix}.$$

The matrix is defined via two levels of recursions, matching the two levels of parent-child relationship among the nodes in Figure 2.1. The D matrices are the diagonal blocks, and the U matrices are the off-diagonal basis matrices (where we assume each U has full column rank).

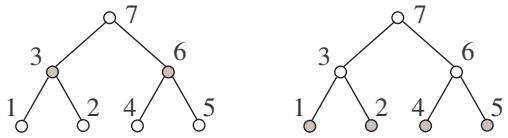


FIG. 2.1. *Two levels of HSS tree nodes (marked in grey) corresponding to (2.1) and (2.2), respectively.*

More generally, an $n \times n$ symmetric HSS matrix A is defined as follows [12, 45]. Let $\mathcal{T}$ be a postordered full binary tree with $\mathbf{k}$ nodes as introduced in (1.2). Each node $\mathbf{i}$ corresponds to a contiguous index set $t_{\mathbf{i}} \subset \{1:n\}$ that satisfies $t_{\mathbf{k}} \equiv \{1:n\}$ and $t_{\mathbf{i}} = t_{\mathbf{c}_1} \cup t_{\mathbf{c}_2}$, $t_{\mathbf{c}_1} \cap t_{\mathbf{c}_2} = \emptyset$ for a nonleaf node $\mathbf{i}$ with children $\mathbf{c}_1$ and $\mathbf{c}_2$ ($\mathbf{c}_1 < \mathbf{c}_2 < \mathbf{i}$). (Figure 2.1 above and Figure 2.2 later can assist in the understanding of this.) The matrix A is in a symmetric HSS form if there exist matrices $D_{\mathbf{i}}$, $U_{\mathbf{i}}$, $R_{\mathbf{i}}$, $B_{\mathbf{i}}$ (called generators) associated with $\mathbf{i}$, such that

(2.3)
$$A|_{t_{\mathbf{i}} \times t_{\mathbf{i}}} \equiv D_{\mathbf{i}} = \begin{pmatrix} D_{\mathbf{c}_1} & U_{\mathbf{c}_1} B_{\mathbf{c}_1} U_{\mathbf{c}_2}^T \\ U_{\mathbf{c}_2} B_{\mathbf{c}_1}^T U_{\mathbf{c}_1}^T & D_{\mathbf{c}_2} \end{pmatrix},$$

(2.4)
$$U_{\mathbf{i}} = \begin{pmatrix} U_{\mathbf{c}_1} & \\ & U_{\mathbf{c}_2} \end{pmatrix} \begin{pmatrix} R_{\mathbf{c}_1} \\ R_{\mathbf{c}_2} \end{pmatrix}.$$

$\mathcal{T}$ is called an HSS tree. Clearly, $U_{\mathbf{i}}$ is a column basis matrix of the off-diagonal block $A|_{t_{\mathbf{i}} \times (t_{\mathbf{k}} \setminus t_{\mathbf{i}})}$. It is usually said to be a nested basis (matrix) due to (2.4).

For notational convenience, we make the following assumptions:

- $\mathbf{c}_1$ and $\mathbf{c}_2$ denote the left and right children of a nonleaf node $\mathbf{i}$ of $\mathcal{T}$, respectively;
- the rank of each off-diagonal block $U_{\mathbf{c}_1} B_{\mathbf{c}_1} U_{\mathbf{c}_2}^T$ is (bounded by) r; more specifically, the order of $B_{\mathbf{c}_1}$ is (bounded by) r;

In our superfast DC method, the HSS matrix is recursively divided and updated. The eigenvalues and eigenvectors are computed thereafter by recursion, with the major computations accelerated by FMM.

2.1. Dividing the HSS matrix. In the "dividing" stage, we recursively write the HSS matrix A as the sum of a block diagonal matrix (with two HSS diagonal blocks) and a rank-r update.

2.1.1. General procedure. Let $\mathbf{i}$ be a nonleaf node of $\mathcal{T}$, and DC is applied to $D_{\mathbf{i}}$. If $\mathbf{i} = \mathbf{k}$, then this is to divide the entire matrix A. It is clear that we can rewrite (2.3) in the following form:

$$D_{\mathbf{i}} = \begin{pmatrix} D_{\mathbf{c}_1} & \\ & D_{\mathbf{c}_2} \end{pmatrix} + \begin{pmatrix} U_{\mathbf{c}_1} & \\ & U_{\mathbf{c}_2} \end{pmatrix} \begin{pmatrix} & B_{\mathbf{c}_1} \\ B_{\mathbf{c}_1}^T & \end{pmatrix} \begin{pmatrix} U_{\mathbf{c}_1}^T & \\ & U_{\mathbf{c}_2}^T \end{pmatrix}.$$

If we further compute an eigendecomposition of $\begin{pmatrix} & B_{\mathbf{c}_1} \\ B_{\mathbf{c}_1}^T & \end{pmatrix}$, this would result in a rank-$2r$ update to $\mathrm{diag}(D_{\mathbf{c}_1}, D_{\mathbf{c}_2})$. However, it turns out that we can write a more compact low-rank update instead:

$$D_{\mathbf{i}} = \mathrm{diag}(\tilde{D}_{\mathbf{c}_1}, \tilde{D}_{\mathbf{c}_2}) + Z_{\mathbf{i}} Z_{\mathbf{i}}^T, \quad \text{with} \tag{2.5}$$

$$\tilde{D}_{\mathbf{c}_1} = D_{\mathbf{c}_1} - U_{\mathbf{c}_1} U_{\mathbf{c}_1}^T, \quad \tilde{D}_{\mathbf{c}_2} = D_{\mathbf{c}_2} - U_{\mathbf{c}_2} B_{\mathbf{c}_1}^T B_{\mathbf{c}_1} U_{\mathbf{c}_2}^T, \quad Z_{\mathbf{i}} = \begin{pmatrix} U_{\mathbf{c}_1} \\ U_{\mathbf{c}_2} B_{\mathbf{c}_1}^T \end{pmatrix}. \tag{2.6}$$

(Here for consistency, $B_{\mathbf{c}_1}^T$ is associated with $U_{\mathbf{c}_2}$ for all such updates.) That is, by modifying the diagonal blocks, we can write $D_{\mathbf{i}}$ as a rank-r update to $\mathrm{diag}(\tilde{D}_{\mathbf{c}_1}, \tilde{D}_{\mathbf{c}_2})$. Note that this is preferable, since the diagonal blocks can be quickly updated with our scheme below. Furthermore, the later conquering stage is usually a much more expensive process, and a rank-$2r$ update would cause its cost to double.

A critical issue is then to *preserve the HSS structure* in the dividing step (2.5)–(2.6), so that the off-diagonal ranks of $\tilde{D}_{\mathbf{c}_1}$ and $\tilde{D}_{\mathbf{c}_2}$ do not increase. In fact, the HSS forms of $\tilde{D}_{\mathbf{c}_1}$ and $\tilde{D}_{\mathbf{c}_2}$ can be quickly updated based on those of $D_{\mathbf{c}_1}$ and $D_{\mathbf{c}_2}$, respectively. This follows from the property of the nested basis $U_{\mathbf{i}}$ as in (2.4). Similar structure updates have been previously exploited in HSS factorization and inversion [44, 45, 46]. Here, we show how to perform the HSS update (2.6) in a more intuitive way as follows.

LEMMA 2.1. *For the nested basis $U_{\mathbf{i}}$ associated with node $\mathbf{i}$, let H be a square matrix with size equal to the column size of $U_{\mathbf{i}}$, and $\mathbf{i}_1$ be the smallest descendant of $\mathbf{i}$. Then $U_{\mathbf{i}} H U_{\mathbf{i}}^T$ is an HSS matrix, with the HSS generators $\hat{D}_{\mathbf{j}}, \hat{U}_{\mathbf{j}}, \hat{R}_{\mathbf{j}}, \hat{B}_{\mathbf{j}}$ for $\mathbf{j} = \mathbf{i}_1, \mathbf{i}_1 + 1, \ldots, \mathbf{i} - 1$:*

$$\hat{U}_{\mathbf{j}} = U_{\mathbf{j}}, \quad \hat{R}_{\mathbf{j}} = R_{\mathbf{j}}, \tag{2.7}$$

$$\hat{B}_{\mathbf{j}} = R_{\mathbf{j}} (R_{\mathbf{j}_l} \cdots R_{\mathbf{j}_1}) H (R_{\mathbf{j}_l} \cdots R_{\mathbf{j}_1})^T R_{\mathrm{sib}(\mathbf{j})}^T, \tag{2.8}$$

$$\hat{D}_{\mathbf{j}} = U_{\mathbf{j}} R_{\mathbf{j}} (R_{\mathbf{j}_l} \cdots R_{\mathbf{j}_1}) H (R_{\mathbf{j}_l} \cdots R_{\mathbf{j}_1})^T R_{\mathbf{j}}^T U_{\mathbf{j}}^T, \quad (\mathbf{j}\text{: leaf}), \tag{2.9}$$

where $\mathbf{j} \to \mathbf{j}_l \to \cdots \to \mathbf{j}_1 \to \mathbf{i}$ is the path connecting the node $\mathbf{j}$ to $\mathbf{i}$.

Proof. (2.7) is obvious. We use induction to show (2.8)–(2.9). That is, let $\mathcal{T}_{\mathbf{i}}$ be the subtree of $\mathcal{T}$ with root $\mathbf{i}$. The induction is done on the number of levels of $\mathcal{T}_{\mathbf{i}}$. For convenience, the nodes are illustrated in Figure 2.2.

If $\mathcal{T}_{\mathbf{i}}$ has two levels, from (2.4), we have

$$U_{\mathbf{i}} H U_{\mathbf{i}}^T = \begin{pmatrix} U_{\mathbf{c}_1} (R_{\mathbf{c}_1} H R_{\mathbf{c}_1}^T) U_{\mathbf{c}_1}^T & U_{\mathbf{c}_1} (R_{\mathbf{c}_1} H R_{\mathbf{c}_2}^T) U_{\mathbf{c}_2}^T \\ U_{\mathbf{c}_2} (R_{\mathbf{c}_2} H R_{\mathbf{c}_1}^T) U_{\mathbf{c}_1}^T & U_{\mathbf{c}_2} (R_{\mathbf{c}_2} H R_{\mathbf{c}_2}^T) U_{\mathbf{c}_2}^T \end{pmatrix}. \tag{2.10}$$

Then

$$\hat{B}_{\mathbf{c}_1} = R_{\mathbf{c}_1} H R_{\mathbf{c}_2}^T, \quad \hat{D}_{\mathbf{c}_1} = U_{\mathbf{c}_1} (R_{\mathbf{c}_1} H R_{\mathbf{c}_1}^T) U_{\mathbf{c}_1}^T, \quad \hat{D}_{\mathbf{c}_2} = U_{\mathbf{c}_2} (R_{\mathbf{c}_2} H R_{\mathbf{c}_2}^T) U_{\mathbf{c}_2}^T.$$

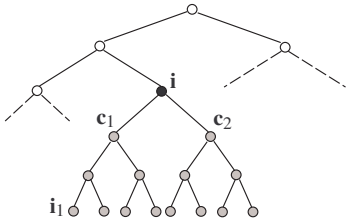


FIG. 2.2. *Node* $\mathbf{i}$ *in the HSS tree* $\mathcal{T}$ *and its descendants* $\mathbf{j}$ *(marked in gray), whose associated HSS generators need to be updated.*

The results follow immediately.

Assume the results are true for $\mathcal{T}_{\mathbf{i}}$ with $2, 3, \ldots, l-1$ levels. We show they are also true for $\mathcal{T}_{\mathbf{i}}$ with l levels. In fact, we still have (2.10), and $\mathcal{T}_{\mathbf{c}_1}$ and $\mathcal{T}_{\mathbf{c}_2}$ has $l-1$ levels. By induction, $U_{\mathbf{c}_1}(R_{\mathbf{c}_1} H R_{\mathbf{c}_1}^T) U_{\mathbf{c}_1}^T$ is an HSS matrix with generators $D_{\mathbf{j}}, U_{\mathbf{j}}, \hat{R}_{\mathbf{j}}, \hat{B}_{\mathbf{j}}$ for $\mathbf{j} = \mathbf{i}_1, \mathbf{i}_1 + 1, \ldots, \mathbf{c}_1 - 1$, where

$$\begin{aligned}\hat{B}_{\mathbf{j}} &= R_{\mathbf{j}}(R_{\mathbf{j}_l} \cdots R_{\mathbf{j}_2})(R_{\mathbf{c}_1} H R_{\mathbf{c}_1}^T)(R_{\mathbf{j}_l} \cdots R_{\mathbf{j}_2})^T R_{\mathrm{sib}(\mathbf{j})}^T \\ &= R_{\mathbf{j}}(R_{\mathbf{j}_l} \cdots R_{\mathbf{j}_2} R_{\mathbf{c}_1}) H (R_{\mathbf{j}_l} \cdots R_{\mathbf{j}_2} R_{\mathbf{c}_1})^T R_{\mathrm{sib}(\mathbf{j})}^T.\end{aligned}$$

This gives (2.8), since $\mathbf{c}_1 = \mathbf{j}_1$ is the child of $\mathbf{i}$ that is in the path from $\mathbf{j}$ to $\mathbf{i}$. Similarly, we get (2.9).

Analogously, applying induction to $U_{\mathbf{c}_2}(R_{\mathbf{c}_2} H R_{\mathbf{c}_2}^T) U_{\mathbf{c}_2}^T$ yields (2.8)–(2.9) for $\mathbf{j} = \mathbf{c}_1 + 1, \mathbf{c}_1 + 2, \ldots, \mathbf{c}_2 - 1$. For the node $\mathbf{j} = \mathbf{c}_1$, (2.8) obviously holds since $\hat{B}_{\mathbf{c}_1} = R_{\mathbf{c}_1} H R_{\mathbf{c}_2}^T$. To summarize, the results hold for all $\mathbf{j} = \mathbf{i}_1, \mathbf{i}_1 + 1, \ldots, \mathbf{i} - 1$. ∎

Thus, by setting $\mathbf{i} \equiv \mathbf{k}$ in Lemma 2.1, we can see that $U_{\mathbf{k}} H U_{\mathbf{k}}^T$ and A have the same U, R generators or are said to share common nested off-diagonal bases. For such matrices, it is convenient to verify the following result.

LEMMA 2.2. *Assume two conformably partitioned symmetric HSS matrices* A *and* C *have the same* U, R *generators, and the off-diagonal ranks of* A *and* C *are bounded by* r*. Then* $A \pm C$ *can be written as an HSS form with the same* U, R *generators as those of* A *and* C*, and with the* D *and* B *generators respectively added or subtracted. Moreover, the off-diagonal ranks of* $A \pm C$ *are bounded by* r*.*

Combining the results in the two lemmas, we have the following theorem for the fast HSS update in the dividing stage.

THEOREM 2.3. *Use the same notation as in Lemmas 2.1 and 2.2 and set* $\mathbf{i} = \mathbf{k}$*. The matrix* $A - U_{\mathbf{k}} H U_{\mathbf{k}}^T$ *has the same* U, R *generators as* A*, and its* D, B *generators can be obtained via the following updates:*

$$B_{\mathbf{j}} \leftarrow B_{\mathbf{j}} - R_{\mathbf{j}}(R_{\mathbf{j}_l} \cdots R_{\mathbf{j}_1}) H (R_{\mathbf{j}_l} \cdots R_{\mathbf{j}_1})^T R_{\mathrm{sib}(\mathbf{j})}^T, \tag{2.11}$$

$$D_{\mathbf{j}} \leftarrow D_{\mathbf{j}} - U_{\mathbf{j}} R_{\mathbf{j}}(R_{\mathbf{j}_l} \cdots R_{\mathbf{j}_1}) H (R_{\mathbf{j}_l} \cdots R_{\mathbf{j}_1})^T R_{\mathbf{j}}^T U_{\mathbf{j}}^T, \quad (\mathbf{j}: \textit{ leaf}), \tag{2.12}$$

and the off-diagonal ranks of $A - U_{\mathbf{k}} H U_{\mathbf{k}}^T$ *are bounded by* r*.*

This process involves the updates of the generators associated with all the descendants $\mathbf{j}$ of $\mathbf{i}$. In the dividing process, H is determined based on whether the above process is applied to the left or the right child branch in (2.6). Setting $\mathbf{i}$ to be $\mathbf{c}_1$ and H to be I gives the HSS structure of $\tilde{D}_{\mathbf{c}_1}$. Setting $\mathbf{i}$ to be $\mathbf{c}_2$ and H to be $B_{\mathbf{c}_1}^T B_{\mathbf{c}_1}$ gives the HSS structure of $\tilde{D}_{\mathbf{c}_2}$. The dividing procedure can then be recursively applied to $\tilde{D}_{\mathbf{c}_1}$ and $\tilde{D}_{\mathbf{c}_2}$. Theorem 2.3 guarantees that the HSS structures are preserved throughout the recursive dividing procedure.

2.1.2. An example. As a simple example, consider the block 4×4 symmetric matrix given in (2.1) and (2.2). At the first level, the dividing scheme (2.5) works as

$$D_7 = \mathrm{diag}(\tilde{D}_3, \tilde{D}_6) + Z_7 Z_7^T,$$

where $Z_7 = \begin{pmatrix} U_3 \\ U_6 B_3^T \end{pmatrix}$. The generators of A are updated as follows to get those of $\tilde{D}_3$ and $\tilde{D}_6$:

- $B_1 \leftarrow B_1 - R_1 R_2^T$,
- $B_4 \leftarrow B_4 - R_4 B_3^T B_3 R_5^T$.
- $D_1 \leftarrow D_1 - U_1 R_1 R_1^T U_1^T$,
- $D_2 \leftarrow D_2 - U_2 R_2 R_2^T U_2^T$,
- $D_4 \leftarrow D_4 - U_4 R_4 B_3^T B_3 R_4^T U_4^T$,
- $D_5 \leftarrow D_5 - U_5 R_5 B_3^T B_3 R_5^T U_5^T$.

At the second level, the two subproblems $\tilde{D}_3$ and $\tilde{D}_6$ are further divided via the following updates to the generators:

- $D_1 \leftarrow D_1 - U_1 U_1^T$,
- $D_2 \leftarrow D_2 - U_2 B_1^T B_1 U_2^T$,
- $D_4 \leftarrow D_4 - U_4 U_4^T$,
- $D_5 \leftarrow D_5 - U_5 B_4^T B_4 U_5^T$.

2.1.3. Reusing computations. In general, to divide $D_{\mathbf{i}}$ as in (2.5), we update all the B generators associated with the left nodes in $\mathcal{T}_{\mathbf{i}}$, and the D generators associated with the leaves. The update of the B, D generators can follow a top-down sweep, so as to reuse some computations. For example, once $B_{\mathbf{j}}$ for a nonleaf node $\mathbf{j}$ has been updated as in (2.11), then the update of $B_{\mathbf{c}}$ for a child $\mathbf{c}$ of $\mathbf{j}$ looks like

$$B_{\mathbf{c}} \leftarrow B_{\mathbf{c}} - R_{\mathbf{c}} R_{\mathbf{j}} (R_{\mathbf{j}_l} \cdots R_{\mathbf{j}_1}) H (R_{\mathbf{j}_l} \cdots R_{\mathbf{j}_1})^T R_{\mathbf{j}}^T R_{\mathrm{sib}(\mathbf{c})}^T,$$

where $R_{\mathbf{j}_l} \cdots R_{\mathbf{j}_1}$ has already been computed in (2.11). This thus can be performed recursively as follows. Initially for node $\mathbf{i}$, let

$$S_{\mathbf{i}} = I.$$

Then for $\mathbf{j}$, the update of $B_{\mathbf{j}}$ in (2.11) becomes

$$B_{\mathbf{j}} \leftarrow B_{\mathbf{j}} - R_{\mathbf{j}} S_{\mathrm{par}(\mathbf{j})} S_{\mathrm{par}(\mathbf{j})}^T R_{\mathrm{sib}(\mathbf{j})}^T.$$

Then let

$$S_{\mathbf{j}} = R_{\mathbf{j}} S_{\mathrm{par}(\mathbf{j})},$$

which is used for later updates. After all the B generators are updated, $S_{\mathbf{j}} = R_{\mathbf{j}} R_{\mathbf{j}_l} \cdots R_{\mathbf{j}_1}$ is already available. We further compute $U_{\mathbf{j}} S_{\mathbf{j}}$ and use it in (2.12) to update $D_{\mathbf{j}}$ as:

$$S_{\mathbf{j}} \leftarrow U_{\mathbf{j}} S_{\mathbf{j}}, \quad D_{\mathbf{j}} \leftarrow D_{\mathbf{j}} - S_{\mathbf{j}} S_{\mathbf{j}}^T, \quad (\mathbf{j}\text{: leaf}).$$

In addition, further computational savings are possible. Clearly, the D, B generators may need to be updated multiple times, depending on the number of ancestor nodes. As an improvement, we may accumulate the updates so as to save the intermediate multiplication costs for forming the updates. In practice, this may be skipped to simplify the implementation, since the cost in the subsequent conquering stage usually dominates the total cost (especially when r is very small).

2.2. Computing the HSS eigendecomposition. In the "conquering" stage, we compute the eigendecomposition of A from those of the subproblems. The rank-r update in (2.5) is split into r rank-1 updates. We start with the following case with a single rank-1 update:

$$\operatorname{diag}(\tilde{D}_{\mathbf{c}_1}, \tilde{D}_{\mathbf{c}_2}) + zz^T. \tag{2.13}$$

Just like in the standard DC (Section 1.1), suppose we have computed the eigendecompositions $\tilde{D}_{\mathbf{c}_1} = \tilde{Q}_{\mathbf{c}_1}\tilde{\Lambda}_{\mathbf{c}_1}\tilde{Q}_{\mathbf{c}_1}^T$, $\tilde{D}_{\mathbf{c}_2} = \tilde{Q}_{\mathbf{c}_2}\tilde{\Lambda}_{\mathbf{c}_2}\tilde{Q}_{\mathbf{c}_2}^T$. Then

$$\operatorname{diag}(\tilde{D}_{\mathbf{c}_1}, \tilde{D}_{\mathbf{c}_2}) + zz^T = \operatorname{diag}(\tilde{Q}_{\mathbf{c}_1}, \tilde{Q}_{\mathbf{c}_2})(\tilde{\Lambda} + vv^T)\operatorname{diag}(\tilde{Q}_{\mathbf{c}_1}^T, \tilde{Q}_{\mathbf{c}_2}^T), \tag{2.14}$$

where

$$\tilde{\Lambda} = \operatorname{diag}(\tilde{\Lambda}_{\mathbf{c}_1}, \tilde{\Lambda}_{\mathbf{c}_2}) \equiv \operatorname{diag}(\tilde{\lambda}_j, j = 1, 2, \ldots), \quad v = \operatorname{diag}(\tilde{Q}_{\mathbf{c}_1}^T, \tilde{Q}_{\mathbf{c}_2}^T)z. \tag{2.15}$$

Here, we can assume all $\tilde{\lambda}_j$'s are distinct, and v has no zero entry. Otherwise, the deflation strategy in Section 2.2.6 is applied. In the following, we discuss how to quickly find the eigendecomposition of (2.13) with the aid of FMM.

REMARK 2.1. We keep each part of this subsection compact, since much of the technical details can be generalized from [10, 23] (although the actual algorithm design and implementation are far less trivial). We include only essential descriptions to introduce necessary notation and to sketch the basic ideas. Some pseudocodes will be included to assist in the understanding.

2.2.1. FMM in one dimension. FMM in one dimension will be used at multiple places in our algorithm. Here, we only briefly mention its basic idea. The reader is referred to [2, 7, 8, 21] for more details.

Suppose we wish to evaluate the following function at multiple points λ:

$$\Phi(\lambda) = \sum_{j=1}^{N} \alpha_j \phi(\lambda - \tilde{\lambda}_j), \tag{2.16}$$

where $\{\tilde{\lambda}_j\}_{j=1}^N$ are given real points, $\{\alpha_j\}_{j=1}^N$ are constants, and $\phi(x)$ is a specific kernel function of interest. In our case, $\phi(x)$ is either $1/x$, $\log(x)$, or $1/x^2$. FMM is designed to quickly evaluate $\Phi(\lambda)$ at M points $\{\lambda_i\}_{i=1}^M$ without using the dense matrix-vector multiplication $K\alpha$, where $K = (\phi(\lambda_i - \tilde{\lambda}_j))_{M\times N}$.

The FMM implementation we use is based on [7], where explicit accuracy and stability estimates are given. We briefly describe the results here. Suppose (a, b) and (c, d) are two well-separated intervals and $\lambda_i \in (a, b), i = 1, \ldots, M$, $\tilde{\lambda}_j \in (c, d), j = 1, \ldots, N$. Compute a truncated Taylor series expansion of ϕ:

$$\phi(\lambda - \tilde{\lambda}) \approx \sum_{k=1}^{p} f_k(\lambda) g_k(\tilde{\lambda}), \tag{2.17}$$

where a proper scaling is applied to f_k and g_k. The relative approximation error is [7, Section 2.1]

$$\epsilon = \frac{1+\eta}{1-\eta}\eta^p, \tag{2.18}$$

where $\eta \in (0,1)$ depends on the separation between (a,b) and (c,d). Thus, p only needs to be $O(\log \tau)$ to reach a desired accuracy τ. Then, (2.17) enables us to write a low-rank approximation

$$K = (\phi(\lambda_i - \tilde{\lambda}_j))_{M\times N} \approx \underset{M\times p}{\hat{U}} \cdot \underset{p\times p}{\hat{C}} \cdot \underset{p\times N}{\hat{V}^T},$$

where the elementwise relative approximation error is (2.18). Furthermore, the proper scaling of the Taylor series expansion guarantees that the entries of $\hat{U}$ and $\hat{V}$ have magnitudes bounded by 1, and the entries of $\hat{C}$ have magnitudes roughly proportional to those of K [7, Section 6.2]. This enables us to stably evaluate $K\alpha$ to a desired accuracy with complexity $O(M+N)$ instead of $O(MN)$.

When λ_i and $\tilde{\lambda}_j$ come from the same set of points, then the process is done hierarchically as in the standard FMM so as to reach the overall linear complexity. In our implementation, we make the separation parameter $\eta \le \frac{2}{3}$ and the accuracy τ to be around 10^{-10} or even smaller.

Note that when FMM is used in our DC algorithm, it implicitly approximates the intermediate matrices. For example, an elementwise relative error ϵ is introduced into the intermediate eigenmatrices. Such an error may be propagated to later computations. Due to the hierarchical DC scheme, it is expected that the error may be magnified by only up to about $\log n$ times, similar to the approximation error results in [1, 20, 40]. Such error propagations are thus well controlled, and in practice, the accuracy of the eigenvalues is consistent with the tolerance (see Section 5). We can similarly understand the behaviors of the numerical errors in the FMM matrix-vector multiplication, just like the stability analysis for a hierarchical matrix factorization in [40].

2.2.2. Computing the eigenvalues by solving the secular equation. As in the tridiagonal DC scheme, the eigenvalues λ are the roots of the secular equation

$$f(\lambda) = 1 + \sum_{j=1}^{n} \frac{v_j^2}{\tilde{\lambda}_j - \lambda} = 0, \tag{2.19}$$

which can be solved with Newton's method. To ensure the quick and stable solution of (2.19), we follow the modified Newton's method in [15], which is based on the Middle Way in [32]. This modified Newton's method involves the evaluation of functions of the forms $\varphi(\lambda) = \sum_{j=1}^{n} \frac{v_j^2}{\tilde{\lambda}_j - \lambda}$ and $\varphi'(\lambda)$ for multiple λ. This can be accelerated by FMM with $\phi(x) = 1/x$ or $1/x^2$ in (2.16).

Just as mentioned in [15], two or three Newton iterations are sufficient to reach the machine precision. This strategy works for all the roots of the secular equation except the largest one, for which we follow [23] and use basic rational interpolation with several safeguards for stability based on the algorithm in [6].

2.2.3. Computing the eigenvectors stably. As has been extensively studied [14, 16, 37], the computation of the eigenvectors via the simple formula $q_j = (\tilde{\Lambda} - \lambda_j I)^{-1} v$ can have stability issues. In particular, if $|\lambda_i - \lambda_j|$ is small for two eigenvalues λ_i and λ_j, the corresponding eigenvectors q_i and q_j may be far from orthogonal [16]. A stable computational strategy [23] is to solve for the eigenvectors of a slightly perturbed problem $\tilde{\Lambda} + \hat{v}\hat{v}^T$, which has the exact eigenvalues λ_j. The vector $\hat{v} =$

$(\hat{v}_i)_{k=1}^n$ is computed based on Löwner's formula [15]:

$$\hat{v}_i = \sqrt{\frac{\prod_{j=1}^{i-1}(\tilde{\lambda}_i - \lambda_j)\prod_{j=i}^{n}(\lambda_j - \tilde{\lambda}_i)}{\prod_{j=1}^{i-1}(\tilde{\lambda}_i - \tilde{\lambda}_j)\prod_{j=i+1}^{n}(\tilde{\lambda}_j - \tilde{\lambda}_i)}}, \tag{2.20}$$

where the eigenvalues are ordered from the largest to the smallest. The vector $\hat{v}$ can be quickly evaluated with FMM applied to $\log \hat{v}_i$ [23]. That is, set $\phi(x) = \log(x)$ in (2.16).

The eigenvectors associated with all the eigenvalues λ_j can be assembled into a matrix $\hat{Q} \equiv \left(\frac{\hat{v}_i}{\tilde{\lambda}_i - \lambda_j}\right)_{i,j}$. While we do not explicitly form this matrix, we still need to normalize its columns to obtain orthonormal eigenvectors and to ensure the stability of later calculations. Let s_j be the inverse of the norm of column j of $\hat{Q}$. It is used to scale that column as in

$$Q_{\mathbf{i}}^{(1)} = \left(\frac{\hat{v}_i s_j}{\tilde{\lambda}_i - \lambda_j}\right)_{i,j}, \quad \text{with} \quad s_j = \left(\sum_{i=1}^{n} \frac{v_i^2}{(\tilde{\lambda}_i - \lambda_j)^2}\right)^{-1/2}, \tag{2.21}$$

where the superscript in $Q_{\mathbf{i}}^{(1)}$ is used to indicate that the result is from a single rank-1 update (2.13). Once again, the computation of s_j can be accelerated by FMM, with $\phi(x) = 1/x^2$ in (2.16). Note that $\hat{Q}$ is now converted into the orthogonal Cauchy-like matrix $Q_{\mathbf{i}}^{(1)}$ (a Cauchy-like matrix is a matrix whose (i,j) entry looks like $\frac{\alpha_i \beta_j}{d_i - f_j}$ for four vectors α, β, d, f).

2.2.4. Rank-r updated eigendecomposition. The above process needs to be repeated r times for the rank-r update in (2.5). We summarize the process in the following lemma and skip the details.

LEMMA 2.4. *Suppose $\tilde{D}_{\mathbf{c}_1} = \tilde{Q}_{\mathbf{c}_1}\tilde{\Lambda}_{\mathbf{c}_1}\tilde{Q}_{\mathbf{c}_1}^T$ and $\tilde{D}_{\mathbf{c}_2} = \tilde{Q}_{\mathbf{c}_2}\tilde{\Lambda}_{\mathbf{c}_2}\tilde{Q}_{\mathbf{c}_2}^T$ are the eigendecompositions of $\tilde{D}_{\mathbf{c}_1}$ and $\tilde{D}_{\mathbf{c}_2}$ in (2.5), respectively. Let*

$$Z_{\mathbf{i}} = (z^{(1)}, \ldots, z^{(r)}), \quad Q^{(0)} = \operatorname{diag}(\tilde{Q}_{\mathbf{c}_1}, \tilde{Q}_{\mathbf{c}_2}), \quad v^{(0)} = (Q^{(0)})^T z, \quad \lambda_j^{(0)} = \tilde{\lambda}_j.$$

Suppose the eigendecomposition of $\operatorname{diag}(\lambda_j^{(i-1)}|_{j=1}^n) + v^{(i)}(v^{(i)})^T$ *is*

$$\operatorname{diag}(\lambda_j^{(i-1)}|_{j=1}^n) + v^{(i)}(v^{(i)})^T = Q_{\mathbf{i}}^{(i)} \operatorname{diag}(\lambda_j^{(i)}|_{j=1}^n)(Q_{\mathbf{i}}^{(i)})^T,$$

where $Q_{\mathbf{i}}^{(i)}$ is in a Cauchy-like form and $v^{(i)} = (Q_{\mathbf{i}}^{(i-1)})^T z^{(i)}$. Then the eigendecomposition of $D_{\mathbf{i}}$ in (2.5) is

$$D_{\mathbf{i}} = (Q_{\mathbf{i}}^{(0)} Q_{\mathbf{i}}) \operatorname{diag}(\lambda_j^{(r)}|_{j=1}^n)(Q_{\mathbf{i}}^{(0)} Q_{\mathbf{i}})^T,$$

where

$$Q_{\mathbf{i}} = Q_{\mathbf{i}}^{(1)} \cdots Q_{\mathbf{i}}^{(r)}. \tag{2.22}$$

That is, $\lambda_j^{(r)}|_{j=1}^n$ are the eigenvalues of $D_{\mathbf{i}}$ in (2.5) and $Q_{\mathbf{i}}^{(0)} Q_{\mathbf{i}}$ is the eigenmatrix of $D_{\mathbf{i}}$. For completeness, if $\mathbf{i}$ is a leaf node of the HSS tree, we set $Q_{\mathbf{i}}^{(0)} = I$ and compute $Q_{\mathbf{i}}$ directly via the eigendecomposition of the diagonal block $D_{\mathbf{i}}$.

2.2.5. Application of the eigenmatrix to vectors and structure of the eigenmatrix. Note that we do not form the eigenmatrix $Q_{\mathbf{i}}^{(0)}(Q_{\mathbf{i}}^{(1)}\cdots Q_{\mathbf{i}}^{(r)})$ of $D_{\mathbf{i}}$ or the eigenmatrix Q of A explicitly. In practical applications, the eigenvectors of A are often used under the following circumstance: applications of the eigenmatrix or its transpose to vectors. In fact, such a process is already needed in the DC process for computing v in (2.15). Thus, we illustrate this as part of the eigendecomposition.

For an individual matrix $Q_{\mathbf{i}}^{(1)}$ of the form (2.21), to multiply $(Q_{\mathbf{i}}^{(1)})^T$ and a vector z, we have

$$\left((Q_{\mathbf{i}}^{(1)})^T z\right)_j = s_j \sum_{i=1}^{n} \frac{\hat{v}_i z_i}{\tilde{\lambda}_i - \lambda_j}. \tag{2.23}$$

Similarly to [10, 23], this can be accelerated by FMM with $\phi(x) = 1/x$ in (2.16). To apply $Q_{\mathbf{i}}$ to a vector, we just need to repeat this r times.

The overall strategy for applying Q or Q^T to a vector z is basically the one in [10, 15]. For our case, this can be done with the aid of the HSS tree $\mathcal{T}$. More specifically, associate $Q_{\mathbf{i}}$ in (2.22) with each node $\mathbf{i}$ of $\mathcal{T}$. Then we use a multilevel procedure to compute the eigenmatrix-vector product.

For convenience, Algorithm 1 shows how to apply Q^T to z, as needed in forming v in (2.15). The multiplication of Q and z can be performed similarly, and can be used if we need to extract any specific column of Q.

Algorithm 1 Application of Q^T to a vector, where Q is the eigenmatrix of A

procedure `eigmv`$(Q_{\mathbf{1}}, \ldots, Q_{\mathbf{k}}, z)$ *Output:* $Q^T z$, where Q is represented by $Q_{\mathbf{1}}, \ldots, Q_{\mathbf{k}}$
  Partition z into pieces $z_{\mathbf{i}}$ following the sizes of $D_{\mathbf{i}}$ for all leaves $\mathbf{i}$
  for $\mathbf{i} = 1, \ldots, \mathbf{k}$ **do** ▷ $\mathbf{k}$*: root of* $\mathcal{T}$
    if $\mathbf{i}$ is a nonleaf node **then** ▷ $\mathbf{c}_1, \mathbf{c}_2$*: children of* $\mathbf{i}$
      $z_{\mathbf{i}} \leftarrow \begin{pmatrix} z_{\mathbf{c}_1} \\ z_{\mathbf{c}_2} \end{pmatrix}$
    end if
    for $i = 1, 2, \ldots, r$ **do** ▷ r*: column size of* $Z_{\mathbf{i}}$ *in (2.5)*
      $z_{\mathbf{i}} \leftarrow (Q_{\mathbf{i}}^{(i)})^T z_{\mathbf{i}}$ (fast evaluation via FMM) ▷ *As in (2.23)*
    end for
  end for
  Output $z_{\mathbf{k}}$ ▷ $z_{\mathbf{k}} = Q^T z$
end procedure

Remark 2.2. Clearly, the data-sparse structure of Q defined by $Q_{\mathbf{1}}, \ldots, Q_{\mathbf{k}}$ in their Cauchy-like forms is very useful for the fast application of Q or Q^T to a vector. On the other hand, we may also understand the data sparsity of Q based on its off-diagonal rank structure. It can be shown that the eigenmatrix of $\tilde{\Lambda} + \hat{v}\hat{v}^T$ has off-diagonal numerical ranks at most $O(\log n)$ for a given tolerance. (This is similar to Lemma 3.2 below.) Thus, the off-diagonal numerical ranks of Q are at most $O(r \log^2 n)$. Since this rank structure of Q is not actually used in our algorithms, we omit the details.

2.2.6. Deflation. If the vector v in (2.14) has a zero entry, or if $\tilde{\Lambda}$ has two equal diagonal entries, deflation strategies can be applied. This is already shown in [16, 23].

For example, if $v_j = 0$, then $\tilde{\Lambda} + vv^T$ has an eigenvalue

$$\lambda_j = \tilde{\lambda}_j.$$

If $\tilde{\Lambda}$ has two (or more) identical diagonal entries $\tilde{\lambda}_i = \tilde{\lambda}_j$, then a Householder transformation can be used to zero out v_j so as to convert into the previous case. (In these cases, the eigenmatrix of $\tilde{\Lambda} + vv^T$ is then block diagonal and may involve Cauchy-like or Householder diagonal blocks.) A similarly strategy can be applied if v_j is small or if the difference between $\tilde{\lambda}_i$ and $\tilde{\lambda}_j$ is small, and the detailed perturbation analysis is provided in [16, 23]. This step is standard but important for the efficiency of the algorithm.

3. Algorithm, complexity, and applications. To facilitate understanding of the algorithm, the framework of the algorithm is shown in Table 3.1, with the details in Algorithm 2. Here, it is assumed that the HSS tree $\mathcal{T}$ is a complete binary tree with $l_{\max} + 1$ levels, with the root at level 0 and the leaves at level $l_{\max}$. We do not count cost reductions due to deflation.

TABLE 3.1
Major operations in the superfast DC algorithm, corresponding lines in Algorithm 2, and their complexity (Section 3.1).

Outer-most loop	Inner-most loop	Operation	Lines in Alg. 2	Complexity subtotal
$l = 1 : l_{\max}-1$	Descendants **j** of **i**	Updating $B_{\mathbf{j}}$ generators	6, 13	$\xi_1 = O(r^2 n \log n)$
	Leaf descendants **j** of **i**	Updating $D_{\mathbf{j}}$ generators	8, 15	$\xi_2 = O(r^2 n \log n)$
$l = l_{\max}$	Leaves **i**	Eigendecomposition of $D_{\mathbf{i}}$	20	$\xi_3 = O(r^2 n)$
$l = l_{\max}-1 : 0$	ith rank-1 update ($i = 1, \ldots, r$)	Intermediate eigenmatrix-vector product	25, 26	$\xi_4 = O(rn \log^2 n)$
		Root-finding	29	$\xi_5 = O(rn \log n)$
		Finding perturbed eigenproblem	31	$\xi_6 = O(rn \log n)$
		Normalization	32	$\xi_7 = O(rn \log n)$

The dividing stage involves three nested loops. The outer-most loop is a top-down sweep through the levels l of the HSS tree $\mathcal{T}$, the next loop is through the nodes **i** at a given level l, and the inner-most loop is through each descendent **j** of **i**.

The conquering stage is also done by three nested loops. The outer-most loop is a bottom-up sweep through the levels l of $\mathcal{T}$, the next loop is through the nodes **i** at a given level, and the inner-most loop is through each of the r rank-one updates. At each step, we complete four tasks. The first is to form the vector v in $\tilde{\Lambda} + vv^T$ as in (2.15). The next task is to solve for the eigenvalues λ of $\tilde{\Lambda} + vv^T$ by finding the roots of the secular equation. The third task is to solve the perturbed eigenvalue problem to find a vector $\hat{v}$ such that λ is an exact eigenvalue of $\tilde{\Lambda} + \hat{v}\hat{v}^T$. Finally, find the orthogonal eigenmatrix of $\tilde{\Lambda} + vv^T$. This eigenmatrix has a structured form.

3.1. Complexity. We now derive analytically the complexity of our algorithm. The numerical results in Section 5 give a view of how the algorithm scales in practice. The results in this section and Section 5 agree asymptotically. Table 3.1 has a summary of the complexity of the major computations and introduces notation. As is often done in HSS algorithms [43, 45], we assume that the leaf level D generators have size $2r$, all the R, B generators have size r, and the HSS tree has $l_{\max} \approx \log(\frac{n}{2r})$ levels (not counting the root level).

Algorithm 2 Superfast divide-and-conquer method

procedure `sdc`
Input: HSS generators $D_{\mathbf{j}}, U_{\mathbf{j}}, R_{\mathbf{j}}, B_{\mathbf{j}}$, HSS tree $\mathcal{T}$
Output: Eigenvalues λ, and structured $Q_{\mathbf{i}}$ as in Lemma 2.4
  for level $l = 0, 1, \ldots, l_{\max} - 1$ **do** ▷ *Dividing stage*
    for each node $\mathbf{i}$ at level l **do**
      $\mathbf{i}_1 \leftarrow$ smallest descendent of $\mathbf{c}_1$, $S_{\mathbf{c}_1} \leftarrow I$ ▷ *Role of S: Section 2.1.3*
      for $\mathbf{j} = \mathbf{c}_1 - 1, \mathbf{c}_1 - 2, \ldots, \mathbf{i}_1$ **do** ▷ *Top-down — left child branch of* $\mathbf{i}$
        $B_{\mathbf{j}} \leftarrow B_{\mathbf{j}} - R_{\mathbf{j}} S_{\mathrm{par}(\mathbf{j})} S_{\mathrm{par}(\mathbf{j})}^T R_{\mathrm{sib}(\mathbf{j})}^T$, $S_{\mathbf{j}} \leftarrow R_{\mathbf{j}} S_{\mathrm{par}(\mathbf{j})}$ ▷ *Updating* $B_{\mathbf{j}}$
        if $\mathbf{j}$ is a leaf **then**
          $S_{\mathbf{j}} \leftarrow U_{\mathbf{j}} S_{\mathbf{j}}$, $D_{\mathbf{j}} \leftarrow D_{\mathbf{j}} - S_{\mathbf{j}} S_{\mathbf{j}}^T$ ▷ *Updating* $D_{\mathbf{j}}$
        end if
      end for
      $\mathbf{i}_2 \leftarrow$ smallest descendent of $\mathbf{c}_2$, $S_{\mathbf{c}_2} \leftarrow B_{\mathbf{c}_1}^T$ ▷ *Role of S: Section 2.1.3*
      for $\mathbf{j} = \mathbf{c}_2 - 1, \mathbf{c}_2 - 2, \ldots, \mathbf{i}_2$ **do** ▷ *Top-down — right child branch of* $\mathbf{i}$
        $B_{\mathbf{j}} \leftarrow B_{\mathbf{j}} - R_{\mathbf{j}} S_{\mathrm{par}(\mathbf{j})} S_{\mathrm{par}(\mathbf{j})}^T R_{\mathrm{sib}(\mathbf{j})}^T$, $S_{\mathbf{j}} \leftarrow R_{\mathbf{j}} S_{\mathrm{par}(\mathbf{j})}$ ▷ *Updating* $B_{\mathbf{j}}$
        if $\mathbf{j}$ is a leaf **then**
          $S_{\mathbf{j}} \leftarrow U_{\mathbf{j}} S_{\mathbf{j}}$, $D_{\mathbf{j}} \leftarrow D_{\mathbf{j}} - S_{\mathbf{j}} S_{\mathbf{j}}^T$ ▷ *Updating* $D_{\mathbf{j}}$
        end if
      end for
    end for
  end for
  Compute the eigendecomposition $D_{\mathbf{i}} = Q_{\mathbf{i}} \tilde{\Lambda} Q_{\mathbf{i}}^T$ for each leaf $\mathbf{i}$
  for level $l = l_{\max} - 1, \ldots, 1, 0$ **do** ▷ *Conquering stage*
    for each node $\mathbf{i}$ at level l **do**
      for $i = 1, 2, \ldots, r$ **do** ▷ *r: column size of* $Z_{\mathbf{i}}$ *in (2.5)*
        $z \equiv \begin{pmatrix} z_1 \\ z_2 \end{pmatrix} \leftarrow$ column i of $Z_{\mathbf{i}}$ ▷ *z: partitioned following (2.13)*
        $\mathbf{i}_1 \leftarrow$ smallest descendent of $\mathbf{c}_1$, $v_1 \leftarrow \texttt{eigmv}(Q_{\mathbf{i}_1}, \ldots, Q_{\mathbf{c}_1}, z_1)$
        $\mathbf{i}_2 \leftarrow$ smallest descendent of $\mathbf{c}_2$, $v_2 \leftarrow \texttt{eigmv}(Q_{\mathbf{i}_2}, \ldots, Q_{\mathbf{c}_2}, z_2)$
        $v \leftarrow \begin{pmatrix} v_1 \\ v_2 \end{pmatrix}$
        Deflate if some previous-step eigenvalues $\tilde{\lambda}_j$ are close to each other or if v has small entries ▷ *Deflation as in Section 2.2.6*
        Solve (2.19) for λ by the Middle Way with FMM acceleration
        $\tilde{\Lambda} \leftarrow$ diag(all such λ's) ▷ *Current-step intermediate eigenvalues*
        Compute $\hat{v}$ in (2.20) with FMM acceleration
          ▷ *Such that* λ *is an exact eigenvalue of* $\tilde{\Lambda} + \hat{v}\hat{v}^T$
        Compute s in (2.21) with FMM acceleration
        Determine structured $Q_{\mathbf{i}}^{(i)}$ as in (2.21) or Section 2.2.6
          ▷ $Q_{\mathbf{i}}^{(i)}$ *may be block diagonal due to deflation*
      end for
      Output structured $Q_{\mathbf{i}}$ as in Lemma 2.4 ▷ *Part of structured Q*
    end for
  end for
  Output diag($\tilde{\Lambda}$) ▷ diag($\hat{\Lambda}$) *associated with level* 0 *— eigenvalues of A*
end procedure

During the dividing stage, at each level l of the HSS tree, there are 2^l nodes $\mathbf{i}$. For each $\mathbf{i}$ at level l, we update $B_{\mathbf{j}}$ generators associated with each descendant $\mathbf{j}$ of $\mathbf{i}$. There are $2^{\tilde{l}-l}$ nodes $\mathbf{j}$ at level $\tilde{l} = l+1, \ldots, l_{\max}$. As in Lines 6 and 13 of Algorithm 2, four matrix multiplications and one matrix subtraction is needed for each $\mathbf{j}$. Thus, the total cost to update all the $B_{\mathbf{j}}$ generators is

$$\xi_1 = \sum_{l=1}^{l_{\max}} 2^l \sum_{\tilde{l}=l+1}^{l_{\max}} 2^{\tilde{l}-l} \cdot (4 \cdot 2r^3) \approx 16r^3 \cdot 2^{l_{\max}} l_{\max} = O(r^2 n \log n),$$

where the low order terms are dropped (this is done similarly later).

The update of the $D_{\mathbf{j}}$ generators at level $l_{\max}$ immediately follows the update of all the $B_{\mathbf{j}}$ generators. See Lines 8 and 15 of Algorithm 2. The cost is

$$\xi_2 = \sum_{l=1}^{l_{\max}} 2^l \cdot 2^{l_{\max}-l}(2 \cdot 2r^3) = O(r^2 n \log n).$$

At the leaf level, we compute the eigendecomposition of $D_{\mathbf{i}}$ for each leaf $\mathbf{i}$ (Line 20 of Algorithm 2). The total cost is

$$\xi_3 = \frac{n}{2r}\left(2 \cdot (2r)^3\right) = O(r^2 n).$$

During the conquering stage, for each node $\mathbf{i}$ at each level l of the HSS tree, a sequence of operations are performed to find the eigenvectors.

One operation is to perform the intermediate eigenmatrix-vector multiplication in terms of $Q_{\mathbf{i}_1}, \ldots, Q_{\mathbf{i}}$ associated with $\mathbf{i}$ and its descendants. Each FMM application involved here has linear complexity $O(\frac{n}{2^{\tilde{l}}})$, where $\frac{n}{2^{\tilde{l}}}$ is the size of $Q_{\mathbf{j}}$ for $\mathbf{j}$ at level $\tilde{l} = l, l+1, \ldots, l_{\max}$. Here, $Q_{\mathbf{j}}$ is further given by r Cauchy-like matrices. This cost of the intermediate eigenmatrix-vector multiplication associated with $\mathbf{i}$ is thus

$$\sum_{\tilde{l}=l}^{l_{\max}} 2^{\tilde{l}-l} \cdot r \cdot O(\frac{n}{2^{\tilde{l}}}) = O(r\frac{n}{2^l}(l_{\max} - l)). \tag{3.1}$$

The subtotal for all $\mathbf{i}$ is

$$\xi_4 = \sum_{l=1}^{l_{\max}} 2^l \sum_{\tilde{l}=l}^{l_{\max}} 2^{\tilde{l}-l} \cdot r \cdot O(\frac{n}{2^{\tilde{l}}}) = O(rn \log^2 n).$$

Another operation is to solve r secular equations associated with each node $\mathbf{i}$ (Line 29 of Algorithm 2). The cost with FMM for each secular equation is $O(\frac{n}{2^l})$. Thus, the subtotal is

$$\xi_5 = \sum_{l=0}^{l_{\max}-1} 2^l \cdot r \cdot O(\frac{n}{2^l}) = O(rn \log n).$$

The costs in the other operations (Lines 31 and 32 of Algorithm 2) are similar:

$$\xi_6 = O(rn \log n), \quad \xi_7 = O(rn \log n).$$

To sum up, we obtain the total cost $\xi = \xi_1 + \cdots + \xi_7$ for our DC algorithm. Clearly, if r is bounded, the cost ξ_4 for applying the intermediate eigenmatrices to vectors

dominates the complexity. In general, the conquering stage costs more than the dividing stage. In addition, by setting $l = 0$ in (3.1), we get the cost $\tilde{\xi}$ for applying Q^T to a vector. The storage for the $Q_{\mathbf{i}}$ matrices in terms of the Cauchy-like/Householder forms can be easily counted. These results are summarized as follows, where the rank structures of banded matrices and Toeplitz matrices are given in the next subsection. The costs are nearly linear in n, so our DC algorithm is said to be superfast [15, Section 5.3].

THEOREM 3.1. *The superfast DC scheme costs ξ flops to find the eigendecomposition (1.1) and $\tilde{\xi}$ flops to apply Q^T to a vector, and the storage for the structured eigenmatrix is σ, where*

$$\xi = O(r^2 n \log n) + O(rn \log^2 n), \quad \tilde{\xi} = O(rn \log n), \quad \sigma = O(rn \log n).$$

Specifically, if A is a banded symmetric matrix with finite bandwidth,

$$\xi = O(n \log^2 n), \quad \tilde{\xi} = O(n \log n), \quad \sigma = O(n \log n),$$

and if A is a symmetric Toeplitz matrix,

$$\xi = O(n \log^3 n), \quad \tilde{\xi} = O(n \log^2 n), \quad \sigma = O(n \log^2 n).$$

3.2. Applications and preprocessing. The algorithm can be used to quickly compute the eigendecomposition of matrices with the low-rank property. Such matrices arising in various fields, and their HSS forms or approximations can be constructed with several strategies. If no additional knowledge is available on the matrix entries, then a direct HSS construction [45] may be used. In practice, this is usually unnecessary. Often, fast analytical or algebraic methods can be used for the HSS construction, and the cost is about $O(n)$ or less. For example, for banded matrices, an HSS form can be constructed on the fly. For Toeplitz matrices, the HSS construction can be done in nearly $O(n)$ flops with randomized methods. These are explained as follows.

If A is banded with blocks A_{jj} on the main diagonal and $A_{j,j+1}$ on the first block superdiagonal, then the HSS generators look like [42]

$$D_{\mathbf{i}} = \begin{pmatrix} A_{j-1,j-1} & A_{j-1,j} & \\ A_{j,j-1} & A_{j,j} & A_{j,j+1} \\ & A_{j+1,j} & A_{j+1,j+1} \end{pmatrix}, \quad U_i = \begin{pmatrix} I & 0 \\ 0 & 0 \\ 0 & I \end{pmatrix},$$
$$R_{\mathbf{c}_1} = \begin{pmatrix} I & 0 \\ 0 & 0 \end{pmatrix}, \; R_{\mathbf{c}_2} = \begin{pmatrix} 0 & 0 \\ 0 & I \end{pmatrix}, \quad B_{\mathbf{c}_1} = \begin{pmatrix} 0 & 0 \\ A_{j+1,j+2} & 0 \end{pmatrix},$$

where the zero and identity blocks have sizes bounded by the half bandwidth. Thus, the bandwidth of A determines its off-diagonal rank bound r.

Our algorithm can also be applied to Toeplitz matrices, and may be modified for other structured matrices (Toeplitz-like, Hankel, and Hankel-like) with the aid of displacement structures [18, 22, 26, 30, 35]. In fact, the rank structure of Toeplitz matrices in Fourier space is known as follows.

LEMMA 3.2. *[13, 33, 36] For a Toeplitz matrix A, let C be a Cauchy-like matrix resulting from the transformation of T into Fourier space through the use of displacement structures. Then the off-diagonal numerical ranks of C are $O(\log n)$ for a given tolerance.*

In particular, to preserve the symmetry as well as the real entries [33], we use the following Cauchy-like form:

$$C = \mathcal{F}_n A \mathcal{F}_n^*, \tag{3.2}$$

where $\mathcal{F}_n$ is the order-n normalized inverse discrete Fourier transform matrix. C can be approximated by an HSS form via a randomized HSS construction [47]. This construction is based on fast Toeplitz matrix-vector multiplication and randomized low-rank approximation, and costs $O(n \log^2 n)$.

For applications involving simple discretized kernel matrices, multipole expansions may be used to construct the HSS form [7].

4. Impact of HSS off-diagonal compression on the accuracy of eigenvalues. For practical problems such as Toeplitz matrices, a dense matrix A is approximated by an HSS form $\tilde{A}$ first. We thus study the impact of off-diagonal compression on the accuracy of the eigenvalues and verify that the accuracy is well controlled by the approximation tolerance (and the FMM accuracy which, as an implementation issue, can be made very high and is not discussed). The study can be viewed as structured perturbation analysis for Hermitian eigenvalue problems. Previously, for special cases such as tridiagonal or banded A, there have been various studies on whether a small off-diagonal entry or block can be neglected [25, 28, 29, 34, 48]. Here for dense A, we are only truncating the singular values of the off-diagonal blocks. A significant benefit of an HSS approximation is to enable us to conveniently assess how the off-diagonal compression affects the accuracy of the eigenvalues.

4.1. General results. We start with a block 2×2 form A and a one-level HSS approximation:

$$A \equiv \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \approx \tilde{A} \equiv \begin{pmatrix} D_1 & U_1 B_1 U_2^T \\ U_2 B_1^T U_1^T & D_2 \end{pmatrix}, \tag{4.1}$$

where $D_1 \equiv A_{11}$, $D_2 \equiv A_{22}$, and U_1 and U_2 are assumed to have orthonormal columns as often used. We study how the eigenvalues $\tilde{\lambda}_i$ of $\tilde{A}$ approximate the eigenvalues λ_i of A due to the approximation $A_{12} \approx U_1 B_1 U_2^T$. For convenience, suppose $U_1 B_1 U_2^T$ is a truncated SVD of A_{12} so that the full SVD of A_{12} looks like

$$A_{12} = \begin{pmatrix} U_1 & \hat{U}_1 \end{pmatrix} \begin{pmatrix} B_1 & \\ & \hat{B}_1 \end{pmatrix} \begin{pmatrix} U_2^T \\ \hat{U}_2^T \end{pmatrix} = U_1 B_1 U_2^T + \hat{U}_1 \hat{B}_1 \hat{U}_2^T. \tag{4.2}$$

Thus,

$$A = \tilde{A} + E, \quad \text{with} \quad E = \begin{pmatrix} 0 & \hat{U}_1 \hat{B}_1 \hat{U}_2^T \\ \hat{U}_2 \hat{B}_1^T \hat{U}_1^T & 0 \end{pmatrix}.$$

As a direct result of Weyl's theorem [15] and the fact that $||E||_2 = ||\hat{B}_1||_2$, we have the following accuracy estimation, which indicates that the off-diagonal compression accuracy controls the eigenvalue accuracy.

LEMMA 4.1. *For A and $\tilde{A}$ in (4.1), suppose $||\hat{B}_1||_2 \le \tau$ in (4.2). Then*

$$|\lambda_i - \tilde{\lambda}_i| \le \tau.$$

More generally, a bound can be obtained for a general multilevel HSS approximation. Suppose we apply truncated SVDs with a tolerance τ to the off-diagonal blocks of A as in [45] to get the HSS approximation $\tilde{A}$. An HSS approximation error bound in [40] may be used, but would yield a conservative estimate of the eigenvalue accuracy. Here instead, we use a much tighter error bound, which was previously also given in [1]. Here, we further show that it is attainable.

THEOREM 4.2. *Suppose a multilevel HSS approximation $\tilde{A}$ to A is constructed via truncated SVDs applied to the off-diagonal blocks of A, so that each B generator is obtained with the accuracy τ like in (4.2). Let l be the total number of levels (excluding the root) in the HSS tree. Then the approximation error matrix $E = A - \tilde{A}$ satisfies the following bound that is* attainable*:*

$$||E||_2 \leq l\tau. \tag{4.3}$$

Thus,

$$|\lambda_i - \tilde{\lambda}_i| \leq l\tau.$$

Proof. The HSS construction means

$$A = \tilde{A} + E, \quad \text{with}$$

$$E = \sum_{\tilde{l}=1}^{l} \operatorname{diag}\left(\left(\begin{array}{cc} 0 & \hat{U}_{\mathbf{i}} \hat{B}_{\mathbf{i}} \hat{U}_{\mathrm{sib}(\mathbf{i})}^T \\ \hat{U}_{\mathrm{sib}(\mathbf{i})} \hat{B}_{\mathbf{i}}^T \hat{U}_{\mathbf{i}}^T & 0 \end{array} \right), \quad \mathbf{i}\text{: all nodes at level } \tilde{l} \right), \tag{4.4}$$

where each $\hat{U}$ matrix has orthonormal columns. Since $||\hat{B}_{\mathbf{i}}|| \leq \tau$,

$$||E||_2 \leq \sum_{\tilde{l}=1}^{l} \max_{\mathbf{i}\text{: all nodes at level } \tilde{l}} \left\| \left(\begin{array}{cc} 0 & \hat{U}_{\mathbf{i}} \hat{B}_{\mathbf{i}} \hat{U}_{\mathrm{sib}(\mathbf{i})}^T \\ \hat{U}_{\mathrm{sib}(\mathbf{i})} \hat{B}_{\mathbf{i}}^T \hat{U}_{\mathbf{i}}^T & 0 \end{array} \right) \right\|_2 \leq \sum_{\tilde{l}=1}^{l} \tau = l\tau.$$

The error in $|\lambda_i - \tilde{\lambda}_i|$ then follows from Weyl's theorem.

To show that the bound in (4.3) is attainable, consider a special approximation error matrix E in (4.4) that looks like

$$E^{(l)} \equiv \sum_{\tilde{l}=1}^{l} \operatorname{diag}\left(\left(\begin{array}{cc} 0 & \tau I \\ \tau I & 0 \end{array} \right), \quad \mathbf{i}\text{: all nodes at level } \tilde{l} \right).$$

Then it can be shown that

$$||E^{(l)}||_2 = l\tau.$$

In fact, the eigenvalues of $E^{(l)}$ are

$$\begin{array}{rl} \pm\tau, \pm 3\tau, \ldots, \pm l\tau, & \text{if } l \text{ is odd, or} \\ 0, \pm 2\tau, \ldots, \pm l\tau, & \text{if } l \text{ is even.} \end{array}$$

This can be proven based on induction. First for $l = 1$, the matrix $\left(\begin{array}{cc} 0 & \tau I \\ \tau I & 0 \end{array} \right)$ has eigenvalues $\pm\tau$. Suppose $j\tau$ is an eigenvalue of $E^{(l)}$ with the corresponding eigenvector q. Then

$$E^{(l+1)} \left(\begin{array}{c} q \\ q \end{array} \right) = \left(\begin{array}{c} E^{(l)} q + \tau q \\ \tau q + E^{(l)} q \end{array} \right) = \left(\begin{array}{c} j\tau q + \tau q \\ \tau q + j\tau q \end{array} \right) = (j+1)\tau \left(\begin{array}{c} q \\ q \end{array} \right),$$

$$E^{(l+1)} \left(\begin{array}{c} q \\ -q \end{array} \right) = \left(\begin{array}{c} E^{(l)} q - \tau q \\ \tau q - E^{(l)} q \end{array} \right) = \left(\begin{array}{c} j\tau q - \tau q \\ \tau q - j\tau q \end{array} \right) = (j-1)\tau \left(\begin{array}{c} q \\ -q \end{array} \right).$$

Thus, $(j+1)\tau$ and $(j-1)\tau$ are eigenvalues of $E^{(l)}$. Based on this, it is not hard to find all the eigenvalues. (This also shows how the eigenvectors of $E^{(l)}$ can be found. In particular, for the eigenvalue $l\tau$ of $E^{(l)}$, the corresponding eigenvector is $(\ 1\ \ \cdots\ \ 1\ )^T/\sqrt{l}$.) ∎

Lemma 4.1 and Theorem 4.2 indicate how the accuracy of the eigenvalues depends on the HSS approximation accuracy. Theorem 4.2 shows that the error in the eigenvalues due to all the off-diagonal compression is only amplified by at most the number of levels of the HSS tree. Note that $l = O(\log \frac{n}{r})$.

4.2. Additional discussions on the accuracy of eigenvalues. There is also a potential to further improve the previous general accuracy results.

First, there are some very useful error diminishing effects so that the approximation errors in some off-diagonal blocks have little impact on the accuracy of certain eigenvalues. Following the eigenvalue perturbation analysis in [34], there is a useful *shielding effect* related to the compression of the off-diagonal blocks of A. That is, for certain eigenvalues λ of A originating from, say, A_{11} in (4.1), the accuracy of λ is roughly shielded from the approximation error within the other subproblem A_{22}. (λ is said to originate from A_{11} in the sense that it is a certain continuous function of the perturbation [34].) More specifically, the HSS approximation error δ in A_{22} appears in the error bound of λ like $O(\delta^2)$.

In particular, if the singular values of the off-diagonal blocks quickly decay to a desired accuracy τ, then a compact HSS form $\tilde{A}$ can be used to compute the eigenvalues with satisfactory accuracies. This type of problem is indeed an important application of HSS methods. For example, when A results from the discretization of a kernel function that is smooth away from the diagonal singularity, then the subblocks of the off-diagonal blocks have a decay property, i.e., they quickly decay when they are farther away from the diagonal. In this case, the accuracy shielding effect can be more rigorously characterized as a multiplicative effect for the off-diagonal compression accuracy. The reader is referred to [34] for more discussions.

Next, the error diminishing effects above and also various existing perturbation analysis imply that the off-diagonal approximation errors have less impact on the eigenvalues that are well separated from the rest of the spectrum (see, e.g., [31, 34]). In addition, for such eigenvalues, their accuracies can also be conveniently estimated. The following result directly follows from Theorem 4.2 and [27], and may give a tighter bound than those in Section 4.1 if the projection of the error matrix E onto the eigenspace of λ_i is much smaller than $||E||_2$.

PROPOSITION 4.3. *Let $E = A - \tilde{A}$ be the HSS approximation error matrix as in Theorem 4.2. Then for any eigenvalue λ_i of A satisfying $|\lambda_i - \lambda_{i+1}| > 2l\tau$, $|\lambda_{i-1} - \lambda_i| > 2l\tau$, we have*

$$|\lambda_i - \tilde{\lambda}_i| \le ||Eq_i||_2, \tag{4.5}$$

where q_i is the eigenvector associated with λ_i.

This result can also yield a useful feature with HSS matrices. That is, when the off-diagonal blocks are compressed, we can conveniently assess the effect of slightly increasing or decreasing the off-diagonal numerical ranks via the treatment of the perturbation as essentially an HSS form. Thus, we can conveniently keep track of the accuracy in (4.5) via a fast HSS matrix-vector multiplication, where q_i can be extracted from the numerical eigenmatrix.

Finally, for eigenvalues that are not well separated from the rest of the spectrum, it is also possible to make certain improvements. In particular, if the gaps between

an eigenvalue λ_i and its neighbor eigenvalues are small and close to the tolerance, then it is possible to save work by truncating some off-diagonal singular values with magnitudes close to these gaps. That is, instead of including such singular values in the low-rank updated eigenvalue computation, we may use some update formulas to directly refine the accuracy of $\tilde{\lambda}_i$. This is useful in saving computations when the off-diagonal singular values decay to magnitudes around the tolerance and then the decay slows down. The details are technical and are skipped. Overall, we hope the discussions in this subsection can lead to possible interesting future research directions in the accuracy of structured eigenvalue solutions.

5. Numerical results. In this section, we demonstrate the efficiency and accuracy of our algorithm in terms of some symmetric HSS matrices. A Toeplitz matrix and a discretized matrix are tested. Similar results are also observed for several other types of symmetric matrices that admit HSS approximations. The algorithm is implemented in Matlab, and is compared with a recent HSS eigensolver in [41] based on bisection and HSS factorization update, also in Matlab. The algorithm in [41] has a cost of over $O(n^2)$ for finding all the eigenvalues (only). The following notation is used throughout the tests:

- `NEW`: our superfast DC eigensolver;
- `XXC14`: the HSS eigensolver in [41];
- λ_i: the eigenvalues of A (here, the results from the Matlab function `eig` are used as the "exact" eigenvalues);
- $\hat{\lambda}_i$: the numerical eigenvalues;
- $\hat{Q}$: the numerical eigenmatrix with column $\hat{q}_i$ being the numerical eigenvector associated with $\hat{\lambda}_i$;
- $\gamma = \frac{\max_i \|A\hat{q}_i - \hat{\lambda}_i \hat{q}_i\|_2}{n\|A\|_2}$: the residual, as used in [23];
- $\theta = \frac{\max_i \|\hat{Q}^T \hat{q}_i - e_i\|_2}{n}$: the loss of orthogonality, as used in [23];
- $e = \frac{\sqrt{\Sigma_{i=1}^n (\lambda_i - \hat{\lambda}_i)^2}}{n\sqrt{\Sigma_{i=1}^n \lambda_i^2}}$: the relative error;
- $\xi, \tilde{\xi}, \sigma$: complexity measurements as in Theorem 3.1.

The HSS block sizes are chosen following the strategies in common HSS practices (e.g., [12, 43, 45]). A tolerance is used in the HSS approximation and FMM (if applicable) so that both `NEW` and `XXC14` reach accuracies e around 10^{-10}. A smaller tolerance is also tested for `NEW` to reach higher or even the machine accuracy. See Remark 5.1 below.

EXAMPLE 1. First, we consider the Kac-Murdock-Szego (KMS) Toeplitz matrix A as in [38], with its entries given by

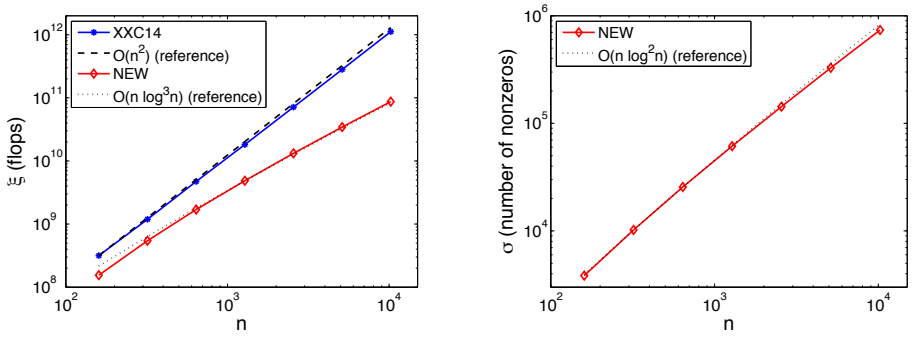
$$A_{ij} = \rho^{|i-j|}, \quad \rho = 0.5.$$

A has the same eigenvalues as the Cauchy-like matrix C in (3.2), and our tests are done on C.

As mentioned in Lemma 3.2, the maximum off-diagonal numerical rank of C grows with n as $O(\log n)$. We test C with sizes n ranging from 160 to 10240, and show the complexity ξ of `NEW` and `XXC14` to reach similar accuracies in the eigenvalues. The performance results are given in Table 5.1. `NEW` takes less work than `XXC14` for all the cases. For $n = 10240$, `NEW` is over 12 times more efficient. We also plot the results in Figure 5.1(i) with reference lines for $O(n \log^3 n)$ and $O(n^2)$. Clearly, the asymptotic complexity scales like $O(n \log^3 n)$ for `NEW` (see Theorem 3.1), and $O(n^2)$ for `XXC14`.

TABLE 5.1
Example 1 (KMS Toeplitz matrix): Complexity ξ of NEW *for finding all the eigenvalues (as compared with* XXC14*), complexity $\tilde{\xi}$ of* NEW *for applying the eigenmatrix to a vector, and storage σ of* NEW *for the eigenmatrix.*

	n	160	320	640	1280	2560	5120	10240
XXC14	ξ	3.17e08	1.19e09	4.72e09	1.82e10	7.14e10	2.82e11	1.12e12
NEW	ξ	1.55e08	5.45e08	1.70e09	4.86e09	1.32e10	3.44e10	8.70e10
	$\tilde{\xi}$	3.40e05	1.26e06	4.53e06	1.36e07	3.81e07	1.01e08	2.61e08
	σ	3.84e03	1.02e04	2.56e04	6.14e04	1.43e05	3.28e05	7.37e05



(i) Eigenvalue solution cost ξ (ii) Structured eigenmatrix storage σ

FIG. 5.1. *Example 1 (KMS Toeplitz matrix): Complexity ξ of* NEW *and* XXC14 *for finding all the eigenvalues, and storage σ of* NEW *for the eigenmatrix.*

NEW further gives a structured eigenmatrix Q, which can be applied quickly to a vector. See Table 5.1 for its cost $\tilde{\xi}$. The storage σ is also plotted in Figure 5.1(ii), and scales like $O(n \log^2 n)$. On the other hand, the eigenmatrix is not available from XXC14.

We have also compared NEW with the Matlab built-in `eig` function, which is highly optimized. Our algorithm is initially slower for smaller n, but scales much better. For $n = 2560, 5120, 10240$, the runtimes of NEW are 12.3, 40.0, 80.9 seconds, respectively (on a MacBook Pro with an Intel Core i7 CPU and 8GB memory), and those of `eig` are 5.1, 36.6, 270.0 seconds, respectively. Clearly, even if our code is far less optimized and the Matlab runtime is pessimistic for non-built-in routines, NEW already shows significant advantages for larger n.

The accuracies are shown in Table 5.2. Both methods reach similar accuracies in the eigenvalues. Since NEW also produces the eigenvectors, we report the residual γ and the orthogonality measurement θ. In particular, θ for NEW reaches nearly machine precision.

REMARK 5.1. We would like to point out that, with a smaller tolerance, the residual and error in NEW can reach nearly machine precision too, as shown in Table 5.3. The corresponding cost of NEW is higher than with the 10^{-10} tolerance, but still scales like $O(n \log^3 n)$. See Figure 5.2.

REMARK 5.2. As mentioned at the beginning of this section, the residual measurement we use follows [23] and is not the regular one, so as to show that our structured DC eigensolver can reach desired accuracies, and can also reach machine

TABLE 5.2
Example 1 (KMS Toeplitz matrix): Accuracy (error e, residual γ, and loss of orthogonality θ) of the methods when the tolerance in the off-diagonal compression and FMM is set to be around 10^{-10}.

	n	160	320	640	1280	2560
XXC14	e	$2.40e-10$	$1.02e-10$	$5.80e-11$	$4.39e-11$	$3.84e-11$
NEW	e	$1.00e-09$	$1.07e-10$	$1.47e-10$	$9.32e-11$	$8.45e-11$
	γ	$3.49e-09$	$1.49e-09$	$7.38e-10$	$2.53e-10$	$9.99e-11$
	θ	$1.79e-16$	$3.69e-16$	$7.94e-16$	$6.56e-16$	$8.53e-16$

TABLE 5.3
Example 1 (KMS Toeplitz matrix): Accuracy (error e, residual γ, and loss of orthogonality θ) of NEW when the tolerance in the off-diagonal compression and FMM is set to be around 10^{-15}.

	n	160	320	640	1280	2560
NEW	e	$9.64e-16$	$1.01e-15$	$1.27e-15$	$1.07e-15$	$1.31e-15$
	γ	$4.14e-15$	$4.40e-15$	$6.69e-15$	$7.62e-15$	$6.26e-15$
	θ	$4.25e-16$	$5.33e-16$	$7.24e-16$	$9.37e-16$	$7.18e-16$

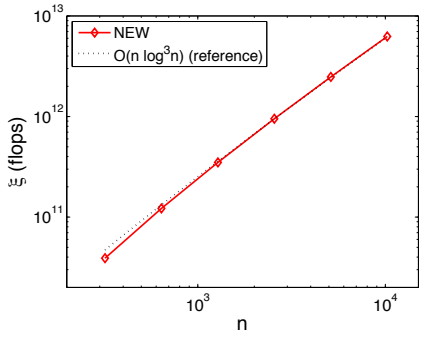


FIG. 5.2. *Example 1 (KMS Toeplitz matrix): Complexity ξ of NEW when the tolerance in the off-diagonal compression and FMM is set to be around 10^{-15}.*

precision. We have also checked the regular accuracy measurements. For the tests in Table 5.3, the regular errors $|\lambda_i - \hat{\lambda}_i|$ are in the magnitudes around $10^{-19} \sim 10^{-16}$, mostly $10^{-19} \sim 10^{-17}$. (The errors are consistent with the bound in Theorem 4.2.) The regular residuals $\|A\hat{q}_i - \hat{\lambda}_i\hat{q}_i\|_2$ are around $10^{-11} \sim 10^{-10}$. We have also computed the gaps $\hat{g}_i = \min_{j\neq i} |\hat{\lambda}_i - \lambda_j|$, which are around $10^{-6} \sim 10^{-3}$. It is known that if $\hat{\lambda}_i$ is the Rayleigh quotient of A and $\hat{q}_i$, then $|\lambda_i - \hat{\lambda}_i| \leq \|A\hat{q}_i - \hat{\lambda}_i\hat{q}_i\|_2^2/\hat{g}_i$ [15]. Here, our results are observed to roughly follow such a relationship.

EXAMPLE 2. In our next example, we consider a matrix A in the following form:

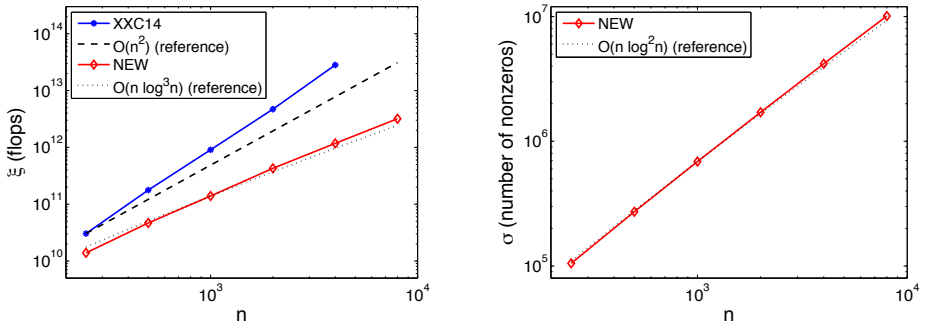
$$A_{i,j} = \sqrt{|x_i^{(n)} - x_j^{(n)}|},$$

where the points $x_i^{(n)} = \cos(\pi(2i+1)/(2n))$ are the zeros of the nth Chebyshev polynomial. Thus, the points are not uniformly distributed.

This is a matrix resulting from the discretization of $\sqrt{|x-y|}$ at the given points. It is well known to have small off-diagonal numerical ranks [11], which also grow with n, but the growth is moderate. Our method still exhibits nearly linear complexity with satisfactory accuracies. See Tables 5.4 and 5.5 and Figure 5.3 for the test results. For $n = 4000$, `NEW` is already over 23 times more efficient than `XXC14`.

TABLE 5.4
Example 2 (discretized kernel matrix): Complexity ξ of `NEW` for finding all the eigenvalues (as compared with `XXC14`), complexity $\tilde{\xi}$ of `NEW` for applying the eigenmatrix to a vector, and storage σ of `NEW` for the eigenmatrix.

	n	250	500	1000	2000	4000	8000
`XXC14`	ξ	$3.04e10$	$1.77e11$	$9.06e11$	$4.70e12$	$2.83e13$	Failed
`NEW`	ξ	$1.39e10$	$4.66e10$	$1.39e11$	$4.28e11$	$1.18e12$	$3.19e12$
	$\tilde{\xi}$	$2.50e07$	$7.10e07$	$1.92e08$	$4.98e08$	$1.26e09$	$3.11e09$
	σ	$1.05e05$	$2.72e05$	$6.87e05$	$1.71e06$	$4.19e06$	$1.01e07$



(i) Eigenvalue solution cost ξ (ii) Structured eigenmatrix storage σ

FIG. 5.3. *Example 2 (discretized kernel matrix): Complexity ξ of `NEW` and `XXC14` for finding all the eigenvalues, and storage σ of `NEW` for the eigenmatrix.*

TABLE 5.5
Example 2 (discretized kernel matrix): Accuracy (error e, residual γ, and loss of orthogonality θ) of the methods.

	n	250	500	1000	2000	4000
`XXC14`	e	$2.51e-10$	$1.52e-10$	$6.01e-11$	$3.60e-11$	$2.52e-11$
`NEW`	e	$2.40e-11$	$8.71e-11$	$1.14e-10$	$7.36e-11$	$2.33e-10$
	γ	$3.68e-10$	$5.05e-10$	$7.36e-10$	$5.08e-10$	$6.47e-10$
	θ	$3.59e-15$	$5.39e-15$	$6.39e-15$	$5.29e-15$	$8.44e-15$

6. Conclusions. This work designs a superfast DC algorithm to compute the eigendecomposition of symmetric matrices with small off-diagonal ranks or numerical ranks. We illustrate the preservation of the rank structure during the recursive DC dividing process, as well as how to quickly and stably perform a sequence of operations in computing the eigenvalues and eigenvectors in the conquering stage. The

nearly linear complexity is proven, and verified with applications such as Toeplitz and discretized matrices. In the tests, for even modest sizes n, the new method takes dramatically less work than a recent HSS eigensolver.

We further show approximation error bounds for the eigenvalues due to hierarchical off-diagonal compression. The analysis confirms that the accuracy is conveniently controlled by the compression tolerance. Some eigenvalues may be accurately evaluated even if the compression accuracy is not so high.

The algorithm and analysis may be modified for the computation of SVDs of nonsymmetric HSS matrices. For matrices with higher off-diagonal ranks, we may approximate them by compact HSS forms and then use our superfast DC method to estimate the eigenvalue distribution. This is useful in preconditioning. Such extensions, as well as more practical implementations will appear in future work.

Acknowledgements. The authors are grateful to the anonymous reviewers for their valuable suggestions, and would also like to thank Xiaobai Sun for some discussions and Difeng Cai for helping with the implementation of the FMM algorithm.

REFERENCES

[1] H. Bagci, J. E. Pasciak, and K. Y. Sirenko, *A convergence analysis for a sweeping preconditioner for block tridiagonal systems of linear equations*, Numer. Linear Algebra Appl., 22 (2015), pp. 371–392.

[2] R. Beatson and L. Greengard, *A short course on fast multipole methods*, Wavelets, Multilevel Methods and Elliptic PDEs, Oxford University Press, (1997), pp. 1–37.

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

[4] D. Bini and V. Y. Pan, *Parallel complexity of tridiagonal symmetric eigenvalue problem*, Proceedings of the 2nd Annual ACM-SIAM Symposium on Discrete Algorithms, 384–393, ACM Press, New York, and SIAM Publications, Philadelphia, 1991.

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

[6] J. R. Bunch, C. P. Nielson, and D. C. Sorenson, *Rank-one modification of the symmetric eigenproblem*, Numer. Math., 41 (1978), pp. 31–48.

[7] D. Cai and J. Xia, *A stable and efficient matrix version of the fast multipole method*, preprint to be submitted, 2015, http://www.math.purdue.edu/~xiaj/work/fmm1d.pdf.

[8] J. Carrier, L. Greengard, and V. Rokhlin, *A fast adaptive multipole algorithm for particle simulations*, SIAM J. Sci. Stat. Comput., 9 (1988), pp. 669–686.

[9] 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.

[10] 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.

[11] S. Chandrasekaran, P. Dewilde, M. Gu, W. Lyons, and T. Pals, *A fast solver for HSS representations via sparse matrices*, SIAM J. Matrix Anal. Appl., 29 (2006), pp. 67–81.

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

[13] 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. 111–143.

[14] J. J. M. Cuppen, *A divide and conquer method for the symmetric tridiagonal eigenproblem*, Numer. Math., 36 (1981), pp. 177–195.

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

[16] J. J. Dongarra and D. C. Sorensen, *A fully parallel algorithm for the symmetric eigenvalue problem*, SIAM J. Sci. Stat. Comput. 8 (1987), pp. s139–s154.

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

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

[19] G. H. Golub, *Some modified matrix eigenvalue problems*, SIAM Rev. 15 (1973), pp. 318–334.
[20] L. Grasedyck and W. Hackbusch, *Construction and arithmetics of $\mathcal{H}$-matrices*, Computing, 70 (2003), pp. 295–334.
[21] L. Greengard and V. Rokhlin. *A Fast algorithm for particle simulations*, J. Comp. Phys., 73 (1987), pp. 325–348.
[22] M. Gu, *Stable and efficient algorithms for structured linear equations*, SIAM J. Matrix Anal. Appl., 19 (1998), pp. 279–306.
[23] 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.
[24] W. Hackbusch, *A sparse matrix arithmetic based on $\mathcal{H}$-matrices. Part I: Introduction to $\mathcal{H}$-matrices*, Computing, 62 (1999), pp. 89–108.
[25] W. W. Hager, *Perturbations in eigenvalues*, Linear Algebra Appl., 42 (1982), pp. 39–55.
[26] G. Heineg, *Inversion of generalized Cauchy matrices and other classes of structured matrices*, Linear Algebra for Signal Processing, IMA Vol. Math. Appl. 69, Springer, New York, 1995, pp. 63–81.
[27] I. C. F. Ipsen and B. Nadler, *Refined perturbation bounds for eigenvalues of Hermitian and non-Hermitian matrices*, SIAM J. Matrix Anal. Appl., 31 (2009), pp. 40–53.
[28] E.-R. Jiang, *Perturbation in eigenvalues of a symmetric tridiagonal matrix*, Linear Algebra Appl., 399 (2005), pp. 91–107.
[29] W. Kahan, *When to neglect off-diagonal elements of symmetric tridiagonal matrices*, Technical Report CS42, Computer Science Department, Stanford University, July 1966.
[30] T. Kailath, S. Kung, and M. Morf, *Displacement ranks of matrices and linear equations*, J. Math. Anal. Appl., 68 (1979), pp. 395–407.
[31] C.-K. Li and R.-C. Li, *A note on eigenvalues of perturbed Hermitian matrices*, Linear Algebra Appl., 395 (2005), pp. 183–190.
[32] R.-C. Li, *Solving secular equations stably and efficiently*, Technical Report UT-CS-94-260, University of Tennessee, Knoxville, TN, Nov. 1994. LAPACK Working Note No. 89.
[33] P. G. Martinsson, V. Rokhlin, and M. Tygert, *A fast algorithm for the inversion of general Toeplitz matrices*, Comput. Math. Appl. 50 (2005), pp. 742–752.
[34] C. C. Paige, *Eigenvalues of perturbed hermitian matrices*, Linear Algebra Appl., 8 (1974), pp. 1–10.
[35] V. Y. Pan, *On computations with dense structured matrices*, Math. Comp., 55 (1990), pp. 179–190.
[36] V. Y. Pan, *Trasnformations of matrix structures work again*, Linear Algebra Appl., 465 (2015), pp. 107–138.
[37] D. C. Sorensen and P. T. P. Tang, *On the orthoganality of eigenvectors computed by divide-and-conquer techniques*, SIAM J. Numer. Anal., 28 (1991), pp. 1752–1775.
[38] W. F. Trench, *Numerical solution of the eigenvalue problem for Hermitian Toeplitz matrices*, SIAM J. Matrix Anal. Appl., 10 (1989), pp. 135–156.
[39] J. H. Wilkinson, *The algebraic eigenvalue problem*, Clarendon Press, Oxford, 1965.
[40] 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., 35 (2014), pp. 44–72.
[41] Y. Xi, J. Xia, and R. Chan, *A fast randomized eigensolver with structured LDL factorization update*, SIAM J. Matrix Anal. Appl., 35 (2014), pp. 974–996.
[42] J. Xia, *Fast direct solvers for structured linear systems of equations*, Ph.D. Thesis, University of California, Berkeley, 2006.
[43] J. Xia, *On the complexity of some hierarchical structured matrix algorithms*, SIAM J. Matrix Anal. Appl., 33 (2012), pp. 388–410.
[44] J. Xia, *Efficient structured multifrontal factorization for general large sparse matrices*, SIAM J. Sci. Comput., 35 (2013), pp. A832–A860.
[45] 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.
[46] J. Xia, Y. Xi, S. Cauley, and V. Balakrishnan, *Fast sparse selected inversion*, SIAM J. Matrix Anal. Appl., revised, 2015.
[47] 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.
[48] Q. Ye, *Relative perturbation bounds for eigenvalues of symmetric positive definite diagonally dominant matrices*, SIAM J. Matrix Anal. Appl., 31 (2009), pp. 11–17.