SUPERDC: SUPERFAST DIVIDE-AND-CONQUER EIGENVALUE DECOMPOSITION WITH IMPROVED STABILITY FOR RANK-STRUCTURED MATRICES*

XIAOFENG OU† AND JIANLIN XIA†

Abstract. For dense symmetric matrices with small off-diagonal (numerical) ranks and in a hierarchically semiseparable form, we give a divide-and-conquer eigendecomposition method with nearly linear complexity (called SuperDC) that significantly improves an earlier basic algorithm in [Vogel, Xia, et al., SIAM J. Sci. Comput., 38 (2016)]. Some stability risks in the original algorithm are analyzed, including potential exponential norm growth, cancellations, loss of accuracy with clustered eigenvalues or intermediate eigenvalues, etc. In the dividing stage, we give a new structured low-rank updating strategy with balancing that eliminates the exponential norm growth and also minimizes the ranks of low-rank updates. In the conquering stage with low-rank updated eigenvalue solution, the original algorithm directly uses the standard fast multipole method (FMM) to accelerate function evaluations, which has the risks of cancellation, division by zero, and slow convergence. Here, we design a triangular FMM to avoid cancellation. Furthermore, when there are clustered intermediate eigenvalues, we design a novel local shifting strategy to integrate FMM accelerations into the solution of shifted secular equations. This helps achieve both the efficiency and the reliability. We also provide a deflation strategy with a user-supplied tolerance and give a precise description of the structure of the resulting eigenvector matrix. The SuperDC eigensolver has significantly improved stability while keeping the nearly linear complexity for finding the entire eigenvalue decomposition. Extensive numerical tests are used to show the efficiency and accuracy of SuperDC.

Key words. superfast eigenvalue decomposition, divide-and-conquer method, rank-structured matrix, triangular fast multipole method, shifted secular equation, local shifting

AMS subject classifications. 65F15, 65F55, 15A18, 15A23

1. Introduction.

In this paper, we consider the full eigenvalue decomposition of $n \times n$ real symmetric matrices A with small off-diagonal ranks (and also A that can be approximated well by matrices with small off-diagonal ranks). Such matrices belong to the class of rank-structured matrices. Examples include banded matrices with finite bandwidth, Toeplitz matrices in Fourier space [33, 47, 55], some matrices arising from discretized PDEs and integral equations [32, 36, 51, 53], some kernel matrices [10, 56], etc. The eigenvalue decompositions of relevant matrices are very useful in computations such as matrix function evaluations [4], discretized linear system solutions [45], matrix equation solutions [35], and quadrature approximations [40]. They are also very useful in fields such as optimization, imaging, Gaussian processes, and machine learning [35].

There are several types of rank-structured forms, such as $\mathcal{H}/\mathcal{H}^2$ matrices [26, 27], hierarchical semiseparable (HSS) matrices [12, 54], quasiseparable/semiseparable matrices [11, 21, 42], BLR matrices [2], and HODLR matrices [1]. Examples of eigensolvers for these rank-structured matrices include divide-and-conquer methods [3, 13, 20, 29, 38, 44], QR iterations [7, 15, 19, 21, 41], bisection, [6, 48], and methods using accelerated characteristic polynomial evaluations [8].

Our work here focuses on the divide-and-conquer method for HSS matrices (that may be dense or sparse). The divide-and-conquer method has previously been well studied for tridiagonal matrices (which may be considered as special HSS forms). See, e.g., [5, 9, 16, 18, 24, 34]. In particular, a stable version is given in [24]. The algorithms

*The research of Jianlin Xia was supported in part by an NSF grant DMS-1819166.

†Department of Mathematics, Purdue University, West Lafayette, IN 47907 (ou17@purdue.edu, xiaj@purdue.edu).

can compute all the eigenvalues in $O(n^2)$ flops and can compute the eigenvectors in $O(n^3)$ flops. It is also mentioned in [24] that it is possible to accelerate the operations in the divide-and-conquer process via the fast multipole method (FMM) [23] to reach nearly linear complexity. However, this was not actually done in [24] or later relevant work [13, 29]. Only recently, the feasibility of the FMM acceleration of the divide-and-conquer process was verified in a structured eigensolver in [44], which works for HSS matrices without the need of tridiagonal reductions. For an HSS matrix with off-diagonal ranks bounded by r (which may be a constant or a power of $\log n$), the method in [44] computes a structured eigendecomposition in $O(r^2 n \log^2 n)$ flops with storage $O(rn \log n)$. The method is then said to be *superfast*.

The work in [44] presents the basic framework of a rank-structured divide-and-conquer eigensolver. It gives a proof-of-concept algorithm and verifies the feasibility of such superfast eigenvalue solution for HSS matrices. Due to the complex nature of the entire framework with many components, that preliminary work has some limitations. It does not consider some crucial stability issues in the HSS divide-and-conquer process, such as the risks of exponential norm growth and potential cancellations in some function evaluations. Moreover, it does not incorporate several key stability strategies that are used in practical tridiagonal divide-and-conquer algorithms. These limitations are due to some major challenges in combining FMM accelerations with those stability strategies, especially for problems with clustered eigenvalues.

Specifically, in the dividing stage, upper-level off-diagonal block information is used to update lower-level diagonal blocks (also as HSS forms) of A in a hierarchical process. The norms of the updated lower-level blocks may grow quickly during the process, which brings stability risks and may even cause overflow. In the conquering stage, multiple types of function evaluations are need in eigenvalue solutions (via a modified Newton's method applied to some secular equations). The application of FMM accelerations needs to assemble these function evaluations into matrix-vector multiplications. However, classical stabilization techniques involve strategies such as splitting function evaluations to avoid cancellation (see, e.g., [5, 9, 18, 28, 24]) and solving certain *shifted* secular equations to guarantee accuracy for clustered eigenvalues [5, 9, 18, 28, 24]. Such splitting and shifting strategies depend on the each individual eigenvalue to be sought so that it is difficult to find all the eigenvalues together with the usual FMM acceleration. (Sections 4.2.1 and 4.3.1 show the details.) The algorithm in [44] directly applies usual FMM accelerations to standard secular equations. This may lose accuracy or even encounter cancellations.

Thus, the main purpose of this paper is to overcome these limitations. We follow the basic framework in [44] but provide some important stability, accuracy, and efficiency improvements. We show how to integrate structured accelerations with several stabilization strategies. A more reliable superfast divide-and-conquer eigensolver (called SuperDC) is then designed to find an approximate eigenvalue decomposition of A:

$$A \approx Q \Lambda Q^T, \tag{1.1}$$

where Λ is a diagonal matrix for the eigenvalues and Q is for the orthogonal eigenvectors. For convenience, we call the matrix Q an *eigenmatrix*. (Our presentation focuses on real symmetric A, and the ideas can be immediately extended to complex Hermitian matrices). The main significance of the work includes the following.

1. We analyze why the original hierarchical dividing strategy in [44] can lead to exponential norm growth. A more stable dividing strategy is designed, where a balancing technique guarantees the norm growth is well under control. We

also give a strategy to choose appropriate low-rank updates in the dividing stage so as to reduce the rank of low-rank updates and save eigenvalue solution costs.

2. In the solution of the secular equations for the eigenvalues, we design a triangular FMM to accommodate the eigenvalue-dependent splitting (for the stability purpose as mentioned above). This enables us to quickly and stably evaluate the functions (after splitting) through matrix-vector multiplications that are not suitable for the standard FMM.
3. When shifted secular equations are used to handle clustered intermediate eigenvalues, we design a *local shifting* strategy that integrates shifts into FMM matrices without destroying the FMM structure. This enhances the stability of eigenvalue solution, leading to improved eigenvalue accuracy and also better convergence of iterative secular equation solution.
4. We also provide clarifications and improvements on several aspects such as the precise structure of the resulting eigenmatrix, the eigenvalue deflation criterion with a user-supplied tolerance, and the stopping criterion in iterative eigenvalue solution.
5. With all the stabilization strategies, SuperDC still nicely preserve the nearly linear complexity. The eigendecomposition complexity is still $O(r^2 n \log^2 n)$, with $O(rn \log n)$ storage. No extra tridiagonal reduction is needed for dense HSS matrices. We provide extensive numerical tests a SuperDC package in Matlab. For modest matrix sizes n, SuperDC already has significantly lower runtime and storage than some other eigensolvers while producing satisfactory accuracies. Benefits of our stabilization strategies are also demonstrated.

In the remaining sections, we begin in Section 2 with a quick review of the basic HSS divide-and-conquer eigensolver in [44]. Then the improved structured dividing strategy is discussed in Section 3, followed by the efficient structured conquering scheme in Section 4. Section 5 gives some numerical experiments to demonstrate the efficiency and accuracy. Then Section 6 concludes the paper. A list of the major algorithms is given in the supplementary materials.

Throughout this paper, the following notation is used.

- Lower-case letters in bold fonts like $\mathbf{u}$ are used to denote vectors.
- $(A_{ij})_{n\times n}$ means an $n \times n$ matrix with the (i, j)-entry A_{ij}.
- Sometimes, a vector $\mathbf{s}$ may be viewed as an ordered set formed by its components s_i. Then $s_i \in \mathbf{s}$ means s_i is a component of $\mathbf{s}$. Accordingly, for vectors $\mathbf{s}$ and $\mathbf{t}$, a matrix $(\kappa(s_i, t_j))_{s_i\in\mathbf{s},t_j\in\mathbf{t}}$ may be defined by the evaluation of a function $\kappa(s, t)$ at the components of $\mathbf{s}$ and $\mathbf{t}$.
- $\mathrm{diag}(\cdots)$ denotes a (block) diagonal matrix.
- $\mathrm{rowsize}(A)$ and $\mathrm{colsize}(A)$ mean the row and column sizes of A, respectively.
- $\mathbf{u} \odot \mathbf{v}$ denotes the entrywise (Hadamard) product of two vectors $\mathbf{u}$ and $\mathbf{v}$.
- For a binary tree $\mathcal{T}$, we suppose it is in postordering so that it has nodes $i = 1, 2, \ldots, \mathrm{root}(\mathcal{T})$, where $\mathrm{root}(\mathcal{T})$ is the root.
- $\mathrm{fl}(x)$ denotes the floating point result of x.
- ϵ_{mach} represents the machine precision.

2. Review of the basic superfast divide-and-conquer eigensolver. We first briefly summarize the basic superfast divide-and-conquer eigensolver in [44], which generalizes the classical divide-and-conquer method for tridiagonal matrices to HSS matrices.

A symmetric HSS matrix A [54] may be defined with the aid of a postordered full

binary tree $\mathcal{T}$ called *HSS tree*, and has a nested structure that looks like

$$D_p = \begin{pmatrix} D_i & U_i B_i U_j^T \\ U_j B_i^T U_i^T & D_j \end{pmatrix}, \tag{2.1}$$

where $p \in \mathcal{T}$ has child nodes i and j, so that D_p with $p = \mathrm{root}(\mathcal{T})$ is the entire HSS matrix A. Here, the U matrices are off-diagonal basis matrices and also satisfy a nested relationship $U_p = \begin{pmatrix} U_i & \\ & U_j \end{pmatrix} \begin{pmatrix} R_i \\ R_j \end{pmatrix}$. The D_i, U_i, B_i matrices are called *HSS generators* associated with node i. The maximum size of the B generators is usually referred as the *HSS rank* of A. We suppose $\mathrm{root}(\mathcal{T})$ is at level 0, and the children of a node i at level l are at level $l+1$.

The superfast divide-and-conquer eigensolver in [44] finds the eigendecomposition (1.1) of A through a dividing stage and a conquering stage as follows.

2.1. Dividing stage. In the dividing stage in [44], A and its submatrices are recursively divided into block-diagonal HSS forms plus low-rank updates. Start with $p = \mathrm{root}(\mathcal{T})$ and its two children i and j. $A = D_p$ in (2.1) can be written as

$$D_p = \begin{pmatrix} D_i - U_i B_i B_i^T U_i^T & \\ & D_j - U_j U_j^T \end{pmatrix} + \begin{pmatrix} U_i B_i \\ U_j \end{pmatrix} \begin{pmatrix} B_i^T U_i^T & U_j^T \end{pmatrix}. \tag{2.2}$$

For notational convenience, we suppose the HSS rank of A is r and each B generator has column size r. Let

$$\hat{D}_i = D_i - U_i B_i B_i^T U_i^T, \quad \hat{D}_j = D_j - U_j U_j^T, \quad Z_p = \begin{pmatrix} U_i B_i \\ U_j \end{pmatrix}, \tag{2.3}$$

and we arrive at

$$D_p = \mathrm{diag}(\hat{D}_i, \hat{D}_j) + Z_p Z_p^T. \tag{2.4}$$

Here, the diagonal blocks D_i and D_j are modified so that a rank-r update $Z_p Z_p^T$ can be used instead of a rank-$2r$ update. The column size of Z_p is referred as the *rank of the low-rank update* and here we have $\mathrm{colsize}(Z_p) = \mathrm{colsize}(B_i)$.

During this process, the blocks $\hat{D}_i$ and $\hat{D}_j$ remain to be HSS forms. In fact, it is shown in [44, 54] that any matrix of the form $D_i - U_i H U_i^T$ can preserve the off-diagonal basis matrices of D_i. Specifically, the following lemma can be used for generator updates.

LEMMA 2.1. *[44] Let $\mathcal{T}_i$ be the subtree of the HSS tree $\mathcal{T}$ that has the node i as the root. Then $D_i - U_i H U_i^T$ has HSS generators $\tilde{D}_k, \tilde{U}_k, \tilde{R}_k, \tilde{B}_k$ for each node $k \in \mathcal{T}_i$ as follows:*

$$\begin{aligned} &\tilde{U}_k = U_k, \quad \tilde{R}_k = R_k, \\ &\tilde{B}_k = B_k - (R_k R_{k_l} \cdots R_{k_1}) H (R_{k_1}^T \cdots R_{k_l}^T R_{\tilde{k}}^T), \\ &\tilde{D}_k = D_k - U_k (R_k R_{k_l} \cdots R_{k_1}) H (R_{k_1}^T \cdots R_{k_l}^T R_k^T) U_k^T \quad \textit{for a leaf } k, \end{aligned} \tag{2.5}$$

where $\tilde{k}$ is the sibling node of k and $k \to k_l \to \cdots \to k_1 \to i$ is the path connecting k to i. Accordingly, $D_i - U_i H U_i^T$ and D_i have the same off-diagonal basis matrices.

Thus, the HSS generators of $\hat{D}_i$ and $\hat{D}_j$ can be conveniently obtained via the generator update procedure (2.5). Then the dividing process can continue on $\hat{D}_i$ and $\hat{D}_j$ like above with p in (2.2) replaced by i and j, respectively.

2.2. Conquering stage. Suppose eigenvalue decompositions of the subproblems $\hat{D}_i$ and $\hat{D}_j$ in (2.3) have been computed respectively as

$$\hat{D}_i = Q_i \Lambda_i Q_i^T, \quad \hat{D}_j = Q_j \Lambda_j Q_j^T. \tag{2.6}$$

Then from (2.4), we have

$$D_p = \mathrm{diag}(Q_i, Q_j) \left(\mathrm{diag}(\Lambda_i, \Lambda_j) + \hat{Z}_p \hat{Z}_p^T \right) \mathrm{diag}(Q_i^T, Q_j^T), \quad \text{with} \tag{2.7}$$

$$\hat{Z}_p = \mathrm{diag}(Q_i^T, Q_j^T) Z_p. \tag{2.8}$$

Consequently, if we can solve the rank-r updating problem

$$\mathrm{diag}(\Lambda_i, \Lambda_j) + \hat{Z}_p \hat{Z}_p^T = \hat{Q}_p \Lambda_p \hat{Q}_p^T, \tag{2.9}$$

then the eigendecomposition of D_p can be simply retrieved as

$$D_p = Q_p \Lambda_p Q_p^T, \quad \text{with} \quad Q_p = \mathrm{diag}(Q_i, Q_j) \hat{Q}_p. \tag{2.10}$$

Therefore, the main task is to compute the eigendecomposition of the low-rank update problem (2.9). To this end, suppose $\hat{Z}_p = \begin{pmatrix} \mathbf{z}_1 & \cdots & \mathbf{z}_r \end{pmatrix}$, where $\mathbf{z}_k$'s are the columns of $\hat{Z}_p$. Then (2.9) can be treated as r rank-1 updating problems $\mathrm{diag}(\Lambda_i, \Lambda_j) + \sum_{k=1}^{r} \mathbf{z}_k \mathbf{z}_k^T$. As a result, a basic component is to quickly find the eigenvalue decomposition of a diagonal plus rank-1 updating problem in the following form:

$$\tilde{\Lambda} + \mathbf{v}\mathbf{v}^T = \tilde{Q} \Lambda \tilde{Q}^T, \tag{2.11}$$

where $\tilde{\Lambda} = \mathrm{diag}(d_1, \ldots, d_n)$ with $d_1 \le \cdots \le d_n$, $\mathbf{v} = (v_1, \ldots, v_n)^T$, $\tilde{Q} = (\tilde{\mathbf{q}}_1, \ldots, \tilde{\mathbf{q}}_n)$, and $\Lambda = \mathrm{diag}(\lambda_1, \ldots, \lambda_n)$.

As in standard divide-and-conquer eigensolvers (see, e.g., [5, 16, 24]), the eigenvalues λ_k are found through the solution of the following secular equation [22]:

$$f(x) = 1 + \sum_{k=1}^{n} \frac{v_k^2}{d_k - x} = 0. \tag{2.12}$$

Newton iterations with rational interpolations may be used and the cost for finding all the n roots is $O(n^2)$. Once λ_k is computed, a corresponding eigenvector looks like $\tilde{\mathbf{q}}_k = (\tilde{\Lambda} - \lambda_k I)^{-1} v$. Such an analytical form is not directly used in general for the stability reason, since any loss of precision in the computed λ_k can be significantly amplified in $(\tilde{\Lambda} - \lambda_k I)^{-1} v$, which will result in the loss of eigenvector orthogonality [18, 24]. A stable way to obtain $\tilde{\mathbf{q}}_k$ is given in [24] based on Löwner's formula.

It is also mentioned in [24] that nearly $O(n)$ complexity may be achieved by assembling multiple operations into matrix-vector multiplications that can be accelerated by the FMM. This is first verified in [44], where the complexity of the algorithm for finding the entire eigendecomposition is $O(r^2 n \log^2 n)$ instead of $O(n^3)$, with the eigenmatrix Q in (1.1) given in a structured form that needs $O(rn \log n)$ storage instead of $O(n^2)$. In the following sections, we give a series of stability enhancements to get an improved superfast divide-and-conquer eigensolver.

3. Improved structured dividing strategy. In this section, we point out a stability risk in the original dividing method as given in (2.2)–(2.3) and propose a more stable dividing strategy. We also design a way to minimize colsize(Z_p).

The stability risk can be illustrated as follows. Consider $\hat{D}_i$ in (2.2) which is the result of updating D_i in the dividing process associated with the parent p of i. Suppose i has children c_1 and c_2 such that

$$D_i = \begin{pmatrix} D_{c_1} & U_{c_1} B_{c_1} U_{c_2}^T \\ U_{c_2} B_{c_2} U_{c_1}^T & D_{c_2} \end{pmatrix}, \quad U_i = \begin{pmatrix} U_{c_1} & \\ & U_{c_2} \end{pmatrix} \begin{pmatrix} R_{c_1} \\ R_{c_2} \end{pmatrix}. \tag{3.1}$$

Then

$$\hat{D}_i = D_i - U_i B_i B_i^T U_i^T = \begin{pmatrix} \tilde{D}_{c_1} & U_{c_1} \tilde{B}_{c_1} U_{c_2}^T \\ U_{c_2} \tilde{B}_{c_1}^T U_{c_1}^T & \tilde{D}_{c_2} \end{pmatrix},$$

where

$$\begin{aligned} &\tilde{D}_{c_1} = D_{c_1} - U_{c_1} R_{c_1} B_i B_i^T R_{c_1}^T U_{c_1}^T, \quad \tilde{D}_{c_2} = D_{c_2} - U_{c_2} R_{c_2} B_i B_i^T R_{c_2}^T U_{c_2}^T, \\ &\tilde{B}_{c_1} = B_{c_1} - R_{c_1} B_i B_i^T R_{c_2}^T. \end{aligned} \tag{3.2}$$

In HSS constructions [54], to ensure stability of HSS algorithms, the U basis generators often have orthonormal columns [46, 47]. Then due to (3.1), the R generators also satisfy that $\begin{pmatrix} R_{c_1} \\ R_{c_2} \end{pmatrix}$ has orthonormal columns. Then each B generator has 2-norm equal to its associated off-diagonal block. For example, $\|B_i\|_2 = \|U_i B_i U_j^T\|_2$. Furthermore, $\|R_{c_1}\|_2 \le 1$, $\|R_{c_2}\|_2 \le 1$, and (3.2) means

$$\|\tilde{B}_{c_1}\|_2 \le \|B_{c_1}\|_2 + \|B_i\|_2^2. \tag{3.3}$$

If the off-diagonal block $U_i B_i U_j^T$ has a large norm, $\|\tilde{B}_{c_1}\|_2$ can potentially be much larger than $\|B_{c_1}\|_2$. We can similarly observe the norm growth with the updated D generators. Moreover, when the dividing process proceeds on $\tilde{D}_{c_1}$, the norms of the updated B, D generators at lower levels can grow exponentially.

PROPOSITION 3.1. *Suppose the U_k generator of A associated with each node k of $\mathcal{T}$ with $k \neq \mathrm{root}(\mathcal{T})$ has orthonormal columns and all the original B_k generators satisfy $\|B_k\|_2 \le \beta$ with $\beta \gg 1$. Also suppose the leaves of $\mathcal{T}$ are at level $l_{\max} \le \log_2 n$. When the original dividing process in Section 2.1 proceeds from $\mathrm{root}(\mathcal{T})$ to a nonleaf node i, immediately after finishing the dividing process associated with node i,*

- *with i at level $l \le l_{\max} - 2$, the updated B_k generator (denoted $\tilde{B}_k$) associated with any descendant k of i satisfies*

$$\|\tilde{B}_k\|_2 = O(\beta^{2^l}) \le O(\beta^{n/4}), \tag{3.4}$$

  where $O(\cdot)$ denotes the asymptotic upper bound and is given in terms of the highest order term in β;
- *with i at level $l \le l_{\max} - 1$, the updated D_k generator (denoted $\tilde{D}_k$) associated with any leaf descendant k of i satisfies*

$$\|\tilde{D}_k\|_2 = \|D_k\|_2 + O(\beta^{2^l}) \le \|D_k\|_2 + O(\beta^{n/2}). \tag{3.5}$$

Proof. Following the update formulas in Lemma 2.1, we just need to show the norm bound for $\|\tilde{B}_k\|_2$. The bound for $\|\tilde{D}_k\|_2$ can be shown similarly.

After the dividing process associated with $\mathrm{root}(\mathcal{T})$ is finished, according to (2.5), $\tilde{B}_k$ associated with any descendant k of a child i of $\mathrm{root}(\mathcal{T})$ looks like

$$\tilde{B}_k = B_k - (R_k R_{k_{m-1}} \cdots R_{k_1}) H_i (R_{k_1}^T \cdots R_{k_{m-1}}^T R_{\tilde{k}}^T), \tag{3.6}$$

where $H_i = B_i B_i^T$ if i is the left child of root($\mathcal{T}$) or $H_i = I$ otherwise, k is supposed to be at level m with sibling $\tilde{k}$, and $k \to k_{m-1} \to \cdots \to k_1 \to i$ is the path connecting k to i in the HSS tree $\mathcal{T}$. Clearly, $\|H_i\|_2 \le \beta^2$. With the orthogonality condition of the U basis generators, $\begin{pmatrix} R_{c_1} \\ R_{c_2} \end{pmatrix}$ also has orthogonal columns. Then we get

$$\|\tilde{B}_k\|_2 \le \|B_k\|_2 + \|H_i\|_2 \le \beta + \beta^2 = O(\beta^2). \tag{3.7}$$

Then in the dividing process associated with node i at level 1, for a child c of i (see Figure 3.1 for an illustration), the generator $\tilde{D}_c$ is further updated to

$$\hat{D}_c = \tilde{D}_c - U_c H_c U_c^T, \tag{3.8}$$

where $H_c = \tilde{B}_c \tilde{B}_c^T$ if c is the left child of i or $H_c = I$ otherwise. We have $\|H_c\|_2 \le \|\tilde{B}_c\|_2^2$ for the first case and $\|H_c\|_2 = 1$ for the second case. From (3.7), we have $\|H_c\|_2 \le (\beta^2 + \beta)^2$. For any descendant k of c with sibling $\tilde{k}$, (3.8) needs to update the generator B_k to

$$\begin{aligned} \tilde{B}_k = \; & B_k - (R_k R_{k_{m-1}} \cdots R_{k_2} R_c) H_i (R_c^T R_{k_2}^T \cdots R_{k_{m-1}}^T R_{\tilde{k}}^T) \\ & - (R_k R_{k_{m-1}} \cdots R_{k_2}) H_c (R_{k_2}^T \cdots R_{k_{m-1}}^T) R_{\tilde{k}}^T, \end{aligned} \tag{3.9}$$

where the last term on the right-hand side is because of the update associated with the dividing of D_i like in (3.6). Then

$$\|\tilde{B}_k\|_2 \le \|B_k\|_2 + \|H_i\|_2 + \|H_c\|_2 \le \beta + \beta^2 + (\beta^2 + \beta)^2 = O(\beta^4). \tag{3.10}$$

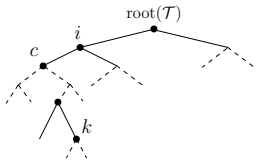


FIG. 3.1. *Nodes involved in the dividing process.*

If the dividing process continues to c, it is similar to obtain $\|\tilde{B}_k\|_2 = O(\beta^8)$ for any descendant k of a child of c. We can then similarly reach the conclusion on the general pattern of the norm growth as in (3.4). Also, if i is at level $l_{\max} - 1$, then B_k associated with a child k of i is not updated, which is why only i at level $l \le l_{\max} - 2$ contributes to the norm growth of lower level B generators. ∎

The bound $O(\beta^{2^l})$ in (3.4) for $\|\tilde{B}_k\|_2$ and the bound $\|D_k\|_2 + O(\beta^{2^l})$ in (3.5) for $\|\tilde{D}_k\|_2$ are attainable. To see this, suppose i is a child of root($\mathcal{T}$) and $\|B_i\|_2 = \beta$. Note the multiplicative forms like H_i below (3.6) and H_c below (3.8). Following the proof, we can see that the asymptotic upper bounds (3.4) and (3.5) can be attained at some leaf level node k after the dividing process associated with the parent of k is completed.

This proposition indicates that, during the original hierarchical dividing process, the updated B, D generators associated with a lower-level node may potentially have exponential norm accumulation, as long as one of its ancestors is associated with a B generator with a large norm. This can cause stability issues or even overflow, as confirmed in the numerical tests later.

To resolve this, we introduce *balancing*/scaling into the updates and propose a new dividing strategy. That is, we replace the original dividing method (2.2) by

$$D_p = \begin{pmatrix} D_i - \frac{1}{\|B_i\|_2} U_i B_i B_i^T U_i^T & \\ & D_j - \|B_i\|_2 U_j U_j^T \end{pmatrix} + \begin{pmatrix} \frac{1}{\sqrt{\|B_i\|_2}} U_i B_i \\ \sqrt{\|B_i\|_2} U_j \end{pmatrix} \begin{pmatrix} \frac{1}{\sqrt{\|B_i\|_2}} B_i^T U_i^T & \sqrt{\|B_i\|_2} U_j^T \end{pmatrix}. \tag{3.11}$$

Then we still have (2.4), but with

$$\hat{D}_i = D_i - \frac{1}{\|B_i\|_2} U_i B_i B_i^T U_i^T, \quad \hat{D}_j = D_j - \|B_i\|_2 U_j U_j^T, \quad Z_p = \begin{pmatrix} \frac{1}{\sqrt{\|B_i\|_2}} U_i B_i \\ \sqrt{\|B_i\|_2} U_j \end{pmatrix}. \tag{3.12}$$

We show how this strategy controls the norms of the updated B, D generators.

PROPOSITION 3.2. *Suppose the same conditions as in Proposition 3.1 hold, except that (2.2) is replaced by (3.11) so that (2.3) is replaced by (3.12). Then (3.4) and (3.5) become, respectively,*

$$\|\tilde{B}_k\|_2 \le 2^l \beta \le \frac{n}{4}\beta, \quad \|\tilde{D}_k\|_2 \le \|D_k\|_2 + 2^l\beta \le \|D_k\|_2 + \frac{n}{2}\beta. \tag{3.13}$$

Proof. The proof follows a procedure similar to the proof for Proposition 3.1. Again, we just show the result for $\|\tilde{B}_k\|_2$. After the dividing process associated with root$(\mathcal{T})$ is finished, we still have (3.6) for any descendant k of a child i of root$(\mathcal{T})$, except that $H_i = \frac{B_i B_i^T}{\|B_i\|_2}$ if i is the left child of root$(\mathcal{T})$ or $H_i = \|B_i\|_2 I$ otherwise. In either case, we have $\|H_i\|_2 \le \beta$. Then (3.7) becomes

$$\|\tilde{B}_k\|_2 \le 2\beta. \tag{3.14}$$

Then in the dividing process associated with node i at level 1, for a child c of i, the generator $\tilde{D}_c$ is updated like in (3.8), except that $H_c = \frac{\tilde{B}_c \tilde{B}_c^T}{\|\tilde{B}_c\|_2}$ if c is the left child of i or $H_c = \|\tilde{B}_c\|_2 I$ otherwise. We have $\|H_c\|_2 \le \|\tilde{B}_c\|_2$ for both cases. From (3.14), $\|H_c\|_2 \le 2\beta$. For any descendant k of c, (3.8) still requires the update of the generator B_k to $\tilde{B}_k$ like in (3.9), except that (3.10) now becomes

$$\|\tilde{B}_k\|_2 \le \|B_k\|_2 + \|H_i\|_2 + \|H_c\|_2 \le \beta + \beta + 2\beta = 4\beta.$$

If the dividing process continues to c, it is similar to obtain $\|\tilde{B}_k\|_2 \le 8\beta$ for any descendant k of the left child of c. We can similarly get the norm growth as in (3.13) in general. ∎

Therefore, the norm growth now becomes at most linear in n and is well controlled, in contrast to the exponential growth in Proposition 3.1. Here again, the upper bounds $2^l\beta$ for $\|\tilde{B}_k\|_2$ and $\|D_k\|_2 + 2^l\beta$ for $\|\tilde{D}_k\|_2$ are attainable.

Next, we can also minimize colsize(Z_p), the rank of the low-rank update. Note that in the original dividing method (2.2) in [44], the updates to the two diagonal blocks involve the B_i generator in different ways. That is, D_i is updated by $-U_i B_i B_i^T U_i^T$ while D_j is updated by $-U_j U_j^T$. In fact, (2.2) may be reformulated so that D_i is updated by $-U_i U_i^T$ while D_j is updated by $-U_j B_i^T B_i U_j^T$. It is not clear from [44] which way is better.

In fact, in (2.2) and also (3.11)–(3.12), the rank of the low-rank update is equal to colsize(B_i). In practice, B_i may not be a square matrix. Thus, (3.12) shall be used only if colsize(B_i) $\leq$ rowsize(B_i). Otherwise, we replace (3.12) by the following:

(3.15)
$$\hat{D}_i = D_i - \|B_i\|_2 U_i U_i^T, \quad \hat{D}_j = D_j - \frac{1}{\|B_i\|_2} U_j B_i^T B_i U_j^T, \quad Z_p = \begin{pmatrix} \sqrt{\|B_i\|_2} U_i \\ \frac{1}{\sqrt{\|B_i\|_2}} U_j B_i^T \end{pmatrix},$$

so that (2.4) still holds. In (3.15), the low-rank update size is now rowsize(B_i). With such a choice between (3.12) and (3.15), we ensure that the size of the low-rank update colsize(Z_p) is always min(rowsize(B_i), colsize(B_i)). This strategy benefits the efficiency in the conquering stage since it reduces the number of rank-1 updates. With these new ideas, we have a more stable and efficient dividing stage.

4. Improved structured conquering stage. In this section, we discuss the solution of the eigenvalues and eigenvectors in the conquering stage via the integration of various stability strategies into FMM accelerations. We first show a flexible deflation strategy. Then we give a triangular FMM idea for accelerating the secular equation solution and a local shifting strategy for solving shifted secular equations and constructing structured eigenvectors. We also discuss the framework of the overall eigendecomposition and the precise structure of the overall eigenmatrix.

4.1. User-controlled deflation. As reviewed in Section 2.2, the key problem in the conquering stage is to quickly find the eigendecomposition of the rank-one updating problem (2.11). Like in earlier studies in [9, 18], an eigenvalue deflation step may be first applied to reduce the size of (2.11) if $|v_j|$ or the difference $|d_j - d_{j+1}|$ is small. In the implementations of the tridiagonal divide-and-conquer eigensolver (see, e.g., [5]), the deflation is performed in a two-step procedure with a tolerance related to ϵ_{mach}. Here, we follow the same steps, but replace ϵ_{mach} with a user-supplied deflation tolerance τ to get a more flexible deflation procedure.

(i) If $|v_j| < \tau$, without loss of generality, we assume $j = n$, $\mathbf{v} = \begin{pmatrix} \mathbf{v}_1 \\ v_n \end{pmatrix}$ and get

$$\tilde{\Lambda} + \mathbf{v}\mathbf{v}^T = \begin{pmatrix} \tilde{\Lambda}_1 & \\ & d_n \end{pmatrix} + \begin{pmatrix} \mathbf{v}_1 \\ v_n \end{pmatrix} \begin{pmatrix} \mathbf{v}_1^T & v_n \end{pmatrix} \approx \begin{pmatrix} \tilde{\Lambda}_1 + \mathbf{v}_1\mathbf{v}_1^T & \\ & d_n \end{pmatrix},$$

where the approximation has an error proportional to τ. Then we only need to find the eigendecomposition of the smaller problem $\tilde{\Lambda}_1 + \mathbf{v}_1\mathbf{v}_1^T$.

(ii) If $|(d_j - d_{j+1})v_j v_{j+1}| < (v_j^2 + v_{j+1}^2)\tau$, we can find a Givens rotation matrix G such that $G \begin{pmatrix} v_j \\ v_{j+1} \end{pmatrix} = \begin{pmatrix} 0 \\ w \end{pmatrix}$ with $w = \sqrt{v_j^2 + v_{j+1}^2}$. Then

$$G \left(\begin{pmatrix} d_j & \\ & d_{j+1} \end{pmatrix} + \begin{pmatrix} v_j \\ v_{j+1} \end{pmatrix} \begin{pmatrix} v_j & v_{j+1} \end{pmatrix}^T \right) G^T = \begin{pmatrix} d_j & \mu \\ \mu & d_{j+1} \end{pmatrix} + \begin{pmatrix} 0 \\ w \end{pmatrix} \begin{pmatrix} 0 & w \end{pmatrix}^T$$
$$\approx \begin{pmatrix} d_j & \\ & d_{j+1} \end{pmatrix} + \begin{pmatrix} 0 \\ w \end{pmatrix} \begin{pmatrix} 0 & w \end{pmatrix}^T = \begin{pmatrix} d_j & \\ & d_{j+1} + w^2 \end{pmatrix},$$

where $\mu = \frac{(d_j - d_{j+1})v_j v_{j+1}}{v_j^2 + v_{j+1}^2}$ and the approximation has a 2-norm error $|\mu| < \tau$. This then leads to a diagonal subproblem.

After the above deflation steps, the problem size of (2.11) is reduced and the simplified problem satisfies

$$|v_j| \geq \tau \quad \text{and} \quad |d_j - d_{j+1}| \geq \frac{(v_j^2 + v_{j+1}^2)\tau}{|v_j v_{j+1}|}. \tag{4.1}$$

The parameter τ offers the flexibility to control the accuracy of the eigenvalues. When only moderate accuracy is needed, a larger τ can be used for a more significant reduction in the problem size. Moreover, this can sometimes avoid the need to deal with situations where $|\lambda_j - d_j|$ or $|\lambda_j - d_{j+1}|$ is too small.

4.2. Fast secular equation solution. Assume (4.1) holds for (2.11) so that no deflation is needed. We consider the solution of the secular equation (2.12) for its eigenvalues $\lambda_k, k = 1, 2, \ldots, n$. Without loss of generality, suppose the diagonal entries d_k of $\tilde{\Lambda}$ are ordered from the smallest to the largest.

4.2.1. Standard FMM accelerations and the limitation. When the modified Newton's method is used to solve for λ_k, it needs to evaluate f (referred to as the secular function) in (2.12) and its derivative f' at certain $x_k \in (d_k, d_{k+1})$. The idea in [13, 24, 44] is to assemble the function evaluations for all k together as matrix-vector multiplications that can be accelerated by the standard FMM. That is, let

$$\mathbf{f} = \begin{pmatrix} f(x_1) & \cdots & f(x_n) \end{pmatrix}^T, \quad \mathbf{f}' = \begin{pmatrix} f'(x_1) & \cdots & f'(x_n) \end{pmatrix}^T,$$

$$\mathbf{v} = \begin{pmatrix} v_1 & \cdots & v_n \end{pmatrix}^T, \quad \mathbf{w} = \mathbf{v} \odot \mathbf{v}, \quad \mathbf{e} = \begin{pmatrix} 1 & \cdots & 1 \end{pmatrix}^T, \tag{4.2}$$

$$C = \left(\frac{1}{d_j - x_i} \right)_{n \times n}, \quad S = \left(\frac{1}{(d_j - x_i)^2} \right)_{n \times n}, \tag{4.3}$$

$$\mathbf{f} = \mathbf{e} + C\mathbf{w}, \quad \mathbf{f}' = S\mathbf{w}. \tag{4.4}$$

The vectors $\mathbf{f}$ and $\mathbf{f}'$ can be quickly evaluated by the FMM with the kernel functions $\kappa(s,t) = \frac{1}{s-t}$ and $\kappa(s,t) = \frac{1}{(s-t)^2}$, respectively.

A basic idea of the FMM for computing, say, $C\mathbf{w}$ is as follows. Note that C is the evaluation of $\kappa(s,t) = \frac{1}{s-t}$ at interlaced points $s \in \{d_j\}_{1 \le j \le n}$ and $t \in \{x_i\}_{1 \le i \le n}$:

$$d_i < x_i < d_{i+1} < x_{i+1}, \quad 1 \le i \le n-1. \tag{4.5}$$

The sets $\{x_i\}_{1 \le i \le n}$ and $\{d_j\}_{1 \le j \le n}$ together are treated as one set and then hierarchically partitioned. This also naturally leads to a hierarchical partition of both $\{x_i\}_{1 \le i \le n}$ and $\{d_j\}_{1 \le j \le n}$. Consider two subsets produced in this partitioning:

$$\mathbf{s}_x \subset \{x_i\}_{1 \le i \le n}, \quad \mathbf{s}_d \subset \{d_j\}_{1 \le j \le n}. \tag{4.6}$$

Use $C_{\mathbf{s}_x, \mathbf{s}_d} = (\kappa(d_j, x_i))_{x_i \in \mathbf{s}_x, d_j \in \mathbf{s}_d}$ to denote the block of C defined by $\mathbf{s}_x$ and $\mathbf{s}_d$, which is often referred as the *interaction* between $\mathbf{s}_x$ and $\mathbf{s}_d$.

- If $\mathbf{s}_x$ and $\mathbf{s}_d$ are well separated (a precise definition of the separation can be found in [23, 37]), then $C_{\mathbf{s}_x, \mathbf{s}_d}$ can be approximated by a low-rank form

$$C_{\mathbf{s}_x, \mathbf{s}_d} \approx U_{\mathbf{s}_x} B_{\mathbf{s}_x, \mathbf{s}_d} V_{\mathbf{s}_d}^T. \tag{4.7}$$

  Such a low-rank approximation can be obtained via a degenerate expansion of $\kappa(s,t)$ and has a bounded rank for any specified approximation accuracy. That is, the size of $B_{\mathbf{s}_x, \mathbf{s}_d}$ is bounded. (See [10] for an example of the accuracy study.) The subsets $\mathbf{s}_x$ and $\mathbf{s}_d$ are also said to be *far-field* clusters and the submatrix $C_{\mathbf{s}_x, \mathbf{s}_d}$ is a far-field interaction/block.
- On the other hand, if $\mathbf{s}_x$ and $\mathbf{s}_d$ are not well separated, then they are said to be *near-field* clusters, and $C_{\mathbf{s}_x, \mathbf{s}_d} = (\kappa(d_j, x_i))_{x_i \in \mathbf{s}_x, d_j \in \mathbf{s}_d}$ is treated as a regular dense block (near-field interaction/block).

The FMM further considers the interactions between parent and child clusters during the hierarchical partitioning, so that the U, V basis matrices in (4.7) satisfy nested relationships (like in (3.1)). The details can be found in [23] and are not our focus here. (Also see [10] particularly for a stable 1D matrix version.) The FMM essentially constructs an *FMM matrix* approximation to C and multiplies it with $\mathbf{w}$. The complexity of each FMM matrix-vector mutliplication is $O(n)$.

In light of (4.3) and (4.4), a straightforward idea in [13, 24, 44] is to apply the standard FMM to C and S for fast evaluations of $\mathbf{f}$ and $\mathbf{f}'$. However, in practical implementations of secular equation solution methods, it is preferred to write $f(x)$ in the following form so as to avoid cancellation (see, [5, 9, 18]):

$$f(x) = 1 + \psi_k(x) + \phi_k(x),$$

where the splitting depends on k (when $\lambda_k \in (d_k, d_{k+1})$ is to be found):

$$\psi_k(x) = \sum_{j=1}^{k} \frac{v_j^2}{d_j - x}, \quad \phi_k(x) = \sum_{j=k+1}^{n} \frac{v_j^2}{d_j - x}. \tag{4.8}$$

Because of the interlacing property (4.5), all the terms in the sum for ψ_k or ϕ_k have the same sign for $x \in (d_k, d_{k+1})$. Furthermore, ψ_k and ϕ_k capture the behavior of f near two poles d_k and d_{k+1} respectively.

A reliable and widely used scheme to find the roots of $f(x)$ is given in [28] based on a modified Newton's method with a hybrid scheme for rational interpolations of $\psi_k(x)$ and $\phi_k(x)$. The scheme mixes a middle way method and a fixed weight method and is implemented in LAPACK [5]. In the middle way method, rational functions $\xi_{k,1}(x) = a_1 + \frac{b_1}{d_k - x}$ and $\xi_{k,2}(x) = a_2 + \frac{b_2}{d_{k+1} - x}$ are decided to interpolate ψ_k and ϕ_k respectively at $x_k \in (d_k, d_{k+1})$, so that

$$\xi_{k,1}(x_k) = \psi_k(x_k), \quad \xi'_{k,1}(x_k) = \psi'_k(x_k), \quad \xi_{k,2}(x_k) = \phi_k(x_k), \quad \xi'_{k,2}(x_k) = \phi'_k(x_k).$$

(We also follow this hybrid scheme to find the first $n-1$ roots $\lambda_1, \lambda_2, \ldots, \lambda_{n-1}$. The last root λ_n has only one pole d_n next to it, so a simple rational interpolation is used as in [5, 28]).

The modified Newton's method requires evaluations of the functions ψ_k, ϕ_k, ψ'_k, and ϕ'_k at some $x_k \in (d_k, d_{k+1})$, $1 \le k \le n-1$. (Note that even though the summands in ψ'_k and ϕ'_k have the same sign, ψ'_k and ϕ'_k are used separately in the rational interpolations by $\xi_{k,1}$ and $\xi_{k,2}$, respectively [28].) Since these functions all depend on individual k, the standard FMM cannot be applied directly. The reason is that the standard FMM handles the evaluation of a kernel $\kappa(s,t)$ at a fixed set of data points, while here it needs to evaluate $\kappa(s,t)$ at different k-dependent subsets of $\{d_j\}_{1 \le j \le n}$ and $\{x_i\}_{1 \le i \le n}$ to produce multiple k-dependent functions.

4.2.2. Triangular FMM for fast evaluations of ψ_k and ϕ_k. To resolve the difficulty of applying FMM accelerations to (4.8), we let

$$\boldsymbol{\psi} = \begin{pmatrix} \psi_1(x_1) & \cdots & \psi_n(x_n) \end{pmatrix}^T, \quad \boldsymbol{\phi} = \begin{pmatrix} \phi_1(x_1) & \cdots & \phi_{n-1}(x_{n-1}) & 0 \end{pmatrix}^T, \tag{4.9}$$

$$\boldsymbol{\psi}' = \begin{pmatrix} \psi'_1(x_1) & \cdots & \psi'_n(x_n) \end{pmatrix}^T, \quad \boldsymbol{\phi}' = \begin{pmatrix} \phi'_1(x_1) & \cdots & \phi'_{n-1}(x_{n-1}) & 0 \end{pmatrix}^T. \tag{4.10}$$

The key idea is to write

$$\mathbf{f} = \mathbf{e} + \boldsymbol{\psi} + \boldsymbol{\phi} = \mathbf{e} + C_L \mathbf{w} + C_U \mathbf{w}, \quad \mathbf{f}' = \boldsymbol{\psi}' + \boldsymbol{\phi}' = S_L \mathbf{w} + S_U \mathbf{w}, \tag{4.11}$$

where $\mathbf{e}$ is given in (4.2), C_L and S_L are the lower triangular parts of C and S, respectively, and C_U and S_U are the strictly upper triangular parts of C and S, respectively. This suggests that the FMM idea should be applied to the lower and upper triangular parts of C and S separately. That is, we need a special *triangular FMM* that can be used to quickly evaluate the triangular matrix-vector products $C_L\mathbf{w}$, $C_U\mathbf{w}$, $S_L\mathbf{w}$, $S_U\mathbf{w}$. We illustrate the triangular FMM in terms of the evaluation of $C_L\mathbf{w}$. For two subsets $\mathbf{s}_x$ and $\mathbf{s}_d$ as in (4.6), we similarly use $(C_L)_{\mathbf{s}_x,\mathbf{s}_d}$ to denote the block of C_L defined by $\mathbf{s}_x$ and $\mathbf{s}_d$.

- When $\mathbf{s}_x$ and $\mathbf{s}_d$ are neighbor clusters, the interlacing property (4.5) means $C_{\mathbf{s}_x,\mathbf{s}_d}$ is a diagonal block of C. Then $(C_L)_{\mathbf{s}_x,\mathbf{s}_d}$ is just the lower triangular part of $C_{\mathbf{s}_x,\mathbf{s}_d}$.
- When $\mathbf{s}_x$ and $\mathbf{s}_d$ are well separated, $(C_L)_{\mathbf{s}_x,\mathbf{s}_d}$ is a far-field block.
  - If $\mathbf{s}_x$ is on the right of $\mathbf{s}_d$ or $\max \mathbf{s}_x > \min \mathbf{s}_d$, the interlacing property (4.5) means $(C_L)_{\mathbf{s}_x,\mathbf{s}_d}$ is in the lower triangular part of C_L. Since C_L is the lower triangular part of C, (4.7) gives

$$(C_L)_{\mathbf{s}_x,\mathbf{s}_d} = C_{\mathbf{s}_x,\mathbf{s}_d} \approx U_{\mathbf{s}_x} B_{\mathbf{s}_x,\mathbf{s}_d} V_{\mathbf{s}_d}^T. \tag{4.12}$$

  - If $\mathbf{s}_x$ is on the left of $\mathbf{s}_d$, $(C_L)_{\mathbf{s}_x,\mathbf{s}_d}$ is in the upper triangular part of C_L and is thus a zero block. This case can still be accommodated by (4.12), with $B_{\mathbf{s}_x,\mathbf{s}_d} = 0$.

The far-field blocks of C_L then have the same U, V basis matrices as those of C. Thus, we can conveniently obtain a lower triangular FMM approximation matrix for C_L based on an FMM approximation matrix for C, just with the difference in the lower triangular diagonal blocks and in some zero B generators. The multiplication of the triangular FMM matrix with $\mathbf{w}$ takes only $O(n)$ operations. The cost of one simultaneous iteration step for all x_k's is then $O(n)$.

4.2.3. Iterative secular equation solution. During the modified Newton's method, let $x_k^{(j)}$ be an approximation to the eigenvalue λ_k at the iteration step j. A correction $\Delta x_k^{(j)}$ is computed to update $x_k^{(j)}$ as

$$x_k^{(j+1)} \leftarrow x_k^{(j)} + \Delta x_k^{(j)}. \tag{4.13}$$

(We sometimes write $x_k^{(j)}$ as x_k when the focus is not on the iteration steps j.)

We adopt the following stopping criterion from [24]:

$$|f(x_k^{(j)})| < cn(1 + |\psi(x_k^{(j)})| + |\phi(x_k^{(j)})|)\epsilon_{\text{mach}}, \tag{4.14}$$

where c is a small constant. This stopping criterion can be conveniently checked after the FMM-accelerated function evaluations, which is an advantage over a criterion in [28]. The factor n in (4.14) is related to error propagations of general matrix multiplications. Although (4.14) might be loose for an extremely large n, it works well in our tests and leads to satisfactory accuracies. It is possible to refine (4.14) to a tighter convergence estimate using the backward stability studies of FMM matrix-vector multiplication algorithms in [10, 46]. This is our ongoing work.

Typically, a very small number of iterations is needed for convergence, similarly to the tridiagonal divide-and-conquer algorithm as mentioned in [17]. (In our tests, each eigenvalue converges in 2 to 5 iterations on average.) With the total number of iterations bounded, the total iterative solution cost for finding all the eigenvalues (from one secular equation) is $O(n)$.

4.3. Local shifting in triangular FMM for shifted secular equation solution. When there are clustered eigenvalues or when updates to previous eigenvalues are small, typically the standard secular equation (2.12) is not directly solved. Instead, shifted secular equations are solved for the purpose of stability and accuracy, as discussed in [9, 18, 24]. However, it is nontrivial to apply FMM to accelerate shifted secular equation solution. In fact, the paper [24] mentions the possibility of FMM accelerations for the standard secular equation but does not consider the shifted ones. The FMM-accelerated algorithm in [44] does not use shifted secular equations either.

In this subsection, we discuss the necessity of shifting and its challenges to FMM accelerations. Moreover, we develop a new strategy that makes it feasible to apply FMM accelerations to the solution of shifted secular equations. In the following, we suppose deflation in Section 4.1 has already been applied.

4.3.1. Shifted secular equation solution and its challenge to FMM accelerations. During the solution for $\lambda_k \in (d_k, d_{k+1})$, if λ_k is very close to d_k or d_{k+1}, evaluating $\frac{v_k^2}{\lambda_k - d_k}$ or $\frac{v_{k+1}^2}{\lambda_k - d_{k+1}}$ in the secular function with a computed λ_k might lose accuracy because of cancellations in the denominator. Without loss of generality, we always assume λ_k is closer to d_k. Let the difference between λ_k and d_k be

$$\eta_k \equiv \lambda_k - d_k,$$

which is also said to be the *gap* between λ_k and d_k. With a very small gap η_k, instead of directly solving for λ_k, the remedy in [9, 18, 24] is to solve a shifted secular equation for η_k. For this purpose, we shift the origin to d_k and rewrite the original secular equation (2.12) as the equivalent *shifted secular equation* (see, e.g., [9, 18, 24]):

$$g_k(y) \equiv f(d_k + y) = 1 + \sum_{j=1}^{n} \frac{v_j^2}{\delta_{jk} - y} = 0, \quad \text{with} \tag{4.15}$$

$$\delta_{jk} = d_j - d_k, \quad j = 1, 2, \ldots, n. \tag{4.16}$$

(4.15) is solved for $y = \eta_k$. The benefits of this shifting within our context are as follows.

One benefit is to avoid catastrophic cancellation or division by zero (see, e.g., [5, 9, 24]). (Basically, for computations like $d_i - \lambda_k$, although $d_i - \lambda_k = \delta_{ik} - \eta_k$ in exact arithmetic, it is preferred to use $\delta_{ik} - \eta_k$ to avoid cancellation [9, 24].) For example, let x_k be an approximation to λ_k. In exact arithmetic, $x_k \in (d_k, d_{k+1})$. At each modified Newton iteration, it needs to guarantee $d_k < \mathrm{fl}(x_k) < d_{k+1}$. However, this might not be satisfied in floating point arithmetic when x_k is very close to d_k:

$$|d_k - x_k| = O(\epsilon_{\text{mach}}) \text{ or smaller}, \tag{4.17}$$

which may lead to cancellation when computing $d_k - \mathrm{fl}(x_k)$:

$$\mathrm{fl}(d_k - \mathrm{fl}(x_k)) = o(\epsilon_{\text{mach}}) \quad \text{or even} \quad \mathrm{fl}(d_k - \mathrm{fl}(x_k)) = 0. \tag{4.18}$$

This will cause stability issues in the numerical solutions of the standard secular function: $\mathrm{fl}\left(\frac{v_k^2}{d_k - \mathrm{fl}(x_k)}\right)$ either is highly inaccurate or becomes ∞.

Note that (4.17) and (4.18) are still possible even if deflation in Section 4.1 has been applied with a tolerance τ that is not too small. To see this, suppose $v_k = O(\tau) \geq \tau$ and the exact root λ_k satisfies $|\lambda_k - d_j| \gg v_j^2$ for $j \neq k$. Substituting λ_k

into the secular equation (2.12) yields $\frac{v_k^2}{d_k-\lambda_k} = -1 + \sum_{j\neq k}^{n} \frac{v_j^2}{\lambda_k - d_j} = O(1)$. In this case, λ_k shall be very close to d_k in the following sense:

$$|d_k - \lambda_k| = v_k^2 \cdot O(1) = O(\tau^2).$$

If $\tau = O(\epsilon_{\text{mach}}^{1/2})$ which is not extremely small, we can have (4.17) so that (4.18) may happen when solving the standard secular equation.

Another benefit for solving the shifted equation is faster convergence. It is observed in our tests that computing with η_k instead of λ_k can speed up the convergence of the modified Newton's method. To illustrate this, suppose λ_k is solved directly from the standard secular equation (2.12), then the approximation $x_k^{(j)}$ at iteration step j is updated as in (4.13). Suppose $|\lambda_k| = O(1)$ and $|\eta_k| = |\lambda_k - d_k| = O(\epsilon_{\text{mach}})$. Since $x_k^{(j)}$ converges to λ_k as $j \to \infty$, we also have $|x_k^{(j)}| = O(1)$ and $|x_k^{(j)} - d_k| = O(\epsilon_{\text{mach}})$ after some iterations. In the modified Newton's method, the correction $\Delta x_k^{(j)}$ approaches 0 as j increase. This may lead to loss of digits in the updated $x_k^{(j+1)}$: $\text{fl}(x_k^{(j+1)}) = \text{fl}(x_k^{(j)} + \Delta x_k^{(j)}) = \text{fl}(x_k^{(j)})$. As a result, the iteration stagnates. On the other hand, if η_k is solved from the shifted secular equation (4.15), as in [5, 9, 18], the update (4.13) is replaced by

$$y_k^{(j+1)} \leftarrow y_k^{(j)} + \Delta x_k^{(j)}, \tag{4.19}$$

where $y_k^{(j)} = x_k^{(j)} - d_k$ is an approximation to η_k at step j of the iterative solution. Although (4.13) and (4.19) are equivalent in exact arithmetic, the latter preserves a lot more digits of accuracy since $|y_k^{(j)}| = O(\epsilon_{\text{mach}})$.

These discussions illustrate the importance of solving the shifted secular equation (4.15) instead of the original equation (2.12). However, in an FMM-accelerated scheme where all λ_k's are solved simultaneously, it is not plausible to shift the secular equation simultaneously for all λ_k's. The reason is the shift in (4.15) depends on each individual eigenvalue and there is no such a uniform shift that would work for all λ_k's.

To see this, let $y_k = x_k - d_k$ be an approximation to η_k during the iterative solution of (4.15). The evaluations of $g_k(y)$ in (4.15) at $y = y_k$ for all $k = 1, 2, \ldots, n$ can be assembled into the matrix form

$$\mathbf{g} = \mathbf{e} + \hat{C}\mathbf{w}, \quad \text{with} \tag{4.20}$$

$$\mathbf{g} = \begin{pmatrix} g_1(y_1) & \cdots & g_n(y_n) \end{pmatrix}^T, \quad \hat{C} = \left(\frac{1}{\delta_{jk} - y_k} \right)_{1\le k,j\le n},$$

where δ_{jk} is given in (4.16). Recall that when the FMM is used to accelerate the matrix-vector product $C\mathbf{w}$ in (4.11), it relies on the separability of s and t in a degenerate approximation of $\kappa(s,t) = \frac{1}{s-t}$. (Note that in $\kappa(d_j, x_k)$, x_k only involves the row index k and d_j only involves the column index j, so that the separability can be understood in terms of the row and column indices.) However, to evaluate $\hat{C}\mathbf{w}$ in (4.20), we have

$$\kappa(d_j, x_k) = \kappa(d_j - d_k, x_k - d_k) = \kappa(\delta_{jk}, y_k). \tag{4.21}$$

δ_{jk} involves both the row and column indices, so that the separability in terms of the row and column indices does not hold. Therefore, we need to adapt the FMM to accelerate the shifted matrix-vector multiplication in (4.20).

4.3.2. FMM accelerations with local shifting. In this subsection, we propose a strategy called *local shifting* that makes it feasible to apply triangular FMM accelerations to solve (4.15). As mentioned in Section 4.2.1, multiple terms involving $x_k - d_j$ are assembled into matrices in order to apply FMM accelerations. See, e.g., (4.3). When $|x_k - d_j|$ is small, the shifting helps get $x_k - d_j$ accurately. However, when k is not near j or when $|k-j|$ is large, $x_k - d_j$ can actually be computed accurately *without involving any shift* d_k. To see this, recall that $d_k < x_k < d_{k+1}$ and also after deflation in Section 4.1, we have (4.1) holds and $|d_j - d_{j+1}| \geq \frac{{v_j}^2+v_{j+1}^2}{|v_j v_{j+1}|} \geq 2\tau$ for all j. Thus, for $j \neq k, k+1$,

$$|x_k - d_j| \geq \min(|d_k - d_j|, |d_{k+1} - d_j|) \geq 2(|k-j|-1)\tau. \tag{4.22}$$

Hence, $|x_k - d_j|$ is not too small and $x_k - d_j$ can be computed accurately when $|k-j|$ is large.

Following this justification, we have the basic ideas of our local shifting strategy: (i) small gaps resulting from shifting is used just in near-field interactions of the FMM, which does not interfere with the FMM rank structure; (ii) it is safe to not shift the numerical eigenvalues in far-field interactions, which makes it feasible to exploit the rank structure. More specifically, the major components are as follows.

1. For $k = 1, 2, \ldots, n$, the shifted secular equations (4.15) are solved together for the gaps $\eta_k = \lambda_k - d_k$ (so as to get the roots λ_k of the original secular equation (2.12)). An intermediate gap during the iterative solution looks like $y_k = x_k - d_k$. The relevant function evaluations in the iterative solutions are assembled into matrix-vector products like in (4.20).
2. The FMM is used to accelerate the resulting matrix-vector products like $\hat{C}\mathbf{w}$ in (4.20) as follows. Suppose two subsets $\mathbf{s}_x$ and $\mathbf{s}_d$ like in (4.6) are well separated. As mentioned above, for $x_k \in \mathbf{s}_x$ and $d_j \in \mathbf{s}_d$, x_k and d_j are far away from each other and $|k-j|$ is large, so $x_k - d_j$ can then be computed accurately because of (4.22). Thus, we can recover x_k from $d_k + y_k$ to directly exploit the low-rank structure like in (4.7). As a result, the far-field block $\hat{C}_{\mathbf{s}_x,\mathbf{s}_d}$ of $\hat{C}$ is now just a block of C in (4.3):
$$\hat{C}_{\mathbf{s}_x,\mathbf{s}_d} = (\kappa(\delta_{jk}, y_k))_{x_k \in \mathbf{s}_x, d_j \in \mathbf{s}_d} = (\kappa(d_j, x_k))_{x_k \in \mathbf{s}_x, d_j \in \mathbf{s}_d} = C_{\mathbf{s}_x,\mathbf{s}_d}$$
3. On the other hand, when two subsets $\mathbf{s}_x$ and $\mathbf{s}_d$ are not well separated, the near-field interaction $\hat{C}_{\mathbf{s}_x,\mathbf{s}_d}$ is dense and each entry $\kappa(\delta_{jk}, y_k)$ can be evaluated accurately via y_k and the gap δ_{jk}. This has no impact on the structures needed for FMM accelerations.
4. These ideas are then combined with the triangular FMM in Section 4.2.2 to stably and quickly perform function evaluations like (4.20) and solve the shifted secular equations.

This local shifting strategy successfully integrates shifting into triangular FMM accelerations. As a result, we can quickly and reliably solve the shifted secular equations (4.15) via the modified Newton's method. The overall complexity to find all the n roots is still $O(n)$. In addition, since the relevant functions are now evaluated more accurately than with the method in [44], the convergence is also improved. When the iterative solution of the shifted secular equations converge, we can use the resulting gaps η_k to recover the desired eigenvalues as

$$\lambda_k = d_k + \eta_k, \quad k = 1, 2, \ldots, n, \tag{4.23}$$

The local shifting strategy can also be used to stably apply FMM accelerations to other operations like finding the eigenmatrix. See the next subsection.

4.4. Structured eigenvectors via FMM with local shifting. With the identified eigenvalues λ_k in (4.23), the eigenvectors can be obtained stably as in [24]. An eigenvector corresponding to λ_k looks like

$$\mathbf{q}_k = \begin{pmatrix} \frac{\hat{v}_1}{d_1-\lambda_k} & \cdots & \frac{\hat{v}_k}{d_k-\lambda_k} & \cdots & \frac{\hat{v}_n}{d_n-\lambda_k} \end{pmatrix}^T, \tag{4.24}$$

where $\hat{\mathbf{v}} \equiv \begin{pmatrix} \hat{v}_1 & \cdots & \hat{v}_n \end{pmatrix}^T$ is given by Löwner's formula

$$\hat{v}_i = \sqrt{\frac{\prod_j (\lambda_j - d_i)}{\prod_{j\neq i}(d_j - d_i)}}, \quad i = 1, 2, \ldots, n. \tag{4.25}$$

To quickly form $\hat{\mathbf{v}}$, the standard FMM acceleration would look like the following [24]. Rewrite (4.25) as

$$\log \hat{v}_i = \frac{1}{2}\sum_{j=1}^{n} \log(|d_i - \lambda_j|) - \frac{1}{2}\sum_{j=1, j\neq i}^{n} \log|d_i - d_j|. \tag{4.26}$$

Now, let $G_1 = (\log|d_i - \lambda_j|)_{n\times n}$, $G_2 = (\log|d_i - d_j|)_{n\times n}$, where the diagonals of G_2 are set to be zero. Then

$$\log \hat{\mathbf{v}} = \frac{1}{2}(G_1\mathbf{e} - G_2\mathbf{e}). \tag{4.27}$$

$G_1\mathbf{e}$ and $G_2\mathbf{e}$ can thus be quickly evaluated by the FMM with the kernel $\log|s-t|$.

As in [24, 44], the eigenvectors are often normalized to form an orthogonal matrix

$$\hat{Q} = \left(\frac{\hat{v}_i b_j}{d_i - \lambda_j}\right)_{n\times n}, \quad \text{with} \tag{4.28}$$

$$\mathbf{b} \equiv \begin{pmatrix} b_1 & \cdots & b_n \end{pmatrix}^T, \quad b_j = \left(\sum_{i=1}^{n} \frac{\hat{v}_i^2}{(d_i - \lambda_j)^2}\right)^{-1/2}. \tag{4.29}$$

The vector $\mathbf{b}$ can be quickly obtained via the FMM with the kernel $\kappa(s,t) = \frac{1}{(s-t)^2}$. $\hat{Q}$ is a Cauchy-like matrix which gives a structured form of the eigenvectors. The FMM with the kernel $\kappa(s,t) = \frac{1}{s-t}$ can be used to quickly multiply $\hat{Q}$ to a vector.

Again, with the same reasons as before, it is challenging to stably apply the standard FMM to accelerate operations like the evaluations of $\log \mathbf{v}$ in (4.27) and $\mathbf{b}$ in (4.29) and the application of $\hat{Q}$ to a vector. On the other hand, just like the discussions in Section 4.3.2, we can integrate the local shifting strategy into FMM accelerations, just with appropriate kernels $\kappa(s,t)$. For example, with the gaps η_k solved from the shifted secular equation solution, it is preferred to use $\delta_{ik} - \eta_k$ in place of $d_i - \lambda_k$ in the computation of some entries of $\mathbf{q}_k$ for accuracy purpose [5, 9, 18, 24] when d_i and λ_k are very close. Note that, with δ_{jk} in (4.16), (4.24) can be written as

$$\mathbf{q}_k = \begin{pmatrix} \frac{\hat{v}_1}{\delta_{1k}-\eta_k} & \cdots & \frac{\hat{v}_k}{-\eta_k} & \cdots & \frac{\hat{v}_n}{\delta_{nk}-\eta_k} \end{pmatrix}^T. \tag{4.30}$$

When an entry of $\mathbf{q}_k$ belongs to a near-field block of $\hat{Q}$, its representation in (4.30) is used. Otherwise, we use its form in (4.24). This preserves the far-field rank structure.

Thus, FMM accelerations with local shifting can be used to reliably represent and apply $\hat{Q}$ or $\hat{Q}^T$. Note that

$$\hat{Q} = \mathrm{diag}(\hat{\mathbf{v}}) \left(\frac{1}{d_i - \lambda_j} \right)_{n \times n} \mathrm{diag}(\mathbf{b}), \tag{4.31}$$

so that $\hat{Q}$ can be stored just via five vectors:

$$\hat{\mathbf{v}},\ \mathbf{b},\ \mathbf{d} \equiv (\ d_1 \quad \cdots \quad d_n\ )^T,\ \boldsymbol{\lambda} \equiv (\ \lambda_1 \quad \cdots \quad \lambda_n\ )^T,\ \boldsymbol{\eta} \equiv (\ \eta_1 \quad \cdots \quad \eta_n\ )^T. \tag{4.32}$$

Here, we have the storage of one more vector $\boldsymbol{\eta}$ than that in [44]. This only slightly increase the storage, but the stability is significantly enhanced.

4.5. Overall eigendecomposition and structure of the eigenmatrix. The overall conquering framework is similar to [44], but with all the new stability strategies integrated. The conquering process is performed following the postordered traversal of the HSS tree $\mathcal{T}$ of A, where at each node $i \in \mathcal{T}$, a local eigenproblem is solved. For a leaf node i, suppose $\hat{D}_i$ is the (small) diagonal generator resulting from the overall dividing process. We just compute the dense eigenproblem $\hat{D}_i = Q_i \Lambda_i Q_i^T$. Then Q_i is a *local eigenmatrix* associated with i.

For a non-leaf node p with children i and j, the local eigenproblem is to find an eigendecomposition like in (2.10) based on (2.6) and (2.7). However, unlike (2.9) where a diagonal plus low-rank update eigendecomposition is computed, it is *necessary to reorder* the diagonal entries of $\mathrm{diag}(\Lambda_i, \Lambda_j)$ in order to explore the FMM structures that rely on the locations of the eigenvalues. Let P_p represent a sequence of permutations for deflation and for ordering the diagonal entries of $\mathrm{diag}(\Lambda_i, \Lambda_j)$ from the smallest to the largest. Also let the eigendecomposition of the *permuted* diagonal plus low-rank update problem be

$$P_p[\mathrm{diag}(\Lambda_i, \Lambda_j) + \hat{Z}_p \hat{Z}_p^T] P_p^T = \hat{Q}_p \Lambda_p \hat{Q}_p^T, \tag{4.33}$$

where $\hat{Z}_p$ is given in (2.8). Write D_p in (2.7) as $\hat{D}_p$ since D_p is likely updated after the hierarchical dividing process. Then we have the following eigendecomposition:

$$\hat{D}_p = Q_p \Lambda_p Q_p^T, \quad \text{with} \quad Q_p = \mathrm{diag}(Q_i, Q_j) P_p^T \hat{Q}_p, \tag{4.34}$$

where Q_i and Q_j are eigenmatrices of $\hat{D}_i$ and $\hat{D}_j$ obtained in steps i and j, respectively. Then the conquering process proceeds similarly.

Here for convenience, we say Q_p is a *local eigenmatrix* and $\hat{Q}_p$ is an *intermediate eigenmatrix*. The difference between the two is that a local eigenmatrix is an eigenmatrix of a local HSS block while the latter is an eigenmatrix of a diagonal plus low-rank update problem. A local eigenmatrix is formed by a sequence of intermediate eigenmatrices. Since $\hat{Q}_p \Lambda_p \hat{Q}_p^T$ in (4.33) is obtained by solving r consecutive rank-1 update eigenproblems, the intermediate eigenmatrix $\hat{Q}_p$ is the product of r Cauchy-like matrices like in (4.28). Of course, when FMM accelerations and deflation are applied, the eigendecomposition is approximate.

Then the overall eigenmatrix Q is given in terms of all the intermediate eigenmatrices, organized with the aid of the tree $\mathcal{T}$. Here, we give an accurate description of its structure as follows.

LEMMA 4.1. *Assemble all the intermediate eigenmatrices and permutation matrices corresponding to the nodes at a level l of $\mathcal{T}$ as*

$$Q^{(l)} = \mathrm{diag}(\hat{Q}_i,\ i\text{: at level } l \text{ of } \mathcal{T}), \quad P^{(l)} = \mathrm{diag}(P_i,\ i\text{: at level } l \text{ of } \mathcal{T}). \tag{4.35}$$

Then the final eigenmatrix Q has the form (illustrated in Figure 4.1)

$$Q = Q^{(L)} \prod_{l=l_{\max}-1}^{0} (P^{(l)} Q^{(l)}), \tag{4.36}$$

where level $l_{\max}$ is the leaf level of $\mathcal{T}$ and $\mathrm{root}(\mathcal{T})$ is at level 0. In addition, Q also corresponds to (4.34) with p set to be $\mathrm{root}(\mathcal{T})$.

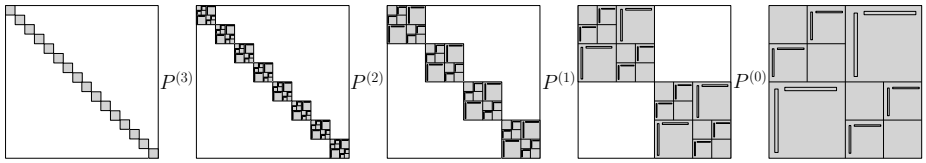


FIG. 4.1. *Illustration of the structure of the eigenmatrix Q, where $l_{\max} = 4$ and each structured diagonal block (marked in gray) is for an intermediate eigenmatrix $\hat{Q}_p$ associated with a nonleaf node p.*

Thus, Q can be understood in terms of either (4.36) or the local eigenmatrices. Lemma 4.1 gives an efficient way to apply Q or Q^T to a vector, where the triangular FMM with local shifting is again used to multiply the intermediate eigenmatrices with vectors. Note that with a very similar procedure, a local eigenmatrix Q_i or its transpose can be conveniently applied to a vector. Such an application process is used to multiply the local eigenmatrices Q_i^T and Q_j^T to Z_p as in (2.8) to quickly form $\hat{Z}_p$ used in (4.33).

The main algorithms used in SuperDC are shown in the supplementary materials. When A is given in terms of an HSS form with HSS rank r, the total complexity for computing the eigendecomposition (1.1) can be counted following [44, Section 3.1] and is $O(r^2 n \log^2 n)$. Note that the use of all the new stability techniques here does not change the overall complexity. Every local eigenmatrix $\hat{Q}_i$ is represented by a sequence of r Cauchy-like matrices like in (4.28). Each such a Cauchy-like matrix is stored with the aid of five vectors like in (4.32). The storage for Q is then $O(rn \log n)$ and the cost to apply Q or Q^T to a vector is $O(rn \log n)$ as in [44].

5. Numerical experiments. In this section, we perform a comprehensive test of the SuperDC eigensolver with different types of matrices and demonstrate its efficiency and accuracy. We compare SuperDC with the following methods.

- BandDC: a usual divide-and-conquer eigensolver that takes advantage of banded structures following the framework in [3]. A sequence of rank-1 updating problems is obtained based on the banded form in each dividing step and is then solved in the conquering stage. The same deflation tolerances as SuperDC are used.
- HSSBIS: an HSS bisection eigensolver [48] that takes advantage of fast HSS LDL factorization update for inertia evaluations. The stopping criterion of bisection is the same as the deflation tolerance of SuperDC.
- eig: the highly optimized Matlab eig function.

In order to run comparisons for larger matrix sizes in Matlab, we use BandDC and HSSBIS to compute only the eigenvalues (with accuracies comparable to those from SuperDC), which also gives them advantages over SuperDC. For HSSBIS, we decide the initial search region with $\tilde{\rho}(A) \equiv \sqrt{\|A\|_1 \|A\|_\infty} \geq \|A\|_2$ as an estimate of

the spectral radius of A. We use the following accuracy measurements:

$$\gamma = \max_{1\le k\le n} \frac{\|A\mathbf{q_k}-\lambda_k\mathbf{q}_k\|_2}{\sqrt{n}\|A\|_2} \quad \text{(residual)},$$
$$\theta = \max_{1\le k\le n} \frac{\|Q^T\mathbf{q}_k-\mathbf{e}_k\|_2}{\sqrt{n}} \quad \text{(loss of orthogonality)},$$
$$\delta_{\rm s} = \frac{\|\boldsymbol{\lambda}-\boldsymbol{\lambda}^*\|_2}{\|\boldsymbol{\lambda}^*\|_2}, \quad \delta_\infty = \frac{\|\boldsymbol{\lambda}-\boldsymbol{\lambda}^*\|_\infty}{\|\boldsymbol{\lambda}^*\|_\infty}, \quad \delta_{\rm m} = \max_{1\le k\le n} \frac{|\lambda_k^*-\lambda_k|}{|\lambda_k^*|} \quad \text{(errors)},$$

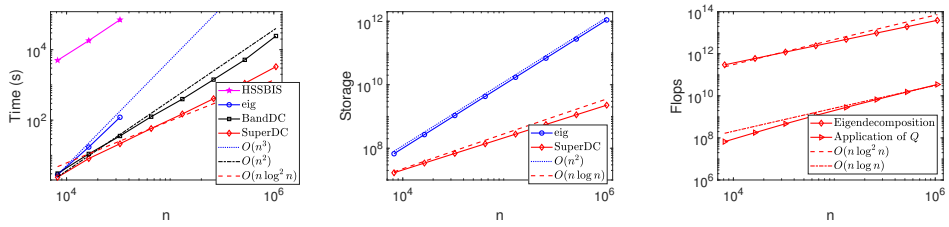
where $\boldsymbol{\lambda}^* = \begin{pmatrix}\lambda_1^* & \cdots & \lambda_n^*\end{pmatrix}^T$ are eigenvalues from eig and are considered as the exact results.

To measure the efficiency, we count the *flops* (total number of floating point arithmetic operations), the *storage* (total number of nonzeros to store the structured eigenmatrix in SuperDC or the dense eigenmatrix in eig), and the *timing* (seconds elapsed when the call of an eigensolver routine is completed). In the flop count, if a built-in routine is used to perform standard operations, we use known flop counts like those given in relevant references such as [17, 39].

SuperDC is available at https://www.math.purdue.edu/~xiaj. It is implemented fully in Matlab. The triangular FMM routine of SuperDC is developed based on a code used in [10], and its accuracy during each call is set to reach full machine precision. In all the tests, the leaf-level diagonal block size of the HSS forms is 2048. The tests are performed with four 2.60GHz cores and 80GB memory on a node at a cluster of Purdue RCAC. The request of 80GB memory is just to accommodate the need of eig for larger matrices.

EXAMPLE 1. First, we consider a symmetric tridiagonal matrix A. For our SuperDC eigensolver, the HSS representation of A can be explicitly written out without any extra cost and its HSS rank is $r = 2$ [49]. (The HSS structure does not rely on the actual nonzero entries, which are 3 on the main diagonal and -1 on the first superdiagonal and subdiagonal. Other numbers such as random ones are also tested with similar performance observed.) The size n of A in the test ranges from 8192 to $1,048,576$. We use $\tau = 10^{-10}$ in the deflation criterion in Section 4.1.

The timing of BandDC, HSSBIS, eig, and SuperDC is reported in Figure 5.1(a). The storage for the eigenmatrix Q from eig and SuperDC is given in Figure 5.1(b). The costs of SuperDC are given in Figure 5.1(c), in terms of the flops to get the eigendecomposition and the flops to apply Q to a vector. SuperDC achieves roughly linear complexity in the timing, flops, and storage. Both BandDC and HSSBIS follow quadratic trends in timing, though HSSBIS is quite slower. eig has a cubic trend in timing and an obvious quadratic storage (which is just n^2 for storing the dense Q).



(a) Eigendecomposition timing (b) Storage (c) Flops of SuperDC

FIG. 5.1. Example 1. *Timing, storage, and flops.*

SuperDC is faster than BandDC and HSSBIS for all the tested sizes, and its breakeven point with eig is around $n = 8192$. With $n = 32,768$, SuperDC is already about 6 times as fast as eig and takes only about 6% of the storage for the eigenmatrix. Note that eig runs out of memory for larger n due to the dense eigenmatrix, while SuperDC takes much less memory and can reach much larger n.

The conquering stage is usually much more time-consuming than the dividing stage. For example, for $n = 65,536$, the dividing stage of SuperDC needs just 1.4 seconds and the conquering stage takes 56.4 seconds. Thus, our strategy for minimizing colsize(Z_p) or reducing the number of rank-one updates is beneficial for the efficiency of the eigensolver since it reduces the amount of work in the conquering stage.

Table 5.1 shows the accuracy of SuperDC. The eigenvalues and eigendecompositions are computed accurately with numerically orthogonal eigenvectors. BandDC and HSSBIS reach comparable accuracies which are then not reported.

TABLE 5.1

Example 1. *Accuracy of SuperDC, where some errors are not reported since* eig *runs out of memory, and γ and θ are not available for $n \geq 262,144$ since they take too long to compute.*

n	$4,096$	$8,192$	$16,384$	$32,768$	$65,536$	$131,072$
γ	$1.2e-15$	$6.4e-15$	$1.1e-13$	$9.4e-14$	$7.5e-14$	$5.3e-14$
θ	$1.8e-14$	$2.9e-14$	$3.8e-14$	$5.5e-14$	$8.6e-14$	$1.2e-13$
δ_{s}	$2.6e-16$	$4.6e-16$	$1.3e-13$	$9.4e-14$		
δ_{∞}	$8.9e-16$	$1.2e-14$	$8.0e-12$	$6.3e-12$		
δ_{m}	$9.7e-16$	$1.2e-14$	$8.0e-12$	$6.3e-12$		

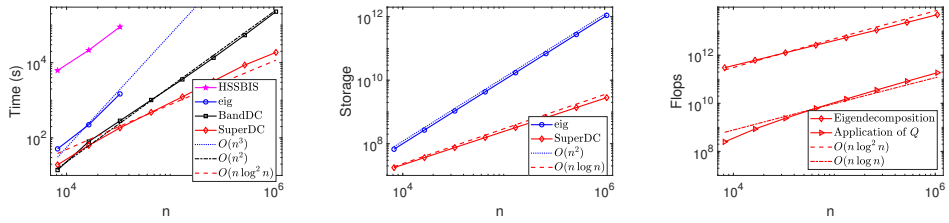
EXAMPLE 2. In this example, we test a symmetric matrix A which is sparse and nearly banded. A has a banded form with half bandwidth 5 together with some nonzero entries away from the band. The HSS form for A can be explicitly written out with the method in [49] and has HSS rank 10. The nonzero entries away from the band are introduced by modifying some HSS generators. The main diagonal entries are set as 30 and the other entries in the band are set as -10 so that the upper bound for all $\|B_k\|_2$ in Proposition 3.1 is $\beta = 35.1 \gg 1$. The size n in the test ranges from 8192 to $1,048,576$. We use $\tau = 10^{-10}$ in the deflation criterion.

The entries away from the band break the banded structure of A. Still, BandDC can be conveniently adapted to A. The efficiency benefit of SuperDC over eig becomes even more significant, as shown in Figure 5.2. At $n = 32,678$, SuperDC is about 8 times as fast as eig and takes only about 7% of the storage for the eigenmatrix. Again, eig runs out of memory when n increases, but SuperDC works for much larger n and demonstrates nearly linear complexity. At $n = 1,048,576$, SuperDC is about 12 times as fast as BandDC (which does not even compute the eigenmatrix).

We also show the advantage of our new dividing strategy over the original one (2.2). In Table 5.2, we show the norm growth of the B, D generators after the dividing stage. For the initial B, D generators of the original HSS form, let $\tilde{B}, \tilde{D}$ denote the updated generators after the entire dividing stage is finished. Let

$$\rho_B = \max_{i<\mathrm{root}(\mathcal{T})} \|B_i\|_2, \quad \rho_D = \max_{i:\ \mathrm{leaf}} \|D_i\|_2, \quad \rho_{\tilde{B}} = \max_{i<\mathrm{root}(\mathcal{T})} \|\tilde{B}_i\|_2, \quad \rho_{\tilde{D}} = \max_{i:\ \mathrm{leaf}} \|\tilde{D}_i\|_2.$$

When n increases, the number of levels in the HSS tree $\mathcal{T}$ increase and $\rho(D)$ and $\rho(B)$ stay about the same for all n. However, $\rho_{\tilde{B}}$ and $\rho_{\tilde{D}}$ grow exponentially with the original dividing stage, as predicted by Proposition 3.1. This poses a stability risk.



(a) Eigendecomposition timing (b) Storage (c) Flops of SuperDC

FIG. 5.2. Example 2. *Timing, storage, and flops.*

(Note that $\rho_{\tilde{D}}$ has a larger magnitude than $\rho_{\tilde{B}}$, which is consistent with Proposition 3.1.) In contrast, the growth of $\rho_{\tilde{D}}$ and $\rho_{\tilde{B}}$ with our new dividing strategy is much slower and roughly follows the linear growth pattern as predicted by Proposition 3.2.

TABLE 5.2
Example 2. *Norms of the D, B generators before and after the dividing stage.*

n		4,096	8,192	16,384	32,768	65,536	131,072
Number of levels $l_{\max}$		2	3	4	5	6	7
Initial	ρ_B	3.5e1	3.5e1	3.5e1	3.5e1	3.5e1	3.5e1
	ρ_D	7.0e1	7.0e1	7.0e1	7.0e1	7.0e1	7.0e1
After the original	$\rho_{\tilde{B}}$	3.5e1	7.4e2	5.5e5	3.0e11	9.2e22	8.5e45
dividing strategy	$\rho_{\tilde{D}}$	1.7e3	1.1e6	6.1e11	1.8e23	1.7e46	1.4e92
After the new	$\rho_{\tilde{B}}$	3.5e1	4.5e1	8.3e1	1.6e2	3.3e2	6.6e2
dividing strategy	$\rho_{\tilde{D}}$	7.3e1	1.3e2	3.0e2	6.3e2	1.3e3	2.6e3

In Table 5.3, we report the accuracies of SuperDC. The accuracies associated with the original dividing strategy deteriorate as n gets larger and the results are highly inaccurate for $n \geq 32,768$ so they are not shown. In contrast, the new dividing strategy yields nice accuracies. This accuracy difference can be understood as follows. For a leaf node k, we need to use a (backward stable) dense method to compute a numerical eigendecomposition of the updated $\tilde{D}$ generators (see Lemma 2.1) with backward error $\Delta\tilde{D}_k$ (see, e.g., [24]):

$$\tilde{D}_k = Q_k \Lambda_k Q_k^T + \Delta\tilde{D}_k, \quad \text{with} \quad \|\Delta\tilde{D}_k\|_2 = O(\|\tilde{D}_k\|_2 \epsilon_{\text{mach}}). \tag{5.1}$$

By Propositions 3.1 and 3.2, $\|\Delta\tilde{D}_k\|_2$ is roughly in the magnitude of $O(\beta^{n/2}\epsilon_{\text{mach}})$ with the original dividing strategy (2.2), or $O(\frac{n}{2}\epsilon_{\text{mach}})$ with the new dividing strategy. Therefore, the original dividing strategy will likely introduce larger errors.

Remark 5.1. In (5.1), we often have $\|\tilde{D}_k\|_2$ in the magnitude of $O(\|A\|_2)$ so that $O(\|\tilde{D}_k\|_2\epsilon_{\text{mach}})$ is roughly $O(\|A\|_2\epsilon_{\text{mach}})$. Therefore, an absolute error of order $O(\|A\|_2\epsilon_{\text{mach}})$ is introduced to the eigenvalues, resulting in loss of digits for those eigenvalues that are tiny (when $\|A\|_2$ is large). Thus, the maximum relative errors δ_{m} for those tiny eigenvalues may be larger, while other measurements such as δ_{s} and γ are still well controlled. In addition, because of the complex nature of SuperDC, it is possible that some other weakness in the numerical stability may still present. Our ongoing work is to attempt to perform a comprehensive backward stability study

TABLE 5.3
Example 2. *Accuracy of SuperDC, where some errors are not reported since* eig *runs out of memory, and γ and θ are not available for $n \geq 262,144$ since they take too long to compute.*

	n	$4,096$	$8,192$	$16,384$	$32,768$	$65,536$	$131,072$
Original dividing	γ	$3.6e-13$	$9.5e-12$	$1.5e-5$			
	θ	$2.2e-13$	$3.0e-13$	$5.3e-13$			
	δ_{s}	$4.7e-13$	$4.2e-13$	$9.7e-7$			
	δ_{∞}	$1.7e-11$	$1.6e-11$	$3.0e-5$			
	δ_{m}	$1.8e-11$	$1.6e-9$	$1.2e-4$			
New dividing	γ	$9.4e-14$	$1.8e-12$	$3.7e-13$	$3.4e-13$	$5.6e-13$	$1.2e-12$
	θ	$1.9e-13$	$4.4e-13$	$5.6e-13$	$1.1e-12$	$1.4e-12$	$2.0e-12$
	δ_{s}	$1.6e-14$	$3.2e-12$	$5.3e-13$	$2.6e-13$		
	δ_{∞}	$6.0e-13$	$1.5e-10$	$2.2e-11$	$1.1e-11$		
	δ_{m}	$1.6e-12$	$2.9e-10$	$3.2e-11$	$2.2e-11$		

based on the stability results for rank-1 updated eigenvalue solution in [24] and for the structured methods in [10].

We also demonstrate the importance of our local shifting strategy by testing the eigensolver with triangular FMM accelerations applied to the standard secular equation. Due to cancellations, Matlab returns `NaN` (not-a-number) for the test matrices with sizes larger than 8192. This shows the risk of directly applying FMM accelerations to the standard secular equation like in [44].

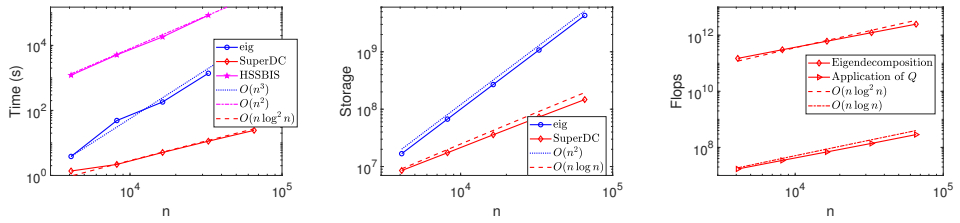
EXAMPLE 3. Next, we consider a dense symmetric Toeplitz matrix A with its first row $\boldsymbol{\xi} = \begin{pmatrix} \xi_1 & \cdots & \xi_n \end{pmatrix}$ given by

$$\xi_1 = 2\alpha, \quad \xi_j = \frac{\sin(2\alpha(j-1)\pi)}{(j-1)\pi}, \ j = 2, 3, \ldots, n,$$

where $0 < \alpha < 1/2$. This is the so-called prolate matrix that appears frequently in signal processing. It is known to be extremely ill-conditioned and has special spectral properties (see, e.g., [43]). In fact, the prolate matrix has many small eigenvalues of magnitude $O(\epsilon_{\mathrm{mach}})$. Here, we set $\alpha = \frac{1}{4}$. It is known that any Toeplitz matrix can be converted into a Cauchy-like matrix $\mathcal{C}$ which has small off-diagonal numerical ranks [14, 33, 44]. That is, $\mathcal{C} = \mathcal{F}A\mathcal{F}^*$, where $\mathcal{F}$ is the normalized inverse DFT matrix. Then the eigendecomposition of A can be done via that of $\mathcal{C}$. An HSS approximation to $\mathcal{C}$ may be quickly constructed based on randomized methods in [30, 31, 51, 55] and fast Toeplitz matrix-vector multiplications. The cost is nearly linear in n. Here, we use a tolerance 10^{-10} in relevant compression steps, which is the same as the deflation tolerance τ. The size n ranges from 4096 to $65,536$.

SuperDC and HSSBIS are applied to the resulting HSS form and compared with eig applied to A. In Figure 5.3, the timing, storage, and flops are shown and they are consistent with the complexity estimates. SuperDC shows a significant efficiency advantage over eig. At $n = 32,768$, SuperDC is about 122 times faster than eig. The accuracy is reported in Table 5.4.

One thing to point out is that the theoretical complexity $O(r^2 n \log^2 n)$ of SuperDC may overestimate the actual cost. For example, here r is typically known to be $O(\log n)$ based on entrywise approximation errors [10, 33, 52, 55]. One reason for the overestimate is that the flop count does not take into consideration a levelwise rank



(a) Eigendecomposition timing (b) Storage (c) Flops of SuperDC

FIG. 5.3. Example 3. *Timing, storage, and flops.*

TABLE 5.4

Example 3. *Accuracy of SuperDC, where the errors δ_s and δ_∞ for $n = 65,536$ are not reported since* eig *runs out of memory. δ_m is not available since many eigenvalues of order $O(\epsilon_{mach})$ and* eig *returns a numerical eigenvalue* 0 *for some of the matrices.*

n	$4,096$	$8,192$	$16,384$	$32,768$	$65,536$
γ	$2.3e-11$	$4.4e-11$	$1.8e-10$	$3.5e-10$	$1.4e-9$
θ	$2.2e-15$	$8.6e-15$	$6.0e-15$	$4.2e-15$	$3.0e-15$
δ_s	$5.5e-12$	$1.4e-11$	$4.5e-10$	$9.3e-10$	
δ_∞	$1.1e-10$	$7.3e-10$	$2.2e-8$	$6.1e-8$	

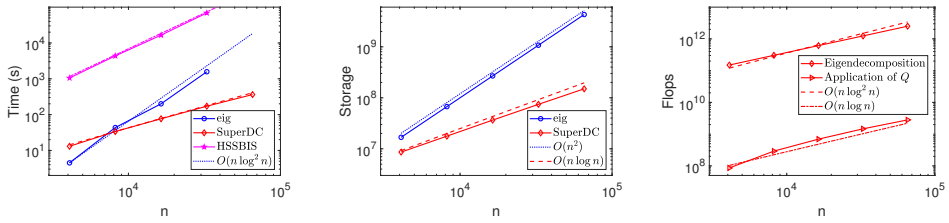
pattern in [50]. Another reason is our flexible deflation strategy in Section 4.1. The matrices actually have highly clustered intermediate eigenvalues and many of them get deflated. This further leads to high efficiency gain.

We have also tested SuperDC on random Toeplitz matrices, where the associated Cauchy-like matrices $\mathcal{C}$ have off-diagonal numerical ranks r quite larger than in the prolate matrix case. Since the complexity of SuperDC is $O(r^2 n \log^2 n)$, it needs larger n to see an obvious advantage in timing over eig. (Of course, we may also use a larger compression tolerance to get smaller r.) In addition, for random Toeplitz matrices, deflation happens much less frequently than in the prolate matrix case.

EXAMPLE 4. The last example is a discretized kernel matrix A in [12] which is the evaluation of the function $\sqrt{|s-t|}$ at the Chebyshev points $\cos\left(\frac{2i-1}{2n}\pi\right), i = 1, 2, \ldots, n$. The HSS construction may be based on direct off-diagonal compression or efficient analytical methods like in [56]. We use an existing routine based on the former one for simplicity. To show the flexibility of accuracy controls, we aim for moderate accuracy in this test by using a compression tolerance 10^{-6} in the HSS construction, which is also the deflation tolerance.

For this example, we can observe similar complexity results as in the previous examples. See Figure 5.4. With the larger tolerance than in the previous examples, we still achieve reasonable eigenvalue errors and residuals with numerically orthogonal eigenvectors. See Table 5.5.

We now show how our local shifting strategy (for triangular FMM-accelerated solution of the shifted secular equations) can also significantly benefit the rate of convergence of the roots. To illustrate this, we perform the following count. Suppose r secular equations are solved because of r rank-1 updates associated with the root node of the HSS tree $\mathcal{T}$. When solving the jth secular equation, let μ_j be the percentage of eigenvalues that have *not* converged after 5 Newton's iterations. Let $\mu = \max_{1\le j\le r} \mu_j$.



(a) Eigendecomposition timing (b) Storage (c) Flops of SuperDC

FIG. 5.4. Example 4. *Timing, storage, and flops.*

TABLE 5.5

Example 4. *Accuracy of SuperDC, where the errors for* $n = 65,536$ *are not reported since* eig *runs out of memory.*

n	$4,096$	$8,192$	$16,384$	$32,768$	$65,536$
γ	$7.4e-9$	$2.7e-9$	$2.2e-9$	$1.6e-9$	$1.1e-9$
θ	$1.4e-13$	$2.6e-13$	$2.5e-13$	$2.4e-13$	$3.8e-13$
$\delta_{\rm s}$	$3.8e-8$	$5.5e-8$	$5.7e-8$	$8.7e-8$	
δ_{∞}	$2.9e-8$	$3.2e-8$	$2.8e-8$	$5.6e-8$	
$\delta_{\rm m}$	$1.2e-4$	$4.2e-4$	$4.7e-4$	$5.8e-4$	

Table 5.6 reports this maximum percentage μ with varying n. With local shifting, a vast majority of those eigenvalues (about 99% or more) converges within 5 iterations. This is significantly better than the case without local shifting (i.e., when the standard secular equation is solved with FMM accelerations).

TABLE 5.6

Maximum percentage (μ) *of eigenvalues not converged within* 5 *iterations for solving the* r *secular equations associated with* root($\mathcal{T}$).

n	$4,096$	$8,192$	$16,384$	$32,768$	$65,536$
With local shifting	1.00%	0.88%	0.34%	0.38%	0.33%
Without local shifting	62.5%	57.6%	57.2%	58.5%	57.7%

6. Conclusions. In this work, we have designed a SuperDC eigensolver that significantly improves a previous development in terms of the stability and efficiency. A series of stability enhancements is built into the different stages of the algorithm. In particular, we avoid an exponential norm growth risk in the dividing stage via a balancing strategy. We further combine FMM accelerations with several key stability safeguards that have been used in practical divide-and-conquer algorithms. The extensive numerical tests confirm the efficiency and the accuracy.

The SuperDC eigensolver makes it feasible to use full eigendecompositions to solve various challenging numerical problems as mentioned at the beginning of the paper. A list of applications is expected to be included in [35]. In addition, we expect that the novel local shifting strategy and triangular FMM accelerations are also useful for other FMM-related matrix computations when stability and accuracy are crucial. In our future work, we plan to provide the proof of backward stability, as well as a high-performance parallel implementation, which will extend the applicability of the

algorithm to large-scale numerical computations.

Acknowledgements. Thank the editor and the three anonymous reviewers for providing valuable suggestions that have helped greatly improve the paper.

REFERENCES

[1] S. Ambikasaran and E. Darve, *An $O(n \log n)$ fast direct solver for partial hierarchically semi-separable matrices*, J. Sci. Comput., 57 (2013), pp. 477–501.
[2] P. Amestoy, C. Ashcraft, O. Boiteau, A. Buttari, J.-Y. L'Excellent, and C. Weisbecker, *Improving multifrontal methods by means of block low-rank representations*, SIAM J. Sci. Comp., 37 (2015), pp. A1451–A1474.
[3] P. Arbenz, *Divide and conquer algorithms for the bandsymmetric eigenvalue problem*, Parallel Comput., 18(10), 1105–1128 (1992).
[4] A. Cortinovis, D. Kressner and S. Massei, *Divide and conquer methods for functions of matrices with banded or hierarchical low-rank structure*, arXiv:2107.04337 (2021).
[5] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen, *LAPACK Users' Guide*, SIAM, Philadelphia, PA, third ed., 1999.
[6] 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.
[7] 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.
[8] D. Bini and V. Y. Pan, *Parallel complexity of tridiagonal symmetric eigenvalue problem*, Proc. Annu. ACM-SIAM Symp. Discrete Algorithms, SIAM, Philadelphia, 1991 pp. 384–393.
[9] J. R. Bunch, C. P. Nielsen, and D. C. Sorensen, *Rank-one modification of the symmetric eigenproblem*, Numer. Math., 31 (1978), pp. 31–48.
[10] D. Cai and J. Xia, *A stable matrix version of the fast multipole method: stabilization strategies and examples*, Electron. Trans. Numer. Anal., 54 (2021), pp. 581-609.
[11] 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.
[12] 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.
[13] 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.
[14] 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.
[15] S. Chandrasekaran, M. Gu, J. Xia, and J. Zhu, *A fast QR algorithm for companion matrices*, Oper. Theory: Adv. Appl., Birkhaeuser Basel, 179 (2007), pp. 111–143.
[16] J. J. M. Cuppen, *A divide and conquer method for the symmetric tridiagonal eigenproblem*, Numer. Math., 36 (1981), pp. 177–195.
[17] J. W. Demmel, *Applied Numerical Linear Algebra*, SIAM, 1997.
[18] J. J. Dongarra and D. C. Sorensen, *A fully parallel algorithm for the symmetric eigenvalue problem*, SIAM J. Sci. Stat. Comput., 8 (1987), s139–s154.
[19] 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.
[20] Y. Eidelman and I. Haimovici, *Divide and conquer method for eigenstructure of quasiseparable matrices using zeroes of rational matrix functions*, Oper. Theory: Adv. Appl., 218 (2012), Springer, Basel, pp. 299–328.
[21] Y. Eidelman, I. Gohberg, and I. Haimovici, *Separable type representations of matrices and fast algorithms*, Vol. 1, 2, Oper. Theory: Adv. Appl., 234, 235.
[22] G. H. Golub, *Some modified matrix eigenvalue problems*, SIAM Rev., 15 (1973), pp. 318–334.
[23] L. Greengard and V. Rokhlin, *A fast algorithm for particle simulations*, J. Comput. Phys., 73 (1987), pp. 325–348.
[24] M. Gu and S. C. Eisenstat, *A stable and efficient algorithm for the rank-one modification of the symmetric eigenproblem*, SIAM J. Matrix Anal. Appl., 15 (1994), pp. 1266-1276.
[25] M. Gu and S. C. Eisenstat, *A divide-and-conquer algorithm for the symmetric tridiagonal*

eigenproblem, SIAM J. Matrix Anal. Appl., 16 (1995), pp. 172–191.
[26] W. Hackbusch and S. Borm, *Data-sparse approximation by adaptive $\mathcal{H}^2$-matrices*, Computing, 69 (2002), pp.1–35.
[27] W. Hackbusch, *A sparse matrix arithmetic based on $\mathcal{H}$-matrices*, Computing, 62 (1999), pp. 89–108.
[28] R. C. Li, *Solving secular equations stably and efficiently*, University of California, Berkeley, Technical Report No. UCB/CSD-94-851 (1994).
[29] X. Liao, S. Li, L. Cheng, and M. Gu, *An improved divide-and-conquer algorithm for the banded matrices with narrow bandwidths*, Comput. Math. Appl., 71 (2016), pp. 1933–1943.
[30] X. Liu, J. Xia, and M. V. De Hoop, *Parallel randomized and matrix-free direct solvers for large structured dense linear systems*, SIAM J. Sci. Comput., 38 (2016), pp. S508–S538.
[31] 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.
[32] P. G. Martinsson and V. Rokhlin, *A fast direct solver for boundary integral equations in two dimensions*, J. Comput. Phys. 205 (2005), pp. 1–23.
[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. 741–752.
[34] D. P. O'Leary and G. W. Stewart, *Computing the eigenvalues and eigenvectors of symmetric arrowhead matrices*, J. Comput. Phys., 90 (1990), pp. 497–505.
[35] X. Ou, J. Vogel, J. Xia, and Z. Xin, *Efficient numerical computations via superfast eigenvalue decompositions*, preprint, 2021.
[36] J. Shen, Y. Wang, and J. Xia, *Fast structured direct spectral methods for differential equations with variable coefficients, I. The one-dimensional case*, SIAM J. Sci. Comput., 38 (2016), pp. A28–A54.
[37] X. Sun and N. P. Pitsianis, *A matrix version of the fast multipole method*, SIAM Rev., 43 (2001), pp. 289–300.
[38] A. Šušnjara and D. Kressner, *A fast spectral divide-and-conquer method for banded matrices*, Numer. Linear Algebra Appl., 28 (2021), e2365.
[39] L. N. Trefethen and D. Bau, *Numerical Linear Algebra*, SIAM, 1997.
[40] M. Tygert, *Recurrence relations and fast algorithms*, Appl. Comput. Harmon. Anal., 28.1 (2010), 121–128.
[41] 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.
[42] R. Vandebril, M. Van Barel, and N. Mastronardi, *Matrix Computations and Semiseparable Matrices*, Vol. 1. Johns Hopkins University Press, Baltimore, MD, 2008.
[43] J. M. Varah, *The prolate matrix*, Linear Algebra Appl., 187 (1993), pp. 269–278.
[44] J. Vogel, J. Xia, S. Cauley, and V. Balakrishnan, *Superfast divide-and-conquer method and perturbation analysis for structured eigenvalue solutions*, SIAM J. Sci. Comput., 38 (2016), pp. A1358–A1382.
[45] Y. Wang, *Fast Structured Spectral Methods*, Ph.D. thesis, Purdue University, 2017.
[46] Y. Xi and J. Xia, *On the stability of some hierarchical rank structured matrix algorithms*, SIAM J. Matrix Anal. Appl., 37 (2016), pp. 1279–1303.
[47] 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.
[48] 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.
[49] J. Xia, *Fast Direct Solvers for Structured Linear Systems of Equations*, Ph.D. thesis, University of California, Berkeley, 2006.
[50] J. Xia, *On the complexity of some hierarchical structured matrix algorithms*, SIAM J. Matrix Anal. Appl., 33 (2012), pp. 388–410.
[51] J. Xia, *Randomized sparse direct solvers*, SIAM J. Matrix Anal. Appl., 34 (2013), pp. 197–227.
[52] J. Xia, *Multi-layer hierarchical structures*, CSIAM Trans. Appl. Math., 2 (2021), pp. 263–296.
[53] J. Xia, S. Chandrasekaran, M. Gu, and X. S. Li, *Superfast multifrontal method for large structured linear systems of equations*, SIAM J. Matrix Anal. Appl., 31 (2009), pp. 1382–1411.
[54] 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.
[55] 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.
[56] X. Ye, J. Xia, and L. Ying, *Analytical low-rank compression via proxy point selection*, SIAM J. Matrix Anal. Appl., 41 (2020), pp. 1059–1085.

SUPPLEMENTARY MATERIALS: LIST OF MAJOR ALGORITHMS

Title of paper: *SuperDC: Superfast divide-and-conquer eigenvalue decomposition with improved stability for rank-structured matrices*

Authors: Xiaofeng Ou and Jianlin Xia

These supplementary materials are pseudocodes that can help better understand the major algorithms in the paper.

- Algorithm 1: the HSS dividing stage.
- Algorithm 2: solving the secular equation for the eigenvalues with triangular FMM accelerations and local shifting.
- Algorithm 3: the conquering stage for producing the eigendecomposition.
- Algorithm 4: application of the a local eigenmatrix Q_i or its transpose to a vector. This is used in Algorithm 3 and also can be used to apply the global eigenmatrix Q or its transpose to a vector when $i = \text{root}(\mathcal{T})$.

For notational convenience, we use r to represent the column sizes of all Z_i matrices in the pseudocodes. $\mathcal{T}_i$ is also used to denote the subtree of $\mathcal{T}$ rooted at node $i \in \mathcal{T}$. $Z(:, j)$ means the j-th column of Z.

The following utility routines are used in the algorithms. To save space, we are not showing pseudocodes for these routines.

- $\mathsf{updhss}(D_i, U_i, H)$: for an HSS block D_i corresponding to the subtree $\mathcal{T}_i$, update its D, B generators to get those of $D_i - U_i H U_i^T$ using Lemma 2.1.
- $\mathsf{trifmm}(\mathbf{d}, \mathbf{x}, \boldsymbol{y}, \mathbf{w}, \kappa)$: compute a matrix-vector product $K\mathbf{w}$ with the triangular FMM and local shifting as in Sections 4.2.2 and 4.3.2, where $K = (\kappa(d_i, x_j))_{d_i \in \mathbf{d}, x_j \in \mathbf{x}}$ is a kernel matrix and $\boldsymbol{y}$ is the gap vector (for accurately evaluating $\mathbf{x} - \mathbf{d}$). Note that the triangular FMM is used to multiply the lower triangular part of K with $\mathbf{w}$ and the strictly upper triangular part of K with $\mathbf{w}$ and the final result is the sum of the two products.
- $\mathsf{mnewton}(\boldsymbol{\psi}, \boldsymbol{\phi}, \boldsymbol{\psi}', \boldsymbol{\phi}')$: use the modified Newton's method to compute corrections to the current approximate gap as in (4.19), where $\boldsymbol{\psi}, \boldsymbol{\phi}, \boldsymbol{\psi}', \boldsymbol{\phi}'$ look like (4.9) and (4.10).
- $\mathsf{iniguess}(\mathbf{d}, \mathbf{w})$: compute the initial guess as in [28] for the solution of the secular equation (2.12).
- $\mathsf{deflate}(\mathbf{d}, \mathbf{v}, \tau)$: apply deflation with the criterion in Section 4.1.

Algorithm 1 SuperDC dividing stage

procedure $\mathsf{divide}(\{D_i\}_{i\in\mathcal{T}}, \{U_i\}_{i\in\mathcal{T}}, \{R_i\}_{i\in\mathcal{T}}, \{B_i\}_{i\in\mathcal{T}})$
  for node $i = \mathrm{root}(\mathcal{T}), \ldots, 1$ **do** ▷ *Dividing D_i in a top-down traversal*
    if i is a non-leaf node **then**
      if $\mathrm{colsize}(B_{c_1}) \le \mathrm{rowsize}(B_{c_1})$ **then** ▷ *c_1, c_2: children of i*
        $D_{c_1} \leftarrow \mathsf{updhss}(D_{c_1}, U_{c_1}, \frac{1}{\|B_{c_1}\|_2} B_{c_1} B_{c_1}^T)$ ▷ *Update generators of D_{c_1} to get those of $D_{c_1} - \frac{1}{\|B_{c_1}\|_2} U_{c_1} B_{c_1} B_{c_1}^T U_{c_1}^T$ like in Lemma 2.1*
        $D_{c_2} \leftarrow \mathsf{updhss}(D_{c_2}, U_{c_2}, \|B_{c_1}\|_2 I)$ ▷ *Update generators of D_{c_2} to get those of $D_{c_2} - \|B_{c_1}\|_2 U_{c_2} U_{c_2}^T$ like in Lemma 2.1*
      else
        $D_{c_1} \leftarrow \mathsf{updhss}(D_{c_1}, U_{c_1}, \|B_{c_1}\|_2 I)$ ▷ *Update generators of D_{c_1} to get those of $D_{c_1} - \|B_{c_1}\|_2 U_{c_1} U_{c_1}^T$ like in Lemma 2.1*
        $D_{c_2} \leftarrow \mathsf{updhss}(D_{c_2}, U_{c_2}, \frac{1}{\|B_{c_1}\|_2} B_{c_1}^T B_{c_1})$ ▷ *Update generators of D_{c_2} to get those of $D_{c_2} - \frac{1}{\|B_{c_1}\|_2} U_{c_2} B_{c_1}^T B_{c_1} U_{c_2}^T$ like in Lemma 2.1*
      end if
    end if
  end for
  for node $i = 1, \ldots, \mathrm{root}(\mathcal{T})$ **do** ▷ *Form Z_i in a bottom-up traversal*
    if i is a non-leaf node **then**
      if $\mathrm{colsize}(B_{c_1}) \le \mathrm{rowsize}(B_{c_1})$ **then** ▷ *c_1, c_2: children of i*
        $Z_i \leftarrow \begin{pmatrix} \frac{1}{\sqrt{\|B_{c_1}\|_2}} U_{c_1} B_{c_1} \\ \sqrt{\|B_{c_1}\|_2} U_{c_2} \end{pmatrix}$ ▷ *Local update Z matrix like in (3.12)*
      else
        $Z_i \leftarrow \begin{pmatrix} \sqrt{\|B_{c_1}\|_2} U_{c_2} \\ \frac{1}{\sqrt{\|B_{c_1}\|_2}} U_{c_2} B_{c_1}^T \end{pmatrix}$ ▷ *Local update Z matrix like in (3.15)*
      end if
      if $i \neq \mathrm{root}(\mathcal{T})$ **then**
        $U_i \leftarrow \begin{pmatrix} U_{c_1} R_{c_1} \\ U_{c_2} R_{c_2} \end{pmatrix}$ ▷ *Assemble U_i for parent node of i*
      end if
    end if
  end for
  return updated generators $\{D_i\}_{i\in\mathcal{T}}, \{B_i\}_{i\in\mathcal{T}}, \{Z_i\}_{i\in\mathcal{T}}$
end procedure

Algorithm 2 Secular equation solution for eigenvalues (of $\mathrm{diag}(\mathbf{d}) + \mathbf{v}\mathbf{v}^T$)

procedure secular($\mathbf{d}, \mathbf{v}$)
  ▷ *Eigenvalue solution via the solution of the shifted secular equation (4.15)*
  $\mathbf{w} \leftarrow \mathbf{v} \odot \mathbf{v}$
  $\mathbf{x}^{(0)} \leftarrow$ iniguess($\mathbf{d}, \mathbf{w}$) ▷ *Computation of the initial guess as in [28]*
  $\boldsymbol{y}^{(0)} \leftarrow \mathbf{x}^{(0)} - \mathbf{d}$
  for $j = 0, 1, \ldots$ **do**
    $[\boldsymbol{\psi}, \boldsymbol{\phi}] \leftarrow$ trifmm($\mathbf{d}, \mathbf{x}^{(j)}, \boldsymbol{y}^{(j)}, \mathbf{w}, \frac{1}{s-t}$) ▷ *Computation of $\boldsymbol{\psi}, \boldsymbol{\phi}$ in (4.9)*
    $[\boldsymbol{\psi}', \boldsymbol{\phi}'] \leftarrow$ trifmm($\mathbf{d}, \mathbf{x}^{(j)}, \boldsymbol{y}^{(j)}, \mathbf{w}, \frac{1}{(s-t)^2}$) ▷ *Computation of $\boldsymbol{\psi}', \boldsymbol{\phi}'$ in (4.10)*
    $\mathbf{f} \leftarrow \mathbf{e} + \boldsymbol{\psi} + \boldsymbol{\phi}$
    if $|\mathbf{f}| < cn(\mathbf{e} + |\boldsymbol{\psi}| + |\boldsymbol{\phi}|)\epsilon$ **then** ▷ *Stopping criterion*
      break
    end if
    $\Delta\mathbf{x}^{(j)} \leftarrow$ mnewton($\boldsymbol{\psi}, \boldsymbol{\phi}, \boldsymbol{\psi}', \boldsymbol{\phi}'$)
      ▷ *Computation of root update with modified Newton's method*
    $\boldsymbol{y}^{(j+1)} \leftarrow \boldsymbol{y}^{(j)} + \Delta\mathbf{x}^{(j)}$ ▷ *Updated gap approximation as in (4.19)*
    $\mathbf{x}^{(j+1)} \leftarrow \boldsymbol{y}^{(j+1)} + \mathbf{d}$ ▷ *Updated eigenvalue approximation*
  end for
  $\boldsymbol{\lambda} \leftarrow \mathbf{x}^{(j)}$, $\boldsymbol{\eta} \leftarrow \boldsymbol{y}^{(j)}$ ▷ *Eigenvalue and gap upon convergence*
  return $\boldsymbol{\lambda}, \boldsymbol{\eta}$
end procedure

Algorithm 3 SuperDC conquering stage

procedure conquer($\{D_i\}_{i\in\mathcal{T}}, \{U_i\}_{i\in\mathcal{T}}, \{R_i\}_{i\in\mathcal{T}}, \{B_i\}_{i\in\mathcal{T}}, \{Z_i\}_{i\in\mathcal{T}}, \tau$)
▷ *The D_i, B_i generators have been updated in the dividing stage*
  for node $i = 1, \ldots, \text{root}(\mathcal{T})$ **do** ▷ *Conquering in a postordered traversal*
    if i is a leaf node **then** ▷ *Leaf-level eigendecomposition*
      $(\boldsymbol{\lambda}_i, \hat{Q}_i) \leftarrow \mathsf{eig}(D_i)$ ▷ *Via Matlab* eig *function*
    else
      $\begin{pmatrix} Z_{i,1} \\ Z_{i,2} \end{pmatrix} \leftarrow Z_i$ ▷ *Partitioning following the sizes of D_{c_1} and D_{c_2}*
      $Z_{i,1} \leftarrow \mathsf{superdcmv}(Q_{c_1}, Z_{i,1}, 1)$ ▷ $Q_{c_1}^T Z_{i,1}$
      $Z_{i,2} \leftarrow \mathsf{superdcmv}(Q_{c_2}, Z_{i,2}, 1)$ ▷ $Q_{c_2}^T Z_{i,2}$
      $Z_i \leftarrow \begin{pmatrix} Z_{i,1} \\ Z_{i,2} \end{pmatrix}$ ▷ *$\hat{Z}_i$ like in (2.8)*
      $[\boldsymbol{\lambda}_i^{(0)}, P_i] \leftarrow \mathsf{sort}(\boldsymbol{\lambda}_{c_1}, \boldsymbol{\lambda}_{c_2})$ ▷ *Ordering of all the diagonal entries of $\boldsymbol{\lambda}_{c_1}, \boldsymbol{\lambda}_{c_2}$ together, with P_i the permutation matrix*
      for $j = 1, 2 \ldots, r$ **do** ▷ $r = \text{colsize}(Z_i)$
        $[\mathbf{d}_i^{(j)}, Z_i(:,j)] \leftarrow \mathsf{deflate}(\boldsymbol{\lambda}_i^{(j-1)}, Z_i(:,j), \tau)$ ▷ *Deflation (Section 4.1)*
        $[\boldsymbol{\lambda}_i^{(j)}, \boldsymbol{\eta}_i^{(j)}] \leftarrow \mathsf{secular}(\mathbf{d}_i^{(j)}, Z_i(:,j))$ ▷ *Secular equation solution*
        $\mathbf{v}_1 \leftarrow \mathsf{trifmm}(\mathbf{d}_i^{(j)}, \boldsymbol{\lambda}_i^{(j)}, \boldsymbol{\eta}_i^{(j)}, \mathbf{e}, \log|s-t|)$ ▷ *$G_1\mathbf{e}$ as needed in (4.27)*
        $\mathbf{v}_2 \leftarrow \mathsf{trifmm}(\mathbf{d}_i^{(j)}, \mathbf{d}_i^{(j)}, \mathbf{0}, \mathbf{e}, \log|s-t|)$ ▷ *$G_2\mathbf{e}$ as needed in (4.27)*
        $\hat{\mathbf{v}}_i^{(j)} \leftarrow \exp\left(\frac{\mathbf{v}_1 - \mathbf{v}_2}{2}\right)$ ▷ *Löwner's formula for $\hat{\mathbf{v}}$ as in (4.25)–(4.27)*
        $\mathbf{b}_i^{(j)} \leftarrow (\mathsf{trifmm}(\mathbf{d}_i^{(j)}, \boldsymbol{\lambda}_i^{(j)}, \boldsymbol{\eta}_i^{(j)}, \hat{\mathbf{v}}_i^{(j)} \odot \hat{\mathbf{v}}_i^{(j)}, \frac{1}{(s-t)^2}))^{-1/2}$
        ▷ *Normalization factor as in (4.29)*
        $\hat{Q}_i^{(j)} \leftarrow \{\hat{\mathbf{v}}_i^{(j)}, \mathbf{b}_i^{(j)}, \mathbf{d}_i^{(j)}, \boldsymbol{\lambda}_i^{(j)}, \boldsymbol{\eta}_i^{(j)}\}$ ▷ *Cauchy-like structured representation of the local eigenmatrix as in (4.28)*
        for $k = j+1, j+2, \ldots, r$ **do** ▷ *Multiplication of $\hat{Q}_i^{(j)}$ to the remaining columns of Z_i via the steps as in (4.31)*
          $Z_i(:,k) \leftarrow \hat{\mathbf{v}}_i^{(j)} \odot Z_i(:,k)$
          $Z_i(:,k) \leftarrow \mathsf{trifmm}(\mathbf{d}_i^{(j)}, \boldsymbol{\lambda}_i^{(j)}, \boldsymbol{\eta}_i^{(j)}, Z_i(:,k), \frac{1}{s-t})$
          $Z_i(:,k) \leftarrow \mathbf{b}_i^{(j)} \odot Z_i(:,k)$
        end for
      end for
      $\boldsymbol{\lambda}_i \leftarrow \boldsymbol{\lambda}_i^{(r)}$ ▷ *Local eigenvalues associated with node i*
    end if
  end for
  $\boldsymbol{\lambda} \leftarrow \boldsymbol{\lambda}_{\text{root}(\mathcal{T})}$, $Q \leftarrow \{\{\hat{Q}_i^{(j)}\}_{j=1}^r, P_i\}_{i\in\mathcal{T}}$ ▷ *Final eigenvalues and eigenmatrix Q in (4.36), with $\hat{Q}_i$ in (4.35) given by $\prod_{j=1}^r \hat{Q}_i^{(j)}$*
  return $\boldsymbol{\lambda}, Q$
end procedure

Algorithm 4 SuperDC eigenmatrix-vector multiplication

procedure $\mathsf{superdcmv}(Q_i, \mathbf{x}, \text{transpose})$ ▷ *Application of a local eigenmatrix Q_i or its transpose to a vector $\mathbf{x}$, depending on whether 'transpose' is 0 or 1*
  $i_1 \leftarrow$ smallest descendant of i
  if transpose $= 0$ **then** ▷ $\mathbf{y} = Q_i\mathbf{x}$
    $\mathbf{y}_i \leftarrow \mathbf{x}$
    for $k = i, i-1, \ldots, i_1$ **do** ▷ *Reverse postordered traversal of $\mathcal{T}_i$*
      if k is leaf **then**
        $\mathbf{y}_k \leftarrow Q_k\mathbf{y}_k$ ▷ *Dense Q_k at the leaf level*
      else
        for $j = r, r-1, \ldots, 1$ **do** ▷ *Multiplication of $\hat{Q}_k^{(j)}$ via the steps like in (4.31)*
          $\mathbf{y}_k \leftarrow \mathbf{b}_k^{(j)} \odot \mathbf{y}_k$
          $\mathbf{y}_k \leftarrow \mathsf{trifmm}(\mathbf{d}_k^{(j)}, \boldsymbol{\lambda}_k^{(j)}, \boldsymbol{\eta}_k^{(j)}, \mathbf{y}_k, \frac{1}{s-t})$
          $\mathbf{y}_k \leftarrow \hat{\mathbf{v}}_k^{(j)} \odot \mathbf{y}_k$
        end for
        $\mathbf{y}_k \leftarrow P_k^T\mathbf{y}_k$ ▷ *Permutation like in (4.34)*
        $\begin{pmatrix}\mathbf{y}_{c_1}\\ \mathbf{y}_{c_2}\end{pmatrix} \leftarrow \mathbf{y}_k$ ▷ *Partitioning following the sizes of Q_{c_1}, Q_{c_2}, with c_1, c_2 the children of k*
      end if
    end for
  else ▷ $\mathbf{y} = Q_i^T\mathbf{x}$
    Partition $\mathbf{x}$ into $\mathbf{x}_k$ pieces following the leaf-level Q_k sizes
    for $k = i_1, i_1+1, \ldots, i$ **do** ▷ *Postordered traversal of $\mathcal{T}_i$*
      if k is leaf **then**
        $\mathbf{y}_k \leftarrow Q_k^T\mathbf{x}_k$ ▷ *Dense Q_k at the leaf level*
      else
        $\mathbf{y}_k \leftarrow \begin{pmatrix}\mathbf{y}_{c_1}\\ \mathbf{y}_{c_2}\end{pmatrix}$ ▷ *c_1, c_2: children of k*
        $\mathbf{y}_k \leftarrow P_k\mathbf{y}_k$ ▷ *Permutation like in (4.34)*
        for $j = 1, 2, \ldots, r$ **do** ▷ *Multiplication of $(\hat{Q}_k^{(j)})^T$ via the steps like in (4.31)*
          $\mathbf{y}_k \leftarrow \hat{\mathbf{v}}_k^{(j)} \odot \mathbf{y}_k$
          $\mathbf{y}_k \leftarrow -\mathsf{trifmm}(\boldsymbol{\lambda}_k^{(j)}, \mathbf{d}_k^{(j)}, \boldsymbol{\eta}_k^{(j)}, \mathbf{y}_k, \frac{1}{s-t})$ ▷ *The negative sign and the switch of $\boldsymbol{\lambda}_k^{(j)}$ and $\mathbf{d}_k^{(j)}$ are because of the transpose*
          $\mathbf{y}_k \leftarrow \mathbf{b}_k^{(j)} \odot \mathbf{y}_k$
        end for
      end if
    end for
  end if
  return y
end procedure