SIAM J. MATRIX ANAL. APPL.
Vol. 35, No. 3, pp. 1038–1057
© 2014 Society for Industrial and Applied Mathematics

AN ACCELERATED DIVIDE-AND-CONQUER ALGORITHM FOR THE BIDIAGONAL SVD PROBLEM*

SHENGGUO LI†, MING GU‡, LIZHI CHENG§, XUEBIN CHI¶, AND MENG SUN‖

Abstract. In this paper, aiming at solving the bidiagonal SVD problem, a classical divide-and-conquer (DC) algorithm is modified, which needs to compute the SVD of broken arrow matrices by solving secular equations. The main cost of DC lies in the updating of singular vectors, which involves two matrix-matrix multiplications. We find that the singular vector matrices of a broken arrow matrix are Cauchy-like matrices and have an off-diagonal low-rank property, so they can be approximated efficiently by hierarchically semiseparable (HSS) matrices. Hereby, by using the HSS techniques, the complexity of computing singular vectors can be reduced significantly. An accelerated DC algorithm is proposed, denoted by ADC. Furthermore, we use a structured low-rank approximation method to construct these HSS approximations. Numerous experiments show ADC is both fast and numerically stable. When dealing with large matrices with few deflations, ADC can be 3x faster than DC in the optimized LAPACK libraries such as Intel MKL without any degradation in accuracy. These techniques can be used to similarly solve the symmetric tridiagonal eigenvalue problem.

Key words. divide-and-conquer algorithm, bidiagonal matrices, HSS matrices, Cauchy-like matrices, Schur complements

AMS subject classifications. 15A12, 65F05, 65F30

DOI. 10.1137/130945995

1. Introduction.

The divide-and-conquer (DC) algorithms for the bidiagonal SVD problem have been well studied, and some different versions have been proposed by Jessup and Sorensen citeJessup-svd, Arbenz and Golub [2], Gu and Eisenstat [23], etc. In the worst case, all these versions cost $\mathcal{O}(N^3)$ flops to compute the singular vectors, though in practice it appears to take $\mathcal{O}(N^{2.3})$ flops on average [10] because of *deflation* [18, 10], where N is the dimension of the matrix. By using the hierarchically semiseparable (HSS) matrix techniques [8], we in this paper propose an accelerated DC algorithm (in short *ADC*), which requires at most $\mathcal{O}(N^2 r)$ flops, where r is a small constant which may depend on the singular values. In practice it is usually true that $r \ll N$ for large N. Note that the fast multipole method (FMM) [19] has been suggested to accelerate the DC algorithm in [23, 24], and its worst case complexity is only $\mathcal{O}(N^2)$. However, the FMM-based method has not been implemented in practice.

*Received by the editors November 20, 2013; accepted for publication (in revised form) by N. Mastronardi April 18, 2014; published electronically August 7, 2014.
http://www.siam.org/journals/simax/35-3/94599.html

†College of Computer Science, the State Key Laboratory for High Performance Computation, National University of Defense Technology, Changsha 410073, China (nudtlsg@gmail.com). The research of this author was supported in part by NSF of China (11101430).

‡Department of Mathematics, University of California, Berkeley, CA 47920 (mgu@math.berkeley.edu). The research of this author was supported in part by NSF Award CCF-1319312.

§College of Science, the State Key Laboratory for High Performance Computation, National University of Defence Technology, Changsha 410073, China (clzcheng@nudt.edu.cn). The research of this author was supported by Natural Science Project of NUDT (JC120201) and NSF of Hunan Province in China (13JJ2001).

¶Supercomputing Center, Computer Network Information Center, Chinese Academy of Science, Beijing, 100190, China (chi@sccas.cn). The research of this author was supported in part by NSF of China (91130019).

‖College of Command Information and Systems, PLA University of Science and Technology, Nanjing 210007, China (sunmengccjs@gmail.com).

The DC algorithm in [23] computes the SVD of a sequence of *broken arrow* matrices, and multiplies these intermediate singular vector matrices with some (dense) orthogonal matrices; see (2.4). What we find in this work is that the singular vector matrices of these broken arrow matrices are *off-diagonal low rank*, which can be efficiently approximated by HSS matrices [7, 8]. By using the fast HSS *construction* and *matrix multiplication* algorithms [8, 37], the cost of updating singular vectors (see (2.4)) is reduced from $\mathcal{O}(N^3)$ to $\mathcal{O}(N^2 r)$, where N is the size of a broken arrow matrix and r is the maximum off-diagonal rank of its singular vector matrices.

Recall that a matrix C is called *Cauchy-like* if it satisfies

$$D \cdot C - C \cdot W = u \cdot v^T,$$

where $D = \mathrm{diag}(d_1, \ldots, d_N), W = \mathrm{diag}(w_1, \ldots, w_N)$, and $u, v \in R^N$. As is shown in section 2.1, the singular vectors of a broken arrow matrix form a Cauchy-like matrix (2.10). Furthermore, a structured low-rank approximation algorithm is proposed for Cauchy-like matrices in section 3.1, which is called *structured rank-revealing Schur-complement factorization* (SRRSC).[1] For numerical stability, SRRSC further incorporates a variant of a *complete pivoting* strategy and a "*swap*" technique. The technique *swap* was first introduced in [39] for computing a rank-revealing LU factorization. By using these techniques, an HSS matrix can be constructed in $\mathcal{O}(Nr)$ memory and $\mathcal{O}(N^2 r)$ flops.

Especially for updating the singular vectors of large broken arrow matrices, the HSS-based method is much faster than plain matrix-matrix multiplications, which ensures the high performance of ADC. We compare ADC with highly optimized linear algebra software such as Intel MKL. Numerical results show that ADC can be more than 3x faster than MKL for some large matrices without any significant loss of accuracy. The numerical stability of ADC is tested via matrices collected in LAPACK tester `stetester` [38]. The techniques introduced here can also be used to solve the symmetric tridiagonal eigenvalue problem [24].

This paper is organized as follows. Section 2 introduces the DC algorithm and some key concepts of HSS matrices, and explains why DC can be accelerated by using rank-structured matrices such as HSS. Section 3 investigates the SRRSC algorithm for Cauchy-like matrices. Section 4 analyzes the approximation error of the HSS construction algorithm based on SRRSC. Section 5 shows the implementation details of ADC. Section 6 reports all the numerical results, and conclusions are drawn in section 7.

2. Preliminaries. We briefly introduce the DC algorithm for the bidiagonal SVD problems; see also [23, 31, 10] for more details. Assume that B is an upper bidiagonal $N \times N$ matrix,

$$B = \begin{bmatrix} a_1 & b_1 & & & & \\ & a_2 & b_2 & & & \\ & & \ddots & \ddots & & \\ & & & a_{N-1} & b_{N-1} & \\ & & & & a_N & \end{bmatrix}. \tag{2.1}$$

The DC algorithm has three stages: (a) divide a big problem into smaller subproblems recursively; (b) solve small problems; (c) synthesize the solutions of small problems.

[1] A more general algorithm has been used to solve Toeplitz linear systems [25, 26], which also works for complex matrices.

First, partition B into

$$B = \begin{bmatrix} B_1 \\ a_{k+1}\mathbf{e}_{k+1}^T & b_k\mathbf{e}_1^T \\ & B_2 \end{bmatrix}, \text{ where } B_1 \in R^{k\times(k+1)}, \quad B_2 \in R^{l\times l}, \tag{2.2}$$

$N = k + 1 + l$, $\mathbf{e}_{k+1}^T = [0, \ldots, 0, 1]$, and $\mathbf{e}_1^T = [1, 0, \ldots, 0]$. Compute the SVD of B_1 and B_2, $B_1 = U_1[\Sigma_1\ 0]V_1^T, B_2 = U_2\Sigma_2V_2^T$, then the SVD of B equals

$$B = \begin{bmatrix} U_1 & & \\ & 1 & \\ & & U_2 \end{bmatrix} \begin{bmatrix} [\Sigma_1 & 0] & \\ x_1^T & & x_2^T \\ & & \Sigma_2 \end{bmatrix} \begin{bmatrix} V_1^T & \\ & V_2^T \end{bmatrix} \tag{2.3}$$

$$= \begin{bmatrix} 0 & U_1 & \\ 1 & & \\ & & U_2 \end{bmatrix} \begin{bmatrix} x_1(k+1) & x_1(1:k) & x_2(1:l) \\ & \Sigma_1 & \\ & & \Sigma_2 \end{bmatrix} \begin{bmatrix} \hat{V}_1^T & \\ & V_2^T \end{bmatrix}, \tag{2.4}$$

where x_1^T equals a_{k+1} times the bottom row of V_1^T, x_2^T equals b_k times the top row of V_2^T, and $\hat{V}_1^T$ is the matrix after moving the bottom row of V_1^T to the top.

Without any loss of generality, we assume the matrix in the middle of (2.4) has the following structure,

$$M = \begin{bmatrix} z_1 & z_2 & \cdots & z_N \\ & d_2 & & \\ & & \ddots & \\ & & & d_N \end{bmatrix}, \tag{2.5}$$

where $0 < d_2 < \cdots < d_N$ and $z_i \neq 0$. The problem boils down to computing the SVD of matrix M, which can be solved by the following lemma.

LEMMA 2.1 (Jessup and Sorensen [31]). *Let $U\Omega V^T$ be the SVD of M with*

$$U = (u_1, \ldots, u_N), \quad \Omega = \operatorname{diag}(\omega_1, \ldots, \omega_N), \text{ and } V = (v_1, \ldots, v_N),$$

where $0 < \omega_1 < \cdots < \omega_N$. Then the singular values $\{\omega_j\}_{j=1}^N$ satisfy the interlacing property

$$0 \equiv d_1 < \omega_1 < d_2 < \cdots < d_N < \omega_N < d_N + \|z\|_2,$$

and the secular equation

$$\mathcal{F}(\omega) = 1 + \sum_{i=1}^{N} \frac{z_i^2}{d_i^2 - \omega^2} = 0. \tag{2.6}$$

The singular vectors satisfy

$$\begin{aligned} u_j &= \left(-1, \frac{d_2z_2}{d_2^2 - w_j^2}, \ldots, \frac{d_Nz_N}{d_N^2 - w_j^2}\right)^T \Bigg/ \sqrt{1 + \sum_{i=2}^{N} \frac{d_i^2z_i^2}{(d_i^2 - w_j^2)^2}}, \\ v_j &= \left(\frac{z_1}{d_1^2 - w_j^2}, \ldots, \frac{z_N}{d_N^2 - w_j^2}\right)^T \Bigg/ \sqrt{\sum_{i=1}^{N} \frac{z_i^2}{(d_i^2 - w_j^2)^2}}, \end{aligned} \tag{2.7}$$

where $j = 1, \ldots, N$.

Even though the computed singular values $\hat{w}_i$ are highly accurate, the singular vectors computed by (2.7) may lose orthogonality. To obtain numerically orthogonal singular vectors, we use the Löwner theorem [36, 23] to recompute z_i.

LEMMA 2.2 (see [10, 36]). *Given a diagonal matrix $D = \mathrm{diag}(d_1, d_2, \ldots, d_N)$ and a set of numbers $\{\hat{w}_j\}$ satisfying the interlacing property*

$$0 \equiv d_1 < \hat{w}_1 < d_2 < \hat{w}_2 < \cdots < d_N < \hat{w}_N, \tag{2.8}$$

there exists a matrix $\hat{M} = \begin{bmatrix} \hat{z}_1 & \hat{z}_2 & \cdots & \hat{z}_N \\ & d_2 & & \\ & & \ddots & \\ & & & d_N \end{bmatrix}$ whose singular values are $\{\hat{w}_j\}_{j=1}^N$, and the vector $\hat{z} = (\hat{z}_1, \hat{z}_2, \ldots, \hat{z}_N)^T$ is determined by

$$|\hat{z}_i| = \sqrt{(\hat{w}_N^2 - d_i^2)\Pi_{j=1}^{i-1}\frac{(\hat{w}_j^2 - d_i^2)}{(d_j^2 - d_i^2)}\Pi_{j=i}^{N-1}\frac{(\hat{w}_j^2 - d_i^2)}{(d_{j+1}^2 - d_i^2)}},$$

where the sign of $\hat{z}_i$ can be chosen arbitrarily.

Remark 1. The matrix M has to be deflated before computing its SVD when

(1) some z_i are small enough in magnitude, $|z_i| < \tau\|M\|_2$; or

(2) some d_i and d_j are close enough, $|d_i - d_j| < \tau\|M\|_2$, for $i, j \geq 2$,

where τ is a small constant of machine precision. See [23] for details.

The SVD of $\hat{M}$ can be used as an approximation to the SVD of M as long as the computed $\{\hat{\omega}_i\}$ are sufficiently accurate. For further details about how to solve secular equations accurately, the reader is referred to [10, 32, 21].

We use the LAPACK routine `dlasd8` to solve the secular equation (2.6), from which we get w_i, $\hat{z}_i$, $\gamma_i = w_i - d_i$ (i.e., the distance between w_i and d_i), and $\mu_i = d_{i+1} - w_i$ (i.e., the distance between w_i and d_{i+1}). μ_n will not be referenced. To achieve high accuracy, $d_i^2 - w_j^2 = (d_i + w_j)(d_i - w_j)$ should be computed via γ and μ, where

$$d_i - w_j = \begin{cases} (d_i - d_j) - \gamma_j & \text{if } i \leq j, \\ (d_i - d_{j+1}) + \mu_j & \text{if } i > j. \end{cases} \tag{2.9}$$

2.1. Structure of singular vectors of M. By Lemma 2.1 we know that the left and right singular vector matrices of M are, respectively,

$$U = \begin{bmatrix} -\beta_1 & -\beta_2 & \cdots & -\beta_N \\ \frac{u_2\beta_1}{d_2^2 - w_1^2} & \frac{u_2\beta_2}{d_2^2 - w_2^2} & \cdots & \frac{u_2\beta_N}{d_2^2 - w_N^2} \\ \vdots & \vdots & & \vdots \\ \frac{u_N\beta_1}{d_N^2 - w_1^2} & \frac{u_N\beta_2}{d_N^2 - w_2^2} & \cdots & \frac{u_N\beta_N}{d_N^2 - w_N^2} \end{bmatrix}, V^T = \begin{bmatrix} \frac{\alpha_1 z_1}{d_1^2 - w_1^2} & \frac{\alpha_1 z_2}{d_2^2 - w_1^2} & \cdots & \frac{\alpha_1 z_N}{d_N^2 - w_1^2} \\ \frac{\alpha_2 z_1}{d_1^2 - w_2^2} & \frac{\alpha_2 z_2}{d_2^2 - w_2^2} & \cdots & \frac{\alpha_2 z_N}{d_N^2 - w_2^2} \\ \vdots & \vdots & & \vdots \\ \frac{\alpha_N z_1}{d_1^2 - w_N^2} & \frac{\alpha_N z_2}{d_2^2 - w_N^2} & \cdots & \frac{\alpha_N z_N}{d_N^2 - w_N^2} \end{bmatrix}, \tag{2.10}$$

where $\beta_i = 1\Big/\sqrt{1 + \sum_{i=2}^N \frac{d_i^2 z_i^2}{(d_i^2 - w_j^2)^2}}$, $\alpha_i = 1\Big/\sqrt{\sum_{i=1}^N \frac{z_i^2}{(d_i^2 - w_j^2)^2}}$, and $u_i = d_i z_i$. Note that V^T is a Cauchy-like matrix, and U is a Cauchy-like matrix with rank-one modification. Furthermore, both U and V usually have low-rank structure, which is shown in the following example.

Example 1. Let A be a 2000×2000 Cauchy-like matrix, $A = (\frac{u_i v_j}{x_i - y_j})_{i,j}$, where the entries of x and y are equally distributed in the interval $[0.1, 9.0]$ and are interlacing, and u, v are random vectors. The numerical ranks of the top-right off-diagonal blocks,

TABLE 1
Example 1: *Ranks of some off-diagonal blocks.*

k	1	2	3	4	5	6	7	8	9	10
Dim	100	200	300	400	500	600	700	800	900	1000
Rank	22	24	25	26	27	28	27	32	31	31

$H_k = A(1 : m \times k, m \times k + 1 : end)$ with $m = 100$, are shown in Table 1, where we used MATLAB notation and $\epsilon = 1.0e^{-13}$ to determine the numerical ranks.

From the second and third rows of Table 1, we can see that the numerical ranks of off-diagonal blocks are much smaller than their dimensions. It is because the elements of x and y are equally spaced and interlacing. Note that not every Cauchy-like matrix has such an off-diagonal low-rank property; the ranks of its off-diagonal blocks may nearly equal the size of the matrix. See the following for more details.

The off-diagonal low-rank property can also be shown from the approximation of function $f(x) = 1/x$. For example, the element $\frac{1}{x_i - y_j}$ can be approximated by sums of exponentials,

$$\frac{1}{x_i - y_j} \approx \sum_{\ell=1}^{k} w_\ell e^{-\alpha_\ell (x_i - y_j)} := s_k(x_i - y_j). \tag{2.11}$$

Assume that $x_i \in I_1 \subset [a, b]$, $y_j \in I_2 \subset [a, b]$, $I_1 \cap I_2 = \emptyset$, and $x_i > y_j$. Denote $\text{dist}(I_1, I_2) = \min_{x_i \in I_1 y_j \in I_2} |x_i - y_j|$, then

$$1 \le \frac{x_i - y_j}{\text{dist}(I_1, I_2)} \le \frac{b - a}{\text{dist}(I_1, I_2)} := R. \tag{2.12}$$

An approximation error bound is given in [5] for the sums of exponentials,

$$\left| \frac{1}{x} - s_k(x) \right| \le 16 e^{-\frac{k\pi^2}{\log(8R)}}, \tag{2.13}$$

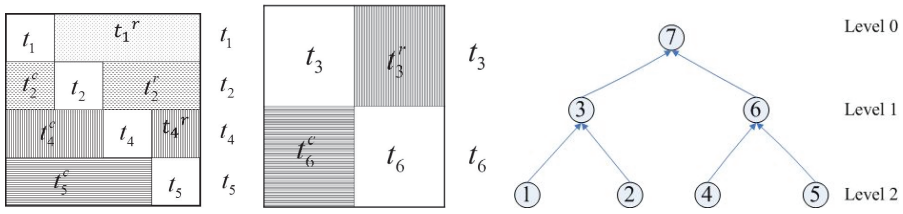
where $s_k(x)$ is defined in (2.11) and $x \in [1, R]$. It shows that the number of approximation terms k (corresponding to the off-diagonal rank) increases as R, defined in (2.12), increases. If the elements of x and y are clustered, i.e., $\text{dist}(I_1, I_2)$ is small, R is large and therefore it requires more approximation terms.

Accelerating the multiplications of a Cauchy matrix with a vector is known as the Trummer's problem [17]. Based on polynomial interpolations, Gerasoulis has proposed one fast algorithm for this problem [15], which costs $\mathcal{O}(N\log^2 N)$ flops. Unfortunately, these kinds of algorithms are not numerically stable; see also [4, 41] for some similar algorithms. Some variants of FMMs [19, 13, 50] can also be used to solve this problem.

For Cauchy matrices which have low-rank property [6], algorithms based on *rank-structured* matrix techniques [7, 27, 28] are better choices, since they are much easier to implement and are numerically stable. In the following sections, we show how to accelerate the multiplications by using HSS matrices [8]. The HSS matrices can be seen as an algebraic counterpart of FMM, and are highly related to $\mathcal{H}^2$-matrices [28].

2.2. HSS matrices. In this section, we introduce some notation, and give a brief introduction to the key concepts of HSS structures. We follow the notation and definition in [48, 49, 35].

Assume an $N \times N$ matrix A is partitioned into $n \times n$ blocks. Let $\mathcal{I} = \{1, 2, \ldots, N\}$ and $\mathcal{T}$ be a binary tree. If i is a nonleaf node of $\mathcal{T}$, its left and right child are



(a) Bottom level (b) Level 1 (c) HSS postordering tree

FIG. 1. *Matrix partition and the corresponding index sets and binary tree* $\mathcal{T}$.

represented by i_1 and i_2, respectively. For our purpose, $\mathcal{T}$ is assumed to be *postordered*. That is, the nodes are ordered so that a nonleaf node i satisfies the ordering $i_1 < i_2 < i$. Each node i is associated with a contiguous subset t_i of $\mathcal{I}$, so that the following conditions are satisfied:

- $t_i \cup t_{\mathrm{sib}(i)} = t_{\mathrm{par}(i)}$, where $\mathrm{sib}(i)$ and $\mathrm{par}(i)$ denote the sibling and parent of node i, respectively;
- $t_{\mathrm{root}(\mathcal{T})} = \mathcal{I}$, where $\mathrm{root}(\mathcal{T})$ denotes the root of $\mathcal{T}$.

Furthermore, for each node i, let t_i^c be the set of all indices less than those in t_i and t_i^r be the set of all indices greater than those in t_i. Thus, $\mathcal{I} = t_i^c \cup t_i \cup t_i^r$ for each node i. Figure 1 illustrates these sets.

The HSS representation uses a recursive row and column partitioning of the matrix. Following the notation of [49], let $A_{t_i t_j}$ denote a submatrix of A with row index set t_i and column index set t_j. A block row (column) of A excluding the diagonal block is called an *HSS block row (column)*, or just *HSS block*. For example, the ith HSS block row and block column are

$$H_i^{row} = \begin{bmatrix} A_{t_i t_i^c} & A_{t_i t_i^r} \end{bmatrix} \quad \text{and} \quad H_i^{col} = \begin{bmatrix} A_{t_i^c t_i} \\ A_{t_i^r t_i} \end{bmatrix},$$

respectively. We call the maximum (numerical) rank of all HSS blocks the *HSS rank* of the matrix.

We introduce the postordered HSS form [49]. Each node i has *generators* D_i, U_i, V_i, R_i, W_i, and B_i, and these generators are nested to present the whole matrix A,

$$D_i = \begin{bmatrix} D_{i_1} & U_{i_1} B_{i_1} V_{i_2}^T \\ U_{i_2} B_{i_2} V_{i_1}^T & D_{i_2} \end{bmatrix}, \quad U_i = \begin{bmatrix} U_{i_1} R_{i_1} \\ U_{i_2} R_{i_2} \end{bmatrix}, \quad V_i = \begin{bmatrix} V_{i_1} W_{i_1} \\ V_{i_2} W_{i_2} \end{bmatrix}. \tag{2.14}$$

For example, a 4×4 block HSS matrix can be written as

$$\begin{bmatrix} D_1 & U_1 B_1 V_2^T & U_1 R_1 B_3 W_4^T V_4^T & U_1 R_1 B_3 W_5^T V_5^T \\ U_2 B_2 V_1^T & D_2 & U_2 R_2 B_3 W_4^T V_4^T & U_2 R_2 B_3 W_5^T V_5^T \\ U_4 R_4 B_6 W_1^T V_1^T & U_4 R_4 B_6 W_2^T V_2^T & D_4 & U_4 B_4 V_5^T \\ U_5 R_5 B_6 W_1^T V_1^T & U_5 R_5 B_6 W_2^T V_2^T & U_5 B_5 V_4^T & D_5 \end{bmatrix}. \tag{2.15}$$

The corresponding HSS tree is shown in Figure 1(c). Once a matrix is represented in its HSS form, there exist many fast algorithms to untilize it. For example, there exist algorithms with complexity $O(Nr)$ for solving a linear system [8] and multiplying with a vector [7], where N is the dimension of the matrix and r is its HSS rank.

3. HSS construction for Cauchy-like matrices. For Cauchy-like matrices, a structured low-rank approximation algorithm was reported in a seminar talk [25, 26],

which was developed for solving Toeplitz-like systems. In this section, we briefly introduce the formulas for the real case and call this algorithm *SRRSC factorization*; see Algorithm 1. We further propose a modified pivoting strategy. Some details are left to the appendix. By using SRRSC, we propose a new HSS construction algorithm for Cauchy-like matrices. Comparing it to the existing ones, this new algorithm has better data locality and only requires $O(N)$ memory rather than $O(N^2)$.

ALGORITHM 1. SRRSC FACTORIZATION. Let A be an $m \times n$ Cauchy-like matrix defined by (3.3), and its generators are d, w, u, and v. Let $u^{(0)} = u$ and $p = (1, 2, \ldots, n)$ which records the row permutations. Assume that A has numerical rank $r < \min(m, n)$ and $k_1 = k - 1$.

`for` $k = 1, \ldots, \min(m, n)$

(1) call $Pivot(d^{(k_1)}, w^{(k_1)}, u^{(k_1)}, v^{(k_1)}, u^{(0)}, p^{(k_1)}, k_1)$;
% The superscript of d, w, u, v, and p will be changed to k.

(2) `if` $|A^{(k_1)}(1,1)|$ is negligible, `then`
$r = k - 1$;
`return`;

(3) `else`
$u^{(k)}(k+1:m) = u^{(k)}(k+1:m) \cdot \frac{d^{(k)}(k+1:m) - d^{(k)}(k)}{d^{(k)}(k+1:m) - w^{(k)}(k)}$;
$v^{(k)}(k+1:n) = v^{(k)}(k+1:n) \cdot \frac{w^{(k)}(k+1:n) - w^{(k)}(k)}{w^{(k)}(k+1:n) - d^{(k)}(k)}$;
$y^{(k)}(1:k-1) = y^{(k)}(1:k-1) \cdot \frac{w^{(k)}(k) - d^{(k)}(1:k-1)}{d^{(k)}(k) - d^{(k)}(1:k-1)}$;
$y^{(k)}(k) = \frac{d^{(k)}(k) - w^{(k)}(k)}{u^{(0)}(k)}$;
`for` $j = 1 : k - 1$
$y^{(k)}(k) = y^{(k)}(k) \cdot \frac{w^{(k)}(j) - d^{(k)}(k)}{d^{(k)}(j) - d^{(k)}(k)}$;
`end for`

`end for`

3.1. Structured Schur complement factorization. We first introduce the definition of Schur complement factorization and some properties of Cauchy-like matrices.

DEFINITION 3.1 (see [40]). *Let $A \in R^{m \times n}$ and k be an integer (less than m and n). The following factorization is called the kth Schur complement factorization of A,*

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} = \begin{bmatrix} I_k & \\ Z^{(k)} & I_{m-k} \end{bmatrix} \begin{bmatrix} A_{11} & A_{12} \\ & A^{(k)} \end{bmatrix}, \tag{3.1}$$

and $A^{(k)}$ is called the kth Schur complement.

One good property of the Cauchy matrix $C = [\frac{1}{x_i - y_j}]$ is that its determinant satisfies

$$\det(C) = \frac{\Pi_{1 \le i < j \le n}(x_j - x_i)(y_i - y_j)}{\Pi_{1 \le i,j \le n}(x_i - y_j)}. \tag{3.2}$$

By exploiting this property, Demmel [11] shows that the Gaussian elimination with complete pivoting (GECP) of Cauchy matrices can be computed with high relative accuracy, as do its singular values. The other property is that *the Schur complements of a Cauchy-like matrix have displacement structures*. Many fast linear solvers are based on this property, such as the well-known GKO algorithm [16] and some variations [43, 30].

Let $u^{(k)}$ and $v^{(k)}$ denote the generators of $A^{(k)}$. The following theorem shows that $u^{(k)}$ and $v^{(k)}$ can be computed recursively and accurately for any $1 \le k \le n$.

THEOREM 3.2 (see [16, 25, 20]). *The kth Schur complement $A^{(k)}$ satisfies that*

$$D_{k+1}A^{(k)} - A^{(k)}W_{k+1} = u^{(k)}(k+1:n)\cdot v^{(k)T}(k+1:n), \tag{3.3}$$

where $D_{k+1} = \mathrm{diag}(d_{k+1}, \ldots, d_n)$ and $W_{k+1} = \mathrm{diag}(w_{k+1}, \ldots, w_n)$. Then,

$$u^{(k)}(k+\ell) = u^{(k-1)}(k+\ell)\cdot \frac{d_{k+\ell} - d_k}{d_{k+\ell} - w_k}, \tag{3.4}$$

$$v^{(k)}(k+\ell) = v^{(k-1)}(k+\ell)\cdot \frac{w_{k+\ell} - w_k}{w_{k+\ell} - d_k}, \tag{3.5}$$

where $1 \le \ell \le n-k$, $k \ge 1$.

COROLLARY 3.3. *For $k \ge 1$ and $1 \le \ell \le n-k$, the generators of $A^{(k)}$ satisfy*

$$u^{(k)}(k+\ell) = \prod_{j=1}^{k} \frac{d_{k+\ell} - d_j}{d_{k+\ell} - w_j}\cdot u^{(0)}(k+\ell),$$

$$v^{(k)}(k+\ell) = \prod_{j=1}^{k} \frac{w_{k+\ell} - w_j}{w_{k+\ell} - d_j}\cdot v^{(0)}(k+\ell).$$

Furthermore, the matrix $Z^{(k)}$ in (3.1) has displacement structures too. Therefore the Schur complement factorization of a Cauchy-like matrix is fully structured. The generators of $Z^{(k)}$ can also be computed recursively.

THEOREM 3.4 (see [25]). *The generators of $Z^{(k)}$ satisfy a displacement equation*

$$D_2^{(k)}Z^{(k)} - Z^{(k)}D_1^{(k)} = u^{(k)}(k+1:n)\cdot y^{(k)}(1:k)^T, \tag{3.6}$$

where $D_1^{(k)} = \mathrm{diag}(d_1, \ldots, d_k)$, $D_2^{(k)} = \mathrm{diag}(d_{k+1}, \ldots, d_n)$, $u^{(k)}(k+1:n)$ are computed recursively by (3.4), and

$$y^{(k)}(i) = \begin{cases} y^{(k-1)}(i)\cdot \frac{w_k - d_i}{d_k - d_i} & \text{if } 1 \le i \le k-1, \\ \prod_{j=1}^{k-1} \frac{d_k - w_j}{d_k - d_j}\cdot \frac{d_k - w_k}{u_k} & \text{if } i = k. \end{cases} \tag{3.7}$$

Since permutations do not destroy the Cauchy-like structure, we can permute the generators $u^{(k)}, v^{(k)}, d$, and w so that the first entry of $A^{(k)}$ is large, $A^{(k)}(1,1) = \frac{u^{(k)}(k+1)v^{(k)}(k+1)}{d_{k+1} - w_{k+1}}$. The complete pivoting (CP) strategy is a good choice to have a stable algorithm. However, CP is not usually used in practice since it severely slows down the algorithm. Some variants of pivoting strategies for Cauchy-like matrices have been proposed in [20] and [40]. In section 3.3, we propose a new pivoting strategy, which is a good tradeoff between stability and speed.

If (3.1) is rank deficient, after some factorization steps the entries of $A^{(k)}$ would become negligible, and then by ignoring $A^{(k)}$, we get a low-rank approximation to A,

$$A \approx P \begin{bmatrix} I \\ Z^{(k)} \end{bmatrix} A(\mathscr{T}, 1:n), \tag{3.8}$$

where $\mathscr{T} = \{i_1, i_2, \ldots, i_k\} \subset \{1, 2, \ldots, n\}$, P is a permutation matrix which records the row permutations during the factorization. To be more specific, $A(\mathscr{T}, 1:n)$ consists of a subset of rows of A and from Theorem 3.4 the entries of $Z^{(k)}$ are

$$Z_{ij}^{(k)} = \frac{u^{(k)}(k+i)y^{(k)}(j)}{d(k+i) - d(j)},$$

where d is the array after being permuted, $1 \le i \le n-k$, and $1 \le j \le k$.

Remark 2. Factorization (3.8) has appeared in strong rank-revealing QR factorization [22] and LU factorization [40], and is also called *interpolation decomposition* in [9, 29]. Algorithm 1 only needs $O(\max(m,n)r)$ memory and costs $O(mnr)$ flops.

3.2. Swap. In [39], Miranian and Gu show that the entries of $Z^{(k)}$ can be computed by using the determinant of submatrices of A; see also [46].

LEMMA 3.5 (see [39]). *Assume an $n \times n$ matrix A is partitioned as $A = \left[\begin{smallmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{smallmatrix}\right]$, where $A_{11} \in R^{k\times k}$, $A_{22} \in R^{(n-k)\times(n-k)}$. If $1 \le i \le k$, $1 \le j \le n-k$, and we interchange rows i and $k+j$ in matrix A, then*

$$\frac{det(\tilde{A}_{11})}{det(A_{11})} = (A_{21}A_{11}^{-1})_{ji}, \tag{3.9}$$

where $\tilde{A}_{11}$ is the top-left $k \times k$ submatrix of A after interchanging rows i and $k+j$.

Equation (3.9) shows that if there is a big entry in Z, for example $|Z_{k\ell}| > \mu$ with $\mu > 1$, by swapping the rows k and $k+\ell$ of A, the entry $Z_{k\ell}$ becomes less than μ in magnitude. After swapping these two rows, the generators u, v, and y can be updated accordingly.

LEMMA 3.6 (see [25]). *If we swap the $\hat{i}$th row with the $\hat{j}$th row, $\hat{i} < \hat{j}$, then the generators are modified by*

$$v^{(k)}(k+\ell) \equiv \frac{d_{\hat{i}} - w_{k+\ell}}{d_{\hat{j}} - w_{k+\ell}} \cdot v^{(k)}(k+\ell), \tag{3.10}$$

$$u^{(k)}(k+\ell) \equiv \begin{cases} \frac{d_{k+\ell} - d_{\hat{j}}}{d_{k+\ell} - d_{\hat{i}}} \cdot u^{(k)}(k+\ell) & \text{if } k+\ell \neq \hat{j}, \\ \prod_{1\le j\le k, j\neq\hat{i}} \frac{d_{\hat{j}} - d_j}{d_{\hat{i}} - w_j} \cdot \frac{d_{\hat{i}} - d_{\hat{j}}}{d_{\hat{i}} - w_{\hat{i}}} u^{(0)}(\hat{i}) & \text{if } k+\ell = \hat{j}, \end{cases} \tag{3.11}$$

$$y^{(k)}(i) \equiv \begin{cases} \frac{d_i - d_{\hat{i}}}{d_i - d_{\hat{j}}} \cdot y^{(k)}(i) & \text{if } i \neq \hat{i}, \\ \prod_{1\le j\le k, j\neq\hat{i}}^{k} \frac{d_{\hat{j}} - w_j}{d_{\hat{j}} - d_j} \cdot \frac{d_{\hat{j}} - w_{\hat{i}}}{u^{(0)}(\hat{j})} & \text{if } i = \hat{i} \end{cases} \tag{3.12}$$

for $1 \le \ell \le n-k$ and $1 \le i \le k$.

Proof. We only show that it is true for $v^{(k)}$ and other cases can be proved similarly. By Corollary 3.3, we know

$$v^{(k)}(k+\ell) = \prod_{j=1}^{k} \frac{w_{k+\ell} - w_j}{w_{k+\ell} - d_j} \cdot v^{(0)}(k+\ell).$$

By exchanging $d_{\hat{i}}$ with $d_{\hat{j}}$, we get (3.10). ∎

3.3. Pivoting. In practice we do not use CP at every factorization step. A much cheaper pivoting strategy is illustrated in the following algorithm. A similar one has been used in [25, 26], but it cannot guarantee that all the entries of A^k (3.1) would be negligible. We modify it by using CP when the chosen pivots are small in magnitude.

ALGORITHM 2. PIVOT $(d^{(k)}, w^{(k)}, u^{(k)}, v^{(k)}, u^{(0)}, p^{(k)}, k)$. The parameters are defined as in Algorithm 1 and for simplicity denote $A^{(k)}$ by V. Let $V(i,:)$ denote its ith row and $V(:,i)$ denote its ith column.

Find $\hat{i}$ and $\hat{j}$ such that $|V(\hat{i},1)| = \max(|V(:,1)|)$, $|V(1,\hat{j})| = \max(|V(1,:)|)$;
`while`(1)
  `if`($|V(ii,1)| > |V(1,jj)|$) `then`
    swap the $(k+1)$th and $(k+\hat{i})$th entries of $u^{(k)}$, $d^{(k)}$, $p^{(k)}$ and $u^{(0)}$;
    find $\hat{j}$, $|V(1,\hat{j})| = \max(|V(1,:)|)$.
    if $\hat{j} = 1$, `return`.
  `else`
    swap the $(k+1)$th and $(k+\hat{j})$th entries of $v^{(k)}$ and $w^{(k)}$;
    find $\hat{i}$, $|V(\hat{i},1)| = \max(|V(:,1)|)$.
    if $\hat{i} = 1$, `return`.
`end while`
`if`($|V(1,1)| > \hat{\tau}$), `then` ($\hat{\tau}$ is in order of machine precision)
  `return`;
`else`
  use CP to find $|V(\hat{i},\hat{j})| = \max(|V|)$;
  swap the $(k+1)$th and $(k+\hat{i})$th entries of $u^{(k)}$, $d^{(k)}$, $p^{(k)}$ and $u^{(0)}$;
  swap the $(k+1)$th and $(k+\hat{j})$th entries of $v^{(k)}$ and $w^{(k)}$.
`end if`

Remark 3. The pivot $A^{(k)}(\hat{i},\hat{j})$ by Algorithm 2 would be the largest in magnitude among the entries of the $\hat{i}$th row and the $\hat{j}$th column of $A^{(k)}$, which is usually large enough to be a pivot. Algorithm 2 is numerically stable since it finally checks whether all the entries of $A^{(k)}$ are small enough.

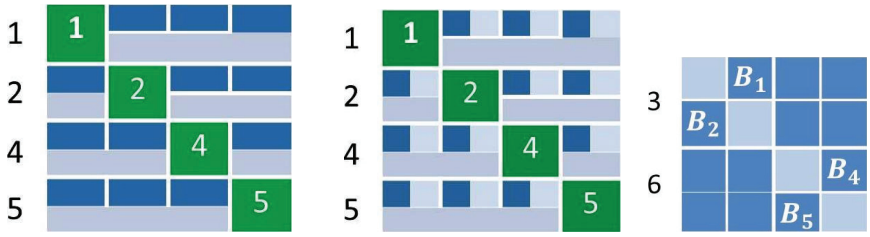
3.4. HSS construction algorithm. The HSS construction algorithm proposed in [49, 35] follows the postordering HSS tree, which must be implemented sequentially. By traversing the HSS tree level by level, a construction algorithm which is naturally parallelizable, is proposed in Algorithm 3; see also [8, 45] for details.

At each level, all HSS block rows are compressed first and then all HSS block columns. After compressing all HSS block columns, generators $\{B_i\}$ are stored and the nonzero off-diagonal blocks are merged together for the next level. Figure 2 shows the compression process for the nodes at the bottom level. The process for a general level is similar.

ALGORITHM 3. HSS CONSTRUCTION ALGORITHM FOR CAUCHY-LIKE MATRICES. Assume HSS tree $\mathcal{T}$ is a full binary tree, and has $L+1$ levels and the bottom level is level L.

1. For each leaf node i, compute the main diagonal block D_i via its generators;
2. `for` $\ell = L : -1 : 1$
   - `if` level ℓ is not the bottom level
     - for node i at level ℓ, construct the generators of its HSS blocks by merging the data of its children;
   - `end if`
   - `for` node i at level ℓ (`row compression`)
     - identify the generators of HSS block row H_i^{row} and compute a low-rank approximation by using SRRSC,
     $$H_i^{row} \approx U_i T_i \ \left(\text{or } \begin{bmatrix} R_{i_1} \\ R_{i_2} \end{bmatrix} T_i \text{ for parent nodes}\right);$$
     - store the matrix U_i (R_{i_1} and R_{i_2});

   `end for`



(a) Row compression at level 2 (b) Column compression at level 2 (c) Merging for level 1

FIG. 2. *The compressions for nodes at the bottom level.*

- `for` node i at level ℓ (`column compression`)
  - identify the generators of HSS block row H_i^{col} and compute a low-rank approximation by using SRRSC,

$$H_i^{col} \approx T_i V_i^T \left(\text{or } T_i \begin{bmatrix} W_{i_1} \\ W_{i_2} \end{bmatrix}^T \text{ for parent nodes} \right);$$

  - store the matrices B_i and V_i (W_{i_1} and W_{i_2});

`end for`

`end for`

Remark 4.

1. Note that when traversing one level up, the sizes of HSS blocks become much smaller than those in the previous level. Thus, the dominant part of the computation is done at the bottom level.
2. For nodes at the same level, different HSS blocks can be compressed independently in parallel.
3. Algorithm 3 only works on several vectors. H_i^{row} and H_i^{col} are not formed explicitly, and generators, U_i, V_i, R_i, W_i, and B_i, are computed via the vectors d, w, u, v, and y.

4. Approximation error for HSS construction. An approximation error bound is given in [47] for the HSS construction algorithm with random sampling in Frobenius norm. In this section, we show the approximation error of Algorithm 3 in 2-norm.

LEMMA 4.1. *The generators U_i and V_i satisfy*

$$1 \le \|U_i\|_2 \le \eta, \quad 1 \le \|V_i\|_2 \le \eta,$$

where η is a constant on the order of r, $\eta = O(r)$.

Proof. We only show it is true for U_i. The truth for V_i can be proved similarly. First, assume $U_i = P \begin{bmatrix} I \\ Z \end{bmatrix}$ and i is a leaf node. Since all the entries $|Z_{ij}| \le \mu$,

$$\|Z\|_2 \le \|Z\|_F = \sqrt{(m-r)r\mu^2} \quad (1 < \mu < 2).$$

Then,

$$\|U_i\|_2 = \sqrt{1 + \|Z\|_2^2} \le \sqrt{1 + (m-r)r\mu^2} = O(r).$$

For parent nodes, the proving process is similar. ∎

To prove the main result, we further need the following two lemmas.

LEMMA 4.2 (Theorem 1 in [44]). *Let A be an $m \times n$ matrix with singular values $\sigma_1 \ge \sigma_2 \ge \cdots \sigma_{\min(m,n)}$. Let B be a $p \times q$ submatrix of A with singular values*

$$\hat{\sigma}_1 \ge \hat{\sigma}_2 \ge \cdots \hat{\sigma}_{\min(p,q)}.$$

Then $\sigma_i \ge \hat{\sigma}_i$ for $i = 1, 2, \ldots, \min(p, q)$.

LEMMA 4.3 (Theorem 3.8 in [40]). *For a matrix $A \in R^{m\times n}$ and any integer k $(1 \le k < \min(m,n))$, there exist permutation matrices Γ and Π such that*

$$\Gamma^T A\Pi \equiv \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix} = \begin{bmatrix} I_k & 0 \\ Z & I_{n-k} \end{bmatrix} \begin{bmatrix} B_{11} & B_{12} \\ 0 & U_{22} \end{bmatrix}, \tag{4.1}$$

where $Z = B_{21}B_{11}^{-1}$, $U_{22} = B_{22} - ZB_{12}$, and

$$\sigma_k(A) \ge \sigma_{\min}(B_{11}) \ge \frac{1}{k(m-k)\mu^2+1}\sigma_k(A), \tag{4.2}$$

$$\sigma_{k+1}(A) \le \|U_{22}\|_2 \le (k(m-k)\mu^2+1)\sigma_{k+1}(A). \tag{4.3}$$

Lemma 4.2 shows the well-known interlacing property of singular values, and Lemma 4.3 shows some error bounds of SRRSC. We further assume that every HSS block H_i satisfies

$$\sigma_{r+1}(H_i) \le \tau\sigma_1(H_i), \tag{4.4}$$

where r is the HSS rank of A and τ is a constant of machine precision.

With the techniques similar to those in [47], we can show that the approximation error of HSS construction algorithm in 2-norm. We leave the proof to the supplementary materials [33].

THEOREM 4.4. *Let A be an HSS approximation to matrix C, and A is computed by SRRSC. Then*

$$C = A + \mathcal{E},$$

where $\|\mathcal{E}\|_2 \le 4\tau\eta^{2L+1}\cdot\frac{1}{\eta^2-2}\cdot\|C\|_2$ and $\eta = 1 + (m-r)r\mu^2$.

COROLLARY 4.5. *If the HSS matrix A is constructed by using the truncated SVD algorithm, the approximate error bound becomes*

$$\|\mathcal{E}\|_2 \le 2(2^{L+1}-2)\tau\sigma_1(C) \le 4n\tau\sigma_1(C). \tag{4.5}$$

5. The implementation details of ADC. Our codes are written in Fortran 90 and modified from the LAPACK routines by adding HSS techniques to them.

The LAPACK routine `dbdsdc` implements a divide-and-conquer algorithm for the bidiagonal SVD problem and it has three options: (1) only computing the singular values (calling `dlasda`); (2) computing both the singular values and singular vectors (calling `dlasd0`); (3) computing the singular values and singular vectors in compacted form (calling `dlasda`). The LAPACK routine `dlasd1` calls `dlasd2` to deflate the matrix M and then calls `dlasd3` to compute the singular values and update the singular vectors via matrix-matrix multiplications.

When the singular vectors are computed explicitly, `dbdsdc` could be modified by the techniques introduced in this paper. We mainly modify `dlasd1` and rename it `mdlasd1`. If the size of the secular equation returned from `dlasd2` is small, `mdlasd1` calls `dlasd3` to compute the singular values and to update the singular vectors, otherwise it calls `mdlasd3` instead. The routine `mdlasd3`, similar to the LAPACK routine `dlasd8`, computes the new singular values w_i, the recomputed $\hat{z}_i$, $\gamma_i = w_i - d_i$, and $\mu_i = d_{i+1} - w_i$. Then the singular vector matrices are approximated by HSS matrices and updated by using HSS matrix multiplication algorithms [37].

5.1. The update of singular vector matrices. As shown in (2.10), the left singular vector matrix U can be written as

$$U = \hat{U} + \mathbf{e}_1 \cdot \beta^T, \tag{5.1}$$

where $\hat{U}(i,j) = \frac{u_i v_j}{d_i^2 - w_j^2}$ with $u_1 = 0$, and $\beta = \begin{bmatrix}\beta_1 & \beta_2 & \cdots & \beta_N\end{bmatrix}^T$. The Cauchy-like matrix $\hat{U}(i,j)$ is approximated by an HSS matrix H_U. Then the left singular vector matrix can be updated by

$$\begin{bmatrix} 0 & U_1 & \\ 1 & & \\ & & U_2 \end{bmatrix} U \approx \begin{bmatrix} 0 & U_1 & \\ 1 & & \\ & & U_2 \end{bmatrix} \cdot H_U + e_{K+1} \cdot \beta^T, \tag{5.2}$$

where $e_{K+1} = [0, \ldots, 0, 1, 0, \ldots, 0]^T$. Similarly, the right singular vector matrix is updated by $H_V \cdot \begin{bmatrix} V_1^T & \\ & V_2^T \end{bmatrix}$, where H_V is an HSS approximation to V^T in (2.10).

Note that `dbdsdc` exploits the sparse structure of $\begin{bmatrix} V_1^T & \\ & V_2^T \end{bmatrix}$ or $\begin{bmatrix} 0 & U_1 \\ 1 & \\ & U_2 \end{bmatrix}$ to reduce the number of floating-point operations. We cannot use this technique since permutations may destroy the low-rank structures of U and V^T.

6. Numerical results. The following examples are tested on a server with 64 GB memory and Intel Xeon(R) CPU E5-2670. For compilation we used Intel Fortran compiler (`ifort`), set the optimization flag `-O2`, and then linked the codes to the optimized dense linear algebra software, Intel MKL (composer_xe_2013.0.079).

The following example shows that by using HSS matrix techniques the multiplications of a Cauchy-like matrix with a general matrix can be much faster than the plain matrix-matrix multiplications. The baseline is from Intel MKL.

Example 2. Assume that a Cauchy-like matrix A is defined as in Example 1 with dimension N and that Q is an $N \times N$ orthogonal matrix. The elapsed times of using the LAPACK routine `dgemm` to multiply A with Q are shown in the row of Table 2 denoted by `dgemm`. The elapsed times of using HSS techniques are shown in the row denoted by HSS, which includes the *HSS construction* times and HSS matrix *multiplication* times. All the times are measured in seconds. Let H denote the HSS approximation to A and define $C = HQ$ and $B = AQ$. The maximum absolute error between C and B is shown in the row denoted by $\|B - C\|_{\max}$. The speedup rates are shown in the bottom row of Table 2.

The results in Table 2 show that the speedup of HSS matrix multiplications over plain matrix-matrix multiplications grows linearly with respect to N. Note that the HSS construction algorithm is much faster than the HSS matrix multiplication algorithm. That is because HSS construction only works on $O(N)$ data which need many fewer data exchanges between the fast and slow memory [3].

TABLE 2
Example 2: *The accuracy and efficiency of using HSS matrix approximations.*

N		1000	2000	4000	8000	16000
HSS	const.	1.6e-02	8.6e-02	3.4e-01	1.3e+00	5.1e+00
	multi.	3.9e-02	1.9e-01	7.6e-01	3.3e+00	1.4e+01
`dgemm`		1.1e-01	8.4e-01	6.6e+01	5.2e+01	4.2e+02
$\|B - C\|_{\max}$		1.5e-14	2.5e-14	8.9e-14	1.3e-13	2.5e-13
Speedup		2.0x	3.0x	6.0x	11.4x	22.2x

TABLE 3
Example 2: *The comparison of Algorithm* 3 *with other HSS construction algorithms.*

N	1000	2000	4000	8000	16000
Hss_svd	1.3e-01	4.7e-01	1.8e-00	7.1e-00	2.9e+01
$\|A - H_1\|_{\max}$	2.1e-13	3.4e-13	1.3e-12	2.5e-12	7.7e-12
Algorithm 3	1.8e-01	2.2e-01	2.8e-00	1.1e-00	4.1e+00
$\|A - H_2\|_{\max}$	1.2e-14	4.1e-14	6.0e-14	1.3e-13	3.2e-13
Speedup	7.3x	6.5x	6.4x	6.6x	7.0x

TABLE 4
Some bidiagonal matrices.

	Descriptions of the upper bidiagonal matrices	Source
1	diagonal entries 2.0 and off-diagonal entries 1.0	
2	Cholesky factor of tridiagonal (1,2,1) matrix	[38]
3	Cholesky factor of Clement-type matrix	[38]
4	Cholesky factor of Legendre-type matrix	[38]
5	Cholesky factor of Hermite-type matrix	[38]

The general HSS construction algorithms [8] form matrix A explicitly and therefore they work on $O(N^2)$ data. In this example we also show that Algorithm 3 can be much faster than the construction algorithms in [8, 48]. The results are shown in Table 3, where we use the following notation:

- `Hss_svd` denotes the HSS construction algorithm in [48] when the truncated SVD is used for compressions and it also denotes the elapsed time cost by itself;
- $\|A - H_1\|_{\max}$ denotes the maximum error between A and the HSS matrix constructed by `Hss_svd`;
- `Algorithm 3` also denotes the elapsed time cost by itself;
- $\|A - H_2\|_{\max}$ denotes the maximum error between A and the HSS matrix constructed by Algorithm 3;
- `Speedup` denotes the speedup of Algorithm 3 over `Hss_svd`.

Example 3. For some matrices [38, 12], very few or no singular values in DC are deflated. The matrices of this example are included in Table 4; see Appendix B for their definitions.

When the size of a secular equation is large, we use HSS matrices to approximate its singular vector matrices. In our implementation, a secular equation is called large as long as its size goes beyond 2000 which is a parameter depending on the optimized BLAS, the compiler, and the architecture of computers. The speedup effects of ADC over DC implemented in MKL are shown in Figure 3.

From this figure, we can see that the speedup of ADC over DC grows linearly. To show the accuracy of the computed singular values, we compare them with those obtained by the dqds algorithms [14, 42, 34] which are believed to have high relative accuracy. The maximum relative errors are listed in Table 5. The backward errors of the complete SVD computed by ADC are included in Table 6, i.e., $\|B - U\Sigma V^T\|_{\max}$, where B is a bidiagonal matrix, $U\Sigma V^T$ is its SVD, and $\|\cdot\|_{\max}$ denotes the maximum norm of matrix.

Example 4. To show the numerical stability of ADC, we further use all the matrices collected in the LAPACK tester `stetester` [38] to test it. There are about 400 matrices of various properties in `stetester`. The maximum relative errors of the

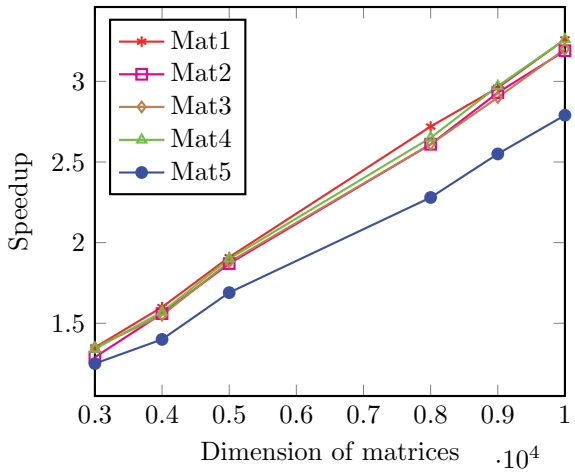


FIG. 3. *The speedups of ADC over DC for matrices in Table* 4.

TABLE 5
Example 3: *The maximum relative errors of computed singular values, k denotes one thousand.*

Matrix	3k × 3k	4k × 4k	5k × 5k	8k × 8k	9k × 9k	10k × 10k
1	8.82*e*-16	9.04*e*-16	8.88*e*-16	9.36*e*-16	1.55*e*-15	8.84*e*-16
2	1.29*e*-15	5.52*e*-15	1.27*e*-15	2.76*e*-15	3.11*e*-15	3.80*e*-15
3	1.75*e*-15	2.32*e*-15	2.57*e*-15	4.39*e*-15	3.94*e*-15	4.69*e*-15
4	1.43*e*-15	6.74*e*-16	6.03*e*-16	7.94*e*-16	1.35*e*-15	8.53*e*-16
5	7.68*e*-16	6.67*e*-16	6.11*e*-16	8.03*e*-16	9.08*e*-16	7.18*e*-16

TABLE 6
Example 3: *The backward errors of computed SVD, k denotes one thousand.*

Matrix	3k × 3k	4k × 4k	5k × 5k	8k × 8k	9k × 9k	10k × 10k
1	1.82*e*-14	1.13*e*-14	2.18*e*-14	2.02*e*-14	3.20*e*-14	3.15*e*-14
2	5.22*e*-15	8.99*e*-15	7.11*e*-15	1.60*e*-14	1.22*e*-14	1.62*e*-14
3	2.49*e*-13	4.90*e*-13	4.76*e*-13	4.90*e*-13	8.95*e*-13	8.85*e*-13
4	5.27*e*-13	1.21*e*-12	8.93*e*-13	2.26*e*-12	2.33*e*-12	1.93*e*-12
5	2.70*e*-13	4.33*e*-13	6.68*e*-13	8.81*e*-13	7.71*e*-13	1.36*e*-12

singular values computed by ADC compared with those by `dbdsdc` are all lower than 7.7*e*-16, most of which are exactly the same in double precision.

When comparing the elapsed times with Intel MKL, we use HSS techniques only when the size of a secular equation is larger than 2000. Thus the HSS techniques help for 27 of them. The dimensions of these matrices and the speedups of ADC over DC in Intel MKL are shown in Figure 4.

7. Conclusion. In this paper we modify a classical divide-and-conquer (DC) algorithm for the bidiagonal SVD problem, denoted by ADC, which needs many fewer flops than DC for large matrices with few deflations. The main idea is to use HSS matrices to approximate some intermediate singular vector matrices and update them via HSS matrix multiplications. Comparison with the optimized numerical linear algebra libraries such as Intel MKL, ADC shows it could be more than 3x faster for some large matrices. The DC algorithm for the symmetric tridiagonal eigenvalue problem could be improved in a similar way.

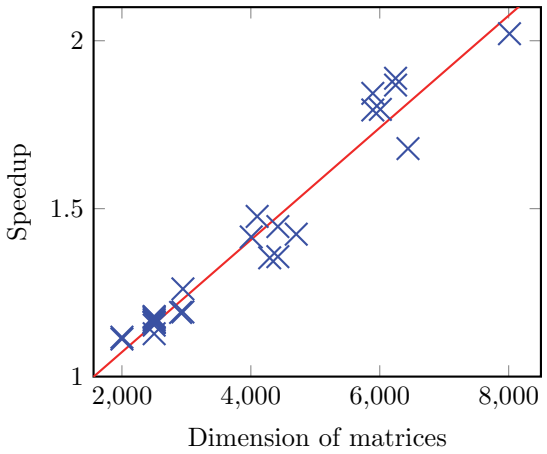


FIG. 4. *The speedups of ADC over DC for matrices in Table* 4.

Appendix A. Proof of Theorems 3.2 and 3.4. For a Cauchy-like matrix $A \in R^{m\times n}$, one step of Gaussian elimination reduces A to

$$A = \begin{bmatrix} \gamma_1 & g^T \\ r & A_{22} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ l & I \end{bmatrix} \begin{bmatrix} \gamma_1 & g^T \\ 0 & A^{(1)} \end{bmatrix}, \tag{A.1}$$

where $l = r/\gamma_1$ and $A^{(1)} = A_{22} - l\cdot g^T$ is the Schur complement. The following lemma shows that the Schur complement $A^{(1)}$ has displacement structures; see [16] and [20] for some more general results.

LEMMA A.1. *Let A satisfy the displacement equation*

$$DA - AW = u\cdot v^T, \tag{A.2}$$

with $D = \mathrm{diag}(d_1, D_2) \in R^{m\times m}$, $W = \mathrm{diag}(w_1, W_2) \in R^{n\times n}$, $u = (u_1, \ldots, u_m)^T \in R^m$, and $v = (v_1, \ldots, v_n)^T \in R^n$. Assume that $\gamma_1 \neq 0$. Then the Schur complement satisfies that

$$D_2A^{(1)} - A^{(1)}W_2 = u^{(1)}(2:m)v^{(1)}(2:n)^T, \tag{A.3}$$

where $u^{(1)}(2:m) = u(2:m) - l\cdot u_1$ and $v^{(1)}(2:n) = v(2:n) - v_1\cdot g^T/\gamma_1$.

Proof of Theorem 3.2. It can be proved by induction. By Lemma A.1, we know that Theorem 3.2 is true for $k = 1$. Assume that it is true for $k-1$,

$$D_kA^{(k-1)} - A^{(k-1)}W_k = u^{(k-1)}(k:n)\cdot v^{(k-1)T}(k:n). \tag{A.4}$$

From (A.1) and Lemma A.1, we have that, for any $1 \le \ell \le n-k$,

$$u^{(k)}(k+\ell) = u^{(k-1)}(k+\ell) - l^{(k-1)}(\ell)\cdot u^{(k-1)}(k), \tag{A.5}$$

where

$$l^{(k-1)}(\ell) = \frac{r^{(k-1)}(\ell)}{\gamma_{k-1}} = \frac{u^{(k-1)}(k+\ell)v^{(k-1)}(k)}{(d_{k+\ell} - w_k)\gamma_{k-1}}, \quad \gamma_{k-1} = \frac{u^{(k-1)}(k)\cdot v^{(k-1)}(k)}{d_k - w_k},$$

$r^{(k-1)}$ is the first column of $A^{(k-1)}$, and γ_{k-1} is the top-left entry of $A^{(k-1)}$.

By substituting these two terms into (A.5), we get

$$u^{(k)}(k+\ell) = u^{(k-1)}(k+\ell)\left(1 - \frac{d_k - w_k}{d_{k+\ell} - w_k}\right) = u^{(k-1)}(k+\ell)\frac{d_{k+\ell} - d_k}{d_{k+\ell} - w_k}.$$

The proof of (3.5) is quite similar. ∎

The generators of $Z^{(k)}$ can also be computed recursively. To show that, we first prove $Z^{(k)}$ is a Cauchy-like matrix.

LEMMA A.2. *Under the same assumptions as in Theorem* 3.2, *the matrix* $Z^{(k)}$ *in* (3.1) *is a Cauchy-like matrix satisfying*

$$D_2 Z^{(k)} - Z^{(k)} D_1 = x \cdot y^T, \tag{A.6}$$

where $Z^{(k)} \in R^{(n-k)\times k}$, $D_2 = \mathrm{diag}(d_{k+1}, \ldots, d_n)$, $D_1 = \mathrm{diag}(d_1, \ldots, d_k)$, $x \in R^{n-k}$, *and* $y \in R^k$ *are suitable vectors.*

Proof. It is easy to see that

$$D_1 A_{11} - A_{11} W_1 = U_1 V_1^T, \tag{A.7}$$

$$D_2 A_{21} - A_{21} W_1 = U_2 V_1^T, \tag{A.8}$$

where $U_1 = (u_1, \ldots, u_k)^T$, $U_2 = (u_{k+1}, \ldots, u_n)^T$, and $V_1 = (v_1, \ldots, v_k)^T$.

From (A.7), we have

$$A_{21} A_{11}^{-1} D_1 = A_{21} W_1 A_{11}^{-1} + A_{21} A_{11}^{-1} U_1 \cdot V_1^T A_{11}^{-1}. \tag{A.9}$$

Similarly, from (A.8), we get

$$D_2 A_{21} A_{11}^{-1} = U_2 \cdot V_1^T A_{11}^{-1} + A_{21} W_1 A_{11}^{-1}. \tag{A.10}$$

Subtracting (A.9) from (A.10) yields

$$D_2 Z^{(k)} - Z^{(k)} D_1 = (U_2 + Z^{(k)} U_1) \cdot (V_1^T A_{11}^{-1}) := x \cdot y^T. \tag{A.11}$$ ∎

By Lemma 3.5, the entries of $Z^{(k)}$ have the following closed formulas.

LEMMA A.3. *Assume that a Cauchy-like matrix A satisfies*

$$DA - AW = u \cdot v^T, \quad A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix},$$

with $D = \mathrm{diag}(d_1, \ldots, d_n)$, $W = \mathrm{diag}(w_1, \ldots, w_n)$, $u, v \in R^n$, *and* $A_{11} \in R^{k\times k}$. *For* $1 \le l \le n-k, 1 \le i \le k$, *the* (ℓ, i)*th entry of* $Z^{(k)} = A_{21} A_{11}^{-1}$ *is*

$$Z_{\ell,i}^{(k)} = \prod_{j=1}^{k} \frac{d_i - w_j}{d_{k+\ell} - w_j} \cdot \prod_{j=1, j\neq i}^{k} \frac{d_{k+\ell} - d_j}{d_i - d_j} \cdot \frac{u_{k+\ell}}{u_i}. \tag{A.12}$$

Proof. By Lemma 3.5, we know $Z_{\ell,i}^{(k)} = \frac{\det(\tilde{A}_{11})}{\det(A_{11})}$, where $\tilde{A}_{11}$ is the top-left $k \times k$ submatrix of A after interchanging rows i and $k + \ell$.

From (3.2) and the fact $A = \mathrm{diag}(u) \cdot C \cdot \mathrm{diag}(v)$, where $C = (\frac{1}{d_i - w_j})$, substitute (3.2) into (3.9) and we can get (A.12). ∎

Now we show how to compute $Z^{(k+1)}$ recursively. By Lemma A.3, matrices $Z^{(k)}$ and $Z^{(k+1)}$ have the following relationship:

$$\frac{Z^{(k+1)}(\ell, i)}{Z^{(k)}(\ell+1, i)} = \frac{d_{k+1} - d_{k+\ell+1}}{w_{k+1} - d_{k+\ell+1}} \cdot \frac{w_{k+1} - d_i}{d_{k+1} - d_i} \tag{A.13}$$

for $1 \le i \le k$ and $1 \le \ell \le n - k - 1$.

Proof of Theorem 3.4 It is easy to verify that Theorem 3.4 is true for $k = 1$. We prove it is true for $k > 1$. Assume

$$D_2^{(k-1)} Z^{(k-1)} - Z^{(k-1)} D_1^{(k-1)} = u^{(k-1)}(k:n) \cdot y^{(k-1)}(1:k-1). \tag{A.14}$$

From (3.4), (A.13), and (A.14), we have that, for $1 \le i \le k-1$,

(A.15)
$$\begin{aligned} d_{k+\ell} Z^{(k)}(\ell, i) - Z^{(k)}(\ell, i) d_i &= \alpha_{k-1}\beta_{k-1}(d_{k+\ell} Z^{(k-1)}(\ell+1, i) - Z^{(k-1)}(\ell+1, i) d_i) \\ &= u^{(k)}(k+\ell) \cdot \beta_{k-1} y^{(k-1)}(i) := u^{(k)}(k+\ell) \cdot y^{(k)}(i), \end{aligned}$$

where $\alpha_{k-1} = \frac{d_k - d_{k+\ell}}{w_k - d_{k+\ell}}$, $\beta_{k-1} = \frac{w_k - d_i}{d_k - d_i}$.

Now we prove it is true for $i = k$. By Lemma A.3,

$$\begin{aligned} Z^{(k)}(\ell, k) &= \prod_{j=1}^{k} \frac{d_k - w_j}{d_{k+\ell} - w_j} \cdot \prod_{j=1}^{k-1} \frac{d_{k+\ell} - d_j}{d_k - d_j} \cdot \frac{u_{k+\ell}}{u_i} \\ &= u_{k+\ell} \prod_{j=1}^{k} \frac{d_{k+\ell} - d_j}{d_{k+\ell} - w_j} \prod_{j=1}^{k-1} \frac{d_k - w_j}{d_k - d_j} \cdot \frac{d_k - w_k}{u_k} \cdot \frac{1}{d_{k+\ell} - d_k} \\ &= u^{(k)}(k+\ell) \cdot y^{(k)}(k) \cdot \frac{1}{d_{k+\ell} - d_k}, \end{aligned} \tag{A.16}$$

$y^{(k)}(k)$ is defined as in (3.7). ∎

Corollary A.4. *The generator $y^{(k)}$ of $Z^{(k)}$ has the following closed form,*

$$y^{(k)}(i) = \frac{\prod_{j=1}^{k}(d_i - w_j)}{\prod_{j=1, j\neq i}^{k}(d_i - d_j)} \cdot \frac{1}{u^{(0)}(i)}. \tag{A.17}$$

Appendix B. Some tridiagonal matrices. Several matrix classes with few deflations are given in [38]. The Clement-type matrix is given by

$$T = \operatorname{tridiag}\begin{pmatrix} n & \sqrt{1(n-1)} & n & \sqrt{2(n-2)} & \cdots & \sqrt{(n-1)1} & n \\ & \sqrt{1(n-1)} & & \sqrt{2(n-2)} & & \sqrt{(n-1)1} & \end{pmatrix}, \tag{B.1}$$

where the off-diagonal entries are $\sqrt{i(n-i)}, i = 1, \ldots, n-1$.

The Legendre-type matrix is defined as [38, 1],

$$T = \operatorname{tridiag}\begin{pmatrix} n & 2/\sqrt{3\cdot 5} & n & 3/\sqrt{5\cdot 7} & \cdots & n/\sqrt{(2n-1)(2n+1)} & n \\ & 2/\sqrt{3\cdot 5} & & 3/\sqrt{5\cdot 7} & & n/\sqrt{(2n-1)(2n+1)} & \end{pmatrix}, \tag{B.2}$$

where the off-diagonal entries are $i/\sqrt{(2i-1)(2i+1)}, i = 2, \ldots, n$.

The Hermite-type matrix is given as [38],

$$T = \operatorname{tridiag}\begin{pmatrix} n & \sqrt{1} & n & \sqrt{2} & \cdots & \sqrt{n-2} & n & \sqrt{n-1} & n \\ & \sqrt{1} & & \sqrt{2} & & \sqrt{n-2} & & \sqrt{n-1} & \end{pmatrix}. \tag{B.3}$$

Acknowledgments. The authors are very grateful to the anonymous referees, and would like to acknowledge many helpful discussions with J. Demmel, W. Kahan, and B. Parlett. The authors also thank J. Xia for some valuable suggestions.

REFERENCES

[1] M. Abramowitz and I. A. Stegun, *Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables*, 9th ed., Dover, New York, 1965.
[2] P. Arbenz and G. H. Golub, *On the spectral decomposition of Hermitian matrices modified by low rank perturbations with applications*, SIAM J. Matrix Anal. Appl., 9 (1988), pp. 40–58.
[3] G. Ballard, J. Demmel, O. Holtz, and O. Schwartz, *Minimizing communication in numerical linear algebra*, SIAM J. Matrix Anal. Appl., 32 (2011), pp. 886–901.
[4] D. Bini and V. Pan, *Polynomial and Matrix Computations, Volume* I: *Fundamental Agorithms*, Process in Theoretical Computer Science 12, Birkhäuser, Boston, 1994.
[5] D. Braess and W. Hackbusch, *Approximation of* $1/x$ *by exponential sums in* $[1, \infty)$, IMA J. Numer. Anal., 25 (2005), pp. 685–697.
[6] 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.
[7] 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 (2007), pp. 67–81.
[8] S. Chandrasekaran, M. Gu, and T. Pals, *A fast ULV decomposition solver for hierarchically semiseparable representations*, SIAM J. Matrix Anal. Appl., 28 (2006), pp. 603–622.
[9] H. Cheng, Z. Gimbutas, P. Martinsson, and V. Rokhlin, *On the compression of low rank matrices*, SIAM J. Sci. Comput., 26 (2005), pp. 1389–1404.
[10] J. Demmel, *Applied Numerical Linear Algebra*, SIAM, Philadelphia, 1997.
[11] J. Demmel, *Accurate singular value decompositions of structured matrices*, SIAM J. Matrix Anal. Appl., 21 (2000), pp. 562–580.
[12] J. Demmel, O. Marques, B. Parlett, and C. Vömel, *Performance and accuracy of LAPACK's symmetric tridiagonal eigensolvers*, SIAM J. Comput., 30 (2008), pp. 1508–1526.
[13] A. Dutt, M. Gu, and V. Rokhlin, *Fast algorithms for polynomial interpolation, integration, and differentiation*, SIAM J. Numer. Anal., 33 (1996), pp. 1689–1711.
[14] K. V. Fernando and B. N. Parlett, *Accurate singular values and differential qd algorithms*, Numer. Math., 67 (1994), pp. 191–229.
[15] A. Gerasoulis, *A fast algorithm for the multiplication of generalized Hilbert matrices with vectors*, Math. Comp., 50 (1988), pp. 179–188.
[16] I. Gohberg, T. Kailath, and V. Olshevsky, *Fast Gaussian elimination with partial pivoting for matrices with displacement structure*, Math. Comp., 64 (1995), pp. 1557–1576.
[17] G. H. Golub, *Trummer problem*, SIGACT News, 17 (1985), p. 12.
[18] G. H. Golub and C. F. Van Loan, *Matrix Computations*, 3rd ed., The Johns Hopkins University Press, Baltimore, MD, 1996.
[19] L. Greenngard and V. Rokhlin, *A fast algorithm for particle simulations*, J. Comput. Phys., 73 (1987), pp. 325–348.
[20] M. Gu, *Stable and efficient algorithms for structured systems of linear equations*, SIAM J. Matrix Anal. Appl., 19 (1998), pp. 279–306.
[21] 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.
[22] M. Gu and S. C. Eisenstat, *Efficient algorithms for computing a strong-rank revealing QR factorization*, SIAM J. Sci. Comput., 17 (1996), pp. 848–869.
[23] M. Gu and S. C. Eisenstat, *A divide-and-conquer algorithm for the bidiagonal SVD*, SIAM J. Matrix Anal. Appl., 16 (1995), pp. 79–92.
[24] 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.
[25] M. Gu and J. Xia, *A numerically stable superfast Toeplitz solver*, Berkeley Matrix Computations and Scientific Computing Seminar, http://math.berkeley.edu/~mgu/Seminar/Fall2009/ToeplitzSeminarTalk.pdf (2009).
[26] M. Gu and J. Xia, *A multi-structured stable and superfast Toeplitz solver*, preprint, 2009.
[27] W. Hackbusch, *A sparse matrix arithmetic based on* $\mathcal{H}$*-matrices. Part* I*: Introduction to* $\mathcal{H}$*-matrices*, Computing, 62 (1999), pp. 89–108.
[28] W. Hackbusch and S. Börm, *Data-sparse approximation by adaptive* $\mathcal{H}^2$*-matrices*, Computing, 69 (2002), pp. 1–35.

[29] N. Halko, P. G. Martinsson, and J. A. Tropp, *Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions*, SIAM Rev., 53 (2011), pp. 217–288.
[30] G. Heinig, *Inversion of generalized Cauchy matrices and other classes of structured matrices*, in Linear Algebra in Signal Processing, IMA Vol. Math. Appl., 69 (1994), pp. 95–114.
[31] E. R. Jessup and D. C. Sorensen, *A parallel algorithm for computing the singular value decomposition of a matrix*, SIAM J. Matrix Anal. Appl., 15 (1994), pp. 530–548.
[32] R.-C. Li, *Solving secular equations stably and efficiently*, LAPACK Working Note 89, University of California, Berkeley, CA, 1994.
[33] S. Li, M. Gu, L. Cheng, X. Chi, and M. Sun, *Supplementary materials*, to an Accelerated Divide-and-Conquer Algorithm for the Bidiagonal SVD Problem, http://simax.siam.org/simax_files/2014/04/16/00055619/02/55619_2_supp_1_n45t74_sc.pdf (2014).
[34] S. Li, M. Gu, and B. N. Parlett, *An improved dqds algorithm*. SIAM J. Sci. Comput., 36 (2014), pp. C290–C308.
[35] S. Li, M. Gu, C. J. Wu, and J. Xia, *New efficient and robust HSS Cholesky factorization of SPD matrices*, SIAM J. Matrix Anal. Appl., 33 (2012), pp. 886–904.
[36] K. Löwner, *Über monotone Matrixfunktionen*, Math. Z., 38 (1934), pp. 177–216.
[37] W. Lyons, *Fast Algorithms with Applications to PDEs*, Ph.D. thesis, University of California, Santa Barbara, CA, 2005.
[38] O. A. Marques, C. Voemel, J. W. Demmel, and B. N. Parlett, *Algorithm* 880: *A testing infrastructure for symmetric tridiagonal eigensolvers*, ACM Trans. Math. Softw., 35 (2008), 8.
[39] L. Miranian and M. Gu, *Strong rank revealing LU factorization*, Linear Algebra Appl., 367 (2003), pp. 1–16.
[40] C.-T. Pan, *On the existence and computation of rank revealing LU factorizations*, Linear Algebra Appl., 316 (2000), pp. 199–222.
[41] V. Y. Pan, *Structured Matrices and Polynomials: Unified Superfast Algorithms*, Birkhäuser, Boston, 2001.
[42] B. N. Parlett and O. Marques, *An implementation of the dqds algorithm (positive case)*, Linear Algebra Appl., 309 (2000), pp. 217–259.
[43] A. H. Sayed and T. Kailath, *Fast algorithms for generalized displacement structures and lossless systems*, Linear Algebra Appl., 219 (1995), pp. 49–78.
[44] R. C. Thompson, *Principal submatrix* IX: *Interlacing inequalities for singular values of submatrices*, Linear Algebra Appl., 5 (1972), pp. 1–12.
[45] S. Wang, X. S. Li, J. Xia, Y. Situ, and M. V. De Hoop, *Efficient scalable algorithms for solving dense linear systems with hierarchically semiseparable structures*, SIAM J. Sci. Comput., 35 (2013), pp. C519–C544.
[46] J. Wilkinson, *The Algebraic Eigenvalue Problem*, Oxford University Press, New York, 1965.
[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] J. Xia, S. Chandrasekaran, M. Gu, and X. S. Li, *Fast algorithm for hierarchically semiseparable matrices*, Numer. Linear Algebra Appl., 17 (2010), pp. 953–976.
[49] J. Xia and M. Gu, *Robust approximate Choleksy factorization of rank-structured symmetric positive definite matrices*, SIAM J. Matrix Anal. Appl., 31 (2010), pp. 2899–2920.
[50] N. Yarvin and V. Rokhlin, *An improved fast multipole algorithm for potential fields on the line*, SIAM J. Numer. Anal., 36 (1999), pp. 629–666.