SIAM J. MATRIX ANAL. APPL.
Vol. 33, No. 3, pp. 837–858

A SUPERFAST STRUCTURED SOLVER FOR TOEPLITZ LINEAR SYSTEMS VIA RANDOMIZED SAMPLING*

JIANLIN XIA†, YUANZHE XI†, AND MING GU‡

Abstract. We propose a superfast solver for Toeplitz linear systems based on rank structured matrix methods and randomized sampling. The solver uses displacement equations to transform a Toeplitz matrix T into a Cauchy-like matrix $\mathcal{C}$, which is known to have low-numerical-rank off-diagonal blocks. Thus, we design a fast scheme for constructing a hierarchically semiseparable (HSS) matrix approximation to $\mathcal{C}$, where the HSS generators have internal structures. Unlike classical HSS methods, our solver employs randomized sampling techniques together with fast Toeplitz matrix-vector multiplications, and thus converts the direct compression of the off-diagonal blocks of $\mathcal{C}$ into the compression of much smaller blocks. A strong rank-revealing QR factorization method is used to generate/preserve certain special structures, and also to ensure stability. A fast ULV HSS factorization scheme is provided to take advantage of the special structures. We also propose a precomputation procedure for the HSS construction so as to further improve the efficiency. The complexity of these methods is significantly lower than some similar Toeplitz solvers for large matrix size n. Detailed flop counts are given, with the aid of a rank relaxation technique. The total cost of our methods includes $O(n)$ flops for HSS operations and $O(n\log^2 n)$ flops for matrix multiplications via FFTs, where n is the order of T. Various numerical tests on classical examples, including ill-conditioned ones, demonstrate the efficiency, and also indicate that the methods are stable in practice. This work shows a practical way of using randomized sampling in the development of fast rank structured methods.

Key words. superfast Toeplitz solver, hierarchically semiseparable matrix, randomized sampling, structure-preserving rank-revealing factorization, precomputation

AMS subject classifications. 65F05, 65F30, 15A06

DOI. 10.1137/110831982

1. Introduction.

In this work, we consider the solution of Toeplitz linear systems. Toeplitz systems arise in many numerical and engineering applications, such as PDE and integral equation solutions, image and signal processing, time series analysis, orthogonal polynomials, etc. A Toeplitz system looks like

$$Tx = b, \tag{1.1}$$

where T is a Toeplitz matrix in the following form:

$$T = \begin{pmatrix} t_0 & t_{-1} & t_{-2} & \cdots & t_{-(n-1)} \\ t_1 & t_0 & t_{-1} & \ddots & \vdots \\ t_2 & t_1 & t_0 & \ddots & t_{-2} \\ \vdots & \ddots & \ddots & \ddots & t_{-1} \\ t_{n-1} & \cdots & t_2 & t_1 & t_0 \end{pmatrix}. \tag{1.2}$$

*Received by the editors April 25, 2011; accepted for publication (in revised form) by M. Van Barel April 17, 2012; published electronically August 2, 2012.
http://www.siam.org/journals/simax/33-3/83198.html

†Department of Mathematics, Purdue University, West Lafayette, IN 47907 (xiaj@math.purdue.edu, yxi@math.purdue.edu). The first author's research was supported in part by NSF grants DMS-1115572 and CHE-0957024.

‡Department of Mathematics, University of California, Berkeley, CA 94720 (mgu@math.berkeley.edu).

That is, the entries along every diagonal of T are constant. T is defined by a vector of $2n-1$ parameters $t_{-(n-1):n-1}$, called the *Toeplitz vector*, that generates T.

The system can be solved quickly. There are *fast solvers* that need about $O(n^2)$ flops, such as Schur-type and Levinson-type methods [28]. Methods requiring less than $O(n^2)$ costs (or roughly $O(n)$ costs) are called *superfast solvers*. However, some fast and superfast algorithms are unstable. See [4, 6, 14, 41] for some discussions. Stable generalized Schur algorithms [13, 27] and look-ahead algorithms [8] are proposed to enhance the numerical stability. Superfast solvers with stabilization also exist [44]. Preconditioning techniques are first studied in [35, 39]. See [7, 34] and a survey paper [9], as well as the references therein. Fast displacement-equation type algorithms in [15, 17, 21, 22, 23] are numerically or analytically shown to be stable. Some of these methods transform T into a Cauchy-like matrix $\mathcal{C}$ using displacement structures [26] and then solve $\mathcal{C}$.

1.1. Displacement structures and a low-rank property. Displacement structures are first introduced in [26]. Selected detailed studies can be found in [15, 21, 24, 36, 38]. It is straightforward to verify that the following Sylvester-type displacement equation for T holds:

$$Z_1 T - T Z_{-1} = G H^T, \tag{1.3}$$

where $G \in \mathbb{R}^{n\times d}$, $H \in \mathbb{R}^{n\times d}$, and $Z_t = \left(\begin{smallmatrix} 0 & t \\ I_{n-1} & 0 \end{smallmatrix}\right)$, with I_{n-1} the size $n-1$ identity matrix. Here, $d = 2$ is called the displacement rank. Since Z_1 is a circulant matrix, it can be diagonalized by the normalized inverse discrete Fourier transform matrix

$$\mathcal{F}_n = \frac{1}{\sqrt{n}}(\omega^{2(i-1)(j-1)})_{1\le i,j\le n}, \quad \omega = e^{\frac{\pi \mathbf{i}}{n}}, \quad \mathbf{i} = \sqrt{-1}. \tag{1.4}$$

This leads to another displacement equation [21],

$$\mathcal{D}_1 \mathcal{C} - \mathcal{C}\mathcal{D}_{-1} = \hat{G}\hat{H}^T, \tag{1.5}$$

where

$$\begin{gathered} \mathcal{C} = \mathcal{F} T \mathcal{D}_0^H \mathcal{F}^H, \quad \hat{G} = \mathcal{F}G, \quad \hat{H} = \mathcal{F}\mathcal{D}_0 H, \\ \mathcal{D}_0 = \operatorname{diag}(1, \omega, \ldots, \omega^{n-1}), \quad \mathcal{D}_1 = \operatorname{diag}(1, \omega^2, \ldots, \omega^{2(n-1)}), \quad \mathcal{D}_{-1} = \omega \mathcal{D}_1. \end{gathered} \tag{1.6}$$

Equation (1.5) indicates that $\mathcal{C}$ is a *Cauchy-like matrix* in the following form:

$$\mathcal{C} = \mathcal{F}_n T \mathcal{D}_0^H \mathcal{F}_n^H = \left(\frac{\hat{G}_i \hat{H}_j^T}{\omega^{2(i-1)} - \omega^{2j-1}} \right)_{1\le i,j\le n}, \tag{1.7}$$

where $\hat{G}_i$ is the ith row of $\hat{G}$, and $\hat{H}_j$ is the jth row of $\hat{H}$. Thus with FFTs, the system (1.1) can be converted into a Cauchy-like system

$$\mathcal{C}\mathbf{x} = \mathbf{b}, \tag{1.8}$$

where $\mathbf{x} = \mathcal{F}_n \mathcal{D}_0 x$ and $\mathbf{b} = \mathcal{F}_n b$. Such a strategy for Toeplitz solutions is first used in [21].

The matrix $\mathcal{C}$ and similar representations have a low-rank property [12, 33]. That is, its off-diagonal blocks have small numerical ranks for a given tolerance, which are bounded by $O(\log n)$. Based on this property, two superfast and numerically stable

Toeplitz solvers are developed in [12, 33]. The solvers compress the off-diagonal blocks of $\mathcal{C}$ or its variations and approximate $\mathcal{C}$ by rank structured matrices. (Here by *compression*, we mean the computation of a truncated SVD or a rank-revealing factorization.) The method in [12] first uses $O(n^2 \log n)$ flops to approximate the following *Cauchy matrix* in a precomputation stage:

$$\mathbf{C} = \left(\frac{1}{\omega^{2(i-1)} - \omega^{2j-1}} \right)_{1 \le i,j \le n}. \tag{1.9}$$

Then a rank structured approximation to $\mathcal{C}$, called a sequentially semiseparable (SSS) matrix [11], can be constructed with about $O(n \log^2 n)$ cost. After this, (1.1) can be solved quickly with about $O(n \log n)$ cost. The method in [33] has similar complexity.

1.2. Main results. The methods in [12, 33] need $O(n^2 \log n) \sim O(n^2 \log^2 n)$ flops for precomputations, and then $O(n \log^2 n)$ flops for the structure construction and solution for (1.8). They use only the low-rank property of $\mathcal{C}$ and ignore its Cauchy-like structure. Here, we consider further taking advantage of this special Cauchy-like structure (in fast matrix-vector multiplications) as well as special matrix forms (e.g., (3.10)–(3.11)), so as to reduce the costs of rank structured operations to $O(n)$ after an $O(n \log^2 n)$ cost for FFTs. We achieve this from several aspects.

One aspect is that we use randomized sampling together with fast Toeplitz matrix-vector multiplications. The ideas of randomized sampling for constructing rank structured approximations have been extensively studied, and are shown to be very useful in dealing with rank deficiency [29, 30, 31, 37] (and others). As one example, to compress a low-rank block Φ, multiply a random matrix X to Φ. The product ΦX is then compressed with rank-revealing factorizations, which gives the compression information of Φ [29]. For a given matrix with small off-diagonal ranks, the method in [31] uses $O(1)$ such matrix multiplications to approximate the matrix by a rank structured form called *hierarchically semiseparable* (HSS) matrix [10, 46]. An HSS matrix is a data-sparse form given by a sequence of internal small dense matrices (called *generators*). The method in [30] uses $O(\log n)$ multiplications without knowing the entries of the original matrix. Here, it is well known that Toeplitz matrix-vector products can be quickly computed [16]. Together with (1.7), we can compute the product of the Cauchy-like matrix $\mathcal{C}$ with an $n \times O(\log n)$ Gaussian random matrix X as needed here in $O(n \log^2 n)$ flops via FFTs. Then, we use a procedure similar to the one in [31] to approximate $\mathcal{C}$ by an HSS matrix.

Another aspect is that we use a special HSS form by incorporating additional structures into the HSS generators. The generators obtained from the off-diagonal block compression via randomized sampling has certain special structure (e.g., (3.5)). Unlike the method in [31] which uses an extra step to reorthonormalize an HSS form, we avoid this step to not only save the cost but also preserve the special structures. In fact, the special structures of the generators can be used to improve the HSS solution efficiency.

By taking advantage of both the Cauchy-like and additional structures, we gain significant improvements to classical HSS algorithms. Unlike some inversion-based algorithms, we propose special ULV-type [10, 46] HSS factorization and solution methods which are often more efficient. The complexity of all these algorithms is $O(n)$, and is significantly lower than both standard HSS methods [45] and the structured operations in the Toeplitz solvers in [12, 33]. We give detailed flop counts, which are not available in the Toeplitz solvers in [12, 33] or the randomized sampling methods in [30, 31]. We also use an idea of rank relaxation [45] so that even if the off-diagonal

rank bound of $\mathcal{C}$ is $O(\log n)$, we can still achieve truly $O(n)$ complexity in the HSS operations, in practice, instead of $O(n \log n)$ or more. The usage of strong RRQR factorizations leads to a stable process, which is verified by numerical experiments. A variety of Toeplitz examples, especially ill-conditioned ones, demonstrate both the efficiency and the accuracy of our methods.

We also discuss a precomputation procedure to first compress the off-diagonal blocks of $\mathbf{C}$, which depends on n only, instead of the entries of T. The information is then used in the HSS construction for $\mathcal{C}$ so as to further reduce or avoid the cost of multiplying $\mathcal{C}$ and the random matrix X. The total precomputation cost is only $O(n)$ (after FFTs), in contrast with $O(n^2 \log n)$ or $O(n^2 \log^2 n)$ in [12, 33]. This version is especially useful when there are multiple T of the same large size.

1.3. Outline. The remaining sections are organized as follows. Section 2 provides brief discussions of HSS representations and the low-rank property of $\mathcal{C}$. The HSS construction, factorization, and solution procedures for $\mathcal{C}$ are presented in section 3. Section 4 then briefly describes an HSS construction method with precomputations. A variety of numerical examples are shown in section 5, and we draw some concluding remarks in section 6. The following notation is used in the presentation:

- Let $\mathbf{I}_i$ be a subset of $\{1:n\} \equiv \{1, 2, \ldots, n\}$ with contiguous indices. We use $\mathbf{I}_i^c$ and $\mathbf{I}_i^r$ to denote the subsets of contiguous indices in $\{1:n\}$ that are smaller and larger than those in $\mathbf{I}_i$, respectively.
- $A|_{\mathbf{I}_i}$ is the submatrix formed by all rows of A with row index set $\mathbf{I}_i$. Also, $A|_{\mathbf{I}_i \times \mathbf{I}_j}$ is the submatrix of A with row index set $\mathbf{I}_i$ and column index set $\mathbf{I}_j$.
- $\mathrm{diag}(D_1, \ldots, D_k)$ is a diagonal matrix with diagonal blocks $D_1, \ldots, D_k$, and $\mathrm{diag}(v_{1:k})$ is a diagonal matrix with diagonal entries $v_1, v_2, \ldots, v_k$.

2. Hierarchically semiseparable matrices and the low-rank property.

2.1. Review of hierarchically semiseparable representations. HSS representations enable us to conveniently take advantage of the low-rank property of the Cauchy-like matrix in (1.7). An $n \times n$ HSS matrix A with postordering notation is defined hierarchically in terms of three components [46]:

(1) There is a full binary tree $\mathcal{T}$ with nodes labeled as $i = 1, 2, \ldots,$ which are postordered. That is, any nonleaf node i has a left child c_1 and a right child c_2, ordered as $c_1 < c_2 < i$.

(2) There is an index set $\mathbf{I}_i$ (with contiguous indices) associated with each node i of $\mathcal{T}$. The index sets are defined so that $\mathbf{I}_i = \{1, 2, \ldots, n\}$ for the root node $i = \mathrm{root}(\mathcal{T})$, and $\mathbf{I}_{c_1} \cup \mathbf{I}_{c_2} = \mathbf{I}_i$, $\mathbf{I}_{c_1} \cap \mathbf{I}_{c_2} = \emptyset$ for the children c_1 and c_2 of each nonleaf node i.

(3) There are a sequence of matrices D_i, U_i, V_i, R_i, W_i, B_i (called *HSS generators*) associated with the nodes $i = 1, 2, \ldots,$ so that for any nonleaf node i and its children c_1 and c_2,

$$D_i \equiv A|_{\mathbf{I}_i \times \mathbf{I}_i} = \begin{pmatrix} D_{c_1} & U_{c_1} B_{c_1} V_{c_2}^T \\ U_{c_2} B_{c_2} V_{c_1}^T & D_{c_2} \end{pmatrix}, \tag{2.1}$$

$$U_i = \begin{pmatrix} U_{c_1} & \\ & U_{c_2} \end{pmatrix} \begin{pmatrix} R_{c_1} \\ R_{c_2} \end{pmatrix}, \quad V_i = \begin{pmatrix} V_{c_1} & \\ & V_{c_2} \end{pmatrix} \begin{pmatrix} W_{c_1} \\ W_{c_2} \end{pmatrix}. \tag{2.2}$$

Initially, the U, V generators are defined for leaf nodes, and R, W generators are defined for all nodes. R, W are called *translation operators* and are used to recursively reconstruct the U, V generators associated with i from those

associated with c_1 and c_2 [10]. If $i = \text{root}(\mathcal{T})$, then $D_i \equiv A$, and U_i, V_i, R_i, W_i, B_i are set to be empty matrices (since root($\mathcal{T}$) is associated with a diagonal block which is the entire A, instead of any off-diagonal block).

Then A is said to be in an *HSS form* with the corresponding *HSS tree* $\mathcal{T}$. See Figure 2.1 for an example. Note that we use matrix transposes V^T in (2.1) even if A is complex. This is convenient for our operations below.

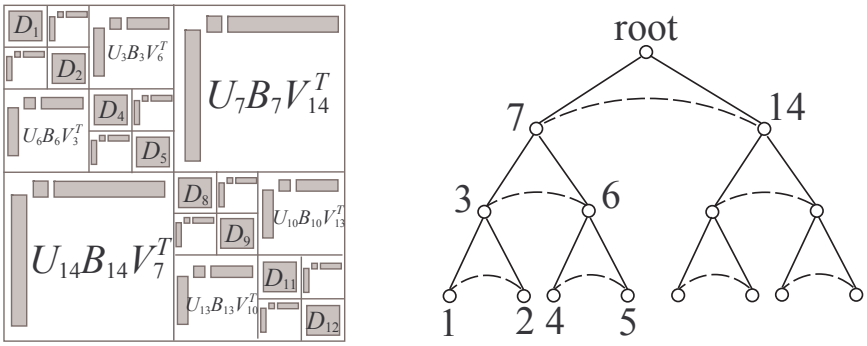


FIG. 2.1. *An HSS matrix and its HSS tree.*

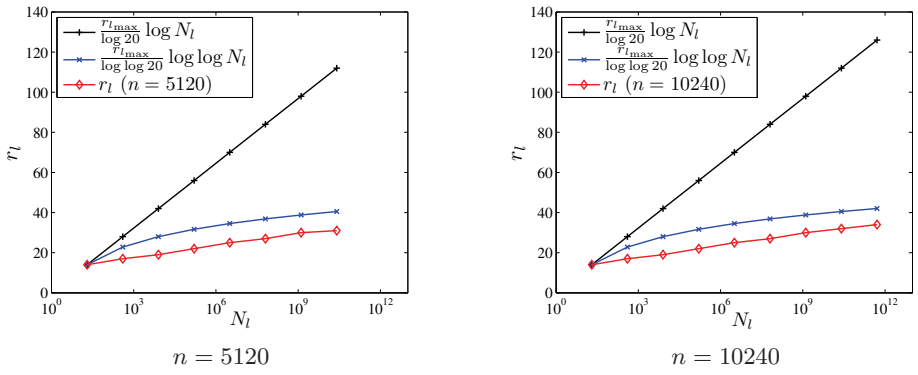


FIG. 2.2. *The numerical ranks r_l (versus the row sizes N_l) of the HSS block rows A_i^- at level l of the HSS partition for order n Cauchy matrices $\mathbf{C}$ in* (1.9)*, where the relative tolerance for computing the ranks is $\tau = 10^{-8}$, the bottom level HSS block row size is $N_{l_{\max}} = 20$, and for comparison purposes, we also plot shifted $\log N_l$ and $\log\log N_l$ curves.*

An HSS form can be constructed by compressing certain off-diagonal blocks, called *HSS blocks* of A associated with each node i of $\mathcal{T}$ and defined as

$$A_i^- = \begin{pmatrix} A|_{\mathbf{I}_i \times \mathbf{I}_i^c} & A|_{\mathbf{I}_i \times \mathbf{I}_i^r} \end{pmatrix}, \quad A_i^| = \begin{pmatrix} A|_{\mathbf{I}_i^c \times \mathbf{I}_i} \\ A|_{\mathbf{I}_i^r \times \mathbf{I}_i} \end{pmatrix}. \tag{2.3}$$

Clearly, the columns of U_i form a basis for the column space of the *HSS block row* A_i^-, and the rows of V_i^T form a basis for the row space of the *HSS block column* $A_i^|$. We call the maximum (numerical) rank of all HSS blocks the *HSS rank* of A.

2.2. Low-rank property of C. According to [12] and section 4, the numerical rank of an HSS block of $\mathcal{C}$ in (1.7) is at most twice that of the corresponding HSS

block of $\mathbf{C}$ in (1.9). Thus to examine the low-rank property of $\mathcal{C}$, it is sufficient to look at $\mathbf{C}$. The HSS rank of $\mathbf{C}$ can be shown to be at most $O(\log n)$ [12, 33]. In Figure 2.2, we plot the numerical ranks of the HSS block rows of two Cauchy matrices $\mathbf{C}$. Each matrix is hierarchically partitioned into $l_{\max}$ levels of HSS blocks. At level $l = 0, 1, \ldots, l_{\max}$, the HSS block has row size $N_l = n/2^l$ and numerical rank r_l. Clearly, when l decreases, N_l doubles, but r_l only increases slightly. In fact, Figure 2.2 indicates that r_l increases much slower than $O(\log N_l)$ (it is roughly in the pattern of $O(\log\log N_l)$ in this computation). Although not yet analytically justified, this is observed to hold numerically, in practice. This observation is useful for the derivation of the truly linear complexity of our HSS construction and solution as in section 3.6.

3. Fast HSS construction and solution for the Cauchy-like system. In this section, we study the fast construction of an HSS approximation to $\mathcal{C}$ in (1.7). Let r be the HSS rank of $\mathcal{C}$. Then $r = O(\log n)$. Unlike the HSS construction algorithm for a general dense matrix (which costs $O(rn^2)$ flops or more), here we need only $O(n\log^2 n)$ flops for $\mathcal{C}$. The main framework of our method is listed in Table 3.1.

TABLE 3.1
Overview of our superfast Toeplitz solver.

Stage	Step	Section
T to $\mathcal{C}$	Conversion of T to $\mathcal{C}$	1.1
$\mathcal{C}$ to HSS approximate	Computation of $\mathcal{C}X$ and $\mathcal{C}^TX$ with random X	3.1
	Compression via randomized sampling	3.2
	HSS construction for $\mathcal{C}$ with $\mathcal{C}X$ and $\mathcal{C}^TX$	3.3
Factorization/solution	HSS factorization	3.4
	HSS solution for $\mathcal{C}\mathbf{x} = \mathbf{b}$	3.5
	Recovery of the solution to $Tx = b$ ($x = \mathcal{D}_0^{-1}\mathcal{F}_n^*\mathbf{x}$)	1.1

Some major ideas used are as follows:

(1) A structure-preserving rank-revealing factorization via randomized sampling.
(2) Fast Toeplitz matrix-vector multiplication via circulant matrix-vector multiplication and FFT.
(3) HSS construction via hierarchical application of the compression and randomized sampling.
(4) Rank relaxation in the detailed analysis of the complexity.

3.1. Fast matrix-vector multiplication for $\mathcal{C}$. Our HSS construction for $\mathcal{C}$ needs the multiplication of $\mathcal{C}$ with an $n \times (r+\mu)$ Gaussian random matrix X, where r is the HSS rank of $\mathcal{C}$, and μ is a small integer. (The values of $r+\mu$ in some practical examples are shown in section 5.) According to (1.7), we have

$$\mathcal{C}X = \mathcal{F}_n T(\mathcal{D}_0^H \mathcal{F}_n^H X). \tag{3.1}$$

The product $\tilde{X} = \mathcal{D}_0^H \mathcal{F}_n^H X$ can be obtained efficiently and stably via FFT and diagonal scaling. Thus, we need to compute $T\tilde{X}$. This can be done quickly by extending the Toeplitz matrix to a circulant matrix (see, e.g., [16])

$$\mathbf{T} = \begin{pmatrix} T & S \\ S & T \end{pmatrix},$$

where S is a Toeplitz matrix generated by the Toeplitz vector $(t_{1:n-1}, 0, t_{-(n-1):-1})^T$. Let $v = (t_{0:n-1}, 0, t_{-(n-1):-1})^T$. Then $\mathbf{T}$ can be diagonalized by the normalized inverse

DFT matrix $\mathcal{F}_{2n}$ (defined in (1.4)):

$$\mathbf{T} = \mathcal{F}_{2n}^H \mathrm{diag}(\mathcal{F}_{2n} v)\mathcal{F}_{2n}.$$

Thus,

$$\mathbf{T}\begin{pmatrix} \tilde{X} \\ 0 \end{pmatrix} = \begin{pmatrix} T\tilde{X} \\ S\tilde{X} \end{pmatrix} = \mathcal{F}_{2n}^H \mathrm{diag}(\mathcal{F}_{2n} t)\mathcal{F}_{2n}\begin{pmatrix} \tilde{X} \\ 0 \end{pmatrix}. \tag{3.2}$$

The last product can be quickly computed via FFTs and diagonal scaling, and its first half is $T\tilde{X}$. The total cost for computing $\mathcal{C}X$ with (3.1) and (3.2) is about $40(r+\mu)n\log n$ flops for FFTs. Note that such a fast multiplication scheme works since $\mathcal{C}$ is obtained from T, and is generally not applicable to an arbitrary Cauchy-like matrix.

3.2. Compression method: Structure-preserving rank-revealing factorization via randomized sampling. The construction of an HSS form usually involves hierarchical compression of the HSS blocks. Without loss of generality, assume Φ is an $M \times N$ block with numerical rank α. Compute a strong rank-revealing RQ factorization [18]

$$\underset{M\times N}{\Phi} \approx \underset{M\times\alpha}{P}\ \underset{M\times\alpha}{R} \cdot \underset{\alpha\times N}{\Omega}, \tag{3.3}$$

where P is a permutation matrix. Let

$$R = \begin{pmatrix} R_1 \\ R_2 \end{pmatrix} \begin{matrix} \alpha \\ M-\alpha \end{matrix}, \quad E = R_2 R_1^{-1}.$$

To ensure stability, the permutations are done by increasing $\det(R_1)$ so that the entries of $|E|$ are small enough [18]. The increase of the determinant can be quickly detected.

Rewrite (3.3) as

$$\Phi \approx P\begin{pmatrix} R_1 \\ R_2 \end{pmatrix}\Omega = P\begin{pmatrix} I \\ E \end{pmatrix}(R_1\Omega) = P\begin{pmatrix} I \\ E \end{pmatrix}\Phi|_{\hat{\mathbf{I}}}, \tag{3.4}$$

where $\Phi|_{\hat{\mathbf{I}}}$ is a submatrix of Φ with row indices $\hat{\mathbf{I}}$. This holds due to the permutation and identity matrices. To emphasize the fact that $\Phi|_{\hat{\mathbf{I}}}$ preserves part of the structure of Φ, we call (3.4) a *structure-preserving rank-revealing* (SPRR) factorization of Φ. This factorization is closely related to skeleton approximations [43], and is also called interpolative decomposition [29, 31] or subset selection decomposition [25]. A similar decomposition called representative row/column selection is also studied in [19] based on strong rank-revealing LU factorizations.

To enhance the efficiency of the compression, especially when the size of Φ is large, (3.4) can be combined with random sampling [29, 31]. Generate an $N \times (\alpha+\mu)$ Gaussian random matrix X_1, where μ is a small integer. Next, apply the SPRR factorization to the product of Φ and X_1, which is a much smaller matrix:

$$Y_1 = \Phi X_1 \approx P_1\begin{pmatrix} I \\ E_1 \end{pmatrix} Y_1|_{\hat{\mathbf{I}}_1}.$$

Then an approximate factorization of A can be obtained as [29]

$$\Phi \approx U_1 \Phi|_{\hat{\mathbf{I}}_1} \text{ with } U_1 = P_1\begin{pmatrix} I \\ E_1 \end{pmatrix}. \tag{3.5}$$

It is shown in [20, 29, 32] that, under very mild assumptions for μ, the probability for the approximation error to satisfy the following relationship is $1-6\mu^{-\mu}$:

$$\left\|\Phi - U_1\Phi|_{\hat{\mathbf{I}}_1}\right\|_2 \le (1+11\sqrt{\alpha+\mu}\sqrt{\min(M,N)})\sigma_{\alpha+1}, \tag{3.6}$$

where $\sigma_{\alpha+1}$ is the $(\alpha+1)$st singular value of Φ. For example, if $\mu = 20$, then the probability of failure is less than 10^{-17}. If, in addition, $M = N = 1000$, $\alpha = 100$, and $\sigma_{\alpha+1} = 10^{-15}$, then the right-hand side in (3.6) is about 3.8×10^{-12}. In practice, the bound often overestimates the error. In our tests below, we use $\mu = 10$ as in [31] and a relative tolerance 10^{-15} (for $\sigma_1/\sigma_{\alpha+1}$).

This SPRR factorization has more advantages other than structure preservation. It uses only matrix-vector multiplications without requiring the explicit entries of Φ. If $\alpha \ll N$, the compression of Y_1 is significantly faster than that of Φ.

3.3. HSS construction for $\mathcal{C}$. Our HSS construction algorithm has a scheme similar to the one in [31], but with additional structures fully considered and without using any extra reorthonormalization. Also, we use postordered traversal of a given HSS tree $\mathcal{T}$. Assume r is the HSS rank of $\mathcal{C}$, and the HSS block row i have row dimension m_i and starting row index l_i. That is,

$$l_i = \begin{cases} \sum_{j:\ \text{leaf},\ j<i} m_j + 1 & \text{if } i \text{ is a leaf,} \\ l_{c_1} & \text{otherwise, if } c_1 \text{ is the left child of } i. \end{cases} \tag{3.7}$$

First, use the method in section 3.1 to compute

$$Y = \mathcal{C}X, \quad Z = \mathcal{C}^T X. \tag{3.8}$$

(In [31], two Gaussian random matrices are generated. Here, one is sufficient.)

Next, we show the HSS construction process following the traversal of the nodes $i = 1, 2, \ldots$ of $\mathcal{T}$. We compress the HSS blocks hierarchically to get the HSS generators U_i, V_i, R_i, W_i, and B_i.

If i is a leaf, let $X_i \equiv X|_{\mathbf{I}_i}$, $Y_i \equiv Y|_{\mathbf{I}_i}$, and $Z_i \equiv Z|_{\mathbf{I}_i}$. Set $D_i \equiv \mathcal{C}|_{\mathbf{I}_i\times\mathbf{I}_i}$. Then we compress $\mathcal{C}_i^-$ and $\mathcal{C}_i^|$ with SPRR factorizations and randomized sampling. We need to form the products $\Phi_i \equiv \mathcal{C}_i^- X|_{(\mathbf{I}_i^c\cup\mathbf{I}_i^r)}$ and $\Theta_i \equiv (\mathcal{C}_i^|)^T X|_{(\mathbf{I}_i^c\cup\mathbf{I}_i^r)}$ for compression. The ith block rows of $\mathcal{C}X$ and $\mathcal{C}^T X$ are

$$D_i X_i + \mathcal{C}_i^- X|_{(\mathbf{I}_i^c\cup\mathbf{I}_i^r)} = Y_i \ \text{ and } \ D_i^T X_i + (\mathcal{C}_i^|)^T X|_{(\mathbf{I}_i^c\cup\mathbf{I}_i^r)} = Z_i,$$

respectively. This leads to

$$\Phi_i = Y_i - D_i X_i, \quad \Theta_i = Z_i - D_i^T X_i. \tag{3.9}$$

See Figure 3.1(i).

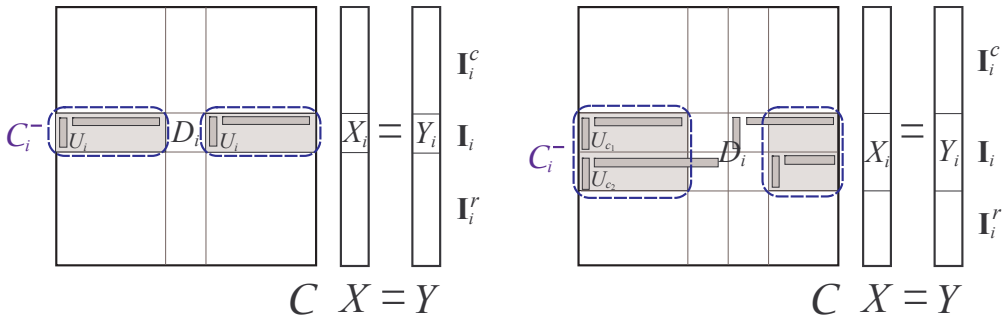
Thus, to compress $\mathcal{C}_i^-$ and $\mathcal{C}_i^|$, we compute SPRR factorizations

$$\Phi_i \approx U_i\Phi_i|_{\hat{\mathbf{I}}_i} \text{ with } U_i = P_i \begin{pmatrix} I \\ E_i \end{pmatrix} \begin{matrix} r \\ m_i - r \end{matrix}, \tag{3.10}$$

$$\Theta_i \approx V_i\Theta_i|_{\hat{\mathbf{J}}_i} \text{ with } V_i = Q_i \begin{pmatrix} I \\ F_i \end{pmatrix} \begin{matrix} r \\ m_i - r \end{matrix}. \tag{3.11}$$

(To simplify the notation, r is used to denote the numerical ranks of all HSS blocks.) Then

$$\mathcal{C}_i^- \approx U_i\mathcal{C}_i^-|_{\hat{\mathbf{I}}_i}, \quad (\mathcal{C}_i^|)^T \approx V_i(\mathcal{C}_i^|)^T|_{\hat{\mathbf{J}}_i}. \tag{3.12}$$



(i) A leaf node i (ii) A nonleaf node i

FIG. 3.1. *How $\Phi_i = \mathcal{C}_i^- X|_{(\mathbf{I}_i^c \cup \mathbf{I}_i^r)}$ is obtained when node i is a leaf node or a nonleaf node, where the index sets satisfy $\mathbf{I}_i^c \cup \mathbf{I}_i \cup \mathbf{I}_i^r = \{1, 2, \ldots, n\}$.*

Also, let

$$\hat{Y}_i = V_i^T X_i = \begin{pmatrix} I & F_i^T \end{pmatrix} Q_i^T X_i, \quad \hat{Z}_i = U_i^T X_i = \begin{pmatrix} I & E_i^T \end{pmatrix} P_i^T X_i, \tag{3.13}$$

where the identity matrices (as additional structures) are used to save the multiplication cost. Since $\mathcal{C}_i^-|_{\hat{\mathbf{I}}_i}$ and $(\mathcal{C}_i^|)^H|_{\hat{\mathbf{J}}_i}$ in (3.12) are blocks in $\mathcal{C}$, we identify their global indices in $\mathcal{C}$ as

$$\tilde{\mathbf{I}}_i = l_i + \hat{\mathbf{I}}_i - 1, \quad \tilde{\mathbf{J}}_i = l_i + \hat{\mathbf{J}}_i - 1. \tag{3.14}$$

If i is a nonleaf node, assume c_1 and c_2 are the left and right children of i, respectively, which have been visited before. Let

$$B_{c_1} = \mathcal{C}|_{\tilde{\mathbf{I}}_{c_1} \times \tilde{\mathbf{J}}_{c_2}}, \quad B_{c_2} = \mathcal{C}|_{\tilde{\mathbf{I}}_{c_2} \times \tilde{\mathbf{J}}_{c_1}}. \tag{3.15}$$

Due to recursion, all generators associated with c_1 and c_2 are available, and

$$Y_{c_1} - D_{c_1} X_{c_1} = U_{c_1} \Phi_{c_1}|_{\mathbf{I}_{c_1}}, \quad Y_{c_2} - D_{c_2} X_{c_2} = U_{c_2} \Phi_{c_2}|_{\mathbf{I}_{c_2}}. \tag{3.16}$$

For example, the first formula in (3.16) is a direct result of the steps associated with c_1, (3.10), and (3.18), together with (2.2). Therefore, we can ignore existing U, V generators in our further compression as follows:

$$\begin{aligned}
&\mathcal{C}_i^- X|_{(\mathbf{I}_i^c \cup \mathbf{I}_i^r)} = Y_i - D_i X_i \\
\approx &\begin{pmatrix} Y_{c_1} \\ Y_{c_2} \end{pmatrix} - \begin{pmatrix} D_{c_1} & U_{c_1} B_{c_1} V_{c_2}^T \\ U_{c_2} B_{c_2} V_{c_1}^T & D_{c_2} \end{pmatrix} \begin{pmatrix} X_{c_1} \\ X_{c_2} \end{pmatrix} \\
= &\begin{pmatrix} (Y_{c_1} - D_{c_1} X_{c_1}) - U_{c_1} B_{c_1} V_{c_2}^T X_{c_2} \\ (Y_{c_2} - D_{c_2} X_{c_2}) - U_{c_2} B_{c_2} V_{c_1}^T X_{c_1} \end{pmatrix} \approx \begin{pmatrix} U_{c_1} \Phi_{c_1}|_{\mathbf{I}_{c_1}} - U_{c_1} B_{c_1} V_{c_2}^T X_{c_2} \\ U_{c_2} \Phi_{c_2}|_{\mathbf{I}_{c_2}} - U_{c_2} B_{c_2} V_{c_1}^T X_{c_1} \end{pmatrix} \\
= &\begin{pmatrix} U_{c_1} & \\ & U_{c_2} \end{pmatrix} \begin{pmatrix} \Phi_{c_1}|_{\mathbf{I}_{c_1}} - B_{c_1} \hat{Y}_{c_2} \\ \Phi_{c_2}|_{\mathbf{I}_{c_2}} - B_{c_2} \hat{Y}_{c_1} \end{pmatrix}.
\end{aligned}$$

See Figure 3.1(ii). Similarly,

$$(\mathcal{C}_i^|)^T X|_{(\mathbf{I}_i^c \cup \mathbf{I}_i^r)} \approx \begin{pmatrix} V_{c_1} & \\ & V_{c_2} \end{pmatrix} \begin{pmatrix} \Theta_{c_1}|_{\mathbf{I}_{c_1}} - B_{c_2}^T \hat{Z}_{c_1} \\ \Theta_{c_2}|_{\mathbf{I}_{c_2}} - B_{c_1}^T \hat{Z}_{c_2} \end{pmatrix}.$$

This means we need only compress

$$\Phi_i \equiv \begin{pmatrix} \Phi_{c_1}|_{\mathbf{I}_{c_1}} - B_{c_1}\hat{Y}_{c_2} \\ \Phi_{c_2}|_{\mathbf{I}_{c_2}} - B_{c_2}\hat{Y}_{c_1} \end{pmatrix}, \quad \Theta_i \equiv \begin{pmatrix} \Theta_{c_1}|_{\mathbf{I}_{c_1}} - B_{c_2}^T\hat{Z}_{c_1} \\ \Theta_{c_2}|_{\mathbf{I}_{c_2}} - B_{c_1}^T\hat{Z}_{c_2} \end{pmatrix}. \tag{3.17}$$

Compute SPRR factorizations

$$\Phi_i \approx \begin{pmatrix} R_{c_1} \\ R_{c_2} \end{pmatrix} \Phi|_{\hat{\mathbf{I}}_i} \text{ with } \begin{pmatrix} R_{c_1} \\ R_{c_2} \end{pmatrix} = P_i \begin{pmatrix} I \\ E_i \end{pmatrix}, \tag{3.18}$$

$$\Theta_i \approx \begin{pmatrix} W_{c_1} \\ W_{c_2} \end{pmatrix} \Theta|_{\hat{\mathbf{J}}_i} \text{ with } \begin{pmatrix} W_{c_1} \\ W_{c_2} \end{pmatrix} = Q_i \begin{pmatrix} I \\ F_i \end{pmatrix}. \tag{3.19}$$

Also, let

$$\begin{aligned} \hat{Y}_i &= \begin{pmatrix} W_{c_1}^T & W_{c_2}^T \end{pmatrix} \begin{pmatrix} \hat{X}_{c_1} \\ \hat{X}_{c_2} \end{pmatrix} = \begin{pmatrix} I & F_i^T \end{pmatrix} Q_i^T \begin{pmatrix} \hat{X}_{c_1} \\ \hat{X}_{c_2} \end{pmatrix}, \\ \hat{Z}_i &= \begin{pmatrix} R_{c_1}^T & R_{c_2}^T \end{pmatrix} \begin{pmatrix} \hat{Z}_{c_1} \\ \hat{Z}_{c_2} \end{pmatrix} = \begin{pmatrix} I & E_i^T \end{pmatrix} P_i^T \begin{pmatrix} \hat{Z}_{c_1} \\ \hat{Z}_{c_2} \end{pmatrix}, \end{aligned} \tag{3.20}$$

where, again, the identity matrices are useful in saving the multiplication cost.

Similarly, use (3.14) to update the indices. Then the process repeats for the nodes in $\mathcal{T}$. The algorithm is summarized as follows.

ALGORITHM 1 (HSS construction for $\mathcal{C}$).

- Compute the products (3.8)
- `for` nodes $i = 1, 2, \ldots$ of $\mathcal{T}$

  `if` i is a leaf
  - Set $D_i \equiv \mathcal{C}|_{\mathbf{I}_i \times \mathbf{I}_i}$
  - Compute Φ_i and Θ_i in (3.9)
  - Compute U_i in (3.10) and V_i as in (3.11)

  `else`
  - Obtain B_{c_1}, B_{c_2} as in (3.15)
  - Compute Φ_i and Θ_i (3.17)
  - Compute R_{c_1}, R_{c_2} in (3.18) and W_{c_1}, W_{c_2} as in (3.19)

  `end`

Remark 1. Unlike the method in [31], here we do not further reorthonormalize the HSS form. That is, we maintain the forms of U_i, V_i in (3.10)–(3.11) and R_i, W_i in (3.18)–(3.19), not only to avoid the extra cost, but also to preserve the internal structures (permutations and identity matrices).

3.4. Fast ULV HSS factorization. For notational convenience, we do not distinguish between $\mathcal{C}$ and its HSS approximation. Here, we directly use the special forms of U_i, V_i, R_i, W_i to compute a ULV-type factorization. (ULV represents a sequence of orthonormal and triangular factors. Here, the factors may be special structured matrices.) Standard ULV HSS factorization [10, 46] has three main steps:

(1) Introduce zeros into HSS block row i by introducing zeros into U_i or R_i.
(2) Partially factorize D_i.
(3) Merge remaining blocks and repeat.

The first step is often expensive. For example, it is usually done via a full QR factorization $U_i = \Omega_i \begin{pmatrix} 0 \\ \tilde{U}_i \end{pmatrix}$. Then Ω_i^H is multiplied to the HSS block row $\mathcal{C}_i^-$ so that

zeros are introduced into $\mathcal{C}_i^-$, or

$$\Omega_i^H U_i = \begin{pmatrix} 0 \\ \tilde{U}_i \end{pmatrix}. \tag{3.21}$$

This also needs a dense matrix multiplication $\tilde{D}_i = \Omega_i^H D_i$. Here, since U_i and R_i have special forms as in (3.10) and (3.18), respectively, this step can be significantly improved. That is, we can avoid the QR step and also reduce the matrix multiplication cost. Our ULV factorization has a framework similar to those in [10, 46] (where pictorial representations are available), but is much faster with internal structured operations. Again, we traverse the tree $\mathcal{T}$ for nodes $i = 1, 2, \ldots$.

If i is a leaf, according to (3.10), the following formula automatically introduces zeros into U_i without any actual cost:

$$\left[\begin{pmatrix} -E_i & I \\ I & 0 \end{pmatrix} P_i^T\right] U_i = \begin{pmatrix} 0 \\ I \end{pmatrix}. \tag{3.22}$$

That is, in (3.21), we simply set

$$\Omega_i^H = \begin{pmatrix} -E_i & I \\ I & 0 \end{pmatrix} P_i^T, \quad \tilde{U}_i = I.$$

Then we also update the diagonal block D_i as

$$\tilde{D}_i = \begin{pmatrix} -E_i & I \\ I & 0 \end{pmatrix} P_i^T D_i.$$

Note that this requires only the multiplication of E_i with the pivot block of $P_i^T D_i$, which reduces the $\Omega_i^H D_i$ dense multiplication cost from $2m_i^3$ to $2r^2(m_i - r)$. Therefore, zeros are introduced into the HSS block row $\mathcal{C}_i^-$, and (3.12) is updated to

$$\begin{pmatrix} -E_i & I \\ I & 0 \end{pmatrix} P_i^T \mathcal{C}_i^- \approx \begin{pmatrix} 0 \\ I \end{pmatrix} \mathcal{C}_i^-|_{\hat{\mathbf{I}}_i}.$$

Then partition $\tilde{D}_i$ as $\tilde{D}_i = \begin{pmatrix} \tilde{D}_{i;1,1} & \tilde{D}_{i;1,2} \\ \tilde{D}_{i;2,1} & \tilde{D}_{i;2,2} \end{pmatrix} \begin{matrix} m_i - r \\ r \end{matrix}$ and compute a QR factorization

$$\begin{pmatrix} \tilde{D}_{i;1,1}^T \\ \tilde{D}_{i;1,2}^T \end{pmatrix} = \tilde{Q}_i \begin{pmatrix} \hat{D}_{i;1,1}^T \\ 0 \end{pmatrix}.$$

Update $\tilde{D}_i$ and the HSS block column $\mathcal{C}_i^|$ by computing

$$\hat{D}_i = \tilde{D}_i \tilde{Q}_i = \begin{pmatrix} L_i & 0 \\ \hat{D}_{i;2,1} & \hat{D}_{i;2,2} \end{pmatrix}, \quad \tilde{V}_i = \tilde{Q}_i^T V_i \equiv \begin{pmatrix} \tilde{V}_{i;1} \\ \tilde{V}_{i;2} \end{pmatrix},$$

where $\tilde{V}_i$ is partitioned conformably. Therefore, $\hat{D}_{i;1,1}$ can be eliminated. Similarly, the blocks associated with the sibling node of i can also be partially eliminated.

If i is a nonleaf node, its children c_1 and c_2 can be removed from $\mathcal{T}$, and i becomes a leaf with new generators

$$D_i = \begin{pmatrix} \hat{D}_{c_1;2,2} & B_{c_1} \tilde{V}_{c_2;2}^T \\ B_{c_2} \tilde{V}_{c_1;2}^T & \hat{D}_{c_2;2,2} \end{pmatrix}, \quad U_i = \begin{pmatrix} R_{c_1} \\ R_{c_2} \end{pmatrix}, \quad V_i = \begin{pmatrix} \tilde{V}_{c_1;2} W_{c_1} \\ \tilde{V}_{c_2;2} W_{c_2} \end{pmatrix}.$$

(Here, D_i, U_i, V_i are used for notational convenience, and they are not the original generators associated with i.) This is the merge process. Note that, due to (3.18)–(3.19), we have

$$U_i = P_i \begin{pmatrix} I \\ E_i \end{pmatrix}, \quad V_i = \begin{pmatrix} \tilde{V}_{c_1;2} & \\ & \tilde{V}_{c_2;2} \end{pmatrix} Q_i \begin{pmatrix} I \\ F_i \end{pmatrix}.$$

The formation of V_i can also take advantage of the identity matrix. Here again, due to the form U_i, we can apply the same procedure to node i as in the step for a leaf. Thus, the fast elimination steps repeat, until we reach the root node where a direct LU factorization is used.

3.5. Fast ULV HSS solution. Similarly, the solution for (1.8) (with the ULV factors of the HSS approximation to $\mathcal{C}$) can also be quickly computed. The framework follows those in [10, 46]. Initially, partition $\mathbf{b}$ into individual pieces $\mathbf{b}_i$ following all the leaf level D_i sizes. The solution consists of two stages.

The first stage is a forward substitution process. For a leaf i, let

$$\tilde{\mathbf{b}}_i = \begin{pmatrix} -E_i & I \\ I & 0 \end{pmatrix} P_i^T \mathbf{b}_i \equiv \begin{pmatrix} \tilde{\mathbf{b}}_{i,1} \\ \tilde{\mathbf{b}}_{i,2} \end{pmatrix} \begin{matrix} m-r \\ r \end{matrix}, \quad \mathbf{y}_i \equiv L_i^{-1} \tilde{\mathbf{b}}_{i,1}. \tag{3.23}$$

The multiplication for $\tilde{\mathbf{b}}_i$ can be quickly conducted. For a nonleaf node i with children c_1 and c_2, let $\mathbf{b}_i = \begin{pmatrix} \tilde{\mathbf{b}}_{c_1,2} \\ \tilde{\mathbf{b}}_{c_2,2} \end{pmatrix}$, and the above process (3.23) repeats for i. When $i = \text{root}\,(\mathcal{T})$ is reached, we compute $\mathbf{y}_i \equiv D_i^{-1} \tilde{\mathbf{b}}_{i,1}$ using the LU factorization of D_i.

The second stage is a backward substitution process. Let $\mathbf{x}_{\text{root}(\mathcal{T})} = \mathbf{y}_{\text{root}(\mathcal{T})}$. For any nonleaf node i and its children c_1 and c_2, partition $\mathbf{x}_i$ as $\mathbf{x}_i = \begin{pmatrix} \mathbf{x}_{i,1} \\ \mathbf{x}_{i,2} \end{pmatrix} \begin{matrix} r \\ r \end{matrix}$. Then compute

$$\mathbf{x}_{c_1} = \tilde{Q}_{c_1} \begin{pmatrix} \mathbf{y}_{c_1} \\ \mathbf{x}_{i,1} \end{pmatrix}, \quad \mathbf{x}_{c_2} = \tilde{Q}_{c_2} \begin{pmatrix} \mathbf{y}_{c_2} \\ \mathbf{x}_{i,2} \end{pmatrix}.$$

This process then continues in a top-down traversal of $\mathcal{T}$. When it finishes, we just need to merge the $\mathbf{x}_i$ pieces associated with all leaves i to form the solution $\mathbf{x}$.

3.6. Complexity, rank relaxation, and accuracy. To estimate the complexity, we list the flop counts of some basic matrix operations in Table 3.2. Here, we need only count the cost of the HSS block row compression, since the block column compression cost is the same. For simplicity, assume all HSS blocks have numerical ranks r and $m_i = m = O(r)$. Also let $\tilde{r} = r + \mu$, which is close to r.

TABLE 3.2

Flop counts of some basic matrix operations, which can be found in, say, [16] *or can be derived based on those. The RRQR factorization in our work is based on the modified Gram–Schmidt process with column pivoting* [16]. *The SPRR factorization uses the RRQR factorization.*

Operation	Flops
Product of an $m \times q$ matrix and an $q \times r$ matrix	$2mqr$
RRQR factorization of an $m \times q$ matrix with rank r	$4mqr - 2r^2(m+q) + \frac{4}{3}r^3$
SPRR factorization of an $m \times q$ matrix with rank r	$4mqr - r^2(m+2q) + \frac{1}{3}r^3$
Full QR factorization of an $m \times q$ tall matrix ($m > q$)	$2q^2\left(m - \frac{q}{3}\right)$
Product of the Q factor and an $m \times r$ matrix	$2rq(2m - q)$
Solution of an order m triangular system $Lx = b$	km^2

The flop counts for the main steps of the HSS construction for $\mathcal{C}$ are listed as follows, with low-order terms dropped:

- Matrix multiplications:
  - Computing the product $\mathcal{C}X$: $40\tilde{r}n\log n$.
  - For each leaf node i,
    - Computing Φ_i in (3.9): $2m^2\tilde{r}$.
    - Computing $\hat{Y}_i$ in (3.13): $2r(m-r)\tilde{r}$.
  - For each nonleaf node i,
    - Computing Φ_i in (3.17): $4r^2\tilde{r}$.
    - Computing $\hat{Y}_i$ in (3.20): $2r^2\tilde{r}$.
- SPRR factorizations:
  - (3.10) for each leaf node i: $4m\tilde{r}r - r^2(m+2\tilde{r}) + \frac{1}{3}r^3$.
  - (3.18) for each nonleaf node i: $8\tilde{r}r^2 - r^2(2r+2\tilde{r}) + \frac{1}{3}r^3$.

The total matrix multiplication cost (for both row and column operations) is thus

$$\xi_0 \equiv \xi_{0,1} + \xi_{0,2}, \tag{3.24}$$

where $\xi_{0,1}$ is for computing $\mathcal{C}X$, and $\xi_{0,2}$ is for updating the products in the construction process, and they are given by

$$\xi_{0,1} = 80\tilde{r}n\log n,$$

$$\xi_{0,2} = 2\left[\sum_{i:\ \text{leaf}} \left(2m^2\tilde{r} + 2r(m-r)\tilde{r}\right) + \sum_{i:\ \text{nonleaf}} \left(4r^2\tilde{r} + 2r^2\tilde{r}\right)\right] = 4\tilde{r}\left(m + r + 2\frac{r^2}{m}\right)n.$$

The total HSS construction cost (for the compression and excluding ξ_0) is

$$\begin{aligned}\xi_1 &= 2\left[\sum_{i:\ \text{leaf}} \left(4m\tilde{r}r - r^2(m+2\tilde{r}) + \frac{1}{3}r^3\right) + \sum_{i:\ \text{nonleaf}} \left(8\tilde{r}r^2 - r^2(2r+2\tilde{r}) + \frac{1}{3}r^3\right)\right] \\ &\approx \frac{2}{3}r\left(12\mu + 9r + 8\frac{r^2}{m} + 12\frac{r\mu}{m}\right)n.\end{aligned} \tag{3.25}$$

Similarly, the ULV factorization and the solution costs are

$$\xi_2 \approx 2\left(\frac{4}{3}m^2 + 4mr - 2r^2 + 16\frac{r^3}{m}\right)n \quad \text{and} \quad \xi_3 \approx 2\left(4m + 4r + 8\frac{r^2}{m}\right)n,$$

respectively.

We then compare the performance of our methods with some similar ones from several aspects. First, if m is chosen to be $2r$ as often used in HSS methods [45], we get the costs as illustrated in Table 3.3. The methods are much more efficient than classical HSS methods.

TABLE 3.3

Flop counts of our new HSS algorithms for $\mathcal{C}$ when $m = 2r$, as compared with classical HSS algorithms, where the terms with μ are dropped since μ is a small constant.

	HSS construction	ULV factorization	ULV solution
Classical HSS	$6rn^2$	$42r^2n$	$37rn$
New HSS	$(80r\log n + \frac{74}{3}r^2)n$	$\frac{58}{3}r^2n$	$16rn$

Next, unlike [31], we do not need an extra step to reorthonormalize the HSS form. Such a step may cost as much as $413r^2n$ flops when $m = 2r$ [45], and it even destroys the special structures in the generators.

Furthermore, even our bounds above can be further improved. Assume, for the HSS block rows corresponding to the nodes at level l of the HSS tree $\mathcal{T}$, the row size is $N_l = N/2^l$ and the maximum numerical rank is r_l. (Here, the root is at level 0 and the leaves are at the largest level $l_{\max}$.) According to the idea of *rank relaxation* in [45], when N_l and r_l increase (as l decreases along the tree levels) following certain patterns, HSS algorithms may achieve similar orders of complexity as when $r_l = O(1)$. Here as observed in section 2, $r_l \le O(\log N_l) = O(l_{\max} - l)$ in practice. Thus, let $m = O(1)$ so that $r_{l_{\max}} = O(1)$. The count (3.25) is essentially improved to

$$\xi_1 = \sum_{i:\ \text{leaf}} O(1) + \sum_{i:\ \text{nonleaf}} O\left((l_{\max}-l)^2\left[\frac{13}{3}(l_{\max}-l)+6\mu\right]\right) = O(n).$$

Therefore, the actual HSS construction cost is truly linear in n, in practice. This also holds for the ULV factorization and solution. Thus, the costs of our methods are

$$\xi_{0,1} = O(n\log^2 n), \quad \xi_{0,2} = O(n), \quad \xi_1 = O(n), \quad \xi_2 = O(n), \quad \xi_3 = O(n). \tag{3.26}$$

Remark 2. The accuracy of the method can be roughly discussed as follows:

(1) The HSS blocks of $\mathcal{C}$ have fast decaying singular values, and $\mathcal{C}$ can be accurately approximated by our HSS construction scheme. The compression of each HSS block introduces an error that satisfies the bound in (3.6) with a high probability. (The bound is related to the tolerance.) Such a bound is then magnified by an appropriate factor in the hierarchical compression. In general, this often overestimates the error, as observed in other HSS methods [10, 46] and randomized HSS methods or similar [30, 31].

(2) The original ULV HSS factorization methods in [10, 46] produce a sequence of intermediate QR factorizations. This converts the HSS matrix into the products of smaller orthogonal and triangular matrices at the hierarchical levels, which can be used to solve HSS systems stably. Here, these QR factorizations are replaced by a stable process (3.22). Also, the pivot growth factor is generally much smaller than the worst case bound in the classical LU factorization [19]. This helps ULV HSS schemes achieve good accuracies. Our numerical tests below indicate that our method works well also for ill-conditioned problems, although a mathematical justification is not yet available. A comprehensive error and stability analysis will be conducted in future work. A closely related study can be found in [3].

4. Precomputations. The previous HSS construction algorithm requires the SPRR factorizations of Φ_i and Θ_i for each node i. We can further improve the efficiency of the HSS construction and randomized sampling by using a precomputation stage. As pointed out in [12], we can first compress the HSS blocks of the Cauchy matrix $\mathbf{C}$ in (1.9). The compression steps in section 3.3 can be significantly improved so that the SPRR factorizations take only $O(\log^3 n)$ flops. Also, the multiplication with X is needed only in this precomputation. This is briefly explained as follows.

Due to the special structure of $\mathbf{C}$, its compression can be performed with high efficiency. That is, at each level of the HSS tree, only one block needs to be compressed. Then a shifting strategy similar to the one in [12] can be used to get the compressed forms for other blocks at the same level. Similarly, the column compression stage can also take advantage of the compression information from the row compression stage.

The compression information from $\mathbf{C}$ can then be used to quickly compute an HSS approximation to $\mathcal{C}$. Without loss of generality, we consider only a leaf node i.

For convenience, we use similar notation as in the previous section, but with different fonts. That is, $\mathcal{C}_i^-$, U_i, etc., are replaced by $\mathbf{C}_i^-$, $\mathbf{U}_i$, etc., respectively. Once we have an approximate factorization $\mathbf{C}_i^- \approx \mathbf{U}_i \mathbf{C}_i^-|_{\hat{\mathbf{I}}_1}$, similar to the method in [12], the corresponding HSS block $\mathcal{C}_i^-$ of $\mathcal{C}$ in (1.7) can be approximated by

$$\begin{aligned}
&\mathcal{C}_i^- \approx \hat{U}_i \hat{\Psi}_i \text{ with} \\
&\hat{U}_i = \begin{pmatrix} \operatorname{diag}\left(\hat{G}|_{(l_i:l_i+m-1),1}\right) \mathbf{U}_i & \operatorname{diag}\left(\hat{G}|_{(l_i:l_i+m-1),2}\right) \mathbf{U}_i \end{pmatrix}, \\
&\hat{\Psi}_i = \begin{pmatrix} \mathbf{C}_i^-|_{\hat{\mathbf{I}}_1} \operatorname{diag}\left(\hat{H}|_{(l_i:l_i+m-1),1}\right) \\ \mathbf{C}_i^-|_{\hat{\mathbf{I}}_1} \operatorname{diag}\left(\hat{H}|_{(l_i:l_i+m-1),2}\right) \end{pmatrix},
\end{aligned}$$

where $\hat{G}$ and $\hat{H}$ are given in (1.6) and $\hat{G}|_{(l_i:l_i+m_i-1),1}$ is the vector given by rows l_i to $l_i + m - 1$ of the first column of $\hat{G}$.

Then to preserve the structure of $\mathcal{C}_i^-$ in the factorization, we further compute an SPRR factorization

$$\hat{U}_i \approx U_i \hat{U}_i|_{\hat{\mathbf{I}}_i} \text{ with } U_i = P_i \begin{pmatrix} I \\ E_i \end{pmatrix}. \tag{4.1}$$

Thus, due to the special structure of U_i (permutation and identity matrix), we have

$$\mathcal{C}_i^- \approx U_i \hat{U}_i|_{\hat{\mathbf{I}}_i} \hat{\Psi}_i = U_i \mathcal{C}_i^-|_{\hat{\mathbf{I}}_i}. \tag{4.2}$$

Therefore, we get a factorization (4.2) that is similar to (3.12) and still preserves the Cauchy-like structure of $\mathcal{C}$ in $\mathcal{C}_i^-|_{\hat{\mathbf{I}}_i}$.

This idea can be modified to handle upper level HSS blocks similarly. After the precomputation stage, the HSS construction for $\mathcal{C}$ costs only $O(n)$ flops.

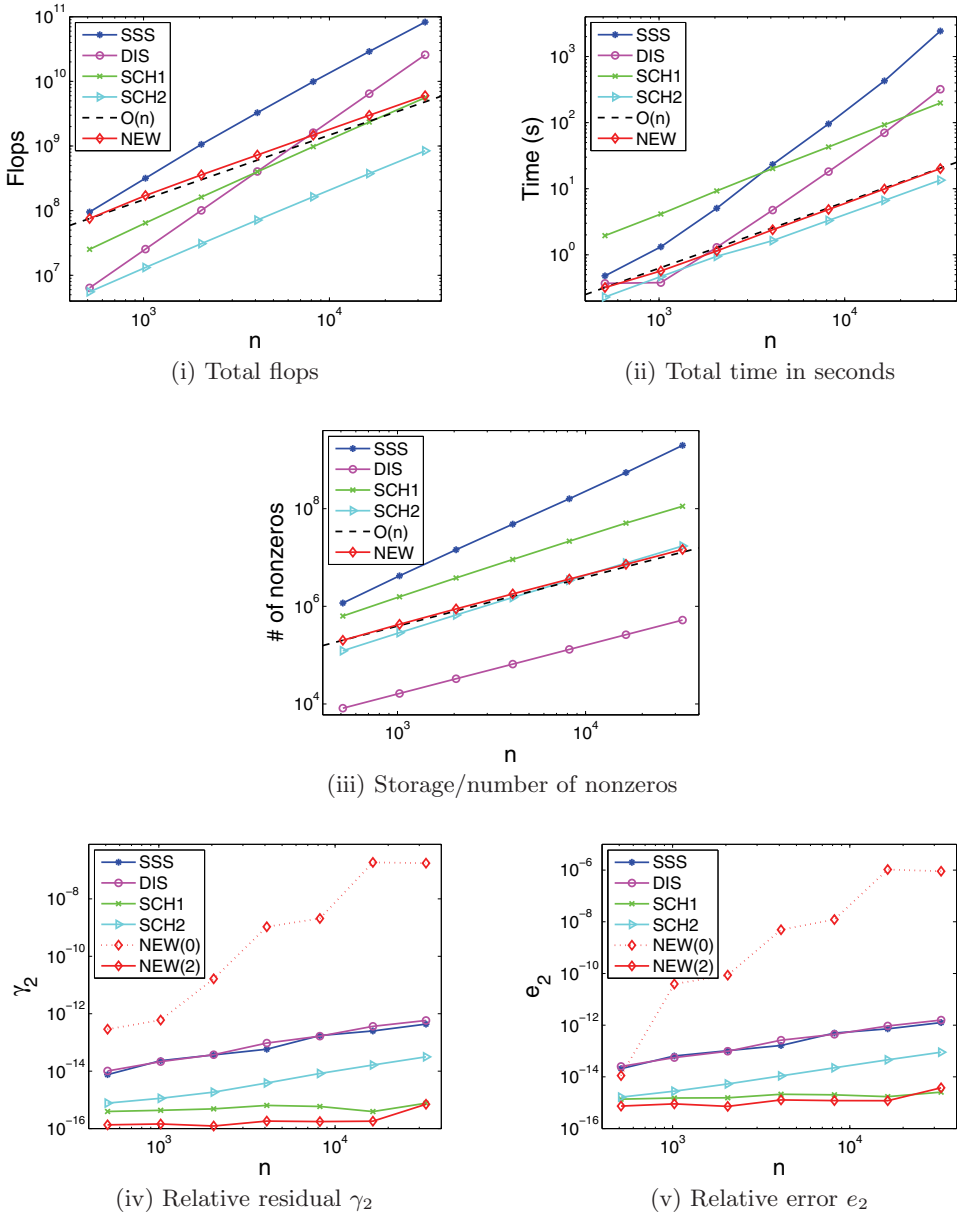
Remark 3. The cost of the SPRR factorization (4.1) is comparable to that of (3.10). However, we save the cost of $\xi_0 = O(n \log^2 n)$ which is needed only in the precomputation. In practice, ξ_0 is insignificant if n is not very large. Hence, the scheme with precomputations is suitable for large n. This section shows the potential of this precomputation stage for multiple T of the same large n, and the details will appear in future work. Our current implementation and numerical experiments in the next section focus on the previous direct HSS construction for $\mathcal{C}$ without precomputations. This direct version gives satisfactory results and is sufficient in general.

5. Numerical experiments. In this section, we test our algorithm (denoted `NEW`) in section 3, which is implemented in MATLAB on a Unix machine. For convenience, we use the following notation in the experiments:

- m is the leaf level HSS block row size.
- $\tilde{r} = r + \mu$ is the column size of X as in section 3.3.
- $\xi_{\text{constr}}(\texttt{NEW}) = \xi_0 + \xi_1$ and $\xi_{\text{sol}}(\texttt{NEW}) = \xi_2 + \xi_3$ as in (3.26) are the flop counts of the HSS construction and solution, respectively, for $\mathcal{C}$ by `NEW`.
- $e_2 = \frac{\|x - \tilde{x}\|_2}{\|x\|_2}$ is the relative error, where x is the exact solution to $Tx = b$ and $\tilde{x}$ is the numerical one.
- $\gamma_2 = \frac{||T\tilde{x} - b||_2}{\|\,|T\tilde{x}| + |b|\,\|_2}$ is the relative residual for the numerical solution.
- σ is the storage size, such as the total number of nonzeros in the factors.

Example 1. Matrix A: The KMS Toeplitz matrix [42] defined by the Toeplitz vector t with

$$t_k = \varphi^{|k|}, \quad 0 < \varphi < 1. \tag{5.1}$$



(i) Total flops

(ii) Total time in seconds

(iii) Storage/number of nonzeros

(iv) Relative residual γ_2

(v) Relative error e_2

FIG. 5.1. *Example 1: Numerical results for the well-conditioned Matrix A with* $\varphi = 0.5$*, where* $m = 80$ *and* $\tilde{r} = 40$ *in* NEW*, and* NEW(0) *(or* NEW(2)*) is for the accuracies of* NEW *without (or with two steps of) iterative refinement.*

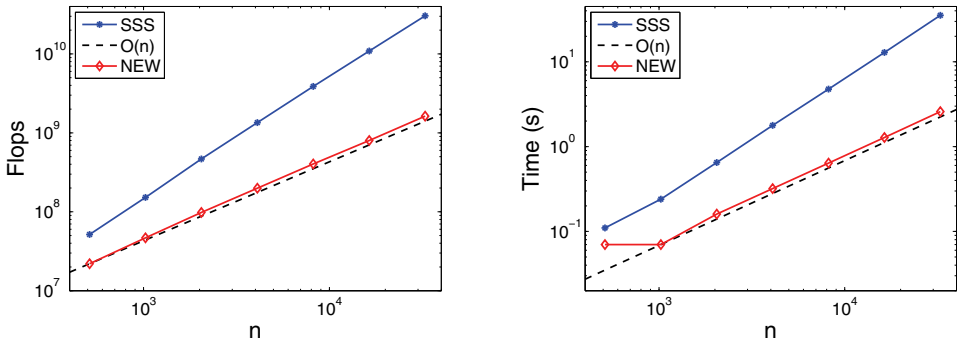
Our method NEW is compared with four other methods (also in MATLAB):

- SSS: The superfast solver in [11] based on SSS structures. SSS uses an $O(n^2)$ complexity precomputation procedure, whose cost is excluded here. (Even if so, NEW is still much faster than SSS in general, as illustrated below.)
- DIS: The fast solver in [2] with displacement structures.
- SCH1: The superfast Schur algorithm in [40] with improved stability.

- SCH2: The superfast Schur algorithm in [1].

For the case when $\varphi = 0.5$ in (5.1), the Toeplitz matrix T is well conditioned. For example, when $n = 10^3$, the 2-norm condition number $\kappa_2 \approx 9$. The flops, timings (in MATLAB), and accuracies of the algorithms are shown in Figure 5.1. The relative tolerance in RRQR factorizations is set to be 10^{-15} in all our tests, although sometimes a larger tolerance may save some costs. We also show the solution costs of NEW and SSS after the factorizations in Figure 5.2, where b is generated with Tx, and x is random. For $\varphi = 1 - 10^{-12}$ in (5.1), T is ill conditioned, and the results are given in Figure 5.3. We make the following observations:

- The total costs, timings, and factorization costs of NEW are roughly linear in n. In Figures 5.1(i)–(iii), 5.2, and 5.3(i)–(iii), we also included a line for $O(n)$. Clearly, this line matches well each performance line for NEW.
- Each flop/timing line for NEW has smaller slopes than those for other methods. In fact, in Figures 5.1(i), NEW starts to be faster than SSS, DIS, and SCH1 around $n = 2^9$, 2^{13}, and 2^{15}, respectively. In Figure 5.3(i), such break-even points are even smaller.
- The storage of NEW is also nearly $O(n)$. When n is around 2^{15}, NEW requires less storage than all the other methods except DIS (which is much slower).
- NEW performs better for the ill-conditioned case, both in the costs and the accuracies. When n gets large, the other methods either require too much memory, are too slow, or break down. For example, in Figure 5.3, SCH1 and SCH2 produce NaN results in MATLAB. For the well-conditioned case, NEW gives relatively lower accuracies. However, high accuracies can be quickly reached with few steps of iterative refinement.
- Note that NEW is also very flexible, and works for general n and (nonsymmetric/indefinite) T. However, SCH1 and SCH2 works only for symmetric positive definite T. Currently, SCH1 also requires n to be a power of 2.



(i) Solution flops (ii) Solution time in seconds

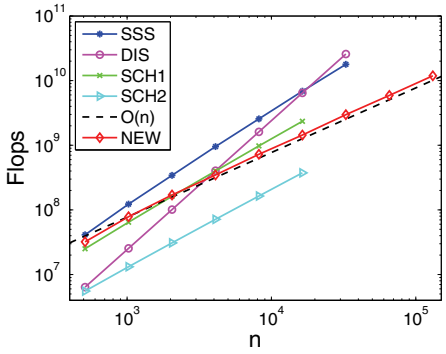
FIG. 5.2. *Example* 1*: Solution flops and timings for* NEW *corresponding to Figure* 5.1.

Example 2. Then we test the algorithm NEW on various matrices as follows:

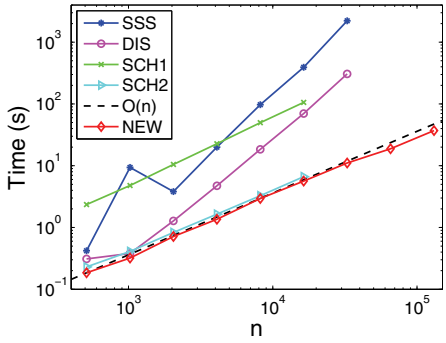
- Matrix B: The Toeplitz vector t that defines the matrix T is

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

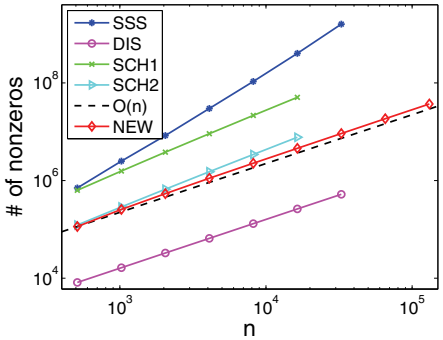
where randn$(n,1)$ generates n normally distributed random numbers. T is mildly ill conditioned.



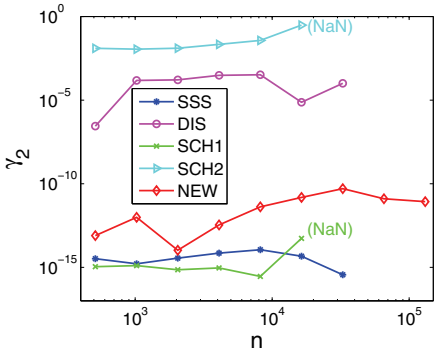
(i) Total flops



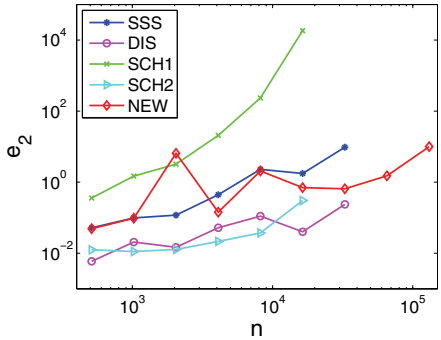
(ii) Total time in seconds



(iii) Storage (number of nonzeros)



(iv) Relative residual γ_2



(v) Relative error e_2

FIG. 5.3. *Example* 1*: Numerical results for the ill-conditioned Matrix A with* $\varphi = 1 - 10^{-12}$*, where* $m = 80$ *and* $\tilde{r} = 40$ *in* NEW.

- Matrix C: The Prolate Toeplitz matrix defined by

$$t_0 = 2\omega, \quad t_k = \frac{\sin(2\omega k\pi)}{k\pi}, \quad k \neq 0, \quad \omega \in [0, 1/2].$$

  This is a classical test matrix which is very ill conditioned [15, 17]. Here, we use $\omega = 1/4$.
- Matrix D: The Toeplitz matrix defined by

$$t_k = \sqrt{\epsilon^2 k^2 + 1}.$$

  The matrix has a 2-norm condition number $\kappa_2 \approx 0.363n^2 e^{3.07/\epsilon}$ [5]. Here, we choose $\epsilon = \frac{1}{8}$, and then $\kappa_2 \approx 1.7n^2 \times 10^{10}$.
- Matrix E: The Toeplitz matrix defined following a Gaussian radial basis function as

$$t_k = e^{-\epsilon^2 k^2}.$$

  The matrix has a condition number $\kappa_2 \approx \frac{1}{2}e^{\pi^2/(4\epsilon^2)}$ and is highly ill conditioned for small ϵ [5]. Here, we choose $\epsilon = \frac{1}{6}$, and then $\kappa_2 \approx 1.9 \times 10^{38}$.

The results are shown in Table 5.1. We see that the construction and solution costs and storage still follow the patterns of nearly linear complexity. `NEW` gives small residuals for all the matrices. In some cases, the solution errors are relatively larger for large n and κ_2. We may similarly use iterative refinement to improve the accuracies.

TABLE 5.1

Example 2: HSS construction cost (ξ_{constr}), ULV solution cost (ξ_{sol}), and relative residual e_2 of `NEW` for the Matrices B–E, where $m = 80$ and $\tilde{r} = 40$.

n		$2^5 \times 10$	$2^6 \times 10$	$2^7 \times 10$	$2^8 \times 10$	$2^9 \times 10$	$2^{10} \times 10$	$2^{11} \times 10$
Matrix B	ξ_{constr}	3.33e7	7.95e7	1.69e8	3.68e8	7.37e8	1.53e9	3.05e9
	ξ_{sol}	1.16e7	2.75e7	5.92e7	1.23e8	2.50e8	5.04e8	1.01e9
	Time	3.40e−1	4.40e−1	9.20e−1	1.96e0	3.91e0	8.44e0	1.61e1
	σ	1.52e5	3.3e5	6.96e5	1.42e6	2.87e6	5.78e6	1.56e7
	γ_2	6.76e−14	1.52e−12	6.35e−12	4.03e−11	8.50e−11	5.92e−10	1.85e−9
	e_2	3.40e−9	4.48e−7	3.15e−6	5.74e−5	6.25e−4	7.16e−2	2.34e−1
Matrix C	ξ_{constr}	3.63e7	8.69e7	1.90e8	3.97e8	8.02e8	1.62e9	3.30e9
	ξ_{sol}	9.96e6	2.58e7	5.95e7	1.23e8	2.50e8	5.05e8	1.01e9
	Time	4.00e−1	5.20e−1	1.19e0	2.47e0	5.04e0	1.05e1	2.05e1
	σ	1.44e5	3.26e5	6.97e5	1.43e6	2.88e6	5.78e6	1.16e7
	γ_2	1.44−15	2.47e−14	4.96e−12	4.78e−12	7.82e−11	8.52e−10	8.00e−10
	e_2	1.66e2	2.85e2	7.68e1	3.32e2	8.52e2	1.21e4	1.14e6
Matrix D	ξ_{constr}	3.42e7	8.08e7	1.80e8	3.73e8	7.87e8	1.55e9	3.16e9
	ξ_{sol}	7.47e6	2.11e7	5.68e7	1.19e8	2.50e8	5.05e8	1.01e9
	Time	3.60e−1	5.00e−1	1.12e0	2.33e0	4.79e0	9.85e0	1.89e1
	σ	1.28e5	2.99e5	6.85e5	1.41e6	2.87e6	5.78e6	1.16e7
	γ_2	4.76−15	6.55e−15	1.72e−13	4.83e−13	6.49e−14	2.66e−13	9.71e−15
	e_2	7.07e−2	2.37e−1	7.05e0	3.17e0	7.47e1	5.93e1	7.88e2
Matrix E	ξ_{constr}	3.61e7	8.60e7	1.86e8	3.91e8	7.99e8	1.62e9	3.29e9
	ξ_{sol}	1.05e7	2.72e7	5.95e7	1.23e8	2.50e8	5.05e8	1.01e9
	Time	3.80e−1	5.40e−1	1.19e0	2.35e0	4.87e0	1.05e1	2.03e1
	σ	1.46e5	3.32e5	6.97e5	1.42e6	2.88e6	5.78e6	1.16e7
	γ_2	4.40−16	9.14e−16	1.85e−13	2.67e−13	5.21e−13	2.85e−12	5.01e−12
	e_2	1.56e1	3.60e1	1.44e1	4.43e1	7.96e1	1.73e2	2.92e2

Example 3. Matrix F: The Toeplitz matrix defined as follows [17]:

$$t_k = \begin{cases} \frac{9}{10} + \frac{1}{10}\,\mathrm{rand}(1) & \text{if } k = 0, \\ -t_0 & \text{if } k > 0, \\ 0 & \text{if } -\frac{n}{2} < k < 0, \\ \mathrm{rand}(1) & \text{otherwise,} \end{cases}$$

where rand(1) generates a random number from uniform distribution in $(0, 1)$.

`NEW` is compared with two methods:

`Levinson`: The Levinson algorithm in [8].

`GEPP`: Gaussian elimination with partial pivoting.

For this matrix, both the straightforward GEPP and the Levinson algorithm fail since they produce huge element growth [17]. See Table 5.2. Here, `NEW` still provides reasonably good accuracies. In comparison, `Levinson` works only for small n, and quickly fails to produce meaningful results. `GEPP` fails for all the matrix sizes in the table. Thus instead, for `GEPP`, we show the growth factors which are very large.

TABLE 5.2

Example 3*: Numerical results for Matrix F, where* $m = 80$, $\tilde{r} = 60$ *in* `NEW`, ρ_{grow} *is the pivot growth factor in* `GEPP`*, and NaN and Inf are outputs in MATLAB to mean certain failures.*

n		$2^5 \times 10$	$2^6 \times 10$	$2^7 \times 10$	$2^8 \times 10$	$2^9 \times 10$	$2^{10} \times 10$	$2^{11} \times 10$
`NEW`	ξ_{constr}	1.03e8	2.38e8	5.32e8	1.05e9	2.18e9	4.45e9	8.84e9
	ξ_{sol}	1.91e7	5.76e7	1.38e8	2.95e8	6.08e8	1.23e9	2.48e9
	γ_2	3.55e−17	9.85e−15	1.66e−12	5.99e−12	4.02e−11	4.30e−10	5.66e−9
	e_2	2.80e−15	3.89e−12	3.18e−10	1.43e−8	2.98e−7	1.05e−5	3.35e−4
`Levinson`	γ_2	7.02e−2	NaN	NaN	NaN			
	e_2	8.28e31	NaN	NaN	NaN			
`GEPP`	ρ_{grow}	1.34e48	1.78e96	4.16e192	Inf			

6. Conclusions and future work. In this paper, we study superfast structured Toeplitz solutions based on an SPRR factorization and randomized sampling. The Cauchy-like matrix $\mathcal{C}$ can be approximated quickly by an HSS matrix in $O(n)$ flops for block compression and $O(n \log^2 n)$ flops for FFTs. One solver we propose is significantly faster than an existing one based on SSS methods and some other methods for large n. Strong rank-revealing factorizations are used for stability. We also briefly show an HSS construction procedure with precomputations, where the compression costs only $O(\log^3 n)$ after FFTs. This construction procedure is especially useful if n is very large and if there are multiple Toeplitz matrices of the same size.

The discussions here provide the possibility of developing new fast versions. For example, it is possible to compute the products such as $\mathbf{C}X$ and Φ_i in (3.9) just once in the precomputation. After this, the HSS construction for $\mathcal{C}$ involves just $O(n)$ cost block compression. This will appear in our future work. The current version without precomputation gives superior results.

This work also shows a practical way of using randomized sampling in the development of new efficient structured methods.

Acknowledgments. The authors are very grateful to the two anonymous referees for their valuable suggestions. The authors would also like to thank J. P. Boyd for some test examples and M. Stewart for his code in [40]. Some other test codes used are from G. Rodriguez's webpage (http://bugs.unica.it/~gppe/soft/).

REFERENCES

[1] G. S. Ammar and W. B. Gragg, *Superfast solution of real positive definite Toeplitz systems*, SIAM J. Matrix Anal. Appl., 9 (1988), pp. 61–76.

[2] A. Aricò, and G. Rodriguez, *A fast solver for linear systems with displacement structure*, Numer. Algorithms, 55 (2010), pp. 529–556.

[3] T. Bella, V. Olshevsky, and M. Stewart, *A nested product decomposition of a quasiseparable matrix*, submitted.

[4] A. W. Bojańczyk, R. P. Brent, F. R. de Hoog, and D. R. Sweet, *On the stability of the Bareiss and related Toeplitz factorization algorithms*, SIAM J. Matrix Anal. Appl., 16 (1995), pp. 40–57.

[5] J. P. Boyd and K. W. Gildersleeve, *Numerical experiments on the condition number of the interpolation matrices for radial basis functions*, Appl. Numer. Math., 61 (2011), pp. 443–459.

[6] J. R. Bunch, *Stability of methods for solving Toeplitz systems of equations*, SIAM J. Sci. Statist. Comput., 6 (1985), pp. 349–364.

[7] R. H. Chan, J. G. Nagy, and R. J. Plemmons, *Displacement preconditioners for Toeplitz least squares iterations*, Electron. Trans. Numer. Anal., 2 (1994), pp. 1–13.

[8] T. F. Chan and P. C. Hansen, *A look-ahead Levinson algorithm for general Toeplitz systems*, IEEE Trans. Signal Process., 40 (1992), pp. 1079–1090.

[9] R. H. Chan and M. K. Ng, *Conjugate gradient methods for Toeplitz systems*, SIAM Rev., 38 (1996) pp. 427–482.

[10] 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.

[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, 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.

[13] S. Chandrasekaran and A. H. Sayed, *Stabilizing the fast generalized Schur algorithm*, SIAM J. Matrix Anal. Appl., 17 (1996), pp. 950–983.

[14] G. Cybenko, *The numerical stability of the Levinson–Durbin algorithm for Toeplitz systems of equations*, SIAM J. Sci. Statist. Comput., 1 (1980), pp. 303–319.

[15] 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.

[16] G. H. Golub and C. V. Loan, *Matrix Computations*, 3rd ed., The Johns Hopkins University Press, Baltimore, MD, 1996.

[17] M. Gu, *Stable and efficient algorithms for structured systems of linear equations*, SIAM J. Matrix Anal. Appl., 19 (1998), pp. 279–306.

[18] 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.

[19] M. Gu and J. Xia, *A numerically stable and superfast algorithm for solving Toeplitz systems of linear equations*, Presentation in the SIAM Conference on Applied Linear Algebra, Monterey, CA, 2009.

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

[21] G. Heinig, *Inversion of generalized Cauchy matrices and other classes of structured matrices*, in Linear Algebra for Signal Processing, IMA Vol. Math. Appl. 69, Springer, New York, 1995, pp. 95–114.

[22] G. Heinig, *Solving Toeplitz systems after extension and transformation*, Calcolo, 33 (1996), pp. 115–129.

[23] G. Heinig and F. Hellinger, *Displacement structure of pseudoinverses*, Linear Algebra Appl., 197/198 (1994), pp. 623–649.

[24] G. Heinig and K. Rost, *Algebraic methods for Toeplitz-like matrices and operators*, Oper. Theory Adv. Appl. 13, Birkhäuser Verlag, Basel, 1984, pp. 109–127.

[25] I. C. F. Ipsen, C. T. Kelley, and S. R. Pope, *Rank-deficient nonlinear least squares problems and subset selection*, SIAM J. Numer. Anal., 49 (2011), pp. 1244–1266.

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

[27] T. Kailath and A. H. Sayed, *Fast algorithms for generalized displacement structures*, in Recent Advances in Mathematical Theory of Systems, Control, Networks, and Signal Pro-

cessing, Vol. II, H. Kimura and S. Kodama, eds., Mita Press, Tokyo, 1992, pp. 27–32.
[28] T. Kailath and A. H. Sayed, eds., *Fast Reliable Algorithms for Matrices with Structure*, SIAM, Philadelphia, 1999.
[29] E. Liberty, F. Woolfe, P.-G. Martinsson, V. Rokhlin, and M. Tygert, *Randomized algorithms for the low-rank approximation of matrices*, Proc. Natl. Acad. Sci. USA, 104 (2007), pp. 20167–20172.
[30] L. Lin, J. Lu, and L. Ying, *Fast construction of hierarchical matrix representation from matrix-vector multiplication*, J. Comput. Phys., 230 (2011), pp. 4071–4087.
[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, V. Rokhlin, and M. Tygert, *A Randomized Algorithm for the Approximation of Matrices*, Technical Report 1361, Department of Computer Science, Yale University, New Haven, CT, 2006.
[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] M. K. Ng and R. H. Chan, *Scientific applications of iterative Toeplitz solvers*, Calcolo, 33 (1996), pp. 249–267.
[35] J. Olkin, *Linear and Nonlinear Deconvolution Problems*, Ph.D. Thesis, Rice University, Houston, TX, 1986.
[36] V. Olshevski, ed., *Fast Algorithms for Structured Matrices: Theory and Applications*, Contemp. Math. 323, Amer. Math. Soc., Providence, RI, 2003.
[37] V. Y. Pan and G. Qian, *Randomized preprocessing of homogeneous linear systems of equations*, Linear Algebra Appl., 432 (2010), pp. 3272–3318.
[38] V. Y. Pan, *On computations with dense structured matrices*, Math. Comp., 55 (1990), pp. 179–190.
[39] G. Strang, *A proposal for Toeplitz matrix calculations*, Stud. Appl. Math., 74 (1986), pp. 171–176.
[40] M. Stewart, *A superfast Toeplitz solver with improved numerical stability*, SIAM J. Matrix Anal. Appl., 25 (2003), pp. 669–693.
[41] D. R. Sweet, *The use of pivoting to improve the numerical performance of algorithms for Toeplitz matrices*, SIAM J. Matrix Anal. Appl., 14 (1993), pp. 468–493.
[42] W. F. Trench, *Numerical solution of the eigenvalue problem for Hermitian Toeplitz matrices*, SIAM J. Matrix Anal. Appl., 10 (1989), pp. 135–146.
[43] E. E. Tyrtyshnikov, *Incomplete cross approximation in the mosaic-skeleton method*, Computing, 64 (2000), pp. 367–380.
[44] M. Van Barel, G. Heinig, and P. Kravanja, *A stabilized superfast solver for nonsymmetric Toeplitz systems*, SIAM J. Matrix Anal. Appl., 23 (2001), pp. 494–510.
[45] J. Xia, *On the complexity of some hierarchical structured matrix algorithms*, SIAM J. Matrix Anal. Appl., 33 (2012), pp. 388–410.
[46] 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.