SIAM J. MATRIX ANAL. APPL.
Vol. 32, No. 4, pp. 1251–1274

A FAST RANDOMIZED ALGORITHM FOR COMPUTING A HIERARCHICALLY SEMISEPARABLE REPRESENTATION OF A MATRIX[*]

P. G. MARTINSSON[†]

Abstract. Randomized sampling has recently been proven a highly efficient technique for computing approximate factorizations of matrices that have low numerical rank. This paper describes an extension of such techniques to a wider class of matrices that are not themselves rank-deficient but have off-diagonal blocks that are; specifically, the class of so-called *hierarchically semiseparable* (HSS) matrices. HSS matrices arise frequently in numerical analysis and signal processing, particularly in the construction of fast methods for solving differential and integral equations numerically. The HSS structure admits algebraic operations (matrix-vector multiplications, matrix factorizations, matrix inversion, etc.) to be performed very rapidly, but only once the HSS representation of the matrix has been constructed. How to rapidly compute this representation in the first place is much less well understood. The present paper demonstrates that if an $N \times N$ matrix can be applied to a vector in $O(N)$ time, and if individual entries of the matrix can be computed rapidly, then provided that an HSS representation of the matrix exists, it can be constructed in $O(N\,k^2)$ operations, where k is an upper bound for the numerical rank of the off-diagonal blocks. The point is that when legacy codes (based on, e.g., the fast multipole method) can be used for the fast matrix-vector multiply, the proposed algorithm can be used to obtain the HSS representation of the matrix, and then well-established techniques for HSS matrices can be used to invert or factor the matrix.

Key words. data-sparse matrix, randomized algorithm, matrix factorization, hierarchically semiseparable matrix, direct solver

AMS subject classifications. 65F05, 65F30, 65F20, 15A23

DOI. 10.1137/100786617

1. Introduction.

A ubiquitous task in computational science is to rapidly perform linear algebraic operations involving very large matrices. Such operations typically exploit special "structure" in the matrix since the costs of standard techniques tend to scale prohibitively fast with matrix size; for a general $N \times N$ matrix, it costs $O(N^2)$ operations to perform a matrix-vector multiplication, $O(N^3)$ operations to perform Gaussian elimination or to invert the matrix, etc. A well-known form of "structure" in a matrix is sparsity. When at most a few entries in each row of the matrix are nonzero (as is the case, e.g., for matrices arising upon the discretization of differential equations, or representing the link structure of the World Wide Web) matrix-vector multiplications can be performed in $O(N)$ operations instead of $O(N^2)$. The description "data-sparse" applies to a matrix that may be dense but that shares the key characteristic of a sparse matrix that some linear algebraic operations, typically the matrix-vector multiplication, can to high precision be executed in fewer than $O(N^2)$ operations (often in close to linear time).

There are many different types of data-sparse representations of a matrix. This paper is concerned with the class of so-called *hierarchically semiseparable* (HSS) matrices [9, 10, 35], which arise upon the discretization of many of the integral operators of mathematical physics, in signal processing, in algorithms for inverting certain fi-

[*]Received by the editors February 22, 2010; accepted for publication (in revised form) by I. C. F. Ipsen July 21, 2011; published electronically November 8, 2011. This work was supported by the NSF under the contracts 0941476, 0748488, and 0610097.
http://www.siam.org/journals/simax/32-4/78661.html

[†]Department of Applied Mathematics, University of Colorado at Boulder, Boulder, CO 80309 (martinss@colorado.edu).

nite element matrices, and in many other applications; see, e.g., [38, 29, 31, 35]. An HSS matrix is a dense matrix whose off-diagonal blocks are rank-deficient in a certain sense. Postponing a precise definition until section 3, we for now simply note that an HSS matrix A can be expressed via a recursive formula in L levels,

$$A^{(j)} = U^{(j)}\,A^{(j-1)}\,V^{(j)} + B^{(j)}, \qquad j = 2, 3, \dots, L, \tag{1.1}$$

where $A = A^{(L)}$ and the sequence $A^{(L)}, A^{(L-1)}, \dots, A^{(1)}$ consists of matrices that are successively smaller. (In principle, one could say that A satisfies the HSS property as long as there is *any* decay in size, but in typical applications, $A^{(j-1)}$ would be roughly half the size of $A^{(j)}$.) In (1.1), the matrices $U^{(j)}$, $V^{(j)}$, and $B^{(j)}$ are all block-diagonal, so the formula directly leads to a fast technique for evaluating a matrix-vector product. The HSS property is similar to many other data-sparse representations in that it exploits rank-deficiencies in off-diagonal blocks to allow matrix-vector products to be evaluated rapidly; the fast multipole method (FMM) [16, 17], the Barnes–Hut method [1], and panel clustering [19] are all similar in this regard. The HSS property is different from these other formats in that it also allows the rapid computation of a matrix inverse, of an LU factorization, etc.; see [8, 9, 12, 30, 36]. The ability to perform algebraic operations other than the matrix-vector multiplication is also characteristic of the $\mathcal{H}$-matrix format of Hackbusch [21]. Comparing the two formats, the $\mathcal{H}$-matrix representation is more general (every HSS matrix is an $\mathcal{H}$-matrix, but not every $\mathcal{H}$-matrix is an HSS matrix) but pays for the generality by requiring more complicated, and typically slower, algorithms for computing matrix inverses, matrix factorizations, etc.

The most straightforward technique for computing the HSS representation of a dense $N \times N$ matrix A is to explicitly form all matrix elements, and then to compress the off-diagonal blocks using, e.g., the SVD. This approach can be executed stably [9, 20], but it is often prohibitively expensive, with an $O(k\,N^2)$ asymptotic cost, where k is the rank of the off-diagonal blocks (in the HSS sense). Fortunately, there exist for specific applications much faster methods for constructing HSS representations. When the matrix A approximates a boundary integral operator in the plane, the technique of [30] computes a representation in $O(k^2\,N)$ time by exploiting representation results from potential theory. In other environments, it is possible to use known regularity properties of the off-diagonal blocks in conjunction with interpolation techniques to obtain rough initial factorizations, and then recompress these to obtain factorizations with close to optimal ranks [5, 31]. A particularly popular version of the "regularity + recompression" method is the so-called *adaptive cross approximation* technique, which was initially proposed for $\mathcal{H}$-matrices [2, 6, 24] but has recently been modified to obtain a representation of a matrix in a format similar to the HSS [14].

The purpose of the present paper is to describe a fast and simple randomized technique for computing an HSS representation of a matrix which can rapidly be applied to a vector. The existence of such a technique means that the advantages of the HSS format—very fast inversion and factorization algorithms in particular—become available for any matrix that can currently be applied via the FMM, via an $\mathcal{H}$-matrix calculation, or by any other existing data-sparse format (provided, of course, that the matrix is in principle compressible in the HSS sense). In order to describe the cost of the algorithm precisely, we introduce some notation: We let A be an $N \times N$ matrix whose off-diagonal blocks have maximal rank k (in the HSS sense; see section 3), we let T_{mult} denote the time required to perform a matrix-vector multiplication $x \mapsto A\,x$ or $x \mapsto A^*\,x$, we let T_{rand} denote the cost of constructing a pseudorandom number from a normalized Gaussian distribution, we let T_{entry} denote the computational cost of evaluating an individual entry of A, and we let T_{flop} denote the cost of a floating point operation. The computational cost T_{total} of the algorithm then satisfies

$$T_{\mathrm{total}} \sim T_{\mathrm{mult}} \times 2\,(k+p) + T_{\mathrm{rand}} \times N\,(k+p) + T_{\mathrm{entry}} \times 2\,N\,k + T_{\mathrm{flop}} \times c\,N\,k^2, \tag{1.2}$$

where c is a small constant and where p is a tuning parameter that balances computational cost against the probability of not meeting the requested accuracy. Setting $p = 10$ is often a good choice which leads to a “failure probability” of less that 10^{-9}; see Remark 1.1. In particular, if T_{mult} is $O(N)$, then the method presented here is $O(N)$ as well.

Remark 1.1. The technique described in this paper utilizes a method for computing approximate low-rank factorizations of matrices that is based on randomized sampling [23, 26, 32]. As a consequence, there is in principle a nonzero risk that the method may fail to produce full accuracy in any given realization of the algorithm. This risk can be controlled by the user via the choice of the tuning parameter p in (1.2); for details see section 2.3. Moreover, unlike some better known randomized algorithms such as Monte Carlo, the accuracy of the output of the algorithms under discussion here is typically very high; in the environment described in the present paper, approximation errors of less than 10^{-10} are entirely typical.

Remark 1.2. There currently is little consistency in terminology in describing different formats for representing “data-sparse” matrices. The property that we here refer to as the “HSS” property is referred to by a range of different names; see, e.g., [31, 30, 33, 36]. It is closely related to the “$\mathcal{H}^2$-matrix” format [5, 3, 4, 22], which is more restrictive than the $\mathcal{H}$-matrix format and often admits $O(N)$ algorithms. The methods described in the present paper are directly applicable to the structures described in [31, 30, 33] and, we believe, with minor modifications to the structures in [3, 5, 4, 22].

Remark 1.3. Since the first version of the present paper [28] appeared, an alternative similar algorithm has been suggested [27]. The approach of [27] is more general than the one presented here in that it accesses the matrix to be compressed *only* via matrix-vector multiplications (no access to matrix entries is required). However, the price to be paid is that the number of matrix-vector multiplications required is very large; in the numerical examples reported in [27], several thousands of applications of the matrix are needed. In contrast, we report in section 5 successful compression via as few as 50 or 100 matrix-vector multiplications. Moreover, in our approach, these matrix-vector multiplications can all be executed in parallel (as opposed to consecutively).

2. Preliminaries.

This section introduces established material that will be needed to derive the new results in section 4. Specifically, we introduce our notation (section 2.1), describe a set of standard matrix factorizations (section 2.2), and describe recently developed randomized techniques for computing a low-rank approximation to a matrix (section 2.3).

2.1. Notation.

Throughout the paper, we measure vectors in $\mathbb{R}^n$ using their Euclidean norm, and matrices using the corresponding operator norm.

For an $m \times n$ matrix B and an integer $k = 1, 2, \ldots, \min(m, n)$, we let $\sigma_k(B)$, or simply σ_k when it is obvious which matrix is being referred to, denote the kth singular value of B. We assume that these are ordered so that $\sigma_1(B) \geq \sigma_2(B) \geq \cdots \geq \sigma_{\min(m,n)}(B) \geq 0$. We say that a matrix B has “ε-rank” k if $\sigma_{k+1}(B) \leq \varepsilon$.

We use the notation of Golub and Van Loan [15] to specify submatrices. In other words, if B is an $m \times n$ matrix with entries b_{ij}, and $I = [i_1, i_2, \dots, i_k]$ and $J = [j_1, j_2, \dots, j_\ell]$ are two index vectors, then we let $B(I, J)$ denote the $k \times \ell$ matrix

$$B(I,J) = \begin{bmatrix} b_{i_1 j_1} & b_{i_1 j_2} & \cdots & b_{i_1 j_\ell} \\ b_{i_2 j_1} & b_{i_2 j_2} & \cdots & b_{i_2 j_\ell} \\ \vdots & \vdots & & \vdots \\ b_{i_k j_1} & b_{i_k j_2} & \cdots & b_{i_k j_\ell} \end{bmatrix}.$$

We let $B(I, :)$ denote the matrix $B(I, [1, 2, \dots, n])$ and define $B(:, J)$ analogously.

Given a set of matrices $\{X_j\}_{j=1}^{\ell}$, we define a block diagonal matrix via

$$\text{diag}(X_1, X_2, \dots, X_\ell) = \begin{bmatrix} X_1 & 0 & 0 & \cdots & 0 \\ 0 & X_2 & 0 & \cdots & 0 \\ 0 & 0 & X_3 & \cdots & 0 \\ \vdots & \vdots & \vdots & & \vdots \\ 0 & 0 & 0 & \cdots & X_\ell \end{bmatrix}.$$

The transpose of B is denoted B^*, and we say that a matrix U is *orthonormal* if its columns form an orthonormal set, so that $U^* U = I$.

2.2. Low-rank factorizations. We say that an $m \times n$ matrix B has exact rank at most k if there exist an $m \times k$ matrix E and a $k \times n$ matrix F such that

$$B = E\,F.$$

In this paper, we will utilize three standard matrix factorizations. (In describing them, we let B denote an $m \times n$ matrix of rank k.) The first is the so-called *QR factorization*

$$B = Q\,R,$$

where Q is an $m \times k$ orthonormal matrix and R is a $k \times n$ matrix with the property that a permutation of its columns is upper triangular. The second is the *singular value decomposition* (SVD)

$$B = U\,D\,V^*,$$

where U and V are orthonormal matrices of sizes $m \times k$ and $n \times k$, and the $k \times k$ matrix D is a diagonal matrix whose diagonal entries are the singular values $\{\sigma_1, \sigma_2, \dots, \sigma_k\}$. The third factorization is the so-called *interpolative decomposition*

$$B = B(:, J)\,X,$$

where J is a vector of indices marking k of the columns of B, and the $k \times n$ matrix X has the $k \times k$ identity matrix as a submatrix and has the property that all its entries are bounded by 1 in magnitude. In other words, the interpolative decomposition picks k columns of B as a basis for the column space of B and expresses the remaining columns in terms of the chosen ones.

The existence for all matrices of the QR factorization and the SVD is well known, as are techniques for computing them accurately and stably; see, e.g., [15]. The interpolatory decomposition is slightly less well known but it too always exists. It can be viewed as a modification to the so-called *rank-revealing QR factorization* [7]. It can be computed in a stable and accurate manner using the techniques of [18], as described in [11]. (Practical algorithms for computing the interpolative decomposition produce a matrix X whose elements slightly exceed 1 in magnitude.) In the pseudocode we use to describe the methods of this paper, we refer to such algorithms as follows:

$$[Q, R] = \texttt{qr}(B), \qquad [U, D, V] = \texttt{svd}(B), \qquad [X, J] = \texttt{interpolate}(B).$$

In the applications under consideration in this paper, matrices that arise are typically only approximately of low rank. Moreover, their approximate ranks are generally not known a priori. As a consequence, the algorithms will typically invoke versions of the factorization algorithms that take the computational accuracy ε as an input parameter. For instance,

$$[U, D, V] = \texttt{svd}(B,\, \varepsilon) \tag{2.1}$$

results in matrices U, D, and V of sizes $m \times k$, $n \times k$, and $k \times k$ such that

$$||U\, D\, V^{*} - B|| \leq \varepsilon.$$

In this case, the number k is of course an output of the algorithm. The corresponding functions for computing an approximate QR factorization or an interpolative decomposition are denoted

$$[Q, R] = \texttt{qr}(B,\, \varepsilon), \qquad [X, J] = \texttt{interpolate}(B,\, \varepsilon). \tag{2.2}$$

In our applications, it is not necessary for the factorizations to be of absolutely minimal rank (i.e., the computed rank k is allowed to slightly exceed the theoretical ε-rank of B).

2.3. Construction of low-rank approximations via randomized sampling. Let B be a given $m \times n$ matrix that can accurately be approximated by a matrix of rank k, and suppose that we seek to determine a matrix Q with orthonormal columns (as few as possible) such that

$$||B - Q\, Q^{*}\, B||$$

is small. In other words, we seek a matrix Q whose columns form an approximate orthornomal basis (ON-basis) for the column space of B. (For now, we assume that the rank k is known in advance; techniques for relaxing this assumption will be described in Remark 2.3.) When we have access to a fast technique for computing matrix-vector products $x \mapsto B\, x$, this task can efficiently be solved via the following randomized procedure:

1. Pick a small integer p representing how much "oversampling" we do. (The choice $p = 10$ is often good.)
2. Form an $n \times (k + p)$ matrix Ω whose entries are drawn independently from a normalized Gaussian distribution.
3. Form the product $S = B\, \Omega$.
4. Construct a matrix Q whose columns form an ON-basis for the columns of S.

Note that each column of the "sample" matrix S is a random linear combination of the columns of B. We would therefore expect the algorithm described to have a high probability of producing an accurate result when p is a large number. It is perhaps less obvious that there exists a lower bound for this probability that depends only on p (not on m or n, or any other properties of B), and that it approaches 1 extremely rapidly as p increases. In fact, one can show that the basis Q determined by the scheme above satisfies

$$||B - Q\,Q^*\,B|| \leq \left[1 + 11\sqrt{k+p}\cdot\sqrt{\min\{m,n\}}\right]\sigma_{k+1}, \tag{2.3}$$

with probability at least $1 - 6\cdot p^{-p}$; see [23, section 1.5].

Remark 2.1. The error bound (2.3) indicates that the error produced by the randomized sampling procedure can be larger than the theoretically minimal error σ_{k+1} by a factor of $1 + 11\sqrt{k+p}\cdot\sqrt{\min\{m,n\}}$. This crude bound is typically very pessimistic; for specific situations sharper bounds have been proven; see [23]. However, a loss of accuracy of one or two digits is often observed, and it is therefore recommended that the matrix-vector multiplication be evaluated as accurately as possible. We demonstrate in section 5 that HSS matrices of practical interest can be approximated to ten digits of accuracy.

Remark 2.2. The task of computing a "thin" orthonormal matrix Q such that $||B - Q\,Q^*\,B||$ is small in an environment where B can rapidly be applied to a vector is well studied. Krylov methods are often recommended, and these can significantly outperform the randomized methods described in this section. In particular, for a fixed number of matrix-vector products, the output of a Krylov method would typically produce a smaller residual error $||B - Q\,Q^*\,B||$ than the randomized methods. A Krylov method achieves high accuracy by picking the next vector to which B is to be applied using information provided by previous matrix-vector products. A randomized method is less accurate but provides much more flexibility, in that the vectors to which B is applied can be picked in advance. This has at least two advantages:

1. The matrix-vector products can be computed in any order or all at once. Significant speed-ups can result, since applying B to k vectors one after another is often much slower than applying B to a matrix with k columns, even though the two tasks are algebraically equivalent. (See Table 1 in section 6 for an illustration.)
2. The vectors to which B is applied do not depend on B itself. This means that a fixed set of k vectors can be used to analyze a whole collection of low-rank matrices.

The flexibility described in the two points above is crucial in the present context since we are interested in approximating not a single low-rank matrix B, but a collection of submatrices $\{B_i\}_i$ of a matrix A which can rapidly be applied to vectors.

Remark 2.3. The approximate rank k is rarely known in advance. In a situation where a single matrix B of numerically low rank is to be analyzed, it is a straightforward matter to modify the algorithm described here to an algorithm that adaptively determines the numerical rank by generating a sequence of samples from the column space of B and simply stops when no more information is added [23]. In the application we have in mind in this paper, however, the randomized scheme will be used in such a way that a single random matrix will be used to create samples from a large set of different matrices. In this case, we make an estimate k of the largest rank that we could possibly encounter, and pick a random matrix with $k + 10$ columns. Then as each off-diagonal block is processed, its true ε-rank will be revealed by executing a rank-revealing QR factorization in "Step 4" of the algorithm listed above.

2.4. Computing interpolative decompositions via randomized sampling.

The randomized sampling technique described in section 2.3 is particularly effective when used in conjunction with the interpolative decomposition. To illustrate, let us suppose that B is an $n \times n$ matrix of rank k for which we can rapidly evaluate the maps $x \mapsto B\,x$ and $x \mapsto B^*\,x$. Using the randomized sampling technique, we then draw a random matrix Ω and construct matrices $S^{\rm col} = B\,\Omega$ and $S^{\rm row} = B^*\,\Omega$ whose columns span the column and the row spaces of B, respectively. If we seek to construct a factorization of B without using the interpolative decomposition, we would then orthonormalize the columns of $S^{\rm col}$ and $S^{\rm row}$,

$$[Q^{\rm col},\, Y^{\rm col}] = \texttt{qr}(S^{\rm col}) \qquad \text{and} \qquad [Q^{\rm row},\, Y^{\rm row}] = \texttt{qr}(S^{\rm row}),$$

whence

$$B = Q^{\rm col}\,\big((Q^{\rm col})^*\,B\,Q^{\rm row}\big)\,(Q^{\rm row})^*. \tag{2.4}$$

The evaluation of (2.4) requires k matrix-vector multiplications involving the large matrix B in order to compute the $k \times k$ matrix $(Q^{\rm col})^*\,B\,Q^{\rm row}$. Using the interpolative decomposition instead, we determine the k rows of $S^{\rm col}$ and $S^{\rm row}$ that span their respective row spaces,

$$[X^{\rm col},\, J^{\rm col}] = \texttt{interpolate}((S^{\rm col})^*) \quad \text{and} \quad [X^{\rm row},\, J^{\rm row}] = \texttt{interpolate}((S^{\rm row})^*).$$

Then by simply extracting the $k \times k$ submatrix $B(J^{\rm col},\, J^{\rm row})$ from B, we directly obtain the factorization

$$B = X^{\rm col}\,B(J^{\rm col},\, J^{\rm row})\,(X^{\rm row})^*. \tag{2.5}$$

More details (including an error analysis for the case when B has only approximate rank k) can be found in section 5.2 of [23].

3. Hierarchically semiseparable matrices.

In this section we rigorously define the concept of an HSS matrix and review some basic results that are slight variations of techniques found in, e.g., [9, 30, 35, 36, 38]. Unfortunately, the notation required can come across as quite daunting at first. (The same is true for essentially all data-sparse formats that we know of.) We therefore start with an attempt to describe the general ideas in section 3.1 before introducing the rigorous notational framework in sections 3.2 and 3.3. In section 3.4, we describe a simple condition for checking whether a matrix satisfies the HSS property. Finally we describe in section 3.5 how the HSS representation of a matrix can be viewed as a telescoping factorization.

3.1. Intuition.

The HSS property is essentially a condition that the off-diagonal blocks of a matrix should have low rank, combined with a condition that the factors used to represent the off-diagonal blocks satisfy certain recursive relations that make them inexpensive to store and to apply.

To illustrate, suppose that a matrix A has been tesselated as shown in Figure 3.1. Then the first condition is that there exists a fixed (small) integer k such that every off-diagonal block in the tessellation should have rank at most k. When this condition holds, every off-diagonal block $A_{i,j}$ admits a factorization

$$A_{i,j} = U_i^{\rm big}\,A'_{i,j},$$

where $U_i^{\rm big}$ is a matrix with k columns that form a basis for the range of $A_{i,j}$. The second condition is now that all the "basis matrices" $U_i^{\rm big}$ can be expressed hierarchically. Consider, for instance, the matrix $A_{2,3}$. The row indices that form $A_{2,3}$ are

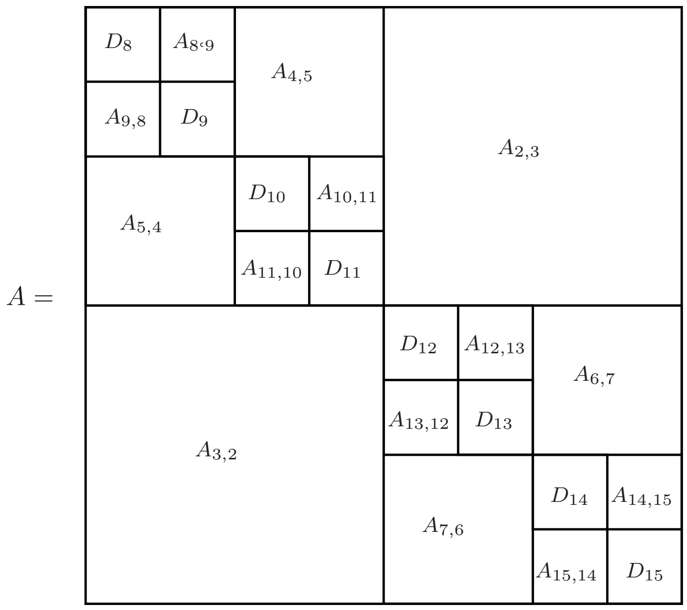


FIG. 3.1. *A matrix A tesselated in accordance with the tree in Figure* 3.2.

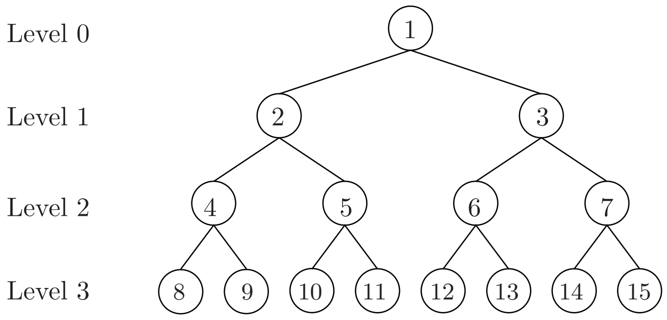


FIG. 3.2. *Numbering of nodes in a fully populated binary tree with $L = 3$ levels.*

the union of the row indices that form $A_{4,5}$ and $A_{5,4}$. We therefore require that there exist a $2k \times k$ matrix U_2 such that

$$U_2^{\text{big}} = \begin{bmatrix} U_4^{\text{big}} & 0 \\ 0 & U_5^{\text{big}} \end{bmatrix} U_2. \tag{3.1}$$

The point is that if U_4^{big} and U_5^{big} have been computed, then we can express U_2^{big} by storing only the small matrix U_2. This process will be continued recursively. Requiring analogously that U_4^{big} and U_5^{big} satisfy

$$U_4^{\text{big}} = \begin{bmatrix} U_8^{\text{big}} & 0 \\ 0 & U_9^{\text{big}} \end{bmatrix} U_4 \qquad \text{and} \qquad U_5^{\text{big}} = \begin{bmatrix} U_{10}^{\text{big}} & 0 \\ 0 & U_{11}^{\text{big}} \end{bmatrix} U_5, \tag{3.2}$$

we find by combining (3.1) with (3.2) that

$$U_2^{\text{big}} = \begin{bmatrix} U_4^{\text{big}} & 0 \\ 0 & U_5^{\text{big}} \end{bmatrix} U_2 = \begin{bmatrix} U_8^{\text{big}} & 0 & 0 & 0 \\ 0 & U_9^{\text{big}} & 0 & 0 \\ 0 & 0 & U_{10}^{\text{big}} & 0 \\ 0 & 0 & 0 & U_{11}^{\text{big}} \end{bmatrix} \begin{bmatrix} U_4 & 0 \\ 0 & U_5 \end{bmatrix} U_2.$$

The end result is that the only "big" basis matrices that actually need to be computed and stored are those for the smallest off-diagonal blocks; all other basis matrices are represented by storing only a small $2k \times k$ matrix.

To turn this informal discussion into a rigorous definition of the HSS property for an $N \times N$ matrix A, we introduce in section 3.2 a tree structure on the index vector $[1, 2, \ldots, N]$ and define in section 3.3 a tessellation of A based on the tree structure. We can then formally introduce the basis matrices for each off-diagonal block, and formulate a hierarchical condition that these basis matrices must satisfy.

Remark 3.1. In our discussion of HSS matrices, we assume that the off-diagonal blocks have exact rank k. In the matrices encountered in practice, it is typically the case that the rank of the off-diagonal blocks is k up to some finite computational tolerance ε. Truncating the actual matrix to its rank-k HSS approximation does not in our experience lead to any loss of accuracy beyond the specified tolerance ε.

3.2. Tree structure. The HSS representation of an $N \times N$ matrix A is given with respect to a specific hierarchical partitioning of the index vector $[1, 2, \ldots, N]$. To keep the presentation simple, we restrict attention to binary trees in which all levels are fully populated, and in which all nodes on a given level contain roughly the same number of indices. We number the nodes as illustrated for a tree with $L = 3$ levels in Figure 3.2. We define the terms *children, parent,* and *sibling* in the natural way; for instance, the children of node 3 are the nodes 6 and 7, and the nodes 6 and 7 form a sibling pair. The childless nodes are called *leaves.*

With each node τ, we associate an index vector I_τ that forms a subset of I. We set $I_1 = I$, and form I_2 and I_3 by splitting I_1 into halves. We form I_4 and I_5 by splitting I_2 into halves, etc. The cutting in half continues until we obtain index vectors with no more than a fixed number, say 50, of entries. For instance, if $N = 400$, we get the following index vectors:

Level 0: $I_1 = I = [1, 2, \ldots, 400]$,
Level 1: $I_2 = [1, 2, \ldots, 200]$, $I_3 = [201, 202, \ldots, 400]$,
Level 2: $I_4 = [1, 2, \ldots, 100]$, $I_5 = [101, 102, \ldots, 200], \ldots, I_7 = [301, 302, \ldots, 400]$,
Level 3: $I_8 = [1, 2, \ldots, 50]$, $I_9 = [51, 52, \ldots, 100], \ldots, I_{15} = [351, 352, \ldots, 400]$.

The *tree* $\mathcal{T}$ is now the set of all index vectors, $\mathcal{T} = \{I_1, I_2, \ldots, I_{2^{L+1}-1}\}$. Finally, we note that if we allow at most, say, 50 nodes in the index vectors for the leaves, then L satisfies

$$L \approx 2 \log_2(N/50).$$

Remark 3.2. The HSS format can handle more general tree structures than the simplistic one used here. It is permissible to split a node into more than two children if desirable, to distribute the points in an index set unevenly among its children, to split only some nodes on a given level, etc. This flexibility is useful when A approximates a boundary integral operator since in this case the optimal way of organizing the index vector into a tree structure is determined by the spatial geometry of the boundary points.

3.3. Basis matrices. The index vectors in a tree $\mathcal{T}$ define submatrices of A. For any node τ, we define the corresponding diagonal block via

$$D_\tau = A(I_\tau,\, I_\tau).$$

For any sibling pair ν_1 and ν_2, we define the corresponding off-diagonal blocks via

$$A_{\nu_1,\nu_2} = A(I_{\nu_1},\, I_{\nu_2}) \qquad \text{and} \qquad A_{\nu_2,\nu_1} = A(I_{\nu_2},\, I_{\nu_1}).$$

A tessellation of A is formed by the collection of all off-diagonal blocks A_{ν_1,ν_2} along with the basis blocks D_τ for all leaf nodes τ, as illustrated in Figure 3.1.

The matrix A is now an HSS matrix with respect to the tree $\mathcal{T}$ if there exists a fixed integer k (called the "HSS-rank") such that the following hold:

Condition 1. For each sibling pair $\{\nu_1, \nu_2\}$, the off-diagonal block A_{ν_1,ν_2} has rank at most k. We factor the blocks

$$(3.3) \qquad A_{\nu_1,\nu_2} = U^{\rm big}_{\nu_1}\, B_{\nu_1,\nu_2}\, (V^{\rm big}_{\nu_2})^* \quad \text{and} \quad A_{\nu_2,\nu_1} = U^{\rm big}_{\nu_2}\, B_{\nu_2,\nu_1}\, (V^{\rm big}_{\nu_1})^*,$$

where $U^{\rm big}_{\nu_j}$ and $V^{\rm big}_{\nu_j}$ are matrices with k columns each, and B_{ν_1,ν_2} and B_{ν_2,ν_1} are $k \times k$ matrices.

Condition 2. The basis matrices $U^{\rm big}_\tau$ and $V^{\rm big}_\tau$ can be represented hierarchically. Specifically, for any nonleaf node τ with children ν_1 and ν_2, there must exist $2k \times k$ matrices U_τ and V_τ such that

$$(3.4) \quad U^{\rm big}_\tau = \begin{bmatrix} U^{\rm big}_{\nu_1} & 0 \\ 0 & U^{\rm big}_{\nu_2} \end{bmatrix} U_\tau \qquad \text{and} \qquad V^{\rm big}_\tau = \begin{bmatrix} V^{\rm big}_{\nu_1} & 0 \\ 0 & V^{\rm big}_{\nu_2} \end{bmatrix} V_\tau.$$

To illustrate the notation, we note that if τ is a nonleaf node with children ν_1 and ν_2,

$$D_\tau = \begin{bmatrix} D_{\nu_1} & A_{\nu_1,\nu_2} \\ A_{\nu_2,\nu_1} & D_{\nu_2} \end{bmatrix} = \begin{bmatrix} D_{\nu_1} & U^{\rm big}_{\nu_1}\, B_{\nu_1,\nu_2}\, (V^{\rm big}_{\nu_2})^* \\ U^{\rm big}_{\nu_2}\, B_{\nu_2,\nu_1}\, (V^{\rm big}_{\nu_1})^* & D_{\nu_2} \end{bmatrix}.$$

Observing that for a leaf node τ the basis matrices $U^{\rm big}_\tau$ and $V^{\rm big}_\tau$ are in fact not big, we define

$$\text{For any leaf node } \tau: \quad U_\tau = U^{\rm big}_\tau, \qquad V_\tau = V^{\rm big}_\tau.$$

Then an HSS matrix A is completely described if
- for every leaf node τ, we are given the diagonal matrices D_τ;
- for every node τ, we are given the small basis matrices U_τ and V_τ;
- for every sibling pair $\{\nu_1,\, \nu_2\}$, we are given the $k \times k$ matrices B_{ν_1,ν_2} and B_{ν_2,ν_1}.

A scheme for evaluating the matrix-vector product $x \mapsto A\,x$ from these factors in $O(N\,k)$ flops is given as Algorithm 1.

Remark 3.3. For notational simplicity, we have described only the case where all HSS blocks are approximated by factorizations of the same rank k. In practice, it is a simple matter to implement algorithms that use variable ranks.

Remark 3.4. It is common to require the matrices U_τ and V_τ that arise in an HSS factorization of a given matrix to have orthonormal columns. We have found it convenient to relax this assumption and allow the use of other well-conditioned bases. In particular, the use of interpolative decompositions (as described in section 2.2) is essential to the performance of the compression technique described in section 4.1. A simple algorithm for converting a factorization based on interpolative bases to one based on orthonormal bases is described in section 4.2.

Algorithm 1.

Given all factors U_τ, V_τ, B_{ν_1,ν_2}, and D_τ of an HSS matrix A, and given a vector x, this scheme computes the product $b = A\,x$.

```
(1) For every leaf node $\tau$, calculate $\tilde{x}_\tau = V_\tau^*\, x(I_\tau)$.
(2) Looping over all nonleaf nodes $\tau$, from finer to coarser, calculate
        $\tilde{x}_\tau = V_\tau^* \begin{bmatrix} \tilde{x}_{\nu_1} \\ \tilde{x}_{\nu_2} \end{bmatrix}$,
    where $\nu_1$ and $\nu_2$ are the children of $\tau$.
(3) Set $\tilde{b}_\tau = 0$ for the root node $\tau$.
(4) Looping over all nonleaf nodes $\tau$, from coarser to finer, calculate
        $\begin{bmatrix} \tilde{b}_{\nu_1} \\ \tilde{b}_{\nu_2} \end{bmatrix} = \begin{bmatrix} 0 & B_{\nu_1,\nu_2} \\ B_{\nu_2,\nu_1} & 0 \end{bmatrix} \begin{bmatrix} \tilde{x}_{\nu_1} \\ \tilde{x}_{\nu_2} \end{bmatrix} + U_\tau\, \tilde{b}_\tau$,
    where $\nu_1$ and $\nu_2$ are the children of $\tau$.
(5) For every leaf node $\tau$, calculate $b(I_\tau) = U_\tau\, \tilde{b}_\tau + D_\tau\, x(I_\tau)$.
```

3.4. Construction of the basis matrices. The definition of the HSS property given in section 3.3 involves the existence of a set of basis matrices satisfying certain recursive relations. It does not directly provide a means of testing whether such matrices exist. In this section we provide a directly verifiable criterion that also leads to a simple (but expensive) technique for computing the basis matrices.

We first define what are called *HSS row blocks* and *HSS column blocks.* To this end, let ℓ denote a level of the tree, and let $\{\tau_1, \tau_2, \dots, \tau_q\}$ denote all nodes on level ℓ. Then let $D^{(\ell)}$ denote the $N \times N$ matrix with the matrices $\{D_{\tau_j}\}_{j=1}^q$ as its diagonal blocks,

$$D^{(\ell)} = \mathrm{diag}(D_{\tau_1},\, D_{\tau_2}, \dots,\, D_{\tau_q}\}. \tag{3.5}$$

For a node τ on level ℓ, we now define the corresponding *HSS row block* A_τ^{row} and *HSS column block* A_τ^{col} by

$$A_\tau^{\mathrm{row}} = A(I_\tau, :) - D^{(\ell)}(I_\tau, :) \qquad \text{and} \qquad A_\tau^{\mathrm{col}} = A(:, I_\tau) - D^{(\ell)}(:, I_\tau).$$

These definitions are illustrated in Figure 3.3.

The following theorem provides a simple test to see whether a matrix A is HSS, and also furnishes a straightforward (but expensive) technique for computing all the basis matrices.

THEOREM 3.1. *Let $\mathcal{T}$ be a tree on the indices* $1, 2, \dots, N$, *and let A be an $N \times N$ matrix. If every HSS row block A_τ^{row} and every HSS column block A_τ^{col} of A has rank at most k, then A is an HSS matrix with respect to $\mathcal{T}$ with HSS-rank at most k.*

The basis matrices U_τ^{big} and V_τ^{big} can be computed via

$$[U_\tau^{\mathrm{big}},\, \cdot\,] = \mathtt{qr}(A_\tau^{\mathrm{row}}) \qquad \text{and} \qquad [V_\tau^{\mathrm{big}},\, \cdot\,] = \mathtt{qr}((A_\tau^{\mathrm{col}})^*). \tag{3.6}$$

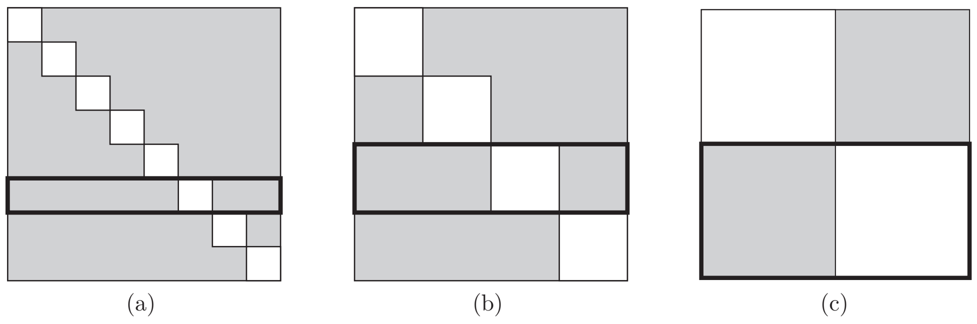


FIG. 3.3. (a) *The matrix* $A - D^{(3)}$ *with nonzero parts shaded. The HSS row block* A^{row}_{13} *is marked with a thick border.* (b) *The matrix* $A - D^{(2)}$ *with* A^{row}_{6} *marked.* (c) *The matrix* $A - D^{(1)}$ *with* A^{row}_{3} *marked.*

Once the big basis matrices have been computed for all blocks, the small ones can easily be constructed. This approach requires $O(N^2\,\log(N)\,k)$ operations to compute a representation of A as an HSS matrix. A more efficient scheme computes the big basis matrices via (3.6) for the leaf nodes only. Then information from the leaf computation is recycled to directly compute the small basis matrices U_τ and V_τ, at a total cost of $O(N^2\,k)$ operations.

3.5. Telescoping factorization. An HSS matrix A can be expressed in terms of the matrices D_τ, U_τ, V_τ, and B_{ν_1,ν_2} as a telescoping factorization. To demonstrate, we introduce for each level $\ell = 1, 2, \dots, L$ the block-diagonal matrices

$$U^{(\ell)} = \mathrm{diag}(U_{\tau_1}, U_{\tau_2}, \dots, U_{\tau_q}) \qquad \text{and} \qquad V^{(\ell)} = \mathrm{diag}(V_{\tau_1}, V_{\tau_2}, \dots, V_{\tau_q}),$$

where $\tau_1, \tau_2, \dots, \tau_q$ is a list of all nodes on level ℓ. Moreover, we define for each nonleaf node τ the $2k \times 2k$ matrices

$$B_\tau = \begin{bmatrix} 0 & B_{\nu_1,\nu_2} \\ B_{\nu_2,\nu_1} & 0 \end{bmatrix},$$

where ν_1 and ν_2 are the children of τ, and set for $\ell = 0, 1, \dots, L-1$

$$B^{(\ell)} = \mathrm{diag}(B_{\tau_1}, B_{\tau_2}, \dots, B_{\tau_q}).$$

Then A can be expressed hierarchically via the relations

(3.7) $$A^{(0)} = B^{(0)},$$

(3.8) $$A^{(\ell)} = U^{(\ell)}\,A^{(\ell-1)}\,(V^{(\ell)})^* + B^{(\ell)} \qquad \text{for } \ell = 1, 2, \dots, L-1,$$

(3.9) $$A = U^{(L)}\,A^{(L-1)}\,(V^{(L)})^* + D^{(L)},$$

where $D^{(L)}$ is defined via (3.5). (The matrices $A^{(\ell)}$ defined by (3.7) and (3.8) satisfy $A^{(\ell)} = A - D^{(\ell)}$.) Rolling out the recursion, we get for, say, $L = 3$ the telescoping factorization

(3.10) $$A = U^{(3)}\big(U^{(2)}\big(U^{(1)}\,B^{(0)}\,V^{(1)^*} + B^{(1)}\big)V^{(2)^*} + B^{(2)}\big)V^{(3)^*} + D^{(3)}.$$

The block structure of the formula (3.10) is shown in Figure 3.4. In practice, the sparsity pattern of the blocks is typically slightly better than that shown in the figure. When interpolative decompositions are used, each of the diagonal blocks in the

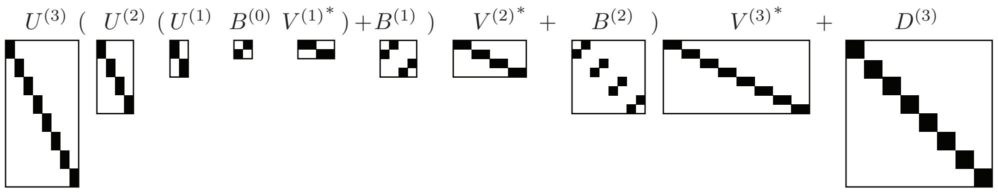

$U^{(3)}$ ($U^{(2)}$ ($U^{(1)}$ $B^{(0)}$ $V^{(1)^*}$) + $B^{(1)}$) $V^{(2)^*}$ + $B^{(2)}$) $V^{(3)^*}$ + $D^{(3)}$

FIG. 3.4. *Block structure of* (3.10).

matrices $U^{(\ell)}$ and $V^{(\ell)}$ contains an identity matrix. When orthonormal bases are used, these can be chosen in such a way that the blocks in the $B^{(\ell)}$ matrices are all diagonal.

4. Fast computation of HSS approximations.

The straightforward technique for computing the HSS factorization of a matrix based on Theorem 3.1 requires that all HSS blocks associated with leaf nodes be formed and then subjected to dense linear algebraic operations. This approach requires at least $O(N^2\,k)$ algebraic operations to factorize an $N \times N$ matrix of HSS rank k. In this section, we describe how the randomized sampling techniques described in section 2.3 can be used to reduce this cost to $O(N\,k^2)$.

The fast technique relies crucially on the use of the interpolative decompositions described in section 2.2 in the HSS factorization. The advantage is that the matrices B_{ν_1,ν_2} are then *submatrices* of the original matrix A and can therefore be constructed directly without a need for projecting the larger blocks onto the bases chosen; cf. section 2.4.

Section 4.1 describes a scheme for rapidly computing the HSS factorization of a symmetric matrix. The scheme described in section 4.1 results in a factorization based on interpolative bases, and the blocks B_{ν_1,ν_2} are submatrices of the original matrix; section 4.2 describes how such a factorization can be converted to one in which the bases for the HSS blocks are orthonormal and the blocks B_{ν_1,ν_2} are diagonal. Section 4.3 describes how to extend the methods to nonsymmetric matrices.

4.1. A scheme for computing an HSS factorization of a symmetric matrix.

Let A be an $N \times N$ symmetric HSS matrix that has HSS-rank k. Suppose further that

(a) matrix-vector products $x \mapsto A\,x$ can be evaluated at a cost $T_{\rm mult}$,

(b) individual entries of A can be evaluated at a cost $T_{\rm entry}$.

In this section, we describe a scheme for computing an HSS factorization of A in time

$$T_{\rm total} \sim T_{\rm mult} \times (k+10) + T_{\rm rand} \times N\,(k+10) + T_{\rm entry} \times 2\,N\,k + T_{\rm flop} \times c\,N\,k^2,$$

where $T_{\rm rand}$ is the time required to generate a random number, $T_{\rm flop}$ is the time required for a floating point operation, and c is a small number that does not depend on N or k.

The core idea of the method is to construct an $N \times (k+10)$ random matrix Ω, and then construct, for each level ℓ, the *sample* matrices

$$S^{(\ell)} = \left(A - D^{(\ell)}\right)\Omega$$

via a procedure to be described. Then, for any cell τ on level ℓ,

$$S^{(\ell)}(I_\tau, :) = A_\tau^{\rm row}\,\Omega,$$

and since A_τ^{row} has rank k, the columns of $S^{(\ell)}(I_\tau, :)$ span the column space of A_τ^{row} with high probability. We can then construct a basis for the column space of the large matrix A_τ^{row} by analyzing the small matrix $S^{(\ell)}(I_\tau, :)$.

What makes the procedure fast is that the sample matrices $S^{(\ell)}$ can be constructed by means of an $O(N)$ process from the result of applying the *entire* matrix A to Ω,

$$S = A\,\Omega.$$

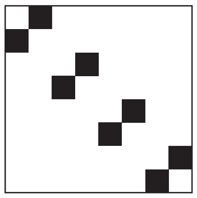
At the finest level, $\ell = L$, we directly obtain $S^{(L)}$ from S by simply subtracting the contribution from the diagonal blocks of A,

$$S^{(L)} = S - D^{(L)}\,\Omega. \tag{4.1}$$

Since $D^{(L)}$ is block diagonal with small blocks, (4.1) can be evaluated cheaply. To proceed to the next coarser level, $\ell = L - 1$, we observe that

$$\begin{aligned} S^{(L-1)} &= (A - D^{(L-1)})\,\Omega = (A - D^{(L)})\,\Omega - (D^{(L-1)} - D^{(L)})\,\Omega \\ &= S^{(L)} - (D^{(L-1)} - D^{(L)})\,\Omega. \end{aligned} \tag{4.2}$$

We observe that $(D^{(L-1)} - D^{(L)})$ has only 2^L nonzero blocks. The pattern of these blocks is illustrated for $L = 3$ below:



Each of these blocks was compressed in the computation at level L, so (4.2) can also be evaluated rapidly. The algorithm then proceeds up towards coarser levels via the formula

$$S^{(\ell-1)} = S^{(\ell)} - (D^{(\ell-1)} - D^{(\ell)})\,\Omega,$$

which can be evaluated rapidly since the blocks of $(D^{(\ell-1)} - D^{(\ell)})$ have at this point been compressed.

The condition that the bases be "nested" in the sense of formula (3.4) can conveniently be enforced by using the interpolative decompositions described in section 2.2: At the finest level, we pick k rows of each HSS row block that span its row space. At the next coarser level, we pick in each HSS row block k rows that span its row space *out of the* $2k$ *rows that span its two children.* By proceeding analogously throughout the upward pass, (3.4) will be satisfied. The k rows that span the row space of A_τ^{row} are kept in the index vector $\tilde{I}_\tau$.

The use of interpolative decompositions has the additional benefit that we do not need to form the entire matrices $S^{(\ell)}$ when $\ell < L$. Instead, we work with the submatrices formed by keeping only the rows of $S^{(\ell)}$ corresponding to the spanning rows at that step.

A complete description of the methods is given as Algorithm 2.

Remark 4.1. For simplicity, Algorithm 2 is described for the case where the off-diagonal blocks of A have exact rank at most k, and where the number k is known in advance. In actual applications, one typically is given a matrix A whose off-diagonal

Algorithm 2. Computing the HSS factorization of a symmetric matrix.

Input: A fast means of computing matrix-vector products $x \mapsto A\,x$.
A method for computing individual entries of A.
An upper bound for the HSS-rank k of A.
A tree $\mathcal{T}$ on the index vector $[1, 2, \ldots, N]$.

Output: Matrices U_τ, B_{ν_1,ν_2}, D_τ that form an HSS factorization of A.
(Note that $V_\tau = U_\tau$ for a symmetric matrix.)

```
Generate an $N \times (k+10)$ Gaussian random matrix $\Omega$.
Evaluate $S = A\,\Omega$ using the fast matrix-vector multiplier.
loop over levels, finer to coarser, $\ell = L, L-1, \ldots, 2, 1$
    loop over all nodes $\tau$ on level $\ell$
        if $\tau$ is a leaf node, then
            $I_{\rm loc} = I_\tau$
            $\Omega_{\rm loc} = \Omega(I_\tau, :)$
            $S_{\rm loc} = S(I_\tau, :) - A(I_\tau, I_\tau)\,\Omega_{\rm loc}$
        else
            Let $\nu_1$ and $\nu_2$ be the two children of $\tau$.
            $I_{\rm loc} = [\tilde{I}_{\nu_1}, \tilde{I}_{\nu_2}]$
            $\Omega_{\rm loc} = \begin{bmatrix} \Omega_{\nu_1} \\ \Omega_{\nu_2} \end{bmatrix}$
            $S_{\rm loc} = \begin{bmatrix} S_{\nu_1} - A(\tilde{I}_{\nu_1}, \tilde{I}_{\nu_2})\,\Omega_{\nu_2} \\ S_{\nu_2} - A(\tilde{I}_{\nu_2}, \tilde{I}_{\nu_1})\,\Omega_{\nu_1} \end{bmatrix}$
        end if
        $[U_\tau, J_\tau] = \texttt{interpolate}(S_{\rm loc}^*)$
        $\Omega_\tau = U_\tau^*\,\Omega_{\rm loc}$
        $S_\tau = S_{\rm loc}(J_\tau, :)$
        $\tilde{I}_\tau = I_{\rm loc}(J_\tau)$
    end loop
end loop
For all leaf nodes $\tau$, set $D_\tau = A(I_\tau, I_\tau)$.
For all sibling pairs $\{\nu_1, \nu_2\}$, set $B_{\nu_1,\nu_2} = A(\tilde{I}_{\nu_1}, \tilde{I}_{\nu_2})$.
```

blocks are not necessarily rank-deficient in an exact sense, but can to high accuracy be approximated by low-rank matrices. In this case, Algorithm 2 needs to be modified slightly to take as an input the computational accuracy ε instead of the rank k, and the line

$$[U_\tau, J_\tau] = \texttt{interpolate}(S_{\rm loc}^*)$$

needs to be replaced by the line

$$[U_\tau,\ J_\tau] = \text{interpolate}(S_{\rm loc}^*,\ \varepsilon).$$

This directly leads to a variable-rank algorithm that is typically far more efficient than the fixed-rank algorithm described.

4.2. Recompression into orthonormal basis functions. The output of Algorithm 2 is an HSS representation of a matrix in which interpolative bases are used.

It is sometimes desirable to convert this representation to one using orthonormal bases (cf. Remark 3.4). In this section, we describe a technique for doing so that is similar to a technique for *sequentially semiseparable* matrices reported in [13] and a technique for HSS matrices reported in [9]. To be precise, suppose that matrices U_τ, D_τ, and B_{ν_1,ν_2} in an HSS factorization of a symmetric matrix have already been generated. (These may have been generated by Algorithm 2, or by some other means. All that matters is that A can be factored as specified by (3.7), (3.8), (3.9).) The method described in this section produces new matrices $U_\tau^{\rm new}$ and $B_{\nu_1,\nu_2}^{\rm new}$ with the property that each $U_\tau^{\rm new}$ has orthonormal columns, and each $B_{\nu_1,\nu_2}^{\rm new}$ is diagonal.

The orthonormalization procedure works hierarchically, starting at the finest level and working upwards. At the finest level, it loops over all sibling pairs $\{\nu_1,\,\nu_2\}$. It orthonormalizes the basis matrices U_{ν_1} and U_{ν_2} by computing their QR factorizations,

$$[W_1,\,R_1] = \texttt{qr}(U_{\nu_1}) \qquad \text{and} \qquad [W_2,\,R_2] = \texttt{qr}(U_{\nu_2}),$$

so that

$$U_{\nu_1} = W_1\,R_1 \qquad \text{and} \qquad U_{\nu_2} = W_2\,R_2,$$

and W_1 and W_2 have orthonormal columns. The matrices R_1 and R_2 are then used to update the diagonal block B_{ν_1,ν_2} to reflect the change in basis vectors,

$$\tilde{B}_{12} = R_1\,B_{\nu_1,\nu_2}\,R_2^*.$$

Then $\tilde{B}_{12}$ is diagonalized via an SVD,

$$\tilde{B}_{12} = \tilde{W}_1\,B_{\nu_1,\nu_2}^{\rm new}\,\tilde{W}_2^*.$$

The new bases for ν_1 and ν_2 are constructed by updating W_1 and W_2 to reflect the diagonalization of $\tilde{B}_{12}$,

$$U_{\nu_1}^{\rm new} = W_1\,\tilde{W}_1 \qquad \text{and} \qquad U_{\nu_2}^{\rm new} = W_2\,\tilde{W}_2.$$

Finally, the basis vectors for the parent τ of ν_1 and ν_2 must be updated to reflect the change in bases at the finer level,

$$U_\tau \leftarrow \left[\begin{array}{cc} \tilde{W}_1^*\,R_1 & 0 \\ 0 & \tilde{W}_2^*\,R_2 \end{array}\right] U_\tau.$$

Once the finest level has been processed, move up to the next coarser one and proceed analogously. A complete description of the recompression scheme is given as Algorithm 3.

4.3. Nonsymmetric matrices. The extension of Algorithms 2 and 3 to the case of nonsymmetric matrices is conceptually straightforward but requires the introduction of more notation. In Algorithm 2, we construct a set of sample matrices $\{S_\tau\}$ with the property that the columns of each S_τ span the column space of the corresponding HSS row block $A_\tau^{\rm row}$. Since A is in that case symmetric, the columns of S_τ automatically span the row space of $A_\tau^{\rm col}$ as well. For nonsymmetric matrices, we need to construct different sample matrices $S_\tau^{\rm row}$ and $S_\tau^{\rm col}$ whose columns span the column space of $A_\tau^{\rm row}$ and the row space of $A_\tau^{\rm col}$, respectively. Note that in practice, we work only with the subsets of all these matrices formed by the respective spanning rows and columns; in the nonsymmetric case, these may be different. The algorithm is given in full as Algorithm 4. The generalization of the orthonormalization technique in Algorithm 3 is entirely analogous.

Algorithm 3. Orthonormalizing an HSS factorization.

Input: The matrices U_τ, B_{ν_1,ν_2}, D_τ in an HSS factorization of a symmetric matrix A.

Output: Matrices U_τ^{new}, $B_{\nu_1,\nu_2}^{\text{new}}$, and D_τ that form an HSS factorization of A such that all U_τ^{new} have orthonormal columns and all $B_{\nu_1,\nu_2}^{\text{new}}$ are diagonal. (The matrices D_τ remain unchanged.)

```
Set $U_\tau^{\text{tmp}} = U_\tau$ for all leaf nodes $\tau$.
loop over levels, finer to coarser, $\ell = L-1,\, L-2, \dots, 0$
    loop over all nodes $\tau$ on level $\ell$
        Let $\nu_1$ and $\nu_2$ denote the two children of $\tau$.
        $[W_1,\, R_1] = \texttt{qr}(U_{\nu_1}^{\text{tmp}})$
        $[W_2,\, R_2] = \texttt{qr}(U_{\nu_2}^{\text{tmp}})$
        $[\tilde{W}_1,\, B_{\nu_1,\nu_2}^{\text{new}},\, \tilde{W}_2] = \texttt{svd}(R_1\, B_{\nu_1,\nu_2}\, R_2^*)$
        $U_{\nu_1}^{\text{new}} = W_1\, \tilde{W}_1$
        $U_{\nu_2}^{\text{new}} = W_2\, \tilde{W}_2$
        $U_\tau^{\text{tmp}} = \begin{bmatrix} \tilde{W}_1^*\, R_1 & 0 \\ 0 & \tilde{W}_2^*\, R_2 \end{bmatrix} U_\tau$
    end loop
end loop
```

Remark: In practice, we let the matrices U_τ^{tmp} and U_τ^{new} simply overwrite U_τ.

5. Numerical examples. In this section, we demonstrate the performance of the techniques described in section 4 by applying them to matrices arising from the discretization of two boundary integral operators associated with Laplace's equation in two dimensions. The purpose of the experiments is (1) to investigate how the computational time of Algorithms 2 and 4 depends on problem size, (2) to see to what extent local errors aggregate, and (3) to compare the speeds of Algorithms 2 and 4 to the speed of the classical FMM. All methods were implemented in MATLAB and run on a desktop PC with a 3.2GHz Pentium IV processor and 2GB of RAM.

5.1. Model problems. The matrices investigated are discrete approximations of the boundary integral operator

$$[Tu](x) = \alpha\, u(x) + \int_\Gamma K(x,y)\, u(y)\, ds(y), \qquad x \in \Gamma, \tag{5.1}$$

where Γ is the contour shown in Figure 5.1 and α and K are chosen as either one of the following two options:

(5.2)
$$\alpha = 0 \quad \text{and} \quad K(x,y) = \log|x-y| \qquad \text{(the “single layer” kernel)},$$

(5.3)
$$\alpha = 1/2 \quad \text{and} \quad K(x,y) = \big(n(y)\cdot(x-y)\big)/|x-y|^2 \quad \text{(the “double layer” kernel)}.$$

For $y \in \Gamma$, $n(y)$ denotes the unit normal of Γ at y. The single layer operator was discretized via the trapezoidal rule with a Kapur–Rokhlin [25] end-point modification of the sixth order for handling the singularity in the kernel $k(x,y)$ as y approaches

Algorithm 4. Computing the HSS factorization of a nonsymmetric matrix.

Input: A fast means of computing matrix-vector products $x \mapsto A\,x$ and $x \mapsto A^*\,x$.
A method for computing individual entries of A.
An upper bound for the HSS-rank k of A.
A tree $\mathcal{T}$ on the index vector $[1, 2, \dots, N]$.

Output: Matrices U_τ, V_τ, B_{ν_1,ν_2}, D_τ that form an HSS factorization of A.

```
Generate two $N \times (k+10)$ Gaussian random matrices $R^{\rm row}$ and $R^{\rm col}$.
Evaluate $S^{\rm row} = A^*\,R^{\rm row}$ and $S^{\rm col} = A\,R^{\rm col}$ using the fast matrix-vector multiplier.
loop over levels, finer to coarser, $\ell = L, L-1, \dots, 1$
    loop over all nodes $\tau$ on level $\ell$
        if $\tau$ is a leaf node, then
            $I^{\rm row}_{\rm loc} = I_\tau$                                   $I^{\rm col}_{\rm loc} = I_\tau$
            $R^{\rm row}_{\rm loc} = R(I_\tau, :)$                             $R^{\rm col}_{\rm loc} = R(I_\tau, :)$
            $S^{\rm row}_{\rm loc} = S^{\rm row}(I_\tau, :) - A(I_\tau, I_\tau)\,R^{\rm row}_{\rm loc}$     $S^{\rm col}_{\rm loc} = S^{\rm col}(I_\tau, :) - A(I_\tau, I_\tau)^*\,R^{\rm col}_{\rm loc}$
        else
            Let $\nu_1$ and $\nu_2$ be the two children of $\tau$.
            $I^{\rm row}_{\rm loc} = [\tilde{I}^{\rm row}_{\nu_1}, \tilde{I}^{\rm row}_{\nu_2}]$     $I^{\rm col}_{\rm loc} = [\tilde{I}^{\rm col}_{\nu_1}, \tilde{I}^{\rm col}_{\nu_2}]$
            $R^{\rm row}_{\rm loc} = \begin{bmatrix} R^{\rm row}_{\nu_1} \\ R^{\rm row}_{\nu_2} \end{bmatrix}$     $R^{\rm col}_{\rm loc} = \begin{bmatrix} R^{\rm col}_{\nu_1} \\ R^{\rm col}_{\nu_2} \end{bmatrix}$
            $S^{\rm row}_{\rm loc} = \begin{bmatrix} S^{\rm row}_{\nu_1} - A(\tilde{I}^{\rm row}_{\nu_1}, \tilde{I}^{\rm col}_{\nu_2})\,R^{\rm row}_{\nu_2} \\ S^{\rm row}_{\nu_2} - A(\tilde{I}^{\rm row}_{\nu_2}, \tilde{I}^{\rm col}_{\nu_1})\,R^{\rm row}_{\nu_1} \end{bmatrix}$     $S^{\rm col}_{\rm loc} = \begin{bmatrix} S^{\rm col}_{\nu_1} - A(\tilde{I}^{\rm row}_{\nu_1}, \tilde{I}^{\rm col}_{\nu_2})\,R^{\rm col}_{\nu_2} \\ S^{\rm col}_{\nu_2} - A(\tilde{I}^{\rm row}_{\nu_2}, \tilde{I}^{\rm col}_{\nu_1})\,R^{\rm col}_{\nu_1} \end{bmatrix}$
        end if
        $[U^{\rm row}_\tau, J^{\rm row}_\tau] = \texttt{interpolate}((S^{\rm row}_{\rm loc})^*)$     $[U^{\rm col}_\tau, J^{\rm col}_\tau] = \texttt{interpolate}((S^{\rm col}_{\rm loc})^*)$
        $R^{\rm row}_\tau = (U^{\rm col}_\tau)^*\,R^{\rm row}_{\rm loc}$                 $R^{\rm col}_\tau = (U^{\rm row}_\tau)^*\,R^{\rm col}_{\rm loc}$
        $S^{\rm row}_\tau = S^{\rm row}_{\rm loc}(J^{\rm row}_\tau, :)$                   $S^{\rm col}_\tau = S^{\rm col}_{\rm loc}(J^{\rm col}_\tau, :)$
        $\tilde{I}^{\rm row}_\tau = I^{\rm row}_{\rm loc}(J^{\rm row}_\tau)$                     $\tilde{I}^{\rm col}_\tau = I^{\rm col}_{\rm loc}(J^{\rm col}_\tau)$
    end loop
end loop
For all leaf nodes $\tau$, set $D_\tau = A(I_\tau, I_\tau)$.
For all sibling pairs $\{\nu_1, \nu_2\}$, set $B_{\nu_1,\nu_2} = A(\tilde{I}^{\rm row}_{\nu_1}, \tilde{I}^{\rm col}_{\nu_2})$.
```

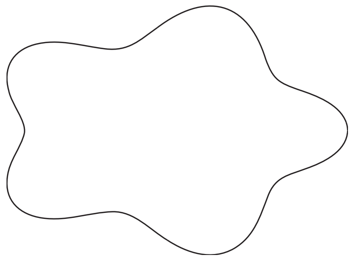


FIG. 5.1. *The contour* Γ *used in* (5.1).

x, resulting in a symmetric coefficient matrix. The double layer operator was discretized using the plain trapezoidal rule, which in the present case has superalgebraic convergence since the integrand is smooth.

Table 1

Time in seconds required by our implementation of the FMM to apply a matrix of size $N \times N$ to N_{vec} vectors simultaneously. The FMM uses multipole expansions of length 40*, leading to about* 15 *accurate digits.*

	$N = 800$	$N = 1\,600$	$N = 3\,200$	$N = 6\,400$	$N = 12\,800$	$N = 25\,600$
$N_{\text{vec}} = 1$	1.328	1.891	2.875	4.531	7.343	13.266
$N_{\text{vec}} = 50$	1.500	2.266	3.578	5.969	10.531	19.375
$N_{\text{vec}} = 100$	1.656	2.563	4.110	7.062	12.844	23.891

Table 2

Results from experiments with Algorithms 2 *and* 4 *when applied to the operator* (5.1) *with the kernels* (5.2) *and* (5.3)*. N is problem size, q is the number of columns in the random matrix Ω, t_{comp} is the compression time in seconds, and e_1 is the error, as defined by* (5.4)*.*

	Single layer				Double layer			
	$q = 50$		$q = 100$		$q = 50$		$q = 100$	
N	t_{comp}	e_1	t_{comp}	e_1	t_{comp}	e_1	t_{comp}	e_1
400	0.047	7.0e-13	0.094	2.2e-15	0.078	2.8e-13	0.172	3.6e-15
800	0.109	1.5e-11	0.219	8.3e-15	0.172	6.9e-13	0.390	1.0e-14
1600	0.235	3.3e-10	0.484	1.7e-14	0.343	5.7e-13	0.828	2.3e-14
3200	0.453	7.9e-10	1.000	5.2e-14	0.688	1.2e-12	1.719	4.2e-14
6400	0.906	7.0e-9	2.015	5.4e-14	1.422	4.0e-12	3.484	9.6e-14
12800	1.828	8.7e-9	4.031	1.0e-13	2.844	7.8e-12	7.046	4.7e-13
25600	3.765	5.8e-8	8.234	5.2e-13	5.719	1.1e-11	14.125	1.5e-12

5.2. The fast multipole method. In the numerical experiments reported here, discrete approximations to the operator (5.1) were applied via the classical FMM of Greengard and Rokhlin [16] with multipole expansions of order 40 to ensure close to double precision accuracy. The cost required for applying an approximation of a matrix A of size $N \times N$ to N_{vec} vectors simultaneously is reported in Table 1. We note that the cost of applying A to $N_{\text{vec}} = 50$ vectors is only slightly higher than the cost of applying A to a single vector. This illustrates a principal advantage of randomized sampling methods over iterative methods (such as, e.g., Krylov), namely that the matrix-vector multiplications can be executed in parallel rather than consecutively.

5.3. Fixed-rank experiments. In our first experiment, we executed Algorithms 2 and 4 as stated, with a fixed preset number of sample vectors q of either 50 or 100. Algorithm 2 was applied to discretizations of the operator (5.1) with the single layer kernel (5.2). For a sequence of problem sizes N we measured the time (wall time) t_{comp} required to compress the matrix A via Algorithm 2, with the time for computing the sample matrix $S = A\,\Omega$ via the FMM excluded (these times are reported separately in Table 1). Once the compressed matrix A_{approx} had been constructed, we computed the error measure

$$e_1 = \frac{||A - A_{\text{approx}}||}{||A||} \tag{5.4}$$

via 20 steps of a power iteration with a random starting vector. Table 2 lists t_{comp} and e_1 for a variety of problem sizes N, and for the two levels of accuracy $q = 50$ and $q = 100$. Table 2 also gives the values of t_{comp} and e_1 for an analogous set of experiments in which Algorithm 4 was applied to the operator (5.1) with the double layer kernel (5.3). Some observations:

Table 3

Results from experiments with Algorithm 2 *applied to the operator* (5.1) *with the kernel* (5.2). *N is problem size, ϵ is the requested accuracy, and q is the number of columns in the sample matrices. t_{comp}, t_{inv}, and t_{apply} are the computational times in seconds (as specified in section* 5.4*). The errors reported are $e_1 = ||A - A_{\text{approx}}||/||A||$ and $e_2 = ||I - A\,G||$, where G is the computed approximation to A^{-1}.*

$\epsilon = 10^{-5}$, $q = 50$							
N	t_{comp}	t_{inv}	t_{apply}	$\|\|A_{\text{approx}}\|\|$	$\|\|G\|\|$	e_1	e_2
400	0.047	0.031	0.000	1.23	6.4e3	5.1e-6	2.9e-3
800	0.078	0.063	0.000	0.77	1.4e4	5.2e-6	2.4e-3
1600	0.140	0.141	0.016	0.57	1.6e5	1.1e-5	2.0e-2
3200	0.297	0.297	0.031	0.57	2.3e5	5.8e-6	1.2e-2
6400	0.625	0.625	0.062	0.57	1.1e6	2.9e-6	1.4e-2
12800	1.281	1.328	0.141	0.57	4.2e6	3.5e-6	8.0e-2
25600	2.625	2.875	0.265	0.57	5.6e6	6.5e-6	1.2e-1
$\epsilon = 10^{-10}$, $q = 100$							
N	t_{comp}	t_{inv}	t_{apply}	$\|\|A_{\text{approx}}\|\|$	$\|\|G\|\|$	e_1	e_2
400	0.047	0.047	0.000	1.24	6.4e3	3.3e-11	1.5e-8
800	0.109	0.094	0.000	0.75	1.4e4	4.3e-11	2.0e-8
1600	0.203	0.203	0.032	0.57	1.6e5	4.3e-11	1.2e-7
3200	0.422	0.406	0.031	0.57	2.3e5	4.3e-11	1.2e-5
6400	0.843	0.844	0.078	0.57	1.1e6	4.4e-11	4.6e-5
12800	1.687	1.703	0.141	0.57	4.2e6	3.3e-11	2.2e-4
25600	3.407	3.547	0.266	0.57	5.6e6	2.6e-11	2.0e-5

(1) The absolute times required for compression are small; e.g., a nonsymmetric matrix of size $25\,600 \times 25\,600$ is compressed to ten digits of accuracy in less that 14 seconds.
(2) The claim that Algorithms 2 and 4 have linear complexity is supported.
(3) The time required by Algorithms 2 and 4 is comparable to the cost of applying the matrix A via the FMM to a single vector.
(4) For the double layer potential, the error e_1 grows only very slowly with problem size.
(5) For the single layer potential, the error e_1 grows substantially with problem size. A principal cause of this growth is that, in this case, the singular kernel causes the HSS-ranks required for any fixed accuracy to grow with problem size.

5.4. Adaptive rank determination. In view of the error growth described in observations (4) and (5) above, we also implemented the modified version of Algorithms 2 and 4 described in Remark 4.1. The interpolatory decomposition is now executed to produce factorizations that satisfy a preset tolerance ϵ_{loc}. For the double layer kernel, in which all HSS-blocks have roughly the same rank, using a fixed tolerance $\epsilon_{\text{loc}} = \epsilon$ for all blocks worked very well. For the single layer potential, in which the rank of an HSS-block depends on which level it belongs to, we found that a global tolerance of roughly ϵ was obtained when we required blocks on level ℓ to be compressed to accuracy $\epsilon_\ell = \epsilon \cdot 10^{-0.5\ell}$. One set of experiments was executed with $\epsilon = 10^{-5}$ and $q = 50$ sample columns, and a second set was executed with $\epsilon = 10^{-10}$ and $q = 100$ sample columns. The time required for compression t_{comp}, and the resulting error e_1 (cf. (5.4)), are reported in Tables 3 and 4. (The tables provide additional performance metrics that will be explained in section 5.5.) Two observations:
(1) By allowing the local rank to vary, the approximation error can be kept constant across a broad range of problem sizes.

TABLE 4
Results from experiments with Algorithm 4 *applied to the operator* (5.1) *with the kernel* (5.3). *Notation as in Table* 3.

$\epsilon = 10^{-5}$, $q = 50$							
N	t_{comp}	t_{inv}	t_{apply}	$\|\|A_{\text{approx}}\|\|$	$\|\|G\|\|$	e_1	e_2
400	0.047	0.031	0.000	1.04	3.6	2.6e-6	5.5e-6
800	0.094	0.031	0.016	1.04	3.6	3.1e-6	6.5e-6
1600	0.219	0.094	0.000	1.04	3.5	2.9e-6	6.3e-6
3200	0.406	0.140	0.016	1.04	3.5	2.6e-6	5.4e-6
6400	0.844	0.297	0.031	1.04	3.5	3.4e-6	7.6e-6
12800	1.688	0.578	0.062	1.04	3.6	3.6e-6	7.8e-6
25600	3.344	1.156	0.141	1.04	3.3	3.4e-6	7.3e-6
$\epsilon = 10^{-10}$, $q = 100$							
N	t_{comp}	t_{inv}	t_{apply}	$\|\|A_{\text{approx}}\|\|$	$\|\|G\|\|$	e_1	e_2
400	0.093	0.032	0.000	1.04	3.6	2.1e-11	4.5e-11
800	0.156	0.079	0.000	1.04	3.6	2.0e-11	4.4e-11
1600	0.297	0.109	0.016	1.04	3.6	1.5e-11	3.1e-11
3200	0.579	0.203	0.015	1.04	3.4	1.9e-11	4.0e-11
6400	1.094	0.344	0.047	1.04	3.6	2.5e-11	5.2e-11
12800	2.141	0.687	0.078	1.04	3.6	2.0e-11	4.2e-11
25600	4.093	1.266	0.141	1.04	3.6	3.4e-11	7.1e-11

(2) The algorithm with variable rank (reported in Tables 3 and 4) is faster than that using fixed rank (reported in Table 2) at any comparable error.

Remark 5.1. The randomized compression algorithms that were employed in section 5.3, as well as the experiments involving the double layer kernel in section 5.4, were all executed "blindly" in the sense that no problem-specific knowledge was used beyond a rough upper estimate of the numerical ranks of the off-diagonal blocks ($q = 50$ is enough for intermediate accuracy, and $q = 100$ is enough for high accuracy). The experiment involving the single layer kernel in section 5.4 is slightly different in that it involved an informed choice of the tolerance to be used in the rank-revealing QR factorizations. To obtain a blind method for a situation such as this, one can first apply the fixed-rank approach used in section 5.3, and then follow up with a round of rank reduction [37] in the computed HSS representation.

5.5. Inversion of the computed HSS representations. A motivation for computing an HSS representation of a given matrix in a situation where a fast matrix-vector multiplier such as the FMM is already available is that the HSS structure admits a broader range of matrix operations to be performed. (As a general matter, it is not known whether the FMM structure can be rapidly inverted, although some intriguing preliminary results were reported in [34].) For instance, linear complexity methods for inverting or computing an LU factorization of an HSS matrix are known [8, 9, 12]. In this section, we illustrate how the compression scheme of this paper can be combined with the inversion scheme of [30] to produce a fast *direct* solver for an equation involving the integral operator (5.1). The particular method of [30] is exact up to round-off errors and works best when the ranks used in the HSS representation are close to the theoretically minimal ranks; we therefore applied it to the output of the adaptive methods described in section 5.4.

In reporting the error of the direct solver, we let G denote the output of applying the inversion scheme of [30] to the HSS representation computed by the randomized compression algorithm. In other words, G is a data-sparse approximation of A^{-1},

$$G \approx A^{-1}.$$

Since A^{-1} is not available we cannot readily compute the error $||A^{-1} - G||$. However, we can bound the relative error in the inverse via

$$\frac{||A^{-1} - G||}{||A^{-1}||} = \frac{||A^{-1}\left(I - A\,G\right)||}{||A^{-1}||} \leq ||I - A\,G||.$$

We therefore define the error measure

$$e_2 = ||I - A\,G|| \tag{5.5}$$

and observe that it can be computed via a power iteration. The error e_2 also provides a useful bound on the residuals incurred when solving $A\,x = b$. To see this, let x_{exact} denote the exact solution

$$x_{\text{exact}} = A^{-1}\,b, \tag{5.6}$$

and let x_{approx} denote the approximate solution constructed by the direct solver

$$x_{\text{approx}} = G\,b. \tag{5.7}$$

Then

$$||A\,x_{\text{exact}} - A\,x_{\text{approx}}|| = ||b - A\,G\,b|| \leq ||I - A\,G||\,||b|| = e_2\,||b||.$$

The errors e_1 and e_2 are reported in Tables 3 and 4. The tables also provide estimates of the quantities $||A_{\text{approx}}||$ and $||G||$ (which together give an indication of the condition number of A), as well as the times t_{inv} required by the inversion and t_{apply} required for applying the inverse to a vector. Some observations:

(1) The inversion takes about the same amount of time as the compression; both of these steps are significantly faster than the application of the matrix to a single vector via the FMM.
(2) Once the inverse has been computed, the cost t_{apply} to solve (5.6) via (5.7) is *far* smaller than the cost of applying A via the FMM.
(3) In the experiments involving the double layer kernel (reported in Table 4) the matrix A is well-conditioned, and the inversion step leads to almost no loss of accuracy.
(4) In the experiments involving the single layer kernel (reported in Table 3) we see significant loss of accuracy as the problem size increases. This loss of accuracy appears to be due to the ill-conditioning of the coefficient matrix, since e_2/e_1 grows at about the same rate as the ratio $||G||/||A_{\text{approx}}||$.

Remark 5.2. The approach taken here of solving a linear system by explicitly computing an approximation to the inverse of the coefficient matrix is slightly unorthodox, and since the matrix inversion step is not unconditionally stable, the approach is not intended as a general recommendation. To justify the use of explicit matrix inversion, we note simply that for the specific context of solving boundary integral equations of mathematical physics it has empirically been observed to perform well [30]. Moreover, any inverse computed can be reliably verified, since the error bound (5.5) is computable.

6. Concluding remarks.

This paper presents a randomized algorithm for computing a compressed representation of a given matrix A in the so-called *hierarchically semiseparable* (HSS) format. The proposed algorithm requires that two functions be provided:

(F1) A fast means of evaluating matrix-vector products $x \mapsto A\,x$ and $x \mapsto A^*\,x$.

(F2) A fast technique for computing individual entries of the matrix. Only the construction of $O(N)$ entries is required.

The point of the proposed method is that while many well-established techniques are available for rapidly computing the matrix-vector product in F1 above (e.g., the FMM [16], panel clustering [19], or the Barnes–Hut method [1]), much less is known about how to rapidly compute the HSS representation of a matrix. Such a representation allows a broad range of matrix operations (matrix inversion, matrix-matrix multiply, LU factorization, etc.) to be performed efficiently.

Numerical examples were presented that indicate that the execution times of the proposed algorithms scale linearly with problem size, with a small constant of proportionality. Moreover, the numerical examples indicate that local errors do not significantly propagate, and that for requested accuracies between 10^{-5} and 10^{-10}, the computed compressed representation is accurate to within the requested accuracy.

Acknowledgments. The initial version of this manuscript [28] was written while the author was visiting the Institute for Pure and Applied Mathematics at UCLA during their long program *Mathematics of Knowledge and Search Engines* in the fall of 2007. The author gratefully acknowledges the many helpful suggestions made by the anonymous referees during the review process.

REFERENCES

[1] J. Barnes and P. Hut, *A hierarchical $o(n \log n)$ force-calculation algorithm*, Nature, 324 (1986), pp. 446–449.

[2] M. Bebendorf, *Approximation of boundary element matrices*, Numer. Math., 86 (2000), pp. 565–589.

[3] S. Börm, *$\mathcal{H}^2$-matrix arithmetics in linear complexity*, Computing, 77 (2006), pp. 1–28.

[4] S. Börm, *Construction of data-sparse $\mathcal{H}^2$-matrices by hierarchical compression*, SIAM J. Comput., 31 (2009), pp. 1820–1839.

[5] S. Börm, *Approximation of solution operators of elliptic partial differential equations by $\langle$ and $\langle^2$-matrices*, Numer. Math., 115 (2010), pp. 165–193.

[6] S. Börm and L. Grasedyck, *Hybrid cross approximation of integral operators*, Numer. Math., 101 (2005), pp. 221–249.

[7] T.F. Chan, *Rank revealing QR factorizations*, Linear Algebra Appl., 88–89 (1987), pp. 67–82.

[8] S. Chandrasekaran and M. Gu, *Fast and stable algorithms for banded plus semiseparable systems of linear equations*, SIAM J. Matrix Anal. Appl., 25 (2003), pp. 373–384.

[9] S. Chandrasekaran, M. Gu, X.S. Li, and J. Xia, *Some Fast Algorithms for Hierarchically Semiseparable Matrices*, CAM report 08-24, UCLA/CAM, Los Angeles, 2008.

[10] S. Chandrasekaran, M. Gu, and W. Lyons, *A fast adaptive solver for hierarchically semiseparable representations*, Calcolo, 42 (2005), pp. 171–185.

[11] H. Cheng, Z. Gimbutas, P.G. Martinsson, and V. Rokhlin, *On the compression of low rank matrices*, SIAM J. Sci. Comput., 26 (2005), pp. 1389–1404.

[12] P. Dewilde and S. Chandrasekaran, *A hierarchical semi-separable Moore-Penrose equation solver*, in Wavelets, Multiscale Systems and Hypercomplex Analysis, Oper. Theory Adv. Appl. 167, Birkhäuser, Basel, 2006, pp. 69–85.

[13] P.M. DeWilde and A.J. van der Veen, *Time-varying Systems and Computations*, Kluwer Academic Publishers, Dordrecht, The Netherlands, 1998.

[14] K. Frederix and M. Van Barel, *Solving a large dense linear system by adaptive cross approximation*, J. Comput. Appl. Math., 234 (2010), pp. 3181–3195.

[15] G.H. Golub and C.F. Van Loan, *Matrix Computations*, 3rd ed., Johns Hopkins Stud. Math. Sci., Johns Hopkins University Press, Baltimore, MD, 1996.

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

[17] L. Greengard and V. Rokhlin, *A new version of the fast multipole method for the Laplace equation in three dimensions*, in Acta Numer. 6, Cambridge University Press, Cambridge, UK, 1997, pp. 229–269.

[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] W. Hackbusch, *The panel clustering technique for the boundary element method (invited contribution)*, in Boundary elements IX, Vol. 1 (Stuttgart, 1987), Comput. Mech., Southampton, 1987, pp. 463–474.

[20] W. Hackbusch and S. Börm, *Data-sparse approximation by adaptive $\mathcal{H}^2$-matrices*, Computing, 69 (2002), pp. 1–35.

[21] W. Hackbusch and Z. P. Nowak, *On the fast matrix multiplication in the boundary element method by panel clustering*, Numer. Math., 54 (1989), pp. 463–491.

[22] W. Hackbusch, B. Khoromskij, and S. Sauter, *On $\mathcal{H}^2$-matrices*, in Lectures on Applied Mathematics, H.-J. Bungartz, R.H.W. Hoppe, C. Zenger, eds., Springer, Berlin, 2000, pp. 9–29.

[23] 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.

[24] D. Huybrechs, *Multiscale and Hybrid Methods for the Solution of Oscillatory Integral Equations*, Ph.D. thesis, Engineering Science, Katholieke Universiteit Leuven, Leuven, Belgium, 2006.

[25] S. Kapur and V. Rokhlin, *High-order corrected trapezoidal quadrature rules for singular functions*, SIAM J. Numer. Anal., 34 (1997), pp. 1331–1356.

[26] 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.

[27] L. Lin, J. Lu, and L. Ying, *Fast Construction of Hierarchical Matrix Representation from Matrix-Vector Multiplication*, preprint, 2010.

[28] P.G. Martinsson, *A Fast Algorithm for Compressing a Matrix into a Data-Sparse Format via Randomized Sampling*, Technical report 2008; online as arXiv.org report 0806.2339.

[29] P.G. Martinsson, *A fast direct solver for a class of elliptic partial differential equations*, J. Sci. Comput., 38 (2009), pp, 316–330.

[30] P.G. Martinsson and V. Rokhlin, *A fast direct solver for boundary integral equations in two dimensions*, J. Comput. Phys., 205 (2005), pp. 1–23.

[31] P.G. Martinsson and V. Rokhlin, *An accelerated kernel-independent fast multipole method in one dimension*, SIAM J. Sci. Comput., 29 (2007), pp. 1160–1178.

[32] P.G. Martinsson, V. Rokhlin, and M. Tygert, *A randomized algorithm for the decomposition of matrices*, Appl. Comput. Harmon. Anal., 30 (2011), pp. 47–68.

[33] E. Michielssen, A. Boag, and W.C. Chew, *Scattering from elongated objects: Direct solution in $O(N \log^2 N)$ operations*, IEE Proc. Microw. Antennas Propag., 143 (1996), pp. 277–283.

[34] T. Pals, *Multipole for Scattering Computations: Spectral Discretization, Stabilization, Fast Solvers*, Ph.D. thesis, Department of Electrical and Computer Engineering, University of California, Santa Barbara, 2004.

[35] Z. Sheng, P. Dewilde, and S. Chandrasekaran, *Algorithms to solve hierarchically semi-separable systems*, in System Theory, The Schur Algorithm and Multidimensional Analysis, Oper. Theory Adv. Appl. 176, Birkhäuser, Basel, 2007, pp. 255–294.

[36] P. Starr and V. Rokhlin, *On the numerical solution of two-point boundary value problems.* II, Comm. Pure Appl. Math., 47 (1994), pp. 1117–1159.

[37] J. Xia, *On the Complexity of Some Hierarchical Structured Matrix Algorithms*, 2010, in preparation.

[38] J. Xia, S. Chandrasekaran, M. Gu, and X.S. Li, *Superfast multifrontal method for large structured linear systems of equations*, SIAM J. Matrix Anal. Appl., 31 (2009), pp. 1382–1411.