Randomized algorithms for the low-rank approximation of matrices

Yale Dept. of Computer Science Technical Report 1388

Edo Liberty, Franco Woolfe, Per-Gunnar Martinsson, Vladimir Rokhlin, and Mark Tygert

We describe two recently proposed randomized algorithms for the construction of low-rank approximations to matrices, and demonstrate their application (*inter alia*) to the evaluation of the singular value decompositions of numerically low-rank matrices. Being probabilistic, the schemes described here have a finite probability of failure; in most cases, this probability is rather negligible (10^{-17} is a typical value). In many situations, the new procedures are considerably more efficient and reliable than the classical (deterministic) ones; they also parallelize naturally. We present several numerical examples to illustrate the performance of the schemes.

randomized | algorithm | low rank | matrix | SVD | PCA

Low-rank approximation of linear operators is ubiquitous in applied mathematics, scientific computing, numerical analysis, and a number of other areas. In this note, we restrict our attention to two classical forms of such approximations, the singular value decomposition (SVD) and the interpolative decomposition (ID). The definition and properties of the SVD are widely known; we refer the reader to reference [1] for a detailed description. The definition and properties of the ID are summarized in Subsection 1.1 below.

Below, we discuss two randomized algorithms for the construction of the IDs of matrices. Algorithm I is designed to be used in situations where the adjoint A^* of the $m \times n$ matrix A to be decomposed can be applied to arbitrary vectors in a "fast" manner, and has CPU time requirements typically proportional to $k \cdot C_{A^*} + k \cdot m + k^2 \cdot n$, where k is the rank of the approximating matrix, and C_{A^*} is the cost of applying A^* to a vector. Algorithm II is designed for arbitrary matrices, and its CPU time requirement is typically proportional to $m \cdot n \cdot \log(k) + k^2 \cdot n$. We also describe a scheme converting the ID of a matrix into its SVD for a cost proportional to $k^2 \cdot (m+n)$.

Space constraints preclude us from reviewing the extensive literature on the subject; for a detailed survey, we refer the reader to reference [2]. Throughout this note, we denote the adjoint of a matrix A by A^*, and the spectral (l^2-operator) norm of A by $\|A\|_2$; as is well known, $\|A\|_2$ is the greatest singular value of A. Furthermore, we assume that our matrices have complex entries (as opposed to real); real versions of the algorithms under discussion are quite similar to the complex ones.

This note has the following structure: Section 1 summarizes several known facts. Section 2 describes algorithms for the low-rank approximation of matrices. Section 3 illustrates the performance of the algorithms via several numerical examples. Section 4 contains conclusions, generalizations, and possible directions for future work.

Section 1: Preliminaries

In this section, we discuss two constructions from numerical analysis, to be used in the remainder of the note.

Subsection 1.1: Interpolative decompositions. In this subsection, we define interpolative decompositions (IDs) and summarize their properties.

The following lemma states that, for any $m \times n$ matrix A of rank k, there exist an $m \times k$ matrix B whose columns constitute a subset of the columns of A, and a $k \times n$ matrix P, such that

1. some subset of the columns of P makes up the $k \times k$ identity matrix,
2. P is not too large, and
3. $B_{m\times k} \cdot P_{k\times n} = A_{m\times n}$.

Moreover, the lemma provides an approximation

$$B_{m\times k} \cdot P_{k\times n} \approx A_{m\times n} \qquad \textbf{[1]}$$

when the exact rank of A is greater than k, but the $(k+1)^{\text{st}}$ greatest singular value of A is small. The lemma is a reformulation of Theorem 3.2 in reference [3] and Theorem 3 in reference [4]; its proof is based on techniques described in references [5], [6], and [7]. We will refer to the approximation in formula **1** as an interpolative decomposition (ID).

Lemma 1. *Suppose that m and n are positive integers, and A is a complex $m \times n$ matrix.*

Then, for any positive integer k with $k \le m$ and $k \le n$, there exist a complex $k \times n$ matrix P, and a complex $m \times k$ matrix B whose columns constitute a subset of the columns of A, such that

1. *some subset of the columns of P makes up the $k \times k$ identity matrix,*
2. *no entry of P has an absolute value greater than 1,*
3. $\|P_{k\times n}\|_2 \le \sqrt{k\,(n-k)+1}$,
4. *the least (that is, the k^{th} greatest) singular value of P is at least 1,*
5. $B_{m\times k} \cdot P_{k\times n} = A_{m\times n}$ *when $k = m$ or $k = n$, and*
6. *when $k < m$ and $k < n$,*
$$\|B_{m\times k} \cdot P_{k\times n} - A_{m\times n}\|_2 \le \sqrt{k\,(n-k)+1}\;\sigma_{k+1}, \qquad \textbf{[2]}$$
where σ_{k+1} is the $(k+1)^{st}$ greatest singular value of A.

Properties 1, 2, 3, and 4 in Lemma 1 ensure that the interpolative decomposition $B\,P$ of A is numerically stable. Also, property 3 follows directly from properties 1 and 2, and property 4 follows directly from property 1.

Observation 1. *Existing algorithms for the construction of the matrices B and P in Lemma 1 are computationally expensive (see Sections 3 and 4 of reference [5]). We use the algorithm of reference [4] to produce matrices B and P which satisfy somewhat weaker conditions than those in Lemma 1. We construct B and P such that*

Abbreviations: ID, interpolative decomposition; SVD, singular value decomposition; CPU, central processing unit; i.i.d., independent and identically distributed

1. *some subset of the columns of P makes up the $k \times k$ identity matrix,*

2. *no entry of P has an absolute value greater than 2,*

3. $\|P_{k\times n}\|_2 \le \sqrt{4k\,(n-k)+1}$,

4. *the least (that is, the k^{th} greatest) singular value of P is at least 1,*

5. $B_{m\times k} \cdot P_{k\times n} = A_{m\times n}$ *when $k = m$ or $k = n$, and*

6. *when $k < m$ and $k < n$,*

$$\|B_{m\times k} \cdot P_{k\times n} - A_{m\times n}\|_2 \le \sqrt{4k\,(n-k)+1}\;\sigma_{k+1}, \quad \textbf{[3]}$$

where σ_{k+1} is the $(k+1)^{st}$ greatest singular value of A.

For any positive real number ε, the algorithm can identify the least k such that $\|B\,P - A\|_2 \approx \varepsilon$. Furthermore, the algorithm computes both B and P using at most

$$C_{\mathrm{ID}} = \mathcal{O}(k \cdot m \cdot n \cdot \log(n)) \quad \textbf{[4]}$$

floating-point operations, typically requiring only

$$C'_{\mathrm{ID}} = \mathcal{O}(k \cdot m \cdot n). \quad \textbf{[5]}$$

Subsection 1.2: A class of random matrices. In this subsection, we describe a class Γ of structured random matrices that will be used in Section 2. A matrix Φ belongs to Γ if it consists of uniformly randomly selected rows of the product of the discrete Fourier transform matrix and a random diagonal matrix. Matrices of this type have been introduced in reference [8] in a slightly different context.

Suppose that l and m are positive integers, such that $l < m$. Suppose further that D is a complex diagonal $m \times m$ matrix whose diagonal entries $d_1, d_2, \ldots, d_m$ are independent and identically distributed (i.i.d.) complex random variables, each distributed uniformly over the unit circle. Suppose in addition that F is the complex $m \times m$ matrix defined by the formula

$$F_{j,k} = \exp(-2\pi i(j-1)(k-1)/m), \quad \textbf{[6]}$$

with $i = \sqrt{-1}$. Suppose moreover that S is a real $l \times m$ matrix whose entries are all zeros, aside from a single 1 in row j of column s_j for $j = 1, 2, \ldots, l$, where $s_1, s_2, \ldots, s_l$ are i.i.d. integer random variables, each distributed uniformly over $\{1, 2, \ldots, m\}$. Suppose finally that Φ is the $l \times m$ matrix defined by the formula

$$\Phi_{l\times m} = S_{l\times m} \cdot F_{m\times m} \cdot D_{m\times m}. \quad \textbf{[7]}$$

Then, the cost of applying Φ to an arbitrary complex vector via the algorithm of reference [9] is

$$C_{m,l} = \mathcal{O}(m \cdot \log(l)). \quad \textbf{[8]}$$

The algorithm of reference [9] is based on the fast Fourier transform, and is quite efficient in practical applications (in addition to having the attractive asymptotic CPU time estimate in formula **8**).

Remark 1. *The integers $s_1, s_2, \ldots, s_l$ used above in the construction of S from formula **7** are drawn uniformly with replacement from $\{1, 2, \ldots, m\}$. Our implementations indicate that the algorithms of this note yield similar accuracies whether $s_1, s_2, \ldots, s_l$ are drawn with or without replacement. However, drawing with replacement simplifies the analysis of these algorithms (see reference [10]).*

Section 2: Description of the algorithms

In this section, we describe two numerical schemes for approximating a matrix A with a low-rank matrix in the form of an ID. We also describe a procedure for converting an ID into an SVD.

The first scheme — Algorithm I — is meant to be used when efficient procedures for applying the matrix A and its adjoint A^* to arbitrary vectors are available; it is more reliable and parallelizable than the classical power and Lanczos algorithms, and is often more efficient. Algorithm I is described in Subsection 2.1 below.

The second scheme — Algorithm II — deals with the case when A is specified by its entire collection of entries; it is more efficient and parallelizable than the classical "$Q\,R$" decomposition, power, and Lanczos algorithms used in current numerical practice. Algorithm II is described in Subsection 2.2 below.

Subsection 2.4 describes an efficient procedure for calculating the SVD of A given an ID of A produced by either Algorithm I or Algorithm II.

Subsection 2.1: Algorithm I. In this subsection, we describe an algorithm for computing an approximation to a matrix A in the form of an ID. The algorithm is efficient when the matrix A^* can be applied rapidly to arbitrary vectors. A detailed analysis of the algorithm of this subsection can be found in reference [11].

Suppose that k, l, m, and n are positive integers with $k < l$, such that $l < m$ and $l < n$. Suppose further that A is a complex $m \times n$ matrix. Suppose in addition that R is a complex $l \times m$ matrix whose entries are i.i.d., each distributed as a complex Gaussian random variable of zero mean and unit variance.

Then, with very high probability whenever $l - k$ is sufficiently large, the following procedure constructs a complex $m \times k$ matrix B whose columns consist of a subset of the columns of A, and a complex $k \times n$ matrix P, such that some subset of the columns of P makes up the $k \times k$ identity matrix, no entry of P has an absolute value greater than 2, and

$$\|B_{m\times k} \cdot P_{k\times n} - A_{m\times n}\|_2 \lesssim \sigma_{k+1}, \quad \textbf{[9]}$$

where σ_{k+1} is the $(k+1)^{\mathrm{st}}$ greatest singular value of A. Thus, $B\,P$ is an ID approximating A.

The algorithm has three steps:

1. Form the product

$$Y_{l\times n} = R_{l\times m} \cdot A_{m\times n}. \quad \textbf{[10]}$$

2. Using the algorithm of Observation 1, collect k appropriately chosen columns of Y into a complex $l \times k$ matrix Z, and construct a complex $k \times n$ matrix P, such that some subset of the columns of P makes up the $k \times k$ identity matrix, no entry of P has an absolute value greater than 2, and

$$\|Z_{l\times k} \cdot P_{k\times n} - Y_{l\times n}\|_2 \lesssim \tau_{k+1}, \quad \textbf{[11]}$$

where τ_{k+1} is the $(k+1)^{\mathrm{st}}$ greatest singular value of Y.

3. Due to Step 2, the columns of Z constitute a subset of the columns of Y. In other words, there exists a finite sequence $i_1, i_2, \ldots, i_k$ of integers such that, for any $j = 1, 2, \ldots, k$, the j^{th} column of Z is the ${i_j}^{\mathrm{th}}$ column of Y. Collect the corresponding columns of A into a complex $m \times k$ matrix B, so that, for any $j = 1, 2, \ldots, k$, the j^{th} column of B is the ${i_j}^{\mathrm{th}}$ column of A.

If we define C_{A^*} to be the cost of applying A^* to a vector, then Step 1 costs $l \cdot C_{A^*}$ floating-point operations. Obviously, Step 2 costs $\mathcal{O}(k \cdot l \cdot n \cdot \log(n))$ operations (see Observation 1). Step 3 consists of moving $k \cdot m$ complex numbers from one location in memory to another, and hence costs $\mathcal{O}(k \cdot m)$ operations.

In all, the algorithm costs at most

$$C_{\mathrm{I}} = l \cdot C_{A^*} + \mathcal{O}(k \cdot m + k \cdot l \cdot n \cdot \log(n)) \qquad [12]$$

floating-point operations, and (as seen from formula **5**) typically costs

$$C_{\mathrm{I}}' = l \cdot C_{A^*} + \mathcal{O}(k \cdot m + k \cdot l \cdot n). \qquad [13]$$

Remark 2. *The estimate in formula* **13** *is based on the assumption that the entries of the matrix A are available at no cost. Sometimes, an algorithm for the application of A to arbitrary vectors is available, but one has no access to individual entries of A. In such cases, the k columns of A required by Algorithm I above can be obtained by applying A to an appropriately chosen collection of vectors, for a cost of $k \cdot C_A$, with C_A denoting the cost of applying A to a vector. Under these conditions, formula* **13** *becomes*

$$C_{\mathrm{I}}'' = l \cdot C_{A^*} + k \cdot C_A + \mathcal{O}(k \cdot l \cdot n). \qquad [14]$$

Remark 3. *Obviously, the algorithm of this subsection has a positive probability $P_{\mathrm{I}}^{\mathrm{fail}}$ of failure. The behavior of $P_{\mathrm{I}}^{\mathrm{fail}}$ as a function of $l - k$ is investigated in detail in reference [11], where the concept of failure for algorithms of this type is defined; reference [11] gives upper bounds on $P_{\mathrm{I}}^{\mathrm{fail}}$ which are complicated but quite satisfactory. For example, $l - k = 20$ results in $P_{\mathrm{I}}^{\mathrm{fail}} < 10^{-17}$, and $l - k = 8$ results in $P_{\mathrm{I}}^{\mathrm{fail}} < 10^{-5}$. Our numerical experience indicates that the bounds given in reference [11] are reasonably tight.*

Subsection 2.2: Algorithm II. In this subsection, we describe a modification of Algorithm I for computing an ID approximation to an arbitrary $m \times n$ matrix A; the modified algorithm does not need a "fast" scheme for the application of A and A^* to vectors. A detailed analysis of the algorithm of this subsection can be found in reference [10].

The modified algorithm, Algorithm II, is identical to Algorithm I of Subsection 2.1, but with the random matrix R used in formula **10** replaced with the matrix Φ defined in formula **7**. With this modification, the three-step algorithm of Subsection 2.1 remains a satisfactory numerical tool for the construction of IDs of matrices.

Now, Step 1 requires applying Φ to each column of A, for a total cost of $\mathcal{O}(n \cdot m \cdot \log(l))$ floating-point operations (see formula **8**). Steps 2 and 3 cost $\mathcal{O}(k \cdot l \cdot n \cdot \log(n))$ and $\mathcal{O}(k \cdot m)$ operations respectively, being identical to Steps 2 and 3 of Algorithm I.

In all, the modified algorithm costs at most

$$C_{\mathrm{II}} = \mathcal{O}(m \cdot n \cdot \log(l) + k \cdot l \cdot n \cdot \log(n)) \qquad [15]$$

floating-point operations, and (as seen from formula **5**) typically costs

$$C_{\mathrm{II}}' = \mathcal{O}(m \cdot n \cdot \log(l) + k \cdot l \cdot n). \qquad [16]$$

Remark 4. *As with Algorithm I of the preceding subsection, the algorithm of this subsection has a positive probability of failure, to be denoted by $P_{\mathrm{II}}^{\mathrm{fail}}$. The behavior of $P_{\mathrm{II}}^{\mathrm{fail}}$ as a function of k and l is investigated in reference [10], where the concept of failure for algorithms of this type is defined; the estimates in reference [10] are considerably weaker than the behavior observed experimentally. Specifically, reference [10] shows that $P_{\mathrm{II}}^{\mathrm{fail}}$ is less than $C \cdot k^2/l$, where C is a positive constant independent of the matrix A; in practice, the failure rate of Algorithm II is similar to that of Algorithm I. This discrepancy is a subject of intensive investigation.*

Subsection 2.3: Adaptive versions of Algorithms I and II. As described in the preceding two subsections, Algorithms I and II require a priori knowledge of the desired rank k of the approximations. This limitation is easily eliminated. Indeed, one can apply Algorithm I or II with k set to some more or less arbitrary number (such as 2, or 20, or 40), and then keep doubling k until the obtained approximation attains the desired precision. The algorithm described in the appendix provides an efficient means for estimating the precision of the obtained approximations. It is easy to see that knowing the desired rank k in advance reduces the cost of constructing the approximation by at most a factor of 2.

The authors have implemented such adaptive versions of both algorithms; in many cases the CPU time penalty is considerably less than a factor of 2. Moreover, much of the data can be reused from one value of k to another, further reducing the penalty. Such modifications are in progress.

Subsection 2.4: From ID to SVD. In this subsection, we describe a procedure for converting an interpolative decomposition (ID) into a singular value decomposition (SVD). A similar method enables the construction of the Schur decomposition of a matrix from its ID (see, for example, Theorem 7.1.3 and the surrounding discussion in reference [1] for a description of the Schur decomposition).

Suppose that k, m, and n are positive integers with $k \le m$ and $k \le n$, B is a complex $m \times k$ matrix, and P is a complex $k \times n$ matrix. Then, the following four steps compute a complex $m \times k$ matrix U whose columns are orthonormal, a complex $n \times k$ matrix V whose columns are orthonormal, and a real diagonal $k \times k$ matrix Σ whose entries are all nonnegative, such that

$$B_{m\times k} \cdot P_{k\times n} = U_{m\times k} \cdot \Sigma_{k\times k} \cdot (V^*)_{k\times n}. \qquad [17]$$

The four steps are:

1. Construct a "$Q\,R$" decomposition of P^*, *i.e.*, form a complex $n \times k$ matrix Q whose columns are orthonormal, and a complex upper-triangular $k \times k$ matrix R, such that

   $$(P^*)_{n\times k} = Q_{n\times k} \cdot R_{k\times k}, \qquad [18]$$

   for a cost of $\mathcal{O}(n \cdot k^2)$ (see, for example, Chapter 5 in reference [1]).

2. Form the complex $m \times k$ matrix S via the formula

   $$S_{m\times k} = B_{m\times k} \cdot (R^*)_{k\times k}, \qquad [19]$$

   for a cost of $\mathcal{O}(m \cdot k^2)$.

3. Form an SVD of S

   $$S_{m\times k} = U_{m\times k} \cdot \Sigma_{k\times k} \cdot (W^*)_{k\times k}, \qquad [20]$$

   where U is a complex $m \times k$ matrix whose columns are orthonormal, W is a complex $k \times k$ matrix whose columns are orthonormal, and Σ is a real diagonal $k \times k$ matrix whose entries are all nonnegative. The cost of this step is $\mathcal{O}(m \cdot k^2)$ (see, for example, Chapter 8 in reference [1]).

4. Form the complex $n \times k$ matrix V via the formula

   $$V_{n\times k} = Q_{n\times k} \cdot W_{k\times k}, \qquad [21]$$

   for a cost of $\mathcal{O}(n \cdot k^2)$.

The four steps above compute the SVD in formula **17** of the matrix $B\,P$ for a cost of

$$C_{\mathrm{ID}\Rightarrow\mathrm{SVD}} = \mathcal{O}((m+n)\cdot k^2). \qquad \textbf{[22]}$$

Combining the algorithm of this subsection and the algorithm of Subsection 2.2 yields an algorithm for computing an SVD approximation to an arbitrary matrix for a total cost of

$$C_{\mathrm{II}}^{\mathrm{SVD}} = \mathcal{O}(m\cdot n\cdot \log(l) + k\cdot l\cdot n\cdot \log(n) + (m+n)\cdot k^2). \qquad \textbf{[23]}$$

Similarly, combining the algorithm of this subsection and the algorithm of Subsection 2.1 yields an algorithm for computing an SVD approximation to a matrix A when both A and A^* can be applied rapidly to arbitrary vectors; however, in this case there exists a more direct algorithm yielding slightly better accuracy (see reference [11]).

Section 3: Numerical examples

In this section, we describe several numerical tests of the algorithms discussed in this note.

Subsections 3.1 and 3.2 illustrate the performance of Algorithms I and II, respectively. Subsection 3.3 illustrates the performance of the combination of Algorithm II with the four-step algorithm of Subsection 2.4.

The algorithms were implemented in Fortran 77 in double-precision arithmetic, compiled using the Lahey/Fujitsu Express v6.2 compiler with maximum optimization, and run on one core of a 1.86 GHz Intel Centrino Core Duo microprocessor with 2 MB of L2 cache and 1 GB of RAM. To perform the FFTs required by Algorithm II, we used a double-precision version of P. N. Swarztrauber's FFTPACK library.

Subsection 3.1: Algorithm I. This subsection reports results of applying Algorithm I to matrices A given by the formula

$$A = \frac{\Delta^{100}}{\|\Delta^{100}\|_2} + \frac{1}{\nu^2}\cdot c\cdot c^{\mathrm{T}}, \qquad \textbf{[24]}$$

where c is the $\nu^2 \times 1$ column vector whose entries are all ones, and Δ is the standard five-point discretization of the Laplacian on a $\nu\times\nu$ uniform grid; in other words, all of the diagonal entries of Δ are equal to -4, $\Delta_{p,q} = 1$ if the grid points p and q are neighbors, and all other entries of Δ are zeros (see, for example, Section 8.6.3 in reference [12]). Thus, A is an $n\times n$ matrix, with $n = \nu^2$.

The results of this set of tests are summarized in Table 1; the contents of the columns in Table 1 are as follows:

- n is the dimensionality of the $n\times n$ matrix A.
- k is the rank of the matrix approximating A.
- l is the first dimension of the $l\times m$ matrix R from formula **10**, with $m = n$.
- σ_{k+1} is the $(k+1)^{\mathrm{st}}$ greatest singular value of A, that is, the spectral norm of the difference between A and the best rank-k approximation to A.
- δ is the spectral norm of the difference between the original matrix A and its approximation obtained via the algorithm of Subsection 2.1.
- t is the CPU time (in seconds) taken both to compute the approximation to A via the algorithm of Subsection 2.1, and to check the accuracy of the approximation via the algorithm of the appendix.

The entries for δ in Table 1 display the maximum values encountered during 30 trials; the entries for t display the average values over 30 trials. Each of the trials was run with an independent realization of the matrix R in formula **10**.

Subsection 3.2: Algorithm II. This subsection reports results of applying Algorithm II to the 4,096 $\times$ 4,096 matrix A defined via the formula

$$A_{4096\times 4096} = U_{4096\times(k+20)}\cdot \Sigma_{(k+20)\times(k+20)}\cdot (V^*)_{(k+20)\times 4096}, \qquad \textbf{[25]}$$

with the matrices Σ, U, and V defined as follows.

The matrix U was constructed by applying the Gram-Schmidt process to the columns of a $4096\times(k+20)$ matrix whose entries were i.i.d. centered complex Gaussian random variables; the matrix V was obtained via an identical procedure. The matrix Σ is diagonal, with the diagonal entries $\Sigma_{j,j} = 10^{-15\cdot(j-1)/(k-1)}$ for $j = 1, 2, \ldots, k$, and $\Sigma_{j,j} = 10^{-15}$ for $j = k+1, k+2, \ldots, k+20$. Obviously, the j^{th} greatest singular value σ_j of A is $\Sigma_{j,j}$ for $j = 1, 2, \ldots, k+20$; the rest of the singular values of A are zeros.

For the direct algorithm, we used a pivoted "$Q\,R$" decomposition algorithm based upon plane (Householder) reflections, followed by the algorithm of reference [4].

The results of this set of tests are summarized in Table 2; the contents of the columns in Table 2 are as follows:

- k is the rank of the matrix approximating A.
- l is the first dimension of the $l\times m$ matrix Φ from formula **7**, with $m = 4{,}096$.
- σ_{k+1} is the $(k+1)^{\mathrm{st}}$ greatest singular value of A, that is, the spectral norm of the difference between A and the best rank-k approximation to A.
- δ_{direct} is the spectral norm of the difference between the original matrix A and its approximation obtained via the algorithm of reference [4], using a pivoted "$Q\,R$" decomposition algorithm based upon plane (Householder) reflections.
- δ is the spectral norm of the difference between the original matrix A and its approximation obtained via the algorithm of Subsection 2.2.
- t_{direct} is the CPU time (in seconds) taken to compute the approximation to A via the algorithm of reference [4], using a pivoted "$Q\,R$" decomposition algorithm based upon plane (Householder) reflections.
- t is the CPU time (in seconds) taken both to compute the approximation to A via the algorithm of Subsection 2.2, and to check the accuracy of the approximation via the algorithm of the appendix.
- t_{direct}/t is the factor by which the algorithm of Subsection 2.2 is faster than the classical algorithm that we used.

The entries for δ in Table 2 display the maximum values encountered during 30 trials; the entries for t display the average values over 30 trials. Each of the trials was run with an independent realization of the matrix Φ in formula **7**.

Subsection 3.3: SVD of an arbitrary matrix. This subsection reports results of applying the algorithm of Subsection 2.2 and then the algorithm of Subsection 2.4 to the 4,096 $\times$ 4,096 matrix A defined in formula **25**.

For the direct algorithm, we used a pivoted "$Q\,R$" decomposition algorithm based upon plane (Householder) reflections, followed by the divide-and-conquer SVD routine `dgesdd` from LAPACK 3.1.1.

The results of this set of tests are summarized in Table 3; the contents of the columns in Table 3 are as follows:

- k is the rank of the matrix approximating A.
- l is the first dimension of the $l \times m$ matrix Φ from formula **7**, with m = 4,096.
- σ_{k+1} is the $(k+1)^{\text{st}}$ greatest singular value of A, that is, the spectral norm of the difference between A and the best rank-k approximation to A.
- δ_{direct} is the spectral norm of the difference between the original matrix A and its approximation obtained via a pivoted "$Q\,R$" decomposition algorithm based upon plane (Householder) reflections, followed up with a call to the divide-and-conquer SVD routine `dgesdd` from LAPACK 3.1.1.
- δ is the spectral norm of the difference between the original matrix A and its approximation obtained via the algorithm of Subsection 2.2, followed by the algorithm of Subsection 2.4.
- t_{direct} is the CPU time (in seconds) taken to compute the approximation to A via a pivoted "$Q\,R$" decomposition algorithm based upon plane (Householder) reflections, followed by a call to the divide-and-conquer SVD routine `dgesdd` from LAPACK 3.1.1.
- t is the CPU time (in seconds) taken both to compute the approximation to A via the algorithm of Subsection 2.2, followed by the algorithm of Subsection 2.4, and to check the accuracy of the approximation via the algorithm of the appendix.
- t_{direct}/t is the factor by which the algorithm of Subsection 2.2, followed by the algorithm of Subsection 2.4, is faster than the classical algorithm that we used.

The entries for δ in Table 3 display the maximum values encountered during 30 trials; the entries for t display the average values over 30 trials. Each of the trials was run with an independent realization of the matrix Φ in formula **7**.

Subsection 3.4: Observations. The following observations can be made from the numerical experiments described in this section, and are consistent with the results of more extensive experimentation performed by the authors:

- The CPU times in Tables 1–3 are compatible with the estimates in formulae **14**, **16**, and **23**.
- The precision produced by each of Algorithms I and II is similar to that provided by formula **3**, even when σ_{k+1} is close to the machine precision.
- Algorithms I and II, as well as the classical pivoted "$Q\,R$" decomposition algorithms, all yield results of comparable accuracies.
- Algorithm II runs noticeably faster than the classical "dense" schemes, unless the rank of the approximation is nearly full, or is less than 6 or so.
- Algorithms I and II are remarkably insensitive to the quality of the pseudorandom number generators used.

Section 4: Conclusions

This note describes two classes of randomized algorithms for the compression of linear operators of limited rank, as well as applications of such techniques to the construction of singular value decompositions of matrices. Obviously, the algorithms of this note can be used for the construction of other matrix decompositions, such as the Schur decomposition. In many situations, the numerical procedures described in this note are faster than the classical ones, while ensuring comparable accuracy.

Whereas the results of numerical experiments are in reasonably close agreement with our estimates of the accuracy of Algorithm I, our numerical experiments indicate that Algorithm II performs better than our estimates guarantee. Comfortingly, the verification scheme of the appendix provides an inexpensive means for determining the precision of the approximation obtained during every run. If (contrary to our experience) Algorithm II were to produce an approximation that were less accurate than desired, then one could run the algorithm again with an independent realization of the random variables involved, in effect boosting the probability of success at a reasonable additional expected cost.

It should be pointed out that although Algorithm II cannot perform worse than our bounds guarantee, it actually performs much better in practice. In fact, Algorithm II yields accuracies similar to those of Algorithm I. This discrepancy is currently a subject of intensive investigation.

Furthermore, there is nothing magical about the random matrix defined in formula **7**. We have tested several classes of random matrices that are faster to apply, and that (in our numerical experience) perform at least as well in terms of accuracy. However, our theoretical bounds for the matrix defined in formula **7** are the strongest that we have obtained to date.

To summarize, the randomized algorithms described in this note are a viable alternative to classical tools for the compression and approximation of matrices; in many situations, the algorithms of this note are more efficient, reliable, and parallelizable. Their applications in several areas of scientific computing are under investigation.

Appendix: Estimating the spectral norm of a matrix

In this appendix, we describe a method for the estimation of the spectral norm of a matrix A. The method does not require access to the individual entries of A; it requires only applications of A and A^* to vectors. It is a version of the classical power method. Its probabilistic analysis summarized below was introduced fairly recently in references [13] and [14]. This appendix is included here for completeness.

Suppose that m and n are positive integers, and A is a complex $m \times n$ matrix. We define ω to be a complex $n \times 1$ column vector with independent and identically distributed entries, each distributed as a complex Gaussian random variable of zero mean and unit variance. We define $\tilde{\omega}$ to be the complex $n \times 1$ column vector $\tilde{\omega} = \omega / \|\omega\|_2$. For any integer $j > 1$, we define $p_j(A)$ by the formula

$$p_j(A) = \sqrt{\frac{\|(A^*\,A)^j\,\tilde{\omega}\|_2}{\|(A^*\,A)^{j-1}\,\tilde{\omega}\|_2}}, \qquad \textbf{[26]}$$

Table 1. Interpolative decomposition (via Algorithm I) of the $n \times n$ matrix A defined in formula 24

n	k	l	σ_{k+1}	δ	t
400	96	104	.504E–16	.380E–14	.53E0
1600	384	392	.513E–16	.974E–14	.91E1
3600	864	872	.647E–16	.181E–13	.58E2
6400	1536	1544	.649E–16	.289E–13	.24E3
400	48	56	.277E–08	.440E–07	.30E0
1600	192	200	.449E–08	.145E–06	.43E1
3600	432	440	.457E–08	.210E–06	.24E2
6400	768	776	.553E–08	.346E–06	.92E2
10000	1200	1208	.590E–08	.523E–06	.12E3

Table 2. Interpolative decomposition (via Algorithm II) of the 4,096 $\times$ 4,096 matrix A defined in formula 25

k	l	σ_{k+1}	δ_{direct}	δ	t_{direct}	t	t_{direct}/t
8	16	.100E–15	.359E–14	.249E–14	.31E1	.27E1	1.1
56	64	.100E–15	.423E–14	.369E–14	.18E2	.31E1	5.6
248	256	.100E–15	.309E–14	.147E–13	.77E2	.70E1	11
1016	1024	.100E–15	.407E–14	.571E–13	.32E3	.11E3	2.8

Table 3. Singular value decomposition (via Algorithm II) of the 4,096 $\times$ 4,096 matrix A defined in formula 25

k	l	σ_{k+1}	δ_{direct}	δ	t_{direct}	t	t_{direct}/t
8	16	.100E–15	.580E–14	.128E–13	.31E1	.22E1	1.4
56	64	.100E–15	.731E–14	.146E–13	.19E2	.34E1	5.6
248	256	.100E–15	.615E–14	.177E–13	.88E2	.19E2	4.6

which is the estimate of the spectral norm of A produced by j steps of the power method, starting with the vector $\tilde{\omega}$ (see, for example, reference [14]).

A somewhat involved analysis shows that the probability that

$$p_j(A) \geq \|A\|_2/10 \qquad [27]$$

is greater than $1 - 4\sqrt{n/(j-1)}\ 100^{-j}$. Needless to say, $p_j(A) \leq \|A\|_2$ for any positive j. Thus, even for fairly small j (we used $j = 6$ in this note), $p_j(A)$ estimates the value of $\|A\|_2$ to within a factor of ten, with very high probability.

This procedure is particularly useful for checking whether an algorithm (such as that described in Subsection 2.2) has produced a good approximation to a matrix, especially when we cannot afford to evaluate all of the entries in the difference between the matrix being approximated and its approximation. For more information, the reader is referred to references [13], [14], or Section 3.4 of [10].

The authors were partially supported by NSF Grant 0610097, NGA Grants HM1582-06-1-2039 and HM1582-06-1-2037, DARPA Grant HR0011-05-1-0002, Air Force STTR Grant F49620-03-C-0031, and AFOSR Grants FA9550-06-1-0197, FA9550-06-1-0239, and FA9550-05-C-0064.

1. Golub GH, Van Loan CF (1996) *Matrix Computations* (Johns Hopkins University Press, Baltimore, Maryland), Third ed.
2. Sarlós T (2006) in *Proc FOCS 2006* (IEEE), pp 143–152.
3. Martinsson P-G, Rokhlin V, Tygert M (2006) *Comm Appl Math Comput Sci* 1:133–142.
4. Cheng H, Gimbutas Z, Martinsson P-G, Rokhlin V (2005) *SIAM J Sci Comput* 26:1389–1404.
5. Gu M, Eisenstat SC (1996) *SIAM J Sci Comput* 17:848–869.
6. Tyrtyshnikov EE (2000) *Computing* 64:367–380.
7. Goreinov SA, Tyrtyshnikov EE (2001) in *Structured Matrices in Mathematics, Computer Science, and Engineering I*, ed Olshevsky V (AMS Publications, Providence, RI), Vol 280, pp 47–52.
8. Ailon N, Chazelle B (2006) in *Proceedings of the Thirty-Eighth Annual ACM Symposium on the Theory of Computing* (ACM Press, New York, NY), pp 557–563.
9. Sorensen HV, Burrus CS (1993) *IEEE Trans Signal Process* 41:1184–1200.
10. Woolfe F, Liberty E, Rokhlin V, Tygert M (2007) *A fast randomized algorithm for the approximation of matrices* (Yale Dept CS), Tech Rep 1386.
11. Martinsson P-G, Rokhlin V, Tygert M (2006) *A randomized algorithm for the approximation of matrices* (Yale Dept CS), Tech Rep 1361.
12. Dahlquist G, Björck Å (1974) *Numerical Methods* (Dover Publications, Mineola, New York).
13. Dixon JD (1983) *SIAM J Numer Anal* 20:812–814.
14. Kuczyński J, Woźniakowski H (1992) *SIAM J Matrix Anal Appl* 13:1094–1122.