SUBSET SELECTION ALGORITHMS: RANDOMIZED VS. DETERMINISTIC

MARY E. BROADBENT*, MARTIN BROWN†, AND KEVIN PENNER‡

Advisors: Ilse Ipsen[1] and Rizwana Rehman[2]

Abstract. Subset selection is a method for selecting a subset of columns from a real matrix, so that the subset represents the entire matrix well and is far from being rank deficient.

We begin by extending a deterministic subset selection algorithm to matrices that have more columns than rows. Then we investigate a two-stage subset selection algorithm that utilizes a randomized stage to pick a smaller number of candidate columns, which are forwarded for to the deterministic stage for subset selection. We perform extensive numerical experiments to compare the accuracy of this algorithm with the best known deterministic algorithm. We also introduce an iterative algorithm that systematically determines the number of candidate columns picked in the randomized stage, and we provide a recommendation for a specific value.

Motivated by our experimental results, we propose a new two stage deterministic algorithm for subset selection. In our numerical experiments, this new algorithm appears to be as accurate as the best deterministic algorithm, but it is faster, and it is also easier to implement than the randomized algorithm.

Key words. singular value decomposition, randomized algorithm, least squares, QR factorization, rank-revealing algorithm

AMS subject classification. 15A18, 15A23, 68W20, 15A42, 65F20, 65F40

1. Introduction.

Given a real $m \times n$ matrix A and an integer k, subset selection attempts to find the k most linearly independent columns that best represent the information in the matrix.

1.1. Example.

In an effort to make the problem concrete, we look at the following simple matrix:

$$A = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & \epsilon \end{pmatrix},$$

where $0 < \epsilon \ll 1$ is small but nonzero. We want to choose $k = 2$ representative columns of A. Which two columns should we choose?

First, suppose we choose the two leading columns of A as the representative columns. We call them A_1, and we call the remaining column A_2; that is,

$$A_1 = \begin{pmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{pmatrix}, \qquad A_2 = \begin{pmatrix} 0 \\ 0 \\ \epsilon \end{pmatrix}.$$

Then the columns of A_1 are linearly independent, and the best linear combination of A_1 is close to A_2, because $\min_z \|A_1 z - A_2\|_2 = \epsilon$ is small.

*Department of Mathematics, Amherst College, Amherst, MA 01002, USA (mbroadbent10@amherst.edu)

†Department of Mathematics, University of California, Berkeley, Berkeley, CA 94720, USA (martinebrown@berkeley.edu)

‡Departments of Mathematics, University of Pittsburgh, Pittsburgh, PA 15213, USA (klp37@pitt.edu)

[1]Department of Mathematics, North Carolina State University, Raleigh, NC 27695-8205, USA (ipsen@ncsu.edu, http://www4.ncsu.edu/~ipsen/)

[2]Department of Mathematics, North Carolina State University, Raleigh, NC 27695-8205, USA (rrehman@unity.ncsu.edu)

Instead, suppose we choose columns one and three of A as the representative columns. We call them A_1, and we call the remaining column A_2; that is,

$$A_1 = \begin{pmatrix} 1 & 0 \\ 0 & 0 \\ 0 & \epsilon \end{pmatrix}, \qquad A_2 = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}.$$

Then the columns of A_1 are still linearly independent, but not as linearly independent as in our first choice. Because ϵ is small, the second column of A_1 is close to the zero vector, so the columns of A_1 are close to being linearly dependent. Also, the best linear combination of A_1 does not come close to A_2 because $\min_z \|A_1 z - A_2\|_2 = 1$. Therefore our first choice of columns is more linearly independent and represents the matrix well.

Gauging linear independence can be made precise by using singular values, which we introduce in section 2.1. Using singular values, we move towards a mathematically precise definition of subset selection. We then present systematic algorithms for subset selection, which are based on QR decompositions, which we describe in section 2.4.

We use a permutation matrix Π that moves the representative columns to the left and all the other columns to the right. Then the problem of subset selection is to find a permutation Π so that

$$A\Pi = \begin{pmatrix} A_1 & A_2 \end{pmatrix}$$

where A_1 contains the k most linearly independent columns in A, and where the best linear combination of A_1 is close to A_2 so that $\min_z \|A_1 z - A_2\|_2$ is small.

1.2. Development of Subset Selection. Subset selection is used to identify the most important columns in a matrix. This is important, for instance, in solving rank-deficient least squares problems [10, Section 12.2]. Subset selection is also used in information retrieval [1, 5]. There the matrices can be so large that there is not enough memory to work with the whole matrix. In these cases, one needs to identify a smaller part of the matrix that represents the whole matrix well. Additional applications can be found in genetics [3] and wireless communication [17].

Many subset selection algorithms use a QR decomposition to find the most representative columns. One such QR decomposition was first introduced in 1965 by Golub and Businger [8, 9], [10, Section 5.4.1]. Since then, many other algorithms have been proposed. In 1994, Chandrasekaran and Ipsen [4] gave a comprehensive analysis of existing algorithms and proposed several hybrid algorithms. In 1996, Gu and Eisenstat [12, Algorithm 4] presented an algorithm that is more accurate than the hybrid algorithms in [4].

The theoretical computer science community has come up with randomized algorithms that use probability distributions to find the most representative columns in a matrix. Some discussion of these algorithms can be found in [13, 15, 7]. One of the recent randomized algorithms is a two-stage algorithm by Boutsidis, Mahoney and Drineas [1, 2]. In the first stage the algorithm uses a probability distribution to select candidates for the most representative columns. In the second stage the algorithm performs subset selection on the smaller candidate set rather than on the whole matrix.

1.3. Contributions. The purpose of our paper is to compare the randomized algorithm [1, Algorithm 1] to Gu and Eisenstat's deterministic algorithm [12, Algorithm 4]. We are interested primarily in the accuracy of the randomized algorithm, although runtime issues are briefly discussed.

We perform experiments on five different classes of test matrices of dimension up to 500 (section 4.1). We find that if the randomized algorithm is run repeatedly, then the columns A_1 it selects are

generally less linearly independent than those from the Gu-Eisenstat algorithm, while the residuals $\min_z \|A_1 z - A_2\|_2$ from the randomized algorithm can be smaller than those produced by algorithm [12, Algorithm 4] presented by Gu and Eisenstat (section 4.3) .

Our experiments indicate that the complicated probability distributions in the randomized algorithm can be unstable (section 4.2), and that they can be replaced by simpler, more intuitive distributions without sacrificing performance (section 4.5).

We have implemented two approaches to remedy the failure rate of the randomized algorithm when it chooses fewer than the desired number of columns (sections 3.2.2 and 4.4).

Finally, we propose a new two-stage deterministic algorithm that performs comparably to the Gu-Eisenstat algorithm [12] on our test matrices, but runs faster (section 5).

The paper proceeds as follows: Section 2 provides mathematical background information for the rest of the paper. Section 3 describes the algorithms from [12] and [1] that we compare in this paper. We present the results from experiments on our test matrices in section 4, and discuss our new two-stage deterministic algorithm in section 5.

2. Mathematical Background. In this section, we present the most important tools for subset selection: singular values and matrix norms for formulating the conditions for subset selection, and QR decompositions for making the conditions easy to check.

We assume A is a real $m \times n$ matrix with rank r, and we want to choose $k \leq r$ columns to represent A. For simplicity, we will assume that $m \geq n$. This means we need to find a permutation Π so that $A\Pi = \begin{pmatrix} A_1 & A_2 \end{pmatrix}$, where the representative columns are in A_1 and the remaining columns in A_2.

2.1. Singular values. We begin with the singular value decomposition (SVD). The SVD of A is defined as [10, Section 2.5.3]

$$A = U\Sigma V^T.$$

Here U is an $m \times m$ orthogonal matrix. This means that $U^T U = UU^T = I_m$, where the superscript T denotes the transpose and I_m is the $m \times m$ identity matrix. The matrix V is a $n \times n$ orthogonal matrix, so $V^T V = VV^T = I_n$. Σ is a $m \times n$ diagonal matrix with $\min\{m, n\}$ non-negative diagonal elements called $\sigma_i(A)$. These are the singular values of A. The singular values are ordered in decreasing magnitude,

$$\sigma_1(A) \geq \cdots \geq \sigma_r(A) > \sigma_{r+1}(A) = \cdots = \sigma_{\min\{m,n\}}(A) = 0. \tag{2.1}$$

For the purposes of subset selection, the most important property of singular values is that the distance of a matrix A to the set of matrices of rank k is equal to $\sigma_{k+1}(A)$ [10, Theorem 2.5.3]. We say that the matrix A has full rank if $\sigma_{\min\{m,n\}}(A) \neq 0$. If $\sigma_{\min\{m,n\}}(A) = 0$ then A is rank-deficient, i.e. $\text{rank}(A) < min\{m, n\}$.

The concept of rank-deficiency is important in subset selection. If a matrix is rank-deficient, then its columns are linearly dependent. Since singular values give us a way to check if a matrix is rank-deficient or close to rank-deficient, they help us gauge the linear independence of a set of columns. We are looking for a subset of columns of A that best represent all the columns of A. For expository purposes, we call the remaining columns "redundant".

2.2. Matrix Norms. In connection with singular values, we quickly review two important matrix norms: the 2-norm and the Frobenius norm. The 2-norm of a matrix is defined as the largest

singular value of that matrix, i.e. $\|A\|_2 = \sigma_1(A)$. The Frobenius norm of a matrix A is

$$\|A\|_F = \sqrt{\sum_{i=1}^{\min\{m,n\}} \sigma_i(A)^2} = \sqrt{\sum_{i=1}^{m}\sum_{j=1}^{n} |a_{ij}|^2}.$$

These definitions allow us to look at both the 2-norm and Frobenius norm in terms of singular values.

2.3. Desirable Conditions for Subset Selection. With singular values and norms in hand, we can now formulate our conditions for subset selection:

1. The k representative columns A_1 should be far from rank deficient, that is, $\sigma_k(A_1)$ should be as large as possible.
2. The best linear combination of A_1 should be close to A_2, that is, $\min_z \|A_1 z - A_2\|_2$ should be as small as possible.

Since the smallest singular value of a set of columns tells us how far the set is from linear dependence, Condition 1 merely states that we want the k representative columns A_1 to be very linearly independent.

Condition 2 relates to minimizing the residual between the representative columns A_1 and the redundant columns A_2. This means our subset has captured a large amount of information in the matrix.

2.4. QR Decompositions. Many algorithms for subset selection are based on QR decompositions, because a QR decomposition gives us an upper triangular matrix R that has the same singular values as A and has a determinant that is easy to calculate.

We discuss the QR decomposition here for matrices where $m \geq n$. The case where $m < n$ is analogous and discussed in section 3.1.2. The general form of a QR decomposition of A with column pivoting is:

$$A\Pi = Q \begin{pmatrix} R \\ 0 \end{pmatrix},$$

where Π is the $n \times n$ column permutation matrix, Q is an $m \times m$ orthogonal matrix, and R is an $n \times n$ upper triangular matrix. The zero matrix below R is of dimension $(m-n) \times n$. Note that applying a column permutation matrix Π' to $A\Pi$ is equivalent to applying it to R since $A\Pi = QR \iff A\Pi\Pi' = QR\Pi'$.

We asserted above that R and A have the same singular values. To see this, take the SVD of R, $R = W\Sigma V^T$. Since $A\Pi = QR$ and Q is orthogonal, $A = (QW)\Sigma(\Pi V)^T$ is the SVD of A, with the same singular values as R.

Since $A\Pi$ and R have the same singular values, and since the i^{th} column of $A\Pi$ corresponds to the i^{th} column of R, we are able to reframe our subset selection in terms of R. Instead of selecting a subset of A, we simplify the problem by performing subset selection on R. To distinguish the representative columns, we partition

$$R = \begin{pmatrix} R_k & B_k \\ 0 & C_k \end{pmatrix},$$

where R_k is a $k \times k$ upper triangular matrix, B_k is $k \times (n-k)$ and C_k is $(n-k) \times (n-k)$. If the column permutation Π is used to reveal the rank of A, as in the algorithm by Golub and Businger

[10, Section 5.4.1], then this permutation Π gives us a nonsingular R_k. The representative and the redundant columns in the two matrices are related by

$$A_1 = Q \begin{pmatrix} R_k \\ 0 \\ 0 \end{pmatrix}, \qquad A_2 = Q \begin{pmatrix} B_k \\ C_k \\ 0 \end{pmatrix}.$$

The advantage of the upper triangular form of R is that it is efficient to compute the important quantities for subset selection:

$$\sigma_i(A_1) = \sigma_i(R_k), \qquad \min_z \|A_1 z - A_2\|_2 = \sigma_1(C_k). \tag{2.2}$$

The first equality says that the tall and skinny matrix A_1 has the same singular values as the small square matrix R_k. The second equality tells us that minimizing the residual between the chosen columns and the redundant columns is equivalent to making the two-norm of C_k small. We prove these two inequalities in Lemma 7.1 in the Appendix (Section 7).

Now we can rephrase the subset selection conditions in section 2.3 in terms of submatrices of R, which makes the conditions easier to check.

Condition 1: The smallest singular value $\sigma_k(R_k)$ should be as large as possible.

Condition 2: The largest singular value $\sigma_1(C_k)$ should be as small as possible.

However, $\sigma_k(R_k)$ cannot be arbitrarily large, and $\sigma_1(C_k)$ cannot be arbitrarily small. This is because the interlacing property of singular values implies that there are bounds on the singular values of principal submatrices of R (R_k and C_k). The relevant inequalities are [10, Corollary 8.6.3]

$$\sigma_i(R_k) \le \sigma_i(R), \qquad 1 \le i \le k, \tag{2.3}$$

and

$$\sigma_j(C_k) \ge \sigma_{k+j}(R), \qquad 1 \le j \le n-k. \tag{2.4}$$

3. The Algorithms. We present two algorithms for subset selection: first a deterministic algorithm, and then a randomized algorithm.

3.1. The Deterministic Algorithm. Gu and Eisenstat [12, Algorithm 4] developed a QR decomposition for subset selection that represents the best deterministic approximation to Conditions 1 and 2 in section 2.4. In Lemma 3.1 (see section 3.1.2) we show that this algorithm can be extended to matrices that are "wide and fat" with $n > m$.

Given k and a parameter $f > 1$, the representative columns chosen in Gu and Eisenstat's algorithm approximate the subset Conditions 1 and 2 by guaranteeing the following bounds for the singular values of R_k and C_k:

$$\begin{aligned} \sigma_i(R_k) &\ge \frac{\sigma_i(A)}{\sqrt{1 + f^2 k(n-k)}}, \qquad 1 \le i \le k \\ \sigma_j(C_k) &\le \sigma_{k+j}(A)\sqrt{1 + f^2 k(n-k)}, \qquad 1 \le j \le n-k, \end{aligned} \tag{3.1}$$

where $f \ge 1$ is a tolerance supplied by the user. Gu and Eisenstat's algorithm also guarantees that $|(R_k^{-1} B_k)_{ij}| \le 1$ for $1 \le i \le k$, $1 \le j \le n-k$.

Gu and Eisenstat's algorithm implies the following for the representative and the redundant columns of the matrix A. Combining the bounds (3.1) with (2.2) implies that Gu and Eisenstat's algorithm produces a permutation Π so that $A\Pi = \begin{pmatrix} A_1 & A_2 \end{pmatrix}$, where

$$\sigma_i(A_1) \geq \frac{\sigma_i(A)}{\sqrt{1 + f^2 k(n-k)}}, \qquad 1 \leq i \leq k \tag{3.2}$$
$$\min_z \|A_1 z - A_2\|_2 \leq \sigma_{k+1}(A)\sqrt{1 + f^2 k(n-k)}.$$

3.1.1. The Algorithm. We work with Gu and Eisenstat's Algorithm 4 [12]. Our version is presented as Algorithm 1 below.

First we use Golub and Businger's QR decomposition with column pivoting [10, section 5.4.1] to produce an initial QR decomposition $A\Pi = Q \begin{pmatrix} R \\ 0 \end{pmatrix}$ where R is guaranteed to have a submatrix R_k that is nonsingular. From now on the algorithm operates only on the matrix R and proceeds the same way as [12, Algorithm 4]. We permute columns of R with the goal of increasing the singular values of R_k. Based on the facts

$$|\det A| = \prod_{i=1}^{n} \sigma_i(A), \quad \sigma_i(A) = \sigma_i(R) \quad \text{and} \quad |\det A| = |\det R| = |\det(R_k)\det(C_k)|,$$

we use the change in $|\det(R_k)|$ as a way to decide whether column i in R_k and column j in $\begin{pmatrix} B_k \\ C_k \end{pmatrix}$ should be permuted. Observe that $|\det(A)|$ is constant since the magnitude of the determinant is unchanged by column permutations, so a permutation that increases $|\det(R_k)|$ must necessarily decrease $|\det(C_k)|$. When $|\det(R_k)|$ increases and $|\det(C_k)|$ decreases, we cannot conclude that all singular values of R_k have increased nor that all singular values of C_k have decreased. However, we can conclude that at least one singular value of R_k must have increased and at least one singular value of C_k must have decreased. These permutations move the selected columns toward satisfying Conditions 1 and 2 since we want the singular values of R_k to be as large as possible and the singular values of C_k to be as small as possible.

Gu and Eisenstat [12, Lemma 3.1] derive a useful expression that can detect an increase in the determinant before actually performing a permutation. To see this, suppose we actually permuted columns i and j of R to get $\tilde{R}$, that is, $\tilde{R} = R\Pi_{ij}$, where Π_{ij} is the identity matrix with columns i and j permuted. Retriangularizing $\tilde{R}$ can be done with a QR decomposition

$$\tilde{R} = Q_{ij} \begin{pmatrix} \tilde{R}_k & \tilde{B}_k \\ 0 & \tilde{C}_k \end{pmatrix},$$

where Q_{ij} is a $n \times n$ orthogonal matrix and $\tilde{R}_k$ is $k \times k$ upper triangular. Gu and Eisenstat [12, Lemma 3.1] show that the ratio of the two determinants can be expressed as

$$\left| \frac{\det \tilde{R}_k}{\det R_k} \right| = \sqrt{({R_k}^{-1} B_k)_{i,j}^2 + \|C_k e_j\|_2^2 \|e_i^T R_k^{-1}\|_2^2}, \tag{3.3}$$

where e_i is the ith column of the identity matrix, so that $C_k e_j$ is the jth column of C_k and $e_i^T R_k^{-1}$ is the ith row of R_k^{-1}. This means, we can test by how much $|\det \tilde{R}_k|$ would increase compared to

$|\det R_k|$ by just looking at submatrices of R, without having to do a permutation. We only permute columns i and j of R if the righthand side of equation (3.3) is greater than f. The algorithm is run until $|\det \tilde{R}_k|/|\det R_k| \leq f$. If $f = 1$ then we permute columns as long as the determinant increases in magnitude.

Algorithm 1 Deterministic Algorithm for Subset Selection

Input: $m \times n$ matrix A with $m \geq n$, integer $k \leq r$, tolerance $f \geq 1$
Output: Permutation Π so that $A\Pi = \begin{pmatrix} A_1 & A_2 \end{pmatrix}$ where A_1 and A_2 satisfy the bounds (3.2).

Compute a QR decomposition with column pivoting on A to produce an initial permutation Π and an initial $R = \begin{pmatrix} R_k & B_k \\ 0 & C_k \end{pmatrix}$ with R_k nonsingular
while there exist i, j such that $\sqrt{({R_k}^{-1}B_k)_{i,j}^2 + \|C_k e_j\|_2^2 \|e_i^T R_k^{-1}\|_2^2} > f$ **do**
    Permute columns i and $j+k$ of R: $\tilde{R} = R\Pi_{i,j+k}$
    Retriangularize: $\tilde{R} = Q_{i,j+k}R$
    Update: $\Pi = \Pi\,\Pi_{i,j+k}$
end while

3.1.2. An Extension. We can extend [12, Theorem 3.2] to prove that Algorithm 1 computes a strong RRQR factorization for wide and fat matrices where $m < n$ and $k = m$.

LEMMA 3.1. *If A is $m \times n$ of rank m and $k = m$ then Algorithm 1 produces a permutation Π so that*

$$\sigma_i(A_1) \geq \frac{\sigma_i(A)}{\sqrt{1 + f^2 k(n-k)}}, \qquad 1 \leq i \leq k.$$

Proof. This is a simplified version of the proof of [12, Theorem 3.2], where C_k is absent.

The initial QR decomposition with column pivoting in Algorithm 1 yields $A = QR$, and in Lemma 7.1 we showed that $\sigma_i(A) = \sigma_i(R)$, $1 \leq i \leq k$. Since A has rank $k = m$ by assumption, $R = \begin{pmatrix} R_k & B_k \end{pmatrix}$ with R_k nonsingular. We factor out R_k^{-1} to get $R = R_k W$, where $W = \begin{pmatrix} I_k & R_k^{-1}B_k \end{pmatrix}$, and apply the product inequality for singular values to get $\sigma_i(A) \leq \sigma_i(R_k)\|W\|_2$. Then

$$\|W\|_2^2 = \|WW^T\|_2 = \|I + R_k^{-1}B_k B_k^T R_k^{-T}\|_2 \leq 1 + \|R_k^{-1}B_k\|_2^2.$$

The proof that $\|R_k^{-1}B_k\|_2 \leq \sqrt{1 + f^2 k(n-k)}$ now follows as in [12, Theorem 3.2]. ∎

3.2. The Randomized Algorithm. Boutsidis, *et al.* [1] present a randomized subset selection algorithm that can achieve better bounds than the deterministic algorithm in section 3.1. The randomized algorithm works in two stages, one randomized and one deterministic. Given that we want to ultimately select k of n columns, the randomized stage chooses $\tilde{c}$ candidate columns. We want $k \leq \tilde{c} < n$ for the next stage. The deterministic stage then applies a deterministic subset selection algorithm to the $\tilde{c}$ candidate columns to obtain the desired k representative columns.

In terms of accuracy, the randomized algorithm [1] aims to satisfy subset selection Condition 2 with a better bound than Algorithm 1. According to [1, Theorem 1], the randomized algorithm produces a permutation Π so that with at least 70 percent probability $A\Pi = \begin{pmatrix} A_1 & A_2 \end{pmatrix}$ where

$$\min_z \|A_1 z - A_2\|_2 \leq \mathcal{O}\left(k^{3/4}(\log k)^{1/2}(n-k)^{1/4}\right)\, \sigma_{k+1}(A). \tag{3.4}$$

3.2.1. The Algorithm. The algorithm works on rows of the right singular vector matrix of A. If we look at the SVD $A = U\Sigma V^T$ from section 2.1, we see that permuting columns of A results in permuting columns of the right singular vector matrix V^T. Moreover selecting particular columns from A amounts to selecting the corresponding columns in V^T.

If we look at the r rows of ΣV^T corresponding to nonzero singular values, we can write them as

$$\begin{pmatrix} \Sigma_k & 0 \\ 0 & \Sigma_{r-k} \end{pmatrix} \begin{pmatrix} V_k^T \\ V_{r-k}^T \end{pmatrix} = \begin{pmatrix} \Sigma_k V_k^T \\ \Sigma_{r-k} V_{r-k}^T \end{pmatrix}.$$

Here we use our convention that we ordered the singular values in decreasing magnitude, see (2.1), and group

$$\Sigma_k = \begin{pmatrix} \sigma_1(A) & & \\ & \ddots & \\ & & \sigma_k(A) \end{pmatrix}, \qquad \Sigma_{r-k} = \begin{pmatrix} \sigma_{k+1}(A) & & \\ & \ddots & \\ & & \sigma_r(A) \end{pmatrix},$$

so that Σ_k contains the k largest singular values, and Σ_{r-k} contains the $r-k$ smallest nonzero singular values. The corresponding singular vectors are

$$V_k = \begin{pmatrix} v_1 & \dots & v_k \end{pmatrix}, \qquad V_{r-k} = \begin{pmatrix} v_{k+1} & \dots & v_r \end{pmatrix},$$

so that the singular vectors in V_k are associated with the k largest singular values.

Now we present our view of the two-stage algorithm.

Randomized Stage. First we form a $n \times n$ diagonal matrix D with diagonal elements $D_{ii} = 1/\sqrt{\min\{1, cp_i\}}$. Here c is an input parameter, which should be a multiple of $k \log k$, and the values p_i represent a probability distribution which we describe further down. We use the diagonal matrix to scale the right singular vector matrix V_k^T, that is, $W = V_k^T D$. Now we pick column i of W with probability $\min\{1, cp_i\}$. The parameter c is an upper bound on the expected value of $\tilde{c}$. Every time we check a column, we are essentially performing a Bernoulli trial with probability of success $\min\{1, cp_i\}$. The trials are independent, so $E(\tilde{c}) = \sum_{i=1}^n \min\{1, cp_i\}$. Since $\sum_{i=1}^n cp_i = c$, $E(\tilde{c}) \le c$. After we examine every column of W and either select or reject it, we end up with a subset of $\tilde{c}$ candidate columns of W, ending the randomized stage. We call the matrix of candidate columns W_R. If Π_R is the permutation that moves the candidate columns in W_R to the left of $V_k^T D$, we can write

$$(V_k^T D)\,\Pi_R = W\Pi_R = \begin{pmatrix} W_R & \hat{W}_R \end{pmatrix}.$$

Deterministic Stage. We begin the deterministic stage with the $\tilde{c}$ columns W_R, hoping that $\tilde{c} \ge k$. We use a deterministic algorithm to select k representative columns from W_R. We call these k columns W_D. That is, the deterministic stage produces a permutation Π_D that moves the representative columns W_D to the left of W_R,

$$W_R\Pi_D = \begin{pmatrix} W_D & \hat{W}_D \end{pmatrix}.$$

Combining the permutations from the randomized and deterministic stages gives

$$(V_k^T D)\,\Pi = W\Pi = \begin{pmatrix} W_D & \hat{W}_D & \hat{W}_R \end{pmatrix},$$

where

$$\Pi = \Pi_R \begin{pmatrix} \Pi_D & 0 \\ 0 & I_{n-\tilde{c}} \end{pmatrix}.$$

Note that the permutation Π_R is $n \times n$, because it works on n columns, while the permutation Π_D is only $\tilde{c} \times \tilde{c}$, because it works only on the leading $\tilde{c}$ columns of $W\Pi_R$. The permutation Π is the one we want. It moves the representative columns of A to the left, $A\Pi = \begin{pmatrix} A_1 & A_2 \end{pmatrix}$.

Probability Distribution. We still need to describe the probabilities p_i. They are computed from nonzero singular values and right singular vectors as follows:

$$p_i = \frac{\|e_i^T V_k\|_2^2}{2\sum_{j=1}^n \|e_j V_k\|_2^2} + \frac{\|(\Sigma_{r-k} V_{r-k}^T) e_i\|_2^2}{2\sum_{j=1}^n \|\Sigma_{r-k} V_{r-k}^T e_j\|_2^2} \tag{3.5}$$

$$= \frac{\|e_i^T V_k\|_2^2}{2k} + \frac{\|Ae_i\|_2^2 - \|AV_k V_k^T e_i\|_2^2}{2(\|A\|_F^2 - \|AV_k V_k^T\|_F^2)}. \tag{3.6}$$

Note that $p_i \geq 0$ and $\sum_{i=1}^n p_i = 1$. Although the distributions are mathematically equivalent, the computation of (3.6) is cheaper. However, we note that in section 4.2 that numerical instability can force the use of (3.5).

The algorithm is presented as Algorithm 2 below. Boutsidis, *et al.* [1, p.16] suggest running the algorithm 40 times so that at least one run achieves the bound (3.4) with probability $1 - 10^{-20}$.

Algorithm 2 Randomized Algorithm for Subset Selection

Input: $m \times n$ matrix A of rank r, integer $k \leq r$, parameter $c = \mathcal{O}(k \log k)$
Output: Permutation Π so that $A\Pi = \begin{pmatrix} A_1 & A_2 \end{pmatrix}$ where A_1 and A_2 try to satisfy the bound (3.4)

Compute the r nonzero singular values of A, and the associated right singular vectors
Compute p_i from (3.6)
{Randomized Stage}
Form the $n \times n$ diagonal matrix D with elements $D_{ii} = 1/\sqrt{\min\{1, cp_i\}}$
Initialize: $\Pi = I_n$, $W = V_k^T D$
$\tilde{c} = 0$
for $i = 1 : n$ **do**
    Draw a q_i from the uniform distribution on $[0, 1]$
    if $q_i \leq cp_i$ **then**
        $\tilde{c} = \tilde{c} + 1$
        Permute columns i and j of W: $W = W\Pi_{ij}$
        Update: $\Pi = \Pi\,\Pi_{ij}$
    end if
end for
Set W_R equal to the leading $\tilde{c}$ columns of W
{Deterministic Stage}
Apply Algorithm 1 to W_R to return the permutation Π_D
Combine the permutations: $\Pi = \Pi_R \begin{pmatrix} \Pi_D & 0 \\ 0 & I_{n-\tilde{c}} \end{pmatrix}$

3.2.2. Our Implementation. In order to implement the probabilities in the first stage, we generate a random number q_i from the uniform distribution on $[0, 1]$. If $q_i \leq cp_i$, which happens with probability $\min\{1, cp_i\}$, the column i of $V_k^T D$ is selected and moved to the left. If $q_i > cp_i$ we don't do anything and do not include column i among the candidate columns.

Since Boutsidis, *et al.* [1, p. 20] give little guidance on what the crucial parameter c should be, in our experiments we begin with $c = 2k$. If the randomized stage of Algorithm 2 produces candidate columns W_R that are too close to being rank deficient, i.e. $\sigma_k(W_R) < 1/2$, then the residual bound (3.4) may not hold [1, Lemma 1]. Thus, we double c and repeat the process until either $\sigma_k(W_R) \geq 1/2$ or else $c \geq n$. We stop once this doubling results in $c \geq n$, because $E(\tilde{c}) \leq c$ and we want $E(\tilde{c}) < n$. This iterative selection of c was proposed by Stan Eisenstat [6]. This systematic selection process has the advantage of relieving the user from having to decide on a value for c, a decision for which the user may have not have the requisite information, and, in particular, of preventing the user from making an unfortunate decision that results in lower accuracy.

For the deterministic stage we use Algorithm 1 to find k representative columns in the $k \times \tilde{c}$ matrix W_R. The proof that the resulting A_1 satisfies (3.1) is given in Lemma 3.1 in the Appendix (section 7). In contrast, Boutsidis, *et al.* [1] use an algorithm based on a rank revealing LU decomposition by Pan [16].

Since the randomized algorithm does not produce the same results in each of the 40 trials, we keep track of the minimum, maximum and mean residuals and singular values of the subsets selected.

Algorithm 3 Randomized Algorithm Iteratively Selecting c

Input: $m \times n$ matrix A of rank r, integer $k \leq r$
Output: Permutation Π so that $A\Pi = \begin{pmatrix} A_1 & A_2 \end{pmatrix}$ where A_1 and A_2 satisfy the bound (3.4)

```
Compute the r nonzero singular values of A, and the associated right singular vectors
Compute p_i from (3.6)
c = k
while σ_k(W_R) < 1/2 and c ≤ n do
        c = 2c
        Run the Randomized Stage in Algorithm 2
end while
{Now σ_k(W_R) ≥ 1/2 or c > n}
Proceed with the Deterministic Stage of Algorithm 2
```

3.2.3. Comments and Concerns. Boutsidis, *et al.* [1] use the complicated probability distribution (3.5) for their algorithm. This distribution is needed to prove [1, Lemma 2] and [1, Lemma 4]. This distribution is hard for us to digest as well as computationally problematic (see section 4.5). In section 4.5 we use a simpler distribution that is more intuitive.

Some care must be put into selecting an appropriate value for c. Algorithm 2 fails if $\tilde{c} < k$, and if c is not large enough, the chance of failure can be non-negligible. However, if we allow c to be too large, then the randomized algorithm essentially reduces to performing the deterministic algorithm on V_k^T instead of A with the added computational cost of performing the SVD and computing the probability distribution.

In addition, Boutsidis, *et al.* do not say anything about the smallest singular value of the representative columns, $\sigma_k(A_1)$. Our experiments investigate how $\sigma_k(A_1)$ from Algorithm 2 compares

to that from the deterministic Algorithm 1.

4. Experimental Results. We performed experiments to answer the following questions about the randomized Algorithm 3:

- How does the randomized Algorithm 3 compare to the deterministic Algorithm 1 with regard to Conditions 1 and 2?
- Can we determine a good c to use?
- Do we need to use the complicated probability distributions (3.5) and (3.6), or can we instead use the simpler distribution $p_i = \|e_i^T V_k\|_2^2/k$ [1, (5)]?
- Can we develop a deterministic algorithm based on the randomized algorithm?

All algorithms were coded in MATLAB. Likewise, all experiments were run in MATLAB. For all algorithms in this section, the tolerance parameter f was set to 1.01. Running the Algorithm 3 40 times, as was necessary, at an optimal $f = 1$ would take exponential time, which is too long given available computing power.

4.1. The Test Matrices. We test the algorithms on five different classes of matrices.

1. Kahan matrices. These are upper triangular matrices, where all columns have a unit two-norm, but the matrices are close to being rank deficient. The Kahan matrices are well-known test matrices for subset selection algorithms, because many algorithms for rank-revealing QR with column pivoting fail to perform any permutations [12, Example 1].
2. Random matrices. These are dense matrices with elements sampled from a uniform distribution on $[0, 1]$. The matrices were created using the rand function in MATLAB.
3. Scaled random matrices. These are $n \times n$ random matrices (with elements sampled uniformly from $[0, 1]$) whose i^{th} row is multiplied by the scalar $\eta^{i/n}$. We used $\eta = 2$.
4. GKS matrices. These are upper triangular matrices. The jth diagonal element is equal to $1/\sqrt{j}$. The remaining nonzero elements in column j are equal to $-1/\sqrt{j}$. These matrices were introduced in [11].
5. SV gap matrices. These are square matrices designed to have a well-defined numerical rank r. They have r very large singular values (on the order of 10^5), and the rest are very small or zero. We use these matrices with $k = r$ to see if the randomized algorithm recognizes this and picks an appropriate set of columns.

Although subset selection is likely to be performed on less structured matrices, we chose these test matrices because they are the test matrices used in both [1] and [12]. We want to replicate their experiments, so we use approximately the same matrix dimensions as [1, 12] as well. For each matrix type, we test one of size 100×100 and one of size 500×500. Due to limits on our computing power, we were not able to replicate experiments on larger matrices. Furthermore, our laptops forced us to use $f = 1.01$ instead of the optimal value $f = 1$ whenever Algorithm 1 is used, including when it is used as a subroutine of a randomized algorithm. This makes the algorithm run faster since fewer column permutations are performed. However, the bounds (3.1) and (3.2) become worse, indicating the subset chosen could possibly be better.

Just as we test matrices similar to those in [1], we use k values similar to those in [1].

4.2. A Numerical Issue. An important problem we found with the randomized algorithms is that distribution (3.6) can be numerically unstable. We constructed a 100×100 SV gap matrix where we get negative values for probabilities and a distribution that does not sum to one. This numerical instability is likely caused by cancellation in the subtractions in (3.6).

If either the sum of all probabilities is not equal to one or the probability associated with a column is negative, then our code for Algorithm 2 or 3 switches to the alternative expression (3.5),

which is equivalent to (3.6) in exact arithmetic.

4.3. Results. We begin by presenting results from the 500×500 matrices, since they are representative of the results from the 100×100 matrices. Remember, we want large values of $\sigma_k(A_1)$ and small values of $\min_z \|A_1 z - A_2\|_2$.

In this section's experiments, we give the randomized algorithm all the chances it needs to satisfy conditions for good results by using Algorithm 3, which iteratively selects c instead of taking it as an input. We run Algorithm 3 forty times, and each run is allowed to choose a value of c.

In Table 4.1 we display $\sigma_k(A_1)$ for various values of k when A is the 500×500 Kahan matrix. When run 40 times, the randomized Algorithm 3 produces average values of $\sigma_k(A_1)$ that are comparable to those of the deterministic Algorithm 1. However, if we look at $\min \sigma_k(R_k)$ for $k = 100, 160, 180$ we see that some runs of the randomized Algorithm 3 can get much smaller values of $\sigma_k(A_1)$. This means, for the Kahan matrix, the randomized Algorithm 3 can produce representative columns that are much closer to being rank deficient than those produced by the deterministic Algorithm 1.

	Algorithm 1	Algorithm 3		
k	$\sigma_k(A_1)$	$\max \sigma_k(A_1)$	$\min \sigma_k(A_1)$	$\text{mean } \sigma_k(A_1)$
20	3×10^{-1}	3×10^{-1}	8×10^{-1}	2×10^{-1}
40	8×10^{-2}	7×10^{-2}	3×10^{-2}	5×10^{-2}
60	2×10^{-2}	1×10^{-2}	7×10^{-3}	1×10^{-2}
80	5×10^{-3}	4×10^{-3}	1×10^{-3}	3×10^{-3}
100	1×10^{-3}	9×10^{-4}	3×10^{-4}	7×10^{-4}
120	3×10^{-4}	2×10^{-4}	1×10^{-10}	2×10^{-4}
140	7×10^{-5}	7×10^{-5}	2×10^{-10}	4×10^{-5}
160	2×10^{-5}	2×10^{-5}	8×10^{-11}	1×10^{-5}
180	4×10^{-6}	4×10^{-6}	3×10^{-10}	2×10^{-6}
200	1×10^{-6}	8×10^{-7}	1×10^{-9}	6×10^{-7}
220	2×10^{-7}	2×10^{-7}	4×10^{-9}	1×10^{-7}
240	6×10^{-8}	6×10^{-8}	2×10^{-8}	4×10^{-8}

TABLE 4.1
$\sigma_k(A_1)$ for Algorithm 1 and 40 runs of Algorithm 3 on a 500×500 Kahan matrix A.

In Table 4.2 we display the residuals $\min_z \|A_1 z - A_2\|_2$ for various values of k when A is the 500×500 Kahan matrix. When run 40 times, the randomized Algorithm 3 produces average values of residuals that are an order of magnitude lower than those of Algorithm 1, but even the largest residuals can be smaller than those of the deterministic Algorithm 1. The residuals in Table 4.2 were collected from the same runs as in Table 4.1. This means, for the Kahan matrix, the randomized Algorithm 3 produces residuals that are as good, if not better, than those of the deterministic Algorithm 1. In Table 4.3 we show the values of c considered by Algorithm 3.

If we look at Tables 4.4 and 4.5, we see that Algorithm 3 performs comparably to Algorithm 1 on a wide variety of matrices. Although the singular values and residuals are comparable, the two algorithms usually do not choose the same columns for A_1. This is understandable, since matrices might not have a unique set of linearly independent columns. For example, there are $\binom{n}{k}$ sets of k linearly independent columns to choose from a $n \times n$ identity matrix.

Algorithm 3 gets better residual results on various matrices, but the results are not more than one order of magnitude better. However, it is evident in Table 4.5 that Algorithm 1 almost always

satisfies Condition 2 better than Algorithm 3. In the presence of random fluctuations and roundoff error, these slightly better residuals do not justify use of the randomized algorithm, especially given that there is no guidance on how to set the parameter c in [1]. In our implementation, the randomized Algorithm 3 has 40 trials, and in each trial it has the opportunity to try different c values. Algorithm 3 always goes through at least two c values, sometimes up to five. Although this set-up is computationally expensive, it might give us an intuition for how to pick one good value for c.

4.4. Estimating a Good c. One way we can improve the randomized Algorithm 2 is to find a single value for c that works for many matrices. This way, each of the 40 trials does not have to go through many values of c, easing the computational load.

The experiments in this section are meant to help us find a single c for the randomized Algorithm

	Algorithm 1	Algorithm 3		
k	residual	max residual	min residual	mean residual
20	7×10^{0}	1×10^{0}	5×10^{-1}	5×10^{-1}
40	2×10^{0}	1×10^{0}	1×10^{-1}	2×10^{-1}
60	4×10^{-1}	9×10^{-1}	3×10^{-2}	5×10^{-2}
80	1×10^{-1}	3×10^{-2}	7×10^{-3}	8×10^{-3}
100	2×10^{-2}	9×10^{-3}	2×10^{-3}	2×10^{-3}
120	6×10^{-3}	2×10^{-3}	4×10^{-4}	5×10^{-4}
140	1×10^{-3}	6×10^{-4}	1×10^{-4}	1×10^{-4}
160	3×10^{-4}	1×10^{-4}	2×10^{-5}	3×10^{-5}
180	8×10^{-5}	2×10^{-5}	6×10^{-6}	7×10^{-6}
200	2×10^{-5}	5×10^{-6}	1×10^{-6}	2×10^{-6}
220	4×10^{-6}	8×10^{-7}	4×10^{-7}	4×10^{-7}
240	1×10^{-6}	1×10^{-7}	9×10^{-8}	9×10^{-8}

TABLE 4.2

Residuals $\min_z \|A_1 z - A_2\|_2$ *for Algorithm 1 and 40 runs of Algorithm 3 on a* 500×500 *Kahan matrix* A.

k	# c values tried	c values tried	mean $\tilde{c}$
20	5	40, 80, 160, 320, 640	51
40	4	80, 160, 320, 640	89
60	4	120, 240, 480, 960	129
80	3	160, 320, 640	154
100	2	200, 400, 800	190
120	3	240, 480, 960	222
140	2	280, 560	242
160	1	640	264
180	1	720	255
200	2	400, 800	253
220	2	440, 880	267
240	2	480, 960	286

TABLE 4.3

The different c *values for the 40 runs of Algorithm 3 in Tables 4.1 and 4.2.*

	Algorithm 1	Algorithm 3		
Matrix Type	residual	max residual	min residual	mean residual
Kahan	7×10^0	6×10^{-1}	5×10^{-1}	5×10^{-1}
SV-Gap	6×10^0	1×10^1	7×10^0	1×10^1
GKS	3×10^0	2×10^0	1×10^0	1×10^0
Random	3×10^1	3×10^1	3×10^1	3×10^1
Scale Rand	4×10^1	5×10^1	4×10^1	5×10^1

TABLE 4.4

Residuals $\min_z \|A_1 z - A_2\|_2$ *from Algorithm 1 and 40 runs of Algorithm 3 on 5 different classes of* 500×500 *matrices A when* $k = 20$*. Here the SV Gap matrix has 20 large singular values.*

	Algorithm 1	Algorithm 3		
Matrix Type	$\sigma_k(A_1)$	max $\sigma_k(A_1)$	min $\sigma_k(A_1)$	mean$\sigma_k(A_1)$
Kahan	3×10^{-1}	2×10^{-1}	1×10^{-6}	2×10^{-1}
SV-Gap	1×10^4	2×10^4	5×10^3	1×10^4
GKS	4×10^{-1}	2×10^{-1}	2×10^{-1}	2×10^{-1}
Random	6×10^0	6×10^0	5×10^0	5×10^0
Scale Rand	8×10^0	8×10^0	7×10^0	8×10^0

TABLE 4.5

Smallest singular values $\sigma_k(A_1)$ *from Algorithm 1 and 40 runs of Algorithm 3 when* $k = 20$ *on five different classes of* 500×500 *matrices A. Here the SV Gap matrix has 20 large singular values.*

2. We do this by looking at how different c values affect the accuracy of the randomized Algorithm 2. If the results do not vary drastically for different values of c, we should be able find one. We already have some guidance on this problem. From Table 4.6, we see that $c = 4k$ appears to be chosen frequently for small values of k. Being chosen frequently indicates that for this c value the candidate columns are sufficiently linearly independent, i.e. $\sigma_k(W_R) \geq 1/2$ in Algorithm 3.

In the following experiments, we ran Algorithm 2 40 times with $c = 2k, 4k, 8k, \ldots, 2^i k$ until $c > n$. Judging by Figures 4.1 – 4.5, it looks like $c = 4k$ would be a good value for two reasons. The first reason is that the residuals $\min_z \|A_1 z - A_2\|_2$ do not vary much with different values of c. The second reason is that the number $\tilde{c}$ of candidate columns chosen in the randomized stage goes up as c goes up. We want $\tilde{c}$ to be as small as possible. This makes the efficiency gains as large as possible. Since $c = 4k$ gets small residuals and low values of $\tilde{c}$, we believe it would be a good c to use in general. We note that this result might not hold for all k, but it tends to work well for small values of k.

4.5. A Simpler Probability Distribution. Expression (3.6) for the probability distribution is numerically unstable, while expression (3.5) is computationally expensive. Boutsidis, *et al.* [1] mention another probability distribution [1, (5)] that bounds the Frobenius norm of the residual. This probability is

$$p_i = \|e_i^T V_k\|_2^2 / k. \tag{4.1}$$

This probability distribution (4.1) is much simpler and more efficient to compute. Below we compare the distributions (3.6) and (4.1) on our test matrices, with 40 runs of Algorithm 3 for each distribution. Once again, in the instances where (3.6) was numerically unstable, we used expression (3.5), which is equivalent in exact arithmetic.

Matrix	# c's tried	c values	most frequent c value	mean $\tilde{c}$
Kahan	5	$40, 80, 160, 320, 640$	$2k = 40$	50
SV-Gap	3	$40, 80, 160$	$4k = 80$	89
GKS	4	$40, 80, 160, 320$	$8k = 160$	132
Random	3	$40, 80, 160$	$4k = 80$	93
Scale Rand	3	$40, 80, 160$	$4k = 80$	96

TABLE 4.6
The different c values from runs of Algorithm 3 in Tables 4.4 and 4.5.

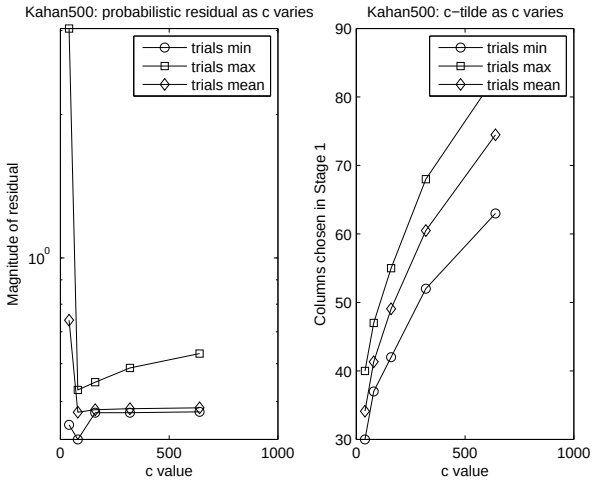

FIG. 4.1. *Different values of c for 40 runs of Algorithm 2 on the* 500×500 *Kahan matrix when* $k = 20$

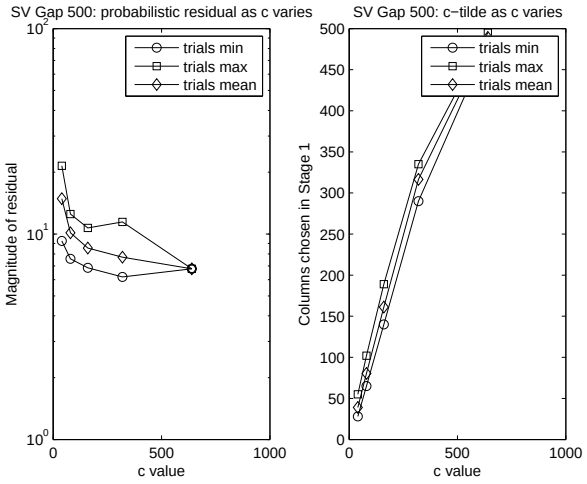

FIG. 4.2. *Different values of c for 40 runs of Algorithm 2 on the* 500×500 *SV gap matrix when* $k = 20$

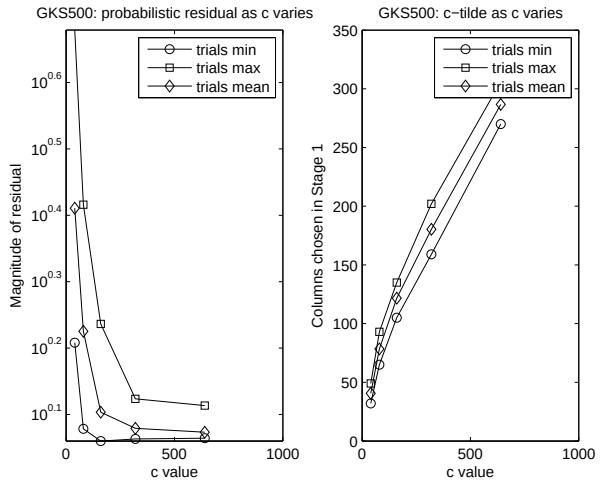

FIG. 4.3. *Different values of c for 40 runs of Algorithm 2 on the* 500×500 *GKS matrix when* $k = 20$

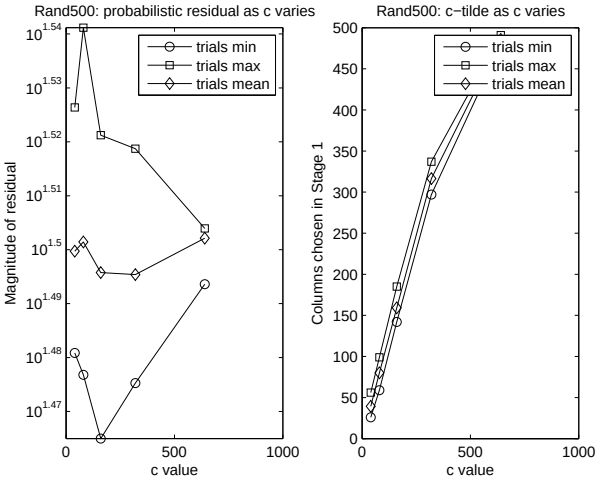

FIG. 4.4. *Different values of c for 40 runs of Algorithm 2 on a* 500×500 *random matrix when k=20*

In Table 4.7, we present the residuals $\min_z \|A_1 z - A_2\|_2$ for the five types of test matrices of order 500. These values are representative of the results for matrices of order 100. As in the earlier experiments, we record the maximum, minimum and average of the residuals. Table 4.7 illustrates that the magnitude of the residuals is essentially the same for both probability distributions. Therefore, the simpler probability distribution (4.1) seems to produce residuals that are as small as the ones produced by (3.6).

In Table 4.8, we present the smallest singular values of the representative columns, $\sigma_k(A_1)$, for the five types of test matrices of order 500. These results were extracted from the same runs as

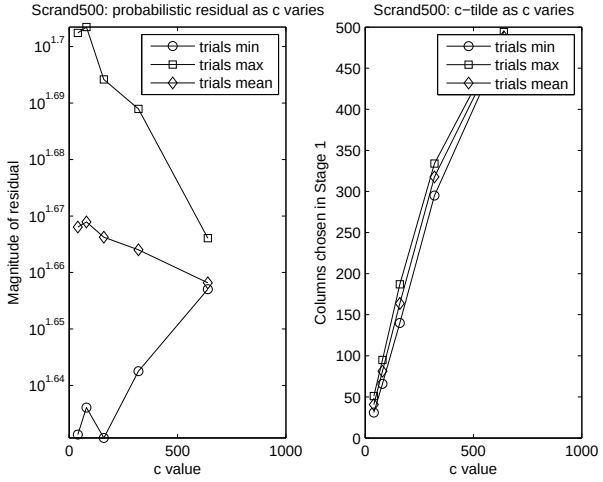

FIG. 4.5. *Different values of c for 40 runs of Algorithm 2 on a* 500×500 *scaled random matrix when* $k = 20$

Matrix Type	Distribution	max residual	min residual	mean residual	$\tilde{c}$
SV gap	Prob (3.5)	1×10^1	7×10^0	1×10^1	89
	Prob (4.1)	1×10^1	6×10^0	9×10^0	96
GKS	Prob (3.6)	2×10^0	1×10^0	1×10^0	132
	Prob (4.1)	1×10^0	1×10^0	1×10^0	98
Rand	Prob (3.6)	3×10^1	3×10^1	3×10^1	93
	Prob (4.1)	3×10^1	3×10^1	3×10^1	82
Scaled rand	Prob (3.6)	5×10^1	4×10^1	5×10^1	96
	Prob (4.1)	5×10^1	4×10^1	5×10^1	82
Kahan	Prob (3.6)	6×10^{-1}	5×10^{-1}	5×10^{-1}	50
	Prob (4.1)	6×10^{-1}	5×10^{-1}	5×10^{-1}	43

TABLE 4.7
Residuals $\min_z \|A_1 z - A_2\|_2$ *for the* 500×500 *matrices when* $k = 20$

those for the residuals above. Table 4.8 illustrates that the simpler probability distribution (4.1) yields representative columns that are as far from being rank deficient as the columns from (3.6).

According to the results of our experiments in Tables 4.7 and 4.8, the simpler probability distribution (4.1) performs as well as (3.6). Although the two distributions are associated with different theoretical bounds [1, Theorem 1], we see no empirical reason to support the use of (3.5) or (3.6).

4.6. Counter-Example to the Theoretical Bound of the Randomized Algorithm. Upon a suggestion by Stan Eisenstat [6], we performed experiments on a matrix of the form

$$\begin{pmatrix} I_k & 1_{k,n-k}/\sqrt{k+2} \\ 0_{m-k,k} & I_{m-k,n-k}/\sqrt{k+2} \end{pmatrix} \tag{4.2}$$

Matrix Type	Distribution	max $\sigma_k(A_1)$	min $\sigma_k(A_1)$	mean $\sigma_k(A_1)$
SV gap	Prob (3.5)	2×10^4	5×10^3	1×10^4
	Prob (4.1)	2×10^4	8×10^3	1×10^4
GKS	Prob (3.6)	2×10^{-1}	2×10^{-1}	2×10^{-1}
	Prob (4.1)	2×10^{-1}	2×10^{-1}	2×10^{-1}
Rand	Prob (3.6)	6×10^0	5×10^0	5×10^0
	Prob (4.1)	6×10^0	5×10^0	5×10^0
Scaled rand	Prob (3.6)	8×10^0	7×10^0	8×10^0
	Prob (4.1)	8×10^0	8×10^0	8×10^0
Kahan	Prob (3.6)	2×10^{-1}	1×10^{-6}	2×10^{-1}
	Prob (4.1)	2×10^{-1}	2×10^{-1}	2×10^{-1}

TABLE 4.8

Smallest singular values for the representative columns of the 500×500 *matrices when* $k = 20$

	Algorithm 1	Algorithm 3		
Matrix Type	residual	max residual	min residual	mean residual
(4.2)	0.2312	3.3808	0.2312	3.2224

TABLE 4.9

Residuals $\min_z \|A_1 z - A_2\|_2$ *from Algorithm 1 and 40 runs of Algorithm 3 on a* 500×500 *matrix of the form (4.2) when* $k = 20$

.

where $1_{j,k}$ denotes the $j \times k$ matrix consisting entirely of ones, while $0_{j,k}$ denotes the $j \times k$ zero matrix. Eisenstat [6] has shown that the randomized algorithm picks columns from this matrix that violate the bound (3.4). In particular, it can be shown that the residual grows faster than $\mathcal{O}(k^{3/4} \log^{1/2} k(n-k)^{1/4})$ as n grows large for a fixed k, implying the probability of achieving any such bound would decrease to 0 as n becomes large.

We confirmed this experimentally. Although Algorithm 3 can do as well as Algorithm 1, as shown in Table 4.6, it never does better during the 40 runs. Additionally, as n grows large for a fixed k, Table 4.6 shows the minimum residuals of Algorithm 3 start to get large.

5. Making the Randomized Algorithm Deterministic. We have two concerns with the randomized Algorithm 2. The first is the use of probability distributions (3.5) and (3.6). Their complexity creates problems: (3.5) is expensive to compute because it requires a SVD, while (3.6) is numerically unstable. The simpler distribution (4.1) seems to work equally well (see section 4.5). Our second concern is the choice of c: Boutsidis, *et al.* [1] do not give a clear way to determine c.

To remedy these concerns, we used our empirical results to construct a two stage deterministic algorithm. We saw that Algorithm 3 often selects $c = 4k$, and we also saw that the simpler probability distribution (4.1) works as well as the complicated one (3.6). So, in our deterministic version of Algorithm 2, we select the $4k$ columns of V_k^T with the largest 2-norms. If multiple columns have the same norm as the $4k^{th}$ largest column, we simply take the first column with that norm. Then we apply the deterministic Algorithm 1 to these $4k$ candidate columns. Since we deterministically select enough candidate columns, there is no need to run the algorithm more than once. This is summarized as Algorithm 4. Although we have no bounds for this algorithm, we view it as a viable alternative for Algorithms 1 and 3.

n	100	250	500	750	1000
Algorithm 1	0.2887	0.2887	0.2887	0.2887	0.2887
min Algorithm 3	0.2887	3.2462	4.5778	0.2887	6.4646

TABLE 4.10

Residual for Algorithm 1and minimum residual for 40 runs of Algorithm 3 with $k = 10$ on an $n \times n$ matrix of the form (4.2).

Algorithm 4 Two Stage Deterministic Algorithm for Subset Selection

Input: $m \times n$ matrix A of rank r, integer $k \leq r$
Output: Permutation Π so that $A\Pi = \begin{pmatrix} A_1 & A_2 \end{pmatrix}$

Compute the r nonzero singular values of A, and the associated right singular vectors
Permute the columns of V_k^T in order of decreasing column norms: $W = V_k^T \Pi_1$
Set $\tilde{A}_{4k}$ equal to the leading $4k$ columns of $A\Pi_1$
Apply Algorithm 1 to $\tilde{A}_{4k}$ to return the permutation Π_2
Combine the permutations: $\Pi = \Pi_1 \begin{pmatrix} \Pi_2 & 0 \\ 0 & I_{n-4k} \end{pmatrix}$

5.1. Intuition behind Algorithm 4. As in section 3.2.1 we partition the SVD of A so that the right singular vectors in V_k are associated with the k largest singular values. We permute the columns of V_k^T (rows of V_k) so that the columns are arranged in order of descending 2-norm. That is, we choose a permutation matrix Π_1 so that

$$W = V_k^T \Pi_1, \qquad \text{where} \qquad \|We_i\|_2 \geq \|We_{i+1}\|_2, \qquad 1 \leq i \leq n-1.$$

Now we distinguish the columns with largest two norm and partition $W = \begin{pmatrix} W_1 & W_2 \end{pmatrix}$, where W_1 is $k \times k$. This is equivalent to sorting the columns of V_k^T using (4.1). Since

$$\|W_1\|_F^2 = \sum_{i=1}^{k} \|W_1 e_i\|_2^2,$$

the permutation Π_1 maximizes $\|W_1\|_F^2$ among all k columns of V_k^T and minimizes $\|W_2\|_F^2$.

We want to make the Frobenius norm $\|W_2\|_F$ small because, intuitively, this might also make the two norm $\|W_2\|_2$ smaller. This intuition comes from the fact that the Frobenius norm of W_2 is an upper bound for the two norm of W_2; specifically $\|W_2\|_2^2 \leq \|W_2\|_F^2 \leq k\|W_2\|_2^2$. Since W has orthonormal rows, we can apply the CS decomposition [10, Theorem 2.6.2] to W_1 and W_2 to conclude

$$\sigma_k(W_1)^2 + \|W_2\|_2^2 = 1.$$

This means, as $\|W_2\|_2$ gets smaller, $\sigma_k(W_1)$ increases. We want $\sigma_k(W_1)$ to increase because this makes W_1 better conditioned; that is, the columns are more linearly independent. Since $\sigma_k(W_1) = 1/\|W_1^{-1}\|_2$, an increase in $\sigma_k(W_1)$ implies a decrease in $\|W_1^{-1}\|_2$.

So far we have argued, based on intuition, that rearranging columns of V_k^T in order of descending two-norms leads to making W_1 as well-conditioned as possible. Now we use [11, Theorem 6.1], [14,

Theorem 1.5] to show what that means for our bounds:

$$\frac{\sigma_k(A)}{\|W_1^{-1}\|_2} \leq \sigma_k(A_1), \qquad \text{and} \qquad \min_z \|A_1 z - A_2\|_2 \leq \|W_1^{-1}\|_2 \, \sigma_{k+1}(A).$$

Therefore permuting columns from V_k^T with large norm to the front, brings representative columns from A to the front, so that $\sigma_k(A_1)$ is close to $\sigma_k(A)$, and $\min_z \|A_1 z - A_2\|$ is close to $\sigma_{k+1}(A)$.

5.2. Comparison with Algorithm 1. Since runtime improvements on large matrices would be a reason for developing an alternative algorithm to Algorithm 1, we ran Algorithm 1 and our new Algorithm 4 on our largest test matrices, of order 2000×2000, with a $k = 40$. We set $f = 1$ to get the best possible bounds with Algorithm 1. This was not feasible in our earlier experiments with the probabilistic Algorithms 2 and 3, for these algorithms are run 40 times, and with $f = 1$, Algorithm 1 may take exponential time. This makes any runtime gains of the new Algorithm 4 more noticeable since it performs Algorithm 1 on a fraction ($4k = 160$) of the 2000 columns of the matrix. Table 5.1 compares the smallest singular values $\sigma_k(A_1)$ and residuals $\min_z \|A_1 z - A_2\|_2$ from Algorithms 1 and 4. We also listed the ratio of run time from Algorithm 4 and Algorithm 1, which we got from Matlab's tic-toc function.

Matrix	Residuals		$\sigma_k(A_1)$		Time Alg 4/Alg 1
	Alg 1	Alg 4	Alg 1	Alg 4	
Kahan	4×10^0	4×10^0	8×10^{-2}	8×10^{-2}	0.11
rand	9×10^1	9×10^1	1×10^1	1×10^1	0.03
scalerand	1×10^2	1×10^2	2×10^1	2×10^1	0.07
GKS	4×10^0	3×10^1	3×10^{-1}	3×10^{-1}	0.02
(4.2)	2×10^{-1}	2×10^{-1}	1×10^0	1×10^0	0.09

TABLE 5.1
Smallest singular values $\sigma_k(A_1)$ and residuals $\min_z \|A_1 z - A_2\|_2$ for three 2000×2000 matrices when $k = 40$

Surprisingly, we see in Table 5.1 that there are no significant differences in the performance of the two deterministic algorithms in 3 out of the 4 cases. Algorithm 4 gives a larger residual than Algorithm 1 on the GKS matrix, but the results are still within one order of magnitude different. Also, Algorithm 4 is not susceptible to the problems that matrix (4.2) caused for Algorithm 3. Algorithms 1 and 4 get the same results on this type of matrix. Finally, when timed with MATLAB's tic-toc function, the new two stage Algorithm 4 is much faster than Algorithm 1.

6. Future Work. We briefly discuss some future research we would like to perform. We would like to use a finer iteration for values of c in Algorithm 3 instead of simply doubling it. Also, we would like to provide bounds for Algorithm 4, as well as running time comparisons for all the algorithms.

Acknowledgements. We would like to extend our sincerest thanks to our advisors, Dr. Ilse Ipsen and Rizwana Rehman of North Carolina State University for all their advice and support. We also thank Dr. Carl Meyer of NCSU, Dr. Stanley Eisenstat of Yale and two anonymous referees for their comments and suggestions. Finally, we thank the NSF and NSA for funding this research, and Dr. Hien Tran and Dr. Aloysius Helminck of NCSU for organizing our REU experience.

7. Appendix. We show (2.2).

LEMMA 7.1. *With the assumptions of section 2.4*

$$\sigma_i(A_1) = \sigma_i(R_k), \qquad \min_z \|A_1 z - A_2\|_2 = \sigma_1(C_k).$$

Proof. First we relate the submatrices of A and R. Distinguishing the leading k columns in A and R gives

$$A = \begin{pmatrix} A_1 & A_2 \end{pmatrix} = \begin{pmatrix} Q_1 & Q_2 & Q_3 \end{pmatrix} \begin{pmatrix} R_k & B_k \\ 0 & C_k \\ 0 & 0 \end{pmatrix}.$$

This implies

$$A_1 = Q_1 R_k, \qquad A_2 = Q_1 B_k + Q_2 C_k. \tag{7.1}$$

From $Q^T Q = I_m$ follows $Q_1^T Q_1 = I_k$, so that Q_1 has orthonormal columns. Hence R_k and $A_1 = Q_1 R_k$ have the same singular values, $\sigma_i(A_1) = \sigma_i(R_k)$, $1 \le i \le k$. This proves the first equality of the lemma.

In order to prove the second equality of the lemma, we express the residual as

$$\min_z \|A_1 z - A_2\|_2 = \|(I_m - A_1 A_1^\dagger) A_2\|_2, \tag{7.2}$$

where $A_1^\dagger$ is the Moore-Penrose inverse. Since by assumption, A_1 has k linearly independent columns, we can write the Moore-Penrose inverse as $A_1^\dagger = (A_1^T A_1)^{-1} A_1^T$. From (7.1) and the fact that Q_1 has orthonormal columns follows $A_1^\dagger = (R_k^T R_k)^{-1} R_k^T Q_1^T$. Now we argue that R_k is invertible. Since by assumption A_1 has full column rank, $\sigma_k(A_1) > 0$. The first equality of the lemma implies that $\sigma_k(R_k) > 0$. Since R_k is a $k \times k$ matrix with all k singular values non zero, this means, R_k is invertible. Hence the Moore-Penrose simplifies to $A_1^\dagger = R_k^{-1} Q_1^T$.

Substituting the expressions for A_1 and A_2 from (7.1) into (7.2) and using the facts $Q_1^T Q_1 = I$ and $Q_2^T Q_1 = 0$ gives

$$\begin{aligned} \|(I_m - A_1 A_1^\dagger) A_2\|_2 &= \|(I_m - Q_1 R_k R_k^{-1} Q_1^T)(Q_1 B_k + Q_2 C_k)\|_2 \\ &= \|(I_m - Q_1 Q_1^T)(Q_1 B_k + Q_2 C_k)\|_2 \\ &= \|Q_1 B_k + Q_2 C_k - (Q_1 Q_1^T)(Q_1 B_k) - Q_1 Q_1^T Q_2 C_k\|_2 \\ &= \|Q_2 C_k - Q_1 Q_1^T Q_2 C_k\|_2 = \|Q_2 C_k\|_2. \end{aligned}$$

At last, $\|Q_2 C_k\|_2^2 = \|(Q_2 C_k)^T Q_2 C_k\|_2 = \|C_k^T C_k\|_2 = \|C_k\|_2^2 = \sigma_1(C_k)^2$. ∎

REFERENCES

[1] C. BOUTSIDIS, M. W. MAHONEY, AND P. DRINEAS, *An improved approximation algorithm for the column subset selection problem*, 2008.

[2] ———, *An improved approximation algorithm for the column subset selection problem*, in SODA '09: Proceedings of the Nineteenth Annual ACM-SIAM Symposium on Discrete Algorithms, Philadelphia, PA, USA, 2009, Society for Industrial and Applied Mathematics, pp. 968–977.

[3] J. Butler, D. T. Bishop, and J. Barrett, *Strategies for selecting subsets of single-nucleotide polymorphisms to genotype in association studies*, BMC Genetics, 6 (2005), p. S72.
[4] S. Chandrasekaran and I. C. F. Ipsen, *On rank-revealing factorizations*, SIAM J. Matrix Anal. Appl., 15 (1994), pp. 592–622.
[5] P. Drineas, M. W. Mahoney, and S. Muthukrishnan, *Relative-error CUR matrix decompositions*, SIAM J. Matrix Anal. 3deAppl., 30 (2008), pp. 844–881.
[6] S. C. Eisenstat, *Private communication*, 2009.
[7] A. Frieze, R. Kannan, and S. Vempala, *Fast Monte-Carlo algorithms for finding low-rank approximations*, J. ACM, 51 (2004), pp. 1025–1041.
[8] G. Golub, *Numerical methods for solving linear least squares problems*, Numer, Math., 7 (1965), pp. 206–216.
[9] G. Golub and P. Businger, *Linear least squares solutions by Householder transformations*, Numer. Math., 7 (1965), pp. 269–276.
[10] G. Golub and C. Van Loan, *Matrix Computations*, The Johns Hopkins University Press, Baltimore, 3rd ed., 1996.
[11] G. H. Golub, V. Klema, and G. W. Stewart, *Rank degeneracy and least squares problems*, Tech. Report TR-456, Dept of Computer Science, University of Maryland, 1976.
[12] 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.
[13] N. Halko, P. G. Martinsson, and J. A. Tropp, *Finding structure with randomness: Stochastic algorithms for constructing approximate matrix decompositions*, acm report, Caltech, Sept. 2009.
[14] H. P. Hong and C.-T. Pan, *The rank-revealing QR decomposition and SVD*, Math. Comp., 58 (1992), pp. 213–32.
[15] E. Liberty, F. Woolfe, P.G. Martinsson, V. Rokhlin, and M. Tygert, *Randomized algorithms for the low-rank approximation of matrices*, PNAS, 104 (2007), pp. 20167–20172.
[16] C. T. Pan, *On the existence and computation of rank-revealing LU factorizations*, Linear Algebra Appl., 316 (2000), pp. 199 – 222.
[17] A. Wilzeck and T. Kaiser, *Antenna subset selection for cyclic prefix assisted MIMO wireless communications over frequency selective channels*, EURASIP J. Adv. Signal Process, 2008 (2008), pp. 1–14.