SIAM J. SCI. COMPUT.
Vol. 38, No. 5, pp. S485–S507

© 2016 Society for Industrial and Applied Mathematics

Downloaded 10/31/16 to 131.211.208.19. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

A RANDOMIZED BLOCKED ALGORITHM FOR EFFICIENTLY COMPUTING RANK-REVEALING FACTORIZATIONS OF MATRICES[*]

PER-GUNNAR MARTINSSON[†] AND SERGEY VORONIN[†]

Abstract. This manuscript describes a technique for computing partial rank-revealing factorizations, such as a partial QR factorization or a partial singular value decomposition. The method takes as input a tolerance ε and an $m \times n$ matrix A and returns an approximate low-rank factorization of A that is accurate to within precision ε in the Frobenius norm (or some other easily computed norm). The rank k of the computed factorization (which is an output of the algorithm) is in all examples we examined very close to the theoretically optimal ε-rank. The proposed method is inspired by the Gram–Schmidt algorithm and has the same $O(mnk)$ asymptotic flop count. However, the method relies on randomized sampling to avoid column pivoting, which allows it to be blocked, and hence accelerates practical computations by reducing communication. Numerical experiments demonstrate that the accuracy of the scheme is for every matrix that was tried at least as good as column-pivoted QR and is sometimes much better. Computational speed is also improved substantially, in particular on GPU architectures.

Key words. low-rank approximation, QR factorization, singular value decomposition, randomized algorithm

AMS subject classifications. 65F15, 65F25

DOI. 10.1137/15M1026080

1. Introduction.

1.1. Problem formulation. This manuscript describes an algorithm based on randomized sampling for computing an approximate low-rank factorization of a given matrix. To be precise, given a real or complex matrix A of size $m \times n$ and a computational tolerance ε, we seek to determine a matrix $\mathsf{A}_{\text{approx}}$ of low rank such that

$$\|\mathsf{A} - \mathsf{A}_{\text{approx}}\| \leq \varepsilon. \tag{1}$$

For any given $k \in \{1, 2, \dots, \min(m, n)\}$, a rank-$k$ approximation to A that is in many ways optimal is given by the partial singular value decomposition (SVD),

$$\begin{array}{ccccc} \mathsf{A}_k & = & \mathsf{U}_k & \mathbf{\Sigma}_k & \mathsf{V}_k^*, \\ m \times n & & m \times k & k \times k & k \times n \end{array} \tag{2}$$

where U_k and V_k are orthonormal matrices whose columns consist of the first k left and right singular vectors, respectively, and $\mathbf{\Sigma}_k$ is a diagonal matrix whose diagonal entries $\{\sigma_j\}_{j=1}^k$ are the leading k singular values of A, ordered so that $\sigma_1 \geq \sigma_2 \geq \sigma_3 \geq \cdots \geq \sigma_k \geq 0$. The Eckart–Young theorem [4] states that for the spectral norm and the Frobenius norm, the residual error is minimal,

$$\|\mathsf{A} - \mathsf{A}_k\| = \inf\{\|\mathsf{A} - \mathsf{C}\| : \mathsf{C} \text{ has rank } k\}.$$

[*] Received by the editors June 15, 2015; accepted for publication (in revised form) October 12, 2015; published electronically October 27, 2016. This research was supported by DARPA under contract N66001-13-1-4050 and by NSF under contract DMS-1407340.
http://www.siam.org/journals/sisc/38-5/M102608.html

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

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Downloaded 10/31/16 to 131.211.208.19. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

However, computing the factors in (2) is computationally expensive. In contrast, our objective is to find an approximant $\mathsf{A}_{\text{approx}}$ that is cheap to compute and *close* to optimal.

The method we present is designed for situations where A is sufficiently large that computing the full SVD is not economical. The method is designed to be highly communication efficient and to execute efficiently on both shared and distributed memory machines. It has been tested numerically for situations where the matrix fits in RAM on a single machine. We will, without loss of generality, assume that $m \geq n$. For the most part we discuss real matrices, but the generalization to complex matrices is straightforward.

1.2. A greedy template. A standard approach in computing low-rank factorizations is to employ a greedy algorithm to build, one vector at a time, an orthonormal basis $\{\mathsf{q}_j\}_{j=1}^{k}$ that approximately spans the columns of A. To be precise, given an $m \times n$ matrix A and a computational tolerance ε, our objective is to determine a rank k and an $m \times k$ matrix $\mathsf{Q}_k = [\mathsf{q}_1 \cdots \mathsf{q}_k]$ with orthonormal column vectors such that $\|\mathsf{A} - \mathsf{Q}_k\mathsf{B}_k\| \leq \varepsilon$, where $\mathsf{B}_k = \mathsf{Q}_k^*\mathsf{A}$. The matrices Q_k and B_k may be constructed jointly via the following procedure:

```
Algorithm 1
(1)   $\mathsf{Q}_0 = [\ ]$; $\mathsf{B}_0 = [\ ]$; $\mathsf{A}^{(0)} = \mathsf{A}$; $j = 0$;
(2)   while $\|\mathsf{A}^{(j)}\| > \varepsilon$
(3)       $j = j + 1$
(4)       Pick a unit vector $\mathsf{q}_j \in \text{ran}(\mathsf{A}^{(j-1)})$.
(5)       $\mathsf{b}_j = \mathsf{q}_j^*\mathsf{A}^{(j-1)}$
(6)       $\mathsf{Q}_j = [\mathsf{Q}_{j-1}\ \mathsf{q}_j]$
(7)       $\mathsf{B}_j = \begin{bmatrix} \mathsf{B}_{j-1} \\ \mathsf{b}_j \end{bmatrix}$
(8)       $\mathsf{A}^{(j)} = \mathsf{A}^{(j-1)} - \mathsf{q}_j\mathsf{b}_j$
(9)   end while
(10)  $k = j$.
```

Note that $\mathsf{A}^{(j)}$ can overwrite $\mathsf{A}^{(j-1)}$. It is straightforward to show that if the algorithm is executed in exact arithmetic, then the matrices generated satisfy

$$\mathsf{A}^{(j)} = \mathsf{A} - \mathsf{Q}_j\mathsf{Q}_j^*\mathsf{A} \qquad \text{and} \qquad \mathsf{B}_j = \mathsf{Q}_j^*\mathsf{A}. \tag{3}$$

The performance of the greedy scheme is determined by how we choose the vector q_j on line (4). If we pick q_j as simply the largest column of $\mathsf{A}^{(j-1)}$, scaled to yield a vector of unit length, then we recognize the scheme as the column-pivoted Gram–Schmidt algorithm for computing a QR factorization. This method often works very well but can lead to suboptimal factorizations. Reference [6] discusses this in detail and also provides an improved pivoting technique that can be proved to yield closer to optimal results. However, both standard Gram–Schmidt (see, e.g., [5, sect. 5.2]) and the improved version in [6] are challenging to implement efficiently on modern multicore processors since they cannot readily be blocked. Expressed differently, they rely on BLAS2 operations rather than BLAS3.

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Downloaded 10/31/16 to 131.211.208.19. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

Another natural choice for $\mathbf{q}_j$ on line (4) is to pick the unit vector that minimizes $\|\mathbf{A}^{(j-1)} - \mathbf{q}\mathbf{q}^*\mathbf{A}^{(j-1)}\|$. This in fact leads to an optimal factorization, with the vectors $\{\mathbf{q}_j\}_{j=1}^k$ being left singular vectors of $\mathbf{A}$. However, finding the minimizer tends to be computationally expensive.

In this manuscript, we propose a scheme that is more computationally efficient than column-pivoted Gram–Schmidt and often yields close to minimal approximation errors. The idea is to choose $\mathbf{q}_j$ as a random linear combination of the columns of $\mathbf{A}^{(j-1)}$. To be precise, we propose the following mechanism for choosing $\mathbf{q}_j$:

```
(4a)  Draw a random vector $\boldsymbol{\omega}$ whose entries are iid Gaussian random variables.
(4b)  Set $\mathbf{y} = \mathbf{A}^{(j-1)}\boldsymbol{\omega}$.
(4c)  Normalize so that $\mathbf{q}_j = \frac{1}{\|\mathbf{y}\|}\,\mathbf{y}$.
```

This scheme is mathematically very close to the low-rank approximation scheme proposed in [8] but is slightly different in the stopping criterion used (the scheme of [8] does not explicitly update the matrix and therefore relies on a probabilistic stopping criterion) and in its performance when executed with finite precision arithmetic. We argue that choosing the vector $\mathbf{q}_j$ using randomized sampling leads to performance very comparable to traditional column pivoting but has a decisive advantage in that the resulting algorithm is easy to block. We will demonstrate substantial practical speed-up on both multicore CPUs and GPUs.

Remark 1. The factorization scheme described in this section produces an approximate factorization of the form $\mathbf{A} \approx \mathbf{Q}_k\mathbf{B}_k$, where $\mathbf{Q}_k$ is orthonormal, but no conditions are a priori imposed on $\mathbf{B}_k$. Once the factors $\mathbf{Q}_k$ and $\mathbf{B}_k$ are available, it is simple to compute many standard factorizations, such as the low-rank QR, SVD, or CUR factorizations. For details, see section 3.3.

2. Technical preliminaries.

2.1. Notation.

Throughout the paper, we measure vectors in $\mathbb{R}^n$ using their Euclidean norm. The default norm for matrices will be the Frobenius norm $\|\mathbf{A}\| = (\sum_{i,j} |\mathbf{A}(i,j)|^2)^{1/2}$, although other norms will also be discussed.

We use the notation of Golub and Van Loan [5] to specify submatrices. In other words, if $\mathbf{B}$ is an $m \times n$ matrix with entries b_{ij}, and $I = [i_1, i_2, \ldots, i_k]$ and $J = [j_1, j_2, \ldots, j_\ell]$ are two index vectors, then we let $\mathbf{B}(I,J)$ denote the $k \times \ell$ matrix

$$\mathbf{B}(I,J) = \begin{bmatrix} b_{i_1j_1} & b_{i_1j_2} & \cdots & b_{i_1j_\ell} \\ b_{i_2j_1} & b_{i_2j_2} & \cdots & b_{i_2j_\ell} \\ \vdots & \vdots & & \vdots \\ b_{i_kj_1} & b_{i_kj_2} & \cdots & b_{i_kj_\ell} \end{bmatrix}.$$

We let $\mathbf{B}(I,:)$ denote the matrix $\mathbf{B}(I,[1, 2, \ldots, n])$, and we define $\mathbf{B}(:,J)$ analogously.

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

2.2. The SVD.

The SVD was introduced briefly in the introduction. Here we define it again, with some more detail added. Let $\mathbf{A}$ denote an $m \times n$ matrix, and set $r = \min(m,n)$. Then $\mathbf{A}$ admits a factorization

$$\underset{m\times n}{\mathbf{A}} = \underset{m\times r}{\mathbf{U}}\ \underset{r\times r}{\mathbf{\Sigma}}\ \underset{r\times n}{\mathbf{V}^*}, \tag{4}$$

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Downloaded 10/31/16 to 131.211.208.19. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

where the matrices U and V are orthonormal, and $\mathbf{\Sigma}$ is diagonal. We let $\{\mathsf{u}_i\}_{i=1}^r$ and $\{\mathsf{v}_i\}_{i=1}^r$ denote the columns of U and V, respectively. These vectors are the left and right singular vectors of A. As in the introduction, the diagonal elements $\{\sigma_j\}_{j=1}^r$ of $\mathbf{\Sigma}$ are the singular values of A. We order these so that $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_r \geq 0$. We let A_k denote the truncation of the SVD to its first k terms, as defined by (2). It is easily verified that

$$\|\mathsf{A} - \mathsf{A}_k\|_{\text{spectral}} = \sigma_{k+1} \qquad \text{and that} \qquad \|\mathsf{A} - \mathsf{A}_k\| = \left(\sum_{j=k+1}^{\min(m,n)} \sigma_j^2\right)^{1/2}, \tag{5}$$

where $\|\mathsf{A}\|_{\text{spectral}}$ denotes the operator norm of A and $\|\mathsf{A}\|$ denotes the Frobenius norm of A. Moreover, the Eckart–Young theorem [4] states that these errors are the smallest possible errors that can be incurred when approximating A by a matrix of rank k.

2.3. The QR factorization.

Any $m \times n$ matrix A admits a *QR factorization* of the form

$$\underset{m\times n}{\mathsf{A}} \quad \underset{n\times n}{\mathsf{P}} \quad = \quad \underset{m\times r}{\mathsf{Q}} \quad \underset{r\times n}{\mathsf{R}}, \tag{6}$$

where $r = \min(m,n)$, Q is orthonormal, R is upper triangular, and P is a permutation matrix. The permutation matrix P can more efficiently be represented via a vector $J_c \in \mathbb{Z}_+^n$ of column indices such that $\mathsf{P} = \mathsf{I}(:, J_c)$, where I is the $n \times n$ identity matrix. Then (6) can be written

$$\underset{m\times n}{\mathsf{A}(:,J_c)} \quad = \quad \underset{m\times r}{\mathsf{Q}} \quad \underset{r\times n}{\mathsf{R}}. \tag{7}$$

The QR factorization is often computed via column pivoting combined with either the Gram–Schmidt process, Householder reflectors, or Givens rotations [5]. The resulting upper triangular R then satisfies various decay conditions [5]. These techniques are all incremental and can be stopped after the first k terms have been computed to obtain a "partial QR factorization of A":

$$\underset{m\times n}{\mathsf{A}(:,J_c)} \quad \approx \quad \underset{m\times k}{\mathsf{Q}_k} \quad \underset{k\times n}{\mathsf{R}_k}. \tag{8}$$

A drawback of the classical column-pivoted QR factorization algorithm is that it cannot be easily blocked, making it hard to approach peak performance on multiprocessor architectures.

2.4. Orthonormalization.

Given an $m \times \ell$ matrix X, with $m \geq \ell$, we introduce the function

$$\mathsf{Q} = \mathtt{orth}(\mathsf{X})$$

to denote orthonormalization of the columns of X. In other words, Q will be an $m \times \ell$ orthonormal matrix whose columns form a basis for the column space of X. In practice, this step is typically achieved most efficiently by a call to a packaged QR factorization; e.g., in MATLAB, we would write $\mathsf{Q} = \mathtt{qr}(\mathsf{X}, 0)$. However, all calls to `orth` in this manuscript can be implemented *without pivoting*, which makes efficient implementation much easier.

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Downloaded 10/31/16 to 131.211.208.19. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

3. Construction of low-rank approximations via randomized sampling.

3.1. A basic randomized scheme. Let $\mathbf{A}$ be a given $m \times n$ matrix, and suppose that we seek to determine a matrix $\mathbf{Q}$ with ℓ orthonormal columns such that

$$\mathbf{A} \approx \mathbf{Q}\,\mathbf{B}, \qquad \text{where} \qquad \mathbf{B} = \mathbf{Q}^*\,\mathbf{A}. \tag{9}$$

In other words, we seek a matrix $\mathbf{Q}$ whose columns form an approximate orthonormal basis for the column space of $\mathbf{A}$. A randomized procedure for solving this task was proposed in [9] and later analyzed and elaborated in [10, 8]. A basic version of the scheme that we call "`randQB`" is given in Figure 1. Once `randQB` has been executed to produce the factors $\mathbf{Q}$ and $\mathbf{B}$ in (9), standard factorizations such as the QR factorization or the truncated SVD can easily be obtained, as described in section 3.3.

For (9) to be an accurate approximation, the singular values of $\mathbf{A}$ need to exhibit some level of decay. It turns out that the faster they decay, the easier it becomes to find "close to optimal" approximations. Some theoretical results describing the performance of `randQB` are given in section 3.2, and in section 5 we describe some modifications to the scheme that enhance the accuracy in situations where the singular values decay slowly.

3.2. Oversampling and theoretical performance guarantees. The algorithm `randQB` described in section 3.1 produces close to optimal results for matrices whose singular values decay rapidly, provided that some slight oversampling is done. To be precise, if we seek to match the minimal error for a factorization of rank k, then choose ℓ in `randQB` as

$$\ell = k + s,$$

where s is a small integer (say, $s = 10$). It was shown in [8, Thm. 10.5] that if $s \geq 2$, then

$$\mathbb{E}\big[\|\mathbf{A} - \mathbf{QB}\|\big] \leq \left(1 + \frac{k}{s-1}\right)\left(\sum_{j=k+1}^{\min(m,n)} \sigma_j^2\right)^{1/2},$$

where $\mathbb{E}$ denotes expectation. Recall from equation (5) that $(\sum_{j=k+1}^{\min(m,n)} \sigma_j^2)^{1/2}$ is the theoretically minimal error in approximating $\mathbf{A}$ by a matrix of rank k, so we miss the optimal bound only by a factor of $(1 + \frac{k}{s-1})$ (except for the oversampling, of course). Moreover, the likelihood of a substantial deviation from the expectation is extremely small; see [8, sect. 10.3] for a proof and section 6.4 for numerical evidence.

Remark 2. When errors are measured in the spectral norm, as opposed to the Frobenius norm, the randomized scheme is slightly further removed from optimality.

```
function $[\mathbf{Q}, \mathbf{B}] = \texttt{randQB}(\mathbf{A}, \ell)$

$\boldsymbol{\Omega} = \texttt{randn}(n, \ell)$
$\mathbf{Q} = \texttt{orth}(\mathbf{A}\boldsymbol{\Omega})$        $C_{\rm mm}\, mn\ell + m\ell^2$
$\mathbf{B} = \mathbf{Q}^*\mathbf{A}$                  $C_{\rm mm}\, mn\ell$
```

FIG. 1. *The most basic version of the randomized range finder. The algorithm takes as input an $m \times n$ matrix $\mathbf{A}$ and a target rank ℓ and produces factors $\mathbf{Q}$ and $\mathbf{B}$ of sizes $m \times \ell$ and $\ell \times n$, respectively, such that $\mathbf{A} \approx \mathbf{QB}$. Text in blue refers to computational cost; see section* 4.4 *for notation.*

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Downloaded 10/31/16 to 131.211.208.19. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

Theorem 10.6 of [8] states that

$$\mathbb{E}\big[\|\mathsf{A}-\mathsf{QB}\|_{\text{spectral}}\big] \leq \left(1+\frac{k}{s-1}\right)\sigma_{k+1} + \frac{e\sqrt{k+s}}{s}\left(\sum_{j=k+1}^{\min(m,n)}\sigma_j^2\right)^{1/2}, \tag{10}$$

where e is the basis of the natural exponent. We observe that in cases where the singular values decay slowly, the right-hand side of (10) is substantially larger than the theoretically optimal value of σ_{k+1}. For such a situation, the "power scheme" described in section 5.1 should be used.

3.3. Computing standard factorizations.

The output of the randomized factorization scheme in Figure 1 is a factorization $\mathsf{A}\approx\mathsf{QB}$ where Q is orthonormal, but no constraints have been placed on B. It turns out that standard factorizations can efficiently be computed from the factors Q and B; in this section we describe how to get the QR, the SVD, and "interpolatory" factorizations.

3.3.1. Computing the low-rank SVD.

To get a low-rank SVD (cf. section 2.2), we perform the full SVD on the $\ell\times n$ matrix B to obtain a factorization $\mathsf{B}=\hat{\mathsf{U}}\,\hat{\mathsf{D}}\,\hat{\mathsf{V}}$. Then,

$$\mathsf{A}\approx\mathsf{QB}=\mathsf{Q}\hat{\mathsf{U}}\hat{\mathsf{D}}\hat{\mathsf{V}}^*.$$

We can now choose a rank k to use based on the decaying singular values of D. Once a suitable rank has been chosen, we form the low-rank SVD factors,

$$\mathsf{U}_k=\mathsf{Q}\hat{\mathsf{U}}(:,1:k),\qquad \mathbf{\Sigma}_k=\hat{\mathsf{D}}(1:k,1:k),\qquad\text{and}\qquad \mathsf{V}_k=\hat{\mathsf{V}}(:,1:k),$$

so that $\mathsf{A}\approx\mathsf{U}_k\mathbf{\Sigma}_k\mathsf{V}_k^*$. Observe that the truncation undoes the oversampling that was done and detects a numerical rank k that is typically very close to the optimal ε-rank.

3.3.2. Computing the partial pivoted QR factorization.

To obtain the factorization $\mathsf{AP}\approx\mathsf{QR}$ (cf. section 2.3) from the QB decomposition, perform a QR factorization of the $\ell\times n$ matrix B to obtain $\mathsf{BP}=\tilde{\mathsf{Q}}\mathsf{R}$. Then, set $\hat{\mathsf{Q}}=\mathsf{Q}\tilde{\mathsf{Q}}$ to obtain

$$\mathsf{AP}\approx\mathsf{QBP}=\mathsf{Q}\tilde{\mathsf{Q}}\mathsf{R}=\hat{\mathsf{Q}}\mathsf{R}.$$

3.3.3. Computing interpolatory and CUR factorizations.

In applications such as data interpretation, it is often of interest to determine a subset of the rows/columns of A that form a good basis for its row/column space. For concreteness, suppose that A is an $m\times n$ matrix of rank k and that we seek to determine an index set J of length k and a matrix Y of size $k\times n$ such that

$$\underset{m\times n}{\mathsf{A}} \approx \underset{m\times k}{\mathsf{A}(:,J)}\;\underset{k\times n}{\mathsf{Y}}. \tag{11}$$

One can prove that there always exist such a factorization for which every entry of Y is bounded in modulus by 1 (which is to say that the columns in $\mathsf{A}(:,J)$ form a well-conditioned basis for the range of A) and for which $\mathsf{Y}(:,J)$ is the $k\times k$ identity matrix [2]. Now suppose that we have available a factorization $\mathsf{A}=\mathsf{QB}$ where B is of size $\ell\times n$. Then determine J and Y such that

$$\underset{\ell\times n}{\mathsf{B}} \approx \underset{\ell\times k}{\mathsf{B}(:,J)}\;\underset{k\times n}{\mathsf{Y}}. \tag{12}$$

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Downloaded 10/31/16 to 131.211.208.19. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

This can be done using the techniques in, e.g., [2] or [6]. Then (11) holds *automatically* for the index set J and the matrix $\mathbf{Y}$ that were constructed. Using similar ideas, one can determine a set of rows that form a well-conditioned basis for the row space, and also the so-called CUR factorization

$$\underset{m\times n}{\mathbf{A}} \approx \underset{m\times k}{\mathbf{C}} \; \underset{k\times k}{\mathbf{U}} \; \underset{k\times n}{\mathbf{R}},$$

where $\mathbf{C}$ and $\mathbf{R}$ consist of subsets of the columns and rows of $\mathbf{A}$, respectively; cf. [15].

4. A blocked version of the randomized range finder.

Highly efficient implementations of linear algebraic algorithms exploit *blocking* to attain high performance. Blocking algorithms make it easier to maintain a high throughput at all levels of the memory hierarchy and to optimally feed multicore processors [7, 3]. The gains are particularly pronounced for the matrix-matrix multiplication, which parallelizes well [16].

In this section, we demonstrate that the basic randomized scheme described in Figure 1 is easily blocked. This allows us to match the very high efficiency of BLAS3 and standard library routines while still being able to incorporate *adaptive rank determination*. The optimal choice of block size depends strongly on what hardware is used and is a topic that is currently under investigation.

The algorithm described in this section is directly inspired by Algorithm 4.2 of [8]; besides blocking, the scheme proposed here is different in that the matrix $\mathbf{A}$ is updated in a manner analogous to “modified” column-pivoted Gram–Schmidt. This updating allows the randomized stopping criterion employed in [8] to be replaced with a precise deterministic stopping criterion.

4.1. Blocking.

Converting the basic scheme in Figure 1 to a blocked scheme is in principle straightforward. Suppose that in addition to an $m \times n$ matrix $\mathbf{A}$ and a rank ℓ, we have set a block size b such that $\ell = sb$ for some integer s. Then draw an $n \times \ell$ Gaussian random matrix $\boldsymbol{\Omega}$ and partition it into slices $\{\boldsymbol{\Omega}_j\}_{j=1}^{s}$, each of size $n \times b$, so that

$$\boldsymbol{\Omega} = \big[\boldsymbol{\Omega}_1,\, \boldsymbol{\Omega}_2,\, \dots,\, \boldsymbol{\Omega}_s\big]. \tag{13}$$

We analogously partition the matrices $\mathbf{Q}$ and $\mathbf{B}$ in groups of b columns and b rows, respectively,

$$\mathbf{Q} = \big[\mathbf{Q}_1,\, \mathbf{Q}_2,\, \dots,\, \mathbf{Q}_s\big] \qquad \text{and} \qquad \mathbf{B} = \begin{bmatrix} \mathbf{B}_1 \\ \mathbf{B}_2 \\ \vdots \\ \mathbf{B}_s \end{bmatrix}.$$

The blocked algorithm then proceeds to build the matrices $\{\mathbf{Q}_i\}_{i=1}^{s}$ and $\{\mathbf{B}_i\}_{i=1}^{s}$ one at a time. We first initiate the algorithm by setting

$$\mathbf{A}^{(0)} = \mathbf{A}. \tag{14}$$

Then step forwards, computing for $i = 1, 2, \dots, s$ the matrices

$$\mathbf{Q}_i = \texttt{orth}\left(\mathbf{A}^{(i-1)}\boldsymbol{\Omega}_i\right), \tag{15}$$

$$\mathbf{B}_i = \mathbf{Q}_i^{*}\mathbf{A}^{(i-1)}, \tag{16}$$

$$\mathbf{A}^{(i)} = \mathbf{A}^{(i-1)} - \mathbf{Q}_i\mathbf{B}_i. \tag{17}$$

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Downloaded 10/31/16 to 131.211.208.19. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

We will next prove that the matrix $\bar{\mathbf{Q}}_i = [\mathbf{Q}_1, \mathbf{Q}_2, \ldots, \mathbf{Q}_i]$ constructed is indeed orthonormal and that the matrix $\mathbf{A}^{(i)}$ defined by (17) is the "remainder" after i steps, in the sense that

$$\mathbf{A}^{(i)} = \mathbf{A} - \left[\mathbf{Q}_1, \mathbf{Q}_2, \ldots, \mathbf{Q}_i\right]\left[\mathbf{Q}_1, \mathbf{Q}_2, \ldots, \mathbf{Q}_i\right]^* \mathbf{A} = \mathbf{A} - \bar{\mathbf{Q}}_i \bar{\mathbf{Q}}_i^* \mathbf{A}.$$

To be precise, we will prove the following proposition.

PROPOSITION 4.1. *Let* $\mathbf{A}$ *be an* $m \times n$ *matrix. Let* b *denote a block size, and let* s *denote the number of steps. Suppose that the rank of* $\mathbf{A}$ *is at least* sb*. Let* $\mathbf{\Omega}$ *be a Gaussian random matrix of size* $n \times sb$*, partitioned as in* (13)*, with each* $\mathbf{\Omega}_j$ *of size* $n \times b$*. Let* $\{\mathbf{A}^{(j)}\}_{j=0}^{i}$*,* $\{\mathbf{Q}_j\}_{j=1}^{i}$*, and* $\{\mathbf{B}_j\}_{j=1}^{i}$*, be defined by* (14)–(17)*. Set*

$$\mathbf{P}_i = \sum_{j=1}^{i} \mathbf{Q}_j \, \mathbf{Q}_j^* \tag{18}$$

and

$$\bar{\mathbf{Q}}_i = \left[\mathbf{Q}_1, \mathbf{Q}_2, \ldots, \mathbf{Q}_i\right], \quad \bar{\mathbf{B}}_i = \left[\mathbf{B}_1^*, \mathbf{B}_2^*, \ldots, \mathbf{B}_i^*\right]^*, \quad \bar{\mathbf{Y}}_i = \left[\mathbf{A\Omega}_1, \mathbf{A\Omega}_2, \ldots, \mathbf{A\Omega}_i\right]. \tag{19}$$

Then for every $i = 1, 2, \ldots, s$*, it is the case that*

(a) *the matrix* $\bar{\mathbf{Q}}_i$ *is ON, so* $\mathbf{P}_i$ *is an orthogonal projection;*

(b) $\mathbf{A}^{(i)} = \left(\mathbf{I} - \mathbf{P}_i\right)\mathbf{A} = \left(\mathbf{I} - \bar{\mathbf{Q}}_i \bar{\mathbf{Q}}_i^*\right)\mathbf{A}$ *and* $\bar{\mathbf{B}}_i = \bar{\mathbf{Q}}_i^* \mathbf{A}$*;*

(c) $R(\bar{\mathbf{Q}}_i) = R(\bar{\mathbf{Y}}_i)$ *(where* $R(\mathbf{X})$ *denotes the range of a matrix* $\mathbf{X}$*).*

Proof. The proof is by induction. We will several times use that if $\mathbf{C}$ is a matrix of size $n \times b$ of full rank, and we set $\mathbf{Q} = \texttt{orth}(\mathbf{C})$, then $R(\mathbf{Q}) = R(\mathbf{C})$. We will also use the fact that if $\mathbf{\Omega}$ is a Gaussian random matrix of size $n \times \ell$, and $\mathbf{E}$ is a matrix of size $m \times n$ with rank at least ℓ, then the rank of $\mathbf{E\Omega}$ is with probability 1 precisely ℓ [8].

Direct inspection of the definitions show that (a), (b), (c) are all true for $i = 1$. Suppose all statements are true for $i - 1$. We will prove that then (a), (b), (c) hold for i.

To prove that (a) holds for i, we use that (b) holds for $i - 1$ and insert this into (15) to get

$$\mathbf{Q}_i = \texttt{orth}((\mathbf{I} - \mathbf{P}_{i-1})\mathbf{A\Omega}_i). \tag{20}$$

Then observe that $\mathbf{P}_{i-1}$ is the orthogonal projection onto a space of dimension $b(i-1)$, which means that the matrix $\mathbf{A}^{(i-1)} = (\mathbf{I} - \mathbf{P}_{i-1})\mathbf{A}$ has rank at least $bs - b(i-1) = b(s - i + 1) \geq b$. Consequently, $\mathbf{A}^{(i-1)}\mathbf{\Omega}_i$ has rank precisely b. This shows that

$$R(\mathbf{Q}_i) \subseteq R(\mathbf{I} - \mathbf{P}_{i-1}) = R(\mathbf{P}_{i-1})^{\perp} = R([\mathbf{Q}_1, \mathbf{Q}_2, \ldots, \mathbf{Q}_{i-1}])^{\perp}.$$

It follows that $\mathbf{Q}_j^* \mathbf{Q}_i = \mathbf{0}$ whenever $j < i$, which shows that $\bar{\mathbf{Q}}_i$ is ON. Next,

$$\mathbf{B}_i = \mathbf{Q}_i^* \mathbf{A}^{(i-1)} = \mathbf{Q}_i^* \left(\mathbf{I} - \mathbf{Q}_{i-1}\mathbf{Q}_{i-1}^*\right)\mathbf{A}^{(i-2)} = \mathbf{Q}_i^* \mathbf{A}^{(i-2)} = \cdots = \mathbf{Q}_i^* \mathbf{A}^{(0)} = \mathbf{Q}_i^* \mathbf{A}.$$

It follows that

$$\bar{\mathbf{Q}}_i^* \mathbf{A} = [\mathbf{Q}_1, \ldots, \mathbf{Q}_i]^* \mathbf{A} = \begin{bmatrix} \mathbf{Q}_1^* \mathbf{A} \\ \vdots \\ \mathbf{Q}_i^* \mathbf{A} \end{bmatrix} = \begin{bmatrix} \mathbf{B}_1 \\ \vdots \\ \mathbf{B}_i \end{bmatrix} = \bar{\mathbf{B}}_i.$$

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Downloaded 10/31/16 to 131.211.208.19. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

Thus, (a) holds for i.

Proving (b) is a simple calculation. Combining (16) and (17) we get

$$\begin{aligned}\mathbf{A}^{(i)} = \mathbf{A}^{(i-1)} - \mathbf{Q}_i\mathbf{Q}_i\mathbf{A}^{(i-1)} = (\mathbf{I} - \mathbf{Q}_i\mathbf{Q}_i^*)\mathbf{A}^{(i-1)} &\overset{(b)}{=} (\mathbf{I} - \mathbf{Q}_i\mathbf{Q}_i^*)(\mathbf{I} - \mathbf{P}_{i-1})\mathbf{A} \\ &= (\mathbf{I} - (\mathbf{P}_{i-1} + \mathbf{Q}_i\mathbf{Q}_i^*))\mathbf{A},\end{aligned}$$

where in the last step we used that $\mathbf{Q}_i^*\mathbf{P}_{i-1} = \mathbf{0}$. Since $\mathbf{P}_i = \mathbf{P}_{i-1} + \mathbf{Q}_i\mathbf{Q}_i^*$, this proves (b).

To prove (c), we observe that (20) implies that

$$R(\mathbf{Q}_i) \subseteq R([\mathbf{A\Omega}_i, \mathbf{P}_{i-1}\mathbf{A\Omega}_i]). \tag{21}$$

Induction assumption (c) tells us that

$$R(\mathbf{P}_{i-1}\mathbf{A\Omega}_i) \subseteq R([\mathbf{Q}_1,\ \mathbf{Q}_2,\ \dots,\ \mathbf{Q}_{i-1}]) = R([\mathbf{A\Omega}_1,\ \mathbf{A\Omega}_2,\ \dots,\ \mathbf{A\Omega}_{i-1}]). \tag{22}$$

Combining (21) and (22), we find

$$R(\mathbf{Q}_i) \subseteq R([\mathbf{A\Omega}_1,\ \mathbf{A\Omega}_2,\ \dots,\ \mathbf{A\Omega}_{i-1},\ \mathbf{A\Omega}_i]). \tag{23}$$

Equation (23) together with the induction assumption (c) imply that $R([\mathbf{Q}_1, \mathbf{Q}_2, \dots, \mathbf{Q}_i]) \subseteq R([\mathbf{A\Omega}_1, \mathbf{A\Omega}_2, \dots, \mathbf{A\Omega}_i])$. But both of these spaces have dimension precisely bi, so the fact that one is a subset of the other implies that they must be identical. □

Let us next compare the blocked algorithm defined by relations (14)–(17) to the unblocked algorithm described in Figure 1. For a fixed Gaussian matrix $\mathbf{\Omega}$, let the output of the blocked version be $\{\mathbf{Q}, \mathbf{B}\}$ and let the output of the unblocked method be $\{\tilde{\mathbf{Q}}, \tilde{\mathbf{B}}\}$. These two pairs of matrices do not need to be identical. (They depend on how exactly the QR factorizations are implemented, for instance). However, Proposition 4.1 demonstrates that the projectors $\mathbf{QQ}^*$ and $\tilde{\mathbf{Q}}\tilde{\mathbf{Q}}^*$ are identical. To be precise, both of these matrices represent the orthogonal projection onto the space $R(\mathbf{A\Omega})$. This means that the errors resulting from the two algorithms are also identical,

$$\underbrace{\mathbf{A} - \mathbf{QQ}^*\mathbf{A}}_{\textit{error from blocked algorithm}} = \underbrace{\mathbf{A} - \tilde{\mathbf{Q}}\tilde{\mathbf{Q}}^*\mathbf{A}}_{\textit{error from nonblocked algorithm}}.$$

Consequently, all theoretical results given in [8] (cf. section 3.2) directly apply to the output of the blocked algorithm too.

4.2. Adaptive rank determination. The blocked algorithm defined by (14)–(17) was presented in section 4.1 for the case where the rank ℓ of the approximation is given in advance. A perhaps more common situation in practical applications is that a precision $\varepsilon > 0$ is specified, and then we seek to compute an approximation of as low rank as possible that is accurate to precision ε. Observe that in the algorithm defined by (14)–(17), we proved that after step i has been completed, we have

$$\|\mathbf{A}^{(i)}\| = \|\mathbf{A} - \mathbf{P}_i\mathbf{A}\| = \|\mathbf{A} - \begin{bmatrix}\mathbf{Q}_1 & \mathbf{Q}_2 & \cdots & \mathbf{Q}_i\end{bmatrix}\begin{bmatrix}\mathbf{Q}_1 & \mathbf{Q}_2 & \cdots & \mathbf{Q}_i\end{bmatrix}^*\mathbf{A}\|.$$

In other words, $\mathbf{A}^{(i)}$ holds precisely the residual remaining after step i. This means that incorporating adaptive rank determining is now trivial—we simply compute

Downloaded 10/31/16 to 131.211.208.19. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

```
         function $[\mathbf{Q}, \mathbf{B}] = \texttt{randQB\_b}(\mathbf{A}, \varepsilon, b)$

(1)      for $i = 1, 2, 3, \ldots$
(2)          $\mathbf{\Omega}_i = \texttt{randn}(n, b)$
(3)          $\mathbf{Q}_i = \texttt{orth}(\mathbf{A}\mathbf{\Omega}_i)$                                   $C_{\rm mm}mnb + C_{\rm qr}mb^2$
(3′)         $\mathbf{Q}_i = \texttt{orth}(\mathbf{Q}_i - \sum_{j=1}^{i-1} \mathbf{Q}_j\mathbf{Q}_j^*\mathbf{Q}_i)$        $2(i-1)C_{\rm mm}mb^2 + C_{\rm qr}mb^2$
(4)          $\mathbf{B}_i = \mathbf{Q}_i^*\mathbf{A}$                                             $C_{\rm mm}mnb$
(5)          $\mathbf{A} = \mathbf{A} - \mathbf{Q}_i\mathbf{B}_i$                                          $C_{\rm mm}mnb$
(6)          if $\|\mathbf{A}\| < \varepsilon$ then stop
(7)      end for
(8)      Set $\mathbf{Q} = [\mathbf{Q}_1\ \cdots\ \mathbf{Q}_i]$ and $\mathbf{B} = [\mathbf{B}_1^*\ \cdots\ \mathbf{B}_i^*]^*$.
```

FIG. 2. *A blocked version of the randomized range finder; cf. Figure* 1. *The algorithm takes as input an $m \times n$ matrix $\mathbf{A}$, a block size b, and a tolerance ε. Its output are factors $\mathbf{Q}$ and $\mathbf{B}$ such that $\|\mathbf{A} - \mathbf{QB}\| \leq \varepsilon$. Note that if the algorithm is executed in exact arithmetic, then line* (3′) *does nothing. Text in blue refers to computational cost; see section* 4.4 *for notation.*

$\|\mathbf{A}^{(i)}\|$ after completing step i and break once $\|\mathbf{A}^{(i)}\| \leq \varepsilon$. The algorithm resulting is shown as `randQB_b` in Figure 2. (The purpose of line (3′) will be explained in section 4.3.)

Remark 3. Recall that our default norm in this manuscript, the Frobenius norm, is simple to compute, which means that the check on whether to break the loop on line (7) in Figure 2 hardly adds at all to the execution time. If circumstances warrant the use of a norm that is more expensive to compute, then some modification of the algorithm would be required. Suppose, for instance, that we seek an approximation in the spectral norm. We could then use the fact that the Frobenius norm is an upper bound on the spectral norm, keep the Frobenius norm as the breaking condition, and then eliminate any "superfluous" degrees of freedom that were included in the postprocessing step; cf. section 3.3.1. (This approach would be practicable only for matrices whose singular values exhibit reasonable decay, as otherwise the discrepancy in the ε-ranks could be prohibitively large.)

4.3. Floating point arithmetic.

When the algorithm defined by (14)–(17) is carried out in finite precision arithmetic, a serious problem often arises in that round-off errors will accumulate and will cause loss of orthonormality among the columns in $\{\mathbf{Q}_1, \mathbf{Q}_2, \ldots\}$. The problem is that as the computation proceeds, the columns of each computed matrix $\mathbf{Q}_i$ will due to round-off errors drift into the span of the columns of $\{\mathbf{Q}_1, \mathbf{Q}_2, \ldots, \mathbf{Q}_{i-1}\}$. This phenomenon occurs for the classical (nonblocked) Gram–Schmidt procedure as well, and the standard solution is to either use Householder projections [5] or to explicitly reproject any new basis vector away from the span of the previous vectors before adding it to the basis [1]. For our purposes, the latter technique generalizes directly, and the resulting operation appears in Figure 2 on line (3′). (Note that if the algorithm is carried out in exact arithmetic, then $\mathbf{Q}_j^*\mathbf{Q}_i = \mathbf{0}$ whenever $j < i$, so line (3′) would have no effect.)

4.4. Comparison of execution times.

Let us compare the computational cost of algorithms `randQB` (Figure 1) and `randQB_b` (Figure 2). To this end, let $C_{\rm mm}$ and

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Downloaded 10/31/16 to 131.211.208.19. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

C_{qr} denote the scaling constants for the cost of executing a matrix-matrix multiplication and a full QR factorization, respectively. In other words, we assume that

- multiplying two matrices of sizes $m \times n$ and $n \times r$ costs $C_{\mathrm{mm}}\, mnr$;
- performing a QR factorization of a matrix of size $m \times n$, with $m \geq n$, costs $C_{\mathrm{qr}}\, mn^2$.

Note that these are rough estimates. Actual costs depend on the actual sizes, but this model is still instructive. The execution time for the algorithm in Figure 1 is easily seen to be

$$T_{\texttt{randQB}} \sim 2C_{\mathrm{mm}}\, mn\ell + C_{\mathrm{qr}}\, m\ell^2. \tag{24}$$

For the blocked algorithm of Figure 2, we assume that it stops after s steps and set $\ell = sb$. Then

$$\begin{aligned} T_{\texttt{randQB\_b}} &\sim \sum_{i=1}^{s} \left[3C_{\mathrm{mm}}mnb + 2(i-1)C_{\mathrm{mm}}mb^2 + C_{\mathrm{qr}}2mb^2\right] \\ &\sim 3sC_{\mathrm{mm}}mnb + s^2C_{\mathrm{mm}}mb^2 + sC_{\mathrm{qr}}2mb^2. \end{aligned}$$

Using that $sb = \ell$ we find

$$T_{\texttt{randQB\_b}} \sim 3C_{\mathrm{mm}}mn\ell + C_{\mathrm{mm}}m\ell^2 + \frac{2}{s}C_{\mathrm{qr}}m\ell^2. \tag{25}$$

Comparing (24) and (25), we see that the blocked algorithm involves one additional term of $C_{\mathrm{mm}}mn\ell$ but on the other hand spends less time executing full QR factorizations, as expected.

Remark 4. All blocked algorithms that we present share the characteristic that they slightly increase the amount of time spent on matrix-matrix multiplication while reducing the amount of time spent performing QR factorization. This is a good trade-off on many platforms but becomes particularly useful when the algorithm is executed on a GPU. These massively multicore processors are particularly efficient at performing matrix-matrix multiplications but struggle with communication intensive tasks such as a QR factorization.

Remark 5. The function `orth` that we use as a key building block can most easily be implemented by a call to a library QR factorization routine, as described in section 2.4. We want to emphasize that pivoting is not necessary here, since the "R" factor is never used. This in principle opens up the possibility of building highly efficient implementations of the QR factorizations.

5. A version of the method with enhanced accuracy.

5.1. Randomized sampling of a power of the matrix. The accuracy of the basic randomized approximation scheme described in section 3, and the blocked version of section 4 is well understood. The analysis of [8] (see the synopsis in section 3.2) shows that the error $\|\mathbf{A}-\mathbf{A}_{\mathrm{approx}}\|$ depends strongly on the quantity $(\sum_{j=k+1}^{\min(m,n)} \sigma_j^2)^{1/2}$. This implies that the scheme is highly accurate for matrices whose singular values decay rapidly but less accurate when the "tail" singular values have substantial weight. The problem becomes particularly pronounced for large matrices. Happily, it was demonstrated in [12] that this problem can with modest cost be resolved when given a matrix with slowly decaying singular values by using a so-called power scheme. To be precise, suppose that we are given an $m \times n$ matrix $\mathbf{A}$, a target rank ℓ, and a small

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Downloaded 10/31/16 to 131.211.208.19. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

```
        function $[\mathbf{Q}, \mathbf{B}] = \texttt{randQB\_p}(\mathbf{A}, \ell, P)$
(1)     $\mathbf{\Omega} = \texttt{randn}(n, \ell)$.
(2)     $\mathbf{Q} = \texttt{orth}(\mathbf{A\Omega})$.                 $C_{\rm mm}mn\ell + C_{\rm qr}m\ell^2$
(3)     for $j = 1 : P$
(4)         $\mathbf{Q} = \texttt{orth}(\mathbf{A}^*\mathbf{Q})$.       $C_{\rm mm}mn\ell + C_{\rm qr}m\ell^2$
(5)         $\mathbf{Q} = \texttt{orth}(\mathbf{AQ})$.                  $C_{\rm mm}mn\ell + C_{\rm qr}m\ell^2$
(6)     end for
(7)     $\mathbf{B} = \mathbf{Q}^*\mathbf{A}$                            $C_{\rm mm}mn\ell$
```

FIG. 3. *An accuracy enhanced version of the basic randomized range finder in Figure* 1. *The algorithm takes as input an $m \times n$ matrix $\mathbf{A}$, a rank ℓ, and a "power" P (see section* 5.1*). The output are matrices $\mathbf{Q}$ and $\mathbf{B}$ of sizes $m \times \ell$ and $\ell \times n$ such that $\mathbf{A} \approx \mathbf{QB}$. Higher P leads to better accuracy but also higher cost. Setting $P = 1$ or $P = 2$ is often sufficient.*

integer P (say, $P = 1$ or $P = 2$). Then the following formula will produce an ON matrix $\mathbf{Q}$ whose columns form an approximation to the range:

$$\mathbf{\Omega} = \texttt{randn}(n, \ell) \qquad \text{and} \qquad \mathbf{Q} = \texttt{orth}\big(\big(\mathbf{AA}^*\big)^P \mathbf{A\Omega}, 0\big).$$

The key observation here is that the matrix $\big(\mathbf{AA}^*\big)^P \mathbf{A}$ has exactly the same left singular vectors as $\mathbf{A}$, but its singular values are σ_j^{2P+1} (observe that our objective is to build an ON-basis for the range of $\mathbf{A}$, and the optimal such basis consists of the leading left singular vectors). Even a small value of P will typically provide enough decay that highly accurate results are attained. For a theoretical analysis, see [8, sect. 10.4].

When the "power scheme" idea is to be executed in floating point arithmetic, substantial loss of accuracy happens whenever the singular values of $\mathbf{A}$ have a large dynamic range. To be precise, if $\epsilon_{\rm mach}$ denotes the machine precision, then any singular components smaller than $\sigma_1\, \epsilon_{\rm mach}^{1/(2P+1)}$ will be lost. This problem can be resolved by orthonormalizing the "sample matrix" between each application of $\mathbf{A}$ and $\mathbf{A}^*$. This results in the scheme we call `randQB_p`, as shown in Figure 3. (Note that this scheme is virtually identical to a classical subspace iteration with a random Gaussian matrix as the start [13].)

5.2. The blocked version of the power scheme. A blocked version of `randQB_p` is easily obtained by a process analogous to the one described in section 4.1, resulting in the algorithm "`randQB_pb`" in Figure 4. Line (8) combats the problem of incremental loss of orthonormality when the algorithm is executed in finite precision arithmetic; cf. section 4.3.

5.3. Computational complexity. When comparing the computational cost of `randQB_p` (cf. Figure 3) versus `randQB_pb` (cf. Figure 4), we use the notation that was introduced in section 4.4. By inspection, we directly find that

$$T_{\texttt{randQB\_p}} \sim C_{\rm mm}(2 + 2P)\, mn\ell + C_{\rm qr}(1 + 2P)m\ell^2.$$

For the blocked scheme, inspection tells us that

$$T_{\texttt{randQB\_pb}} \sim \sum_{i=1}^{s} \left[C_{\rm mm}(3 + 2P)\, mnb + 2(i-1)C_{\rm mm}mb^2 + C_{\rm qr}(2 + 2P)mb^2\right].$$

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Downloaded 10/31/16 to 131.211.208.19. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

```
        function $[\mathbf{Q}, \mathbf{B}] = \texttt{randQB\_pb}(\mathbf{A}, \varepsilon, P, b)$
(1)     for $i = 1, 2, 3, \ldots$
(2)         $\mathbf{\Omega}_i = \texttt{randn}(n, b)$.
(3)         $\mathbf{Q}_i = \texttt{orth}(\mathbf{A}\mathbf{\Omega}_i)$.                          $C_{\rm mm} mnb + C_{\rm qr}\, mb^2$
(4)         for $j = 1 : P$
(5)             $\mathbf{Q}_i = \texttt{orth}(\mathbf{A}^*\mathbf{Q}_i)$.                      $C_{\rm mm} mnb + C_{\rm qr} mb^2$
(6)             $\mathbf{Q}_i = \texttt{orth}(\mathbf{A}\mathbf{Q}_i)$.                       $C_{\rm mm} mnb + C_{\rm qr} mb^2$
(7)         end for
(8)         $\mathbf{Q}_i = \texttt{orth}(\mathbf{Q}_i - \sum_{j=1}^{i-1} \mathbf{Q}_j \mathbf{Q}_j^* \mathbf{Q}_i)$    $2(i-1)C_{\rm mm} mb^2 + C_{\rm qr} mb^2$
(9)         $\mathbf{B}_i = \mathbf{Q}_i^* \mathbf{A}$                                     $C_{\rm mm} mnb$
(10)        $\mathbf{A} = \mathbf{A} - \mathbf{Q}_i \mathbf{B}_i$                               $C_{\rm mm} mnb$
(11)        if $\|\mathbf{A}\| < \varepsilon$ then stop
(12)    end while
(13)    Set $\mathbf{Q} = [\mathbf{Q}_1 \ \cdots \ \mathbf{Q}_i]$ and $\mathbf{B} = [\mathbf{B}_1^* \ \cdots \ \mathbf{B}_i^*]^*$.
```

FIG. 4. *A blocked and adaptive version of the accuracy enhanced algorithm shown in Figure* 3. *Its input and output are identical, except that we now provide a tolerance* ε *as an input (instead of a rank), and also a block size* b.

Executing the sum, and utilizing that $\ell = sb$, we get

$$T_{\texttt{randQB\_pb}} \sim C_{\rm mm}(3 + 2P)\, mn\ell + C_{\rm mm} m\ell^2 + \frac{1}{s} C_{\rm qr}(2 + 2P) m\ell^2.$$

In other words, the blocked algorithm again spends slightly more time executing matrix-matrix multiplications and quite a bit less time on QR factorizations. This trade is often favorable, and particularly so when the algorithm is executed on a GPU (cf. Remark 4). On the other hand, when $\ell \ll n$, the benefit to saving time on QR factorizations is minor.

5.4. Is reorthonormalizing truly necessary? Looking at algorithm `randQB_p`, it is very tempting to skip the intermediate QR factorizations and simply execute steps (2)–(6) as

```
(2)    $\mathbf{Y} = \mathbf{A}\mathbf{\Omega}$.
(3)    for $j = 1 : P$
(4)        $\mathbf{Y} = \mathbf{A}\big(\mathbf{A}^*\mathbf{Y}\big)$
(5)    end for
(6)    $\mathbf{Q} = \texttt{orth}(\mathbf{Y})$
```

This simplification does speed things up substantially, but as we mentioned earlier, it can lead to loss of accuracy. In this section we state some conjectures about when reorthonormalization is necessary. These conjectures appear to show that the blocked scheme is much more resilient to skipping reorthonormalization.

To describe the issue, let us fix a (small) integer P and define the matrix

$$\mathbf{A}_P = \big(\mathbf{A}\mathbf{A}^*\big)^P \mathbf{A}.$$

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Downloaded 10/31/16 to 131.211.208.19. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

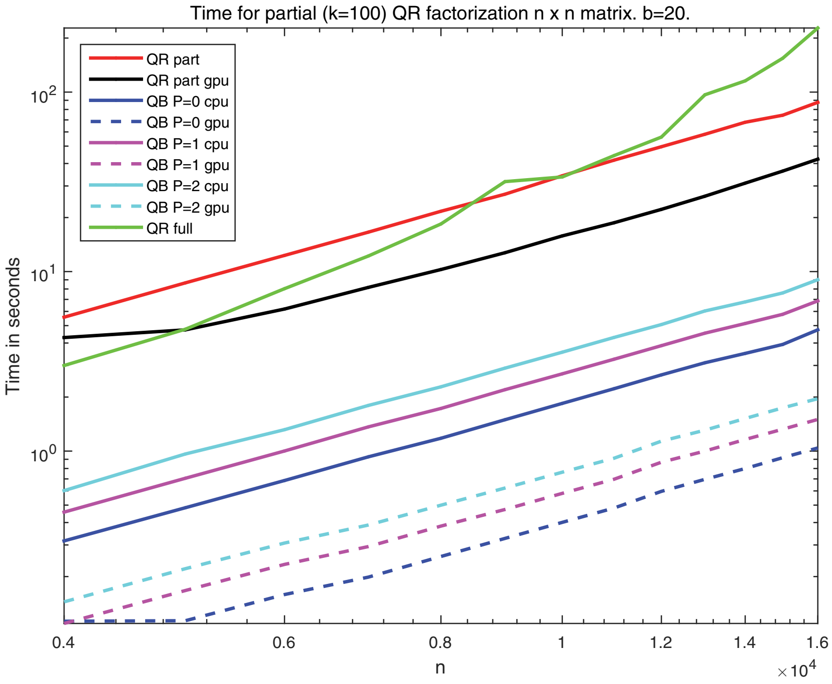

Fig. 5. *Timing results for different algorithms on CPU and GPU. The integer P denotes the parameter in the "power scheme" described in section* 5.

If the SVD of A is $\mathsf{A} = \mathsf{U}\mathbf{\Sigma}\mathsf{V}^*$, then the SVD of A_P is

$$\mathsf{A}_P = \mathsf{U}\mathbf{\Sigma}^{2P+1}\mathsf{V}^*.$$

In computing $\mathsf{Y} = \mathsf{A}_P\mathbf{\Omega}$, we lose all information about any singular value σ_i (and its associated singular vectors) for which $\sigma_i^{2P+1} \leq \sigma_1^{2P+1}\epsilon_{\text{mach}}$, where ϵ_{mach} is the machine precision. In other words, in order to accurately resolve the first k singular modes, reorthogonalization is needed if

$$\frac{\sigma_1}{\sigma_k} > \epsilon_{\text{mach}}^{1/(2P+1)}. \tag{26}$$

As an example, with $P = 2$ and $\epsilon_{\text{mach}} = 10^{-15}$, we find that $\epsilon_{\text{mach}}^{1/(2P+1)} = 10^{-3}$, so reorthonormalization is imperative to resolve any components smaller than $\sigma_1\,10^{-3}$. Moreover, if we skip reorthonormalization, we are likely to see an overall loss of accuracy affecting singular values and singular vectors associated with larger singular values.

Next consider the blocked scheme. The crucial observation is that now, instead of trying to extract the whole range of singular values $\{\sigma_j\}_{j=1}^k$ (and their associated eigenvectors) at once, we now extract them in s groups of b modes each, where $k \approx sb$. This means that we can expect to get reasonable accuracy as long as

$$\frac{\sigma_{(i-1)b+1}}{\sigma_{ib}} \leq \epsilon_{\text{mach}}^{1/(2P+1)} \qquad \text{for } i = 1,\, 2,\, \ldots,\, s. \tag{27}$$

Comparing (26) and (27), we see that (27) is a much milder condition, since the block size b is smaller than k.

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Downloaded 10/31/16 to 131.211.208.19. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

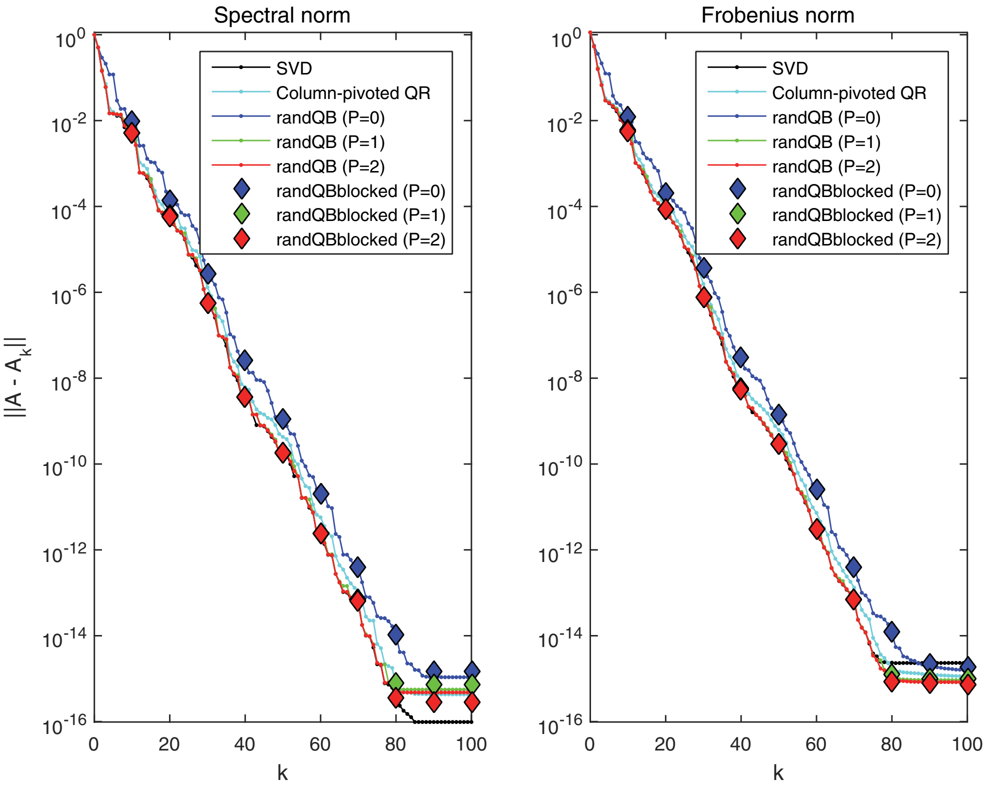

FIG. 6. *Errors for the* 800 × 600 *"Matrix 1" whose singular values decay very rapidly. The block size is* $b = 10$.

All claims in this section are *heuristics*. However, while they have not been rigorously proven, they are supported by extensive numerical experiments; see section 6.3.

6. Numerical experiments.

In this section, we present numerical examples that illustrate the computational efficiency and the accuracy of the proposed scheme, see sections 6.1 and 6.2, respectively. The codes we used are available at http://amath.colorado.edu/faculty/martinss/main_codes.html and we encourage any interested reader to try the methods out and explore different parameter sets than those included here. Additional numerical experiments which did not fit in the journal version of this article can be reviewed in the technical report [11].

6.1. Comparison of execution speeds.

We first compare the run times of different techniques for computing a partial (rank k) QR factorization of a given matrix $\mathbf{A}$ of size $n \times n$. Observe that the choice of matrix is immaterial for a run time comparison (we investigate accuracy in section 6.2). We compared three sets of techniques:

- Truncating a *full* QR factorization, computed using the Intel MKL libraries.
- Taking k steps of a column-pivoted Gram–Schmidt process. The implementation was accelerated by using MKL library functions whenever practicable.
- The blocked "QB" scheme, followed by postprocessing of the factors to obtain a QR factorization. We used the "power method" described in section 5 with parameters $P = 0, 1, 2$.

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Downloaded 10/31/16 to 131.211.208.19. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

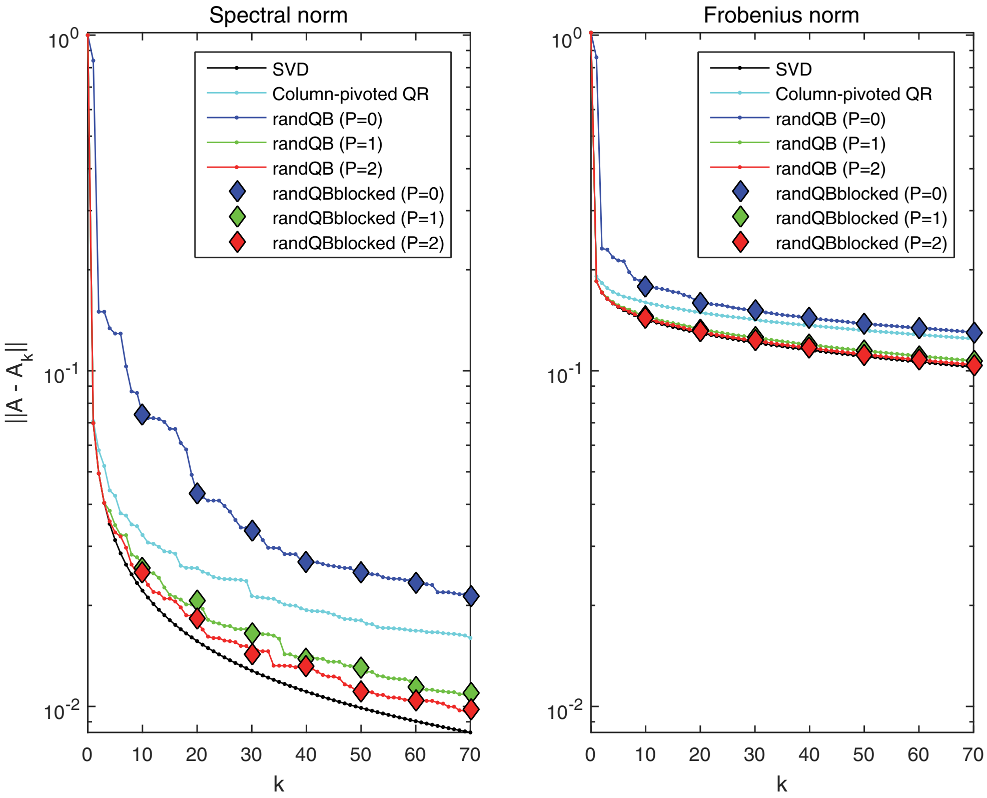

FIG. 7. *Errors for the* 800×600 *"Matrix 2" whose singular values decay slowly. The block size is* $b = 10$.

The algorithms were all implemented in C and run on a desktop with a 6-core Intel Xeon E5-1660 CPU (3.30 GHZ) and 128 GB of RAM. We also ran the blocked "QB" scheme on an NVIDIA Tesla K40c GPU installed on the same machine. The results are shown in Figure 5. Figure 5 shows that our blocked algorithms (blue, magenta, and cyan lines) compare favorably to both of the two benchmarks we chose—full QR using MKL libraries (green) and partial factorization using column pivoting (red). However, it must be noted that our implementation of column-pivoted QR is *far* slower than the built-in QR factorization in the MKL libraries. Even for as low a rank as $k = 100$, we do not break even with a full factorization until $n = 8000$. This implies that column pivoting can be implemented far more efficiently than we were able to. The point is that in order to attain the efficiency of the MKL libraries, very careful coding that is customized to any particular computing platform would have to be done. In contrast, our blocked code is able to exploit the very high efficiency of the MKL libraries with minimal effort. (Observe that in implementing the operation `orth` in our blocked randomized schemes, we did not use our version of column-pivoted QR but rather used the MKL library function for QR factorization.)

Finally, it is worth noting how particularly efficient our blocked algorithms are when executed on a GPU. We gained a substantial integer factor speed-up over CPU speed in every test we conducted.

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Downloaded 10/31/16 to 131.211.208.19. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

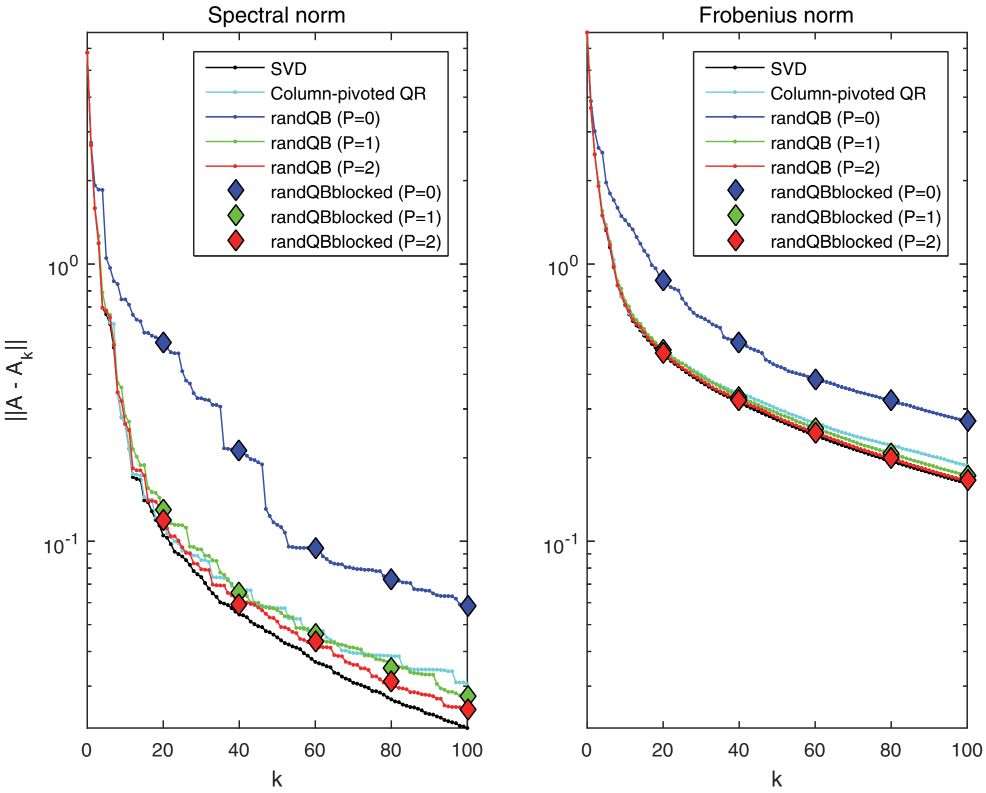

FIG. 8. *Errors for the* 800×600 *"Matrix 3." This is a sparse matrix for which column-pivoted Gram–Schmidt performs exceptionally well. However,* `randQB` *still gives better accuracy whenever a power* $P \geq 1$ *is used.*

Remark 6 (choice of blocking parameter b). The algorithms described rely on library routines to attain high efficiency for operations like matrix-matrix multiplication and QR factorization of small matrices. A key advantage of this setup is that it allows us to exploit the power and easy of use of autotuning that is built into these routines [16]. The blocking parameter b that a user must specify need only be large enough that these library routines have enough data to work with. In other words, while picking b to be very small has a deleterious effect on performance, there is no need to carefully optimize it once it is larger than a (small) threshold.

6.2. Accuracy of the randomized scheme. We next investigate the accuracy of the randomized schemes versus column-pivoted QR on the one hand (easy to compute, not optimal) and versus the truncated SVD on the other (expensive to compute, optimal). We used five classes of test matrices that each have different characteristics:

Matrix 1 (fast decay). Let $\mathbf{A}_1$ denote an $m \times n$ matrix of the form $\mathbf{A}_1 = \mathbf{UDV}^*$, where $\mathbf{U}$ and $\mathbf{V}$ are randomly drawn matrices with orthonormal columns (obtained by performing QR on a random Gaussian matrix), and where $\mathbf{D}$ is a diagonal matrix with entries roughly given by $d_j = g_j^2\,\beta^{j-1}$, where g_j is a random number drawn from a uniform distribution on $[0, 1]$ and $\beta = 0.65$. To precision 10^{-15}, the rank of $\mathbf{A}_1$ is about 75.

Matrix 2 (slow decay). The matrix $\mathbf{A}_2$ is formed just like $\mathbf{A}_1$, but now the diagonal entries of $\mathbf{D}$ decay very slowly, with $d_j = (1 + 200(j-1))^{1/2}$.

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Downloaded 10/31/16 to 131.211.208.19. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

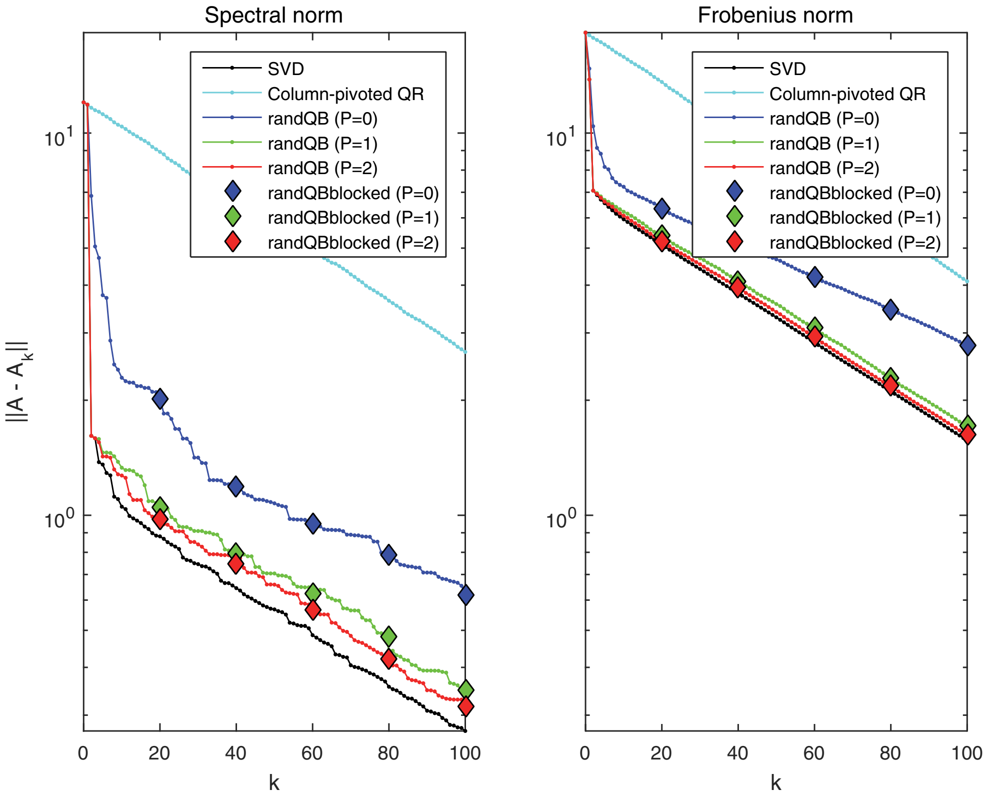

FIG. 9. *Errors for the* 1000 × 1000 *"Matrix 4." This matrix is a variation of the "Kahan counterexample" and is designed specifically to give poor performance for column-pivoted QR. Here* $b = 20$.

Matrix 3 (sparse). The matrix $\mathbf{A}_3$ is a sparse matrix given by $\mathbf{A}_3 = \sum_{j=1}^{10} \frac{2}{j}\, \mathbf{x}_j\, \mathbf{y}_j^* + \sum_{j=11}^{\min(m,n)} \frac{1}{j}\, \mathbf{x}_j\, \mathbf{y}_j^*$ where $\mathbf{x}_j$ and $\mathbf{y}_j$ are random sparse vectors generated by the MATLAB commands $\texttt{sprand}(m, 1, 0.01)$ and $\texttt{sprand}(n, 1, 0.01)$, respectively. We used $m = 800$ and $n = 600$, which resulted in a matrix with roughly 6% nonzero elements. This matrix was borrowed from Sorensen and Embree [14] and is an example of a matrix for which column-pivoted Gram–Schmidt performs particularly well.

Matrix 4 (Kahan). This is a variation of the "Kahan counterexample," which is a matrix designed so that Gram–Schmidt performs particularly poorly. The matrix here is formed via the matrix matrix product $\mathbf{SK}$, where

$$\mathbf{S} = \begin{bmatrix} 1 & 0 & 0 & 0 & \cdots \\ 0 & \zeta & 0 & 0 & \cdots \\ 0 & 0 & \zeta^2 & 0 & \cdots \\ 0 & 0 & 0 & \zeta^3 & \cdots \\ \vdots & \vdots & \vdots & \vdots & \ddots \end{bmatrix} \quad \text{and} \quad \mathbf{K} = \begin{bmatrix} 1 & -\phi & -\phi & -\phi & \cdots \\ 0 & 1 & -\phi & -\phi & \cdots \\ 0 & 0 & 1 & -\phi & \cdots \\ 0 & 0 & 0 & 1 & \cdots \\ \vdots & \vdots & \vdots & \vdots & \ddots \end{bmatrix}$$

with random $\zeta, \phi > 0$, $\zeta^2 + \phi^2 = 1$. Then $\mathbf{SK}$ is upper triangular, and for many choices of ζ and ϕ, classical column pivoting will yield poor performance as the different column norms will be similar and pivoting will generally fail. The

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Downloaded 10/31/16 to 131.211.208.19. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

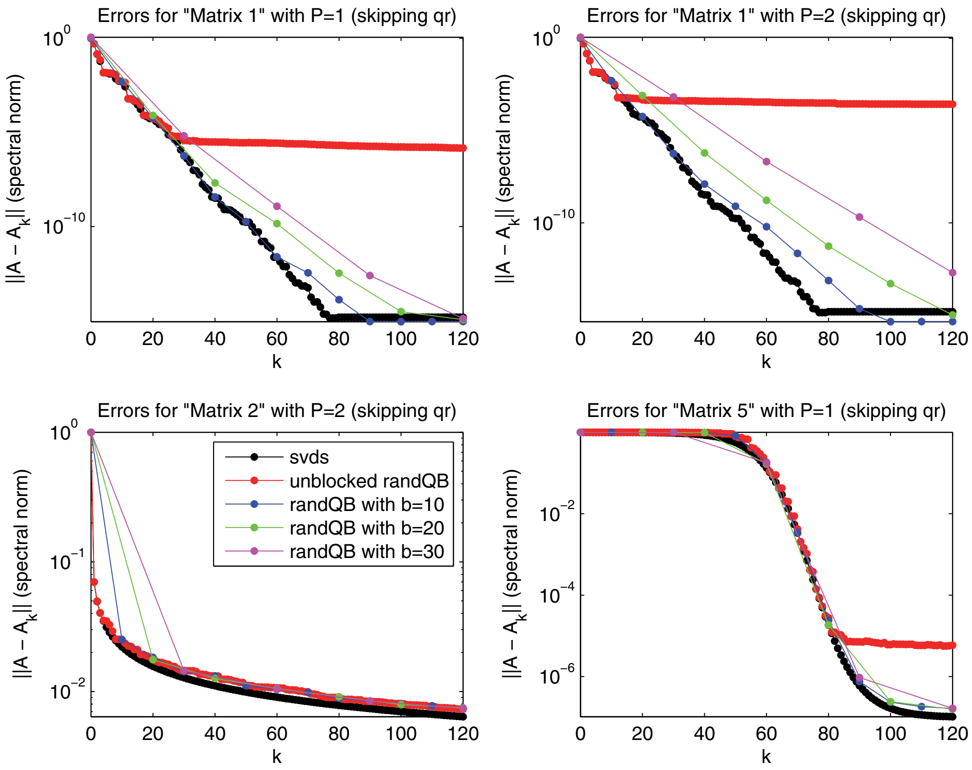

Fig. 10. *Errors incurred when not reorthonormalizing between applications of* $\mathbf{A}$ *and* $\mathbf{A}^*$ *in the "power method"; cf. sections 5.4 and 6.3. The nonblocked scheme (red) performs precisely as predicted and cannot resolve anything beyond precision* 10^{-5} *when* $P = 1$ *and* 10^{-3} *when* $P = 2$*. The blocked version converges slightly slower when skipping reorthonormalization but always reaches full precision.*

rank-k approximation resulting from column-pivoted QR is substantially less accurate than the optimal rank-k approximation resulting from truncating the full SVD [6]. However, we obtain much better results than QR with the QB algorithm.

We compare four different techniques for computing a rank-k approximation to our test matrices:

SVD. We computed the full SVD (using the MATLAB command `svd`) and then truncated to the first k components.

Column-pivoted QR. We implemented this using modified Gram–Schmidt with reorthogonalization to ensure that orthonormality is strictly maintained in the columns of $\mathbf{Q}$.

`randQB`—single vector. This is the greedy algorithm labeled "Algorithm 1" in section 1.2, implemented with $\mathbf{q}_j$ on line (4) chosen as $\mathbf{q}_j = \mathbf{y}/\|\mathbf{y}\|$, where $\mathbf{y} = \left(\mathbf{A}\mathbf{A}^*\right)^P \mathbf{A}\,\boldsymbol{\omega}$ and where $\boldsymbol{\omega}$ is a random Gaussian vector.

`randQB`—blocked. This is the algorithm `randQB_pb` shown in Figure 4.

The results are shown in Figures 6–9. We make three observations: (1) When the "power method" described in section 5 is used, the accuracy of `randQB_pb` exceeds that of column-pivoted QR in every example we tried, even for as low a power as

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Downloaded 10/31/16 to 131.211.208.19. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

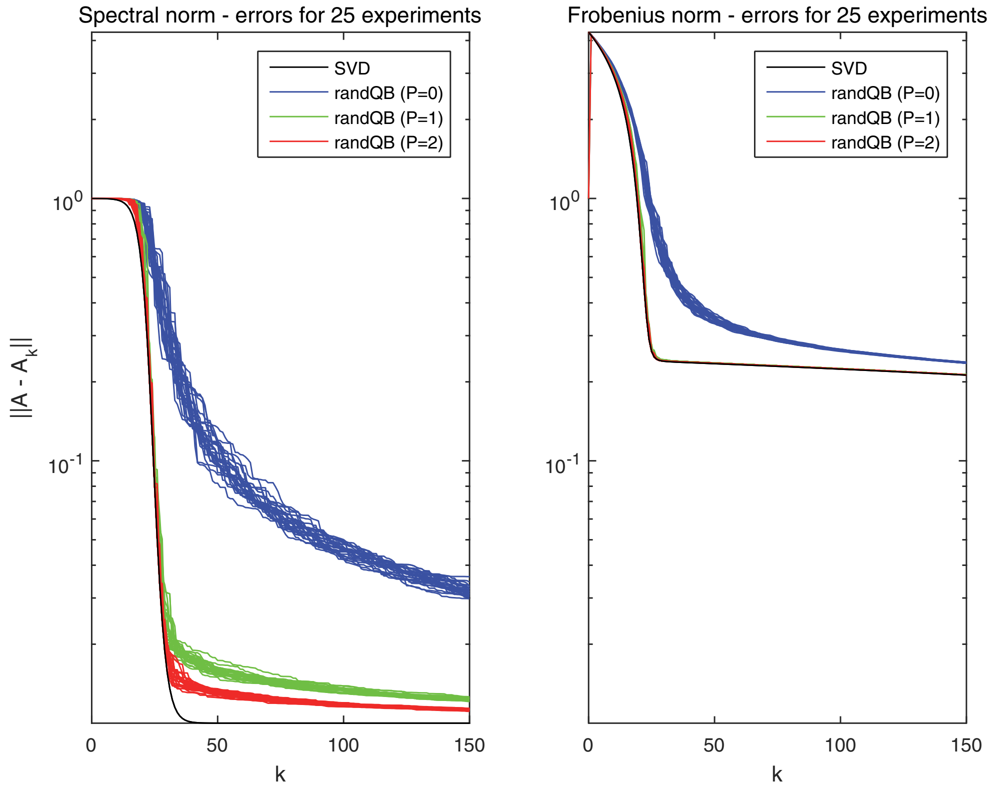

FIG. 11. *The error paths for* 25 *instantiations of the randomized factorization algorithm applied to Matrix* 5.

$P = 1$. (2) Blocking appears to lead to no loss of accuracy. In most cases, there is no discernible difference in accuracy between the blocked and the nonblocked versions. (3) The accuracy of `randQB_pb` is particularly good when errors are measured in the Frobenius norm. In almost all cases we investigated, essentially optimal results are obtained even for $P = 1$. (All results shown in this section refers to a single instantiation of the randomized algorithm. See section 6.4 for statistical properties.)

6.3. When reorthonormalization is required. We claimed in section 5.4 that the blocked scheme is more robust toward loss of orthonormality than the nonblocked scheme presented in [8]. To examine this hypothesis, we tested what happens if we skip the reorthonormalization between applications of A and A^* in the algorithms shown in Figures 3 and 4. The results are shown in Figure 10. The key observation here is that the blocked versions of `randQB` *still* always yield excellent precision. When the block size is large, the convergence is slowed down a bit compared to the more meticulous implementation, but essentially optimal accuracy is nevertheless obtained relatively quickly.

Remark 7. The numerical results in Figure 10 substantiate the claim that for the unblocked version, the best accuracy attainable is $\sigma_1\, \epsilon_{\rm mach}^{1/(2P+1)}$. In all examples, we have $\sigma_1 = 1$, so the prediction is that for $P = 1$ the maximum precision is $\left(10^{-15}\right)^{1/3} = 10^{-5}$ and for $P = 2$ it is $\left(10^{-15}\right)^{1/5} = 10^{-3}$. The results shown precisely follow this pattern. Observe that for A_2, no loss of accuracy is seen at all since the singular values we are interested in level out at about 10^{-2}.

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Downloaded 10/31/16 to 131.211.208.19. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

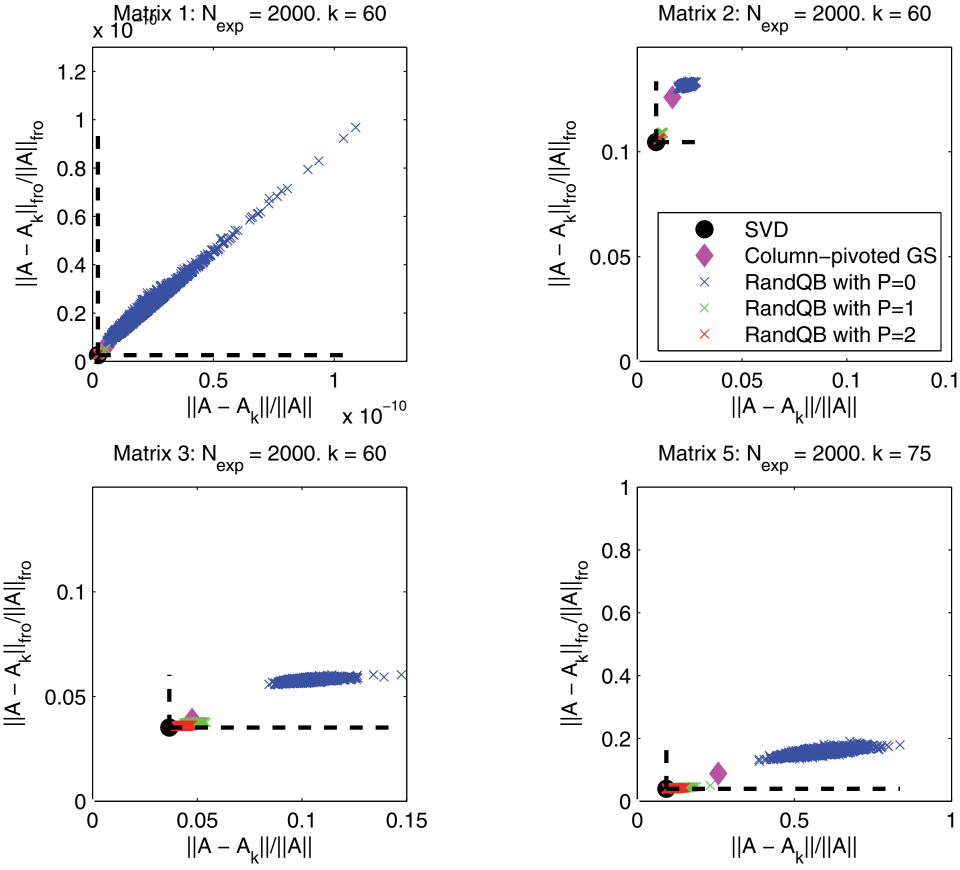

FIG. 12. *Each blue cross in the graphs represents one instantiation of the randomized blocked algorithm. The x- and y-coordinates show the relative errors in the spectral and Frobenius norms, respectively. For reference, we also include the error from classical column-pivoted Gram–Schmidt (the magenta diamond) and the error incurred by the truncated SVD. The dashed lines are the horizonal and vertical lines cutting through the point representing the SVD—since these errors are minimal, every other dot must be located above and to the right of these lines.*

6.4. Distribution of errors. The output of our randomized blocked approximation algorithms is a random variable, since it depends on the drawing of a Gaussian matrix $\mathbf{\Omega}$. It has been proven (see, e.g., [8]) that due to concentration of mass, the variation in this random variable is tiny. The output is for practical purposes always very close to the expectation of the output. For this reason, when we compared the accuracy of the randomized method to classical methods in section 6.2, we simply presented the results from one particular draw of $\mathbf{\Omega}$. In this section, we investigate numerically how much spread is incurred for the following example:

Matrix 5 (S shaped decay). The matrix A_5 is built in the same manner as A_1 and A_2, but now the diagonal entries of D are chosen to first hover around 1, then decay rapidly, and then level out at a relatively high plateau; cf. Figure 11.

Figure 11 shows the error paths from 25 different instantiations of the randomized algorithm. We observe that the errors are tightly clustered, in particular for $P = 1$ and $P = 2$. We also observe that the clustering is stronger in the Frobenius norm than in the spectral norm.

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Downloaded 10/31/16 to 131.211.208.19. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

For our final experiment, we tested how the algorithm performs over 2000 instantiations, applied to matrices 1, 2, 3, and 5. To keep the plots legible, we plot the errors only for a fixed value of k; see Figure 12. These experiments further substantiate our claim that the results are tightly clustered, in particular when $P \geq 1$, and when errors are measured in the Frobenius norm.

7. Concluding remarks. We have described a randomized algorithm for the low-rank approximation of matrices. The algorithm is based on the randomized sampling paradigm described in [9, 12, 8, 10]. In this article, we introduce a *blocking* technique, which allows us to incorporate adaptive rank determination without sacrificing computational efficiency, and an *updating* technique that allows us to replace the randomized stopping criterion proposed in [8] with a deterministic one. Through theoretical analysis and numerical examples, we demonstrate that while the blocked scheme is mathematically equivalent to the nonblocked scheme of [9, 12, 8, 10] when executed in exact arithmetic, the blocked scheme is slightly more robust toward accumulation of round-off errors.

The updating strategy that we propose is directly inspired by a classical scheme for computing a partial QR factorization via the column-pivoted Gram–Schmidt process. We demonstrate that the randomized version that we propose is more computationally efficient than this classical scheme (since it is hard to block the column pivoting scheme). Our numerical experiments indicate that the randomized version not only improves speed but also leads to higher accuracy. In fact, in all examples we present, the errors resulting from the blocked randomized scheme are very close to the optimal error obtained by truncating a full SVD. In particular, when errors are measured in the Frobenius norm, there is almost no loss of accuracy at all compared to the optimal factorization, even for matrices whose singular values decay slowly.

The scheme described can output any of the standard low-rank factorizations of matrices such as, e.g., a partial QR or SVD factorization. It can also with ease produce less standard factorizations such as the CUR and interpolative decompositions; cf. section 3.3.

In this manuscript, we presented a basic version of the blocked scheme that is easy to implement either using a high-level language such as MATLAB or Python or using standard library functions in BLAS, LAPACK, etc.

REFERENCES

[1] Å. Björck, *Numerics of Gram-Schmidt orthogonalization*, Linear Algebra Appl., 197/198 (1994), pp. 297–316.

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

[3] J. Demmel, L. Grigori, M. Hoemmen, and J. Langou, *Communication-optimal parallel and sequential QR and LU factorizations*, SIAM J. Sci. Comput., 34 (2012), pp. A206–A239.

[4] C. Eckart and G. Young, *The approximation of one matrix by another of lower rank*, Psychometrika, 1 (1936), pp. 211–218.

[5] G. H. Golub and C. F. Van Loan, *Matrix Computations*, 4th ed., Johns Hopkins Stud. Math. Sci., Johns Hopkins University Press, Baltimore, MD, 2013.

[6] 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.

[7] F. G. Gustavson, *Cache blocking for linear algebra algorithms*, in Parallel Processing and Applied Mathematics, Springer, New York, 2012, pp. 122–132.

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

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Downloaded 10/31/16 to 131.211.208.19. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

[9] P.-G. Martinsson, V. Rokhlin, and M. Tygert, *A Randomized Algorithm for the Approximation of matrices*, Yale CS research report YALEU/DCS/RR-1361, Computer Science Department, Yale University, New Haven, CT, 2006.

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

[11] P. G. Martinsson and S. Voronin, *A randomized blocked algorithm for efficiently computing rank-revealing factorizations of matrices*, arXiv:1503.07157v2, 2015.

[12] V. Rokhlin, A. Szlam, and M. Tygert, *A randomized algorithm for principal component analysis*, SIAM J. Matrix Anal. Appl., 31 (2009), pp. 1100–1124.

[13] Y. Saad, *Overview of Krylov subspace methods with applications to control problems*, in Signal Processing, Scattering and Operator Theory, and Numerical Methods (Amsterdam, 1989), Progr. Systems Control Theory 5, Birkhäuser Boston, Boston, MA, 1990, pp. 401–410.

[14] D. C. Sorensen and M. Embree, *A DEIM induced CUR factorization*, preprint, arXiv:1407.5516, 2014.

[15] S. Voronin and P.-G. Martinsson, *A CUR factorization algorithm based on the interpolative decomposition*, arXiv.1412.8447, 2014.

[16] R. C. Whaley and J. J. Dongarra, *Automatically tuned linear algebra software*, in Proceedings of the 1998 ACM/IEEE Conference on Supercomputing, Washington, DC, 1998, IEEE Computer Society, pp. 1–27.

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.