SIAM J. MATRIX ANAL. APPL.
Vol. 39, No. 3, pp. 1339–1359

EFFICIENT RANDOMIZED ALGORITHMS FOR THE FIXED-PRECISION LOW-RANK MATRIX APPROXIMATION*

WENJIAN YU†, YU GU‡, AND YAOHANG LI§

Abstract. Randomized algorithms for low-rank matrix approximation are investigated, with the emphasis on the fixed-precision problem and computational efficiency for handling large matrices. The algorithms are based on the so-called QB factorization, where $\boldsymbol{Q}$ is an orthonormal matrix. First, a mechanism for calculating the approximation error in the *Frobenius norm* is proposed, which enables efficient adaptive rank determination for a large and/or sparse matrix. It can be combined with any QB-form factorization algorithm in which $\boldsymbol{B}$'s rows are incrementally generated. Based on the blocked randQB algorithm by Martinsson and Voronin, this results in an algorithm called `randQB_EI`. Then, we further revise the algorithm to obtain a pass-efficient algorithm, `randQB_FP`, which is mathematically equivalent to the existing randQB algorithms and also suitable for the fixed-precision problem. Especially, `randQB_FP` can serve as a single-pass algorithm for calculating leading singular values, under a certain condition. With large and/or sparse test matrices, we have empirically validated the merits of the proposed techniques, which exhibit remarkable speedup and memory saving over the blocked randQB algorithm. We have also demonstrated that the single-pass algorithm derived by `randQB_FP` is much more accurate than an existing single-pass algorithm. And with data from a scenic image and an information retrieval application, we have shown the advantages of the proposed algorithms over the adaptive range finder algorithm for solving the fixed-precision problem.

Key words. adaptive rank determination, randomized algorithm, low-rank matrix approximation, pass-efficient algorithm, fixed-precision problem

AMS subject classifications. 15A18, 65F30, 65F15, 68W20, 60B20

DOI. 10.1137/17M1141977

1. Introduction.

Low-rank matrix factorizations, like the partial singular value decomposition (SVD) and the rank-revealing QR factorization, play a crucial role in data analysis and scientific computing. In recent years, techniques based on randomization have been investigated for performing the computation and low-rank factorization of large matrices [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]. They involve the same or fewer floating-point operations (*flops*) than classical algorithms and are more efficient by exploiting modern computing architectures.

A basic idea of the randomized techniques is using random projection to approximate the dominant subspace of a matrix. For an $m \times n$ matrix $\boldsymbol{A}$, suppose the orthogonal basis vectors of this approximate subspace form an $m \times k$ orthonormal matrix $\boldsymbol{Q}$. Then, we have [1, 2]

$$\boldsymbol{A} \approx \boldsymbol{Q}\boldsymbol{B}, \tag{1}$$

where $\boldsymbol{B}$ is a $k \times n$ matrix, and

$$\boldsymbol{B} = \boldsymbol{Q}^T\boldsymbol{A}. \tag{2}$$

*Received by the editors August 3, 2017; accepted for publication (in revised form) by P. Drineas July 17, 2018; published electronically August 30, 2018.
http://www.siam.org/journals/simax/39-3/M114197.html
Funding: This work was partially supported by the NSFC under grants 61872206, 6172811, and NSF under grant 1066471.

†BNRist, Department of Computer Science and Technology, Tsinghua University, Beijing 100084, China (yu-wj@tsinghua.edu.cn).
‡Department of Computer Science and Technology and Institute for Interdisciplinary Information Sciences, Tsinghua University, Beijing 100084, China (guyu13@mails.tsinghua.edu.cn).
§Department of Computer Science, Old Dominion University, Norfolk, VA 23529 (yaohang@cs.odu.edu).

The basic randQB algorithm

Input: $\boldsymbol{A},\ k,\ s$.
Output: $\boldsymbol{Q},\ \boldsymbol{B}$.
(1) $\boldsymbol{\Omega}$= randn$(n,\ k+s)$
(2) $\boldsymbol{Q}$ = orth$(\boldsymbol{A\Omega})$
(3) $\boldsymbol{B} = \boldsymbol{Q}^T\boldsymbol{A}$

(a) The `randQB` algorithm

The blocked randQB algorithm

Input: $\boldsymbol{A},\ \varepsilon,\ b$.
Output: $\boldsymbol{Q},\ \boldsymbol{B}$.
(1) **for** $i = 1, 2, 3, \cdots$
(2) $\quad\boldsymbol{\Omega}_i$= randn$(n, b)$
(3) $\quad\boldsymbol{Q}_i$ = orth$(\boldsymbol{A\Omega}_i)$
(4) $\quad\boldsymbol{Q}_i$ = orth$(\boldsymbol{Q}_i - \sum_{j=1}^{i-1}\boldsymbol{Q}_j\boldsymbol{Q}_j^T\boldsymbol{Q}_i)$
(5) $\quad\boldsymbol{B}_i = \boldsymbol{Q}_i^T\boldsymbol{A}$
(6) $\quad\boldsymbol{A} = \boldsymbol{A} - \boldsymbol{Q}_i\boldsymbol{B}_i$
(7) $\quad$**if** $\|\boldsymbol{A}\| < \varepsilon$ **then stop**
(8) **end for**
(9) $\boldsymbol{Q} = [\boldsymbol{Q}_1\ \cdots\ \boldsymbol{Q}_i];\ \boldsymbol{B} = [\boldsymbol{B}_1^T\ \cdots\ \boldsymbol{B}_i^T]^T$.

(b) The `randQB_b` algorithm

FIG. 1. *The randomized algorithms for the QB factorization.*

Standard factorizations, e.g., SVD, can be further performed on the smaller matrix $\boldsymbol{B}$, to obtain the low-rank factorizations of $\boldsymbol{A}$.

The approximation presented by (1) and (2) can also be regarded as a kind of low-rank factorization of $\boldsymbol{A}$, called *QB factorization* or *QB approximation* in this work. In [2], a basic randomized scheme for computing the QB approximation was presented, as shown in Figure 1(a). For producing close to optimal rank-k approximation, the oversampling scheme using a random Gaussian matrix $\boldsymbol{\Omega}$ with $k+s$ columns is employed, where s is a small integer. We use `randQB` to denote this algorithm.

Usually, the problem of low-rank matrix approximation falls into two categories:

- the *fixed-rank* problem, where the rank parameter k is given,
- the *fixed-precision* problem, where we seek $\boldsymbol{Q}$ and $\boldsymbol{B}$ with as small as possible size such that $\|\boldsymbol{A} - \boldsymbol{QB}\| < \varepsilon$, where ε is a given accuracy tolerance.

A blocked variant of the `randQB` algorithm proposed in [1], i.e., the `randQB_b` algorithm in Figure 1(b), is suitable for the fixed-precision problem. It incrementally builds the factors $\boldsymbol{Q}$ and $\boldsymbol{B}$ based on the combination of the `randQB` algorithm and the *blocked* Gram–Schmidt scheme and measures the approximation error by explicitly maintaining the residual matrix. However, it is inefficient or even fails for handling a large matrix, because maintaining the residual matrix is costly in runtime and memory usage.

In this work, the randomized algorithms for the fixed-precision problem are investigated considering their adaptability to large and/or sparse matrices. First, a mechanism is proposed for calculating the error of QB approximation in the *Frobenius norm* during the iterative process of building $\boldsymbol{Q}$ and $\boldsymbol{B}$. It does not require maintaining the residual matrix (or updating matrix $\boldsymbol{A}$) and thus avoids fill-in while handling a sparse $\boldsymbol{A}$. This mechanism is also applicable to other iterative computing procedures, e.g., the `trQRCP` algorithm [14]. Second, the algorithm is further revised to largely reduce the number of passes over matrix $\boldsymbol{A}$, in order to adapt the scenarios where the cost of accessing matrix $\boldsymbol{A}$ is expensive. These techniques result in two algorithms called `randQB_EI` and `randQB_FP`, which inherit the merits of `randQB`/`randQB_b` algorithms and have extra benefits. Numerical experiments are carried out on a multicore computer to validate the efficiency and accuracy of the proposed algorithms for handling large or sparse matrices in practical scenarios. The results show that our `randQB_EI` and `randQB_FP` algorithms have up to 3X speedup and 3X memory

saving over an implementation of the `randQB_b` algorithm for dense matrices. They also exhibit up to 22X speedup over the implementation of the `randQB_b` algorithm for sparse matrices. Compared with the single-pass algorithm and the adaptive randomized range finder in [2], the proposed algorithms exhibit much better accuracy or avoid the large overestimation of the sizes of $\boldsymbol{Q}$ and $\boldsymbol{B}$. For reproducibility, we have shared the MATLAB codes of the proposed algorithms and experimental data on https://github.com/WenjianYu/randQB_auto.

2. Technical preliminaries. This section summarizes the background we need for presenting the proposed techniques. Throughout the paper, we measure vectors with their Euclidean norm. Two kinds of matrix norm are usually considered: Frobenius norm and spectral norm (l_2-norm). The spectral norm of a matrix is relatively difficult to calculate, though it is often more informative for noisy data [23]. The Frobenius norm $\|\boldsymbol{A}\|_{\mathrm{F}} = (\sum_{i,j} |\boldsymbol{A}(i,j)|^2)^{1/2}$ is easier for calculation and thus more widely used in data analysis and machine learning applications [10]. We measure matrices with their Frobenius norm by default. We also assume that all matrices are real valued, although the generalization to complex matrices is of no difficulty.

2.1. Randomized algorithms. To produce a rank-k factorization for an $m \times n$ matrix $\boldsymbol{A}$, with the basic randQB algorithm in Figure 1(a) one obtains an $m \times l$ orthonormal matrix $\boldsymbol{Q}$ and an $l \times n$ matrix $\boldsymbol{B}$, where $l > k$ due to oversampling. With this QB approximation, the standard factorizations can be efficiently computed. For example, the standard SVD algorithm can be performed on $\boldsymbol{B}$, which results in $\boldsymbol{B} = \tilde{\boldsymbol{U}}\tilde{\boldsymbol{\Sigma}}\tilde{\boldsymbol{V}}^T$. Then,

$$\boldsymbol{A} \approx \boldsymbol{Q}\boldsymbol{B} = \boldsymbol{Q}\tilde{\boldsymbol{U}}\tilde{\boldsymbol{\Sigma}}\tilde{\boldsymbol{V}}^T. \tag{3}$$

The first k columns of matrices $\boldsymbol{Q}\tilde{\boldsymbol{U}}$ and $\tilde{\boldsymbol{V}}$ and the $k \times k$ upper-left submatrix of $\tilde{\boldsymbol{\Sigma}}$ approximate the rank-k SVD factors of $\boldsymbol{A}$. Similarly, by changing the factorizations made on $\boldsymbol{B}$, one obtains the approximate QR factorization and CUR factorization, etc. [1, 4]. Notice that the accurate truncated SVD provides the optimal low-rank approximation [15]. However, computing accurate SVD of a large matrix is costly, and in many applications the optimality is not necessary. It is thus acceptable to compute the approximate low-rank factorizations for gains in computational efficiency.

The approximation error of the randomized algorithm is a random variable. The authors of [2] have studied the properties of the error in terms of spectral and Frobenius norms and given the bounds on their expectation and variance.

The truncated QR factorization with column pivoting can also be used for low-rank matrix approximation [11, 12]. Classical pivoted QR factorization has the disadvantage that it is hard to be parallelized or to take usage of BLAS-3 operation. Recently, the pivoted QR factorization was largely accelerated through utilizing a randomized technique, which achieves the efficiency comparable to the unpivoted QR decomposition [13, 14]. This makes the truncated pivoted QR factorization competitive for low-rank approximation. Notice that QR factorization can be regarded as a special case of the QB factorization. Therefore, one of the techniques proposed here (cf. section 3) could benefit the solution of the fixed-precision problem based on QR factorizations, as well.

The major computation of the `randQB` algorithm lies at the multiplication of $\boldsymbol{A}$ and $\boldsymbol{\Omega}$. If $\boldsymbol{A}$ is sparse or a structured matrix (often *implicitly* defined) for which matrix-vector products can be rapidly evaluated, the cost of multiplication can be

largely reduced (even to $O(m+n)$ flops). The implicitly defined structured matrix often arises from physical problems, such as a discretized integral operator applied via the fast multipole method and similar techniques [24] and is sometimes referred to as an *implicit sparse matrix*.

For the fixed-precision problem, an adaptive randomized range finder was proposed in [2]. It employs the incremental sampling approach with a probabilistic error estimator to determine the size of $\boldsymbol{Q}$ and $\boldsymbol{B}$. It is based on the statement that

$$\|\boldsymbol{A}-\boldsymbol{Q}\boldsymbol{Q}^T\boldsymbol{A}\|_2 \le 10\sqrt{\frac{2}{\pi}}\max_{i=1,\dots,r}\|(\boldsymbol{A}-\boldsymbol{Q}\boldsymbol{Q}^T\boldsymbol{A})\boldsymbol{\omega}^{(i)}\| \tag{4}$$

with probability at least $1-10^{-r}$ [2]. Here $\|\cdot\|_2$ stands for the spectral norm, $\boldsymbol{\omega}^{(i)}$ is a random vector, and r is a small integer, e.g., $r=10$. However, the error estimator often overestimates the approximation error, yielding much larger output matrices than what is necessary.

The `randQB_b` algorithm in [1] is based on the single-vector version of `randQB` algorithm with the Gram–Schmidt procedure, which allows one to construct the QB factorization and to evaluate its error step by step. In order to exploit *blocking* to attain high performance of linear algebraic computation, the algorithm is then converted to the blocked randQB algorithm in Figure 1(b), where step (6) is for calculating the residual matrix. Therefore, the `randQB_b` algorithm allows a precise error calculation for the fixed-precision problem. Notice that in step (4) there is a reorthogonalization operation which eases the accumulation of round-off error under floating point arithmetic.

The blocked randQB algorithm is able to produce $\boldsymbol{Q}$ and $\boldsymbol{B}$ in smaller sizes than the adaptive randomized range finder. However, explicitly maintaining the residual matrix brings extra time and memory cost if a large matrix $\boldsymbol{A}$ is handled. This disadvantage becomes more serious if $\boldsymbol{A}$ is also sparse, because the fill-in phenomena leads to a dense residual matrix.

In order to reveal the difference among the relevant randomized algorithms, we give a brief comparison of them for the fixed-rank problem, presented as Table 1. `svds` denotes the MATLAB built-in command for truncated SVD [21], which is based on a Krylov subspace iterative method. `trQRCP` denotes the randomized pivoted QR factorization [14]. `randQB_FP` is one of the contributions in this paper, which is a pass-efficient algorithm (cf. section 4). To depict the performance of the algorithms in the situation where the cost of accessing matrix entries is expensive (e.g., $\boldsymbol{A}$ are stored in slow memory) [2], we include the number of passes over matrix $\boldsymbol{A}$ in Table 1. From the table, we see that the `randQB_FP` algorithm inherits the merits of the `randQB` algorithm and is also suitable for the fixed-precision problem.

TABLE 1
Properties of the randomized algorithms for the fixed-rank matrix approximation. (P is a small integer for power scheme, and b is the block size.)

	`svds`	`randQB`	`randQB_b`	`trQRCP`	`randQB_FP`
Computational efficiency	low	high	high	high	high
Adaptive rank determination	no	no	yes	yes[†]	yes
Number of passes over matrix	αk[*]	2 or 2+2P	$4k/b$	$3k/b+1$	1 or 1+2P[‡]

[*] α is a number larger than 1.
[†] This was not mentioned in [14]. For more detail, please see Remark 3.2.
[‡] Suppose matrix $\boldsymbol{A}$ is stored in the row-major format.

3. An efficient Frobenius-norm error indicator and its application. In this section, we first propose an error indicator for measuring the approximation error in the Frobenius norm and an efficient framework for solving the fixed-precision problem of QB factorization. Then, the `randQB_EI` algorithm is derived based on the blocked randQB algorithm. Finally, the accuracy and validity of the error indicator in floating point arithmetic is discussed.

3.1. An error indicator. We first give a theorem regarding the Frobenius norm of the error of the QB approximation.

THEOREM 1. *Let $\boldsymbol{A}$ be an $m \times n$ matrix. $\boldsymbol{Q}$ denotes an $m \times k$ orthonormal matrix ($k < m$), and $\boldsymbol{B} = \boldsymbol{Q}^T\boldsymbol{A}$. Then,*

$$\|\boldsymbol{A} - \boldsymbol{Q}\boldsymbol{B}\|_{\mathrm{F}}^2 = \|\boldsymbol{A}\|_{\mathrm{F}}^2 - \|\boldsymbol{B}\|_{\mathrm{F}}^2. \tag{5}$$

Proof. Due to the property of the Frobenius norm, we know for any matrix $\boldsymbol{M}$,

$$\|\boldsymbol{M}\|_{\mathrm{F}}^2 = tr(\boldsymbol{M}^T\boldsymbol{M}), \tag{6}$$

where $tr(\cdot)$ calculates the trace of a matrix. Because $\boldsymbol{Q}$ is orthonormal and $\boldsymbol{B} = \boldsymbol{Q}^T\boldsymbol{A}$,

$$\begin{aligned}(\boldsymbol{A} - \boldsymbol{Q}\boldsymbol{B})^T(\boldsymbol{A} - \boldsymbol{Q}\boldsymbol{B}) &= (\boldsymbol{A} - \boldsymbol{Q}\boldsymbol{Q}^T\boldsymbol{A})^T(\boldsymbol{A} - \boldsymbol{Q}\boldsymbol{Q}^T\boldsymbol{A}) \\ &= \boldsymbol{A}^T\boldsymbol{A} - 2\boldsymbol{A}^T\boldsymbol{Q}\boldsymbol{Q}^T\boldsymbol{A} + \boldsymbol{A}^T\boldsymbol{Q}\boldsymbol{Q}^T\boldsymbol{Q}\boldsymbol{Q}^T\boldsymbol{A} \\ &= \boldsymbol{A}^T\boldsymbol{A} - \boldsymbol{A}^T\boldsymbol{Q}\boldsymbol{Q}^T\boldsymbol{A} \\ &= \boldsymbol{A}^T\boldsymbol{A} - \boldsymbol{B}^T\boldsymbol{B}.\end{aligned} \tag{7}$$

Now, applying the trace operation to both sides of (7), and according to (6), we obtain (5). ∎

Theorem 1 suggests that, if we have access to $\boldsymbol{B}$ and $\|\boldsymbol{A}\|_{\mathrm{F}}$ is known a priori, we can calculate the error of QB approximation without referring to the residual matrix. This leads to a framework for solving the fixed-precision problem, presented as Algorithm 1. It suits any algorithm that incrementally generates the rows of $\boldsymbol{B}$, including `randQB_b`, `trQRCP`, and the `randQB_FP` algorithm presented in section 4. Below we prove the correctness of Algorithm 1.

Algorithm 1 A framework for solving the fixed-precision QB factorization problem

Input: an $m \times n$ matrix $\boldsymbol{A}$; desired accuracy tolerance ε.
Output: $\boldsymbol{Q}$, $\boldsymbol{B}$, such that $\|\boldsymbol{A} - \boldsymbol{Q}\boldsymbol{B}\|_{\mathrm{F}} < \varepsilon$.

```
Q = [ ]; B = [ ];                          # empty matrices
E = ||A||_F^2                              # initialization of the error indicator
for i = 1, 2, 3, ..., do
    Generate Q_i and B_i, s.t. [Q, Q_i] is orthonormal and B_i = Q_i^T A.
    Q = [Q, Q_i]
    B = [ B   ]
        [ B_i ]
    E = E - ||B_i||_F^2                    # update the error indicator
    if E < ε^2 then stop
end for
```

THEOREM 2. *After the ith iteration of loop in Algorithm* 1 *is executed,*

$$E^{(i)} = \|\boldsymbol{A} - \boldsymbol{Q}^{(i)}\boldsymbol{B}^{(i)}\|_{\mathrm{F}}^2, \tag{8}$$

where $E^{(i)}$, $\boldsymbol{Q}^{(i)}$, *and* $\boldsymbol{B}^{(i)}$ *denote the values of* E, $\boldsymbol{Q}$, *and* $\boldsymbol{B}$ *after the ith iteration of the loop is executed, respectively.*

Proof. Based on step 7 of Algorithm 1 and the property of the Frobenius norm,

$$E^{(i)} = \|\boldsymbol{A}\|_{\mathrm{F}}^2 - \sum_{j=1}^{i} \|\boldsymbol{B}_j\|_{\mathrm{F}}^2 = \|\boldsymbol{A}\|_{\mathrm{F}}^2 - \|\boldsymbol{B}^{(i)}\|_{\mathrm{F}}^2. \tag{9}$$

Because $\boldsymbol{Q}^{(i)}$ is an orthonormal matrix and $\boldsymbol{B}^{(i)} = \boldsymbol{Q}^{(i)T}\boldsymbol{A}$, (8) can be obtained by applying Theorem 1. ∎

From Theorem 2, we see that E in Algorithm 1 equals to the square of the approximation error. It is an error indicator updated through calculating the Frobenius norm of $\boldsymbol{B}_i$. This yields two benefits: we no longer need to maintain the residual $\boldsymbol{A} - \boldsymbol{QB}$, and the approximation error can be calculated with very small cost.

Steps 7 and 8 in Algorithm 1 can be replaced by a row-by-row calculation scheme.

```
7a: for j = 1, 2, ..., m_i do      # m_i denotes the number of rows of B_i
7b:    E = E − ‖B_i(j,:)‖_F^2
8a:    if E < ε^2 then
8b:       remove B_i(j+1 : m_i, :) from B; remove Q_i(:, j+1 : m_i) from Q
8c:       stop
8d:    end if
8e: end for
```

This adds negligible cost but allows us to determine the certain row of $\boldsymbol{B}_i$ where the accuracy tolerance is just attained. It makes the column (row) number of outputted $\boldsymbol{Q}$ ($\boldsymbol{B}$) an arbitrary integer, instead of a multiple of block size b in `randQB_b` algorithm.

Remark 3.1. If $\boldsymbol{A}$ is an implicit sparse matrix, the proposed framework needs more effort for calculating its Frobenius norm. The columns of $\boldsymbol{A}$ can be solved by multiplying $\boldsymbol{A}$ with the canonical basis vectors. Therefore, the cost for calculating $\|\boldsymbol{A}\|_{\mathrm{F}}$ will be $O(n(m+n))$ flops, if each matrix-vector product costs $O(m+n)$ flops. This might be affordable, as the Frobenius norm is a property of the matrix and we need to compute it just once.

3.2. The `randQB_EI` algorithm. The combination of Algorithm 1 and the `randQB_b` algorithm results in Algorithm 2 (called `randQB_EI`), whose steps 4–7 replace step 4 in Algorithm 1. Notice that step 7 in Algorithm 2 looks the same as step (5) in the `randQB_b` algorithm but is actually different. And, step (3) in the `randQB_b` algorithm becomes step 5 in Algorithm 2, which is the blocked Gram–Schmidt orthogonalization of $\boldsymbol{A\Omega}_i$ (notice $\boldsymbol{B} = \boldsymbol{Q}^T\boldsymbol{A}$).

Due to Theorem 2 and the orthogonality of $\boldsymbol{Q}$, we have the following proposition.

PROPOSITION 1. *The* `randQB_EI` *algorithm (Algorithm* 2*) is equivalent to the* `randQB_b` *algorithm, when executed in exact arithmetic.*

Assuming that multiplying two dense matrices of sizes $m \times n$ and $n \times l$ costs $C_{mm}mnl$ flops, and performing an economic QR factorization of an $m \times n$ dense matrix

Algorithm 2 The `randQB_EI` algorithm for the fixed-precision problem

Input: an $m \times n$ matrix $\boldsymbol{A}$; desired accuracy tolerance ε; block size b.
Output: $\boldsymbol{Q}$, $\boldsymbol{B}$, such that $\|\boldsymbol{A} - \boldsymbol{QB}\|_{\mathrm{F}} < \varepsilon$.

$\boldsymbol{Q} = [\,]$; $\boldsymbol{B} = [\,]$;
$E = \|\boldsymbol{A}\|_{\mathrm{F}}^2$
for $i = 1, 2, 3, \ldots,$ **do**
  $\boldsymbol{\Omega}_i = \mathrm{randn}(n, b)$
  $\boldsymbol{Q}_i = \mathrm{orth}(\boldsymbol{A\Omega}_i - \boldsymbol{Q}(\boldsymbol{B\Omega}_i))$
  $\boldsymbol{Q}_i = \mathrm{orth}(\boldsymbol{Q}_i - \boldsymbol{Q}(\boldsymbol{Q}^T\boldsymbol{Q}_i))$ # reorthogonalization
  $\boldsymbol{B}_i = \boldsymbol{Q}_i^T\boldsymbol{A}$ # no need to calculate $\boldsymbol{Q}_i^T(\boldsymbol{A} - \boldsymbol{QB})$
  $\boldsymbol{Q} = [\boldsymbol{Q},\ \boldsymbol{Q}_i]$
  $\boldsymbol{B} = \begin{bmatrix} \boldsymbol{B} \\ \boldsymbol{B}_i \end{bmatrix}$
  $E = E - \|\boldsymbol{B}_i\|_{\mathrm{F}}^2$
  if $E < \varepsilon^2$ **then stop**
end for

costs $C_{qr}mn\min(m,n)$ flops, we can analyze the flop counts of the relevant algorithms and compare their performance for handling a *dense* $\boldsymbol{A}$. If T_{randQB} and T_{randQB_b} denote the runtime of the algorithms `randQB` and `randQB_b`, respectively, [1] then

$$T_{randQB} \sim 2C_{mm}mnl + C_{qr}ml^2 \ , \tag{10}$$

$$T_{randQB\_b} \sim 3C_{mm}mnl + C_{mm}ml^2 + \frac{2}{t}C_{qr}ml^2, \tag{11}$$

where l is the number of columns in the resulting matrix $\boldsymbol{Q}$, and t satisfies $l = tb$. Note that b is practically much smaller than l (say, b =10 or 20), although the optimal choice of block size depends strongly on what hardware is used [1].

For the `randQB_EI` algorithm, the runtime can be similarly depicted:

$$T_{randQB\_EI} \sim 2C_{mm}mnl + C_{mm}(2m+n)l^2 + \frac{2}{t}C_{qr}ml^2 \ . \tag{12}$$

Because l is usually much smaller than m and n, we see that the flop count of `randQB_EI` is about 2/3 of that of `randQB_b` and is comparable to that of the basic randQB algorithm. Notice that C_{qr} is several times larger than C_{mm}. So, the flop count of `randQB_EI` could be smaller than that of `randQB` in the situation where t is a large number. If $\boldsymbol{A}$ is sparse, this would more possibly happen, because the `randQB` algorithm loses the benefit brought by the BLAS-3 operation.

The advantage of `randQB_EI` over `randQB_b` becomes more prominent if $\boldsymbol{A}$ is a sparse matrix. With the proposed error indicator, we no longer need the residual matrix. In contrast, it is always a dense matrix in the `randQB_b` algorithm and costs much larger memory and induces many more computations.

Remark 3.2. Algorithm 1 can also be combined with the `trQRCP` algorithm [14]. Although `trQRCP` generates a column permutation matrix as well, it does not affect the Frobenius norm of each partial or the entire matrix of $\boldsymbol{B}$. Therefore, this will produce another efficient algorithm for adaptive low-rank matrix approximation, which also adapts to sparse matrices.

3.3. Floating point arithmetic. Below we discuss the accuracy of the error indicator in floating point arithmetic. We use $\mathcal{E}(\cdot)$ and $\mathcal{E}_r(\cdot)$ to denote the functions of error and relative error, respectively.

As the error indicator $E = \|\boldsymbol{A}\|_{\mathrm{F}}^2 - \|\boldsymbol{B}\|_{\mathrm{F}}^2$, its calculated value $\hat{E}$ cannot be accurate when E is very small, due to the cancellation in calculation. In floating-point arithmetic, the machine precision ϵ_{mach} characterizes the maximum relative error of converting a real number to its floating-point representation, i.e.,

$$\forall x, \quad |\mathcal{E}_r(x)| \leq \epsilon_{mach}. \tag{13}$$

According to the definition of Frobenius norm, $\|\boldsymbol{A}\|_{\mathrm{F}}^2$ is the summation of squares of matrix entries. Therefore, the relative error of $\|\boldsymbol{A}\|_{\mathrm{F}}^2$ is bounded by $2\epsilon_{mach}$. The same thing applies to $\|\boldsymbol{B}\|_{\mathrm{F}}^2$. So,

$$\begin{aligned} |\mathcal{E}(E)| = |\mathcal{E}(\|\boldsymbol{A}\|_{\mathrm{F}}^2 - \|\boldsymbol{B}\|_{\mathrm{F}}^2)| &\leq |\mathcal{E}(\|\boldsymbol{A}\|_{\mathrm{F}}^2)| + |\mathcal{E}(\|\boldsymbol{B}\|_{\mathrm{F}}^2)| \\ &\leq 2\epsilon_{mach}(\|\boldsymbol{A}\|_{\mathrm{F}}^2 + \|\boldsymbol{B}\|_{\mathrm{F}}^2) \\ &< 4\epsilon_{mach}\|\boldsymbol{A}\|_{\mathrm{F}}^2. \end{aligned} \tag{14}$$

If we want to guarantee that E has a relative error no more than δ, i.e., $|\mathcal{E}(E)| \leq \delta E$, we shall enforce

$$4\epsilon_{mach}\|\boldsymbol{A}\|_{\mathrm{F}}^2 \leq \delta E. \tag{15}$$

This means the preset accuracy tolerance ε, which is larger than $\sqrt{E}$ at the termination of the algorithm, should satisfy

$$\varepsilon > \sqrt{E} \geq \sqrt{\frac{4\epsilon_{mach}\|\boldsymbol{A}\|_{\mathrm{F}}^2}{\delta}} = \sqrt{\frac{4\epsilon_{mach}}{\delta}}\|\boldsymbol{A}\|_{\mathrm{F}}. \tag{16}$$

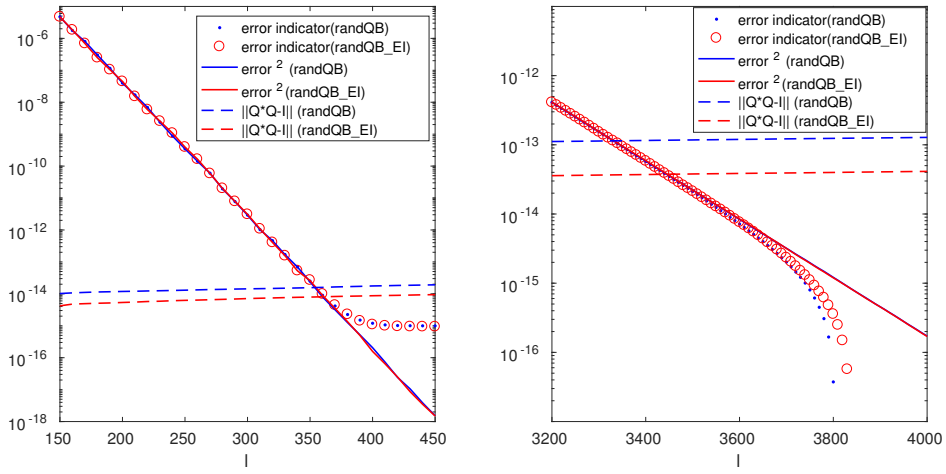
So, we obtain the following theorem.

THEOREM 3. *Suppose matrix $\boldsymbol{A}$ and accuracy tolerance ε are the input to the* `randQB_EI` *algorithm. If $\varepsilon > \sqrt{\frac{4\epsilon_{mach}}{\delta}}\|\boldsymbol{A}\|_{\mathrm{F}}$, the relative error of the calculated error indicator E must be no more than δ, e.g., if $\varepsilon > 2.1 \times 10^{-7}\|\boldsymbol{A}\|_{\mathrm{F}}$, the error of E is within 1% in the double-precision floating arithmetic, where $\epsilon_{mach} \approx 1.11 \times 10^{-16}$.*

Notice that, as an error indicator for the fixed-precision problem, E should have sufficient accuracy (e.g., with relative error $\sim 1\%$ or less). Otherwise, the outputted QB factorization would not satisfy the preset accuracy tolerance.

Besides, the orthogonality of $\boldsymbol{Q}$ also affects the accuracy of E. As the number of columns in $\boldsymbol{Q}$ increases, its orthogonality gradually degrades. This issue occurs for $\boldsymbol{Q}$ produced either by a single run of QR factorization (based on Householder transformation) or by a Gram–Schmidt procedure followed by the reorthogonalization step. We will investigate its effect in the following experiment.

An $n \times n$ matrix $\boldsymbol{A}$ is constructed to have singular values according to a decaying exponential. Two instances are tested, with singular value $\sigma_j = e^{-j/20}$ and $\sigma_j = e^{-j/200}$, $j = 1, 2, \ldots$, respectively. The results obtained from executing `randQB_EI` algorithm ($b = 10$) and `randQB` algorithm with different values of rank parameter l are shown in Figure 2. Note that for some large value of l, the error indicator can be of negative value, such that it cannot be drawn in the log-scale plot. From the figure, we can validate the correctness of Theorem 3. Providing that the square of



(a) $n = 3{,}000,\ \sigma_j = e^{-j/20}$ (b) $n = 20{,}000,\ \sigma_j = e^{-j/200}$

FIG. 2. *Normalized error indicator* $E/\|\boldsymbol{A}\|_{\mathrm{F}}^2$, *error square* $\|\boldsymbol{A}-\boldsymbol{QB}\|_{\mathrm{F}}^2/\|\boldsymbol{A}\|_{\mathrm{F}}^2$, *and the loss of orthogonality of* $\boldsymbol{Q}$ *vs.* $\boldsymbol{Q}$*'s column number* l *in* `randQB` *algorithm and* `randQB_EI` *algorithm.*

error $\|\boldsymbol{A}-\boldsymbol{QB}\|_{\mathrm{F}}^2/\|\boldsymbol{A}\|_{\mathrm{F}}^2 > (2.1\times 10^{-7})^2 = 4.4\times 10^{-14}$, the error indicator matches the square of error very well. This holds even when l is larger than 3,000, which corresponds to the situation with larger accumulated round-off error. In Figure 2, the value of $\|\boldsymbol{Q}^T\boldsymbol{Q}-\boldsymbol{I}\|_\infty$ [1] is also plotted, which reveals the loss of orthogonality of $\boldsymbol{Q}$. The results show that this issue is not severe in both the `randQB` and `randQB_EI` algorithms, although it gradually increases as the columns of $\boldsymbol{Q}$ are increased.

Remark 3.3. Theorem 3 suggests the limitation of the error indicator and the proposed algorithms for the fixed-precision problem. It means that the efficient framework for adaptive rank determination would not work, in double-precision floating arithmetic, for the problem with the accuracy tolerance ε less than $2.1\times 10^{-7}\|\boldsymbol{A}\|_{\mathrm{F}}$.

4. A pass-efficient algorithm for the QB factorization. The technique in last section efficiently solves the fixed-precision problem measured in the Frobenius norm. However, the `randQB_EI` algorithm is not suitable for the scenarios where accessing matrix entries is very expensive (e.g., $\boldsymbol{A}$ is too large and has to be stored on hard disk), as it visits $\boldsymbol{A}$ for considerable time (see Table 1). In this section, we propose a pass-efficient algorithm which largely reduces the number of passes over $\boldsymbol{A}$.

4.1. The version without reorthogonalization. We first consider the fixed-rank problem where the rank parameter k is given. A preliminary pass-efficient algorithm (presented as Algorithm 3) can be derived from the `randQB_b` or `randQB_EI` algorithm. The steps correspond to those of `randQB_EI` in Algorithm 2, one by one, except that the applications of $\boldsymbol{A}$ are moved out of the loop, and step 6 for reorthogonalization is ignored. In Algorithm 3, step 6 is the same as step 4 of `randQB_EI` algorithm, and steps 7 and 8 correspond to step 5 of `randQB_EI`. In step 9, $\boldsymbol{R}_i^{-T}$ is the inverse of the transpose of an upper triangular matrix $\boldsymbol{R}_i$. This step can be regarded as solving linear equations with the coefficient matrix $\boldsymbol{R}_i^T$, which is implemented by the "\" operator in MATLAB.

[1] This measure of loss of orthogonality follows Cleve Moler's blog with the title "Compare Gram–Schmidt and Householder Orthogonalization Algorithms" posted on October 17, 2016.

Algorithm 3 A preliminary pass-efficient algorithm for fixed-rank QB factorization

Input: $\boldsymbol{A} \in \mathbb{R}^{m\times n}$, k, s, block size b.
Output: $\boldsymbol{Q}$, $\boldsymbol{B}$.

1: $\boldsymbol{Q} = [\,]$; $\boldsymbol{B} = [\,]$; $l = k + s$;
2: $\boldsymbol{\Omega} = \mathrm{randn}(n, l)$
3: $\boldsymbol{G} = \boldsymbol{A\Omega}$
4: $\boldsymbol{H} = \boldsymbol{A}^T\boldsymbol{G}$
5: **for** $i = 1, 2, 3, \ldots, l/b$ **do**
6: $\quad\boldsymbol{\Omega}_i = \boldsymbol{\Omega}(:,\ (i-1)b+1 : ib)$
7: $\quad\boldsymbol{Y}_i = \boldsymbol{G}(:,\ (i-1)b+1 : ib) - \boldsymbol{Q}(\boldsymbol{B\Omega}_i)$
8: $\quad[\boldsymbol{Q}_i,\ \boldsymbol{R}_i] = \mathrm{qr}(\boldsymbol{Y}_i)$
9: $\quad\boldsymbol{B}_i = \boldsymbol{R}_i^{-T}(\boldsymbol{H}(:,\ (i-1)b+1 : ib)^T - \boldsymbol{\Omega}_i^T\boldsymbol{B}^T\boldsymbol{B})$
10: $\quad\boldsymbol{Q} = [\boldsymbol{Q},\ \boldsymbol{Q}_i]$
11: $\quad\boldsymbol{B} = \begin{bmatrix} \boldsymbol{B} \\ \boldsymbol{B}_i \end{bmatrix}$
12: **end for**

Because $\boldsymbol{Y}_i = \boldsymbol{Q}_i\boldsymbol{R}_i$,

$$\boldsymbol{Q}_i = \boldsymbol{Y}_i\boldsymbol{R}_i^{-1}. \tag{17}$$

Substituting it into step 7 of the `randQB_EI` algorithm, we have

$$\begin{aligned}
\boldsymbol{B}_i &= (\boldsymbol{Y}_i\boldsymbol{R}_i^{-1})^T\boldsymbol{A} \\
&= \boldsymbol{R}_i^{-T}\boldsymbol{Y}_i^T\boldsymbol{A} \\
&= \boldsymbol{R}_i^{-T}(\boldsymbol{G}(:,\ (i-1)b+1 : ib)^T - \boldsymbol{\Omega}_i^T\boldsymbol{B}^T\boldsymbol{Q}^T)\boldsymbol{A} \\
&= \boldsymbol{R}_i^{-T}((\boldsymbol{A}^T\boldsymbol{G}(:,\ (i-1)b+1 : ib))^T - \boldsymbol{\Omega}_i^T\boldsymbol{B}^T\boldsymbol{Q}^T\boldsymbol{A}) \\
&= \boldsymbol{R}_i^{-T}(\boldsymbol{H}(:,\ (i-1)b+1 : ib)^T - \boldsymbol{\Omega}_i^T\boldsymbol{B}^T\boldsymbol{B}).
\end{aligned} \tag{18}$$

This means that step 9 of Algorithm 3 is equivalent to step 7 of `randQB_EI`. Therefore, we obtain the following proposition.

PROPOSITION 2. *Algorithm* 3 *is mathematically equivalent to the fixed-rank version of the* `randQB_EI` *algorithm without reorthogonalization.*

4.2. The version with reorthogonalization. In reality, the loss of orthogonality among the columns of $\{\boldsymbol{Q}_1, \boldsymbol{Q}_2, \ldots, \}$ occurs due to the accumulation of round-off error. This means the reorthogonalization step, i.e., step 6 in Algorithm 2, cannot be ignored. Below we derive the revisions to Algorithm 3 to address this issue.

The reorthogonalization step can be expressed as

$$\tilde{\boldsymbol{Q}}_i\tilde{\boldsymbol{R}}_i = \boldsymbol{Q}_i - \boldsymbol{Q}\boldsymbol{Q}^T\boldsymbol{Q}_i, \tag{19}$$

where $\tilde{\boldsymbol{Q}}_i \neq \boldsymbol{Q}_i$ and $\tilde{\boldsymbol{R}}_i \neq \boldsymbol{I}$ due to the loss of orthogonality. $\tilde{\boldsymbol{Q}}_i$ is better orthogonal to the previously generated $\{\boldsymbol{Q}_1, \boldsymbol{Q}_2, \ldots, \boldsymbol{Q}_{i-1}\}$ than $\boldsymbol{Q}_i$. Now, we need to derive a formula for calculating $\boldsymbol{B}_i$ which does not involve $\boldsymbol{A}$ explicitly. Based on (17),

$$\tilde{\boldsymbol{Q}}_i = (\boldsymbol{I} - \boldsymbol{Q}\boldsymbol{Q}^T)\boldsymbol{Y}_i\boldsymbol{R}_i^{-1}\tilde{\boldsymbol{R}}_i^{-1}. \tag{20}$$

So, the corresponding formula for $\boldsymbol{B}_i$ is

$$\begin{aligned}\tilde{\boldsymbol{B}}_i =& \tilde{\boldsymbol{Q}}_i^T \boldsymbol{A} \\ =& (\tilde{\boldsymbol{R}}_i \boldsymbol{R}_i)^{-T} \boldsymbol{Y}_i^T (\boldsymbol{I} - \boldsymbol{Q}\boldsymbol{Q}^T)\boldsymbol{A} \\ =& (\tilde{\boldsymbol{R}}_i \boldsymbol{R}_i)^{-T} (\boldsymbol{\Omega}_i^T \boldsymbol{A}^T - \boldsymbol{\Omega}_i^T \boldsymbol{B}^T \boldsymbol{Q}^T)(\boldsymbol{A} - \boldsymbol{Q}\boldsymbol{B}),\end{aligned} \tag{21}$$

where the formula of $\boldsymbol{Y}_i$ and the equality $\boldsymbol{B} = \boldsymbol{Q}^T\boldsymbol{A}$ is made use of. The product of the last two brackets can be further simplified.

$$\begin{aligned}&(\boldsymbol{\Omega}_i^T \boldsymbol{A}^T - \boldsymbol{\Omega}_i^T \boldsymbol{B}^T \boldsymbol{Q}^T)(\boldsymbol{A} - \boldsymbol{Q}\boldsymbol{B}) \\ &= \boldsymbol{H}_i^T - \boldsymbol{G}_i^T \boldsymbol{Q}\boldsymbol{B} - \boldsymbol{\Omega}_i^T \boldsymbol{B}^T \boldsymbol{B} + \boldsymbol{\Omega}_i^T \boldsymbol{B}^T \boldsymbol{Q}^T \boldsymbol{Q}\boldsymbol{B} \\ &= \boldsymbol{H}_i^T - \boldsymbol{Y}_i^T \boldsymbol{Q}\boldsymbol{B} - \boldsymbol{\Omega}_i^T \boldsymbol{B}^T \boldsymbol{B}.\end{aligned} \tag{22}$$

In the deduction, $\boldsymbol{G}_i$ and $\boldsymbol{H}_i$ denote $\boldsymbol{G}(:, (i-1)b+1 : ib)$ and $\boldsymbol{H}(:, (i-1)b+1 : ib)$ in Algorithm 3, respectively. Therefore,

$$\tilde{\boldsymbol{B}}_i = (\tilde{\boldsymbol{R}}_i \boldsymbol{R}_i)^{-T} (\boldsymbol{H}_i^T - \boldsymbol{Y}_i^T \boldsymbol{Q}\boldsymbol{B} - \boldsymbol{\Omega}_i^T \boldsymbol{B}^T \boldsymbol{B}) \ . \tag{23}$$

Based on (19) and (23), we can derive the version with reorthogonalization for Algorithm 3. We just need to replace the ninth step with the following steps.

9a: $[\boldsymbol{Q}_i,\ \tilde{\boldsymbol{R}}_i] = \mathrm{qr}(\boldsymbol{Q}_i - \boldsymbol{Q}(\boldsymbol{Q}^T\boldsymbol{Q}_i))$
9b: $\boldsymbol{R}_i = \tilde{\boldsymbol{R}}_i \boldsymbol{R}_i$
9c: $\boldsymbol{B}_i = \boldsymbol{R}_i^{-T}(\boldsymbol{H}(:,\ (i-1)b+1 : ib)^T - \boldsymbol{Y}_i^T \boldsymbol{Q}\boldsymbol{B} - \boldsymbol{\Omega}_i^T \boldsymbol{B}^T \boldsymbol{B})$

Notice that $\boldsymbol{Q}_i$ and $\boldsymbol{B}_i$ are overwritten to stand for $\tilde{\boldsymbol{Q}}_i$ and $\tilde{\boldsymbol{B}}_i$, respectively. Based on Proposition 2 and the above deduction, we see that the pass-efficient algorithm with reorthogonalization is also mathematically equivalent to the fixed-rank versions of `randQB_EI` and `randQB_b` algorithms.

This algorithm with *fewer passes* over $\boldsymbol{A}$ is called `randQB_FP`. Based on the notation in section 3, its flop count analysis is as follows.

$$T_{randQB\_FP} \sim 2C_{mm}mnl + 2C_{mm}(m+n)l^2 + \frac{2}{t}C_{qr}ml^2 \ , \tag{24}$$

where t satisfies $l = tb$. Compared with the `randQB_EI` algorithm, the `randQB_FP` algorithm has a slightly larger flop count. However, while handling a dense $\boldsymbol{A}$ its actual runtime may be shorter because it lumps the multiplications with $\boldsymbol{A}$.

Remark 4.1. The round-off error may affect the accuracy of $\boldsymbol{B}_i$, and it increases as the number of iterations increases. However, this may not be an issue for practical low-rank approximation problems. In section 5, we will present numerical experiments to validate the effectiveness of the `randQB_FP` algorithm, which shows it works very well for many applications with the rank parameter up to several thousands or the relative Frobenius-norm error of approximation as small as 10^{-7}.

Remark 4.2. The `randQB_FP` algorithm can derive a single-pass algorithm, if matrix $\boldsymbol{A}$ is stored in the row-major format or is revealed row(s) by row(s). Suppose $\boldsymbol{A}_{i,:}$ denotes the ith row of $\boldsymbol{A}$. With it we have the ith row of $\boldsymbol{G}$, $\boldsymbol{G}_{i,:} = \boldsymbol{A}_{i,:}\boldsymbol{\Omega}$. Then, because $\boldsymbol{H} = \boldsymbol{A}^T\boldsymbol{G} = \sum_i \boldsymbol{A}_{i,:}^T \boldsymbol{G}_{i,:}$, the ith term in this summation can be obtained. With all rows of $\boldsymbol{A}$, in this way we can accomplish steps 3 and 4 in the `randQB_FP` algorithm with only one pass over $\boldsymbol{A}$. It should be pointed out that this algorithm is not a general single-pass algorithm, as it has the restriction of the matrix. For more general single-pass algorithms for low-rank matrix approximation, please refer to the recent work [22].

4.3. The inclusion of power iteration scheme. The error of randomized QB factorization could be large for the matrix whose singular value decays slowly [2]. So, the *power iteration scheme* has been proposed to relieve this weakness [1, 2, 7]. Conceptually, the power iteration means replacing $\boldsymbol{A}$ with $(\boldsymbol{A}\boldsymbol{A}^T)^P\boldsymbol{A}$, where P is an integer. However, in floating-point computation any singular components smaller than $\sigma_1\epsilon_{mach}^{1/(2P+1)}$ will be lost. This makes the orthonormalization steps after the applications of $\boldsymbol{A}$ and $\boldsymbol{A}^T$ necessary, and P should not be set to a large number. Incorporating the power iteration, we have the `randQB_FP` algorithm for the fixed-precision problem presented as Algorithm 4, where the error indicator E is utilized.

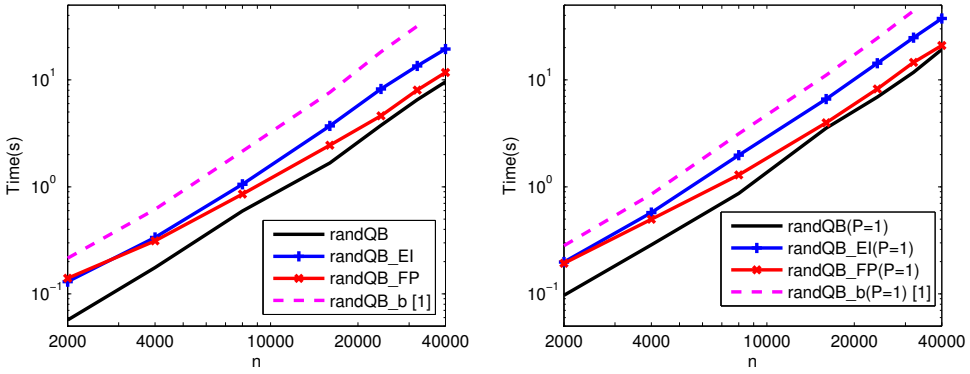
Algorithm 4 The `randQB_FP` with power scheme for the fixed-precision problem

Input: $\boldsymbol{A} \in \mathbb{R}^{m\times n}$, desired accuracy tolerance ε, block size b, power parameter P.
Output: $\boldsymbol{Q}$, $\boldsymbol{B}$, such that $\|\boldsymbol{A}-\boldsymbol{Q}\boldsymbol{B}\|_{\mathrm{F}} \le \varepsilon$.

$\boldsymbol{Q} = [\,]$; $\boldsymbol{B} = [\,]$;
$\boldsymbol{\Omega} = \mathrm{randn}(n, \tilde{l})$, where $\tilde{l}$ is a sufficiently large number.
for $i = 1 : P$ **do**
  $\boldsymbol{G} = \mathrm{orth}(\boldsymbol{A}\boldsymbol{\Omega})$
  $\boldsymbol{\Omega} = \mathrm{orth}(\boldsymbol{A}^T\boldsymbol{G})$
end for
$\boldsymbol{G} = \boldsymbol{A}\boldsymbol{\Omega}$
$\boldsymbol{H} = \boldsymbol{A}^T\boldsymbol{G}$
$E = \|\boldsymbol{A}\|_{\mathrm{F}}^2$
for $i = 1, 2, 3, \ldots,$ **do**
  $\boldsymbol{\Omega}_i = \boldsymbol{\Omega}(:,\ (i-1)b+1 : ib)$
  $\boldsymbol{Y}_i = \boldsymbol{G}(:,\ (i-1)b+1 : ib) - \boldsymbol{Q}(\boldsymbol{B}\boldsymbol{\Omega}_i)$
  $[\boldsymbol{Q}_i,\ \boldsymbol{R}_i] = \mathrm{qr}(\boldsymbol{Y}_i)$
  $[\boldsymbol{Q}_i,\ \tilde{\boldsymbol{R}}_i] = \mathrm{qr}(\boldsymbol{Q}_i - \boldsymbol{Q}(\boldsymbol{Q}^T\boldsymbol{Q}_i))$
  $\boldsymbol{R}_i = \tilde{\boldsymbol{R}}_i\boldsymbol{R}_i$
  $\boldsymbol{B}_i = \boldsymbol{R}_i^{-T}(\boldsymbol{H}(:,\ (i-1)b+1 : ib)^T - \boldsymbol{Y}_i^T\boldsymbol{Q}\boldsymbol{B} - \boldsymbol{\Omega}_i^T\boldsymbol{B}^T\boldsymbol{B})$
  $\boldsymbol{Q} = [\boldsymbol{Q},\ \boldsymbol{Q}_i]$
  $\boldsymbol{B} = \begin{bmatrix} \boldsymbol{B} \\ \boldsymbol{B}_i \end{bmatrix}$
  $E = E - \|\boldsymbol{B}_i\|_{\mathrm{F}}^2$
  if $\sqrt{E} < \varepsilon$ **then stop**
end for

In Algorithm 4, a sufficient large value of $\tilde{l}$ should be set according to problem-specific experience and the concern of computing time. If the set $\tilde{l}$ is not large enough for attaining the specified accuracy criterion, we need to regenerate the $\boldsymbol{\Omega}$ matrix and rerun the algorithm to collect additional columns/rows of $\boldsymbol{Q}$ and $\boldsymbol{B}$. This situation and the power scheme both increase the number of passes over $\boldsymbol{A}$. But compared to other algorithms for the fixed-precision problem, this fixed-precision `randQB_FP` algorithm involves much fewer passes over $\boldsymbol{A}$.

5. Numerical results. In this section we compare the proposed algorithms against several existing algorithms in terms of execution time, memory usage, and accuracy. All experiments are carried out on a Linux server with two 12-core Intel Xeon E5-2630 CPUs at 2.30 GHz and 32GB RAM. For comparison of speed, the proposed algorithms have been implemented in C based on the codes shared by the authors of [1, 16]. The program is coded with OpenMP derivatives and compiled with the Intel ICC compiler with MKL libraries [17], to take full advantage of the



(a) Without the power scheme (b) With the power scheme

FIG. 3. *Runtime of the randomized QB-factorization algorithms for dense matrices* ($l = 200$).

multicore CPUs. The QR factorization and other basic linear algebra operations are implemented through LAPACK routines which are automatically executed in parallel.

5.1. Comparison of speed. We compute the QB factorization of an $n \times n$ matrix $\boldsymbol{A}$. The matrix is stored in RAM, though the pass-efficient algorithm `randQB_FP` is suitable for handling a large matrix stored on hard disk. For the latter situation, the runtime advantage of `randQB_FP` can be estimated by the number of passes over $\boldsymbol{A}$. Here we want to show that `randQB_FP` has little sacrifice while owning the pass-efficient property. Notice the singular value distribution of the matrix is immaterial for this runtime comparison. Four algorithms are compared:

- The `randQB` algorithm in Figure 1(a);
- The `randQB_b` algorithm in Figure 1(b), obtained from [16];
- The `randQB_EI` algorithm presented in section 3;
- The `randQB_FP` algorithm presented in section 4.

We compare their speed using both dense and sparse matrices, both as a function of the dimension of the matrix and the parameter l denoting the number of the output $\boldsymbol{Q}$'s columns. The block size is $b = 20$ for the `randQB_b`, `randQB_EI`, and `randQB_FP` algorithms. For each runtime measurement, the average time over 20 runs is reported. Notice that the compared `randQB_b` algorithm is an efficient parallel implementation open-sourced in [16], also based on Intel MKL libraries.

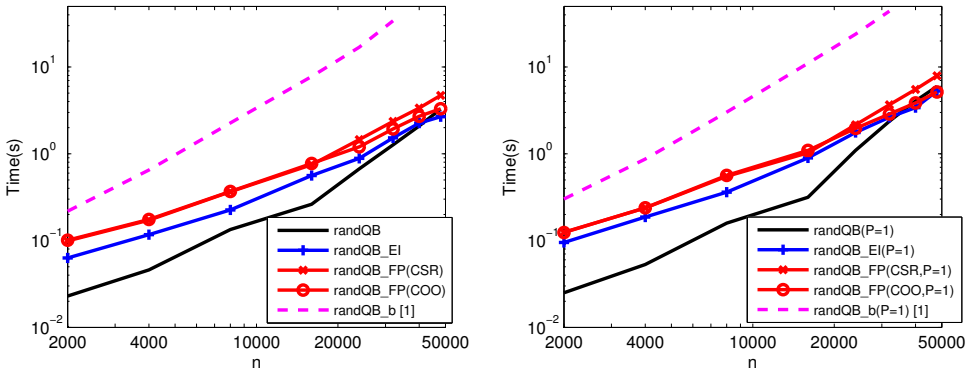
In the first experiment we test the algorithms on dense matrices of varying size. n ranges from 2,000 to 40,000. The value of l is always 200. The results are shown in Figure 3 for the situations without and with the power scheme. The data of the blocked randQB algorithm for the matrix with $n = 40{,}000$ are not available due to the unreasonably long runtime of the program from [16]. From the results in Figure 3(a), we see that the `randQB_EI` and `randQB_FP` algorithms are 2.4X (13.47s vs. 31.78s) and 4.0X (8.01s vs. 31.78s) faster than the implementation of `randQB_b` algorithm, respectively, when $n = 32{,}000$. If the power scheme is imposed, the acceleration ratios decrease to 1.8X and 3.0X, respectively, which are still remarkable. The `randQB` algorithm has the fastest computational speed, but its advantage over the `randQB_FP` algorithm becomes marginal when the matrix size is large.

Here we only show the runtime results with the power parameter $P = 1$, as for many applications this already achieves sufficient accuracy.

The memory costs for some large matrices are listed in Table 2. For the `randQB`, `randQB_EI`, and `randQB_FP` algorithms, the memory cost is mainly due to storing

TABLE 2
The memory usage of the randomized QB-factorization algorithms for dense matrices ($l = 200$).

n	randQB	randQB_EI	randQB_FP	randQB_b [1]
16,000	2,308 MB	2,303 MB	2,357 MB	6,237 MB
24,000	4,792 MB	4,796 MB	4,873 MB	13,618 MB
32,000	8,253 MB	8,253 MB	8,356 MB	23,931 MB
40,000	12,694 MB	12,581 MB	12,714 MB	N.A.



(a) Without the power scheme (b) With the power scheme

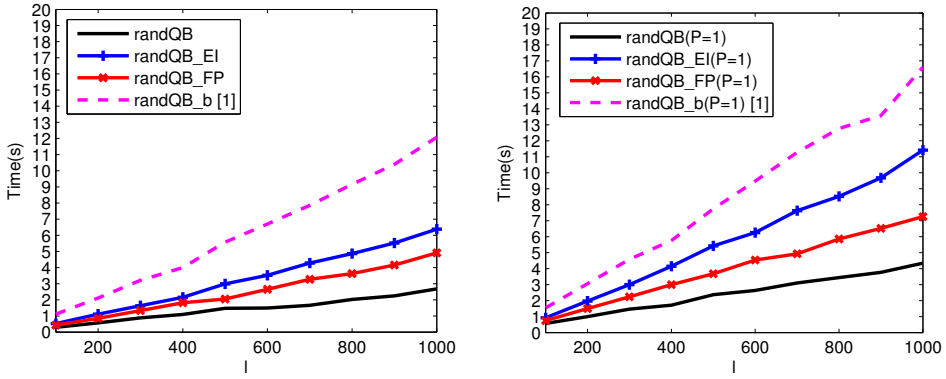
FIG. 4. *Runtime of the randomized QB-factorization algorithms for sparse matrices* ($l = 200$).

matrix $\boldsymbol{A}$. For the randQB_b algorithm, it needs additional memory to store the residual matrix and the product of $\boldsymbol{QB}$. So, the proposed algorithms consume about 1/3 of that used by the blocked randQB algorithm. If we allow that $\boldsymbol{A}$ can be overwritten by the residual matrix, the memory cost of the randQB_b algorithm [1] can be reduced, but will still be 2X larger than the proposed algorithms. For the largest case with $n = 40{,}000$, the randQB_b algorithm actually requests more memory than the size of RAM ($\sim$ 32 GB), which explains the aforementioned long runtime of randQB_b.

The second experiment is about the algorithms' efficiency for sparse matrices. We generate sparse matrices with roughly 0.3% nonzero elements. They are stored in compressed sparse row (CSR) format [18]. The runtimes of the algorithms are shown in Figure 4. The results of the randQB_b algorithm for the matrices with $n \geq 40{,}000$ are not available due to unreasonably long runtime. In contrast, it only takes a couple of seconds for the other algorithms to process the largest matrix with $n = 48{,}000$. We see that the proposed algorithms take usage of the sparsity, while the blocked randQB algorithm cannot. The speedup ratios of the former to the latter increase as the matrix size increases. For $n = 32{,}000$, the randQB_EI and randQB_FP algorithms are more than 22X and 14X faster than the implementation of randQB_b algorithm, respectively. Different from the situation for dense matrices, the randQB_EI algorithm becomes faster than randQB_FP. This implies that lumping the multiplications of a sparse matrix all together brings less benefit. And, randQB_EI could run faster than the randQB algorithm for a matrix size over 48,000. This can be explained by the comparison of (10) and (12), the inefficiency of orthogonalizing the whole matrix of l columns, and that the sparse matrix removes the benefit of BLAS-3 operation to randQB. Another interesting phenomenon is that if we instead store a large sparse matrix with the COO (coordinate) format, the runtime of the

TABLE 3
The memory usage of the randomized QB-factorization algorithms for sparse matrices ($l = 200$).

n	randQB	randQB_EI	randQB_FP	randQB_b [1]
16,000	162 MB	174 MB	223 MB	6,153 MB
24,000	232 MB	239 MB	312 MB	13,572 MB
32,000	293 MB	303 MB	402 MB	23,917 MB
40,000	338 MB	359 MB	488 MB	N.A.
48,000	405 MB	426 MB	582 MB	N.A.



(a) Without the power scheme (b) With the power scheme

FIG. 5. *Runtime of the randomized QB-factorization algorithms for a dense matrix* ($n = 8{,}000$) *with varying value of* l.

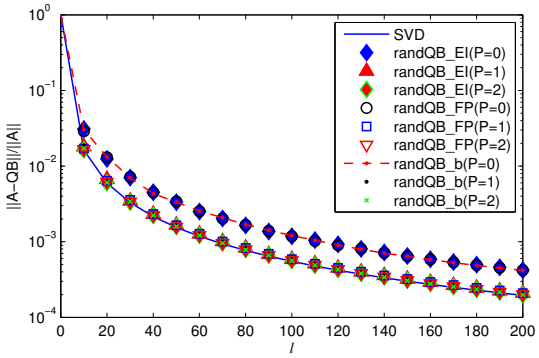
randQB_FP algorithm can be reduced by 30%. This means that the COO format is more adaptive to parallel computing. If we set $P = 1$, similar observations regarding the experimental results can be drawn, as shown in Figure 4(b).

The memory cost of these algorithms is listed in Table 3, from which we see more prominent memory saving of the proposed algorithms over the randQB_b algorithm. While compared with randQB, the proposed algorithms consume comparable memory.

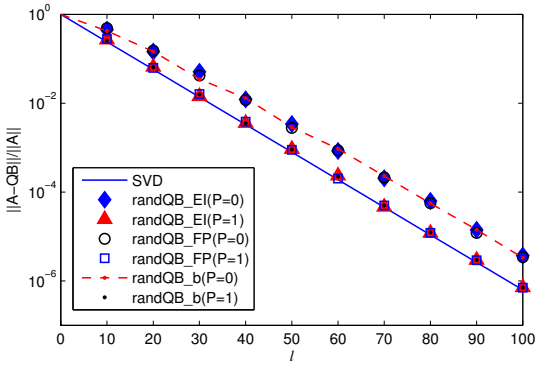
Last, we test a dense matrix with size $n = 8{,}000$, and vary the value of l. The trends of the runtime are plotted in Figure 5. It shows that the randQB_EI and randQB_FP algorithms without the power scheme are about 1.9X and 2.5X faster than randQB_b, respectively. If the power scheme is imposed, the speedup ratios to randQB_b decrease, but randQB_FP is still more than 2X faster than randQB_b.

5.2. Comparison of accuracy. Three kinds of matrices are tested standing for different distribution patterns of singular values:

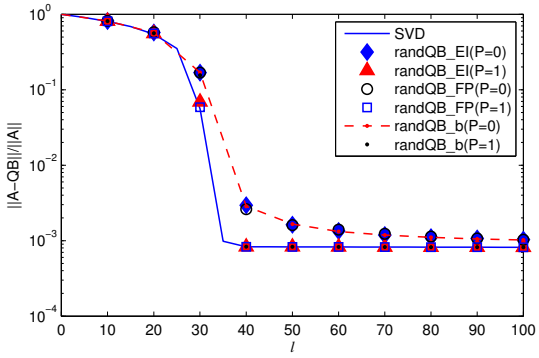
- Matrix 1 (slow decay): $\boldsymbol{A} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}$, where $\boldsymbol{U}$ and $\boldsymbol{V}$ are randomly drawn matrices with orthonormal columns, and the diagonal matrix $\boldsymbol{\Sigma}$ has diagonal elements $\sigma_j = 1/j^2$.
- Matrix 2 (fast decay): $\boldsymbol{A}$ is formed just like Matrix 1, but the diagonal elements of $\boldsymbol{\Sigma}$ is given by $\sigma_j = e^{-j/7}$. It reflects a fast decay of singular values.
- Matrix 3 (S-shape decay): $\boldsymbol{A}$ is built in the same manner as Matrices 1 and 2, but the diagonal elements of $\boldsymbol{\Sigma}$ are given by $\sigma_j = 0.0001 + (1 + e^{j-30})^{-1}$. It makes the singular values first hover around 1, then decay rapidly, and finally level out at about 0.0001.



(a) Errors of Matrix 1 whose singular values decay slowly



(b) Errors of Matrix 2 whose singular values decay rapidly



(c) Errors of Matrix 3 with S-shape decay of singular values

FIG. 6. *Errors of different algorithms for approximating the test matrices* ($n = 2{,}000$, $b = 10$).

For each kind, we generate a $2{,}000 \times 2{,}000$ matrix, for which we compare the errors of the proposed techniques and the blocked randQB scheme [1] for varying l values. The results are shown in Figure 6, where we see that the proposed techniques have just the same accuracy as the blocked randQB algorithm. If we use the power scheme, even with a power parameter as small as $P = 1$, the errors of the `randQB_EI` and `randQB_FP` algorithms are remarkably reduced. And, the power schemes with $P = 1$ and $P = 2$ produce indistinguishable results for the tested matrices. Both are extremely close to the optimal results from SVD.

A single-pass algorithm for low-rank approximation

Input: $\boldsymbol{A}$, l.

Output: $\boldsymbol{Q}$, $\tilde{\boldsymbol{Q}}$, $\hat{\boldsymbol{B}}$ such that $\boldsymbol{A} \approx \boldsymbol{Q}\hat{\boldsymbol{B}}\tilde{\boldsymbol{Q}}^T$.

(1) $\boldsymbol{\Omega} = \text{randn}(n, l)$; $\tilde{\boldsymbol{\Omega}} = \text{randn}(m, l)$;

(2) $\boldsymbol{Y} = \boldsymbol{A}\boldsymbol{\Omega}$; $\tilde{\boldsymbol{Y}} = \boldsymbol{A}^T\tilde{\boldsymbol{\Omega}}$;

(3) $\boldsymbol{Q} = \text{orth}(\boldsymbol{Y})$; $\tilde{\boldsymbol{Q}} = \text{orth}(\tilde{\boldsymbol{Y}})$;

(4) $\hat{\boldsymbol{B}} = (\tilde{\boldsymbol{\Omega}}^T\boldsymbol{Q})^{-1}\tilde{\boldsymbol{Y}}^T\tilde{\boldsymbol{Q}}$.

FIG. 7. *An existing single-pass algorithm for low-rank matrix factorization* [2].

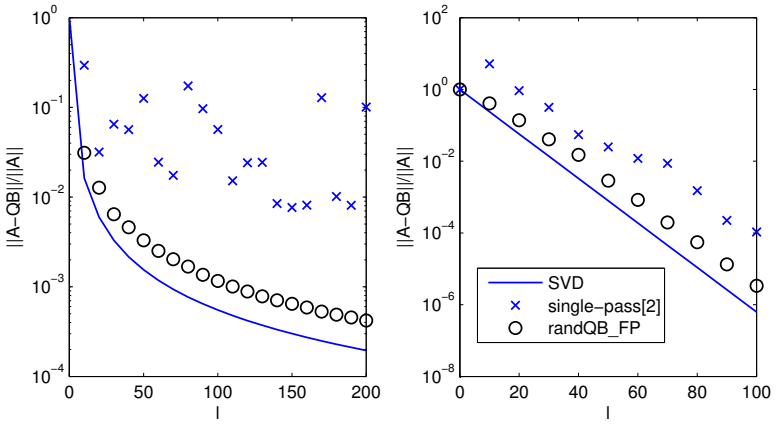


FIG. 8. *The approximation errors of the two single-pass algorithms and truncated SVD for Matrix* 1 *(left) and Matrix* 2 *(right).*

5.3. Performance of the single-pass algorithm. Without the power scheme, the `randQB_FP` algorithm is a single-pass algorithm (see Algorithm 3). This is because $\boldsymbol{G} = \boldsymbol{A}\boldsymbol{\Omega}$ and $\boldsymbol{H} = \boldsymbol{A}^T\boldsymbol{G}$ can be executed through one pass over matrix $\boldsymbol{A}$, providing that $\boldsymbol{A}$ is in the row-major format. Another single-pass algorithm was proposed in [2], as a remedy to the `randQB` algorithm. It is shown in Figure 7, whose step (3) produces matrices $\boldsymbol{Q}$ and $\tilde{\boldsymbol{Q}}$ such that $\boldsymbol{A} \approx \boldsymbol{Q}\boldsymbol{Q}^T\boldsymbol{A}\tilde{\boldsymbol{Q}}\tilde{\boldsymbol{Q}}^T$. Then, a small matrix $\hat{\boldsymbol{B}} = \boldsymbol{Q}^T\boldsymbol{A}\tilde{\boldsymbol{Q}}$ is approximately solved in step (4), because $\tilde{\boldsymbol{Q}}^T\tilde{\boldsymbol{Y}} \approx \tilde{\boldsymbol{Q}}^T\boldsymbol{A}^T\boldsymbol{Q}\boldsymbol{Q}^T\tilde{\boldsymbol{\Omega}} = \hat{\boldsymbol{B}}^T\boldsymbol{Q}^T\tilde{\boldsymbol{\Omega}}$. This single-pass algorithm corresponds to the low-rank factorization in the form of $\boldsymbol{A} \approx \boldsymbol{Q}\hat{\boldsymbol{B}}\tilde{\boldsymbol{Q}}^T$. Obviously, it includes more approximations and is not equivalent to the `randQB` algorithm. In contrast, Algorithm 3 is mathematically equivalent to `randQB` (see Proposition 2) and is supposed to be more accurate. With Matrices 1 and 2 from section 5.2, we can compare the accuracy of the both algorithms, whose results are plotted in Figure 8.

From Figure 8 we see that the approximation error of the single-pass algorithm in [2] is often one order of magnitude larger than that of our `randQB_FP` based algorithm. Actually, it does not even decrease as the rank of the approximation matrix increases. We also calculate the top 50 singular values, and the oversampling with $s = 10$ is applied to the both algorithms. The results are shown in Figure 9, along with those obtained from the `randQB` algorithm, where the results of `randQB_FP` and `randQB` are indistinguishable. For the matrix with slow decay of singular value the result from `randQB_FP` shows moderate accuracy on the top singular values, whose error is usually orders of magnitude smaller than that of the single-pass algorithm in [2].

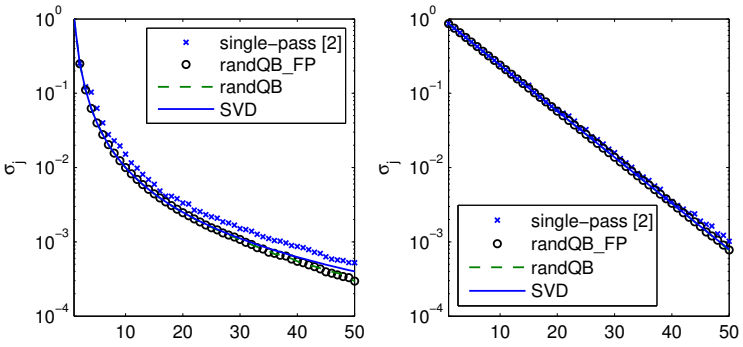


FIG. 9. *The computational results of top singular values for Matrix* 1 *(left) and Matrix* 2 *(right).*

TABLE 4
The results on solving the fixed-precision problems based on three test matrices.

Matrix type	ϵ	randQB_EI			randQB_FP		Truncated SVD		RangeFinder [2]	
		Rank	Time(s)	Error	Rank	Time(s)	Rank	Time(s)	Rank	Time(s)
Matrix 1	1e-2	**15**	1.19	9.3e-3	**15**	3.13	**15**	123	115	1.1
	1e-4	**327**	8.29	9.98e-5	**328**	3.71	**313**		2,084	29.1
Matrix 2	1e-4	**66**	2.16	8.37e-5	**66**	2.73	**65**	115	101	1.1
	1e-5	**82**	2.68	8.86e-6	**82**	3.17	**81**		113	1.1
Matrix 3	1e-2	**33**	1.56	4.1e-3	**33**	3.62	**32**	126	3,618	87.8
	1.5e-3	**1,588**	18.7	1.499e-3	**1,587**	15.8	**1,587**		7,916	379

5.4. Results of solving the fixed-precision problems. In this subsection we test the proposed algorithms with some fixed-precision problems. The optimal solution is the factorization with the smallest sizes of $\boldsymbol{Q}$ and $\boldsymbol{B}$, which corresponds to a lower amount of subsequent computation. It can be achieved by first calculating SVD of the input matrix $\boldsymbol{A}$, and then checking $(\sum_{j=k+1}^{\min(m,n)} \sigma_j^2)^{1/2}$, where σ_j is $\boldsymbol{A}$'s jth singular value, to determine the smallest k satisfying the accuracy criterion. Here, we always consider the accuracy criterion with a relative tolerance: $\|\boldsymbol{A}-\boldsymbol{QB}\|_{\mathrm{F}} < \epsilon\|\boldsymbol{A}\|_{\mathrm{F}}$.

The proposed algorithms are compared with the SVD based method and the *adaptive randomized range finder* (Algorithm 4.2 in [2]). The row-by-row calculation scheme mentioned in section 3 is implemented into our algorithms. The experiments are carried out with MATLAB on the aforementioned Linux server. The built-in commands like "`svd`," "`qr`," etc., are employed, which naturally take advantage of parallel computing.

For the `randQB_FP` algorithm, we empirically set $\tilde{l} = 50b$. With a suitable value of b, this produces a large enough $\tilde{l}$ for attaining the accuracy criteria in the experiments. A more sophisticated approach for setting $\tilde{l}$ and b can be investigated in the future.

We first construct the three kinds of matrices in section 5.2, each of $8{,}000 \times 8{,}000$ size, and test them with the four methods. Their results are shown in Table 4. For `randQB_EI` and `randQB_FP`, the power scheme with $P = 1$ is used. The block size is set to $b = 10$ in all tests, except the last one for which $b = 40$. In Table 4, "ϵ" stands for the threshold for relative error, and "error" means the relative Frobenius-norm error of the produced QB factorization. From the table we see that the results of `randQB_EI` and `randQB_FP` algorithms all satisfy the set accuracy demands. And, the corresponding ranks (i.e., the number of columns in $\boldsymbol{Q}$) are

TABLE 5
The results on solving the fixed-precision problems based on two real data.

Matrix	ϵ	Parameters	randQB_EI			randQB_FP		Truncated SVD		RangeFinder [2]	
			Rank	Time(s)	Error	Rank	Time(s)	Rank	Time(s)	Rank	Time(s)
Image	0.1	P=1, b=10	468	8.1	0.0999	471	3.25	**426**	44.2	2,913	79.0
		P=1, b=20	468	4.23	0.0999	472	4.44				
		P=2, b=10	**441**	9.98	0.0999	**443**	3.47				
		P=2, b=20	441	5.76	0.0999	443	7.26				
AMiner	0.5	P=1, b=50	2440	108	0.4999	2,449	143	**2,115**	1,049	8,018	399
		P=2, b=50	**2,229**	134	0.4999	**2,242**	205				

(a) original image (b) compressed image (10% error)

FIG. 10. *The original image and the compressed image obtained with the proposed algorithm.*

very close to the optimal values from the SVD based approach. As for the runtime, the proposed algorithms are usually several tens times faster than SVD. Notice that our MATLAB programs are less optimized than the built-in `svd` command. So, more significant speedup could be expected for the implementation in C. Although the adaptive range finder is built on a theory with spectral norm of matrix, in our experiments it always produces a QB factorization satisfying the accuracy demand in Frobenius norm. However, its results (factor matrices) are much larger than necessary.

We then test the algorithms with two real data. One is from a scenic image, and the other is from an information retrieval application "AMiner" [19]. The colored image is represented by a 9,504 × 4,752 matrix. The other is an 8,130 × 100,000 keyword-person matrix produced with the term frequency and inverse document frequency model [20]. This sparse matrix has about 0.2% nonzero elements. The computational results are listed in Table 5, with different power parameters and block sizes. They again validate that the proposed algorithms can automatically satisfy the accuracy criterion. And, with $P = 2$ the result of rank is substantially reduced, approaching the optimal value. For the same power scheme, setting larger block size b we can reduce the runtime of `randQB_EI`. In contrast, the runtime of `randQB_FP` increases with the block size, as we have set $\tilde{l} = 50b$. Notice that with the relative error tolerance $\epsilon = 0.1$, the image is largely compressed ($\sim$ 7X size reduction), with little loss of quality (see Figure 10). And, the singular value of the AMiner matrix decays very slowly, but even with large approximation error its low-rank approximation could bring improved performance of information retrieval (cf. [20], section 11.3).

For the second data, which is a large sparse matrix, the Krylov subspace iterative method `svds` [18, 20] is also tested. However, it costs 2,281 seconds for computing the first 1,000 singular values/vectors. It is much slower than executing `svd` to the matrix's dense version. Besides, `svd` requests more than 20 GB memory, while the proposed randomized algorithms only costs 3 GB memory or so for this case.

6. Conclusions. Efficient techniques are proposed for the fixed-precision low-rank approximation of large matrices. Our contributions are as follows.

- A simple and accurate error indicator in the Frobenius norm is proposed, which enables efficient rank determination and can be used in the blocked randQB algorithm [1] and other incremental QB-form factorization algorithms (like that in [14]). We have proved its accuracy and validity for the problems with relative accuracy tolerance larger than 2.1×10^{-7}. Numerical experiments on large dense and sparse matrices have shown that the proposed rank determination scheme brings several to several tens times speedup and memory saving to the blocked randQB algorithm, without loss of accuracy.
- Based on the blocked randQB algorithm, we propose a pass-efficient algorithm called `randQB_FP`. It is mathematically equivalent to the blocked randQB algorithm, but reduces the passes over matrix $\boldsymbol{A}$ to the fewest. The `randQB_FP` algorithm also suits the fixed-precision problem and can derive a single-pass algorithm under a certain condition. Numerical results have validated the efficiency and accuracy of the `randQB_FP` algorithm and shown that the derived single-pass algorithm is much more accurate than an existing counterpart.
- Real data are tested to demonstrate the effectiveness of the proposed algorithms for the fixed-precision problem. Compared with the adaptive range finder approach [2], the proposed algorithms run faster and produce much smaller factor matrices while attaining the accuracy criterion.

Future work includes extending and applying the proposed algorithms to more practical data mining and machine learning scenarios.

Acknowledgments. The authors thank Prof. P.-G. Martinsson for sharing the source code of the blocked randQB algorithm, which makes this work possible. The authors are also grateful to Prof. Jie Tang with Tsinghua University for providing the data of Aminer.

REFERENCES

[1] P.-G. Martinsson and S. Voronin, *A randomized blocked algorithm for efficiently computing rank-revealing factorizations of matrices*, SIAM J. Sci. Comput., 38 (2016), pp. S485–S507.

[2] 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.

[3] P. Drineas and M. W. Mahoney, *RandNLA: Randomized numerical linear algebra*, Commun. ACM, 59 (2016), pp. 80–90.

[4] S. Voronin and P.-G. Martinsson, *RSVDPACK: Subroutines for Computing Partial Singular Value Decompositions Via Randomized Sampling on Single Core, Multi Core, and GPU Architectures*, preprint, arXiv:1502.05366v3 [math.NA], 2016.

[5] S. Eriksson-Bique, M. Solbrig, M. Stefanelli, S. Warkentin, R. Abbey, and I. C. F. Ipsen, *Importance sampling for a Monte Carlo matrix multiplication algorithm, with application to information retrieval*, SIAM J. Sci. Comput., 33 (2011), pp. 1689–1706.

[6] P. Drineas, R. Kannan, and M. W. Mahoney, *Fast Monte Carlo algorithms for matrices* II: *Computing a low-rank approximation to a matrix*, SIAM J. Sci. Comput., 36 (2006), pp. 158–183.

[7] V. Rokhlin, A. Szlam, and M. Tygert, *A randomized algorithm for principal component analysis*, SIAM J. Matrix Anal. Appl., 31 (2009), pp. 1100–1124.
[8] H. Ji and Y. Li, *GPU accelerated randomized singular value decomposition and its application in image compression*, in Proceedings of the Modeling, Simulation, and Visualization Capstone Conference, Sulfolk, VA, 2014.
[9] T. Mary, I. Yamazaki, J. Kurzak, P. Luszczek, S. Tomov, and J. Dongarra, *Performance of random sampling for computing low-rank approximations of a dense matrix on GPUs*, in Proceedings of SC'2015, Austin, TX, 2015.
[10] M. W. Mahoney, *Randomized algorithms for matrices and data*, Found. Trends Mach. Learn., 3 (2011), pp. 123–224.
[11] G. H. Golub and C. F. Van Loan, *Matrix Computations*, 3rd ed., Johns Hopkins University Press, Baltimore, MD, 1996.
[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] P. G. Martinsson, G. Quintana-Orti, N. Heavner, and R. van de Geijn, *Householder QR factorization with randomization for column pivoting (HQRRP)*, SIAM J. Sci. Comput., 39 (2017), pp. C96–C115.
[14] J. A. Duersch and M. Gu, *Randomized QR with column pivoting*, SIAM J. Sci. Comput., 39 (2017), pp. C263–C291.
[15] C. Eckart and G. Young, *The approximation of one matrix by another of lower rank*, Psychometrika, 1 (1936), pp. 211–218.
[16] S. Voronin and P. G. Martinsson, RandQR, http://amath.colorado.edu/faculty/martinss/main_codes.html, 2015.
[17] Intel Parallel Studio XE Cluster Edition for Linux, https://software.intel.com/en-us/intel-parallel-studio-xe, 2017.
[18] Y. Saad, *Iterative Methods for Sparse Linear Systems*, SIAM, Philadelphia, 2003.
[19] *AMiner*, http://www.aminer.org.
[20] L. Elden, *Matrix Methods in Data Mining and Pattern Recognition*, SIAM, Philadelphia, 2007.
[21] R. Lehoucq, D. Sorensen, and C. Yang, *ARPACK User's Guide: Solution of Large-Scale Eigenvalue Problems with Implicitly Restarted Arnoldi Methods*, SIAM, Philadelphia, 1998.
[22] J. A. Tropp, A. Yurtsever, M. Udell, and V. Cevher, *Practical sketching algorithms for low-rank matrix approximation*, SIAM J. Matrix Anal. Appl., 38 (2017), pp. 1454–1485.
[23] H. Li, G. C. Linderman, A. Szlam, K. P. Stanton, Y. Kluger, and M. Tygert, *Algorithm* 971: *An implementation of a randomized algorithm for principal component analysis*, ACM Trans Math Software, 43 (2017), 28.
[24] W. Yu and X. Wang, *Advanced Field-Solver Techniques for RC Extraction of Integrated Circuits*, Springer, New York, 2014.