A Fast Adaptive Randomized PCA Algorithm

Xu Feng and **Wenjian Yu**[*]
Department of Computer Science and Technology, BNRist, Tsinghua University, Beijing, China
fx17@mails.tsinghua.edu.cn, yu-wj@tsinghua.edu.cn

Abstract

It is desirable to adaptively determine the number of dimensions (rank) for PCA according to a given tolerance of low-rank approximation error. In this work, we aim to develop a fast algorithm solving this adaptive PCA problem. We propose to replace the QR factorization in randQB_EI algorithm with matrix multiplication and inversion of small matrices, and propose a new error indicator to incrementally evaluate approximation error in Frobenius norm. Combining the shifted power iteration technique for better accuracy, we finally build up an algorithm named **farPCA**. Experimental results show that farPCA is much faster than the baseline methods (randQB_EI, randUBV and `svds`) in practical setting of multi-thread computing, while producing nearly optimal results of adpative PCA.

1 Introduction

Principal component analysis (PCA) is widely used for dimensionality reduction and embedding of input data in applications of machine learning. However, the application of PCA to real-world problems which require processing large data accurately, often costs prohibitive computational time. Accelerating the PCA computing (i.e. truncated SVD) for large data is of an absolute necessity.

Randomized matrix decomposition has gained significant increases in popularity with the rapid increasing of the size of data. It has been revealed that randomization can be a powerful computational resource for developing algorithms with improved runtime and stability properties [Halko *et al.*, 2011; Drineas and Mahoney, 2016; Martinsson and Tropp, 2020]. Compared with traditional algorithms, the randomized algorithm involves the same or fewer floating-point operations (flops), and is more efficient for large high-dimensional data sets, by exploiting modern computing architectures. An idea of randomization is using random embedding to identify the subspace capturing the dominant actions of a matrix $\mathbf{A}$ [Martinsson and Tropp, 2020] as $\mathbf{A\Omega}$, where $\mathbf{\Omega}$ is the random matrix for embedding. Usually, $\mathbf{\Omega}$ is a Guassian i.i.d random matrix [Halko *et al.*, 2011]. And, there are some variants of random matrix to accelerate the computation of $\mathbf{A\Omega}$, such as subsampled random Fourier transform matrix [Woolfe *et al.*, 2008], subsampled random Hadamard transform matrix [Ailon and Chazelle, 2006], and sparse sign matrix [Martinsson and Tropp, 2020]. With the subspace's orthonormal basis matrix $\mathbf{Q} = \mathrm{orth}(\mathbf{A\Omega})$, a so-called QB approximation is obtained: $\mathbf{A} \approx \mathbf{QB}$. This produces a smaller sketch matrix $\mathbf{B}$, and facilitates the computation of near-optimal approximation of $\mathbf{A}$ and further its PCA or SVD.

The basic randomized SVD algorithm was presented in [Halko *et al.*, 2011]. Later on, a lot of variants were proposed for the application problems like image processing [Benjamin Erichson *et al.*, 2017], network embedding [Zhang *et al.*, 2019b; Qiu *et al.*, 2021], streaming data processing [Yu *et al.*, 2017; Zhang *et al.*, 2019a], etc. The algorithm was also accelerated for better trade-off against accuracy or for large sparse matrices in application areas [Li *et al.*, 2017; Voronin and Martinsson, 2015; Bjarkason, 2019; Feng *et al.*, 2018b; Feng *et al.*, 2018a; Ding *et al.*, 2020].

Usually, the problem of low-rank matrix approximation falls into two categories [Yu *et al.*, 2018]:

- the fixed-rank problem, where the rank parameter k (i.e. the number of columns of $\mathbf{Q}$) is given.
- the fixed-precision (or fixed-accuracy, fixed-error) problem, where we seek the low-rank approximation with rank as small as possible providing the approximation error is within a given tolerance (e.g. $\|\mathbf{A} - \mathbf{QB}\| \le \varepsilon$).

Most exiting algorithms, including the golden standard `svds` in Matlab, solve the fixed-rank problem. The algorithm for the fixed-precision problem is relatively rarely seen, but it is essential for the PCA applications where the most suitable number of dimensions (rank k) is unknown. Usually, a smaller k causes inaccuracy in subsequent application, while a larger k induces large computational cost.

The fixed-precision problem of randomized QB factorization was addressed in [Martinsson and Voronin, 2016; Yu *et al.*, 2018]. The proposed algorithms are mathematically equivalent to the basic randomized algorithm in [Halko *et al.*, 2011], but allow incremental computation with increase of rank k. Based on an approximation error indicator which can be efficiently calculated, a randQB_EI algorithm was proposed in [Yu *et al.*, 2018] to enable the adaptive computation of PCA (or truncated SVD) subject to a given approximation error. Another algorithm for the fixed-precision prob-

[*]Corresponding author.

lem was proposed (with name randUBV) in [Hallman, 2022], based on the block Lanczos bidiagonalization process [Golub *et al.*, 1981]. Although randUBV costs less runtime than randQB_EI, it lacks the flexibility to produce more accurate results as there is not a mechanism like the power iteration in randQB_EI. Besides, the randUBV computes a much higher dimensional subspace than the rank finally determined, which implies the memory cost of randUBV is relatively larger. Therefore, randQB_EI is more suitable for some real applications with large sparse matrices. It is currently the foundation for the Matlab function `svdsketch`. An improvement and application of randQB_EI was presented in [Ding *et al.*, 2020], where the fixed-precision randomized algorithm is used to automatically determine a number of latent factors for achieving the near-optimal prediction accuracy during the subsequent model-based collaborative filtering. However, the algorithm in [Ding *et al.*, 2020] was implemented in Python and the comparison was made on CPU time, which makes it not validated for actual multi-thread computing. And, that algorithm has a flaw resulting in the accuracy issue when more power iterations are preformed.

In this work, our aim is to develop a fast adaptive randomized PCA algorithm to handle real-world dense or sparse matrices with multi-thread computing. Firstly, we remove the QR factorization (which has poor parallel efficiency[1]) in randQB_EI algorithm, while employing a new formula to evaluate the approximation error in the Frobenius norm during the iterative process. Then, we add the power iteration into the algorithm in a more efficient manner to enable the fixed-precision factorization for better quality. Finally, combing the shifted power iteration technique [Feng *et al.*, 2022] for better accuracy, we build up a fast adaptive randomized PCA (**farPCA**) algorithm. The codes of farPCA are shared on GitHub (https://github.com/XuFengthucs/farPCA).

Experimental results show that, if fixing rank k farPCA produces more accurate result than randQB_EI with same number of power iterations (p), and the reduction of error is up to 2.9X with p=10. Compared with randUBV, farPCA costs similar runtime and produce results with lower rank, while owning the flexibility to produce better approximation with more power iterations. For the adaptive PCA problem, the farPCA algorithm costs less runtime than randQB_EI on all test cases, and the speed-up ratio is up to **1.9X**. Compared with `svds`, the farPCA implemented in C produces nearly the same optimal results while reducing the runtime by **16X** through **96X** and reducing the memory cost by 75%.

2 Preliminaries

We follow Matlab conventions in pseudo-codes of algorithm.

2.1 Singular Value Decomposition and PCA

Suppose matrix $\mathbf{A} \in \mathbb{R}^{m\times n}$. Without loss of generality, we assume $m \geq n$. The economic SVD of $\mathbf{A}$ is $\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^{\mathrm{T}}$, where $\mathbf{U} = [\mathbf{u}_1, \mathbf{u}_2, \cdots, \mathbf{u}_n]$ and $\mathbf{V} = [\mathbf{v}_1, \mathbf{v}_2, \cdots, \mathbf{v}_n]$ are orthonormal matrices with left and right singular vectors respectively, and the diagonal matrix $\mathbf{\Sigma} \in \mathbb{R}^{n\times n}$ contains the singular values $\sigma_1, \sigma_2, \cdots$, in descending order. Suppose $\mathbf{U}_k$ and $\mathbf{V}_k$ are matrices with the first k columns of $\mathbf{U}$ and $\mathbf{V}$, and $\mathbf{\Sigma}_k$ is the $k \times k$ upper-left submatrix of $\mathbf{\Sigma}$. Then, $\mathbf{A}$ is approximated by its truncated SVD $\mathbf{A}_k$:

$$\mathbf{A} \approx \mathbf{A}_k = \mathbf{U}_k\mathbf{\Sigma}_k\mathbf{V}_k^{\mathrm{T}}. \tag{1}$$

Notice that $\mathbf{A}_k$ is the best rank-k approximation of $\mathbf{A}$ in either spectral norm or Frobenius norm [Eckart and Young, 1936].

The approximation properties of the SVD explain the equivalence between SVD and PCA. Suppose each row of matrix $\mathbf{A}$ is an observed data. The matrix is assumed to be centered, i.e., the mean of each column is equal to zero. Then, the leading left singular vectors $\{\mathbf{u}_i\}$ of $\mathbf{A}$ represent the principal components. Specifically, $\mathbf{u}_1$ is the normalized first principal component.

2.2 Randomized Algorithms for PCA and Fixed-Precision Matrix Factorization

The basic randomized SVD algorithm [Halko *et al.*, 2011] can be described as Algorithm 1, where $\mathbf{\Omega}$ is a Gaussian i.i.d random matrix, and the orthonormalization operation "orth($\cdot$)" can be implemented with a call to a packaged QR factorization. The power iteration in Step 3 through 5 is for improving the accuracy of result, where the orthonormalization alleviating the round-off error in floating-point computation is performed after every matrix-matrix multiplication.

With the power iteration not considered, the $m\times l$ orthonormal matrix $\mathbf{Q} = \mathrm{orth}(\mathbf{A}\mathbf{\Omega})$ includes the orthogonal basis vectors of approximate dominant subspace of $range(\mathbf{A})$, i.e., $span\{\mathbf{u}_1, \mathbf{u}_2, \cdots, \mathbf{u}_l\}$. Therefore, $\mathbf{A} \approx \mathbf{Q}\mathbf{Q}^{\mathrm{T}}\mathbf{A} = \mathbf{Q}\mathbf{B}$ according to Step 6. When the economic SVD is performed on the short-and-fat $l \times n$ matrix $\mathbf{B}$, the approximate truncated SVD of $\mathbf{A}$ is finally obtained. Employing the power iteration, one obtains $\mathbf{Q} = \mathrm{orth}((\mathbf{A}\mathbf{A}^{\mathrm{T}})^p\mathbf{A}\mathbf{\Omega})$, if the intermediate orthonormalization steps are ignored. This makes $\mathbf{Q}$ better approximate the basis of dominant subspace of $range((\mathbf{A}\mathbf{A}^{\mathrm{T}})^p\mathbf{A})$, same as that of $range(\mathbf{A})$, because $(\mathbf{A}\mathbf{A}^{\mathrm{T}})^p\mathbf{A}$'s singular values decay more quickly than those of $\mathbf{A}$ [Halko *et al.*, 2011]. So, the computed results are more accurate, and the larger p makes more accurate results and more computational cost as well.

Algorithm 1 Basic randomized PCA with power iteration

```
Input: $\mathbf{A} \in \mathbb{R}^{m\times n}$, rank parameter $k$, oversampling parameter $s$, power parameter $p$
Output: $\mathbf{U} \in \mathbb{R}^{m\times k}$, $\mathbf{S} \in \mathbb{R}^{k\times k}$, $\mathbf{V} \in \mathbb{R}^{n\times k}$
1: $l \leftarrow k + s,\ \mathbf{\Omega} \leftarrow \mathrm{randn}(n, l)$
2: $\mathbf{Q} \leftarrow \mathrm{orth}(\mathbf{A}\mathbf{\Omega})$
3: for $j \leftarrow 1, 2, \cdots, p$ do
4:     $\mathbf{G} \leftarrow \mathrm{orth}(\mathbf{A}^{\mathrm{T}}\mathbf{Q}),\ \mathbf{Q} \leftarrow \mathrm{orth}(\mathbf{A}\mathbf{G})$
5: end for
6: $\mathbf{B} \leftarrow \mathbf{Q}^{\mathrm{T}}\mathbf{A}$
7: $[\mathbf{U}, \mathbf{S}, \mathbf{V}] \leftarrow \mathrm{svd}(\mathbf{B}, '\mathrm{econ}')$
8: $\mathbf{U} \leftarrow \mathbf{Q}\mathbf{U}(:, 1{:}k),\ \mathbf{S} \leftarrow \mathbf{S}(1{:}k, 1{:}k),\ \mathbf{V} \leftarrow \mathbf{V}(:, 1{:}k)$
```

[1]Although there are highly-efficient paralell QR algorithms for tall and skinny matrices, e.g. [Demmel *et al.*, 2012], they have not been employed in Matlab or other libraries yet.

In [Yu *et al.*, 2018], a randomized QB factorization algorithm for the fixed-precision problem was proposed, where $\mathbf{Q}$ and $\mathbf{B}$ are sought to ensure $\|\mathbf{A}-\mathbf{QB}\|_F < \varepsilon$. It is a variant of the procedure producing $\mathbf{Q}$ and $\mathbf{B}$ in Alg. 1 executed in an incremental manner, with an efficient technique to calculate the Frobenius norm of approximation error. It is described as Alg. 2, where a block Gram-Schmidt procedure is used to realize the orthonormalization to compute $\mathbf{Q}$ in Alg. 1. This enables the incremental construction of the QB factorization. Due to $\|\mathbf{A}-\mathbf{QB}\|_F^2 = \|\mathbf{A}\|_F^2 - \|\mathbf{B}\|_F^2$, the approximation error in Frobenious norm can be evaluated easily for large or sparse $\mathbf{A}$ (by Steps 2 and 12 in Alg. 2) [Yu *et al.*, 2018]. Besides, the power iteration on matrix $\mathbf{A}-\mathbf{QB}$ is applied in Alg. 2 to enhance the accuracy of result. Notice that because the power iteration in Ding's Algorithm 3 [Ding *et al.*, 2020] approximates the dominant subspace of $range(\mathbf{A})$ instead of that of $range(\mathbf{A}-\mathbf{QB})$ in randQB_EI, it leads to inaccurate results when p increases. This is validated by the experimental results in the Appendix. The error tolerance is often set $\varepsilon = \epsilon\|\mathbf{A}\|_F$, $\epsilon \in (0,1)$. For obtaining the results of PCA, Steps 7 and 8 in Alg. 1 can be executed afterwards. And, a post-processing step can be done to determine the smallest rank r meeting the error tolerance.

We use the count of floating-point operations (*flops*) to analyze the time cost of algorithm. The *flop* count of Alg. 2 with Step 7 and 8 in Alg. 1 is

$$\begin{aligned}\mathrm{FC}_2 =& 2C_{mul}nnz(\mathbf{A})k + C_{mul}(2m+n)k^2 + 2C_{qr}mkb \\ &+ C_{svd}nk^2 + p(C_{mul}nnz(\mathbf{A})(k+\frac{k^2}{b}) \\ &+ C_{mul}(m+n)(k-b)^2 + C_{qr}(m+n)kb),\end{aligned} \tag{2}$$

where $2C_{mul}nnz(\mathbf{A})k$ reflects the matrix-matrix multiplication on $\mathbf{A}$ in Step 4 and 10, $C_{mul}(2m+n)k^2$ reflects the matrix-matrix multiplication of dense matrices, $2C_{qr}mkb$ reflects the QR factorization in Step 4 and 9, $C_{svd}nk^2$ reflects the economic SVD on $\mathbf{B}$, and $p(C_{mul}nnz(\mathbf{A})(k+\frac{k^2}{b}) + C_{mul}(m+n)(k-b)^2 + C_{qr}(m+n)kb)$ reflects the operations in power iteration in Step 5 through 8.

3 Fast Adaptive PCA Based on Randomized Matrix Factorization Skills

As QR factorization used in randQB_EI has relatively poor efficiency in multi-thread computing, our idea for acceleration is to replace it with matrix-matrix multiplication and the inversion of small matrix, which are of better parallel efficiency. We first accelerate the randQB_EI without power iteration, with a new formula to evaluate the approximation error in Frobenius norm. Then, we add the power iteration into the algorithm in an more efficient manner, where the employment of QR is reduced to the least. Finally, combing the shifted power iteration technique with dynamic shifts [Feng *et al.*, 2022] and the proposed techniques for accelerating randQB_EI for fix-precision factorization, we propose the fast adaptive randomized PCA algorithm (named farPCA) with better quality of results.

Algorithm 2 randQB_EI with power iteration

```
Input: $\mathbf{A} \in \mathbb{R}^{m\times n}$, error tolerance $\varepsilon$, block size $b$, power
    parameter $p$
Output: $\mathbf{Q}\in\mathbb{R}^{m\times k}, \mathbf{B}\in\mathbb{R}^{k\times n}$, $s.t.$ $\|\mathbf{A}-\mathbf{QB}\|_F<\varepsilon$
 1: $\mathbf{Q} \leftarrow [\,],\ \mathbf{B} \leftarrow [\,]$
 2: $E \leftarrow \|\mathbf{A}\|_F^2,\ tol \leftarrow \varepsilon^2$
 3: for $i \leftarrow 1,2,\cdots$ do
 4:     $\mathbf{\Omega}_i \leftarrow \mathrm{randn}(n,b),\ \mathbf{Q}_i \leftarrow \mathrm{orth}(\mathbf{A\Omega}_i - \mathbf{QB\Omega}_i)$
 5:     for $j \leftarrow 1,2,\cdots,p$ do
 6:         $\mathbf{G}_i \leftarrow \mathrm{orth}(\mathbf{A}^{\mathrm{T}}\mathbf{Q}_i - \mathbf{B}^{\mathrm{T}}\mathbf{Q}^{\mathrm{T}}\mathbf{Q}_i)$
 7:         $\mathbf{Q}_i \leftarrow \mathrm{orth}(\mathbf{AG}_i - \mathbf{QBG}_i)$
 8:     end for
 9:     $\mathbf{Q}_i \leftarrow \mathrm{orth}(\mathbf{Q}_i - \mathbf{QQ}^{\mathrm{T}}\mathbf{Q}_i)$
10:     $\mathbf{B}_i \leftarrow \mathbf{Q}_i^{\mathrm{T}}\mathbf{A}$
11:     $\mathbf{Q} \leftarrow [\mathbf{Q},\mathbf{Q}_i],\ \mathbf{B} \leftarrow [\mathbf{B};\mathbf{B}_i]$
12:     $E \leftarrow E - \|\mathbf{B}_i\|_F^2$
13:     if $E < tol$ then break
14: end for
```

3.1 Remove QR from the randQB_EI Algorithm

According to Alg. 2 without power iteration, the QR factorization has poor efficiency in multi-thread computing. So, how to replace the the QR factorization with other operations for better parallel efficiency is the focus. According to the following proposition, we can remove QR factorization to produce $\mathbf{Q}$ and $\mathbf{B}$ with the same accuracy of approximation as Alg. 1 without the power iteration.

Proposition 1. *Suppose* $\mathbf{A} \in \mathbb{R}^{m\times n}$, $\mathbf{\Omega} \in \mathbb{R}^{n\times k}$ *is a Gaussian i.i.d random matrix,* $\mathbf{Y} = \mathbf{A\Omega}$, $\mathbf{W} = \mathbf{A}^{\mathrm{T}}\mathbf{Y}$ *and the economic SVD of* $\mathbf{Y}$ *is* $\mathbf{Y} = \hat{\mathbf{U}}\hat{\mathbf{\Sigma}}\hat{\mathbf{V}}^{\mathrm{T}}$. *Then, setting*

$$\mathbf{Q} = \mathbf{Y}\hat{\mathbf{V}}\hat{\mathbf{\Sigma}}^{-1}, \mathbf{B} = (\mathbf{W}\hat{\mathbf{V}}\hat{\mathbf{\Sigma}}^{-1})^{\mathrm{T}}. \tag{3}$$

produces the approximation of $\mathbf{QB}$ *to* $\mathbf{A}$ *which is of same accuracy as that in Alg. 1 without power iteration and oversampling. And, if* $tr(\cdot)$ *denotes the trace of a matrix,*

$$\|\mathbf{B}\|_F^2 = tr(\mathbf{W}^{\mathrm{T}}\mathbf{W}(\mathbf{Y}^{\mathrm{T}}\mathbf{Y})^{-1}). \tag{4}$$

Proof. Because $\hat{\mathbf{U}}$ is the matrix with left singular vectors of $\mathbf{Y}$, $\hat{\mathbf{U}}$ is also the solution of $\mathrm{orth}(\mathbf{Y})$. Therefore, we can set

$$\mathbf{Q} = \mathrm{orth}(\mathbf{A\Omega}) = \mathrm{orth}(\mathbf{Y}) = \hat{\mathbf{U}} = \mathbf{Y}\hat{\mathbf{V}}\hat{\mathbf{\Sigma}}^{-1}. \tag{5}$$

Then, combing (5) and $\mathbf{W} = \mathbf{A}^{\mathrm{T}}\mathbf{Y}$ we have

$$\mathbf{B} = \mathbf{Q}^{\mathrm{T}}\mathbf{A} = (\mathbf{W}\hat{\mathbf{V}}\hat{\mathbf{\Sigma}}^{-1})^{\mathrm{T}}. \tag{6}$$

Because $\mathbf{Q}$ is the orthonormalization of $\mathbf{A\Omega}$, this approximation $\mathbf{QB}$ performs with the same accuracy as $\mathbf{A} \approx \mathbf{QB}$ in Alg. 1 without power iteration and oversampling. Then,

$$\begin{aligned}\|\mathbf{B}\|_F^2 &= \|\mathbf{W}\hat{\mathbf{V}}\hat{\mathbf{\Sigma}}^{-1}\|_F^2 = tr(\hat{\mathbf{\Sigma}}^{-1}\hat{\mathbf{V}}^{\mathrm{T}}\mathbf{W}^{\mathrm{T}}\mathbf{W}\hat{\mathbf{V}}\hat{\mathbf{\Sigma}}^{-1}) \\ &= tr(\mathbf{W}^{\mathrm{T}}\mathbf{W}\hat{\mathbf{V}}\hat{\mathbf{\Sigma}}^{-2}\hat{\mathbf{V}}^{\mathrm{T}}) = tr(\mathbf{W}^{\mathrm{T}}\mathbf{W}(\mathbf{Y}^{\mathrm{T}}\mathbf{Y})^{-1}).\end{aligned} \tag{7}$$

□

In Proposition 1, Eq. (3) reflects that we can compute $\mathbf{Y} = \mathbf{A\Omega}$ and $\mathbf{W} = \mathbf{A}^{\mathrm{T}}\mathbf{Y}$ incrementally to generate the

Algorithm 3 Faster randQB_EI without power iteration

```
Input: $\mathbf{A} \in \mathbb{R}^{m\times n}$, error tolerance $\varepsilon$, block size $b$
Output: $\mathbf{Q}\in\mathbb{R}^{m\times k}$, $\mathbf{B}\in\mathbb{R}^{k\times n}$, $s.t.$ $\|\mathbf{A}-\mathbf{QB}\|_F<\varepsilon$
 1: $\mathbf{Y} \leftarrow [\,]$, $\mathbf{W} \leftarrow [\,]$
 2: $E \leftarrow \|\mathbf{A}\|_F^2$, $tol \leftarrow \varepsilon^2$
 3: for $i \leftarrow 1, 2, \cdots$ do
 4:     $\mathbf{\Omega}_i \leftarrow \mathrm{randn}(n, b)$
 5:     $\mathbf{Y}_i \leftarrow \mathbf{A}\mathbf{\Omega}_i$, $\mathbf{W}_i \leftarrow \mathbf{A}^{\mathrm{T}}\mathbf{Y}_i$
 6:     $\mathbf{Y} \leftarrow [\mathbf{Y}, \mathbf{Y}_i]$, $\mathbf{W} \leftarrow [\mathbf{W}, \mathbf{W}_i]$
 7:     $\mathbf{Z} \leftarrow \mathbf{Y}^{\mathrm{T}}\mathbf{Y}$, $\mathbf{T} \leftarrow \mathbf{W}^{\mathrm{T}}\mathbf{W}$
 8:     if $E - tr(\mathbf{T}\mathbf{Z}^{-1}) < tol$ then break
 9: end for
10: $[\hat{\mathbf{V}}, \hat{\mathbf{D}}] \leftarrow \mathrm{eig}(\mathbf{Z})$
11: $\mathbf{Q} \leftarrow \mathbf{Y}\hat{\mathbf{V}}\mathrm{sqrt}(\hat{\mathbf{D}})^{-1}$, $\mathbf{B} \leftarrow (\mathbf{W}\hat{\mathbf{V}}\mathrm{sqrt}(\hat{\mathbf{D}})^{-1})^{\mathrm{T}}$
```

matrices $\mathbf{Q}$ and $\mathbf{B}$ at the end of process, instead of computing $\mathbf{Q}$ and $\mathbf{B}$ incrementally with QR factorization in Step 4 and 9 of Alg. 2. Therefore, QR factorization can be removed in randQB_EI. Besides, we can derive a new approach to evaluate the approximation error in the Frobenius norm according to (4) in Proposition 1. Combing (4) and $\|\mathbf{A}-\mathbf{QB}\|_F^2 = \|\mathbf{A}\|_F^2 - \|\mathbf{B}\|_F^2$ in randQB_EI yields

$$\|\mathbf{A}-\mathbf{QB}\|_F^2 = \|\mathbf{A}\|_F^2 - tr(\mathbf{W}^{\mathrm{T}}\mathbf{W}(\mathbf{Y}^{\mathrm{T}}\mathbf{Y})^{-1}), \quad (8)$$

which reflects $tr(\mathbf{W}^{\mathrm{T}}\mathbf{W}(\mathbf{Y}^{\mathrm{T}}\mathbf{Y})^{-1})$ can be used to evaluate the approximation error in the Frobenius norm. So, a faster randQB_EI without power iteration is derived in Algorithm 3.

In Alg. 3, matrices $\mathbf{Y}$ and $\mathbf{W}$ are computed incrementally in Step 5 and 6. Then, $\mathbf{Z} = \mathbf{Y}^{\mathrm{T}}\mathbf{Y}$ and $\mathbf{T} = \mathbf{W}^{\mathrm{T}}\mathbf{W}$ are computed in Step 7. Notice that $\mathbf{Z}$ and $\mathbf{T}$ can be computed incrementally in each iterative step to reduce the time cost. In Step 8, the new approach to evaluate the approximation error in the Frobenius norm is used according to (8). Usually, $\mathbf{T}\mathbf{Z}^{-1}$ is implemented by solving linear equations. When the results satisfy the error tolerance, the eigen-decomposition of $\mathbf{Z}$ is computed firstly in Step 10, and $\mathbf{Q}$ and $\mathbf{B}$ are computed in Step 11 according to (3) to replace the economic SVD of $\mathbf{Y}$ for reducing the time cost. Because the QR factorization in Alg. 2 is replaced by matrix-matrix multiplication in Step 7 and 11, solving linear equations of two small matrices in Step 8 and eigen-decomposition of a small matrix in Step 10 in Alg. 3, Alg. 3 is of better parallel efficiency.

3.2 Inclusion of Power Iteration

Although Alg. 3 is a faster randQB_EI procedure for fixed-precision problems, it lacks the flexibility to produce better approximation with the power iteration scheme. Our idea is to develop an efficient power iteration scheme with the computed matrices.

Suppose $\mathbf{H} = \mathbf{A} - \mathbf{QB}$. In the power iteration of Alg. 2, the matrix-matrix multiplication on the left and right side of $\mathbf{H}$ is performed in Step 6 and Step 7. Therefore, how to change the formulation of $\mathbf{QB}$ in $\mathbf{H}$ with the computed matrices in Alg. 3 is the focus. Because $\mathbf{Y}$, $\mathbf{W}$ and $\mathbf{Z} = \mathbf{Y}^{\mathrm{T}}\mathbf{Y}$ are computed in Alg. 3, combining (3) in Proposition 1 yields

$$\mathbf{QB} = \mathbf{Y}\hat{\mathbf{V}}\hat{\mathbf{\Sigma}}^{-2}\hat{\mathbf{V}}^{\mathrm{T}}\mathbf{W}^{\mathrm{T}} = \mathbf{Y}(\mathbf{Y}^{\mathrm{T}}\mathbf{Y})^{-1}\mathbf{W}^{\mathrm{T}}, \quad (9)$$

Algorithm 4 Faster fixed-precision QB factorization

```
Input: $\mathbf{A} \in \mathbb{R}^{m\times n}$, error tolerance $\varepsilon$, block size $b$, power
    parameter $p$
Output: $\mathbf{Q}\in\mathbb{R}^{m\times k}$, $\mathbf{B}\in\mathbb{R}^{k\times n}$, $s.t.$ $\|\mathbf{A}-\mathbf{QB}\|_F<\varepsilon$
 1: $\mathbf{Y} \leftarrow [\,]$, $\mathbf{W} \leftarrow [\,]$
 2: $E \leftarrow \|\mathbf{A}\|_F^2$, $tol \leftarrow \varepsilon^2$
 3: for $i \leftarrow 1, 2, \cdots$ do
 4:     $\mathbf{\Omega}_i \leftarrow \mathrm{randn}(n, b)$
 5:     for $j \leftarrow 1, 2, \cdots, p$ do
 6:         $\mathbf{W}_i \leftarrow \mathbf{A}^{\mathrm{T}}\mathbf{A}\mathbf{\Omega}_i - \mathbf{W}\mathbf{Z}^{-1}\mathbf{W}^{\mathrm{T}}\mathbf{\Omega}_i$
 7:         $\mathbf{\Omega}_i \leftarrow \mathrm{orth}(\mathbf{W}_i)$
 8:     end for
 9:     $\mathbf{Y}_i \leftarrow \mathbf{A}\mathbf{\Omega}_i$, $\mathbf{W}_i \leftarrow \mathbf{A}^{\mathrm{T}}\mathbf{Y}_i$
10:     $\mathbf{Y} \leftarrow [\mathbf{Y}, \mathbf{Y}_i]$, $\mathbf{W} \leftarrow [\mathbf{W}, \mathbf{W}_i]$
11:     $\mathbf{Z} \leftarrow \mathbf{Y}^{\mathrm{T}}\mathbf{Y}$, $\mathbf{T} \leftarrow \mathbf{W}^{\mathrm{T}}\mathbf{W}$
12:     if $E - tr(\mathbf{T}\mathbf{Z}^{-1}) < tol$ then break
13: end for
14: $[\hat{\mathbf{V}}, \hat{\mathbf{D}}] \leftarrow \mathrm{eig}(\mathbf{Z})$
15: $\mathbf{Q} \leftarrow \mathbf{Y}\hat{\mathbf{V}}\mathrm{sqrt}(\hat{\mathbf{D}})^{-1}$, $\mathbf{B} \leftarrow (\mathbf{W}\hat{\mathbf{V}}\mathrm{sqrt}(\hat{\mathbf{D}})^{-1})^{\mathrm{T}}$
```

which reflects that $\mathbf{QB}$ can be represented by $\mathbf{Y}\mathbf{Z}^{-1}\mathbf{W}^{\mathrm{T}}$. So, we can perform the power iteration with $\mathbf{H}=\mathbf{A}-\mathbf{Y}\mathbf{Z}^{-1}\mathbf{W}^{\mathrm{T}}$. Besides, we can reduce one orthonormalization in each step of power iteration as that in [Voronin and Martinsson, 2015; Feng *et al.*, 2018a] to reduce the time cost with little sacrifice on accuracy. With these developments, the following power iteration steps can be applied after Step 4 in Alg. 3

```
5: for $j \leftarrow 1, 2, \cdots, p$ do
6:     $\mathbf{Y}_i \leftarrow \mathbf{A}\mathbf{\Omega}_i - \mathbf{Y}\mathbf{Z}^{-1}\mathbf{W}^{\mathrm{T}}\mathbf{\Omega}_i$
7:     $\mathbf{W}_i \leftarrow \mathbf{A}^{\mathrm{T}}\mathbf{Y}_i - \mathbf{W}\mathbf{Z}^{-1}\mathbf{Y}^{\mathrm{T}}\mathbf{Y}_i$
8:     $\mathbf{\Omega}_i \leftarrow \mathrm{orth}(\mathbf{W}_i)$
9: end for
```

In each step of the power iteration, $\mathbf{W}_i = \mathbf{H}^{\mathrm{T}}\mathbf{H}\mathbf{\Omega}_i$ is computed. Combining $\mathbf{Y}_i \leftarrow \mathbf{A}\mathbf{\Omega}_i - \mathbf{Y}\mathbf{Z}^{-1}\mathbf{W}^{\mathrm{T}}\mathbf{\Omega}_i$ and $\mathbf{W}_i \leftarrow \mathbf{A}^{\mathrm{T}}\mathbf{Y}_i - \mathbf{W}\mathbf{Z}^{-1}\mathbf{Y}^{\mathrm{T}}\mathbf{Y}_i$ in Step 6 and Step 7 above, and the fact $\mathbf{W} = \mathbf{A}^{\mathrm{T}}\mathbf{Y}$, we can derive that

$$\begin{aligned}
&\mathbf{W}_i = \mathbf{A}^{\mathrm{T}}\mathbf{Y}_i - \mathbf{W}\mathbf{Z}^{-1}\mathbf{Y}^{\mathrm{T}}\mathbf{Y}_i \\
&= \mathbf{A}^{\mathrm{T}}(\mathbf{A}\mathbf{\Omega}_i - \mathbf{Y}\mathbf{Z}^{-1}\mathbf{W}^{\mathrm{T}}\mathbf{\Omega}_i) - \mathbf{W}\mathbf{Z}^{-1}\mathbf{Y}^{\mathrm{T}}(\mathbf{A}\mathbf{\Omega}_i - \mathbf{Y}\mathbf{Z}^{-1}\mathbf{W}^{\mathrm{T}}\mathbf{\Omega}_i) \\
&= \mathbf{A}^{\mathrm{T}}\mathbf{A}\mathbf{\Omega}_i - \mathbf{W}\mathbf{Z}^{-1}\mathbf{W}^{\mathrm{T}}\mathbf{\Omega}_i - \mathbf{W}\mathbf{Z}^{-1}\mathbf{W}^{\mathrm{T}}\mathbf{\Omega}_i + \mathbf{W}\mathbf{Z}^{-1}\mathbf{W}^{\mathrm{T}}\mathbf{\Omega}_i \\
&= \mathbf{A}^{\mathrm{T}}\mathbf{A}\mathbf{\Omega}_i - \mathbf{W}\mathbf{Z}^{-1}\mathbf{W}^{\mathrm{T}}\mathbf{\Omega}_i.
\end{aligned} \quad (10)$$

This means we can reduce two times of matrix-matrix multiplication on $\mathbf{Y}$ for $\mathbf{Y}(\mathbf{Z}^{-1}\mathbf{W}^{\mathrm{T}}\mathbf{\Omega}_i)$ and $\mathbf{Y}^{\mathrm{T}}\mathbf{Y}_i$ and once solution of linear equations for $\mathbf{Z}^{-1}(\mathbf{Y}^{\mathrm{T}}\mathbf{Y}_i)$ to compute $\mathbf{W}_i$ efficiently. Combining the efficient power iteration, the faster fixed-precision QB factorization is described in Algorithm 4.

In Step 6 of Alg. 4, the computation of $\mathbf{W}_i$ is simplified as (10). And, the QR factorization is just used once in Step 7 to alleviate the round-off error in floating-point computation. Compared with Alg. 2, Alg. 4 is of better parallel efficiency due to fewer times of QR factorization and matrix-matrix multiplication and more solutions of linear equations for two small matrices in the power iteration.

3.3 Shifted Power Iteration for Better Accuracy

When computing $\mathbf{W}_i = \mathbf{H}^{\mathrm{T}}\mathbf{H}\mathbf{\Omega}_i$ with $\mathbf{H} = \mathbf{A}–\mathbf{Y}\mathbf{Z}^{-1}\mathbf{W}^{\mathrm{T}}$ in the power iteration of Alg. 4, the shift power iteration scheme in [Feng *et al.*, 2022] can be applied to improve the accuracy. The following lemma [Feng *et al.*, 2022] states how to use the shift properly to perform the shifted power iteration.

Lemma 1. *Suppose* $0 < \alpha \leq \sigma_b(\mathbf{H}^{\mathrm{T}}\mathbf{H})/2$ *and* $i \leq b$*, where* $\sigma_i(\cdot)$ *denotes the i-th singular value. Then,* $\sigma_i(\mathbf{H}^{\mathrm{T}}\mathbf{H} - \alpha\mathbf{I}) = \sigma_i(\mathbf{H}^{\mathrm{T}}\mathbf{H}) - \alpha$*. And, the left singular vector corresponding to the i-th singular value of* $\mathbf{H}^{\mathrm{T}}\mathbf{H} - \alpha\mathbf{I}$ *is the same as the left singular vector corresponding to the i-th singular value of* $\mathbf{H}^{\mathrm{T}}\mathbf{H}$.

Lemma 1 shows that, if we choose a positive shift $\alpha \leq \sigma_b(\mathbf{H}^{\mathrm{T}}\mathbf{H})/2$, the computation $\mathbf{H}^{\mathrm{T}}\mathbf{H}\mathbf{\Omega}_i$ can be changed to $(\mathbf{H}^{\mathrm{T}}\mathbf{H} - \alpha\mathbf{I})\mathbf{\Omega}_i$ in the power iteration, with the same approximated dominant subspace. This is called the *shifted power iteration* in [Feng *et al.*, 2022], which can improve the accuracy with the same power parameter p. Because computing $\sigma_b(\mathbf{H}^{\mathrm{T}}\mathbf{H})$ directly is difficult, we can use the singular values of $\mathbf{H}^{\mathrm{T}}\mathbf{H}\mathbf{\Omega}_i$ to approximate $\sigma_b(\mathbf{H}^{\mathrm{T}}\mathbf{H})$ according to the following lemma [Feng *et al.*, 2022].

Lemma 2. *Suppose* $\mathbf{A} \in \mathbb{R}^{m\times m}$ *and* $\mathbf{\Omega}_i \in \mathbb{R}^{m\times b}$ $(b \leq \min(m, n))$ *is an orthonormal matrix. Then,*

$$\sigma_i(\mathbf{A}\mathbf{\Omega}_i) \leq \sigma_i(\mathbf{A})\ , \textit{for any } i \leq b\ . \tag{11}$$

Suppose $\mathbf{\Omega}_i \in \mathbb{R}^{n\times b}$ is the orthonormal matrix in power iteration of Alg. 4. According to Lemma 2,

$$\sigma_i(\mathbf{H}^{\mathrm{T}}\mathbf{H}\mathbf{\Omega}_i) \leq \sigma_i(\mathbf{H}^{\mathrm{T}}\mathbf{H}),\ i \leq b, \tag{12}$$

which means that we can set $\alpha = \sigma_b(\mathbf{H}^{\mathrm{T}}\mathbf{H}\mathbf{\Omega}_i)/2$ to guarantee the requirement of α in Lemma 1 for performing the shifted power iteration. In order to do the orthonormalization for alleviating round-off error and calculate $\sigma_b(\mathbf{H}^{\mathrm{T}}\mathbf{H}\mathbf{\Omega}_i)$, we implement "orth(·)" with the eigSVD algorithm from [Feng *et al.*, 2022]. eigSVD applies eigen-decomposition to compute the results of economic SVD efficiently, and the resulted matrix of left singular vectors includes the orthonormal basis of same subspace compared with QR factorization. Because $\mathbf{\Omega}_i$ at the first step of power iteration is not an orthonormal matrix, we obtain the value of α at the second step of power iteration, and then perform $\mathrm{eigSVD}((\mathbf{H}^{\mathrm{T}}\mathbf{H} - \alpha\mathbf{I})\mathbf{\Omega}_i)$ in the following iteration steps.

Because the singular values of $(\mathbf{H}^{\mathrm{T}}\mathbf{H}–\alpha\mathbf{I})\mathbf{\Omega}_i$ are computed in the shifted power iteration, according to Lemmas 1 and 2, combining $0 < \alpha \leq \sigma_b(\mathbf{H}^{\mathrm{T}}\mathbf{H})/2$ and $i \leq b$ yields

$$\sigma_i((\mathbf{H}^{\mathrm{T}}\mathbf{H} - \alpha\mathbf{I})\mathbf{\Omega}_i) + \alpha \leq \sigma_i(\mathbf{H}^{\mathrm{T}}\mathbf{H} - \alpha\mathbf{I}) + \alpha = \sigma_i(\mathbf{H}^{\mathrm{T}}\mathbf{H}), \tag{13}$$

which states how to use the singular values of $(\mathbf{H}^{\mathrm{T}}\mathbf{H}–\alpha\mathbf{I})\mathbf{\Omega}_i$ to approximate $\sigma_i(\mathbf{H}^{\mathrm{T}}\mathbf{H})$. Therefore, in each step of the shifted power iteration we can obtain a valid value of shift and update α with it if we have a larger α for better accuracy.

3.4 The Overall Algorithm

Apart from the shifted power iteration for better accuracy, we can accelerate the computation of economic SVD of $\mathbf{B}$ for faster approximate PCA. Inspired by eigSVD algorithm, we use the eigen-decomposition of $\mathbf{B}\mathbf{B}^{\mathrm{T}}$ to compute the economic SVD of $\mathbf{B}$. Practically, we use the computed $\mathbf{Z}$ and $\mathbf{T}$ to produce $\mathbf{B}\mathbf{B}^{\mathrm{T}}$ according to (3) as

$$\mathbf{B}\mathbf{B}^{\mathrm{T}} = (\mathbf{W}\hat{\mathbf{V}}\hat{\mathbf{\Sigma}}^{-1})^{\mathrm{T}}(\mathbf{W}\hat{\mathbf{V}}\hat{\mathbf{\Sigma}}^{-1}) = \hat{\mathbf{\Sigma}}^{-1}\hat{\mathbf{V}}^{\mathrm{T}}\mathbf{W}^{\mathrm{T}}\mathbf{W}\hat{\mathbf{V}}\hat{\mathbf{\Sigma}}^{-1}, \tag{14}$$

where $\hat{\mathbf{V}}$ and $\hat{\mathbf{\Sigma}}$ are the matrices with right singular vectors and singular values of $\mathbf{Y}$ computed by the eigen-decomposition of $\mathbf{Z}$. According to (14), we can firstly compute the eigen-decomposition of $k \times k$ $\mathbf{Z}$ and then compute $\hat{\mathbf{\Sigma}}^{-1}\hat{\mathbf{V}}^{\mathrm{T}}\mathbf{T}\hat{\mathbf{V}}\hat{\mathbf{\Sigma}}^{-1}$ with two times of matrix-matrix multiplication on two $k \times k$ matrices to compute $\mathbf{B}\mathbf{B}^{\mathrm{T}}$, which costs much less time compared with computing $\mathbf{B}\mathbf{B}^{\mathrm{T}}$ directly when $k \ll n$. After executing the eigen-decompostion of $\mathbf{B}\mathbf{B}^{\mathrm{T}}$: $[\tilde{\mathbf{V}}, \tilde{\mathbf{D}}] = \mathrm{eig}(\mathbf{B}\mathbf{B}^{\mathrm{T}})$, we can compute the approximate PCA results of $\mathbf{A}$ as

$$\mathbf{S} \leftarrow \mathrm{sqrt}(\tilde{\mathbf{D}}),\ \mathbf{U} \leftarrow \mathbf{Q}\tilde{\mathbf{V}},\ \mathbf{V} \leftarrow \mathbf{B}^{\mathrm{T}}\tilde{\mathbf{V}}\mathbf{S}^{-1}. \tag{15}$$

Combining this fast approach to produce approximate PCA and the shifted power iteration scheme with dynamic shift, we propose the fast adaptive randomized PCA in Algorithm 5 named farPCA. In Alg. 5, the shifted power iteration is used in Step 6 to improve the accuracy. In Step 7, the eigSVD algorithm is used to compute both the left singular vectors and the singular values. The shift is updated dynamically when the new shift is larger according to (13) in Step 8. Finally, the computations in Step 15 through 19 are used to compute the approximate PCA of $\mathbf{A}$ efficiently according to (14) and (15). In order to find better PCA result, a post-processing step which searches the computed singular values to find the smallest r such that $\|\mathbf{A} - \mathbf{U}_r\mathbf{S}_r\mathbf{V}_r^{\mathrm{T}}\|_F < \varepsilon$ is actually exe-

Algorithm 5 Fast adaptive randomized PCA (farPCA)

```
Output: $\mathbf{A} \in \mathbb{R}^{m\times n}$, error tolerance $\varepsilon$, block size $b$, power
    parameter $p$
Input: $\mathbf{U} \in \mathbb{R}^{m\times k}$, $\mathbf{S} \in \mathbb{R}^{k\times k}$, $\mathbf{V} \in \mathbb{R}^{n\times k}$ such that $\|\mathbf{A} - \mathbf{U}\mathbf{S}\mathbf{V}^{\mathrm{T}}\|_F < \varepsilon$
 1: $\mathbf{Y} \leftarrow [\,],\ \mathbf{W} \leftarrow [\,]$
 2: $E \leftarrow \|\mathbf{A}\|_F^2,\ tol \leftarrow \varepsilon^2$
 3: for $i = 1, 2, \cdots$ do
 4:     $\mathbf{\Omega}_i \leftarrow \mathrm{randn}(n, b),\ \alpha \leftarrow 0$
 5:     for $j \leftarrow 1, 2, \cdots, p$ do
 6:         $\mathbf{W}_i \leftarrow \mathbf{A}^{\mathrm{T}}\mathbf{A}\mathbf{\Omega}_i - \mathbf{W}\mathbf{Z}^{-1}\mathbf{W}^{\mathrm{T}}\mathbf{\Omega}_i - \alpha\mathbf{\Omega}_i$
 7:         $[\mathbf{\Omega}_i, \hat{\mathbf{S}}, \sim] \leftarrow \mathrm{eigSVD}(\mathbf{W}_i)$
 8:         if ($j > 1$ and $\alpha < \hat{\mathbf{S}}(b, b)$) then $\alpha \leftarrow (\alpha + \hat{\mathbf{S}}(b, b))/2$
 9:     end for
10:     $\mathbf{Y}_i \leftarrow \mathbf{A}\mathbf{\Omega}_i,\ \mathbf{W}_i \leftarrow \mathbf{A}^{\mathrm{T}}\mathbf{Y}_i$
11:     $\mathbf{Y} \leftarrow [\mathbf{Y}, \mathbf{Y}_i],\ \mathbf{W} \leftarrow [\mathbf{W}, \mathbf{W}_i]$
12:     $\mathbf{Z} \leftarrow \mathbf{Y}^{\mathrm{T}}\mathbf{Y},\ \mathbf{T} \leftarrow \mathbf{W}^{\mathrm{T}}\mathbf{W}$
13:     if $E - tr(\mathbf{T}\mathbf{Z}^{-1}) < tol$ then break
14: end for
15: $[\hat{\mathbf{V}}, \hat{\mathbf{D}}] \leftarrow \mathrm{eig}(\mathbf{Z})$
16: $\mathbf{P} \leftarrow \hat{\mathbf{V}}\mathrm{sqrt}(\hat{\mathbf{D}})^{-1}$
17: $[\tilde{\mathbf{V}}, \tilde{\mathbf{D}}] \leftarrow \mathrm{eig}(\mathbf{P}^{\mathrm{T}}\mathbf{T}\mathbf{P})$ #calculate $\mathrm{eig}(\mathbf{B}\mathbf{B}^{\mathrm{T}})$
18: $\mathbf{S} \leftarrow \mathrm{sqrt}(\tilde{\mathbf{D}})$
19: $\mathbf{U} \leftarrow \mathbf{Y}\mathbf{P}\tilde{\mathbf{V}},\ \mathbf{V} \leftarrow \mathbf{W}\mathbf{P}\tilde{\mathbf{V}}\mathbf{S}^{-1}$
```

cuted [Yu *et al.*, 2018]. The *flop* count of Alg. 5 is

$$\begin{aligned}\mathrm{FC}_5 =&2C_{mul}nnz(\mathbf{A})k + C_{mul}(2m+2n)k^2\\ &+p(C_{mul}nnz(\mathbf{A})(k+\frac{k^2}{b}) + C_{mul}n(k-b)^2\\ &+2C_{mul}nkb),\end{aligned} \tag{16}$$

where $2C_{mul}nnz(\mathbf{A})k$ reflects the matrix-matrix multiplication on $\mathbf{A}$ in Step 10, C_{mul} $(2m+2n)k^2$ reflects the matrix-matrix multiplication in Step 12 and 19, and $p(C_{mul}$ $nnz(\mathbf{A})(k+\frac{k^2}{b}) + C_{mul}n(k-b)^2 + 2C_{mul}nkb)$ reflects the operations in power iteration in Step 5 through 9. Suppose we use eigSVD to replace the economic SVD in Alg. 2, the $C_{svd}nk^2$ in FC_2 is replaced by $2C_{mul}nk^2$. Then, we can derive

$$\begin{aligned}\mathrm{FC}_2-\mathrm{FC}_5 =&C_{mul}nk^2 + 2C_{qr}mkb + q(C_{mul}m(k-b)^2\\ &+C_{qr}(m+n)kb - 2C_{mul}nkb).\end{aligned} \tag{17}$$

Because of $C_{qr} > 2C_{mul}$ in practice, $\mathrm{FC}_2 - \mathrm{FC}_5 > 0$, which reflects the less *flop* count of farPCA.

4 Experiments

All experiments are carried out on a Ubuntu server with two 8-core Intel Xeon CPU (at 2.10 GHz) and 512 GB RAM. The proposed algorithms are implemented both in Matlab and in C with MKL[2] and OpenMP directives for multi-thread parallel computing. We use the shared codes of randUBV [3] and randQB_EI [4] for comparison. We also implemented randQB_EI in C with MKL and OpenMP. `svds` in Matlab 2020b is used for computing the accurate results. All the programs are evaluated with wall-clock runtime in 16-thread computing. We firstly compare Alg. 2 (randQB_EI), Alg. 4 and Alg. 5 (farPCA) to validate the proposed techniques. Then, we compare farPCA with randQB_EI, randUBV and `svds` to validate the overall efficiency of farPCA.

Four real-world matrices are considered for testing. One is a scenic image [Yu *et al.*, 2018] (named Image), represented by a $9,504 \times 4,752$ matrix. The other three are sparse matrices: an $82,168 \times 82,168$ social network matrix from SNAP [Leskovec and Krevl, 2014] with 948,464 nonzero elements, a $138,493 \times 26,744$ matrix from Movielens dataset [Harper and Konstan, 2016] named Movielens-20m with 20,000,263 nonzero elements, and a $270,896 \times 45,115$ matrix from Movielens dataset named Movielens with 26,024,289 nonzero elements, which is larger than Movielens-20m. The average number of nonzeros per row ranges from 12 to 144 for the three matrices.

4.1 Validation of the Proposed Techniques

Under spectral (or Frobenius) norm, the computational result ($\hat{\mathbf{U}}$, $\hat{\mathbf{\Sigma}}$ and $\hat{\mathbf{V}}$) of the randomized algorithms has the following multiplicative guarantee [Musco and Musco, 2015]:

$$\|\mathbf{A} - \hat{\mathbf{U}}\hat{\mathbf{\Sigma}}\hat{\mathbf{V}}^{\mathrm{T}}\| \le (1+\epsilon)\|\mathbf{A}-\mathbf{A}_k\|, \tag{18}$$

[2]https://software.intel.com/content/www/us/en/develop/tools/oneapi/components/onemkl.html

[3]https://github.com/erhallma/randUBV

[4]https://github.com/WenjianYu/randQB_auto

with high probability. Based on (18), we use

$$\epsilon_{\mathrm{F}} = \frac{\|\mathbf{A} - \hat{\mathbf{U}}\hat{\mathbf{\Sigma}}\hat{\mathbf{V}}^{\mathrm{T}}\|_F - \|\mathbf{A}-\mathbf{A}_k\|_F}{\|\mathbf{A}-\mathbf{A}_k\|_F}, \quad \text{and} \tag{19}$$

$$\epsilon_{\mathrm{s}} = \frac{\|\mathbf{A} - \hat{\mathbf{U}}\hat{\mathbf{\Sigma}}\hat{\mathbf{V}}^{\mathrm{T}}\|_2 - \|\mathbf{A}-\mathbf{A}_k\|_2}{\|\mathbf{A}-\mathbf{A}_k\|_2}, \tag{20}$$

as first two error metrics to evaluate the accuracy of randomized PCA algorithms in Frobenius norm and spectral norm.

Another guarantee proposed in [Musco and Musco, 2015], which is stronger and more meaningful in practice, is:

$$\forall i \le k, \;\; |\mathbf{u}_i^{\mathrm{T}}\mathbf{A}\mathbf{A}^{\mathrm{T}}\mathbf{u}_i - \hat{\mathbf{u}}_i^{\mathrm{T}}\mathbf{A}\mathbf{A}^{\mathrm{T}}\hat{\mathbf{u}}_i| \le \epsilon\sigma_{k+1}(\mathbf{A})^2, \tag{21}$$

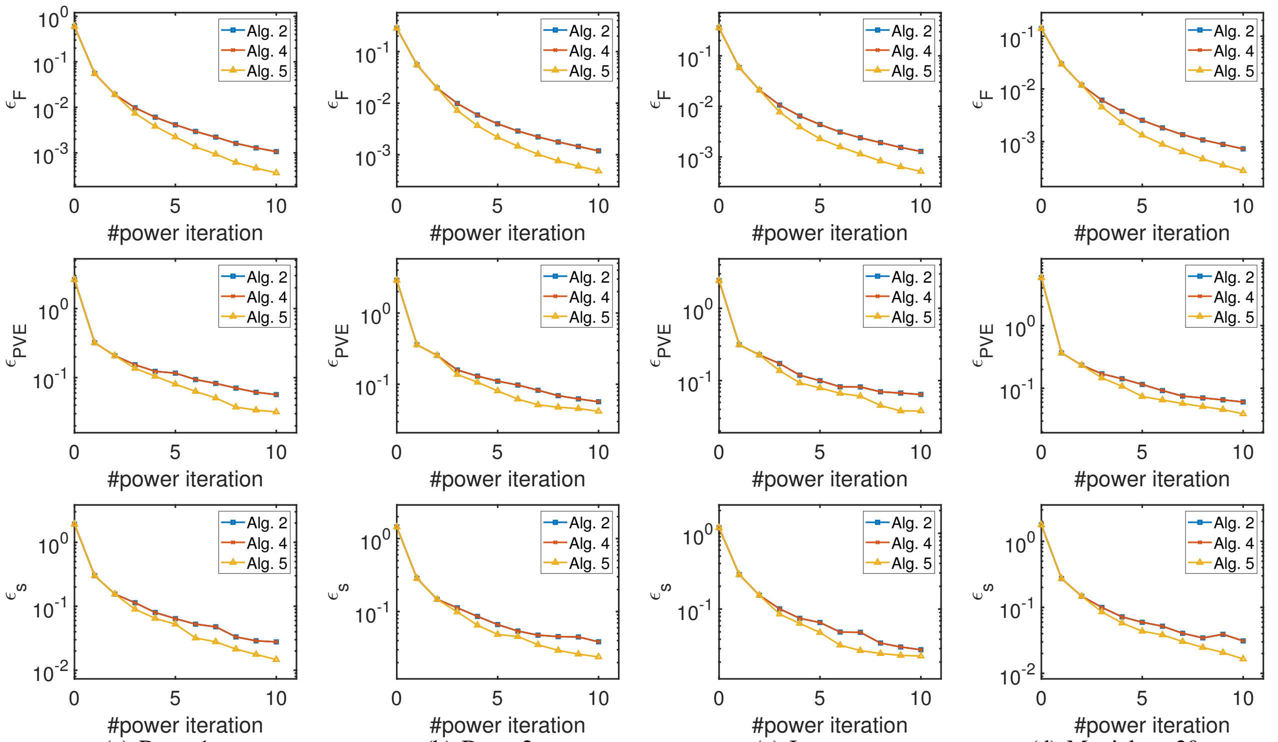
where $\mathbf{u}_i$ is the i-th left singular vector of $\mathbf{A}$, and $\hat{\mathbf{u}}_i$ is the computed i-th left singular vector. This is called *per vector error* bound for singular vectors. In [Musco and Musco, 2015], it is demonstrated that the per vector guarantee (21) requires each computed singular vector to capture nearly as much variance as the corresponding accurate singular vector, which is better to evaluate the accuracy of computed singular vectors compared with (18). Based on (21), we also use

$$\epsilon_{\mathrm{PVE}} = \max_{i\le k} \frac{|\mathbf{u}_i^{\mathrm{T}}\mathbf{A}\mathbf{A}^{\mathrm{T}}\mathbf{u}_i - \hat{\mathbf{u}}_i^{\mathrm{T}}\mathbf{A}\mathbf{A}^{\mathrm{T}}\hat{\mathbf{u}}_i|}{\sigma_{k+1}(\mathbf{A})^2} \tag{22}$$

to evaluate the accuracy. Notice that the metric (19), (20) and (22) were also used in [Allen-Zhu and Li, 2016] with name "Fnorm", "spectral" and "rayleigh(last)".

In order to validate the presented techniques, we vary power parameter p while keeping $b = 20$ and $k = 200$, and compare the randQB_EI (Alg. 2), the algorithm accelerated by matrix skills (Alg. 4), the proposed farPCA with shifted power iteration (Alg. 5) on two synthetic dense matrices, Image and Movielens-20m. The synthetic matrices are of 1,000 $\times$ 1,000 (denoted by Dense1 and Dense2), and randomly generated with the i-th singular value following $\sigma_i = 1/i$ and $\sigma_i = 1/\sqrt{i}$, respectively. This means the singular values of Dense2 decay slower than those of Dense1. With the accurate results obtained from `svds`, the corresponding error metrics (19), (20) and (22) are calculated to plotted in Fig. 1 with varied number of power iterations.

From Fig. 1 we see that, using the shift technique in Section 3.3 (Alg. 5) consistently results in more accurate results when the number of power iteration is larger than 2. Alg. 4 with more efficient matrix computation produces the same results as the original Alg. 2, which validates the correctness of proposed techniques in Section 3.1 and 3.2. On Dense1, Alg. 5 with $p = 5$ produces the results with similar accuracy of Alg. 2/Alg. 4 with $p = 8$, which means about 38% times of power iteration are reduced. And, the reduction of error ϵ_F of farPCA increases with the number of power iteration, it ranges from 2.5X to 2.9X if the number of power iteration is 10. Although not depicted in Fig. 1, we want to mention that Alg. 4 runs faster than Alg. 2 with the speed-up ratios up to 1.3X and 1.2X for case Image and Movielens-20m, respectively. This validates the efficiency of the techniques proposed in Section 3.1 and 3.2.



(a) Dense1 (b) Dense2 (c) Image (d) Movielens-20m

Figure 1: Error curves of the randomized algorithms with varied p value for case Dense1, Dense2, Image and Movielens-20m (k=200, b=20).

Case	randUBV			randQB_EI ($p=1$)			farPCA ($p=1$)				randQB_EI ($p=5$)			farPCA ($p=5$)			
	Time	k	r	Time	k	r	Time	k	r	SP	Time	k	r	Time	k	r	SP
Image	1.47	611	451	1.68	470	467	1.29	470	467	**1.3**	3.56	470	427	2.85	470	427	**1.3**
SNAP	339	7389	5655	337	6568	5482	229	6568	5482	**1.5**	662	5747	5086	377	5747	5078	**1.8**
Movielens-20m	112	1335	1230	162	1068	999	140	1068	999	**1.2**	469	1068	875	415	1068	873	**1.1**
Movielens	243	1804	1062	308	1353	1062	265	1353	1062	**1.2**	878	1353	973	761	1353	972	**1.2**

Table 1: Comparison of randQB_EI, randUBV and farPCA in Matlab ($b = \min(m, n)/100$, $\varepsilon = 0.1\|\mathbf{A}\|_F$ for Image and $\varepsilon = 0.5\|\mathbf{A}\|_F$ for the other cases). The unit of runtime is second, and SP is the speed-up ratio of farPCA to randQB_EI with same power parameter p.

Case	randQB_EI ($p=1$)			farPCA ($p=1$)				randQB_EI ($p=5$)			farPCA ($p=5$)					`svds`		
	Time	k	r	Time	k	r	SP	Time	k	r	Time	k	r	SP	SP-s	Time	k	r
Image	1.21	470	467	1.00	470	467	**1.2**	3.03	470	427	2.29	470	427	**1.3**	**17**	39.9	427	426
SNAP	221	6568	5482	158	6568	5481	**1.4**	419	5747	5086	227	5747	5078	**1.9**	**96**	21760	5078	5073
Movielens-20m	30.6	1068	999	23.8	1068	999	**1.3**	87.2	1068	875	60.4	1068	873	**1.4**	**17**	1026	873	872
Movielens	79.2	1353	1062	51.5	1353	1062	**1.5**	200	1353	973	127	1353	972	**1.6**	**16**	2063	972	972

Table 2: Comparison of randQB_EI and farPCA in C with MKL ($b = \min(m, n)/100$, $\varepsilon = 0.1\|\mathbf{A}\|_F$ for Image and $\varepsilon = 0.5\|\mathbf{A}\|_F$ for the other cases). The unit of runtime is second. SP and SP-s are the speed-up ratios of farPCA to randQB_EI and to `svds`, respectively.

4.2 Validation of farPCA

We compare the finally developed farPCA (Alg. 5) with randUBV [Hallman, 2022], randQB_EI [Yu *et al.*, 2018] and `svds` in Matlab with the fixed-precision low-rank factorization task on real-world test cases. The error tolerance is set $\varepsilon = 0.1\|\mathbf{A}\|_F$ for the case Image, and $\varepsilon = 0.5\|\mathbf{A}\|_F$ for the rest cases. The block parameter b is set to $\min(m, n)/100$. We not only record the runtime, but also list k and r in tables, where k is the rank at which the computing process is terminated and r is the final output rank after the post-processing. The smaller k and r represent smaller space cost and better quality of result, respectively.

The comparison results of the algorithms' Matlab versions are listed in Table 1. From it we see that, farPCA produces the same or smaller rank r than randQB_EI. The smaller r is due to the shifted power iteration used in farPCA. As for the runtime, with the proposed techniques farPCA is faster than randQB_EI with up to 1.8X speedup. The peak memory usages of farPCA are 0.50 GB, 15.9 GB, 3.18 GB and 7.54 GB for Image, SNAP, Movielens-20m and Movielens, respectively, which are similar to those of randQB_EI (0.50 GB, 17.4 GB, 3.09 GB and 7.44 GB). Although randUBV costs similar or less runtime than that of farPCA ($p = 1$), the larger k reflects that randUBV often results larger r and much larger k (implying much larger memory cost). The peak memory usages of randUBV for the four cases are 0.50 GB, 16.2 GB, 3.60 GB and 8.79 GB, showing prominent increase for large sparse matrices. Notice that farPCA can produce better results with larger p, while randUBV lacks the flexibility to reach better approximation.

As our aim is to develop a PCA tool for practical usage, we should also compare the algorithms implemented in C. The results are listed in Table 2. The results of `svds` are also listed for a reference, which also runs in multi-thread computing. As we know, there is not a C implementation of `svds`, possibly because implementing it in C cannot remarkably improves its performance due to its inside algorithm. Since `svds` is not able to solve the fixed-precision factorization problem, we use the r computed by farPCA with $p = 5$ as the input rank (k) for `svds`, and record its runtime and further obtain the optimal rank r. From Table 2, we see that the algorithms implemented in C with MKL cost less runtime than those implemented in Matlab (Table 1), especially for large cases. This speed-up is as large as 6.9X for Movielens-20m with $p = 5$, and much larger than that of randQB_EI algorithm (4.4X for the same case). This reflects that farPCA is more friendly for actual multi-thread computing. While comparing farPCA and randQB_EI, larger speed-up ratios are observed (up to 1.9X). Moreover, the peak memory usages of farPCA are also similar to those of randQB_EI (0.45 GB, 14.1 GB, 2.81 GB and 6.52 GB for the four cases), which are smaller than those of Matlab programs due to better memory management in the C implementation.

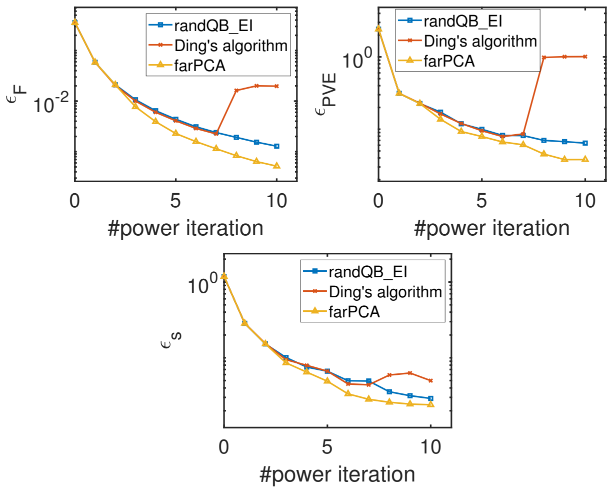


Figure 2: Error curves of randQB_EI, the algorithm in [Ding2020] and farPCA with varied p value for case Image (k=200, b=20).

With the results of `svds` we see that, the speed-up ratio of farPCA ($p = 5$) in Matlab to `svds` ranges from 2.5X to 58X, while that of farPCA ($p = 5$) in C ranges from 16X to 96X. This shows the practical efficiency of proposed farPCA. For result quality, the r of farPCA with $p = 5$ is nearly the same as that of `svds`, which implies farPCA produces the nearly optimal low-rank approximation. The peak memory usages of farPCA with $p = 5$ are 0.45 GB, 12.2 GB, 2.81 GB and 6.52 GB for the cases, which are much smaller than 0.78 GB, 48.7 GB, 7.93 GB and 16.4 GB consumed by `svds`. All these suggest that the proposed farPCA is able to produce near-optimal results with much less computational cost than `svds`, while owning the ability of performing adaptive PCA. An additional experiment with varied number of threads in Appendix shows that farPCA is more friendly for multi-thread computing.

5 Conclusion

A faster randomized PCA algorithm named farPCA is proposed for adaptively determining the number of dimensions (rank) of PCA for a given error tolerance. It includes optimized matrix operations for multi-thread parallel computing, efficient incremental error indicator, and shifted power iteration for better approximation accuracy or quality of result. Experimental results show that farPCA runs up to 1.9X faster than randQB_EI, and produces nearly the same optimal results while reducing the runtime by 16X through 96X when compared with `svds`.

A Appendix

A.1 The Flaw in Ding's Algorithm 3

The Step 14 in Ding's Algorithm 3 [Ding *et al.*, 2020], i.e. $[\mathbf{Q}_l, \sim] \leftarrow \mathrm{lu}(\mathbf{A}(\mathbf{A}^{\mathrm{T}}\mathbf{Q}_l))$ approximates the dominant subspace of $range(\mathbf{A})$ instead of that of $range(\mathbf{A} - \mathbf{QB})$ in randQB_EI, which leads to inaccurate results when p increases. We plot the error curves of randQB_EI, Ding's algorithm and farPCA on Image in Fig. 2 which shows that Ding's algorithm produces results with large error when $p > 7$.

A.2 Experiments with Different Number of Threads

We compare the randQB_EI and farPCA implemented in C with multi-thread computing and different number of threads. The results are listed in Table 3, which show the speed-up ratio increases with the number of threads.

Case	Method	$n_t = 2$	$n_t = 4$	$n_t = 8$	$n_t = 16$
Movielens	Time of randQB_EI	862	526	268	200
	Time of farPCA	671	342	171	127
	SP	1.28	1.54	1.56	**1.57**
SNAP	Time of randQB_EI	1766	1091	614	419
	Time of farPCA	1343	633	335	227
	SP	1.31	1.72	1.83	**1.85**

Table 3: The runtimes and speed-up ratios of randQB_EI and farPCA in C with MKL ($p = 5$). The unit of runtime is second. n_t is the number of threads. SP is the speed-up ratio of farPCA to randQB_EI.

Acknowledgments

This work is supported by NSFC under grant No. 61872206.

References

[Ailon and Chazelle, 2006] Nir Ailon and Bernard Chazelle. Approximate nearest neighbors and the fast johnson-lindenstrauss transform. In *Proceedings of the thirty-eighth annual ACM symposium on Theory of computing*, pages 557–563, 2006.

[Allen-Zhu and Li, 2016] Zeyuan Allen-Zhu and Yuanzhi Li. LazySVD: Even faster SVD decomposition yet without agonizing pain. In *Advances in Neural Information Processing Systems*, pages 974–982, 2016.

[Benjamin Erichson *et al.*, 2017] N. Benjamin Erichson, Steven L. Brunton, and J. Nathan Kutz. Compressed singular value decomposition for image and video processing. In *Proc. IEEE International Conference on Computer Vision (ICCV)*, pages 1880–1888, Oct. 2017.

[Bjarkason, 2019] Elvar K Bjarkason. Pass-efficient randomized algorithms for low-rank matrix approximation using any number of views. *SIAM Journal on Scientific Computing*, 41(4):A2355–A2383, 2019.

[Demmel *et al.*, 2012] James Demmel, Laura Grigori, Mark Hoemmen, and Julien Langou. Communication-optimal parallel and sequential qr and lu factorizations. *SIAM Journal on Scientific Computing*, 34(1):A206–A239, 2012.

[Ding *et al.*, 2020] Xiangyun Ding, Wenjian Yu, Yuyang Xie, and Shenghua Liu. Efficient model-based collaborative filtering with fast adaptive PCA. In *2020 IEEE 32nd International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 955–960. IEEE, 2020.

[Drineas and Mahoney, 2016] Petros Drineas and Michael W Mahoney. Randnla: Randomized numerical linear algebra. *Communications of the ACM*, 59(6):80–90, 2016.

[Eckart and Young, 1936] Carl Eckart and Gale Young. The approximation of one matrix by another of lower rank. *Psychometrika*, 1(3):211–218, 1936.

[Feng *et al.*, 2018a] Xu Feng, Yuyang Xie, Mingye Song, Wenjian Yu, and Jie Tang. Fast randomized PCA for sparse data. In *Proc. the 10th Asian Conference on Machine Learning (ACML)*, pages 710–725, 14–16 Nov 2018.

[Feng *et al.*, 2018b] Xu Feng, Wenjian Yu, and Yaohang Li. Faster matrix completion using randomized svd. In *Proc. the 30th IEEE International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 608–615, 2018.

[Feng *et al.*, 2022] Xu Feng, Wenjian Yu, and Yuyang Xie. Pass-efficient randomized SVD with boosted accuracy. In *Proc. the European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML-PKDD)*, 19–23 Sep. 2022.

[Golub *et al.*, 1981] Gene H Golub, Franklin T Luk, and Michael L Overton. A block lanczos method for computing the singular values and corresponding singular vectors of a matrix. *ACM Transactions on Mathematical Software (TOMS)*, 7(2):149–169, 1981.

[Halko *et al.*, 2011] N. Halko, P. G. Martinsson, and J. A. Tropp. Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions. *SIAM Review*, 53(2):217–288, 2011.

[Hallman, 2022] Eric Hallman. A block bidiagonalization method for fixed-accuracy low-rank matrix approximation. *SIAM Journal on Matrix Analysis and Applications*, 43(2):661–680, 2022.

[Harper and Konstan, 2016] F. Maxwell Harper and Joseph A. Konstan. The Movielens datasets: History and context. *ACM Transactions on Interactive Intelligent Systems (TiiS)*, 5(4):19, 2016.

[Leskovec and Krevl, 2014] Jure Leskovec and Andrej Krevl. SNAP datasets: Stanford large network dataset collection. http://snap.stanford.edu/data, June 2014.

[Li *et al.*, 2017] 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 Transactions on Mathematical Software*, 43(3):1–14, 2017.

[Martinsson and Tropp, 2020] Per-Gunnar Martinsson and Joel A. Tropp. Randomized numerical linear algebra: Foundations and algorithms. *Acta Numerica*, 29:403–572, 2020.

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

[Musco and Musco, 2015] Cameron Musco and Christopher Musco. Randomized block Krylov methods for stronger and faster approximate singular value decomposition. In *Advances in Neural Information Processing Systems*, pages 1396–1404, 2015.

[Qiu *et al.*, 2021] Jiezhong Qiu, Laxman Dhulipala, Jie Tang, Richard Peng, and Chi Wang. Lightne: A lightweight graph processing system for network embedding. In *Proc. the 2021 ACM SIGMOD international conference on Management of data*, pages 2281–2289, 2021.

[Voronin and Martinsson, 2015] Sergey Voronin and Per-Gunnar Martinsson. RSVDPACK: An implementation of randomized algorithms for computing the singular value, interpolative, and CUR decompositions of matrices on multi-core and GPU architectures. *arXiv preprint arXiv:1502.05366*, 2015.

[Woolfe *et al.*, 2008] Franco Woolfe, Edo Liberty, Vladimir Rokhlin, and Mark Tygert. A fast randomized algorithm for the approximation of matrices. *Applied and Computational Harmonic Analysis*, 25(3):335–366, 2008.

[Yu *et al.*, 2017] Wenjian Yu, Yu Gu, Jian Li, Shenghua Liu, and Yaohang Li. Single-pass PCA of large high-dimensional data. In *Proc. International Joint Conference on Artificial Intelligence (IJCAI)*, pages 3350–3356, Aug. 2017.

[Yu *et al.*, 2018] Wenjian Yu, Yu Gu, and Yaohang Li. Efficient randomized algorithms for the fixed-precision low-rank matrix approximation. *SIAM Journal on Matrix Analysis and Applications*, 39(3):1339–1359, 2018.

[Zhang *et al.*, 2019a] Jiabao Zhang, Shenghua Liu, Wenjian Yu, Wenjie Feng, and Xueqi Cheng. Eigenpulse: detecting surges in large streaming graphs with row augmentation. In *Proc. the 23rd Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 501–513, 2019.

[Zhang *et al.*, 2019b] Jie Zhang, Yuxiao Dong, Yan Wang, Jie Tang, and Ming Ding. Prone: Fast and scalable network representation learning. In *Proc. the 28th International Joint Conference on Artificial Intelligence*, volume 19, pages 4278–4284, 2019.