



# Algorithm 1043: Faster Randomized SVD with Dynamic Shifts

XU FENG, WENJIAN YU, YUYANG XIE, and JIE TANG, Department of Computer Science and Technology, BNRist, Tsinghua University, Beijing, China

Aiming to provide a faster and convenient truncated SVD algorithm for large sparse matrices from real applications (i.e., for computing a few of the largest singular values and the corresponding singular vectors), a dynamically shifted power iteration technique is applied to improve the accuracy of the randomized SVD method. This results in a *dynamic shifts*-based randomized SVD (dashSVD) algorithm, which also collaborates with the skills for handling sparse matrices. An accuracy-control mechanism is included in the dashSVD algorithm to approximately monitor the per vector error bound of computed singular vectors with negligible overhead. Experiments on real-world data validate that the dashSVD algorithm largely improves the accuracy of a randomized SVD algorithm or attains the same accuracy with fewer passes over the matrix, and provides an efficient accuracy-control mechanism to the randomized SVD computation, while demonstrating the advantages on runtime and parallel efficiency. A bound of the approximation error of the randomized SVD with the shifted power iteration is also proved.

CCS Concepts: • **Computing methodologies** → **Parallel algorithms**; • **Mathematics of computing** → **Solvers**; • **Theory of computation** → **Numeric approximation algorithms**;

Additional Key Words and Phrases: Truncated singular value decomposition, random embedding, shifted power iteration, sparse matrix

## ACM Reference format:

Xu Feng, Wenjian Yu, Yuyang Xie, and Jie Tang. 2024. Algorithm 1043: Faster Randomized SVD with Dynamic Shifts. *ACM Trans. Math. Softw.* 50, 2, Article 14 (June 2024), 27 pages.  
<https://doi.org/10.1145/3660629>

## 1 Introduction

In machine learning and data mining, truncated **singular value decomposition (SVD)**, i.e., computing a few of largest singular values and corresponding singular vectors, is widely used in dimension reduction, information retrieval, matrix completion, and so forth [14, 16, 29]. The truncated SVD computes optimal low-rank approximation and **principal component analysis (PCA)**. Specifically, the top singular vector  $\mathbf{u}_1$  of matrix  $\mathbf{A}$  provides the top principal component, which reflects the direction of the largest variance within  $\mathbf{A}$ . The  $i$ th left singular vector  $\mathbf{u}_i$  provides

This work was supported by the National Natural Science Foundation of China under Grant 61872206.

Authors' Contact Information: Xu Feng, Department of Computer Science and Technology, BNRist, Tsinghua University, Beijing, China; e-mail: fx17@mails.tsinghua.edu.cn; Wenjian Yu (Corresponding author), Department of Computer Science and Technology, BNRist, Tsinghua University, Beijing, China; e-mail: yu-wj@tsinghua.edu.cn; Yuyang Xie, Department of Computer Science and Technology, BNRist, Tsinghua University, Beijing, China; e-mail: xyy18@mails.tsinghua.edu.cn; Jie Tang, Department of Computer Science and Technology, BNRist, Tsinghua University, Beijing, China; e-mail: jietang@tsinghua.edu.cn.



This work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivatives International 4.0 License.

© 2024 Copyright held by the owner/author(s).

ACM 1557-7295/2024/6-ART14

<https://doi.org/10.1145/3660629>

the  $i$ th principal component, which reflects the direction of the largest variance orthogonal to all higher principal components.

However, for large and high-dimensional input data from social network analysis, natural language processing and recommender system, computing truncated SVD of the corresponding matrix often consumes tremendous time and memory usage. For large data matrices which are usually very sparse, the preferred method of computing truncated SVD is using svds in MATLAB [8]. svds is still the most robust and efficient in general cases compared with the variant algorithms of svds like lansvd [23]. To tackle the difficulty of computing truncated SVD for large sparse data, various algorithms have been proposed [6, 16, 18, 25, 29, 33, 34]. They consume less computational resource but usually induce a little accuracy loss. Among these algorithms, the randomized method based on random embedding through multiplying a random matrix [27] gains a lot of attention. The randomized method produces a near-optimal truncated SVD of the matrix, while exhibiting performance advantages over the classical methods (like less runtime, fewer passes over the matrix and better parallelizability, etc.). Therefore, the randomized method is very favorable for the scenarios where only low-accuracy SVD is required, like in the field of machine learning. More analysis on relevant techniques and theories can be found in [18, 27].

In this work, we aim to provide an algorithm for computing the truncated SVD of large sparse matrices efficiently on a multi-core computer. Inspired by the shift technique in the power method [17], we develop a **dynamic shifts-based randomized SVD (dashSVD)** algorithm to improve the computational efficiency of the randomized SVD. An efficient accuracy-control scheme is also developed and integrated into the dashSVD algorithm. Our major contributions and results are as follows.

- A dynamic scheme for setting the shift values in the shifted power iteration is developed to accelerate the randomized SVD algorithm. It improves the accuracy of the result or reduces the number of power iterations for attaining same accuracy. Combining the scheme with the accelerating skills for handling sparse matrices, we have further developed a dashSVD algorithm.
- Based on the **per vector error (PVE)** criterion [29], an efficient accuracy-control mechanism is developed and integrated into the dashSVD algorithm. It resolves the difficulty of setting a suitable power parameter  $p$  and enables automatic termination of the power iteration according to the PVE bound-based accuracy criterion.
- Experiments on real-world data have validated the efficiency of these techniques, demonstrating that dashSVD runs faster than the state-of-the-art algorithms for attaining a not very high accuracy (e.g., with PVE error  $\epsilon_{\text{PVE}} \geq 10^{-2}$ ), with comparable memory usage. On dataset uk-2005, dashSVD runs  $3.2\times$  faster than the LanczosBD algorithm in svds for attaining the accuracy corresponding to PVE error  $\epsilon_{\text{PVE}} = 10^{-1}$  with serial computing and runs  $4.0\times$  faster than the PRIMME-SVDS algorithm [34] with parallel computing employing eight threads. The experiments also reveal that dashSVD is more robust than the existing fast SVD algorithms [23, 34].

It should be noted that, in this work we just consider the efficient implementation of the randomized SVD on a shared-memory machine. Its efficient implementation in a distributed-memory parallelism context is also of great interest [30]. The codes of dashSVD are shared on GitHub (<https://github.com/THU-numbda/dashSVD>).

The rest of this article is structured as follows. Section 2 introduces the basics of the randomized SVD via random embedding and the state-of-the-art truncated SVD algorithms. In Section 3, we present the dashSVD algorithm with the dynamic scheme for setting the shift values in the shifted power iteration and the accuracy-control mechanism based on the PVE criterion. Numerical experiments for performance analysis are presented in Section 4. Finally, we draw the conclusions. More theoretical proofs and experimental results are given in Appendices A and B.

## 1.1 Related Work

A basic randomized SVD algorithm was presented in [18], where a power iteration technique was employed to improve accuracy while sacrificing computational time. A couple of works were later proposed to accelerate it. The amount of orthonormalization in the power iteration is reduced and the **eigenvalue decomposition (EVD)** is used to compute economic SVD faster but sacrificing little numerical stability [33]. Li et al. proposed to replace the QR factorization for orthonormalization in power iteration with LU factorization [25], to reduce the computational time. Erichson et al. employed a variant of the randomized SVD algorithm with several acceleration tricks in image and video processing problems [10]. Later, an algorithm called frPCA [16] was proposed which collaborates with the skills in [10, 25, 33] and is specialized to accelerate the truncated (top- $k$ ) SVD computation of large sparse matrices.

As an alternative of power iteration, the technique of block Krylov iteration also improves the accuracy of the randomized SVD. It consumes less CPU time than the algorithm with power iteration to attain the same accuracy [29] but consumes much larger memory. SLEPc in [21] is an efficient library to compute both SVD and generalized SVD in both distributed and shared-memory environment. PRIMME\_SVDS in [34] is based on eigenvalue package PRIMME [32] and could efficiently compute both the largest and smallest singular values, which is more efficient than SLEPc for computing truncated SVD. LazySVD in [6] is based on the Lanczos process. It lacks accuracy control and only computes the left singular vectors.

## 2 Preliminary

We use the conventions from MATLAB in this article to specify indices of matrices and functions.

### 2.1 Basics of Truncated SVD

The economic SVD of a matrix  $\mathbf{A} \in \mathbb{R}^{m \times n}$  ( $m \geq n$ ) is

$$\mathbf{A} = \mathbf{U}\Sigma\mathbf{V}^T, \quad (1)$$

where  $\mathbf{U} = [\mathbf{u}_1, \mathbf{u}_2, \dots, \mathbf{u}_n]$  and  $\mathbf{V} = [\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_n]$  are matrices containing the left and right singular vectors of  $\mathbf{A}$ , respectively.  $\Sigma$  is an  $n \times n$  diagonal matrix containing the singular values  $(\sigma_1, \sigma_2, \dots, \sigma_n)$  of  $\mathbf{A}$  in descending order.  $\{\mathbf{u}_i, \mathbf{v}_i, \sigma_i\}$  is called the  $i$ th singular triplet, and  $\sigma_i(\cdot)$  also denotes the  $i$ th largest singular value. From Equation (1), one can obtain the truncated SVD of  $\mathbf{A}$

$$\mathbf{A}_k = \mathbf{U}_k \Sigma_k \mathbf{V}_k^T, \quad k < \min(m, n), \quad (2)$$

where  $\mathbf{U}_k$  and  $\mathbf{V}_k$  are matrices with the first  $k$  columns of  $\mathbf{U}$  and  $\mathbf{V}$ , respectively, and the diagonal matrix  $\Sigma_k$  is the  $k \times k$  upper-left submatrix of  $\Sigma$ . Notice that  $\mathbf{A}_k$  is the best rank- $k$  approximation of  $\mathbf{A}$  in both spectral norm and Frobenius norm [15]. svds in MATLAB is based on the Lanczos bidiagonalization process and an augmented restarting scheme which reduces the memory cost and ensures accuracy [8]. svds has been widely regarded as the standard tool for computing the truncated SVD.

### 2.2 Randomized SVD Algorithm with Power Iteration

The basic randomized SVD algorithm [18] can be described as Algorithm 1, where  $\Omega$  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 Steps 3 through 5 is for improving the accuracy of the result, where the orthonormalization alleviating the round-off error in floating-point computation is performed. Here we employ the skill of performing the orthonormalization after every other matrix-matrix multiplication to save computational cost with little accuracy loss [16, 33], as compared to doing orthonormalization after every matrix-matrix multiplication [18].

**Algorithm 1** Basic Randomized SVD 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:  $l = k + s$ ,  $\Omega = \text{randn}(n, l)$
- 2:  $\mathbf{Q} = \text{orth}(\mathbf{A}\Omega)$
- 3: **for**  $j = 1, 2, \dots, p$  **do**
- 4:    $\mathbf{Q} = \text{orth}(\mathbf{A}\mathbf{A}^T\mathbf{Q})$
- 5: **end for**
- 6:  $\mathbf{B} = \mathbf{Q}^T\mathbf{A}$
- 7:  $[\mathbf{U}, \mathbf{S}, \mathbf{V}] = \text{svd}(\mathbf{B}, \text{'econ'})$
- 8:  $\mathbf{U} = \mathbf{Q}\mathbf{U}(:, 1:k)$ ,  $\mathbf{S} = \mathbf{S}(1:k, 1:k)$ ,  $\mathbf{V} = \mathbf{V}(:, 1:k)$

If there is no power iteration, the  $m \times l$  orthonormal matrix  $\mathbf{Q} = \text{orth}(\mathbf{A}\Omega)$  is an approximation of the basis of dominant subspace of  $\text{range}(\mathbf{A})$ , i.e.,  $\text{span}\{\mathbf{u}_1, \mathbf{u}_2, \dots, \mathbf{u}_l\}$ . Therefore,  $\mathbf{A} \approx \mathbf{Q}\mathbf{Q}^T\mathbf{A} = \mathbf{Q}\mathbf{B}$  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  $\mathbf{Q} = \text{orth}((\mathbf{A}\mathbf{A}^T)^p\mathbf{A}\Omega)$ , if the intermediate orthonormalization steps are ignored. This makes  $\mathbf{Q}$  better approximate the basis of dominant subspace of  $\text{range}((\mathbf{A}\mathbf{A}^T)^p\mathbf{A})$ , same as that of  $\text{range}(\mathbf{A})$ , because  $(\mathbf{A}\mathbf{A}^T)^p\mathbf{A}$ 's singular values decay more quickly than those of  $\mathbf{A}$  [18]. So, the computed singular triplets are more accurate, and the larger  $p$  leads to more accurate results and more computational cost as well. According to the following Lemma, i.e., (3.3.17) and (3.3.18) in [22], we can derive Proposition 1 which shows the relationship between the singular values of  $\mathbf{A}$  and the singular values computed by Algorithm 1.

**LEMMA 1.** Suppose  $\mathbf{A}, \mathbf{C} \in \mathbb{R}^{m \times n}$ . The following inequalities hold for the decreasingly ordered singular values of  $\mathbf{A}$ ,  $\mathbf{C}$ , and  $\mathbf{A}\mathbf{C}^T$  ( $1 \leq i, j, i+j-1 \leq \min(m, n)$ ):

$$\sigma_{i+j-1}(\mathbf{A}\mathbf{C}^T) \leq \sigma_i(\mathbf{A})\sigma_j(\mathbf{C}), \quad (3)$$

and

$$\sigma_{i+j-1}(\mathbf{A} + \mathbf{C}) \leq \sigma_i(\mathbf{A}) + \sigma_j(\mathbf{C}). \quad (4)$$

**PROPOSITION 1.** Suppose  $\mathbf{A} \in \mathbb{R}^{m \times n}$  and  $\mathbf{Q} \in \mathbb{R}^{n \times l}$  ( $l \leq \min(m, n)$ ) is an orthonormal matrix. Then,

$$\sigma_i(\mathbf{A}\mathbf{Q}) \leq \sigma_i(\mathbf{A}), \text{ for any } i \leq l. \quad (5)$$

**PROOF.** We append zero columns to  $\mathbf{Q}$  to get an  $n \times m$  matrix  $\mathbf{C}^T = [\mathbf{Q}, \mathbf{0}] \in \mathbb{R}^{n \times m}$ . Since  $\mathbf{Q}$  is an orthonormal matrix,  $\sigma_1(\mathbf{C}) = 1$ . According to Equation (3) in Lemma 1,

$$\sigma_i(\mathbf{A}\mathbf{C}^T) \leq \sigma_i(\mathbf{A})\sigma_1(\mathbf{C}) = \sigma_i(\mathbf{A}). \quad (6)$$

Because  $\mathbf{A}\mathbf{C}^T = [\mathbf{A}\mathbf{Q}, \mathbf{0}]$ , for any  $i \leq l$ ,  $\sigma_i(\mathbf{A}\mathbf{Q}) = \sigma_i(\mathbf{A}\mathbf{C}^T)$ . Then, combining Equation (6) we can prove Equation (5).  $\square$

**Remark 1.** Because the singular values of  $\mathbf{B}$  in Step 6 of Algorithm 1 are equal to those of  $\mathbf{B}^T = \mathbf{A}^T\mathbf{Q}$ , Proposition 1 infers that the singular value computed with Algorithm 1 is equal to or less than its accurate value, i.e.,  $\sigma_i(\mathbf{B}) \leq \sigma_i(\mathbf{A})$ . This explains that the singular value curve computed with the randomized SVD algorithm is always underneath the accurate curve of singular value, as shown in literature.

For a sparse matrix  $\mathbf{A}$ , the power iteration costs a larger portion of the total time as it involves the manipulation of dense matrices. So, accelerating the computations in power iteration becomes important. Besides reducing the amount of orthonormalization, the skills of using LU factorization to replace QR factorization and EVD based SVD to do orthonormalization have been employed. Along with the techniques for efficiently handling the matrix with more columns than rows and allowing an odd number of passes over  $\mathbf{A}$ , an algorithm called frPCA was developed in [16] for faster truncated SVD of sparse matrices. However, for most real-world sparse matrices whose singular values decay slowly, a large number of power iteration steps are required to attain satisfied accuracy, and there is a lack of efficient mechanism to determine the power parameter  $p$ .

Below we use the **floating-point operation (flop)** count to analyze the computational cost. Suppose  $C_{\text{mul}}$ ,  $C_{\text{qr}}$ , and  $C_{\text{svd}}$  represent the constants in the flop counts of matrix–matrix multiplication, QR factorization and economic SVD, respectively.  $\text{nnz}(\mathbf{A})$  is the number of nonzero elements in  $\mathbf{A}$ . According to the procedure, the flop count of Algorithm 1 is

$$\text{FC}_1 = (2p + 2)C_{\text{mul}}\text{nnz}(\mathbf{A})l + (p + 1)C_{\text{qr}}ml^2 + C_{\text{svd}}nl^2 + C_{\text{mul}}mlk, \quad (7)$$

where  $(2p + 2)C_{\text{mul}}\text{nnz}(\mathbf{A})l$  reflects the matrix–matrix multiplication on  $\mathbf{A}$ ,  $(p + 1)C_{\text{qr}}ml^2$  reflects the QR factorization in Steps 2 and 4,  $C_{\text{svd}}nl^2$  reflects the economic SVD in Step 7 and  $C_{\text{mul}}mlk$  reflects the matrix–matrix multiplication in Step 8.

### 2.3 Algorithms in svds and PRIMME\_SVDS

svds is the most robust and well-known tool to compute truncated SVD based on the Lanczos bidiagonalization process with augmented restarting scheme in [8]. In svds, the relative residual of singular vectors

$$\max_{i \leq k} \frac{\|\mathbf{A}^T \hat{\mathbf{u}}_i - \hat{\sigma}_i \hat{\mathbf{v}}_i\|_2}{\hat{\sigma}_i} \quad (8)$$

is calculated to control the accuracy, where  $\{\hat{\mathbf{u}}_i, \hat{\sigma}_i, \hat{\mathbf{v}}_i\}$  is the computed  $i$ th singular triplet. The Lanczos bidiagonalization process with augmented restarting scheme is called LanczosBD in MATLAB. svds first runs LanczosBD until the first  $k$  singular triplets make Equation (8) smaller than a preset tolerance. Then, one or more extra invocations of LanczosBD for computing the first  $k + 1$  singular triples are executed to ensure the robustness of svds. Therefore, svds can produce accurate truncated SVD in almost all scenarios.

Recently, a high-performance parallel SVD solver called PRIMME\_SVDS [34] was developed. For a matrix  $\mathbf{A}$ , PRIMME\_SVDS first uses the state-of-the-art truncated EVD library PRIMME [32] to compute EVD of  $\mathbf{A}^T \mathbf{A}$ . Then, the obtained results are used as input vectors to solve the truncated EVD of  $\begin{bmatrix} \mathbf{0} & \mathbf{A}^T \\ \mathbf{A} & \mathbf{0} \end{bmatrix}$  with PRIMME, to ensure the accuracy of computed singular vectors. The relative residuals of eigenvectors and eigenvalues are also used in PRIMME for accuracy control, which is similar to Equation (8) in svds.

## 3 Randomized SVD with Dynamic Shifts

In this section, we first introduce how to improve the accuracy of the randomized SVD with the shifted power iteration and a dynamic scheme of setting the shift value. Then, we discuss the termination criteria for accuracy control and devise an efficient algorithm based on the PVE criterion.

### 3.1 Shifted Power Iteration and Setting Dynamic Shifts

The computation  $\mathbf{Q} = \mathbf{A}\mathbf{A}^T \mathbf{Q}$  in the power iteration of Algorithm 1 is the same as that in the power method for computing the largest eigenvalue and corresponding eigenvector of  $\mathbf{A}\mathbf{A}^T$ . The shift skill can be used to accelerate the convergence of the power method because of the reduction of the ratio between the second largest eigenvalue and the largest one [17]. This inspires the idea of using

shifts in the power iteration of a randomized SVD algorithm. We first give two Lemmas [17] to derive the power iteration with the shift value.

LEMMA 2. *For a symmetric real-valued matrix  $\mathbf{A}$ , its singular values are the absolute values of its eigenvalues. For any eigenvalue  $\lambda$  of  $\mathbf{A}$ , the left singular vector corresponding to singular value  $|\lambda|$  is the normalized eigenvector for  $\lambda$ .*

LEMMA 3. *Suppose matrix  $\mathbf{A} \in \mathbb{R}^{n \times n}$ , and a shift  $\alpha \in \mathbb{R}$ . For any eigenvalue  $\lambda$  of  $\mathbf{A}$ ,  $\lambda - \alpha$  is an eigenvalue of  $\mathbf{A} - \alpha\mathbf{I}$ , where  $\mathbf{I}$  is the identity matrix. And the eigenspace of  $\mathbf{A}$  for  $\lambda$  is the same as the eigenspace of  $\mathbf{A} - \alpha\mathbf{I}$  for  $\lambda - \alpha$ .*

For any matrix  $\mathbf{A}$ ,  $\mathbf{A}\mathbf{A}^T$  is a symmetric positive semi-definite matrix, so its singular value is its eigenvalue according to Lemma 2. Lemma 3 shows that  $\sigma_i(\mathbf{A}\mathbf{A}^T) - \alpha$  is the eigenvalue of  $\mathbf{A}\mathbf{A}^T - \alpha\mathbf{I}$ . And  $|\sigma_i(\mathbf{A}\mathbf{A}^T) - \alpha|$  is the singular value of  $\mathbf{A}\mathbf{A}^T - \alpha\mathbf{I}$  according to Lemma 2. In Algorithm 1, the decay trend of the largest  $l$  singular values of handled matrix affects the accuracy of the resulted SVD. When  $\sigma_i(\mathbf{A}\mathbf{A}^T) - \alpha > 0$  for any  $i \leq l$ , and they are the  $l$  largest singular values of  $\mathbf{A}\mathbf{A}^T - \alpha\mathbf{I}$ , they obviously exhibit faster decay. The following Proposition states when these conditions are satisfied.

PROPOSITION 2. *Suppose  $0 < \alpha \leq \sigma_l(\mathbf{A}\mathbf{A}^T)/2$  and  $i \leq l$ . Then,  $\sigma_i(\mathbf{A}\mathbf{A}^T - \alpha\mathbf{I}) = \sigma_i(\mathbf{A}\mathbf{A}^T) - \alpha$ . Moreover, if  $\sigma_i(\mathbf{A}\mathbf{A}^T - \alpha\mathbf{I}) \neq \sigma_{l+1}(\mathbf{A}\mathbf{A}^T - \alpha\mathbf{I})$ , the left singular vector corresponding to the  $i$ th largest singular value of  $\mathbf{A}\mathbf{A}^T - \alpha\mathbf{I}$  is the left singular vector corresponding to the  $i$ th largest singular value of  $\mathbf{A}\mathbf{A}^T$ , and vice versa.*

PROOF. When  $0 < \alpha \leq \sigma_l(\mathbf{A}\mathbf{A}^T)/2$ , we can derive  $2\alpha \leq \sigma_l(\mathbf{A}\mathbf{A}^T)$  and  $0 < \alpha \leq \sigma_l(\mathbf{A}\mathbf{A}^T) - \alpha$ . Therefore,  $\alpha - \sigma_i(\mathbf{A}\mathbf{A}^T) \leq \alpha \leq \sigma_l(\mathbf{A}\mathbf{A}^T) - \alpha$ . For any  $i > l$

$$\sigma_i(\mathbf{A}\mathbf{A}^T) - \alpha \leq \sigma_l(\mathbf{A}\mathbf{A}^T) - \alpha. \quad (9)$$

Then, we can derive

$$|\sigma_i(\mathbf{A}\mathbf{A}^T) - \alpha| \leq \sigma_l(\mathbf{A}\mathbf{A}^T) - \alpha, \text{ for any } i > l. \quad (10)$$

Besides, for any  $i \leq l$  we can derive

$$|\sigma_i(\mathbf{A}\mathbf{A}^T) - \alpha| = \sigma_i(\mathbf{A}\mathbf{A}^T) - \alpha \geq \sigma_l(\mathbf{A}\mathbf{A}^T) - \alpha > 0, \text{ for any } i \leq l. \quad (11)$$

Notice that  $\sigma_i(\mathbf{A}\mathbf{A}^T)$  is also an eigenvalue of  $\mathbf{A}\mathbf{A}^T$ , and  $|\sigma_i(\mathbf{A}\mathbf{A}^T) - \alpha|$  is a singular value of  $\mathbf{A}\mathbf{A}^T - \alpha\mathbf{I}$  according to Lemmas 2 and 3. So, combining Equations (10) and (11) we can see that  $|\sigma_i(\mathbf{A}\mathbf{A}^T) - \alpha|$ ,  $1 \leq i \leq l$ , are the  $l$  largest singular values of  $\mathbf{A}\mathbf{A}^T - \alpha\mathbf{I}$ , and

$$\sigma_i(\mathbf{A}\mathbf{A}^T - \alpha\mathbf{I}) = \sigma_i(\mathbf{A}\mathbf{A}^T) - \alpha, \text{ for any } i \leq l. \quad (12)$$

Lemma 2 shows that, for any  $i \leq l$  the left singular vector corresponding to  $\sigma_i(\mathbf{A}\mathbf{A}^T)$  is the same as the normalized eigenvector for eigenvalue  $\sigma_i(\mathbf{A}\mathbf{A}^T)$  of  $\mathbf{A}\mathbf{A}^T$ , and the left singular vector corresponding to  $\sigma_i(\mathbf{A}\mathbf{A}^T) - \alpha$  is the same as the normalized eigenvector for eigenvalue  $\sigma_i(\mathbf{A}\mathbf{A}^T) - \alpha$  of  $\mathbf{A}\mathbf{A}^T - \alpha\mathbf{I}$ . Notice that  $\sigma_i(\mathbf{A}\mathbf{A}^T - \alpha\mathbf{I}) \neq \sigma_{l+1}(\mathbf{A}\mathbf{A}^T - \alpha\mathbf{I})$  ensures that  $\sigma_i(\mathbf{A}\mathbf{A}^T - \alpha\mathbf{I}) = \sigma_i(\mathbf{A}\mathbf{A}^T) - \alpha$  is a value only existing among the largest  $l$  singular values  $\sigma_1(\mathbf{A}\mathbf{A}^T - \alpha\mathbf{I}), \dots, \sigma_l(\mathbf{A}\mathbf{A}^T - \alpha\mathbf{I})$ . According to Lemma 3, the eigenspace of  $\mathbf{A}\mathbf{A}^T$  for  $\sigma_i(\mathbf{A}\mathbf{A}^T)$  is the same as the eigenspace of  $\mathbf{A}\mathbf{A}^T - \alpha\mathbf{I}$  for  $\sigma_i(\mathbf{A}\mathbf{A}^T) - \alpha$  for any  $i \leq l$ . Therefore, combining Equation (12) yields that for any  $i \leq l$ , the left singular vector corresponding to the  $i$ th largest singular value  $\sigma_i(\mathbf{A}\mathbf{A}^T)$  of  $\mathbf{A}\mathbf{A}^T$  is the left singular vector corresponding to the  $i$ th largest singular value  $\sigma_i(\mathbf{A}\mathbf{A}^T) - \alpha$  of  $\mathbf{A}\mathbf{A}^T - \alpha\mathbf{I}$ , and vice versa.  $\square$

Proposition 2 shows that, if we choose a shift  $0 < \alpha \leq \sigma_l(\mathbf{A}\mathbf{A}^T)/2$ , we can change the computation  $\mathbf{Q} = \mathbf{A}\mathbf{A}^T\mathbf{Q}$  to  $\mathbf{Q} = (\mathbf{A}\mathbf{A}^T - \alpha\mathbf{I})\mathbf{Q}$  in the power iteration, with the same approximated dominant subspace. We called this *shifted power iteration*. For each step of the shifted power iteration, this makes  $\mathbf{Q}$  approximate the basis of dominant subspace of  $\text{range}(\mathbf{A})$  to a larger extent



**Algorithm 2** Randomized SVD with Dynamic Shifts**Input:**  $A \in \mathbb{R}^{m \times n}$ , parameters  $k, s, p$ **Output:**  $U \in \mathbb{R}^{m \times k}$ ,  $S \in \mathbb{R}^{k \times k}$ ,  $V \in \mathbb{R}^{n \times k}$ 

- 1:  $l = k + s$ ,  $\Omega = \text{randn}(n, l)$
- 2:  $Q = \text{orth}(A\Omega)$ ,  $\alpha = 0$
- 3: **for**  $j = 1, 2, \dots, p$  **do**
- 4:    $[Q, \hat{S}, \sim] = \text{svd}(AA^T Q - \alpha Q, \text{'econ'})$
- 5:   **if**  $\hat{S}(l, l) > \alpha$  **then**  $\alpha = \frac{\hat{S}(l, l) + \alpha}{2}$
- 6: **end for**
- 7:  $B = Q^T A$
- 8:  $[U, S, V] = \text{svd}(B, \text{'econ'})$
- 9:  $U = QU(:, 1:k)$ ,  $S = S(1:k, 1:k)$ ,  $V = V(:, 1:k)$

than executing an original power iteration step, because the singular values of  $AA^T - \alpha I$  decay faster than those of  $AA^T$ . Therefore, the shifted power iteration can improve the accuracy of the randomized SVD algorithm with same power parameter  $p$ . Then, how to set the shift  $\alpha$  properly is the remaining problem.

Consider the change of ratio of singular values from  $\frac{\sigma_i(AA^T)}{\sigma_j(AA^T)}$  to  $\frac{\sigma_i(AA^T - \alpha I)}{\sigma_j(AA^T - \alpha I)}$ , for  $j < i \leq l$  and  $\sigma_j(AA^T) > \sigma_i(AA^T)$ . It is easy to see  $\frac{\sigma_i(AA^T - \alpha I)}{\sigma_j(AA^T - \alpha I)} < \frac{\sigma_i(AA^T)}{\sigma_j(AA^T)}$  if the assumption of  $\alpha$  in Proposition 2 holds. The larger value of  $\alpha$ , the smaller the ratio  $\frac{\sigma_i(AA^T - \alpha I)}{\sigma_j(AA^T - \alpha I)}$ , reflecting faster decay of singular values. Therefore, to maximize the effect of the shifted power iteration on improving the accuracy, we should choose the shift  $\alpha$  as large as possible while satisfying  $\alpha \leq \sigma_l(AA^T)/2$ . Notice that calculating  $\sigma_l(AA^T)$  directly is very difficult. Our idea is to use the singular value of  $AA^T Q$  in the power iteration to approximate  $\sigma_l(AA^T)$  and set the shift  $\alpha$ . Suppose  $Q \in \mathbb{R}^{m \times l}$  is the orthonormal matrix in the power iteration of Algorithm 1. According to Proposition 1,

$$\sigma_i(AA^T Q) \leq \sigma_i(AA^T), \quad \text{for any } i \leq l, \quad (13)$$

which means that we can set  $\alpha = \sigma_l(AA^T Q)/2$  to guarantee the requirement of  $\alpha$  in Proposition 2 for performing the shifted power iteration. To do the orthonormalization for alleviating round-off error and calculate  $\sigma_l(AA^T Q)$ , we implement “orth(·)” with the economic SVD. This has similar computational cost as using QR factorization, and the resulted matrix of left singular vectors includes the orthonormal basis of same subspace.

So far, we can compute the value of  $\alpha$  at the first step of the power iteration and then we perform  $Q = (AA^T - \alpha I)Q$  in the following iteration steps. Notice computing the singular values of  $(AA^T - \alpha I)Q$  is convenient. Then, according to Propositions 1 and 2, we can derive with  $0 < \alpha \leq \sigma_l(AA^T)/2$

$$\sigma_i((AA^T - \alpha I)Q) + \alpha \leq \sigma_i(AA^T - \alpha I) + \alpha = \sigma_i(AA^T), \quad \text{for any } i \leq l, \quad (14)$$

which states how to use the singular values of  $(AA^T - \alpha I)Q$  to approximate  $\sigma_i(AA^T)$ . Therefore, in each iteration step we can obtain a valid value of shift and update  $\alpha$  with it if we have a larger  $\alpha$ , for faster decay of singular values.

Combining the shifted power iteration with the dynamic updating scheme of  $\alpha$ , we derive a randomized SVD algorithm based on the dynamically shifted power iteration as Algorithm 2. In Step 5, it checks if  $\frac{\sigma_l((AA^T - \alpha I)Q) + \alpha}{2}$  is larger than  $\alpha$ . For the same setting of power parameter  $p$ , Algorithm 2 has a similar computational time to Algorithm 1 (with just QR for a tall-and-skinny

**Algorithm 3** eigSVD**Input:** a dense matrix  $C \in \mathbb{R}^{m \times n}$  ( $m \geq n$ )**Output:**  $U \in \mathbb{R}^{m \times n}$ ,  $S \in \mathbb{R}^{n \times n}$ ,  $V \in \mathbb{R}^{n \times n}$ 

- 1:  $[V, D] = \text{eig}(C^T C)$
- 2:  $S = \text{sqrt}(D)$
- 3:  $U = CV S^{-1}$
- 4:  $U = \text{fliplr}(U)$ ,  $V = \text{fliplr}(V)$ ,  $S = \text{rot90}(S, 2)$

matrix replaced with SVD) but produces results with much better accuracy. For matrix  $Q$  computed with Algorithm 2, we derive a bound of  $\|QQ^T A - A\|$ , which reflects how close the computed truncated SVD is to optimal.

**THEOREM 1.** Suppose  $A \in \mathbb{R}^{m \times n}$  ( $m \geq n$ ), and  $k, s$ , and  $p$  are the parameters in Algorithm 2.  $l = k + s \leq n - k$ , and  $Q$  in size  $m \times l$  is the obtained orthonormal matrix with Algorithm 2.

If positive integer  $j < k$ , and real numbers  $\beta, \gamma > 1$  satisfying  $\phi = \frac{1}{\sqrt{2\pi(l-j+1)}} \left( \frac{e}{(l-j+1)\beta} \right)^{l-j+1} + \frac{1}{4\gamma(\gamma^2-1)} \left( \frac{1}{\sqrt{\pi(n-k)}} \left( \frac{2\gamma^2}{e\gamma^2-1} \right)^{n-k} + \frac{1}{\sqrt{\pi l}} \left( \frac{2\gamma^2}{e\gamma^2-1} \right)^l \right) \leq 1$ , then

$$\|QQ^T A - A\| \leq 2 \sqrt{\left( 2l^2 \beta^2 \gamma^2 \prod_{c=1}^p \left( \frac{\sigma_{j+1}^2 - \alpha_c}{\sigma_j^2 - \alpha_c} \right)^2 + 1 \right) \sigma_{j+1}^2 + \left( 2l(n-k) \beta^2 \gamma^2 \prod_{c=1}^p \left( \frac{\sigma_{k+1}^2 - \alpha_c}{\sigma_j^2 - \alpha_c} \right)^2 + 1 \right) \sigma_{k+1}^2}, \quad (15)$$

with probability not less than  $1 - \phi$ , where  $\alpha_c$  denotes the shift value in the  $c$ th shifted power iteration in Algorithm 2 and  $\sigma_j$  denotes the  $j$ th largest singular value of  $A$ .

**Remark 2.**  $\phi$  can be a small value, and the bound Equation (15) is smaller than that derived in [31] for the  $Q$  computed with the basic randomized SVD algorithm (see Appendix A).

Algorithm 2 can collaborate with more skills to achieve further acceleration while handling sparse matrices. Firstly, we can use EVD to compute the economic SVD or the orthonormal basis of a tall-and-skinny matrix in Steps 2, 4, and 8. This resorts to the eigSVD algorithm in [16], which is described in Algorithm 3. The algorithm is similar to that in [16] but outputs singular values in the correct order. “eig(·)” computes the EVD. The “rot90( $S$ , 2)” in Step 4 is the function to rotate  $S$  180°, and “fliplr(·)” in Step 4 is the function to flip array or matrix left to right. Therefore, the singular values are in descending order, while the singular vectors corresponding to them are at correct positions. Secondly, for  $A \in \mathbb{R}^{m \times n}$  with  $m > n$ , it is more efficient to compute SVD of  $A^T$  with Algorithm 2. Therefore, we can derive two efficient versions of the randomized SVD algorithm, for the cases with  $m \geq n$  and  $n \geq m$ , respectively. The version for the matrices with  $m \geq n$  is described in Algorithm 4, and all the previous theoretic analyses can be applied, if we replace  $A$  with  $A^T$ .

Suppose  $p$  power iterations are executed to attain a certain accuracy, and  $C_{\text{eig}}$  represents the constant in the flop count of EVD. The flop count of Algorithm 4 is

$$FC_4 = (2p+2)C_{\text{mul}}\text{nnz}(A)l + (p+1)(2C_{\text{mul}}nl^2 + C_{\text{eig}}l^3) + 2C_{\text{mul}}ml^2 + C_{\text{eig}}l^3 + pC_{\text{mul}}nl + C_{\text{mul}}nlk, \quad (16)$$

where  $(2p+2)C_{\text{mul}}\text{nnz}(A)l$  reflects the  $2p+2$  times matrix–matrix multiplications on  $A$ ,  $(p+1)(2C_{\text{mul}}nl^2 + C_{\text{eig}}l^3)$  reflects executing eigSVD on an  $n \times l$  matrix for  $p+1$  times,  $2C_{\text{mul}}ml^2 + C_{\text{eig}}l^3$  reflects once eigSVD on matrix  $B$ ,  $pC_{\text{mul}}nl$  reflects executing the operation “ $-\alpha Q$ ” for  $p$  times, and  $C_{\text{mul}}nlk$  reflects the matrix–matrix multiplication to generate  $V$ . From this, we see that the time complexity of Algorithm 4 is  $O(p\text{lnnz}(A) + pnl^2)$  for  $m \times n$  matrix  $A$  with  $m \geq n$ . The space



**Algorithm 4** Dynamic Shifts Based Randomized SVD for Matrices with  $m \geq n$ **Input:**  $\mathbf{A} \in \mathbb{R}^{m \times n}$ , parameters  $k, s, p$ **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:  $l = k + s$ ,  $\Omega = \text{randn}(m, l)$
- 2:  $[\mathbf{Q}, \sim, \sim] = \text{eigSVD}(\mathbf{A}^T \Omega)$ ,  $\alpha = 0$
- 3: **for**  $j = 1, 2, \dots, p$  **do**
- 4:    $[\mathbf{Q}, \hat{\mathbf{S}}, \sim] = \text{eigSVD}(\mathbf{A}^T (\mathbf{A}\mathbf{Q}) - \alpha \mathbf{Q})$
- 5:   **if**  $\hat{\mathbf{S}}(l, l) > \alpha$  **then**  $\alpha = \frac{\hat{\mathbf{S}}(l, l) + \alpha}{2}$
- 6: **end for**
- 7:  $\mathbf{B} = \mathbf{A}\mathbf{Q}$
- 8:  $[\mathbf{U}, \mathbf{S}, \mathbf{V}] = \text{eigSVD}(\mathbf{B})$
- 9:  $\mathbf{U} = \mathbf{U}(:, 1:k)$ ,  $\mathbf{S} = \mathbf{S}(1:k, 1:k)$ ,  $\mathbf{V} = \mathbf{Q}\mathbf{V}(:, 1:k)$

complexity of Algorithm 4 is about  $O((2m + n)l + \text{nnz}(\mathbf{A}))$ , as the peak memory usage occurs at Step 8. Because  $C_{\text{qr}}$  and  $C_{\text{svd}}$  are multiple times larger than  $C_{\text{mul}}$ , we can clearly see that  $\text{FC}_4 < \text{FC}_1$  when  $m \geq n$  and  $n \gg l$ , which reflects the efficiency of Algorithm 4.

**3.2 Accuracy Control Based on PVE Bound**

Theoretical research has revealed that the randomized SVD with power iteration produces the rank- $k$  approximation close to optimal. Under spectral norm (or Frobenius norm) the computational results ( $\hat{\mathbf{U}}$ ,  $\hat{\mathbf{S}}$ , and  $\hat{\mathbf{V}}$ ) satisfy the following multiplicative guarantee with high probability

$$\|\mathbf{A} - \hat{\mathbf{U}}\hat{\mathbf{S}}\hat{\mathbf{V}}^T\| \leq (1 + \epsilon)\|\mathbf{A} - \mathbf{A}_k\|, \quad (17)$$

where  $\epsilon \in (0, 1)$  is a distortion parameter. Another guarantee proposed in [29], which is more meaningful in machine learning problems, is

$$\forall i \leq k, \quad |\mathbf{u}_i^T \mathbf{A} \mathbf{A}^T \mathbf{u}_i - \hat{\mathbf{u}}_i^T \mathbf{A} \mathbf{A}^T \hat{\mathbf{u}}_i| \leq \epsilon \sigma_{k+1}(\mathbf{A})^2, \quad (18)$$

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 PVE bound for singular vectors. In [29], it is demonstrated that the  $(1 + \epsilon)$  error bound in Equation (17) may not guarantee any accuracy in the computed singular vectors. In contrary, the per vector guarantee Equation (18) requires each computed singular vector to capture nearly as much variance as the corresponding accurate singular vector, thus guaranteeing the accuracy of the principal components in PCA. Notice that the smaller  $\epsilon$  in Equations (17) and (18) is, the more power iterations in the randomized SVD algorithms are needed to meet the corresponding accuracy guarantee.

In existing work on randomized SVD, the power parameter  $p$  (or pass parameter) is assumed known in prior. For attaining certain accuracy, multiple examinations may be performed to find a suitable and not too large  $p$ . This brings a lot of computation cost. Therefore, how to adaptively and efficiently determine how many power iterations should be executed to ensure certain accuracy of the computed SVD is of concern.

There are different criteria to evaluate the accuracy of SVD, like the relative residual in Equation (8) and that derived from the PVE bound. Below we find out that if collaborating with the PVE criterion we can develop an efficient randomized SVD algorithm with accuracy control. The PVE bound in Equation (18) can be approximately expressed as

$$\forall i \leq k, \quad \frac{|\hat{\sigma}_i^{(j)}(\mathbf{A})^2 - \hat{\sigma}_i^{(j-1)}(\mathbf{A})^2|}{\hat{\sigma}_{k+1}^{(j)}(\mathbf{A})^2} \leq \text{tol}, \quad (19)$$

where  $\hat{\sigma}_i(\mathbf{A})$  denotes the computed  $i$ th singular value and superscript  $(j)$  denotes the computed value with  $p = j$ . “tol” stands for an error tolerance. This formula (19) is an approximate PVE criterion for accuracy control and “tol” is supposed to be specified by user. Setting “tol” suitably often guarantees the PVE bound Equation (18) with  $\epsilon$  comparable to “tol,” as shown in our experiments. In contrast,  $p$  gives no insight on the error and varies largely for different cases. The approximate criterion in Equation (19) is based on estimating the error of  $\hat{\sigma}_i^{(j-1)}(\mathbf{A})^2$  by regarding  $\hat{\sigma}_i^{(j)}(\mathbf{A})^2$  as the accurate result. This skill is a traditional method to control the accuracy in computing truncated EVD [17] and has been used in other numerical methods with iterative steps [20]. Notice that constructing an accuracy criterion based on Equations (8) or (17) leads to difficulties in implementation, as it involves expensive computations for large matrices.

The above idea of the PVE criterion derives a randomized SVD algorithm with accuracy control. A straightforward implementation moves the generation and SVD of matrix  $\mathbf{B} = \mathbf{Q}^T \mathbf{A}$  into the power iteration and then imposes the check of the termination criterion. Below we develop a more efficient way to realize the accuracy control of the PVE criterion.

**PROPOSITION 3.** *Suppose matrix  $\mathbf{A} \in \mathbb{R}^{m \times n}$ , orthonormal matrix  $\mathbf{Q} \in \mathbb{R}^{m \times l}$  and positive value  $\alpha$  are the quantities before executing Step 4 in Algorithm 2. Then, for any  $i \leq l$ ,*

$$\sigma_i(\mathbf{Q}^T \mathbf{A}) \leq \sqrt{\sigma_i(\mathbf{A} \mathbf{A}^T \mathbf{Q} - \alpha \mathbf{Q}) + \alpha} \leq \sigma_i(\mathbf{A}). \quad (20)$$

**PROOF.** For any  $i \leq l$ , according to Equation (3) in Lemma 1,

$$\sigma_i(\mathbf{Q}^T \mathbf{A})^2 = \sigma_i(\mathbf{Q}^T \mathbf{A} \mathbf{A}^T \mathbf{Q}) \leq \sigma_i(\mathbf{Q}^T \mathbf{A} \mathbf{A}^T) \sigma_1(\mathbf{Q}) = \sigma_i(\mathbf{A} \mathbf{A}^T \mathbf{Q}). \quad (21)$$

According to Equation (4) in Lemma 1, we can derive

$$\sigma_i(\mathbf{A} \mathbf{A}^T \mathbf{Q}) = \sigma_i(\mathbf{A} \mathbf{A}^T \mathbf{Q} - \alpha \mathbf{Q} + \alpha \mathbf{Q}) \leq \sigma_i(\mathbf{A} \mathbf{A}^T \mathbf{Q} - \alpha \mathbf{Q}) + \alpha, \quad (22)$$

and according to Equation (14) we derive

$$\sqrt{\sigma_i(\mathbf{A} \mathbf{A}^T \mathbf{Q} - \alpha \mathbf{Q}) + \alpha} \leq \sqrt{\sigma_i(\mathbf{A} \mathbf{A}^T)} = \sigma_i(\mathbf{A}). \quad (23)$$

Therefore, combining Equations (21)–(23) yields Proposition 3.  $\square$

According to Proposition 3,  $\sqrt{\sigma_i(\mathbf{A} \mathbf{A}^T \mathbf{Q} - \alpha \mathbf{Q}) + \alpha}$  is closer to  $\sigma_i(\mathbf{A})$  than  $\sigma_i(\mathbf{Q}^T \mathbf{A})$  which is the computed singular value with the power iteration with one fewer step. Therefore,  $\sigma_i(\mathbf{A} \mathbf{A}^T \mathbf{Q} - \alpha \mathbf{Q}) + \alpha$  can be regarded as a convenient surrogate of  $\hat{\sigma}_i(\mathbf{A})^2$ , as the former is readily available after Step 4 of Algorithm 2. Substituting it to Equation (19) we can derive the dashSVD algorithm, whose version for  $m \geq n$  is described as Algorithm 5. Here, Step 5 for accuracy control costs negligible time,  $p_{max}$  is a parameter denoting the upper bound of the number of power iterations (i.e., parameter  $p$ ), and the symbols with “ $r$ ” are the quantities in last iteration step. Algorithm 4 is also a version of the dashSVD algorithm; it just does not include the accuracy control. The analysis of computational complexity at the end of the last subsection is also valid for the dashSVD algorithm (Algorithm 5).

#### 4 Performance Analysis

The developed algorithms are implemented in MATLAB, and also in C with **Intel Math Kernel Library (MKL)** [2] and OpenMP [7] directives for multi-thread parallel computing. Some of the codes follow the implementation in [3, 28]. Numerical experiments are conducted to compare them with existing truncated SVD algorithms, frPCA in [16] written in C with MKL [4], the LanczosBD in svds, and the PRIMME\_SVDS written in C with MKL which also invokes PETSc [9] and MPI for parallel computing [5]. We first validate the performance of the shifted power iteration. Then, we compare the dynamic shifts based the SVD algorithm (Algorithm 4) with

**Algorithm 5** The DashSVD with PVE Accuracy Control**Input:**  $\mathbf{A} \in \mathbb{R}^{m \times n}$  ( $m \geq n$ ), parameters  $k, s, p_{\max}$ , an error tolerance  $\text{tol}$ **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:  $l = k + s, \Omega = \text{randn}(m, l)$ 
2:  $[\mathbf{Q}, \sim, \sim] = \text{eigSVD}(\mathbf{A}^T \Omega), \alpha = \alpha' = 0, \hat{\mathbf{S}}' = \text{zeros}(l, 1)$ 
3: for  $j = 1, 2, 3, \dots, p_{\max}$  do
4:    $[\mathbf{Q}, \hat{\mathbf{S}}, \sim] = \text{eigSVD}(\mathbf{A}^T (\mathbf{A}\mathbf{Q}) - \alpha \mathbf{Q})$ 
5:   if  $\forall i \leq k, \frac{|\hat{\mathbf{S}}'(i, i) + \alpha' - \hat{\mathbf{S}}(i, i) - \alpha|}{\hat{\mathbf{S}}(k+1, k+1) + \alpha} \leq \text{tol}$  then break
6:   if  $\hat{\mathbf{S}}(l, l) > \alpha$  then  $\alpha = \frac{\hat{\mathbf{S}}(l, l) + \alpha}{2}$ 
7:    $\hat{\mathbf{S}}' = \hat{\mathbf{S}}, \alpha' = \alpha$ 
8: end for
9:  $\mathbf{B} = \mathbf{A}\mathbf{Q}$ 
10:  $[\mathbf{U}, \mathbf{S}, \mathbf{V}] = \text{eigSVD}(\mathbf{B})$ 
11:  $\mathbf{U} = \mathbf{U}(:, 1 : k), \mathbf{S} = \mathbf{S}(1 : k, 1 : k), \mathbf{V} = \mathbf{Q}\mathbf{V}(:, 1 : k)$ 

```

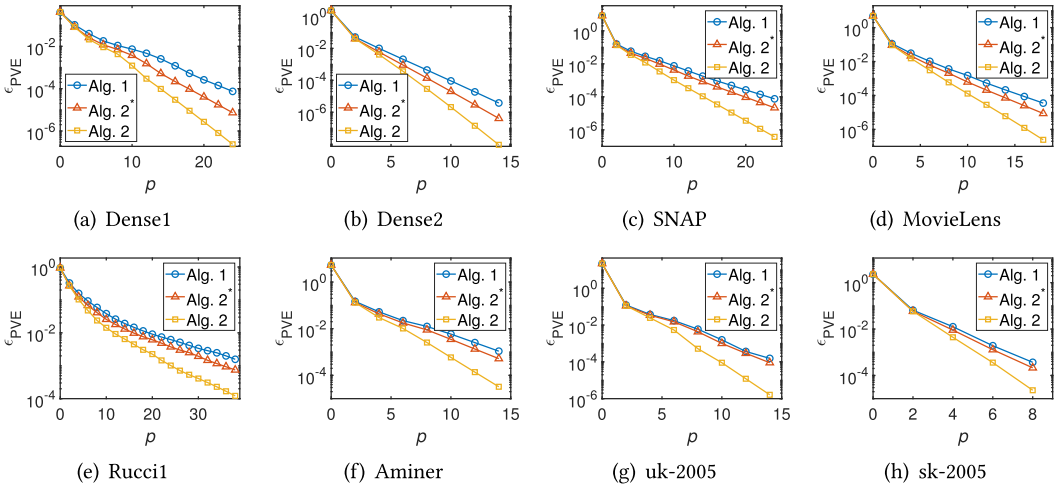
state-of-the-art algorithms. Finally, we validate the proposed accuracy-control mechanism in dashSVD. The accuracy and efficiency of these algorithms are evaluated. Four error metrics for evaluating accuracy are considered. Based on the PVE bound Equation (18), the first error is  $\epsilon_{\text{PVE}} = \max_{i \leq k} \frac{|\mathbf{u}_i^T \mathbf{A} \mathbf{A}^T \mathbf{u}_i - \hat{\mathbf{u}}_i^T \mathbf{A} \mathbf{A}^T \hat{\mathbf{u}}_i|}{\sigma_{k+1}(\mathbf{A})^2}$ . It is called “rayleigh(last)” metric in [6]. Based on relative residual error in Equation (8), we define  $\epsilon_{\text{res}} = \max_{i \leq k} \frac{\|\mathbf{A}^T \hat{\mathbf{u}}_i - \hat{\sigma}_i \hat{\mathbf{v}}_i\|_2}{\sigma_i(\mathbf{A})}$  which measures the accuracy of singular triplets. Notice that when the computed results are the singular triplets other than the leading singular triplets,  $\epsilon_{\text{res}}$  can also be very small. This implies  $\epsilon_{\text{res}}$  may not be a good criterion. svds avoids this issue of  $\epsilon_{\text{res}}$  by executing extra invocations of LanczosBD for  $k + 1$  singular triplets. The third metric is  $\epsilon_{\text{spec}} = \frac{\|\mathbf{A} - \hat{\mathbf{U}} \hat{\mathbf{S}} \hat{\mathbf{V}}^T\|_2 - \|\mathbf{A} - \mathbf{A}_k\|_2}{\|\mathbf{A} - \mathbf{A}_k\|_2}$ , which represents the spectral relative error based on Equation (17). Lastly, we use  $\epsilon_{\sigma} = \max_{i \leq k} \frac{|\sigma_i(\mathbf{A}) - \hat{\sigma}_i|}{\sigma_i(\mathbf{A})}$  to measure the accuracy of computed singular values. In these metrics, the accurate truncated SVD results are computed with svds in MATLAB 2020b. Notice there is a little run-to-run variability for randomized SVD algorithms, but that is not a serious issue [26]. In Appendix B, we show the experimental result demonstrating this variability increases with the number of power iterations in dashSVD (parameter  $p$ ). While for small  $p$  corresponding to computing low-accuracy SVD, the standard deviation of error metric is mostly no more than 5% of the corresponding mean value. Therefore, except for that experiment we present the results of dashSVD in an arbitrary run.

The following sparse matrices from real applications are tested, which are listed in Table 1. Three of them were used in [16]: a social network matrix from SNAP [24], a matrix from MovieLens dataset [19] and a person-keyword matrix from the information retrieval application “AMiner” [1]. Rucci1 is a sparse matrix with singular values decaying very slowly, obtained from SuiteSparse matrix collection [13]. The last two sparse matrices uk-2005 and sk-2005 are from larger web graphs [11, 12], also downloaded from SuiteSparse matrix collection. The average number of nonzeros per row ranges from 3.9 to 237.

In our experiments, the rank parameter  $k$  is set to 100 by default. For other values of  $k$ , the effectiveness of the proposed dashSVD is also validated. The relevant experimental results are shown in Appendix B, which even imply that with larger  $k$  the advantage of dashSVD would be slightly more prominent. The oversampling parameter  $s$  is fixed to  $k/2$ . For fair comparison with respect to memory cost, the subspace dimensionalities in LanczosBD and PRIMME\_SVDS are fixed

Table 1. Test Cases

Matrix	Dimension	No. of nonzero elements
SNAP (soc-Slashdot0922)	$82,168 \times 82,168$	948,464
MovieLens	$270,896 \times 45,115$	26,024,289
Rucci1	$1,977,885 \times 109,900$	7,791,168
Aminer	$12,869,521 \times 323,899$	206,501,139
uk-2005	$39,459,925 \times 39,459,925$	936,364,282
sk-2005	$50,639,401 \times 50,639,401$	1,949,412,601

Fig. 1. The PVE error vs. power parameter curves of three randomized SVD algorithms ( $k = 100$ ).

to  $1.5k$ . All experiments are carried out on a computer with two 8-core Intel Xeon E5-2620 v4 CPUs (at 2.10 GHz) and 512 GB RAM. We compile these algorithms with GCC version 9.4.0 and the “-O3” option. The MKL is of version 2019.4.0, while PETSc is of version 3.17.1.

#### 4.1 Validation of the Shifted Power Iteration

To validate the power iteration scheme with dynamic shifts, we set different power parameter  $p$  and perform the basic randomized SVD (Algorithm 1) and the proposed dashSVD (Algorithm 2) on test matrices. With the computational results, the corresponding error metric  $\epsilon_{PVE}$  is calculated to evaluate the accuracy. The curves of  $\epsilon_{PVE}$  are drawn in Figure 1, where “Alg. 2\*” denotes the algorithm with the fixed shift value  $\alpha = \sigma_l(\mathbf{A}\mathbf{A}^T\mathbf{Q})/2$  set at the first step of iteration. Two additional  $1,000 \times 1,000$  random matrices generated by randn in MATLAB (“Dense1”) and with the  $i$ th singular value following  $\sigma_i = 1/\sqrt{i}$  (“Dense2”) are also tested. From Figure 1 we see that, the dynamic scheme for setting the shift consistently produces more accurate results than the scheme with the fixed shift. And the proposed Algorithm 2 results in much better accuracy (and efficiency as well) than the basic randomized SVD with power iteration. The reduction of the result’s error increases with  $p$ , and is up to  $100\times$ .

The effectiveness of the shifted power iteration can also be validated by comparing dashSVD with frPCA [16]. The major difference between them is that dashSVD employs the shifted power iteration. The comparison results are given in Appendix B.

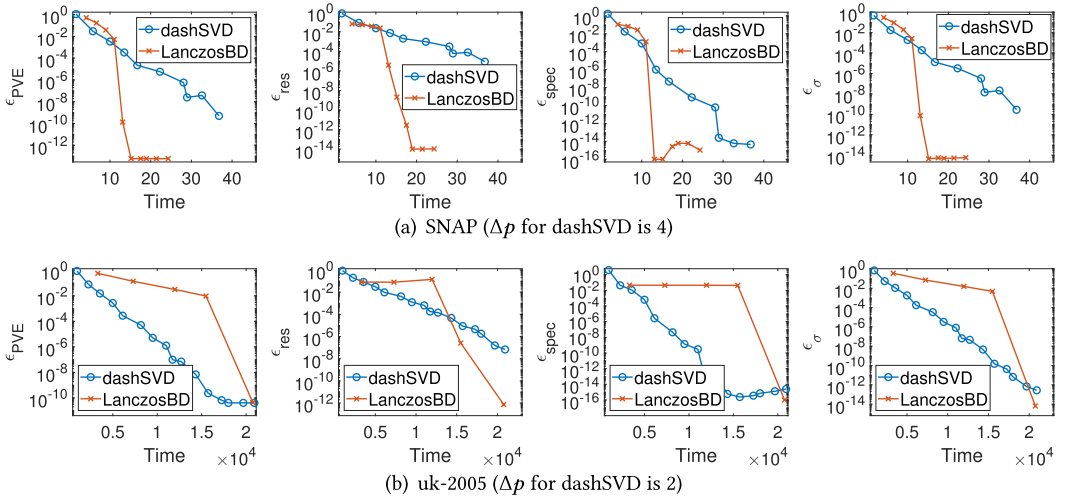


Fig. 2. The error vs. time curves of dashSVD and LanczosBD in svds with single-thread computing ( $k = 100$ ). Time is given in seconds.

## 4.2 Validation of the dashSVD Algorithm

For computing truncated SVD, most existing methods can tradeoff time against accuracy. In this subsection, we compare the efficiency of dashSVD with other methods at various accuracy levels. For this aim, we also implement the dashSVD without accuracy control, i.e., Algorithm 4, where  $p$  can be altered to tradeoff time against accuracy. Notice it has the same runtime as dashSVD (Algorithm 5) because the step for accuracy control costs negligible time.

Firstly, we compare dashSVD and svds both in MATLAB. For fair comparison, we run them in single-thread by executing command “`maxNumCompThreads(1)`” in MATLAB a prior, and the LanczosBD in svds is actually compared. We vary the  $p$  in dashSVD (Algorithm 4) and the number of restartings in LanczosBD to plot the curves of runtime vs. error. The results of SNAP and uk-2005 in the four error metrics are shown in Figure 2, with the results of other matrices in Appendix B. The results reveal that dashSVD always costs comparable or less time than LanczosBD for achieving a not very high accuracy, while for some cases (like uk-2005) LanczosBD needs a considerable number of restartings to converge to accurate result. For the metrics  $\epsilon_{PVE}$ ,  $\epsilon_{spec}$ , and  $\epsilon_{\sigma}$ , dashSVD has remarkable advantage on computing time (3.2× and 3.4× for attaining error  $10^{-1}$  in  $\epsilon_{PVE}$  and  $\epsilon_{\sigma}$  for uk-2005 respectively).

We compare the C version of dashSVD with PRIMME\_SVDS for eight-thread parallel computing (as parallel svds is not efficient). To avoid the **Non-Uniform Memory Access (NUMA)** complications, we stick with one NUMA domain for comparison in one socket with eight threads by setting the environment variables “`OMP_PROC_BIND=close`” and “`OMP_PLACES=cores.`” The error vs. runtime curves of SNAP and uk-2005 are plotted in Figure 3. The results show that the proposed dashSVD runs remarkably faster than PRIMME\_SVDS for producing the results with same accuracy. For example, to achieve the results with  $10^{-1}$  relative residual error ( $\epsilon_{res}$ ) dashSVD runs 3.2× and 1.3× faster than PRIMME\_SVDS for SNAP and uk-2005, respectively. With metric  $\epsilon_{PVE}$ , the corresponding speedup ratios become 4.0× and 3.9×. Comparing the results in Figures 2 and 3, we can also find out that the parallel speedup of dashSVD is about 5.5 for eight-thread computing. More error vs. runtime curves of MovieLens, Rucci1, Aminer, and sk-2005 are plotted in Appendix B. The memory costs of dashSVD, LanczosBD, and PRIMME\_SVDS are listed in Table 2. Table 2 shows

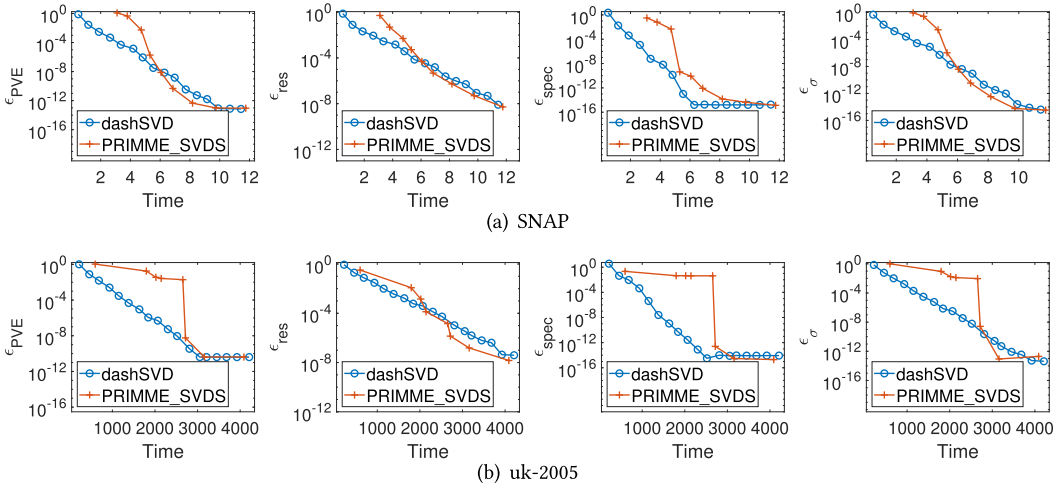


Fig. 3. The error vs. runtime curves of dashSVD and PRIMME\_SVDS with eight-thread computing ( $k = 100$ ). Time is given in seconds.

Table 2. Memory Cost (in GB) of the dashSVD, LanczosBD, and PRIMME\_SVDS

Matrix	dashSVD	LanczosBD	PRIMME_SVDS
SNAP	0.30	0.29	0.50
MovieLens	0.96	1.54	1.00
Rucci1	4.66	4.01	3.62
Aminer	31.5	27.7	23.1
uk-2005	147	135	133
sk-2005	200	184	179

that our dashSVD algorithm costs the same or comparable memory compared with LanczosBD and PRIMME\_SVDS.

To validate dashSVD for matrices with multiple singular values, a matrix with many multiple singular values called LargeRegFile obtained from SuiteSparse matrix collection [13] is tested in addition. This is a matrix in size  $2,111,154 \times 801,374$  with 2.3 nonzero elements per row on average. Firstly, we plot the error vs. runtime curves for LargeRegFile when  $k = 100$  in Figure 4, as those in Figures 2 and 3. Figure 4(a) shows that LanczosBD needs a long time to converge to accurate result for LargeRegFile, which corresponds to a considerable number of restarting, and our dashSVD has remarkable advantages on computing time for the four metrics. From Figure 4(b), although PRIMME\_SVDS reaches lower  $\epsilon_{\text{res}}$  within the same runtime compared with our dashSVD, the other error metrics are really large, which shows the singular triplets computed by PRIMME\_SVDS are wrong. The memory cost of dashSVD, LanczosBD, and PRIMME\_SVDS is 5.78 GB, 5.03 GB, and 5.09 GB, respectively. This also shows that our dashSVD algorithm costs comparable memory to LanczosBD and PRIMME\_SVDS.

Then, we draw the error curve of Algorithm 1, Algorithms 2\* and 2 with different values of power parameter for  $k = 100$  in Figure 5(a). From it we see the shifted power iteration improves the accuracy. We test dashSVD (Algorithm 5), LanczosBD, PRIMME\_SVDS, and lansvd [23] for computing the truncated SVD with  $k = 100$ . They are run with 16 threads. The subspace dimensionalities of



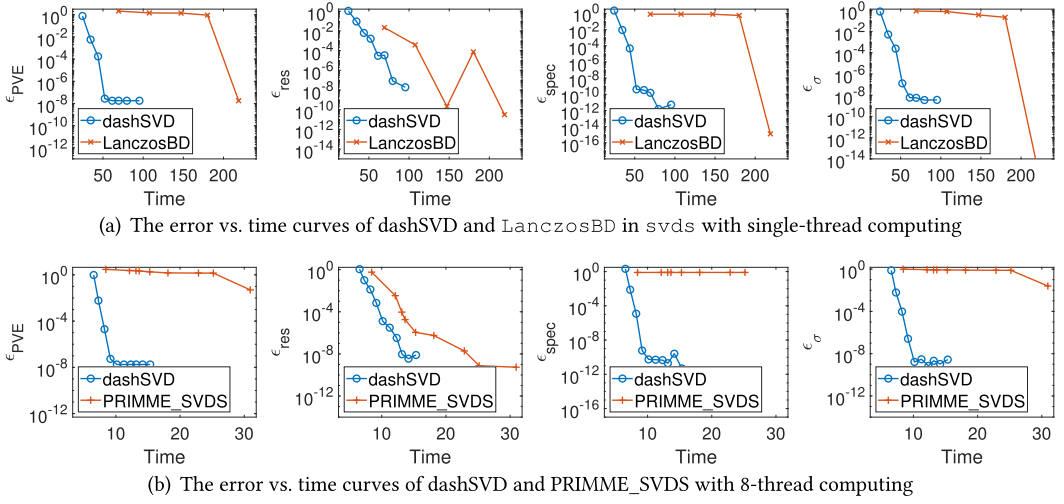


Fig. 4. The error vs. time curves of dashSVD compared with LanczosBD in svds and PRIMME\_SVDS for LargeRegFile ( $k = 100$ ). Time is given in seconds.

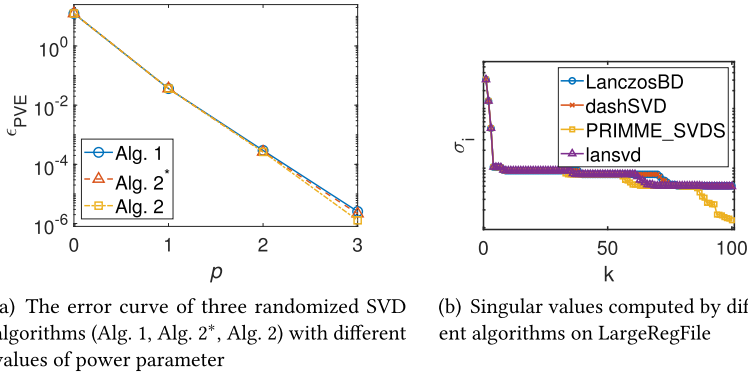


Fig. 5. Experimental results on LargeRegFile ( $k = 100$ ).

LanczosBD, lansvd, and PRIMME\_SVDS are set to 150 for fair comparison. lansvd in PROPACK is an algorithm for accelerating svds. We test its Fortran version V2.14 parallelized with OpenMP. Notice that lansvd is faster than svds in single-thread computing, but the parallel performance of lansvd is poor. Besides, the resulted singular vectors computed by lansvd occasionally contains NaN. This makes difficulty of comparison. The convergence tolerance parameter in LanczosBD, PRIMME\_SVDS is set to  $10^{-10}$  and the convergence tolerance parameter in lansvd is set to  $10^{-6}$  to make these algorithms produce results with low relative residual error, while the tol in dashSVD is set to  $10^{-3}$  for producing results with low PVE. The computed singular values are plotted in Figure 5(b), and more results are listed in Table 3. These results show that accuracy issue occurs for both lansvd and PRIMME\_SVDS on this matrix with multiple singular values. With the results in Figure 4(a) together we can see that PRIMME\_SVDS is not robust for matrix with multiple singular values. On contrary, dashSVD performs very well. Its speedup to LanczosBD is  $6\times$ , with a small  $\epsilon_{PVE}$  of  $1.8 \times 10^{-8}$ .

Table 3. Computational Results of LanczosBD, lansvd, PRIMME\_SVDS, and dashSVD (tol =  $10^{-3}$ ) on LargeRegFile ( $k = 100$ )

Method	Time (s)	Mem (GB)	$\epsilon_{\text{PVE}}$	$\epsilon_{\text{res}}$	$\epsilon_{\text{spec}}$	$\epsilon_{\sigma}$
LanczosBD	77.3	5.02	-	-	-	-
lansvd	516	6.42	NA <sup>a</sup>	NA <sup>a</sup>	NA <sup>a</sup>	0.36
PRIMME_SVDS	16.3	5.09	1.42	3.4E-9	0.79	0.74
dashSVD	12.7	5.78	1.8E-8	2.0E-5	1.1E-10	4.0E-9

Mem represents memory.

<sup>a</sup>NA represents the singular vectors computed by lansvd contains NaN, which makes some metrics of the error criterion unavailable.

Table 4. Results of the dashSVD with PVE Accuracy Control in 16-Thread Computing

Matrix	Algorithm 5						Algorithm 5 without surrogate		
	Time (s)	$N_p$	$\epsilon_{\text{PVE}}$	$\epsilon_{\text{res}}$	$\epsilon_{\text{spec}}$	$\epsilon_{\sigma}$	Time (s)	Inc. (%)	$\epsilon_{\text{PVE}}$
SNAP	1.93	7	5.7E-3	3.1E-2	4.3E-4	3.3E-3	2.31	20	5.3E-3
MovieLens	14.4	6	2.6E-3	2.8E-2	6.0E-5	1.7E-3	15.4	7	3.0E-3
Rucci1	13.6	9	1.9E-2	3.9E-2	9.9E-3	1.0E-2	26.4	94	2.3E-3
Aminer	341	7	3.1E-3	2.7E-2	3.6E-4	1.8E-3	353	4	6.6E-3
uk-2005	877	6	2.4E-3	2.6E-2	2.8E-4	1.5E-3	986	12	2.2E-3
sk-2005	1,385	5	8.4E-4	2.0E-2	5.2E-5	6.3E-4	1525	10	1.2E-3

$k = 100$ , tol =  $10^{-2}$ , and Inc. represents the increment of time.

### 4.3 Validation of the Accuracy-Control Scheme Based on PVE Criterion

In this subsection, we validate the effectiveness of the proposed PVE accuracy control, and the efficiency brought by the surrogate strategy, i.e., using  $\sigma_i(\mathbf{A}^T \mathbf{A} \mathbf{Q} - \alpha \mathbf{Q}) + \alpha$  to monitor  $\hat{\sigma}_i(\mathbf{A})^2$ . We set tol =  $10^{-2}$  and compare the results of Algorithm 5 with those obtained without the surrogate strategy. The latter realizes the accuracy control through iteratively computing SVD of matrix  $\mathbf{B} = \mathbf{A} \mathbf{Q}$  and checking the termination criterion Equation (19). The results are listed in Table 4.  $N_p$  denotes the number of power iteration steps executed when the algorithm terminates with satisfied criterion. If without the surrogate strategy, the resulted  $N_p$  has same value or just smaller for 1. We can see that the dashSVD with accuracy control obtains results with good accuracy in the four metrics, and the result's  $\epsilon_{\text{PVE}}$  is about or less than  $10^{-2}$ . The surrogate strategy enables convenient monitor of singular value, without which the runtime is increased by 4% to 94%.

## 5 Conclusions

We have developed a randomized algorithm named dashSVD to efficiently compute the truncated SVD in the regime of low accuracy. dashSVD is built on a shifted power iteration scheme with dynamic shifts and an accuracy-control mechanism monitoring the PVE-based accuracy criterion. Inheriting the parallelizability of the randomized SVD algorithm, it exhibits much shorter runtime than the state-of-the-art SVD algorithms for sparse matrices. dashSVD is expected to be useful in the application scenarios where low-accuracy SVD is allowed.

In the future, efficient implementation of the randomized SVD algorithm on distributed-memory parallel architecture can be explored.

## Appendices

### A Proof of Theorem 1

We first give the following two Lemmas [31]. Lemma 4 is equivalent to Lemma A.1 in [31] and Lemma 5 contains the equivalent forms of Equations (A.18) and (A.40) in [31].  $\|\cdot\|$  denotes the spectral norm.

LEMMA 4. Suppose that  $i, k, l, m$ , and  $n$  are positive integers with  $k \leq l \leq n \leq m$ . Suppose further that  $\mathbf{A}$  is a real  $m \times n$  matrix,  $\mathbf{Q}$  is a real  $m \times k$  matrix whose columns are orthonormal,  $\mathbf{R}$  is a real  $k \times l$  matrix,  $\mathbf{F}$  is a real  $n \times l$  matrix, and  $\mathbf{G}$  is a real  $l \times n$  matrix. Then,

$$\|\mathbf{Q}\mathbf{Q}^T\mathbf{A} - \mathbf{A}\| \leq 2\|\mathbf{F}\mathbf{G}(\mathbf{A}^T\mathbf{A})^i\mathbf{A}^T - \mathbf{A}^T\| + 2\|\mathbf{F}\|\|\mathbf{Q}\mathbf{R} - (\mathbf{G}(\mathbf{A}^T\mathbf{A})^i\mathbf{A}^T)^T\|. \quad (24)$$

LEMMA 5. Suppose that  $i, k, l, m$ , and  $n$  are positive integers with  $n \leq m$  and  $l = k + s \leq n - k$ . Suppose further that  $\mathbf{A}$  is a real  $m \times n$  matrix,  $\mathbf{G}$  is a real  $l \times n$  matrix whose entries are i.i.d. Gaussian random variables of zero mean and unit variance, positive integer  $j < k$  such that the  $j$ th largest singular value  $\sigma_j$  of  $\mathbf{A}$  is positive. If real numbers  $\beta, \gamma > 1$  satisfying  $\phi = \frac{1}{\sqrt{2\pi(l-j+1)}} \left( \frac{e}{(l-j+1)\beta} \right)^{l-j+1} + \frac{1}{4\gamma(\gamma^2-1)} \left( \frac{1}{\sqrt{\pi(n-k)}} \left( \frac{2\gamma^2}{e\gamma^2-1} \right)^{n-k} + \frac{1}{\sqrt{\pi l}} \left( \frac{2\gamma^2}{e\gamma^2-1} \right)^l \right) \leq 1$ . Then, there exists a real  $n \times l$  matrix  $\mathbf{F}$  such that

$$\|\mathbf{F}\mathbf{G}(\mathbf{A}^T\mathbf{A})^i\mathbf{A}^T - \mathbf{A}^T\|^2 \leq \left( 2l^2\beta^2\gamma^2 \left( \frac{\sigma_{j+1}}{\sigma_j} \right)^{4i} + 1 \right) \sigma_{j+1}^2 + \left( 2l(n-k)\beta^2\gamma^2 \left( \frac{\sigma_{k+1}}{\sigma_j} \right)^{4i} + 1 \right) \sigma_{k+1}^2, \quad (25)$$

and

$$\|\mathbf{F}\| \leq \frac{\sqrt{l}\beta}{\sigma_j^{2i}}, \quad (26)$$

with probability not less than  $1 - \phi$ .

According to Lemmas 4 and 5, the following Lemma is derived, which bounds the approximation error of Algorithm 1's result [31]. Notice that we assume  $l = k + s \leq n - k$  as it always holds in practical scenarios.

LEMMA 6. Suppose  $\mathbf{A} \in \mathbb{R}^{m \times n}$  ( $m \geq n$ ), and  $k, s$ , and  $p$  are the parameters in Algorithm 1.  $l = k + s \leq n - k$ , and  $\mathbf{Q}$  in size  $m \times l$  is the obtained orthonormal matrix with Algorithm 1. If positive integer  $j < k$ , and real numbers  $\beta, \gamma > 1$  satisfying  $\phi = \frac{1}{\sqrt{2\pi(l-j+1)}} \left( \frac{e}{(l-j+1)\beta} \right)^{l-j+1} + \frac{1}{4\gamma(\gamma^2-1)} \left( \frac{1}{\sqrt{\pi(n-k)}} \left( \frac{2\gamma^2}{e\gamma^2-1} \right)^{n-k} + \frac{1}{\sqrt{\pi l}} \left( \frac{2\gamma^2}{e\gamma^2-1} \right)^l \right) \leq 1$ , then

$$\|\mathbf{Q}\mathbf{Q}^T\mathbf{A} - \mathbf{A}\| \leq 2\sqrt{\left( 2l^2\beta^2\gamma^2 \left( \frac{\sigma_{j+1}}{\sigma_j} \right)^{4p} + 1 \right) \sigma_{j+1}^2 + \left( 2l(n-k)\beta^2\gamma^2 \left( \frac{\sigma_{k+1}}{\sigma_j} \right)^{4p} + 1 \right) \sigma_{k+1}^2}, \quad (27)$$

with probability not less than  $1 - \phi$ , where  $\sigma_j$  denotes the  $j$ th largest singular value of  $\mathbf{A}$ .

PROOF. In Algorithm 1, the result matrix  $\mathbf{Q}$  is formed by a set of orthonormal basis of range  $(\mathbf{A}(\mathbf{A}^T\mathbf{A})^p\mathbf{\Omega})$ , where  $\mathbf{\Omega}$  in size  $n \times l$  is a Gaussian random matrix and  $p$  is the power parameter. Therefore, each column of  $\mathbf{A}(\mathbf{A}^T\mathbf{A})^p\mathbf{\Omega} = (\mathbf{\Omega}^T(\mathbf{A}^T\mathbf{A})^p\mathbf{A}^T)^T$  can be expressed as a linear combination of  $\mathbf{Q}$ 's columns. This means there is a matrix  $\mathbf{R}$  in  $l \times l$  satisfying  $(\mathbf{\Omega}^T(\mathbf{A}^T\mathbf{A})^p\mathbf{A}^T)^T = \mathbf{Q}\mathbf{R}$ . So,

$$\|\mathbf{Q}\mathbf{R} - (\mathbf{\Omega}^T(\mathbf{A}^T\mathbf{A})^p\mathbf{A}^T)^T\| = 0. \quad (28)$$

Suppose  $\mathbf{F}$  is a real  $m \times l$  matrix,  $k = l, p = i$ ,  $\mathbf{G} = \mathbf{\Omega}^T$ , and  $\mathbf{R}$  is the one in Equation (28). According to Lemma 4, we have

$$\begin{aligned} \|\mathbf{Q}\mathbf{Q}^T\mathbf{A} - \mathbf{A}\| &\leq 2\|\mathbf{F}\mathbf{\Omega}^T(\mathbf{A}^T\mathbf{A})^p\mathbf{A}^T - \mathbf{A}^T\| + 2\|\mathbf{F}\|\|\mathbf{Q}\mathbf{R} - (\mathbf{\Omega}^T(\mathbf{A}^T\mathbf{A})^p\mathbf{A}^T)^T\| \\ &= 2\|\mathbf{F}\mathbf{\Omega}^T(\mathbf{A}^T\mathbf{A})^p\mathbf{A}^T - \mathbf{A}^T\|. \end{aligned} \quad (29)$$

Then, according to Lemma 5, there exists  $\mathbf{F}$  such that Equation (25) holds. Combining it and Equation (29) we derive Equation (27).  $\square$

Before proving Theorem 1, we prove Lemma 7 which is similar to the proof of Lemma A.2 in [31].

LEMMA 7. Suppose that  $i, k, l, m$ , and  $n$  are positive integers with  $n \leq m$  and  $l = k + s \leq n - k$ . Suppose further that  $\mathbf{A}$  is a real  $m \times n$  matrix,  $\mathbf{G}$  is a real  $l \times n$  matrix whose entries are i.i.d. Gaussian random variables of zero mean and unit variance, positive integer  $j < k$  such that the  $j$ th largest singular value  $\sigma_j$  of  $\mathbf{A}$  is positive. If real numbers  $\beta, \gamma > 1$  satisfying  $\phi = \frac{1}{\sqrt{2\pi(l-j+1)}} \left( \frac{e}{(l-j+1)\beta} \right)^{l-j+1} + \frac{1}{4\gamma(\gamma^2-1)} \left( \frac{1}{\sqrt{\pi(n-k)}} \left( \frac{2\gamma^2}{e\gamma^2-1} \right)^{n-k} + \frac{1}{\sqrt{\pi l}} \left( \frac{2\gamma^2}{e\gamma^2-1} \right)^l \right) \leq 1$ . Then, there exists a real  $n \times l$  matrix  $\mathbf{F}$  such that

$$\begin{aligned} \left\| \mathbf{F}\mathbf{G}\mathbf{A}^T \prod_{c=1}^i (\mathbf{A}\mathbf{A}^T - \alpha_c \mathbf{I}) - \mathbf{A}^T \right\|^2 &\leq \left( 2l^2 \beta^2 \gamma^2 \prod_{c=1}^i \left( \frac{\sigma_{j+1}^2 - \alpha_c}{\sigma_j^2 - \alpha_c} \right)^2 + 1 \right) \sigma_{j+1}^2 \\ &\quad + \left( 2l(n-k) \beta^2 \gamma^2 \prod_{c=1}^i \left( \frac{\sigma_{k+1}^2 - \alpha_c}{\sigma_j^2 - \alpha_c} \right)^2 + 1 \right) \sigma_{k+1}^2, \end{aligned} \quad (30)$$

and

$$\|\mathbf{F}\| \leq \frac{\sqrt{l}\beta}{\prod_{c=1}^i (\sigma_j^2 - \alpha_c)}, \quad (31)$$

with probability not less than  $1 - \phi$ , where  $\mathbf{I}$  is the identity matrix and  $0 \leq \alpha_c \leq \sigma_l^2/2$  for any  $c \leq i$ .

PROOF. We following the proof of Lemma A.2 in [31] to construct  $\mathbf{F}$  satisfying Equations (30) and (31). The formulation of SVD of  $\mathbf{A}$  is  $\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$ , where  $\mathbf{U}$  is a real  $m \times n$  matrix whose columns are orthonormal,  $\mathbf{\Sigma}$  is a real diagonal  $n \times n$  matrix, and  $\mathbf{V}$  is a real unitary  $n \times n$  matrix, such that  $\Sigma(y, y) = \sigma_y$  for  $y = 1, 2, \dots, n$ , and  $\sigma_y$  is the  $y$ th largest singular value of  $\mathbf{A}$ . Then, several auxiliary matrices  $\mathbf{H}$ ,  $\mathbf{\Lambda}$ ,  $\mathbf{\Gamma}$ ,  $\mathbf{S}_1$ ,  $\mathbf{S}_2$ , and  $\mathbf{S}_3$  are defined to assist the proof. Suppose  $\mathbf{H}$  is the leftmost  $l \times j$  block of the  $l \times n$  matrix  $\mathbf{G}\mathbf{V}$ ,  $\mathbf{\Lambda}$  is the  $l \times (k-j)$  block of  $\mathbf{G}\mathbf{V}$  whose first column is the  $(k+1)$ -th column of  $\mathbf{G}\mathbf{V}$ , and  $\mathbf{\Gamma}$  is the rightmost  $l \times (n-k)$  block of  $\mathbf{G}\mathbf{V}$ , so that

$$\mathbf{G}\mathbf{V} = [\mathbf{H}, \mathbf{\Lambda}, \mathbf{\Gamma}]. \quad (32)$$

Notice that  $\mathbf{V}$  is real and unitary, and the entries of  $\mathbf{G}$  are i.i.d. Gaussian random variables of zero mean and unit variance. Then, the entries of  $\mathbf{H}$  are also i.i.d. Gaussian random variables of zero mean and unit variance, as are the entries of  $\mathbf{\Lambda}$ , and as are the entries of  $\mathbf{\Gamma}$ . Suppose  $\mathbf{H}^\dagger$  is the real  $j \times l$  matrix given by the formula

$$\mathbf{H}^\dagger = (\mathbf{H}^T\mathbf{H})^{-1}\mathbf{H}^T, \quad (33)$$

where  $\mathbf{H}^T\mathbf{H}$  is invertible with high probability due to Lemma 2.7 in [31]. Suppose  $\mathbf{S}_1$  to be the leftmost uppermost  $j \times j$  block of  $\mathbf{\Sigma}$ ,  $\mathbf{S}_2$  is the leftmost uppermost  $(k-j) \times (k-j)$  block of  $\mathbf{\Sigma}$  whose

leftmost uppermost entry is the entry in the  $(j + 1)$ -th row and  $(j + 1)$ -th column of  $\Sigma$ , and  $\mathbf{S}_3$  is the rightmost lowermost  $(n - k) \times (n - k)$  block of  $\Sigma$ , so that

$$\Sigma = \begin{pmatrix} \mathbf{S}_1 & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{S}_2 & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{S}_3 \end{pmatrix}. \quad (34)$$

Suppose  $\mathbf{P}$  is a real  $n \times l$  matrix given by

$$\mathbf{P} = \begin{pmatrix} \prod_{c=1}^i (\mathbf{S}_1^2 - \alpha_{i-c+1} \mathbf{I})^{-1} \mathbf{H}^\dagger \\ \mathbf{0} \\ \mathbf{0} \end{pmatrix}. \quad (35)$$

Finally,  $\mathbf{F}$  is constructed as an  $n \times l$  matrix given by

$$\mathbf{F} = \mathbf{V}\mathbf{P} = \mathbf{V} \begin{pmatrix} \prod_{c=1}^i (\mathbf{S}_1^2 - \alpha_{i-c+1} \mathbf{I})^{-1} \mathbf{H}^\dagger \\ \mathbf{0} \\ \mathbf{0} \end{pmatrix}. \quad (36)$$

According to the proof of Lemma A.2 in [31],

$$\|\mathbf{H}^\dagger\| \leq \sqrt{l}\beta, \quad (37)$$

with probability not less than

$$1 - \frac{1}{\sqrt{2\pi(l-j+1)}} \left( \frac{e}{(l-j+1)\beta} \right)^{l-j+1}. \quad (38)$$

Combining Equations (34), (36), and (37), the fact  $\Sigma$  is zero off its main diagonal, and the fact that  $\mathbf{V}$  is unitary yields Equation (31).

We now show that  $\mathbf{F}$  defined in Equation (36) satisfies Equation (30).

Suppose  $\hat{\mathbf{S}}_1 = \prod_{c=1}^i (\mathbf{S}_1^2 - \alpha_{i-c+1} \mathbf{I})^{-1}$ . Combining Equations (32) and (36) yields

$$\begin{aligned} & \mathbf{F}\mathbf{G}\mathbf{A}^\mathbf{T} \prod_{c=1}^i (\mathbf{A}\mathbf{A}^\mathbf{T} - \alpha_c \mathbf{I}) - \mathbf{A}^\mathbf{T} \\ &= \mathbf{V} \begin{pmatrix} \hat{\mathbf{S}}_1 \mathbf{H}^\dagger \\ \mathbf{0} \\ \mathbf{0} \end{pmatrix} [\mathbf{H}, \mathbf{\Lambda}, \mathbf{\Gamma}] \Sigma \prod_{c=1}^i (\Sigma^2 - \alpha_c \mathbf{I}) \mathbf{U}^\mathbf{T} - \mathbf{V} \Sigma \mathbf{U}^\mathbf{T} \\ &= \mathbf{V} \begin{pmatrix} \hat{\mathbf{S}}_1 & \hat{\mathbf{S}}_1 \mathbf{H}^\dagger \mathbf{\Lambda} & \hat{\mathbf{S}}_1 \mathbf{H}^\dagger \mathbf{\Gamma} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} \end{pmatrix} \prod_{c=1}^i (\Sigma^2 - \alpha_c \mathbf{I}) \Sigma \mathbf{U}^\mathbf{T} - \mathbf{V} \Sigma \mathbf{U}^\mathbf{T} \\ &= \mathbf{V} \left( \begin{pmatrix} \mathbf{I} & \hat{\mathbf{S}}_1 \mathbf{H}^\dagger \mathbf{\Lambda} \prod_{c=1}^i (\mathbf{S}_2^2 - \alpha_c \mathbf{I}) & \hat{\mathbf{S}}_1 \mathbf{H}^\dagger \mathbf{\Gamma} \prod_{c=1}^i (\mathbf{S}_3^2 - \alpha_c \mathbf{I}) \\ \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} \end{pmatrix} - \mathbf{I} \right) \Sigma \mathbf{U}^\mathbf{T} \\ &= \mathbf{V} \begin{pmatrix} \mathbf{0} & \hat{\mathbf{S}}_1 \mathbf{H}^\dagger \mathbf{\Lambda} \prod_{c=1}^i (\mathbf{S}_2^2 - \alpha_c \mathbf{I}) \mathbf{S}_2 & \hat{\mathbf{S}}_1 \mathbf{H}^\dagger \mathbf{\Gamma} \prod_{c=1}^i (\mathbf{S}_3^2 - \alpha_c \mathbf{I}) \mathbf{S}_3 \\ \mathbf{0} & -\mathbf{S}_2 & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & -\mathbf{S}_3 \end{pmatrix} \mathbf{U}^\mathbf{T}. \end{aligned} \quad (39)$$

Furthermore,

$$\begin{aligned} & \left\| \begin{pmatrix} \mathbf{0} & \hat{\mathbf{S}}_1 \mathbf{H}^\dagger \Lambda \prod_{c=1}^i (\mathbf{S}_2^2 - \alpha_c \mathbf{I}) \mathbf{S}_2 & \hat{\mathbf{S}}_1 \mathbf{H}^\dagger \Gamma \prod_{c=1}^i (\mathbf{S}_3^2 - \alpha_c \mathbf{I}) \mathbf{S}_3 \\ \mathbf{0} & -\mathbf{S}_2 & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & -\mathbf{S}_3 \end{pmatrix} \right\|^2 \\ & \leq \left\| \hat{\mathbf{S}}_1 \mathbf{H}^\dagger \Lambda \prod_{c=1}^i (\mathbf{S}_2^2 - \alpha_c \mathbf{I}) \mathbf{S}_2 \right\|^2 + \left\| \hat{\mathbf{S}}_1 \mathbf{H}^\dagger \Gamma \prod_{c=1}^i (\mathbf{S}_3^2 - \alpha_c \mathbf{I}) \mathbf{S}_3 \right\|^2 + \|\mathbf{S}_2\|^2 + \|\mathbf{S}_3\|^2. \end{aligned} \quad (40)$$

Moreover,

$$\left\| \hat{\mathbf{S}}_1 \mathbf{H}^\dagger \Lambda \prod_{c=1}^i (\mathbf{S}_2^2 - \alpha_c \mathbf{I}) \mathbf{S}_2 \right\| \leq \|\hat{\mathbf{S}}_1\| \|\mathbf{H}^\dagger\| \|\Lambda\| \left\| \prod_{c=1}^i (\mathbf{S}_2^2 - \alpha_c \mathbf{I}) \right\| \|\mathbf{S}_2\|, \quad (41)$$

and

$$\left\| \hat{\mathbf{S}}_1 \mathbf{H}^\dagger \Gamma \prod_{c=1}^i (\mathbf{S}_3^2 - \alpha_c \mathbf{I}) \mathbf{S}_3 \right\| \leq \|\hat{\mathbf{S}}_1\| \|\mathbf{H}^\dagger\| \|\Gamma\| \left\| \prod_{c=1}^i (\mathbf{S}_3^2 - \alpha_c \mathbf{I}) \right\| \|\mathbf{S}_3\|. \quad (42)$$

Then, according to Proposition 2 and the fact  $0 \leq \alpha_c \leq \sigma_l^2/2$  for any  $c \leq i$  we can get

$$\|\hat{\mathbf{S}}_1\| \leq \prod_{c=1}^i \frac{1}{\sigma_j^2 - \alpha_c}, \quad (43)$$

$$\left\| \prod_{c=1}^i (\mathbf{S}_2^2 - \alpha_c \mathbf{I}) \right\| \leq \prod_{c=1}^i (\sigma_{j+1}^2 - \alpha_c), \quad (44)$$

$$\left\| \prod_{c=1}^i (\mathbf{S}_3^2 - \alpha_c \mathbf{I}) \right\| \leq \prod_{c=1}^i (\sigma_{k+1}^2 - \alpha_c), \quad (45)$$

$$\|\mathbf{S}_2\| \leq \sigma_{j+1}, \quad (46)$$

$$\|\mathbf{S}_3\| \leq \sigma_{k+1}. \quad (47)$$

Then, combining Equations (40)–(47) and the fact that the columns of  $\mathbf{U}$  and  $\mathbf{V}$  are orthonormal yields

$$\begin{aligned} \left\| \mathbf{F} \mathbf{G} \mathbf{A}^\top \prod_{c=1}^i (\mathbf{A} \mathbf{A}^\top - \alpha_c \mathbf{I}) - \mathbf{A}^\top \right\|^2 & \leq \left( \|\mathbf{H}^\dagger\|^2 \|\Lambda\|^2 \prod_{c=1}^i \left( \frac{\sigma_{j+1}^2 - \alpha_c}{\sigma_j^2 - \alpha_c} \right)^2 + 1 \right) \sigma_{j+1}^2 \\ & \quad + \left( \|\mathbf{H}^\dagger\|^2 \|\Gamma\|^2 \prod_{c=1}^i \left( \frac{\sigma_{k+1}^2 - \alpha_c}{\sigma_j^2 - \alpha_c} \right)^2 + 1 \right) \sigma_{k+1}^2. \end{aligned} \quad (48)$$

Combining Lemma 2.6 in [31] and the fact that the entries of  $\Lambda$  are i.i.d. Gaussian random variables of zero mean and unit variance, as are the entries of  $\Gamma$ , yields

$$\|\Lambda\| \leq \sqrt{2l}\gamma, \quad \|\Gamma\| \leq \sqrt{2(n-k)}\gamma, \quad (49)$$

with probability not less than

$$1 - \frac{1}{4(\gamma^2 - 1)\sqrt{\pi(n-k)}\gamma^2} \left( \frac{2\gamma^2}{e^{\gamma^2-1}} \right)^{(n-k)} - \frac{1}{4(\gamma^2 - 1)\sqrt{\pi l}\gamma^2} \left( \frac{2\gamma^2}{e^{\gamma^2-1}} \right)^l. \quad (50)$$



Finally, combining Equations (37), (48), and (49) yields

$$\begin{aligned} \left\| \mathbf{F} \mathbf{G} \mathbf{A}^T \prod_{c=1}^i (\mathbf{A} \mathbf{A}^T - \alpha_c \mathbf{I}) - \mathbf{A}^T \right\|^2 &\leq \left( 2l^2 \beta^2 \gamma^2 \prod_{c=1}^i \left( \frac{\sigma_{j+1}^2 - \alpha_c}{\sigma_j^2 - \alpha_c} \right)^2 + 1 \right) \sigma_{j+1}^2 \\ &\quad + \left( 2l(n-k) \beta^2 \gamma^2 \prod_{c=1}^i \left( \frac{\sigma_{k+1}^2 - \alpha_c}{\sigma_j^2 - \alpha_c} \right)^2 + 1 \right) \sigma_{k+1}^2, \end{aligned} \quad (51)$$

with a probability not less than  $1 - \phi$ .

□

Now, we can prove Theorem 1.

PROOF. Suppose  $\mathbf{I}$  is an identity matrix. In Algorithm 2, the result matrix  $\mathbf{Q}$  is formed by a set of orthonormal basis of  $\text{range}(\prod_{c=1}^p (\mathbf{A} \mathbf{A}^T - \alpha_{p-c+1} \mathbf{I}) \mathbf{A} \mathbf{\Omega})$ , where  $\mathbf{\Omega}$  in size  $n \times l$  is a Gaussian random matrix and  $p$  is the power parameter. Therefore, each column of  $\prod_{c=1}^p (\mathbf{A} \mathbf{A}^T - \alpha_{p-c+1} \mathbf{I}) \mathbf{A} \mathbf{\Omega} = (\mathbf{\Omega}^T \mathbf{A}^T \prod_{c=1}^p (\mathbf{A} \mathbf{A}^T - \alpha_c \mathbf{I}))^T$  can be expressed as a linear combination of  $\mathbf{Q}$ 's columns. This means there is a matrix  $\mathbf{R}$  in  $l \times l$  satisfying  $(\mathbf{\Omega}^T \mathbf{A}^T \prod_{c=1}^p (\mathbf{A} \mathbf{A}^T - \alpha_c \mathbf{I}))^T = \mathbf{Q} \mathbf{R}$ . So,

$$\left\| \mathbf{Q} \mathbf{R} - (\mathbf{\Omega}^T \mathbf{A}^T \prod_{c=1}^p (\mathbf{A} \mathbf{A}^T - \alpha_c \mathbf{I}))^T \right\| = 0. \quad (52)$$

Suppose  $\mathbf{F}$  is a real  $m \times l$  matrix,  $k = l$ ,  $p = i$ ,  $\mathbf{G} = \mathbf{\Omega}^T$  and  $\mathbf{R}$  is the one in Equation (52). According to the proof of Lemma A.1 in [31], we have

$$\begin{aligned} \|\mathbf{Q} \mathbf{Q}^T \mathbf{A} - \mathbf{A}\| &\leq 2 \left\| \mathbf{F} \mathbf{\Omega}^T \mathbf{A}^T \prod_{c=1}^p (\mathbf{A} \mathbf{A}^T - \alpha_c \mathbf{I}) - \mathbf{A}^T \right\| + 2 \|\mathbf{F}\| \left\| \mathbf{Q} \mathbf{R} - \left( \mathbf{\Omega}^T \mathbf{A}^T \prod_{c=1}^p (\mathbf{A} \mathbf{A}^T - \alpha_c \mathbf{I}) \right)^T \right\| \\ [6pt] &= 2 \left\| \mathbf{F} \mathbf{\Omega}^T \mathbf{A}^T \prod_{c=1}^p (\mathbf{A} \mathbf{A}^T - \alpha_c \mathbf{I}) - \mathbf{A}^T \right\|. \end{aligned} \quad (53)$$

According to Lemma 7, there exists  $\mathbf{F}$  such that Equation (30) holds. Combining it and Equation (53) we derive Equation (15) in Theorem 1.

□

COROLLARY 1. If  $\mathbf{A} \in \mathbb{R}^{m \times n}$  ( $m \geq n$ ), Gaussian random matrix  $\mathbf{\Omega} \in \mathbb{R}^{n \times l}$  and parameters  $p$ ,  $l$ ,  $j$ ,  $k$ ,  $\beta$ , and  $\gamma$  are the same, the bound of approximation error of Algorithm 2's result given in the right-hand side of Equation (15) in Theorem 1 is not larger than that of Algorithm 1's result given in the right-hand side of Equation (27) in Lemma 6. Moreover, the equality of the both bounds only occurs when  $\sigma_j = \sigma_{j+1} = \dots = \sigma_{k+1}$ .

PROOF. Because for any  $c$ ,  $1 < c \leq p$ , we have positive number  $\alpha_c \leq \sigma_l^2/2$  according to Algorithm 2 ( $\alpha_1 = 0$  in Algorithm 2), we can derive

$$\frac{\sigma_{j+1}^2 - \alpha_c}{\sigma_j^2 - \alpha_c} \leq \frac{\sigma_{j+1}^2}{\sigma_j^2}, \quad \frac{\sigma_{k+1}^2 - \alpha_c}{\sigma_j^2 - \alpha_c} \leq \frac{\sigma_{k+1}^2}{\sigma_j^2}. \quad (54)$$

Therefore,

$$\prod_{c=1}^p \left( \frac{\sigma_{j+1}^2 - \alpha_c}{\sigma_j^2 - \alpha_c} \right)^2 \leq \left( \frac{\sigma_{j+1}}{\sigma_j} \right)^{4p}, \quad \prod_{c=1}^p \left( \frac{\sigma_{k+1}^2 - \alpha_c}{\sigma_j^2 - \alpha_c} \right)^2 \leq \left( \frac{\sigma_{k+1}}{\sigma_j} \right)^{4p}, \quad (55)$$

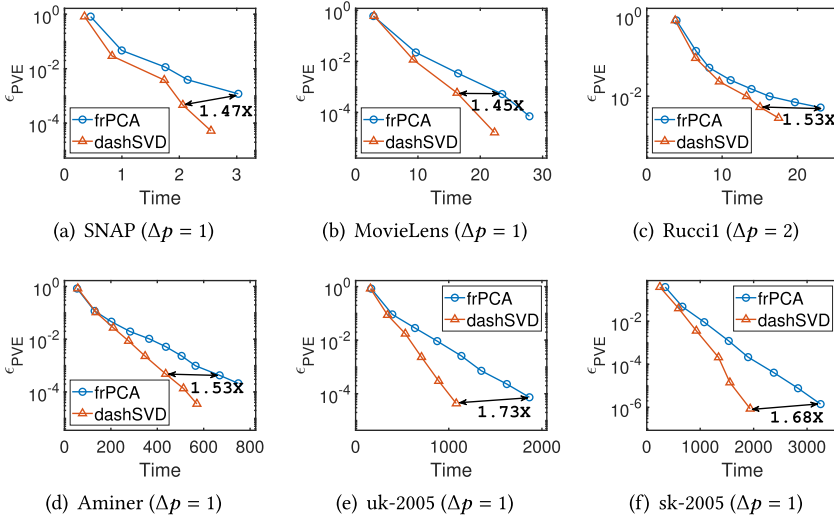


Fig. 6. The  $\epsilon_{PVE}$  vs. runtime curves of dashSVD (Algorithm 4) and frPCA with the number of power iterations  $p$  increased from 0 and in a step  $\Delta p$ . Time is given in seconds.

which suggests that the bound of approximation error of Algorithm 2's result given in the right-hand side of Equation (15) in Theorem 1 is not larger than that of Algorithm 1's result given in the right-hand side of Equation (27) in Lemma 6.

Because  $j < j+1 < k+1$ ,  $\sigma_j = \sigma_{k+1}$  only occurs when  $\sigma_j = \sigma_{j+1} = \dots = \sigma_{k+1}$ . Otherwise,  $\sigma_j > \sigma_{k+1}$  which derives

$$\frac{\sigma_{k+1}^2 - \alpha_c}{\sigma_j^2 - \alpha_c} < \frac{\sigma_{k+1}^2}{\sigma_j^2}, \quad \prod_{c=1}^p \left( \frac{\sigma_{k+1}^2 - \alpha_c}{\sigma_j^2 - \alpha_c} \right)^2 < \left( \frac{\sigma_{k+1}}{\sigma_j} \right)^{4p} \quad (56)$$

and the bound of approximation error given in Equation (15) is less than that given in Equation (27). Therefore, the equality of the both bounds only occurs when  $\sigma_j = \sigma_{j+1} = \dots = \sigma_{k+1}$ .  $\square$

## B More Comparison Results

Firstly, we present the comparison results of our dashSVD program in C with frPCA [16] to further validate the shifted power iteration for the sparse matrices. The both programs are parallelly executed with 16 threads. Because for frPCA the number of power iterations ( $p$ ) should be specified, we use the dashSVD without accuracy control, i.e., Algorithm 4, in this experiment. It is actually an improved version of frPCA, as they use the same accelerating skills for handling sparse matrices. By setting the different power parameter  $p$ , we do the comprehensive investigation on the relationship of runtime and  $\epsilon_{PVE}$  for the two algorithms when producing results with similar  $\epsilon_{PVE}$ .  $p$  is increased with a step  $\Delta p$ . The results for the sparse matrices SNAP, MovieLens, Rucci1, Aminer, uk-2005, and sk-2005 are shown in Figure 6. The curves in Figure 6 show that dashSVD can reach more accurate results during the same runtime compared with frPCA for all tested matrices, and the speedup ratio is from 1.45 $\times$  to 1.73 $\times$ . Besides, the memory cost of dashSVD on SNAP, MovieLens, rucci1, Aminer, uk-2005, and sk-2005 is 0.30 GB, 0.96 GB, 4.66 GB, 31.5 GB, 147 GB, and 200 GB, respectively, which is smaller than 0.35 GB, 0.98 GB, 4.69 GB, 31.5 GB, 162 GB, and 220 GB of frPCA on these matrices.

The results of LanczosBD and dashSVD with single-thread computing for MovieLens, Rucci1, Aminer and sk-2005 are plotted in Figure 7. They show that our algorithm always costs less runtime

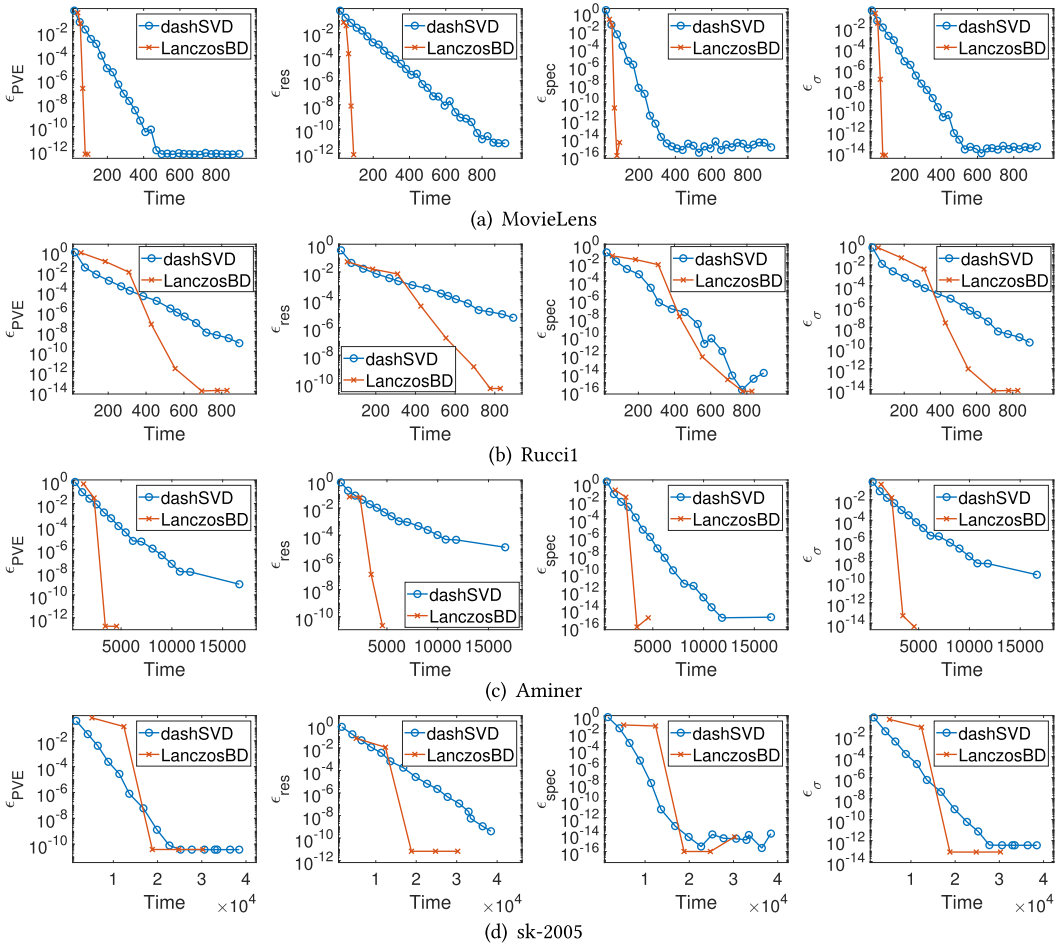


Fig. 7. More error vs. time curves of dashSVD and LanczosBD in svds with single-thread computing ( $k = 100$ ). Time is given in seconds.

than LanczosBD when producing results with not very high accuracy in error metrics  $\epsilon_{PVE}$ ,  $\epsilon_{spec}$ , and  $\epsilon_{\sigma}$ . For the datasets with slower decaying trend of singular values (Rucci1 and sk-2005), our dashSVD gains more acceleration than other datasets. For the matrices with faster decaying trend of singular values, e.g. MovieLens and Aminer, LanczosBD performs better than dashSVD on the  $\epsilon_{res}$  criterion.

The results of PRIMME\_SVDS and dashSVD with eight-thread computing for MovieLens, Rucci1, Aminer, and sk-2005 are plotted in Figure 8. Figure 8 shows that our dashSVD costs less runtime than PRIMME\_SVDS when producing results with not high accuracy. Because PRIMME\_SVDS firstly computes the EVD of  $A^T A$ , it performs better on the matrix when  $m \gg n$ , e.g., MovieLens, Rucci1 and Aminer. Therefore, the speedup ratios of our dashSVD on the three matrices are smaller than those on SNAP, uk-2005, and sk-2005.

Then, the computational results with similar accuracy of LazySVD, lansvd, and dashSVD with 16 threads are listed in Table 5. For fair comparison, LazySVD is also implemented in C with MKL and the subspace dimensionality of lansvd is fixed to  $1.5k$ . We varies the tol in lansvd to get the results with similar accuracy compared with dashSVD. From Table 5, we see that dashSVD is  $3.9\times$  to  $11\times$

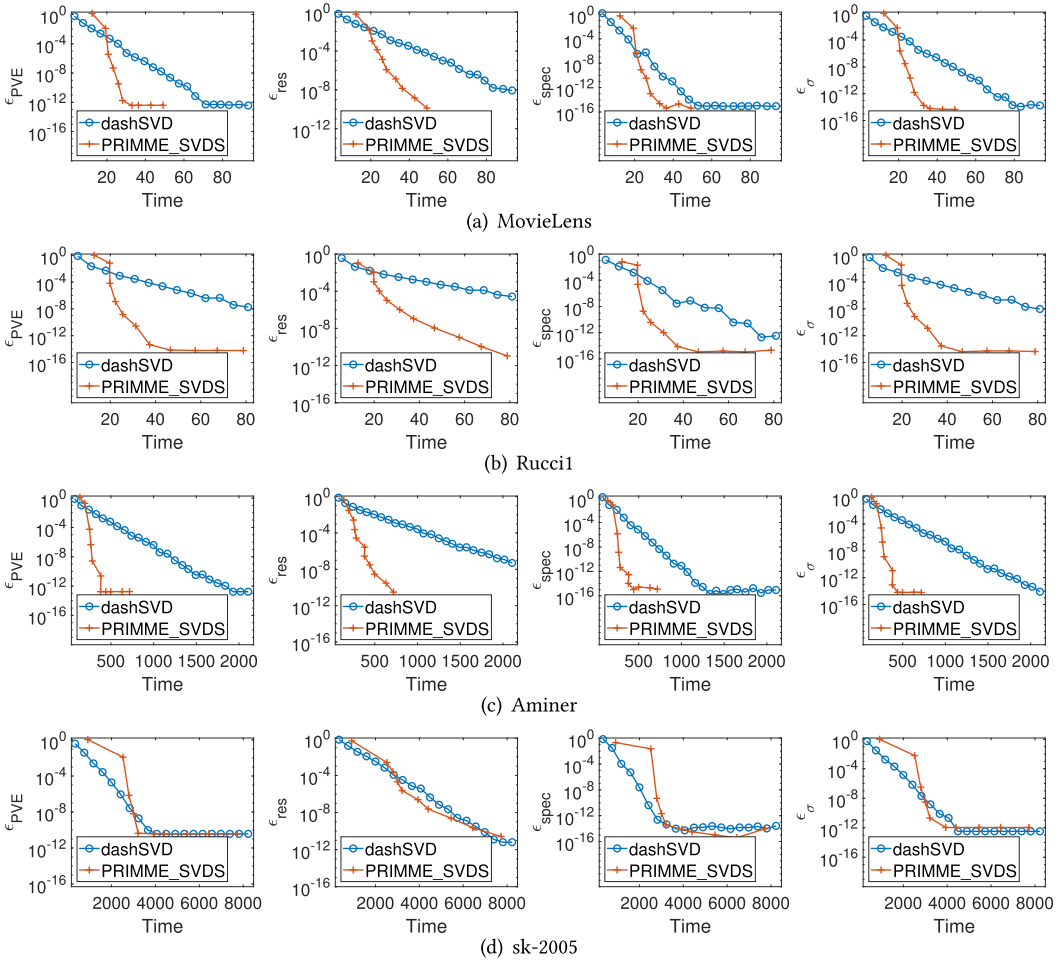


Fig. 8. More error vs. runtime curves of dashSVD and PRIMME\_SVDS with eight-thread computing ( $k = 100$ ). Time is given in seconds.

Table 5. Computational Results of LazySVD, lansvd, and dashSVD (tol =  $1e-2$ ) with 16-Thread Computing ( $k = 100$ )

Matrix	LazySVD			lansvd			dashSVD					
	Time (s)	$\epsilon_{PVE}$	$\epsilon_{\sigma}$	Time (s)	$\epsilon_{PVE}$	$\epsilon_{\sigma}$	Time(s)	Mem (GB)	$\epsilon_{PVE}$	$\epsilon_{\sigma}$	SP1	SP2
SNAP	9.58	6.5E-3	3.2E-3	4.31	NA <sup>1</sup>	1.9E-2	1.93	0.30	5.7E-3	3.3E-3	5.0	2.2
MovieLens	55.5	3.3E-3	1.6E-3	41.6	5.0E-2	2.6E-2	14.4	0.96	2.6E-3	1.7E-3	3.9	2.9
Rucc1	152	2.5E-2	1.2E-2	49.8	NA <sup>a</sup>	3.0E-2	13.6	4.66	1.9E-2	1.0E-2	11	3.7
Aminer	1961	6.8E-3	3.4E-3	796	NA <sup>a</sup>	1.6E-2	341	31.5	3.1E-3	1.8E-3	5.8	2.3
uk-2005	7707	3.6E-3	1.7E-3	NA <sup>b</sup>	NA <sup>b</sup>	NA <sup>b</sup>	877	147	2.4E-3	1.5E-3	8.8	NA <sup>b</sup>
sk-2005	13634	1.5E-3	7.4E-4	NA <sup>b</sup>	NA <sup>b</sup>	NA <sup>b</sup>	1385	200	8.4E-4	6.2E-4	9.8	NA <sup>b</sup>

Mem represents the memory cost. SP1 and SP2 represent the speed-up ratio from dashSVD to LazySVD and lansvd.

<sup>a</sup>NA represents the singular vectors computed by lansvd contains NaN, which makes some metrics of error criterion unavailable.

<sup>b</sup>NA represents program breaks down due to segment fault.

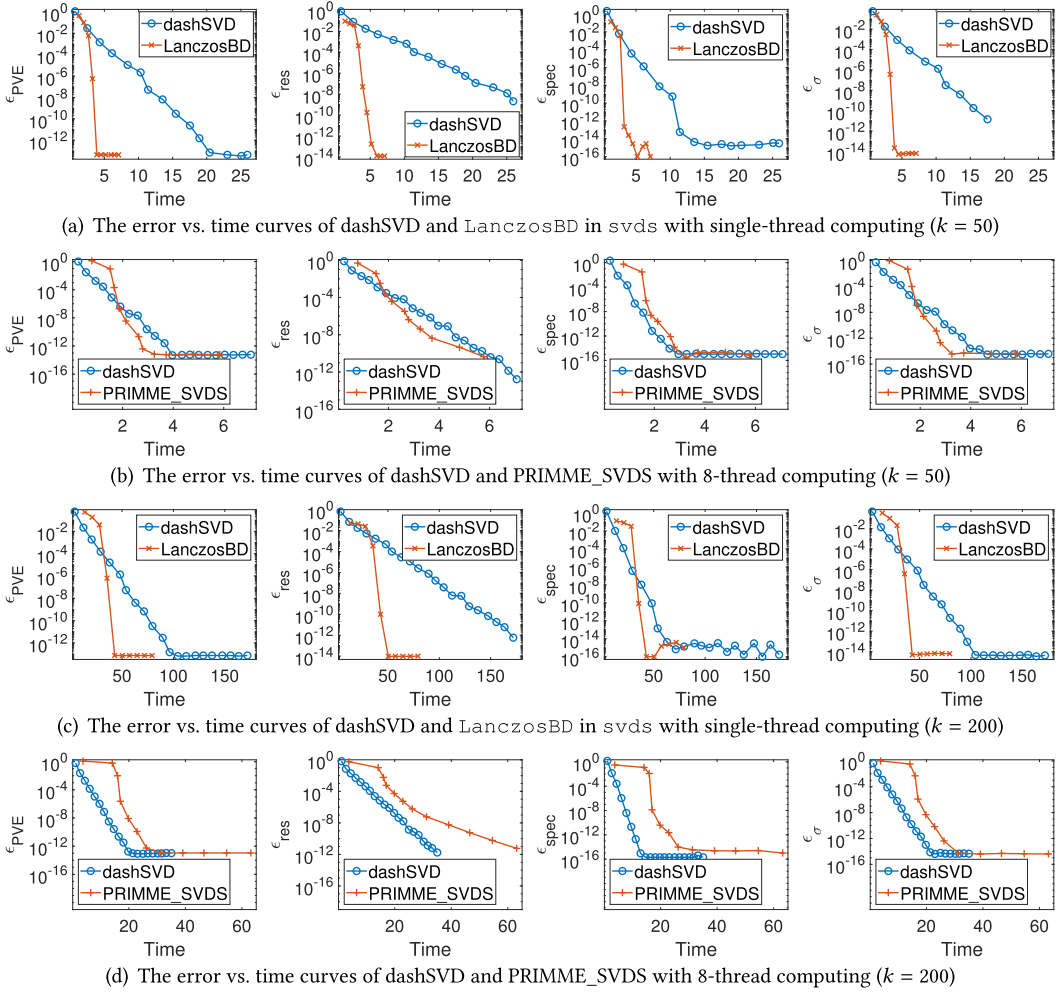


Fig. 9. The error vs. time curves of dashSVD compared with LanczosBD in svds and PRIMME\_SVDS for SNAP ( $k = 50$  and  $200$ ). Time is given in seconds.

faster than LazySVD and  $2.2\times$  to  $3.7\times$  faster than lansvd for achieving similar accuracy. Besides, LazySVD and lansvd costs less memory than our dashSVD when the dimensionalities of subspace are the same. Notice that lansvd may produce singular vectors with NaN, which implies lansvd is not robust for computing the truncated SVD of large datasets.

Besides, we set  $k = 50, 200$  on SNAP dataset to validate the efficiency of dashSVD with different rank parameters. The error vs. runtime curves are plotted in Figure 9. The results of dashSVD compared with PRIMME\_SVDS and LanczosBD show that dashSVD performs better than the competitors when  $k = 200$ , rather than when  $k = 50$  and  $k = 100$ . This implies that the advantage of dashSVD would increase when  $k$  increases. Besides, the memory cost of dashSVD, LanczosBD, and PRIMME\_SVDS are 0.16 GB, 0.17 GB, and 0.37 GB, respectively, for  $k = 50$ , and 0.59 GB, 0.59 GB, and 0.74 GB, respectively, for  $k = 200$ .

Finally, to show the variance of dashSVD when computing truncated SVD, we run dashSVD on SNAP with  $k = 100$  for 100 times for each  $p = 0, 4, 8, 12, 16, 20$ . The whisker plots of the four error

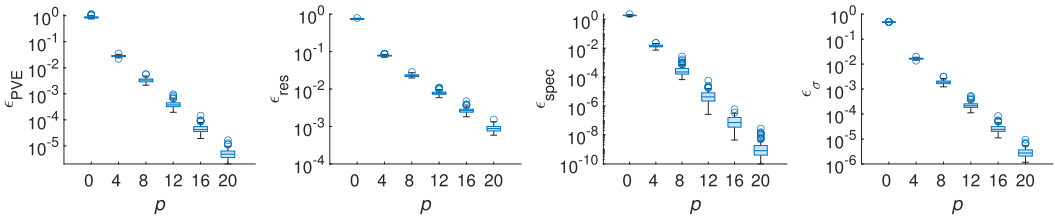


Fig. 10. The whisker plots of four error metrics vs.  $p$  of dashSVD for SNAP ( $k = 100$ ).

metrics are shown in Figure 10. It shows that when computing low-accuracy SVD (with  $p = 0, 4, 8$ ), dashSVD produces results with little variance. For these cases, the standard deviations of most error metrics are no more than 5% of mean value.

## References

- [1] AMiner. 2021. Aminer. Retrieved from <https://www.aminer.cn>
- [2] Intel. 2021. Intel oneAPI Math Kernel Library. Retrieved from <https://software.intel.com/content/www/us/en/develop/tools/oneapi/components/onemkl.html>
- [3] The University of Texas at Austin. 2021. RandQR. Retrieved from [https://users.oden.utexas.edu/pgm/Codes/randqb\\_codes\\_intel\\_mkl.zip](https://users.oden.utexas.edu/pgm/Codes/randqb_codes_intel_mkl.zip)
- [4] GitHub. 2022. frPCA\_sparse. Retrieved from [https://github.com/XuFengthucs/frPCA\\_sparse](https://github.com/XuFengthucs/frPCA_sparse)
- [5] GitHub. 2022. Primme. Retrieved from <https://github.com/primme/primme>
- [6] Zeyuan Allen-Zhu and Yuanzhi Li. 2016. LazySVD: Even faster SVD decomposition yet without agonizing pain. In *Proceedings of the Advances in Neural Information Processing Systems*. 974–982.
- [7] The OpenMP ARB. 2022. The OpenMP API Specification for Parallel Programming. Retrieved from <https://www.openmp.org/>
- [8] James Baglama and Lothar Reichel. 2005. Augmented implicitly restarted Lanczos bidiagonalization methods. *SIAM Journal on Scientific Computing* 27, 1 (2005), 19–42.
- [9] Satish Balay, Shrirang Abhyankar, Mark F. Adams, Steven Benson, Jed Brown, Peter Brune, Kris Buschelman, Emil M. Constantinescu, Lisandro Dalcin, Alp Dener, Victor Eijkhout, William D. Gropp, Václav Hapla, Tobin Isaac, Pierre Jolivet, Dmitry Karpeev, Dinesh Kaushik, Matthew G. Knepley, Fande Kong, Scott Kruger, Dave A. May, Lois Curfman McInnes, Richard T. Mills, Lawrence Mitchell, Todd Munson, Jose E. Roman, Karl Rupp, Patrick Sanan, Jason Sarich, Barry F. Smith, Stefano Zampini, Hong Zhang, Hong Zhang, and Junchao Zhang. 2022. PETSc Web page. Retrieved from <https://petsc.org/>
- [10] N. Benjamin Erichson, Steven L. Brunton, and J. Nathan Kutz. 2017. Compressed singular value decomposition for image and video processing. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV '17)*. 1880–1888.
- [11] Paolo Boldi, Marco Rosa, Massimo Santini, and Sebastiano Vigna. 2011. Layered label propagation: A multiresolution coordinate-free ordering for compressing social networks. In *Proceedings of the 20th International Conference on World Wide Web*. ACM Press, 587–596.
- [12] Paolo Boldi and Sebastiano Vigna. 2004. The WebGraph Framework I: Compression Techniques. In *Proceedings of the 13th International World Wide Web Conference (WWW '04)*. ACM Press, Manhattan, USA, 595–601.
- [13] Timothy A. Davis and Yifan Hu. 2011. The University of Florida sparse matrix collection. *ACM Transactions on Mathematical Software (TOMS)* 38, 1 (2011), 1–25.
- [14] Xiangyun Ding, Wenjian Yu, Yuyang Xie, and Shenghua Liu. 2020. Efficient model-based collaborative filtering with fast adaptive PCA. In *Proceedings of the IEEE 32nd International Conference on Tools with Artificial Intelligence (ICTAI '20)*. IEEE, 955–960.
- [15] Carl Eckart and Gale Young. 1936. The approximation of one matrix by another of lower rank. *Psychometrika* 1, 3 (1936), 211–218.
- [16] Xu Feng, Yuyang Xie, Mingye Song, Wenjian Yu, and Jie Tang. 2018. Fast randomized PCA for sparse data. In *Proceedings of the 10th Asian Conference on Machine Learning (ACML '18)*. 710–725.
- [17] Gene H. Golub and Charles F. Van Loan. 2012. *Matrix Computations*. JHU Press.
- [18] Nathan Halko, Per-Gunnar Martinsson, and Joel A. Tropp. 2011. Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions. *SIAM Review* 53, 2 (2011), 217–288.



- [19] Franklin M. Harper and Joseph A. Konstan. 2016. The MovieLens datasets: History and context. *ACM Transactions on Interactive Intelligent Systems (TiiS)* 5, 4 (2016), 19.
- [20] Michael T. Heath. 2018. *Scientific Computing: An Introductory Survey* (2nd. ed.). SIAM.
- [21] Vicente Hernandez, José E. Roman, and Andrés Tomas. 2008. A robust and efficient parallel SVD solver based on restarted Lanczos bidiagonalization. *Electronic Transactions on Numerical Analysis* 31 (2008), 68–85.
- [22] Roger A. Horn and Charles R. Johnson. 1991. *Topics in Matrix Analysis*. Cambridge University Press. DOI: <https://doi.org/10.1017/CBO9780511840371>
- [23] Rasmus M. Larsen. 2004. PROPACK-Software for Large and Sparse SVD Calculations. Retrieved from <http://sun.stanford.edu/rmunk/PROPACK>
- [24] Jure Leskovec and Andrej Krevl. 2014. SNAP Datasets: Stanford Large Network Dataset Collection. Retrieved from <http://snap.stanford.edu/data>
- [25] Huamin Li, George C. Linderman, Arthur Szlam, Kelly P. Stanton, Yuval Kluger, and Mark Tygert. 2017. Algorithm 971: An implementation of a randomized algorithm for principal component analysis. *ACM Transactions on Mathematical Software (TOMS)* 43, 3 (2017), 1–14.
- [26] Michael W. Mahoney. 2011. Randomized algorithms for matrices and data. *Foundations and Trends® in Machine Learning* 3, 2 (2011), 123–224.
- [27] Per-Gunnar Martinsson and Joel A. Tropp. 2020. Randomized numerical linear algebra: Foundations and algorithms. *Acta Numerica* 29 (2020), 403–572.
- [28] Per-Gunnar Martinsson and Sergey Voronin. 2016. A randomized blocked algorithm for efficiently computing rank-revealing factorizations of matrices. *SIAM Journal on Scientific Computing* 38 (2016), S485–S507.
- [29] Cameron Musco and Christopher Musco. 2015. Randomized block Krylov methods for stronger and faster approximate singular value decomposition. In *Proceedings of the Advances in Neural Information Processing Systems*. 1396–1404.
- [30] Balabanov Oleg, Beaupere Matthias, Grigori Laura, and Lederer Victor. 2023. Block subsampled randomized Hadamard transform for Nyström approximation on distributed architectures. In *Proceedings of the International Conference on Machine Learning*. 1564–1576.
- [31] Vladimir Rokhlin, Arthur Szlam, and Mark Tygert. 2010. A randomized algorithm for principal component analysis. *SIAM Journal on Matrix Analysis and Applications* 31, 3 (2010), 1100–1124.
- [32] Andreas Stathopoulos and James R. McCombs. 2010. PRIMME: Preconditioned iterative multimethod eigensolver: Methods and software description. *ACM Transactions on Mathematical Software (TOMS)* 37, 2 (2010), 1–30.
- [33] Sergey Voronin and Per-Gunnar Martinsson. 2015. RSVDPACK: An implementation of randomized algorithms for computing the singular value, interpolative, and CUR decompositions of matrices on multi-core and GPU architectures. arXiv:1502.05366. Retrieved from <https://arxiv.org/abs/1502.05366>
- [34] Lingfei Wu, Eloy Romero, and Andreas Stathopoulos. 2017. PRIMME\_SVDS: A high-performance preconditioned SVD solver for accurate large-scale computations. *SIAM Journal on Scientific Computing* 39, 5 (2017), S248–S271.

Received 2 December 2022; revised 2 April 2024; accepted 11 April 2024