Randomized Singular Value Decomposition: A Study

Chen Chen[a]

[a]*Department of Computer Science, University of Maryland*

Abstract

In this project we study randomized Singular Value Decomposition (SVD). Apart from the standard randomized SVD algorithm, two modified versions, one for eliminating the influence of the round-off error and one for adaptively determining a desired error threshold, are introduced and discussed. Their efficiency and stability are demonstrated by experiments on two test matrices, one is a given matrix which has full rank and slowly-decaying singular values and the other one is generated from the numerical solutions of a piece-wise constant diffusion equation. A special treatment when the input matrix is positive semi-definite is also discussed and validated with numerical results.

1. Introduction

In the area of applied mathematics, scientific computing and machine learning, matrix decompositions are fundamental and perhaps well-developed tools that has a long history. In particular, low-rank matrix decompositions play a vital role in data analysis and compression, dimension reduction methods and tensor methods. However, with the high-speed development of computer hardware, massive datasets as well as new tasks that emerge in the information sciences have posed a computational challenge for traditional algorithms, making them inadequate in terms of memory requirements and time cost in many situations. Recently, the simple but powerful concept of randomness has been introduced into traditional algorithms for matrix decompositions as a strategy to alleviate

*Corresponding author
Email address: `cchen24@cs.umd.edu` (Chen Chen)

such problems. The intuitions and ideas behind these random-informed algorithms are simple but surprisingly effective. People have compared them with standard deterministic algorithms, and found that the randomized algorithms is capable to give matrix decompositions that are accurate to any error tolerance above machine precision. This offers possibility to make trade-offs between accuracy and speed if desired. Also, randomized algorithms are claimed to be faster in many cases and even more robust.

The essential idea of randomized algorithms is to first use some amount of randomness in order to generate a much smaller matrix from the original matrix (potentially of big size). Here the generated matrix is expected to capture the action of the original matrix as much as possible, thus can be called the range approximator. Then the smaller matrix is used to compute the desired low-rank approximation. In this project, we focus on the randomized Singular Value Decomposition (SVD) as well as several of its variants and examine their accuracy and stability. The rest of this report is organized as follows: in the following section, we introduce the standard randomized SVD and another two modified versions that focus on the generation of the range approximator. Special post-processing for a given positive semi-definite matrix is also included in the following section. In section 3 we give some discussions and potential future works.

2. Algorithm and Numerical Studies

2.1. General Framework

The task of computing an randomized SVD to a given matrix can be naturally split into two computational stages. In the first stage, a low-dimensional subspace that captures the action of the matrix, which is also known as range approximator, is constructed by random sampling. The second one is to restrict the matrix to the calculated subspace in the first stage and then compute a standard SVD of the reduced matrix. According to [1], we present the general framework for randomized SVD in Algorithm 1.

Algorithm 1 Randomized SVD

```
1: procedure
2: Input: An $m \times n$ matrix $\boldsymbol{A}$, a target number $k$ of singular vectors, and an exponent $q$ (a small number like 1 or 2).
3: Output: An approximate rank-$2k$ factorization $\widehat{\boldsymbol{U}}\widehat{\boldsymbol{\Sigma}}\widehat{\boldsymbol{V}}^{\boldsymbol{T}}$ where $\widehat{\boldsymbol{U}}$ and $\widehat{\boldsymbol{V}}$ are orthonormal, and $\widehat{\boldsymbol{\Sigma}}$ is nonnegative and diagonal.
4: Stage A (Power Iteration):
5:     Generate an $n \times 2k$ Gaussian test matrix $\boldsymbol{\Omega}$.
6:     Form $\boldsymbol{Y} = (\boldsymbol{A}\boldsymbol{A}^T)^q \boldsymbol{A}\boldsymbol{\Omega}$ by multiplying alternately with $\boldsymbol{A}$ and $\boldsymbol{A}^T$.
7:     Construct a $m \times 2k$ matrix $\boldsymbol{Q}$ whose columns form an orthonormal basis for the range of Y via the QR factorization $\boldsymbol{Y} = \boldsymbol{Q}\boldsymbol{R}$.
8: Stage B:
9:     Form $\boldsymbol{B} = \boldsymbol{Q}^T\boldsymbol{A}$.
10:    Compute an SVD of the small matrix: $\boldsymbol{B} = \widetilde{\boldsymbol{U}}\widehat{\boldsymbol{\Sigma}}\widehat{\boldsymbol{V}}^T$.
11:    Set $\widehat{\boldsymbol{U}} = \boldsymbol{Q}\widetilde{\boldsymbol{U}}$.
```

2.1.1. Numerical Study

Figure 1 presents the results of Algorithm 1 on the matrix from [2], where the approximation error is defined as

$$\textit{Approximation error} = \left\| \boldsymbol{A} - \widehat{\boldsymbol{U}}\widehat{\boldsymbol{\Sigma}}\widehat{\boldsymbol{V}}^{\boldsymbol{T}} \right\| \tag{1}$$

where $\|\cdot\|$ denotes the l_2 operator norm. It can be observed that with the increase of q, the approximation error decreases under the same value of k. In this case, $q = 2$ achieves the same performance with $q = 3$. In terms of accuracy, we can see that the approximation error decreases as k increases, which indicated that the randomized SVD is working. The phenomenon that the approximation error decreases not that fast is that the input matrix has a full rank and its singular values also decrease very slowly. Another point is that when k reached around 100 then different q values lead to the same approximation error. This is due to the round-off errors in the power iteration and we will discuss this issue in detail later.

In next two sections, we first focus on Stage A, and introduce two variant to Algorithm 1 will be introduced and demonstrated by numerical experiments. After that, we will present how Stage B can be further improved when the input matrix is positive semi-definite.

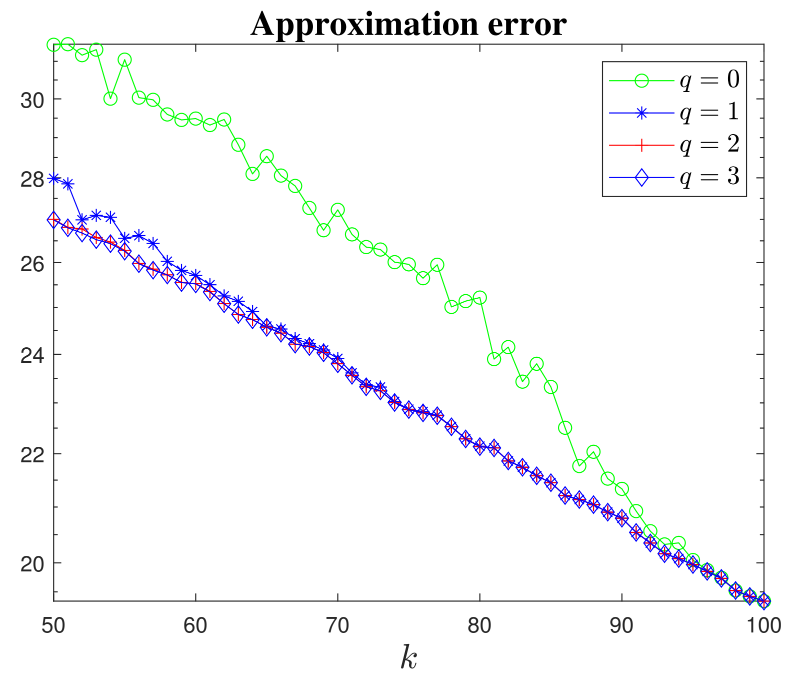


Figure 1: Performance of Algorithm 1 on the matrix from [2] under different q

2.2. Randomized Subspace Iteration

The power iteration in randomized SVD algorithm was designed to make the algorithm perform better when the given matrix has a flat singular spectrum or a very large size [1]. It has been shown that when the original scheme ($q = 0$) produces a basis whose approximation error is within a factor C of the optimum, the power scheme produces an approximation error within $C^{1/(2q+1)}$ of the optimum. However, this consequently results in that all information associated with singular values smaller than roughly $\mu^{1/(2q+1)} \|\boldsymbol{A}\|$ (here $\mu \approx 10^{-16}$ is the machine precision) is lost. To this end, a modified version of power iteration called randomized subspace iteration [3] is proposed, which is shown in Algorithm 2. Basically Algorithm 2 addresses this problem by orthonormalizing the columns of the sample matrix between each application of $\boldsymbol{A}$ and $\boldsymbol{A}^T$.

Algorithm 2 Randomized Subspace Iteration

```
1: procedure
2: Input: An $m \times n$ matrix $\boldsymbol{A}$ and integers $k$ and $q$.
3: Output: An $m \times n$ orthonormal matrix $\boldsymbol{Q}$ whose range approximates the
   range of $\boldsymbol{A}$.
4:     Generate an $n \times 2k$ Gaussian test matrix $\boldsymbol{\Omega}$.
5:     Form $\boldsymbol{Y}_0 = \boldsymbol{A\Omega}$ and compute its QR factorization $\boldsymbol{Y}_0 = \boldsymbol{Q}_0\boldsymbol{R}_0$.
6:     for $j = 1, 2 \ldots, q$ do
7:         Form $\widetilde{\boldsymbol{Y}}_j = \boldsymbol{A}^T\boldsymbol{Q}_{j-1}$ and compute its QR factorization $\widetilde{\boldsymbol{Y}}_j = \widetilde{\boldsymbol{Q}}_j\widetilde{\boldsymbol{R}}_j$.
8:         Form $\boldsymbol{Y}_j = \boldsymbol{A}\widetilde{\boldsymbol{Q}}_j$ and compute its QR factorization $\boldsymbol{Y}_j = \boldsymbol{Q}_j\boldsymbol{R}_j$.
9:     $\boldsymbol{Q} = \boldsymbol{Q}_q$.
```

2.2.1. Numerical Study

We test this algorithm with another matrix that comes from the piece-wise component diffusion equation introduced in [4]. We consider the following governing equations posed on the physical domain $D = (-1,\ 1) \times (-1,\ 1)$:

$$-\nabla \cdot [a(x,\xi)\nabla u_{sol}(x,\xi)] = 1 \quad \text{in} \quad D \times \Gamma, \tag{2}$$

$$u_{sol}(x,\xi) = 0 \quad \text{on} \quad \partial D \times \Gamma. \tag{3}$$

Dividing the physical domain D into N_D subdomains, each of which is denoted by D_k for $k = 1, \ldots, N_D$, the permeability coefficient $a(x,\xi)$ is defined to be a piecewise constant function

$$a(x,\xi)\,|_{D_k} = \xi_k,\ \ k = 1, \ldots, N_D, \tag{4}$$

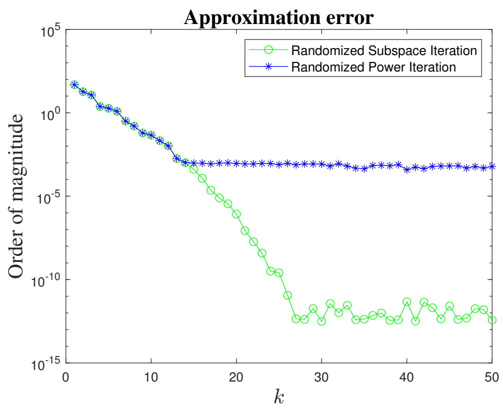
where $\xi_1, \ldots, \xi_{N_D}$ are independently and uniformly distributed in $[0.01, 1]$. Here we consider the case $N_D = 16$ (4×4). For each realization of ξ, the simulator for (2)–(3) is set to the finite element method [5], where a bilinear $\mathcal{Q}_1$ finite element approximation is used to discretize the physical domain with a 33×33 grid, i.e., the dimension of the simulator output is $d = 1089$. We calculate solutions for 500 realizations of ξ and stack them into a 500×1089 matrix $\boldsymbol{A}$:

$$\boldsymbol{A} = \begin{bmatrix} -\boldsymbol{u}_1^T- \\ -\boldsymbol{u}_2^T- \\ -\cdots- \\ -\boldsymbol{u}_n^T- \end{bmatrix} \tag{5}$$

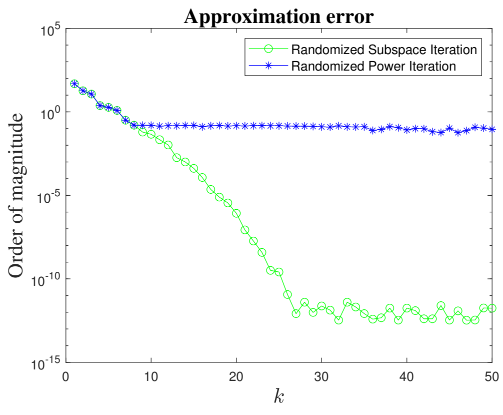
where $n = 500$. Computing the SVD for such matrix is essential in reduced-order modeling [4].

We replace Stage A in Algorithm 1 with Algorithm 2 and test randomized SVD using randomized subspace iteration on matrix $\boldsymbol{A}$. Setting $q = 1, 2, 3$, we test the algorithm with k ranging from 1 to 50. The results are shown in Figure 2. The approximation error is defined the same with that in Section 2.1.1. As we can see, the approximation error of randomized subspace iteration and randomized power iteration are generally the same when k is small, and start to diverge when k reaches some point. Since then, the approximation error for the power iteration does not decrease while that of the randomized subspace iteration continues decreasing until a stable level (after k equals the real rank of the matrix). Also it can be seen that the bigger the power coefficient q is, the smaller that divergence point is. We also record the divergence points (denoted as k') as well as the corresponding singular values and calculated the rough bound $\mu^{1/(2q+1)} \|\boldsymbol{A}\|$ mentioned before. This data are shown in Table 1 below. From the table we can see that in our experiment the first time σ_k go below the threshold $\mu^{1/(2q+1)} \|\boldsymbol{A}\|$, the approximation errors for the above two algorithms start to diverge. This observation is consistent with the motivation behind the randomized subspace iteration.

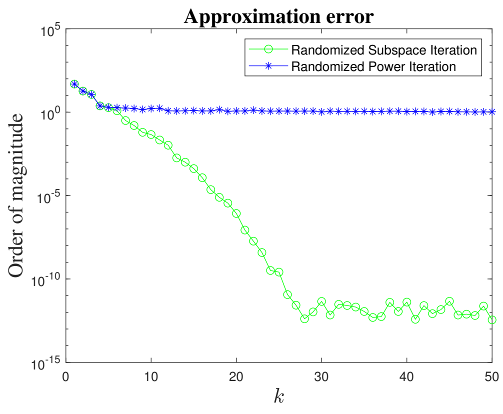
The randomized subspace iteration, summarized as Algorithm 2, is algebraically equivalent to Stage A in Algorithm 1 when executed in exact arithmetic [1]. It is recommended to use Algorithm 2 because its computational costs are similar to those of Stage A in Algorithm 1, even though the former is



(a) $q = 1$



(b) $q = 2$



(c) $q = 3$

Figure 2: Performance of randomize SVD with randomized subspace iteration described in Algorithm 2 on the matrix $\boldsymbol{A}$ in equation (5) under different q

Table 1: Divergence points and the corresponding bounds and singular values

q	k'	$\mu^{1/(2q+1)} \|\boldsymbol{A}\|$	$\sigma_{k'}$	$\sigma_{k'-1}$
1	14	0.0020	0.0018	0.0104
2	9	0.2737	0.1586	0.3157
3	6	2.2466	1.8651	2.3819

substantially more accurate in floating-point arithmetic.

2.3. Adaptive Randomized Range Finder

Algorithm 1 is designed to solve the fixed-rank problem where the target rank of the input matrix is specified in advance, which in many cases is not flexible and needs much human artificial attempt. In stead, we want a scheme for estimating how well a generated matrix $\boldsymbol{Q}$ captures the range of $\boldsymbol{A}$. To this end, authors in [6] developed a probabilistic error estimator to adaptively construct matrix $\boldsymbol{Q}$. Using the lemma proved in [6], one can shown that draw a sequence $\{\boldsymbol{\omega}^{(i)} : i = 1, 2, ..., r\}$ of standard Gaussian vectors where r is a small integer, then

$$\left\|(\boldsymbol{I} - \boldsymbol{Q}\boldsymbol{Q}^{\boldsymbol{T}})\boldsymbol{A}\right\| \leq 10\sqrt{\frac{2}{\pi}} \max_{i=1,\ldots,r} \left\|(\boldsymbol{I} - \boldsymbol{Q}\boldsymbol{Q}^{\boldsymbol{T}})\boldsymbol{A}\boldsymbol{\omega}^{(i)}\right\|, \tag{6}$$

with probability at least $1 - 10^{-r}$.

The problem in Stage A of Algorithm 1 is then reformulated as follows: suppose that $\boldsymbol{A}$ is an $m \times n$ matrix and ϵ is a computational tolerance, we seek an integer l and an $m \times l$ orthonormal matrix $\boldsymbol{Q}^{(l)}$ such that

$$\left\|(\boldsymbol{I} - \boldsymbol{Q}^{(l)}(\boldsymbol{Q}^{(l)})^T)\boldsymbol{A}\right\| \leq \epsilon \tag{7}$$

with a probability very close to 1. This is called the fixed-precision problem. Based on the above probabilistic error estimator, Algorithm 3 was proposed to solve the fixed-precision problem.

The algorithm keep track of an additional r random samples and utilize them both for the construction of matrix $\boldsymbol{Q}$ and the error estimation. Step $10 - 11$ is for numerical stable and accuracy since without reprojection and normalization the $\boldsymbol{q}^{(i)}$ vectors become small as the basis starts to capture most of the action of $\boldsymbol{A}$.

Algorithm 3 Randomized Subspace Iteration

```
1: procedure
2: Input: An $m \times n$ matrix $\boldsymbol{A}$ and integers $r$, a tolerance $\epsilon$.
3: Output: An $m \times l$ orthonormal matrix $\boldsymbol{Q}$ that satisfies equation (7) with
   probability at least $1 - 10^{-r}$. Here $l$ is not pre-defined.
4:     Draw standard Gaussian vectors $\boldsymbol{\omega}^{(1)}, \boldsymbol{\omega}^{(2)} \dots, \boldsymbol{\omega}^{(r)}$ of length $n$.
5:     for $i = 1, 2 \dots, r$ do
6:         Compute $\boldsymbol{y}^{(i)} = A\boldsymbol{\omega}^{(i)}$.
7:     $j = 0$.
8:     while $\max\{\|\boldsymbol{y}^{j+1}\|, \|\boldsymbol{y}^{j+2}\|, \dots, \|\boldsymbol{y}^{j+r}\|\} > \epsilon/(10\sqrt{2/\pi})$ do
9:         j=j+1.
10:        $\boldsymbol{y}^{(j)} \leftarrow (\boldsymbol{I} - \boldsymbol{Q}^{(j-1)}(\boldsymbol{Q}^{(j-1)})^T)\boldsymbol{y}^{(j)}$.
11:        $\boldsymbol{q}^{(j)} = \boldsymbol{y}^{(j)} / \|\boldsymbol{y}^{(j)}\|$
12:        $\boldsymbol{Q}^{(j)} \leftarrow [\boldsymbol{Q}^{(j)} \;\; \boldsymbol{q}^{j}]$.
13:        Draw a standard Gaussian vector $\boldsymbol{\omega}^{(j+r)}$ of length $n$.
14:        $\boldsymbol{y}^{(j+r)} \leftarrow (\boldsymbol{I} - \boldsymbol{Q}^{(j)}(\boldsymbol{Q}^{(j)})^T)\boldsymbol{A}\boldsymbol{\omega}^{(j+r)}$.
15:        for $i = j+1, (j+1), \dots, (j+r-1)$ do
16:            $\boldsymbol{y}^{(i)} \leftarrow \boldsymbol{y}^{(i)} - \boldsymbol{q}^{j} \langle \boldsymbol{q}^{j}, \boldsymbol{y}^{(i)} \rangle$.
17:    $\boldsymbol{Q}^{(l)} = \boldsymbol{Q}^{(j)}$.
```

2.3.1. Numerical Study

In this section we carry our experiments partially following section 7.1. in [1]. The matrix $\boldsymbol{A}$ in (5) is used for testing the algorithm. First, we let the algorithm run a fixed l steps (the algorithm does not auto-stop) with $r = 5$. At each step, we record the following values:

1 The minimum rank-l approximation error σ_{l+1} ;

2 The actual error $e_l = \left\| (\boldsymbol{I} - \boldsymbol{Q}^{(l)}(\boldsymbol{Q}^{(l)})^T)\boldsymbol{A} \right\|$;

3 The error estimator f_l for the actual error e_l .

The first two are obtained using functions *svd* and *norm* in MATLAB. The third one is through the right hand side of (6). We set l to 80 and Figure 3 presents the data we record.

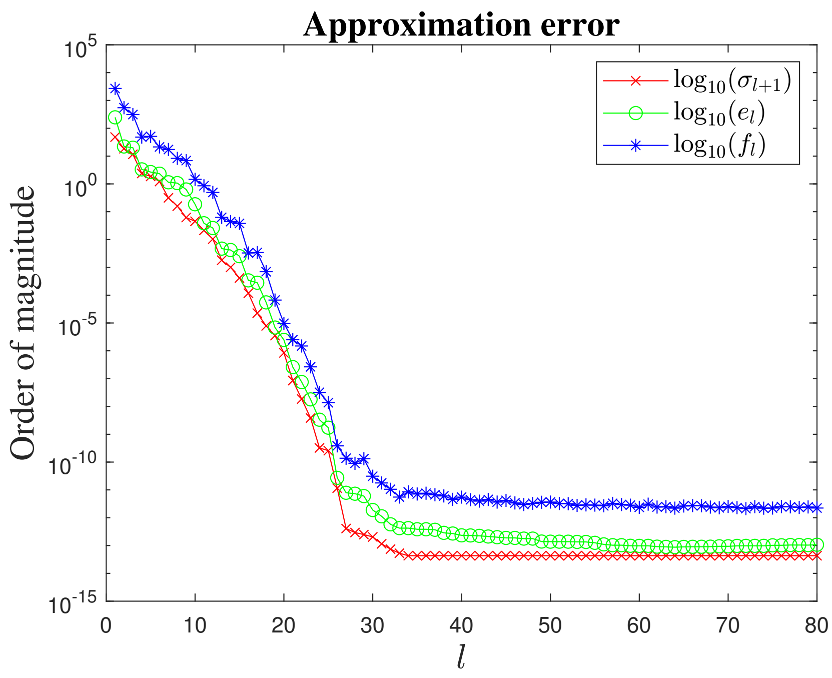


Figure 3: Optimum, real and approximated approximation errors recorded when extincting Algorithm 3 on the test matrix (5) for $l = 80$ steps

We make similar observations with [1]: 1) The real error in practice is very close to the theoretical optimum; 2) The error estimation is always larger than the real error. Roughly the estimated error is 10 times larger than the real error; 3) The basis constructed by the algorithm essentially reaches full double-precision accuracy. This indicates that in the algorithm one can use $\sqrt{\frac{2}{\pi}} \max_{i=1,\ldots,r} \left\| (\boldsymbol{I} - \boldsymbol{Q}\boldsymbol{Q}^{\boldsymbol{T}})\boldsymbol{A}\boldsymbol{\omega}^{(i)} \right\|$ as the error indicator (ten times smaller than before) to make more aggressive estimations and it will not hurt the accuracy too much.

To test the stability of this algorithm, we run the algorithm for 2000 independent trials and we record the real errors and the estimated errors at four points of running: $l = 20, 40, 60, 80$. We collect and present the results in Figure 4. It can be seen that what happens in Figure 3 is typically the general outcome: the error estimator is always pessimistic by a factor of about ten. No abnormal case ($f_l < e_l$) is observed. Both the actual and estimated error concentrate around their values (like a two-dimensional Gaussian distribution). Another observation is that as l increases, the real error slowly float away from the optimum value.

Remark1. The CPU time requirements of Algorithms 3 is essentially identical to that of Stage A in Algorithm 1. Although the last few samples computed by Algorithms 3 are only for obtaining the error estimation, this extra cost is cancelled out by the fact that in Algorithm 1 we always include an oversampling factor to gain a better approximation. Also the experimental results above demonstrate that the failure probability claimed for Algorithms 3 can actually be ignored. In practice, setting $r = 10$ or 5 is enough to obtain a reliable error estimator.

2.4. Stage B for a Positive Semidefinite Matrix

In this section we turn our focus to Stage B and study how to better the performance of randomized SVD for a positive semi-definite (PSD) matrix $\boldsymbol{A}$ by massaging Stage B. When the input matrix $\boldsymbol{A}$ is positive semi-definite, the well-known Nystrom method [7] can be used to improve the quality of standard SVD factorization at almost no additional cost [1].

The Nystrom method construct a more complex approximation as follows:

$$\begin{aligned}\boldsymbol{A} &\approx (\boldsymbol{AQ})(\boldsymbol{Q}^T\boldsymbol{AA})^{-1}(\boldsymbol{AQ})^T & (8)\\ &= \left[(\boldsymbol{AQ})(\boldsymbol{Q}^T\boldsymbol{AQ})^{(-1/2)}\right]\left[(\boldsymbol{AQ})(\boldsymbol{Q}^T\boldsymbol{AQ})^{(-1/2)}\right]^T = \boldsymbol{FF}^T. & (9)\end{aligned}$$

Here $\boldsymbol{F}$ is an approximate Cholesky factor of $\boldsymbol{A}$ with dimension $n \times k$. The formal calculation process is presented in Algorithm 4.

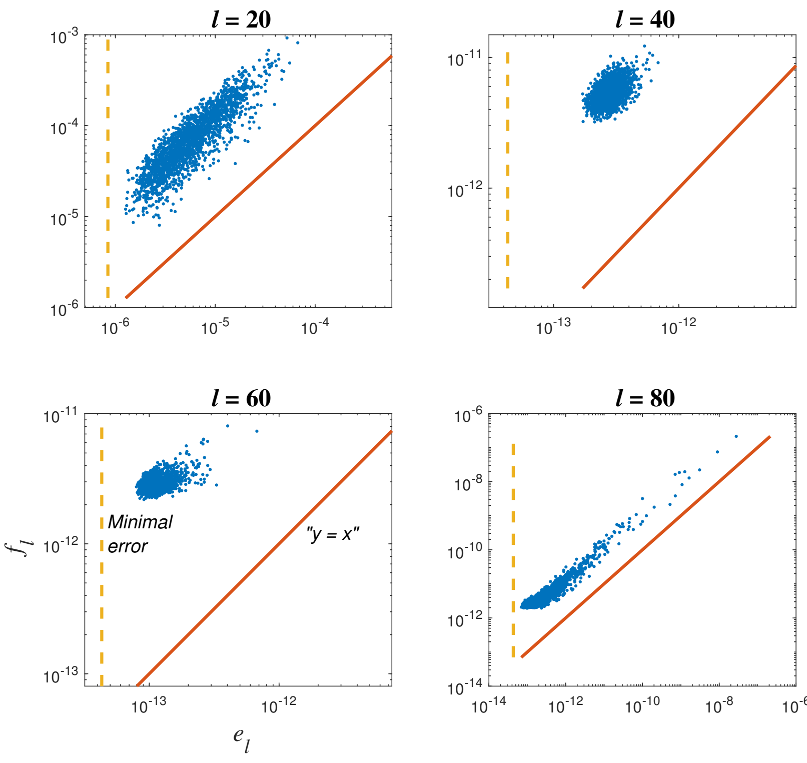


Figure 4: Comparison between the real errors and the estimated errors for in total 2000 trails. The panels isolate the moments at which $l = 20, 40, 60, 80$ random samples have been drawn. Each solid point compares the estimated error f_l versus the actual error e_l in one trial. The dashed line identifies the minimal error σ_{l+1}, and the solid line marks the contour where the error estimator would equal the actual error.

Algorithm 4 Randomized Subspace Iteration

```
1: procedure
2: Input: A PSD matrix $\boldsymbol{A}$ and a basis matrix $\boldsymbol{Q}$ such that (7) holds for some
   threshold $\epsilon$.
3: Output: An approximate SVD (or eigenvalue decomposition) $\widehat{\boldsymbol{U}}\widehat{\boldsymbol{\Sigma}}\widehat{\boldsymbol{U}}^{\boldsymbol{T}}$
   where $\widehat{\boldsymbol{U}}$ is orthonormal, and $\widehat{\boldsymbol{\Sigma}}$ is nonnegative and diagonal.
4:     Form the matrices $\boldsymbol{B}_1 = \boldsymbol{A}\boldsymbol{Q}$ and $\boldsymbol{B}_2 = \boldsymbol{Q}^T\boldsymbol{B}_1$.
5:     Perform a Cholesky factorization $\boldsymbol{B}_2 = \boldsymbol{C}^T\boldsymbol{C}$.
6:     Form $\boldsymbol{F} = \boldsymbol{B}_1\boldsymbol{C}^{(-1)}$ using a triangular solve.
7:     Compute a SVD $\boldsymbol{F} = \widehat{\boldsymbol{U}}\boldsymbol{\Sigma}\boldsymbol{V}^T$ and set $\widehat{\boldsymbol{\Sigma}} = \boldsymbol{\Sigma}^2$.
```

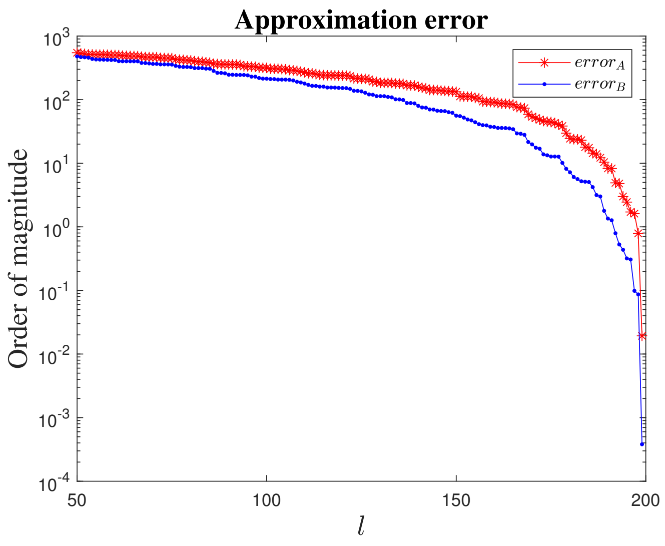
It is claimed that in the spectral norm, the Nystrom approximation error never exceed $\left\|(\boldsymbol{I} - \boldsymbol{Q}(\boldsymbol{Q})^T)\boldsymbol{A}\right\|$. It is also expected that the Nystrom approximation error is smaller than that of the standard randomized SVD, since that the upper bound for the standard randomized SVD is 2ϵ as claimed in [1]. We will demonstrate it is true in later experiments.

Remark2. In both cases, the dominant cost occurs when we form $\boldsymbol{A}\boldsymbol{Q}$, so the two procedures have roughly the same running time. On the other hand, the Nystrom approximation is typically much more accurate than the standard process. In a sense, we are exploiting the fact that $\boldsymbol{A}$ is positive semi-definite to take one step of subspace iteration (Algorithm 3) for free. [1]

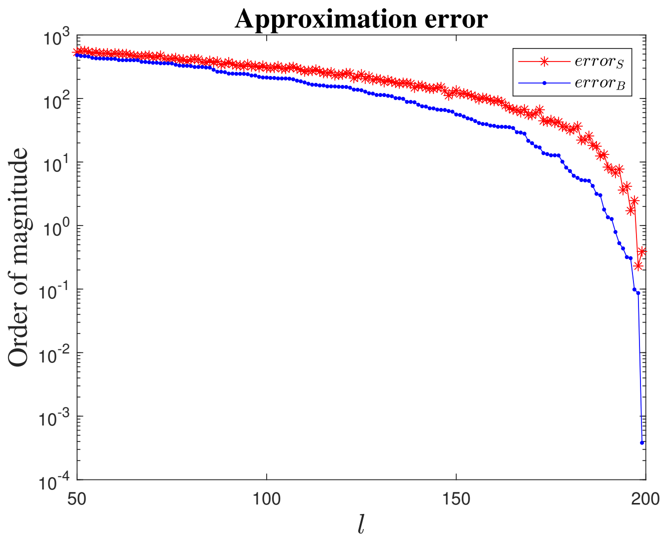
2.4.1. Numerical Study

In this section we examine the performance of the Nystrom approximation. We utilize the matrix from [2] which is denoted as $\boldsymbol{B}$ and form the test matrix $\boldsymbol{A} = \boldsymbol{B}\boldsymbol{B}^{\boldsymbol{T}}$ to make it PSD. Then we combine Algorithm 2 with $l = 50, \ldots, 200$ (fixed l value) and Algorithm 4 to obtain approximate SVD for $\boldsymbol{A}$. For each value of l, we record the following statistic: $error_A = \left\|(\boldsymbol{I} - \boldsymbol{Q}(\boldsymbol{Q})^T)\boldsymbol{A}\right\|$ which is the range approximation error in Stage A; $error_B = \left\|\widehat{\boldsymbol{U}}\widehat{\boldsymbol{\Sigma}}\widehat{\boldsymbol{U}}^{\boldsymbol{T}} - \boldsymbol{A}\right\|$ which is the final approximation error for the SVD (after Stage B); $error_s = \left\|\boldsymbol{A} - \widehat{\boldsymbol{U}}\widehat{\boldsymbol{\Sigma}}\widehat{\boldsymbol{V}}^{\boldsymbol{T}}\right\|$ which is obtained using the standard Algorithm 1. Figure 5 presented the collected results. It can be seen that $error_B$ ans $error_S$ are always smaller than

170 $error_A$ which is consistent with Remark 2.



(a) $error_A$ versus $error_B$



(b) $error_S$ versus $error_B$

Figure 5: Comparison between performance of standard randomize SVD and the Nystrom-approximation-equipped randomized SVD. Random samples are obtained using Algorithm 2.

3. Discussion and Future Work

In this project we numerically demonstrate the efficiency and stability of two randomized SVD algorithms, randomized subspace iteration method and adaptive randomized range finder both with standard Stage B. A special post-processing strategy for PSD matrices aiming at improving the accuracy is also discussed and validated empirically. Potential future works include: detailed discussion on their time-cost and storage-cost; post-processing techniques via row extraction and randomized tensor decomposition.

References

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

[2] E. Howard, Test matrix for hw5 from cmsc 763, fall 2019, `https://www.cs.umd.edu/users/elman/763.19/hw/hw5.mat`, 2019.

[3] G. Stewart, Accelerating the orthogonal iteration for the eigenvectors of a hermitian matrix, Numerische Mathematik 13 (4) (1969) 362–376.

[4] H. C. Elman, Q. Liao, Reduced basis collocation methods for partial differential equations with random coefficients, SIAM/ASA Journal on Uncertainty Quantification 1 (1) (2013) 192–217.

[5] H. C. Elman, A. Ramage, D. J. Silvester, Ifiss: A computational laboratory for investigating incompressible flow problems, SIAM Review 56 (2) (2014) 261–273.

[6] F. Woolfe, E. Liberty, V. Rokhlin, M. Tygert, A fast randomized algorithm for the approximation of matrices, Applied and Computational Harmonic Analysis 25 (3) (2008) 335–366.

[7] P. Drineas, M. W. Mahoney, On the nyström method for approximating a gram matrix for improved kernel-based learning, journal of machine learning research 6 (Dec) (2005) 2153–2175.