Spectrum-Revealing Randomized Matrix Factorization: Theory and Algorithms

by

Jianwei Xiao

A dissertation submitted in partial satisfaction of the

requirements for the degree of

Doctor of Philosophy

in

Applied Mathematics

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Professor Ming Gu, Chair
Professor James Demmel
Professor John Strain
Professor Laurent El Ghaoui

Fall 2018

Spectrum-Revealing Randomized Matrix Factorization: Theory and Algorithms

Abstract

Spectrum-Revealing Randomized Matrix Factorization: Theory and Algorithms

by

Jianwei Xiao

Doctor of Philosophy in Applied Mathematics

University of California, Berkeley

Professor Ming Gu, Chair

This thesis contains my work on Spectrum-revealing randomized matrix algorithms. This thesis has been divided into three chapters. Each chapter is self-contained.

In chapter 1, we discuss spectrum-revealing Cholesky factorization and its applications to kernel methods. Kernel methods represent some of the most popular machine learning tools for data analysis. Since exact kernel methods can be prohibitively expensive for large problems, reliable low-rank matrix approximations and high-performance implementations have become indispensable for practical applications of kernel methods. We introduce spectrum-revealing Cholesky factorization, a reliable low-rank matrix factorization, for kernel matrix approximation. We also develop an efficient and effective randomized algorithm for computing this factorization. Our numerical experiments demonstrate that this algorithm is as effective as other Cholesky factorization based kernel methods on machine learning problems but significantly faster.

In chapter 2, we discuss spectrum-revealing QR factorization and also distributed memory implementation of randomized QRCP. Factorizing large matrices by QR with column pivoting (QRCP) is substantially more expensive than QR without pivoting, owing to communication costs required for pivoting decisions. In contrast, randomized QRCP (RQRCP) algorithms have proven themselves empirically to be highly competitive with high-performance implementations of QR in processing time, on uniprocessor and shared memory machines, and as reliable as QRCP in pivot quality. We show that RQRCP algorithms can be as reliable as QRCP with failure probabilities exponentially decaying in oversampling size. We also analyze efficiency differences among different RQRCP algorithms. More importantly, we develop distributed memory implementations of RQRCP that are significantly better than QRCP implementations in ScaLAPACK. As a further development, we introduce the concept of and develop algorithms for computing spectrum-revealing QR factorizations for low-rank matrix approximations, and demonstrate their effectiveness against leading low-rank approximation methods in both theoretical and numerical reliability and efficiency.

In chapter 3, we present Flip-Flop Spectrum-Revealing QR (Flip-Flop SRQR) factorization, a significantly faster and more reliable variant of the QLP factorization of Stewart, for

low-rank matrix approximations. Flip-Flop SRQR uses SRQR factorization to initialize a partial column pivoted QR factorization and then compute a partial LQ factorization. As observed by Stewart in his original QLP work, Flip-Flop SRQR tracks the exact singular values with "considerable fidelity". We develop singular value lower bounds and residual error upper bounds for Flip-Flop SRQR factorization. In situations where singular values of the input matrix decay relatively quickly, the low-rank approximation computed by Flip-Flop SRQR is guaranteed to be as accurate as truncated SVD. We also perform a complexity analysis to show that for the same accuracy, Flip-Flop SRQR is faster than randomized subspace iteration for approximating the SVD, the standard method used in Matlab tensor toolbox. We additionally compare Flip-Flop SRQR with alternatives on two applications, tensor approximation and nuclear norm minimization, to demonstrate its efficiency and effectiveness.

Contents

List of Figures

List of Tables

Acknowledgments

I would like to thank my advisor Prof. Ming Gu for motivating and encouraging me to work on these fascinating research topics. He is not only an outstanding advisor who taught me a lot of matrix techniques, but also a good friend that I can frankly share my thoughts. It has been a pleasant experience to work with him.

Appreciation goes to my other committee members as well. Prof. James Demmel introduced me to the fascinating world of parallel computing. Prof. John Strain taught me matrix computations and led me to choose numerical linear algebra as my Ph.D. research area. His incredible sense of humor while teaching impressed me. Prof. Laurent El Ghaoui taught me convex optimization which is full of amazing facts and useful techniques.

I am fortunate to have benefited from numerous people. I would like to thank Yuehua Feng for many helpful discussions during her stay at UC Berkeley. Her passionate attitude towards research has indeed motivated me. I also would like to thank my classmates Qiaochu Yuan and Ruochen Liang for their helpful career advice. I would like to thank Prof. Julien Langou for pointing out the improvement technique we can use when I wrote the distributed randomized QRCP implementation. I also would like to thank my friends Zeming Wang and Zheng Cai for their support these years.

Last but not least, I would like to express my deepest gratitude to my parents. This dissertation would not have been possible without their warmest love, continued patience, and endless support.

Chapter 1

Spectrum-Revealing Cholesky Factorization

1.1 Introduction

1.1.1 Kernel Methods and Their Practical Performance

Kernel methods have become increasingly popular tools for machine learning tasks such as classification, prediction, novelty detection, and clustering, with diverse applications including inverse distance weighting, 3D image reconstruction, information extraction, and handwriting recognition. [1]

Kernel methods owe their names to the use of positive definite kernel functions, which enable them to operate in a high-dimensional feature space, typically defined through the inner products between the images of all pairs of data points in the feature space. However, one critical drawback of kernel methods is their inability to solve very large-scale learning problems owing to their computational complexity. Given a set of n data points, the kernel matrix K is of size $n \times n$, which means a computational complexity of at least $\Omega(n^2)$. More importantly, most kernel methods require matrix inversions or eigenvalue decompositions as their computational cores, leading to complexities as high as $O(n^3)$.

There are two major approaches to significantly improving the practical performance of kernel methods. Firstly, the aforementioned complexities can typically be reduced by approximating the kernel matrices with their low-rank approximations, and this is one of the major reasons for the practical success of kernel methods. The underlying intuition is that the kernel matrices usually have rapidly decaying singular value spectra and thus have relatively small numerical ranks [117]. The kernel matrix K can be approximated in the form

$$K \approx LL^T, \quad \text{where} \quad L \in \mathbb{R}^{n \times k}, \tag{1.1}$$

[1] Materials in this chapter are mainly from the paper titled Spectrum-Revealing Cholesky Factorization for Kernel Methods [120], which was published in 2016 IEEE 16th International Conference on Data Mining.

and the approximate rank k is generally chosen so that $k \ll n$. Moreover, it is often possible to reformulate kernel methods to make direct use of the matrix L instead of K. This can result in learning methods of a much lowered computational complexity of $O(k^3 + k^2 n)$, i.e., linear in n [40, 5].

Another approach is to develop highly-tuned software libraries for major machine learning algorithms for kernel methods [18, 88, 83]. By re-organizing their internal computations to take maximum advantage of high-performance linear algebra packages such as BLAS [72, 10] and LAPACK [2], significant practical speedups can be produced without major mathematical changes to the algorithms themselves.

1.1.2 Cholesky Factorization Based Low-rank Approximation

One of the most popular methods to obtain a low-rank approximation of a kernel matrix in the form (1.1) is based on the Cholesky factorization, due to its relative simplicity and computational efficiency. Such approximations have been used in many areas: SVM training [40], kernel independent component analysis [5], predictive low-rank decomposition for kernel methods [6], and computation of the two-electron integral matrix [54]. The essential part in finding a low-rank approximation of kernel matrix using Cholesky factorization is to find representative training samples, which is equivalent to doing Cholesky factorization of kernel matrix with certain pivoting strategy in numerical linear algebra. Diagonal pivoting is commonly used in these Cholesky factorization based algorithms.

However, there are two major well-known drawbacks of this diagonal pivoting strategy. Firstly, pivots (representative training samples) are computed one at a time, which results in mostly level-2 BLAS operations (matrix-vector multiplications), much less efficient than level-3 BLAS operations (matrix-matrix multiplications). Secondly, the pivots chosen by diagonal pivoting may occasionally fail to produce a reliable low-rank approximation to the original kernel matrix.

1.1.3 Randomized Cholesky for Reliable Low-rank Approximation

In recent works, randomization has emerged as an especially potent tool for large-scale data analysis. Reliable and efficient randomized algorithms have been successfully developed for low-rank approximation of general matrices [42, 52, 48], sketching problems [30, 119, 26], and fast solution to the least squares problem [4, 31, 19]. [81] is a good review paper of randomized algorithms for matrices.

In this chapter, we develop a randomized blocked Cholesky factorization algorithm for reliable low-rank matrix approximation of the kernel matrix. This algorithm is run in two stages. In the first stage, we first use randomization to project the original kernel matrix into a matrix of much smaller dimension; we then find pivots on the smaller-dimensioned matrix, and finally, we apply these pivots to the kernel matrix in a block form to fully take advantage

of level-3 BLAS performance. We repeat this process until we reach the approximate rank k. Despite the inherent randomness, this approach works very well in practical applications.

However, every now and then the approach above may not lead to reliable low-rank approximations. To guard against this possibility, in the second stage we further employ a novel follow-up pivoting strategy that simultaneously ensures a reliable low-rank approximation and separates the representative training samples from those that are collinear to them. This separation feature of our pivoting strategy is of significant interest in its own right in some applications [118, 53]. We will establish strong singular value and matrix error bounds to demonstrate the effectiveness of this pivoting strategy.

Although the computational complexity of our algorithm is no longer linear in n, the implementation of our algorithm is still faster than other Cholesky factorization based algorithms. The main reason is that runtime of an algorithm is not only dependent on arithmetic cost but also dependent on communication cost, which represents the time (or energy) of moving data, either between levels of a memory hierarchy or between processors over a network. The communication cost of an algorithm often dominates arithmetic cost, and technological trends indicate this cost gap will continue to increase in the future as new platforms become available. Level-3 BLAS operations have significantly lower communication cost than level-2 BLAS operations. As a blocked algorithm, our novel method fully utilizes level-3 BLAS. As our empirical experiments demonstrate, while our method is at least as reliable as other Cholesky factorization based algorithms in all applications, it is much faster for large-scale low-rank approximations.

1.1.4 Our Contributions and Chapter Outline

- **Spectrum-revealing Cholesky factorization:** We demonstrate the existence of the Spectrum-revealing Cholesky factorization (SRCH) and develop strong singular value and matrix error bounds for SRCH. Our analysis shows that SRCH provides a highly reliable low-rank approximation to the kernel matrix for any given approximate rank k.

- **A randomized algorithm for computing an SRCH:** Unlike existing Cholesky factorization algorithms, this randomized algorithm is efficient and correctly computes an SRCH. It is especially suitable to obtain quality low-rank approximations for matrices with fast decaying singular-value spectra, which are ubiquitous in kernel matrices in machine learning.

- **Empirical validation:** We compare our method with other Cholesky factorization based algorithms in two different applications: a prediction problem and the Gaussian process. All of these methods show similar effectiveness while our method is significantly faster.

In Section 1.2 we briefly introduce previous work of pivoted Cholesky. In Section 1.3.1 we develop a randomized blocked left-looking algorithm to compute a pivoted Cholesky

Matlab notation for $1 \leq i \leq p \leq m$ and $1 \leq j \leq q \leq n$		
Notation	**Dimensions**	**Description**
$A(:,:)$	$\mathbb{R}^{m \times n}$	entire matrix A
$A(i,:)$	row in $\mathbb{R}^n$	ith row of A
$A(:,j)$	column in $\mathbb{R}^m$	jth column of A
$A(i, j:q)$	row in $\mathbb{R}^{q-j+1}$	jth through qth entries of ith row of A
$A(i:p, j)$	column in $\mathbb{R}^{p-i+1}$	ith through pth entries of jth column of A
$A(i:p, j:q)$	$\mathbb{R}^{(p-i+1)\times(q-j+1)}$	submatrix from intersection of ith through pth rows and jth through qth columns of A

Table 1.1: Matlab notations.

factorization without explicitly updating the Schur complement. In Section 1.3.2 we define and discuss properties of an SRCH, and develop an efficient modification to the algorithm in Section 1.3.1 to reliably compute an SRCH. In Section 1.4 we compare our algorithm and other alternatives in different applications. In Section 1.5 we introduce results of incomplete randomized Cholesky factorization.

1.2 The Setup and Background

1.2.1 Notation

$\sigma_j(A)$ denotes the jth largest singular value of A. If A is real symmetric, $\lambda_j(A)$ denotes its jth largest eigenvalue. $\Pi_{i,j}$ denotes the permutation matrix that interchanges the ith and jth columns of the identity matrix. $\mathbf{diag}(A)$ is the vector of the main diagonal elements of A. $\mathbf{diag}(v)$ is the square diagonal matrix with the elements of vector v on the main diagonal. In this chapter we follow Matlab notation, summarized in Table 1.1.

1.2.2 Diagonal Pivoted Cholesky Factorization

Diagonal pivoting is the most popular pivoting strategy in computing a Cholesky factorization for low-rank approximation. This strategy simply chooses the largest diagonal entry as the pivot at every pivoting step within the Cholesky factorization process. Lucas [79] developed an LAPACK-style code, double precision subroutine `DPSTRF.f`, for the diagonal pivoted Cholesky factorization of a symmetric positive semidefinite matrix. In order to be able to compare our new algorithm with `DPSTRF.f`, we modified its stopping criterion from error tolerance to approximate rank and present it as Algorithm 1. The difference $D - W$ stores the diagonal elements of Schur complement, and the function $\mathbf{swap}(x, y)$ swaps entries in x and y. Algorithm 1 performs blocked right-looking diagonal pivoted Cholesky factorization, and returns $L\,L^T$ as a low-rank approximation of A under permutation Π.

Algorithm 1 `DPSTRF.f`. Blocked Right-Looking Diagonal Pivoted Cholesky Factorization

Inputs:

Positive semidefinite $A \in \mathbb{R}^{n \times n}$. Block size b. Approximate rank k.

Outputs:

Permutation vector $\Pi \in \mathbb{R}^n$. $L \stackrel{def}{=}$ lower triangular part of $A(1:n, 1:k)$.

Algorithm:

$\Pi = (1:n) \in \mathbb{R}^n, D = \mathbf{diag}(A) \in \mathbb{R}^n$, $W \in \mathbb{R}^n, W(1:n) = 0$.

for $i = 1 : b : k$ **do**

  $\overline{b} = \mathbf{min}(b, k - i + 1)$

  for $j = i : i + \overline{b} - 1$ **do**

    for $l = j : n$ **do**

      $W(l) \;=\; W(l) \;+\; A(l, j-1) \cdot A(l, j-1)$

    end for

    $q = \mathbf{argmax}_{j \le s \le n}\{D(s) - W(s)\}$

    swap $([A(j:n, j), A(j, 1:n)], [A(j:n, q), A(q, 1:n)])$

    swap $([D(j), W(j), \Pi(j)], [D(q), W(q), \Pi(q)])$

    $A(j, j) = \sqrt{D(q) - W(q)}$

    $A(j+1:n, j) \;=\; A(j+1:n, j) - A(j+1:n, i:j-1)\; A(j, i:j-1)^T$

    $A(j+1:n, j) \;=\; A(j+1:n, j) \;/\; A(j, j)$

  end for

$$A(i+\overline{b}:n, i+\overline{b}:n) \mathrel{-}= A(i+\overline{b}:n, i:i+\overline{b}-1)\; A(i+\overline{b}:n, i:i+\overline{b}-1)^T \tag{1.2}$$

end for

1.3 New Algorithms and Main Results

There are two major problems with Algorithm 1. Firstly, most of its work is in updating the Schur complement, the matrix $A(i+\overline{b}:n, i+\overline{b}:n)$ in (1.2). However, the Schur complement on exit from Algorithm 1 is typically discarded in a low-rank matrix approximation, meaning most of this work is unnecessary if $k \ll n$. Secondly, Algorithm 1 is a greedy algorithm for computing a low-rank approximation by pivoting to the largest diagonal element in every Schur complement. There are well-known classes of matrices for which this strategy fails to compute a reliable low-rank approximation [57, 59, 58]. [50] provides an algorithm that can always compute a reliable low-rank approximation by doing suitable columns and rows swaps after obtaining a partial Cholesky factorization with a certain pivoting strategy like diagonal pivoting.

In Section 1.3.1 we develop a randomized blocked left-looking algorithm to compute a pivoted Cholesky factorization without explicitly updating the Schur complement. In Section 1.3.2 we define and discuss the desired properties of an SRCH, and develop an efficient modification to the algorithm in Section 1.3.1 to reliably compute an SRCH.

1.3.1 A Randomized Blocked Left-Looking Cholesky Factorization

The Cholesky factorization can be computed in a number of different, but mathematically equivalent, variants. Algorithm 2 is a left-looking variant that computes the full Cholesky factorization without directly updating the Schur complement. For a symmetric positive definite X, $\mathbf{chol}(X)$ is the Cholesky factor such that $(\mathbf{chol}(X))\,(\mathbf{chol}(X))^T = X$.

Algorithm 2 Blocked Left-Looking Cholesky Factorization

Inputs:
Positive semidefinite $A \in \mathbb{R}^{n\times n}$. Block size b.
Outputs:
$L \stackrel{def}{=}$ lower triangular part of A.
Algorithm:
for $j = 1 : b : n$ **do**

$$\begin{aligned}
&\overline{b} = \mathbf{min}(b, n-j+1)\\
&A(j:n, j:j+\overline{b}-1) \mathrel{-}= A(j:n, 1:j-1)A(j:j+\overline{b}-1, 1:j-1)^T \qquad (1.3)\\
&A(j:j+\overline{b}-1, j:j+\overline{b}-1) = \mathbf{chol}(A(j:j+\overline{b}-1, j:j+\overline{b}-1))\\
&A(j+\overline{b}:n, j:j+\overline{b}-1) = A(j+\overline{b}:n, j:j+\overline{b}-1)A(j:j+\overline{b}-1, j:j+\overline{b}-1)^{-T}
\end{aligned}$$

end for

Unlike Algorithm 1, most of the work in Algorithm 2 is in updating the matrix $A(j:n, j:j+\overline{b}-1)$ in (1.3). This work starts small but increases linearly with j. Thus, Algorithm 2 would be much faster than Algorithm 1 if we restricted $j \le k$ for some $k \ll n$. But such restriction, without the benefit of any pivoting strategy, might not lead to a very meaningful approximation of the matrix A.

Based on this consideration, we now introduce a novel pivoting strategy into Algorithm 2. For a given small integer p and $A \in \mathbb{R}^{n\times n}$, we generate a random matrix $\Omega \in \mathcal{N}(0,1)^{(b+p)\times n}$ where the entries are independent and identically distributed (i.i.d.) random variables drawn from standard normal distribution $\mathcal{N}(0,1)$. We compute a random projection $B = \Omega A \in \mathbb{R}^{(b+p)\times n}$, which is significantly smaller than A in row dimension if $b+p \ll n$. We compute a partial QR factorization with column pivoting (QRCP) [13, 46] on B to obtain b pivot columns and apply them as b diagonal pivots on A. Intuitively, good pivots for B should also be good pivots for A. For this strategy to work, we need to compute a random projection for the Schur complement $A(j+\overline{b}:n, j+\overline{b}:n)$ for each j, without explicitly computing $A(j+\overline{b}:n, j+\overline{b}:n)$. Remarkably, such a random projection can indeed be quickly computed via an updating formula. Algorithm 3 computes a partial Cholesky factorization, with diagonal pivots chosen by partial QRCP on successive random projections.

Most work in Algorithm 3 is done in (1.4), which increases in j. On the other hand, most work in Algorithm 1 is done in (1.2), which decreases in $n-i$. Thus, Algorithm 3 is much

Algorithm 3 Randomized Blocked Left-Looking Cholesky Factorization

Inputs:

Positive semidefinite $A \in \mathbb{R}^{n\times n}$. Block size b. Oversampling size p. Approximate rank $k \ll n$.

Outputs:

Permutation vector $\Pi \in \mathbb{R}^n$. $L \stackrel{def}{=}$ lower triangular part of $A(1:n, 1:k)$.

Algorithm:

Generate $\Omega \in \mathcal{N}(0,1)^{(b+p)\times n}$; compute $B = \Omega A$; initialize $\Pi = (1:n) \in \mathbb{R}^n$.

for $j = 1 : b : k$ **do**

$\overline{b} = \mathbf{min}(b, k - j + 1)$

Compute partial QRCP on $B(:, j:n)$ to obtain $\overline{b}$ column pivots $(j_1, j_2, \ldots, j_{\overline{b}})$

Swap $(j_1, j_2, \ldots, j_{\overline{b}})$ and $(j, j+1, \ldots, j+\overline{b}-1)$ columns in B, Ω, and entries in Π

Swap corresponding rows and columns in A

$$\begin{aligned} A(j:n, j:j+\overline{b}-1) &\text{-}= A(j:n, 1:j-1)A(j:j+\overline{b}-1, 1:j-1)^T \\ A(j:j+\overline{b}-1, j:j+\overline{b}-1) &= \mathbf{chol}(A(j:j+\overline{b}-1, j:j+\overline{b}-1)) \\ A(j+\overline{b}:n, j:j+\overline{b}-1) &= A(j+\overline{b}:n, j:j+\overline{b}-1)A(j:j+\overline{b}-1, j:j+\overline{b}-1)^{-T} \end{aligned} \tag{1.4}$$

if $j + \overline{b} - 1 < k$ **then**

$B(:, j+\overline{b}:n)$ -= $\Omega(:, j:n)\ A(j:n, j:j+\overline{b}-1)\ A(j+\overline{b}:n, j:j+\overline{b}-1)^T$

end if

end for

more efficient than Algorithm 1 when $k \ll n$. Numerical experiments suggest that Algorithm 3 typically computes a better low-rank approximation than Algorithm 1. However, for a full Cholesky factorization (i.e., $k = n$), Algorithm 1 is still better.

Updating formula for B: The formula for successively computing $B(:, j+b:n)$ for increasing j in Algorithm 3 is what makes Algorithm 3 so efficient. To derive it, we first compute the random projection $B = \Omega A$. Algorithm 3 then computes b pivots based on a partial QRCP on B and performs a block Cholesky step. To continue, Algorithm 3 will need to compute a random projection on the corresponding Schur complement, which it does not directly compute. It turns out that we can re-use the initial random matrix Ω and the corresponding random projection B to compute a special random projection for the Schur complement.

After the necessary row and column swaps and the block Cholesky step, we can write the matrices A, and Ω as $\begin{pmatrix} L_{11} & \\ L_{21} & I \end{pmatrix}\begin{pmatrix} L_{11}^T & L_{21}^T \\ & S_2 \end{pmatrix}$, and $\begin{pmatrix} \Omega_1 & \Omega_2 \end{pmatrix}$, respectively, where S_2 is the aforementioned Schur complement. The column swapped B can be written as

$$\begin{pmatrix} B_1 & B_2 \end{pmatrix} = \begin{pmatrix} \Omega_1 & \Omega_2 \end{pmatrix}\begin{pmatrix} L_{11} & \\ L_{21} & I \end{pmatrix}\begin{pmatrix} L_{11}^T & L_{21}^T \\ & S_2 \end{pmatrix},$$

which, in turn, implies a special random projection formula

$$\Omega_2\, S_2 = B_2 - \begin{pmatrix} \Omega_1 & \Omega_2 \end{pmatrix} \begin{pmatrix} L_{11} \\ L_{21} \end{pmatrix} L_{21}^T. \tag{1.5}$$

In plain English, with Ω_2 as the random matrix, we can compute the random projection $\Omega_2\, S_2$ for S_2 via the right hand side expression in (1.5). Generalizing this consideration for all j results in the formula for computing $B(:, j+b:n)$ in Algorithm 3.

Complexity analysis: The most computationally intensive parts of Algorithm 3 are computing the initial random projection matrix $B = \Omega A$ and updating pivoted block $A(j : n, j+b-1)$ for each j. Computing B requires $O(n^2(b+p))$ operations and updating all pivoted blocks requires $O(nk^2)$ operations, leading to the overall complexity of $O(n^2(b+p)+nk^2)$ operations. It is interesting to note that if both $p \ll n$ and $k \ll n$, then the complexity is $O(n^2(b+p))$, i.e., the dominant computation is in the overhead – computing the random projection matrix. Of course, if the matrix A is itself sparse then the overhead cost will be much lower.

1.3.2 Spectrum-Revealing Cholesky Factorization

Greedy pivoting strategies in Algorithm 1 and Algorithm 3 typically compute good quality low-rank approximations, but not always. Below we first discuss what low-rank approximations are possible based on diagonal pivoting alone, and then develop a swap strategy to modify the truncated Cholesky factorization computed from Algorithm 3 to ensure such an approximation. With a slight abuse of notation, Π in Theorem 1 denotes a permutation matrix. Recall that for any matrix X, $||X||_{2,1}$ is equal to the largest of the column l_2 norms of X, and $||X||_{\max}$ is equal to the largest entry of X in absolute value.

Theorem 1. *Let $A \in \mathbb{R}^{n\times n}$ be symmetric positive definite with a partial Cholesky factorization for $k < n$:*

$$\Pi^T A \Pi = LL^T + \begin{pmatrix} 0 & 0 \\ 0 & S \end{pmatrix}, \tag{1.6}$$

$$\textit{with} \quad L = \begin{pmatrix} L_{11} \\ l^T \\ L_{21} \end{pmatrix}, \quad \textit{and} \quad S = \begin{pmatrix} \alpha & s^T \\ s & \widehat{S} \end{pmatrix},$$

where $L_{11} \in \mathbb{R}^{k\times k}, l \in \mathbb{R}^{k\times 1}, L_{21} \in \mathbb{R}^{(n-k-1)\times k}, \alpha \in \mathbb{R}, s \in \mathbb{R}^{(n-k-1)\times 1}$, $\widehat{S} \in \mathbb{R}^{(n-k-1)\times(n-k-1)}$.

Assume that $\alpha = ||S||_{max}$ and that for a given $g > 1$,

$$\frac{1}{\sqrt{\alpha}} \geq \frac{1}{\sqrt{g}} \left|\left| \begin{pmatrix} L_{11} & \\ l^T & \sqrt{\alpha} \end{pmatrix}^{-1} \right|\right|_{2,1}, \tag{1.7}$$

then there exists a $\tau \leq g(n-k)(k+1)$ *such that for* $1 \leq j \leq k$,

$$\lambda_{k+1}(A) \leq \left\|\Pi^T A \Pi - L\,L^T\right\|_2 \leq \tau\ \lambda_{k+1}(A), \tag{1.8}$$

$$\lambda_j(A) \geq \sigma_j^2(L) \geq \frac{\lambda_j(A)}{1+\tau\ \boldsymbol{min}\left\{1, (1+\tau)\dfrac{\lambda_{k+1}(A)}{\lambda_j(A)}\right\}}. \tag{1.9}$$

Proof of Theorem 1.

$$\begin{aligned}\lambda_j(A) &= \lambda_j\left(\Pi^T A \Pi\right) = \lambda_j\left(LL^T + \begin{pmatrix} 0 & 0 \\ 0 & S \end{pmatrix}\right) \\ &\leq \lambda_j\left(LL^T\right) + \|S\|_2 = \sigma_j^2(L)\left(1 + \frac{\|S\|_2}{\sigma_j^2(L)}\right),\end{aligned} \tag{1.10}$$

where we have used the fact that $\lambda_j\left(LL^T\right) = \sigma_j^2(L)$.

Since α is the largest diagonal entry of the Schur complement S and $\alpha = ||S||_{\max}$,

$$||S||_2 \leq ||S||_F \leq trace(S) \leq (n-k)\alpha. \tag{1.11}$$

On the other hand,

$$\begin{aligned}\frac{1}{\sqrt{\alpha}} &\geq \frac{1}{\sqrt{g}}\left\|\begin{pmatrix} L_{11} & \\ l^T & \sqrt{\alpha} \end{pmatrix}^{-1}\right\|_{2,1} \\ &\geq \frac{1}{\sqrt{g}}\frac{1}{\sqrt{k+1}}\left\|\begin{pmatrix} L_{11} & \\ l^T & \sqrt{\alpha} \end{pmatrix}^{-1}\right\|_F \\ &\geq \frac{1}{\sqrt{g}}\frac{1}{\sqrt{k+1}}\left(\sigma_{k+1}\begin{pmatrix} L_{11} & \\ l^T & \sqrt{\alpha} \end{pmatrix}\right)^{-1},\end{aligned} \tag{1.12}$$

therefore

$$\begin{aligned}\alpha &\leq g(k+1)\sigma_{k+1}^2\begin{pmatrix} L_{11} & \\ l^T & \sqrt{\alpha} \end{pmatrix} \\ &= g(k+1)\lambda_{k+1}\begin{pmatrix} L_{11}L_{11}^T & L_{11}l \\ l^T L_{11}^T & l^T l + \alpha \end{pmatrix} \\ &\leq g(k+1)\lambda_{k+1}(A). \quad \text{(Cauchy interlacing theorem)}\end{aligned}$$

Combining the last inequality with relation (1.11),

$$\|S\|_2 \leq (n-k)\alpha \leq \tau\lambda_{k+1}(A), \tag{1.13}$$

which is the desired matrix error upper bound in relation (1.8).

For the singular value lower bound (1.9), relation (1.12) implies

$$\frac{1}{\sqrt{\alpha}} \geq \frac{1}{\sqrt{g}} \frac{1}{\sqrt{k+1}} \left\| L_{11}^{-1} \right\|_F \geq \frac{1}{\sqrt{g}} \frac{1}{\sqrt{k+1}} (\sigma_k(L_{11}))^{-1},$$

$$\text{or} \qquad \alpha \leq g(k+1)\, \sigma_k^2(L_{11}).$$

Together with relation (1.11),

$$\|S\|_2 \leq g(k+1)(n-k)\, \sigma_k^2(L_{11}) = \tau\, \sigma_k^2(L_{11}). \tag{1.14}$$

Now combine relations (1.10) and (1.14)

$$\lambda_j(A) \leq \sigma_j^2(L) \left(1 + \frac{\tau \sigma_k^2(L_{11})}{\sigma_j^2(L)} \right) \leq \sigma_j^2(L)(1+\tau). \tag{1.15}$$

Rewrite relation (1.10) as

$$\lambda_j(A) \leq \sigma_j^2(L) \left(1 + \frac{\|S\|_2}{\lambda_j(A)} \frac{\lambda_j(A)}{\sigma_j^2(L)} \right).$$

Replacing $\|S\|_2$ in the first ratio by its upper bound (1.13), and $\sigma_j^2(L)$ in the second ratio by its lower bound (1.15),

$$\lambda_j(A) \leq \sigma_j^2(L) \left(1 + \tau(1+\tau) \frac{\lambda_{k+1}(A)}{\lambda_j(A)} \right).$$

In view of relation (1.15), we finally have

$$\sigma_j^2(L) \geq \frac{\lambda_j(A)}{1 + \tau \mathbf{min} \left\{ 1, (1+\tau) \frac{\lambda_{k+1}(A)}{\lambda_j(A)} \right\}},$$

which is relation (1.9). □

If conditions on α hold, then Theorem 1 asserts that the matrix approximation error is at most a factor of τ away from being optimal in 2-norm; and all the singular values of L are at most a factor of $\sqrt{1+\tau}$ away from being optimal. In addition, for the largest singular values of A where $\dfrac{\lambda_{k+1}(A)}{\lambda_j(A)} \ll 1$, the corresponding approximate singular values $\sigma_j(L)$ are very close to $\sqrt{\lambda_j(A)}$, the best possible for any rank-k approximation.

A partial Cholesky factorization of the form (1.6) is said to be *Spectrum-revealing* if it satisfies the conditions on α in Theorem 1. The singular value lower bound in relation (1.9) represents a unique feature in an SRCH.

Algorithm 1 ensures the condition $\alpha = ||S||_{\max}$ by performing diagonal pivoting, but may not satisfy condition (1.7), leading to potentially poor approximations.

Algorithm 4 is a randomized algorithm that efficiently computes a Spectrum-revealing Cholesky factorization. It initializes the permutation with Algorithm 3. If Algorithm 3 fails condition (1.7), Algorithm 4 makes additional randomized column and row swaps to ensure it. In the algorithm, we denote $\widehat{L} = \begin{pmatrix} L_{11} & \\ l^T & \sqrt{\alpha} \end{pmatrix}$, with $L = \begin{pmatrix} L_{11} \\ l^T \\ L_{21} \end{pmatrix} \stackrel{def}{=}$ lower triangular part of $A(1:n, 1:k)$.

Algorithm 4 A Randomized Algorithm to Compute an SRCH

Inputs:
Positive semidefinite $A \in \mathbb{R}^{n\times n}$. Block size b. Oversampling size p. Parameter $g > 1$. Approximate rank $k \ll n$.
Outputs:
Permutation vector $\Pi \in \mathbb{R}^n$. Matrix L.
Algorithm:
Compute Π and L with Algorithm 3
Generate random matrix $\Omega \in \mathcal{N}(0,1)^{d\times(k+1)}$
Compute $\alpha = \mathbf{max}(\text{diag(Schur complement)})$
while $\frac{1}{\sqrt{\alpha}} < \frac{1}{\sqrt{gd}} \left\|\Omega\widehat{L}^{-1}\right\|_{2,1}$ **do**
  $\imath = \mathbf{argmax}_{1\le i\le k+1}\{i\text{th column } l_2 \text{ norm of } \Omega\widehat{L}^{-1}\}$
  Swap $\imath$-th and $(k+1)$-st columns and rows of A
  Swap $\imath$-th and $(k+1)$-st entries in Π
  Givens-rotate L back into lower-triangular form.
  Compute $\alpha = \mathbf{max}(\text{diag(Schur complement)})$
end while

Remarks:

1. We only need to look through the diagonal entries of the Schur complement to find α, thereby avoid computing the Schur complement itself.

2. The while loop in Algorithm 4 will eventually stop, after a finite number of swaps, leading to a permutation that satisfies condition (1.7). The number of swaps is bounded above by $O(n)$, but in practice at most a few swaps are enough. See the proof of correctness of Algorithm 4 below.

3. Each swap will make the $\imath$-th row out of the lower-triangular form. A round-robin rotation is applied to the rows of L and a quick sequence of Givens rotations are right multiplied to L to restore its lower-triangular form. These Givens rotations are orthogonal and will cancel themselves out of the matrix product $L\,L^T$.

4. In some practical applications where more accurate singular values are desirable, one can compute an SRCH for a rank-$\widehat{k}$ approximation $\widehat{L}\,\widehat{L}^T$ with $\widehat{k} > k$ and then SVD-

truncate the matrix $\widehat{L}$ into a rank-k matrix L. This will lead to a rank-k approximation $L\,L^T$ that satisfies

$$\lambda_{k+1}(A) \leq \left\|\Pi^T A\Pi - L\,L^T\right\|_2 \leq \lambda_{k+1}(A) \ + \widehat{\tau}\,\lambda_{\widehat{k}+1}(A),$$
$$\lambda_j(A) \geq \sigma_j^2(L) \geq \frac{\lambda_j(A)}{1+\widehat{\tau}\,\mathbf{min}\left\{1,(1+\widehat{\tau})\frac{\lambda_{\widehat{k}+1}(A)}{\lambda_j(A)}\right\}},$$

for $1 \leq j \leq k$, and scalar $\widehat{\tau} \leq g(n-\widehat{k})(\widehat{k}+1)$. Especially for rapidly decaying singular values, i.e., $\lambda_{\widehat{k}+1}(A) \ll \lambda_{k+1}(A)$, these bounds make $L\,L^T$ almost indistinghishable from the best possible, the SVD-truncated rank-k approximation.

Complexity analysis: In addition to initialization, Algorithm 4 repeatedly computes the diagonal entries in the Schur complement, of which there are $n-k$. As in Algorithm 1, this can be done in $O(k(n-k))$ operations. We need to swap the column with the largest diagonal element to the front of trailing matrix and update the pivoted column, which needs a matrix-vector multiplication, costing another $O(k(n-k))$ operations. Then we need to compute $\Omega\widehat{L}^{-1}$ and the corresponding column l_2 norms, costing $O(dk^2)$ operations. The program stops if the while loop condition fails. Otherwise, we need to swap the column with the largest column l_2 norm of $\Omega\widehat{L}^{-1}$ and the leading column in the Schur complement. The Givens rotations required to restore lower-triangular form in L cost $O(nk)$ operations. In total, the cost of performing one swap in Algorithm 4 is $O(nk+dk^2)$ operations. Assuming $k \ll n$, this cost becomes $O(nk)$, which is negligible compared to $O(n^2(b+p))$, the cost of initialization in Algorithm 3.

Proof of correctness of Algorithm 4 For this analysis, we need the following version of a result by Johnson and Lindenstrauss.

Theorem 2. *(Random Projection (Johnson-Lindenstrauss) [20, 63, 111]). Let $x_1, \cdots, x_n \in \mathbb{R}^m$ be non-zero, $\epsilon > 0$ and $\Delta > 0$. Assume that the entries of $\Omega \in \mathbb{R}^{d\times m}$ are sampled independently from $\mathcal{N}(0,1)$. Then for $d \geq \frac{4}{\epsilon^2-\epsilon^3}\,\boldsymbol{log}\frac{2n}{\Delta}$,*

$$P\left(\sqrt{1-\epsilon} \leq \frac{||\Omega x_j||_2}{\sqrt{r}\ ||x_j||_2} \leq \sqrt{1+\epsilon}\right) \geq 1-\Delta,\ 1 \leq j \leq n.$$

Theorem 2 is the basis of many a randomized numerical linear algebra algorithm. The magic of random projection rests on the fact that the required row dimension d depends only logarithmically on Δ and n, and not on m.

Given a partial Cholesky factorization with $L_{11} \in \mathbb{R}^{k\times k}$

$$\Pi^T A\Pi = \begin{pmatrix} L_{11} & & \\ l^T & \sqrt{\alpha} & \\ L_{21} & s & I \end{pmatrix}\begin{pmatrix} L_{11} & & \\ l^T & \sqrt{\alpha} & \\ L_{21} & s & \widehat{S} \end{pmatrix}^T.$$

Assume that Algorithm 4 decides to swap the j-th and $(k+1)$-st columns and rows of $\Pi^T A \Pi$. For computational efficiency, Algorithm 4 performs a round-robin rotation by permuting the j-th through the $(k+1)$-st columns and rows of $\Pi^T A \Pi$ to the $(k+1)$-st, and the j-th through the k-th columns and rows. This corresponds to the same round-robin rotation on the rows of $\begin{pmatrix} L_{11} \\ l^T \\ L_{21} \end{pmatrix}$. Denote the new permutation matrix as $\overline{\Pi}$. Algorithm 4 then right applies $k - j + 1$ Givens rotations on the rows of $\begin{pmatrix} L_{11} \\ l^T \\ L_{21} \end{pmatrix}$ to restore its lower-triangular form, leading to a new partial Cholesky factorization

$$\overline{\Pi}^T A \overline{\Pi} = \begin{pmatrix} \overline{L}_{11} & & \\ \bar{l}^T & \sqrt{\overline{\alpha}} & \\ \overline{L}_{21} & \overline{s} & I \end{pmatrix} \begin{pmatrix} \overline{L}_{11} & & \\ \bar{l}^T & \sqrt{\overline{\alpha}} & \\ \overline{L}_{21} & \overline{s} & \widehat{S} \end{pmatrix}^T.$$

Since the round-robin rotation involves only the rows of the matrix

$$\widehat{L} \stackrel{def}{=} \begin{pmatrix} L_{11} & \\ l^T & \sqrt{\alpha} \end{pmatrix},$$

and since the Givens rotations are orthogonal and right applied on its rows, it follows that its determinant remains invariant. In other words,

$$\left| \mathbf{det} \left(\begin{pmatrix} L_{11} & \\ l^T & \sqrt{\alpha} \end{pmatrix} \right) \right| = \left| \mathbf{det} \left(\begin{pmatrix} \overline{L}_{11} & \\ \bar{l}^T & \sqrt{\overline{\alpha}} \end{pmatrix} \right) \right|.$$

This quickly leads to the identity

$$\frac{\det^2(\overline{L}_{11})}{\det^2(L_{11})} = \frac{\alpha}{\overline{\alpha}}. \tag{1.16}$$

The round-robin rotation re-arranges rows in $\widehat{L}$, and consequently the column l_2 norms in $\widehat{L}^{-1}$. The Givens rotations, being orthogonal, will leave them invariant. Since the column l_2 norm of the $(k+1)$-st column of $\begin{pmatrix} \overline{L}_{11} & \\ \bar{l}^T & \sqrt{\overline{\alpha}} \end{pmatrix}$ is simply $\frac{1}{\sqrt{\overline{\alpha}}}$, it must also be the j-th column l_2 norm of $\widehat{L}^{-1}$. Therefore,

$$\frac{\det^2(\overline{L}_{11})}{\det^2(L_{11})} = \frac{\alpha}{\overline{\alpha}} = \alpha \cdot \left\{ j\text{th column } l_2 \text{ norm of } \widehat{L}^{-1} \right\}^2.$$

Define $\imath = \mathbf{argmax}_{1\leq i\leq k+1}\left\{i\text{th column } l_2 \text{ norm of } \widehat{L}^{-1}\right\}$. If Algorithm 4 performed a swap with $j = \imath$ every time it found that $\frac{1}{\sqrt{\alpha}} \leq \frac{\left\|\widehat{L}^{-1}\right\|_{2,1}}{\sqrt{g}}$, by (1.16) we would have

$$\frac{\det^2(\overline{L}_{11})}{\det^2(L_{11})} \geq g.$$

In other words, for any given $g > 1$, each swap in Algorithm 4 would have increased the determinant of $L_{11}L_{11}^T$ by a factor of at least g. Since $L_{11}L_{11}^T$ is a permuted principal submatrix of A, this determinant is bounded above by $\|A\|_{\max}^{n-k}$. Thus Algorithm 4 would only be able to perform a finite number ($O(n)$ based on the analysis in [50]) of swaps in the worst case.

To reduce the cost in computing $\imath$, Algorithm 4 chooses to compute the maximum column l_2 norm on the random projection matrix $\Omega\widehat{L}^{-1}$. For any given $\Delta > 0, \epsilon > 0$, and $d \geq \frac{4}{\epsilon^2 - \epsilon^3}\mathbf{log}\frac{2(k+1)}{\Delta}$, Algorithm 4 correctly estimates the maximum column l_2 norm of $\widehat{L}^{-1}$ within a factor of $\sqrt{\frac{1+\epsilon}{1-\epsilon}}$ with probability at least $1 - \Delta$, according to Theorem 2. Thus, by choosing $g \gg \frac{1+\epsilon}{1-\epsilon}$, Algorithm 4 increases the determinant of $L_{11}L_{11}^T$ by a factor of at least $g\,\frac{1-\epsilon}{1+\epsilon}$ for every swap it performs with probability at least $1 - \Delta$. In practice, rarely any swaps will be required for machine learning applications.

1.4 Numerical Experiments

Data and source code are available at following URL [2]. Section 1.4.1 compares the run times of `DPSTRF.f` and `SRCH` to obtain a partial Cholesky factorization. Section 1.4.2 compares the approximation effectiveness of low-rank approximations computed by `DPSTRF.f` and `SRCH`. Section 1.4.3 and Section 1.4.4 compare `SRCH` with other pivoted Cholesky factorization based algorithms in a prediction problem and Gaussian process, respectively. `DPSTRF.f` is an LAPACK Fortran routine, while our `SRCH` is also written in Fortran. Both `DPSTRF.f` and `SRCH` call the same version of BLAS. All experiments are implemented on a laptop with a 2.7 GHz Intel Core i5 CPU and 8GB of RAM.

1.4.1 Runtime Comparison

We compare the run times of `DPSTRF.f` and `SRCH` on a kernel matrix of Combined Cycle Power Plant Data (CCPP) [66, 107] with size of 9568×4. We use radial basis function (RBF) kernel $k(x_i, x_j) = \exp(-\|x_i - x_j\|_2^2/2\sigma^2)$ and set $\sigma = 1.0$. In `SRCH`, we choose block size $b = 20$ and oversampling size $p = 10$. Run times are in Figure 1.1.

[2] `https://math.berkeley.edu/~jwxiao/`

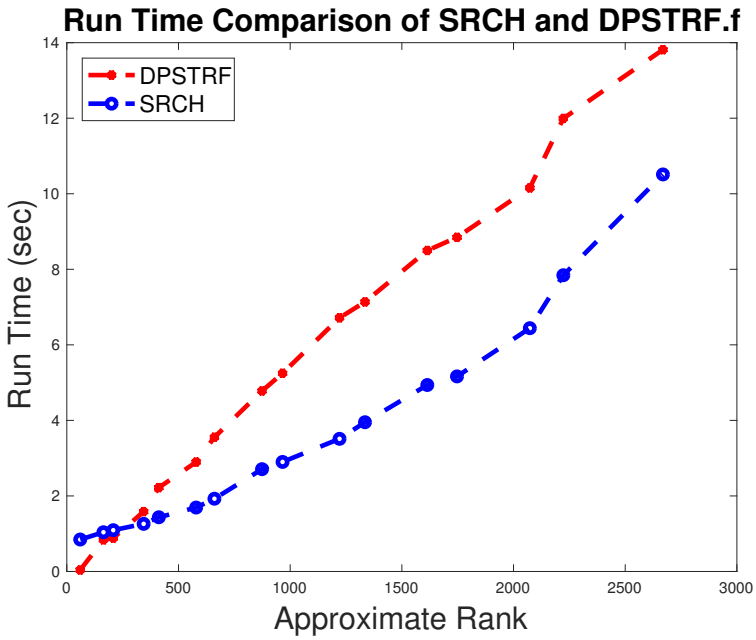

Figure 1.1: Run time comparison between SRCH and DPSTRF.f.

`DPSTRF.f` is faster when the approximate rank is small. This is because `SRCH` must compute the random projection $B = \Omega A$, which is an overhead. As approximate rank increases, `SRCH` becomes faster than `DPSTRF.f` as predicted.

1.4.2 Approximation Effectiveness Comparison

For the same kernel matrix used in Section 1.4.1, we compare the approximation effectiveness of `SRCH` and `DPSTRF.f` with their low-rank approximations. We choose $k = 20, 40, 60$, $b = 20$ and $p = 10$. Figure 1.2 shows the approximation relative errors of the top 10 eigenvalues of A. Although `SRCH` will be slower than `DPSTRF.f` because of the overhead cost of computing $B = \Omega A$, the approximation effectiveness of `SRCH` is much better than `DPSTRF.f`.

If we choose a large approximate rank k, both `SRCH` and `DPSTRF.f` will provide very high quality low-rank approximations and there will be little difference between `SRCH` and `DPSTRF.f` in relative approximation errors for the leading singular values, but `SRCH` is significantly faster than `DPSTRF.f`, as Figure 1.1 suggests.

1.4.3 Cholesky Factorization with Side Information (CSI)

[6] presents an algorithm that exploits side information in the prediction of unlabeled data with low-rank approximations for kernel matrices. To compute a low-rank approximation,

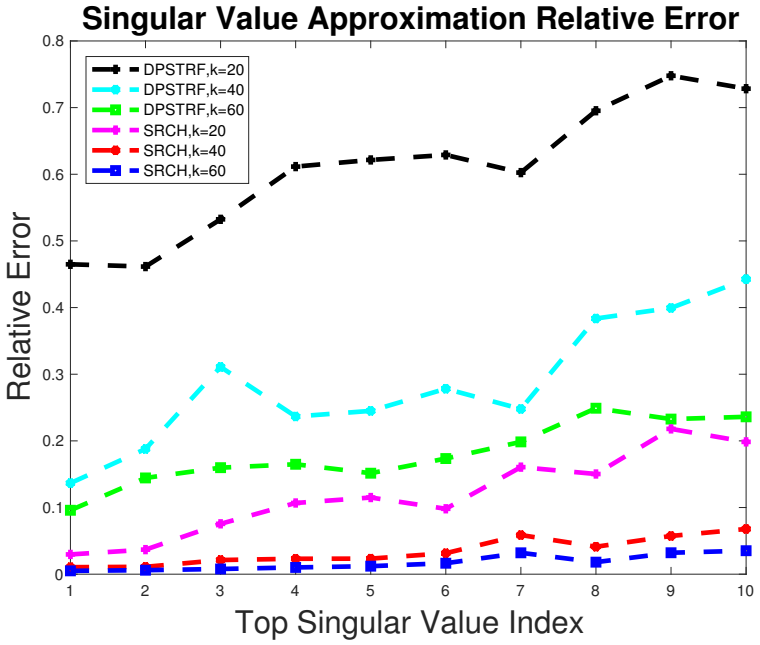

Figure 1.2: Top 10 singular value approximation relative error $\frac{\lambda_j(A)-\sigma_j^2(L)}{\lambda_j(A)}$.

this algorithm minimizes the objective function with a greedy strategy to incrementally select representative samples.

In this section, we apply `SRCH` on the kernel matrix for a low-rank approximation *without* the benefit of side information and make predictions on unlabeled data with this low-rank approximation. Details of the prediction formulas can be found in Section 6 of [6].

We compare approximation effectiveness of the low-rank approximation, run time, and prediction error on unlabeled data. We compare four methods: CSI decomposition with 40 look-ahead steps, CSI decomposition with 80 look-ahead steps, diagonal pivoted Cholesky without look-ahead, and `SRCH`. The first three methods are from [6]. We test these four methods on handwritten digit (MNIST) [73]. We use RBF kernel $k(x_i, x_j) = \exp(-\|x_i - x_j\|_2^2/2\sigma^2)$ and set $\sigma = 1.0$. We set block size $b = 50$ and oversampling size $p = 5$ in `SRCH`.

We choose 3000 training samples, 3000 testing samples, and approximate rank $k = 200$. Figure 1.3 and Figure 1.4 show approximation effectiveness and prediction accuracy, respectively. There is a slight advantage of `SRCH` on both approximation effect and prediction accuracy. We define the approximation error as $\frac{trace(K-LL^T)}{trace(K)}$.

The more impressive improvement is in run time. Figure 1.5 shows the run time comparison on the kernel matrix $K \in \mathbb{R}^{3000\times 3000}$ for different approximate ranks k. `SRCH` is significantly faster than the other three methods.

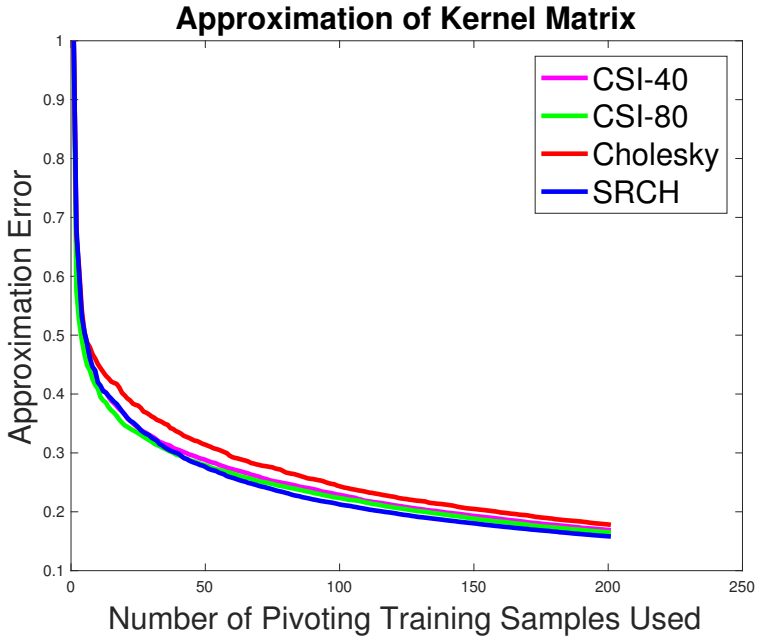

Figure 1.3: Approximation quality comparison on MNIST.

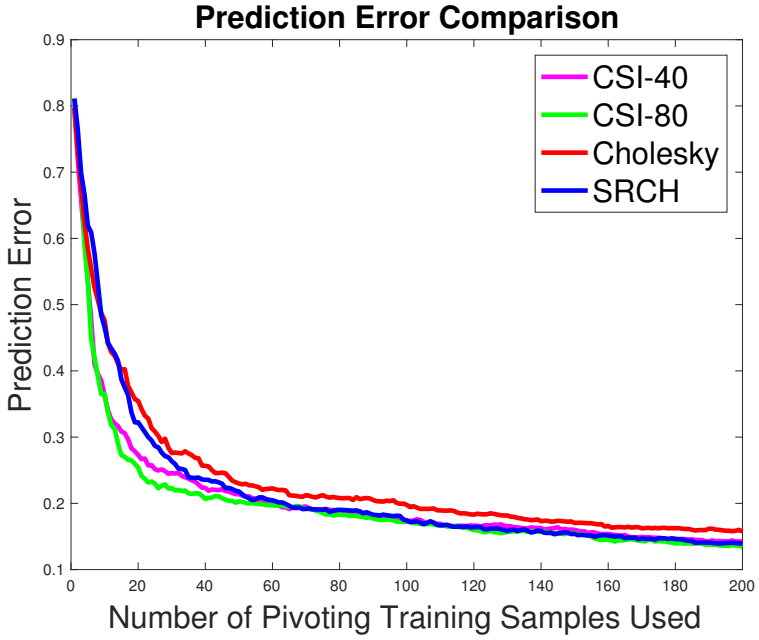

Figure 1.4: Prediction error comparison on MNIST.

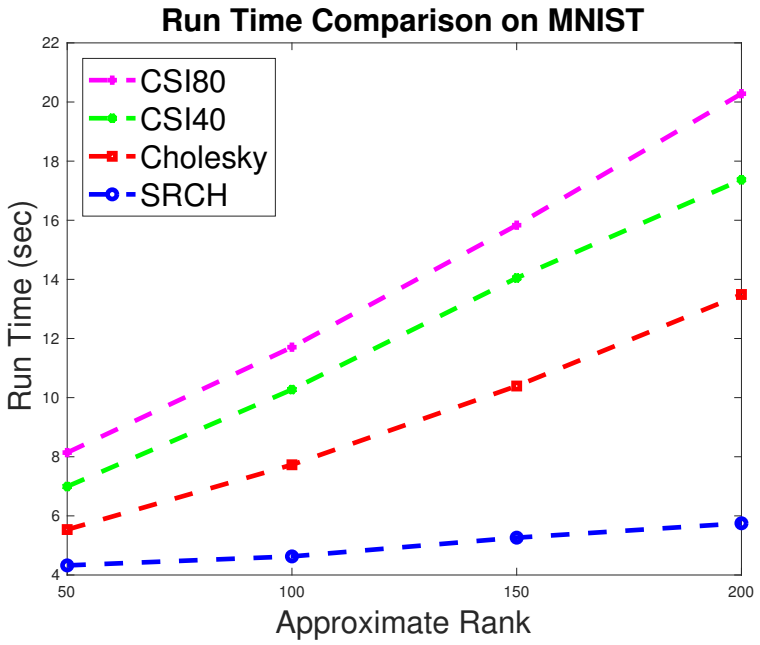

Figure 1.5: Run time comparison on MNIST.

1.4.4 Gaussian Process

Supervised learning is the problem of learning input-output mappings using empirical data. We assume the training data is a $n \times d$ matrix X. Each row represents one data sample and each column represents one feature. We assume the target values of X is a $n \times 1$ vector y. The task is to use the training data (both X and y) to make predictions on new data with a trained model. Let the new data be a $n^* \times d$ matrix X^*. The $n^* \times 1$ vector y^* will represent the target values to X^*. The goal is to predict y^* given X, y, and X^* [41].

In the Gaussian process approach the prediction of y^* involves a covariance function $\kappa(x, x')$, where x and x' are vectors with d components. It is required that the covariance function be positive semidefinite [116] which implies that the $n \times n$ covariance matrix A with entries $A_{ij} = \kappa(x_i, x_j)$ where x_i and x_j are rows of X is symmetric positive semidefinite. The covariance function can be used to construct A and also the $n^* \times n$ cross covariance matrix A^* where $A^*_{ij} = \kappa(x^*_i, x_j)$ where x^*_i is the ith row of X^*. The prediction $\widehat{y^*}$ for y is given by the Gaussian process equation [116]

$$\widehat{y^*} = A^*(\lambda I + A)^{-1} y \tag{1.17}$$

where λ is a regularization parameter.

It is not practical to solve (1.17) with large n since the number of floating point operations required is $O(n^3)$. Therefore for large n it is useful to develop approximate solutions to equation (1.17).

To this end, we choose an approximate rank $k < n$ and apply permutation Π to matrices A and A^* and partition them

$$\Pi^T A \Pi = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} = \begin{pmatrix} A_1 & A_2 \end{pmatrix}, \; A^*\Pi = \begin{pmatrix} A_1^* & A_2^* \end{pmatrix},$$

where A_{11} is $k \times k$, A_{21} is $(n-k) \times k$, $A_{12} = A_{21}^T$ is $k \times (n-k)$, A_{22} is $(n-k) \times (n-k)$, A_1 is $n \times k$, A_2 is $n \times (n-k)$, A_1^* is $n^* \times k$, and A_2^* is $n^* \times (n-k)$. We approximate $\Pi^T A \Pi$ and $A^*\Pi$ with

$$\Pi^T A \Pi \approx \widehat{A} \equiv A_1 A_{11}^{-1} A_1^T, \; A^*\Pi \approx \widehat{A}^* \equiv A_1^* A_{11}^{-1} A_1^T.$$

It follows that

$$\widehat{y}^* = A^*(\lambda I + A)^{-1} y = A^*\Pi(\lambda I + \Pi^T A \Pi)^{-1}\Pi^T y$$

can be approximated by

$$\begin{aligned} \tilde{y}^* &= \widehat{A}^* \left(\lambda I + \widehat{A}\right)^{-1} \Pi^T y \\ &= A_1^* A_{11}^{-1} A_1^T \left(\lambda I + A_1 A_{11}^{-1} A_1^T\right)^{-1} \Pi^T y \\ &= A_1^* A_{11}^{-1} (\lambda I + A_1^T A_1 A_{11}^{-1})^{-1} A_1^T \Pi^T y \\ &= A_1^* (\lambda A_{11} + A_1^T A_1)^{-1} A_1^T \Pi^T y. \end{aligned}$$

The partial Cholesky factorization

$$\begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} = \begin{pmatrix} L_{11} & \\ L_{21} & I_{n-k} \end{pmatrix} \begin{pmatrix} L_{11}^T & L_{21}^T \\ & S \end{pmatrix},$$

implies

$$A_{11} = L_{11} L_{11}^T, \quad A_1 = \begin{pmatrix} L_{11} \\ L_{21} \end{pmatrix} L_{11}^T = L L_{11}^T,$$

and therefore

$$\begin{aligned} \tilde{y}^* &= A_1^* (\lambda A_{11} + A_1^T A_1)^{-1} A_1^T \Pi^T y \\ &= A_1^* (\lambda L_{11} L_{11}^T + L_{11} L^T L L_{11}^T)^{-1} L^T \Pi^T y \\ &= A_1^* L_{11}^{-T} (\lambda I_k + L^T L)^{-1} L^T \Pi^T y. \end{aligned} \tag{1.18}$$

(1.18) suggests that to compute $\tilde{y}^*$, we need L, Π, and A_1^*, but not the Schur complement S or A_2^*. L and Π can be computed by a partial Cholesky factorization and A_1^* can be computed directly using covariance function after obtaining Π.

We compute the Gaussian process on CCPP dataset with `DPSTRF.f` and `SRCH`. The test was implemented on a single node of the NERSC machine Edison using Intel math kernel library [113]. The training data matrix is of size 3000×4 and the testing data matrix is of size 6000×4. The covariance function is $\kappa(x_i, x_j) = \exp(-\|x_i - x_j\|_2^2 / 2\sigma^2)$. We choose $\lambda = 0.0001$ and $\sigma = 2.0$. In `SRCH`, we choose block size $b = 30$ and oversampling size $p = 5$.

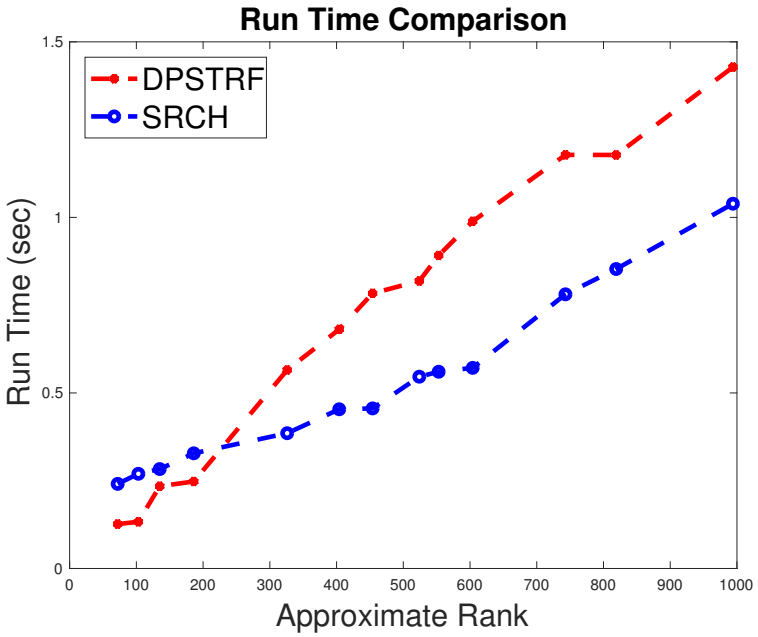

Figure 1.6: Run time comparison on CCPP.

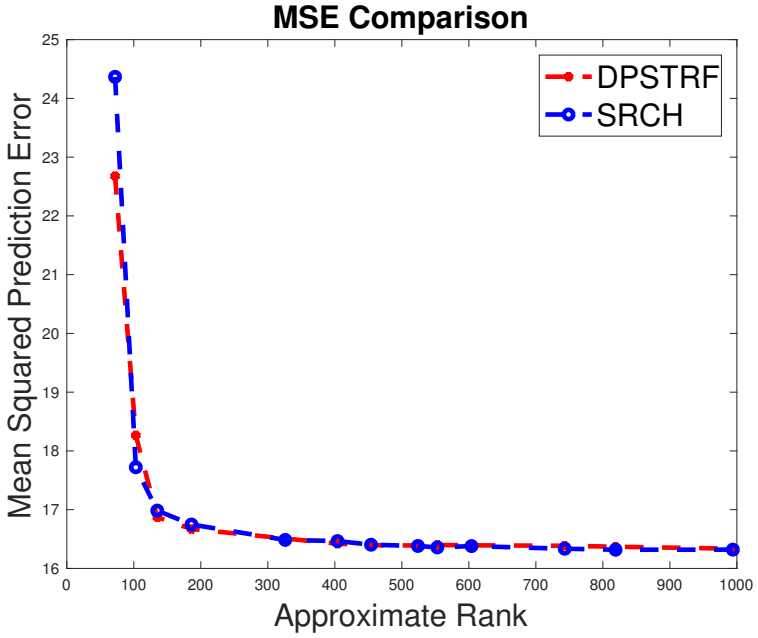

Figure 1.7: Mean squared prediction error comparison on CCPP.

We compare the runtime and mean squared prediction error in Figure 1.6 and Figure 1.7, respectively.

In Figure 1.6, the run times of `DPSTRF.f` and `SRCH` intersect at around 250. `SRCH` out-performs `DPSTRF.f` for larger approximate ranks. Figure 1.7 demonstrates that while `DPSTRF.f` makes better predictions than `SRCH` for smaller approximate ranks k, they are not the range in which the best predictions are made. For larger k, `SRCH` provides slightly smaller prediction error than `DPSTRF.f`, as suggested in Figure 1.6.

1.5 Incomplete Randomized Cholesky

This section contains my unpublished results about incomplete randomized Cholesky. We start with Lemma 1.

Lemma 1. *Assume*

$$R = \begin{pmatrix} A & B \\ & C \end{pmatrix},$$

where $R \in \mathbb{R}^{m\times n}$, $A \in \mathbb{R}^{l\times l}$, $B \in \mathbb{R}^{l\times(n-l)}$, *and* $C \in \mathbb{R}^{(m-l)\times(n-l)}$, *then*

$$\sigma_j \begin{pmatrix} A & B \end{pmatrix} \geq \frac{\sigma_j(R)}{\sqrt{1+\frac{\|C\|_2^2}{\sigma_j^2\begin{pmatrix} A & B \end{pmatrix}}}} \quad (1 \leq j \leq l).$$

Proof of Lemma 1. $R_1 \overset{def}{=} \begin{pmatrix} A & B \end{pmatrix} \in \mathbb{R}^{l\times n}, R_2 \overset{def}{=} \begin{pmatrix} O & C \end{pmatrix} \in \mathbb{R}^{(m-l)\times n}$, for $1 \leq j \leq l$,

$$\begin{aligned} \sigma_j^2(R) &= \lambda_j\left(R^TR\right) = \lambda_j\left(\begin{pmatrix} R_1^T & R_2^T \end{pmatrix}\begin{pmatrix} R_1 \\ R_2 \end{pmatrix}\right) \\ &= \lambda_j\left(R_1^TR_1 + R_2^TR_2\right) \leq \lambda_j\left(R_1^TR_1\right) + \left\|R_2^TR_2\right\|_2 = \sigma_j^2(R_1) + \|R_2\|_2^2. \end{aligned}$$

Recall the definitions of R_1 and R_2, we have

$$\sigma_j \begin{pmatrix} A & B \end{pmatrix} \geq \frac{\sigma_j(R)}{\sqrt{1+\frac{\|C\|_2^2}{\sigma_j^2\begin{pmatrix} A & B \end{pmatrix}}}} \quad (1 \leq j \leq l).$$

□

Now we are ready to introduce a new low-rank approximation to a given symmetric positive semidefinite matrix.

Theorem 3. *Consider an incomplete Cholesky factorization of a symmetric positive semidefinite matrix* $A \in \mathbb{R}^{n\times n}$ *with any permutation matrix* Π,

$$\Pi A \Pi^T = LDL^T, \quad \textit{where} \quad L = \begin{pmatrix} A_l \\ B_l & I_{n-l} \end{pmatrix}, \quad D = \begin{pmatrix} I_l & \\ & C \end{pmatrix}.$$

Consider the QR factorization of $\begin{pmatrix} A_l \\ B_l \end{pmatrix} = \widehat{Q}\widehat{R}$, *we define* $\widehat{A} \overset{def}{=} \widehat{Q}\widehat{Q}^T \Pi A \Pi^T \widehat{Q}\widehat{Q}^T$, *which is equivalent to* $\begin{pmatrix} A_l \\ B_l \end{pmatrix} \left(I_l + \left(A_l^T A_l + B_l^T B_l \right)^{-1} B_l^T C B_l \left(A_l^T A_l + B_l^T B_l \right)^{-1} \right) \begin{pmatrix} A_l^T & B_l^T \end{pmatrix}$.

$$\sigma_j\left(\widehat{A}\right) \geq \frac{\sigma_j(A)}{\sqrt{1 + \frac{2\|C\|_2^2}{\sigma_j^2\left(\widehat{A}\right)}}} \quad (1 \leq j \leq l).$$

Proof. We do Cholesky factorization of the trailing matrix $C = C_l C_l^T$,

$$\Pi A \Pi^T = \begin{pmatrix} A_l & \\ B_l & C_l \end{pmatrix} \begin{pmatrix} A_l^T & B_l^T \\ & C_l^T \end{pmatrix}.$$

We do QR decomposition of $\begin{pmatrix} A_l & \\ B_l & C_l \end{pmatrix} = \begin{pmatrix} Q_1 & Q_2 \end{pmatrix} \begin{pmatrix} R_{11} & R_{12} \\ & R_{22} \end{pmatrix}$, and observe that

$$\begin{aligned} \begin{pmatrix} Q_1^T \\ Q_2^T \end{pmatrix} \Pi A \Pi^T &= \begin{pmatrix} Q_1^T \\ Q_2^T \end{pmatrix} \begin{pmatrix} A_l & \\ B_l & C_l \end{pmatrix} \begin{pmatrix} A_l^T & B_l^T \\ & C_l^T \end{pmatrix} \\ &= \begin{pmatrix} R_{11} & R_{12} \\ & R_{22} \end{pmatrix} \begin{pmatrix} A_l^T & B_l^T \\ & C_l^T \end{pmatrix} \\ &= \begin{pmatrix} R_{11}A_l^T & R_{11}B_l^T + R_{12}C_l^T \\ & R_{22}C_l^T \end{pmatrix}. \end{aligned}$$

Applying Lemma 1,

$$\sigma_j^2\left(R_{11}A_l^T, R_{11}B_l^T + R_{12}C_l^T\right) + \left\|R_{22}C_l^T\right\|_2^2 \geq \sigma_j^2\left(\begin{pmatrix} Q_1^T \\ Q_2^T \end{pmatrix} \Pi A \Pi^T\right) = \sigma_j^2(A).$$

Define $H \overset{def}{=} R_{11}R_{11}^T + R_{12}R_{12}^T$, $\widehat{H} \overset{def}{=} R_{12}R_{22}^T$, then

$$\begin{aligned} & \begin{pmatrix} R_{11}A_l^T & R_{11}B_l^T + R_{12}C_l^T \end{pmatrix} \\ = \; & \begin{pmatrix} R_{11} & R_{12} \end{pmatrix} \begin{pmatrix} A_l^T & B_l^T \\ & C_l^T \end{pmatrix} = \begin{pmatrix} R_{11} & R_{12} \end{pmatrix} \begin{pmatrix} R_{11}^T & \\ R_{12}^T & R_{22}^T \end{pmatrix} \begin{pmatrix} Q_1^T \\ Q_2^T \end{pmatrix} \\ = \; & \begin{pmatrix} R_{11}R_{11}^T + R_{12}R_{12}^T & R_{12}R_{22}^T \end{pmatrix} \begin{pmatrix} Q_1^T \\ Q_2^T \end{pmatrix} = \begin{pmatrix} H & \widehat{H} \end{pmatrix} \begin{pmatrix} Q_1^T \\ Q_2^T \end{pmatrix}. \end{aligned}$$

Notice that

$$\begin{aligned} & \begin{pmatrix} A_l^T & B_l^T \\ & C_l^T \end{pmatrix} \begin{pmatrix} A_l & \\ B_l & C_l \end{pmatrix} = \begin{pmatrix} R_{11}^T & \\ R_{12}^T & R_{22}^T \end{pmatrix} \begin{pmatrix} R_{11} & R_{12} \\ & R_{22} \end{pmatrix} \\ \Leftrightarrow \; & \begin{pmatrix} A_l^T A_l + B_l^T B_l & B_l^T C_l \\ C_l^T B_l & C_l^T C_l \end{pmatrix} = \begin{pmatrix} R_{11}^T R_{11} & R_{11}^T R_{12} \\ R_{12}^T R_{11} & R_{12}^T R_{12} + R_{22}^T R_{22} \end{pmatrix}, \end{aligned}$$

therefore

$$R_{11}^T R_{12} = B_l^T C_l \Rightarrow R_{12} = R_{11}^{-T} B_l^T C_l,$$

and

$$\begin{aligned} R_{12}R_{12}^T &= R_{11}^{-T} B_l^T C_l C_l^T B_l R_{11}^{-1} \\ &= R_{11}\left(\left(R_{11}^T R_{11}\right)^{-1} B_l^T C_l C_l^T B_l \left(R_{11}^T R_{11}\right)^{-1}\right) R_{11}^T \\ &= R_{11}\left(A_l^T A_l + B_l^T B_l\right)^{-1} B_l^T C B_l \left(A_l^T A_l + B_l^T B_l\right)^{-1} R_{11}^T, \end{aligned}$$

and

$$R_{11}R_{11}^T + R_{12}R_{12}^T = R_{11}\left(I_l + \left(A_l^T A_l + B_l^T B_l\right)^{-1} B_l^T C B_l \left(A_l^T A_l + B_l^T B_l\right)^{-1}\right) R_{11}^T.$$

Notice that

$$\begin{aligned} \widehat{A} &= \begin{pmatrix} A_l \\ B_l \end{pmatrix} \left(I_l + \left(A_l^T A_l + B_l^T B_l\right)^{-1} B_l^T C B_l \left(A_l^T A_l + B_l^T B_l\right)^{-1}\right) \begin{pmatrix} A_l^T & B_l^T \end{pmatrix} \\ &= Q \begin{pmatrix} R_{11} \\ O \end{pmatrix} \left(I_l + \left(A_l^T A_l + B_l^T B_l\right)^{-1} B_l^T C B_l \left(A_l^T A_l + B_l^T B_l\right)^{-1}\right) \begin{pmatrix} R_{11}^T & O \end{pmatrix} Q^T \\ &= Q_1 R_{11}\left(I_l + \left(A_l^T A_l + B_l^T B_l\right)^{-1} B_l^T C B_l \left(A_l^T A_l + B_l^T B_l\right)^{-1}\right) R_{11}^T Q_1^T \\ &= Q_1\left(R_{11}R_{11}^T + R_{12}R_{12}^T\right) Q_1^T, \end{aligned}$$

and hence

$$\sigma_j\left(\widehat{A}\right) = \sigma_j\left(R_{11}R_{11}^T + R_{12}R_{12}^T\right) = \sigma_j(H) \quad (1 \le j \le l).$$

Therefore

$$\begin{aligned} & \sigma_j^2 \begin{pmatrix} R_{11}A_l^T & R_{11}B_l^T + R_{12}C_l^T \end{pmatrix} = \sigma_j^2 \begin{pmatrix} H & \widehat{H} \end{pmatrix} \\ = \;& \sigma_j\left(HH^T + \widehat{H}\widehat{H}^T\right) = \lambda_j\left(HH^T + \widehat{H}\widehat{H}^T\right) \\ \le \;& \sigma_j^2(H) + \left\|\widehat{H}\widehat{H}^T\right\|_2 = \sigma_j^2\left(\widehat{A}\right) + \left\|\widehat{H}\right\|_2^2, \end{aligned}$$

and hence

$$\begin{aligned} \sigma_j^2(A) &\le \sigma_2^2 \begin{pmatrix} R_{11}A_l^T & R_{11}B_l^T + R_{12}C_l^T \end{pmatrix} + \left\|R_{22}C_l^T\right\|_2^2 \\ &\le \sigma_j^2\left(\widehat{A}\right) + \left\|\widehat{H}\right\|_2^2 + \left\|R_{22}C_l^T\right\|_2^2. \end{aligned}$$

Notice that $C_l^T C_l = R_{12}^T R_{12} + R_{22}^T R_{22}$, then $\|R_{12}\|_2 \le \|C_l\|_2$ and $\|R_{22}\|_2 \le \|C_l\|_2$. Therefore $\left\|R_{22}C_l^T\right\|_2^2 \le \|C_l\|_2^4 = \|C\|_2^2$ and $\left\|\widehat{H}\right\|_2^2 \le \|C_l\|_2^4 = \|C\|_2^2$. Finally we obtain

$$\sigma_j^2(A) \le \sigma_j^2\left(\widehat{A}\right) + 2\|C\|_2^2,$$

which is equivalent to

$$\sigma_j\left(\widehat{A}\right) \geq \frac{\sigma_j(A)}{\sqrt{1+\frac{2\|C\|_2^2}{\sigma_j^2(\widehat{A})}}} \quad (1 \leq j \leq l).$$

□

In Theorem 3, we use the whole matrix A to compute $\widehat{A}$. Now we design a different low-rank approximation A_{deep}, which only uses part of A. In fact, we do incomplete Cholesky factorization of A and then we do another incomplete Cholesky factorization of the trailing matrix C,

$$\begin{aligned}
\Pi_1 A \Pi_1^T &= \begin{pmatrix} A_l \\ B_l \end{pmatrix} \begin{pmatrix} A_l^T & B_l^T \end{pmatrix} + \begin{pmatrix} O & O \\ O & \Pi_2 C \Pi_2^T \end{pmatrix} \quad (l \text{ steps Cholesky}) \\
&= \begin{pmatrix} A_l \\ B_l \end{pmatrix} \begin{pmatrix} A_l^T & B_l^T \end{pmatrix} + \begin{pmatrix} O & O \\ O & H_l H_l^T + \widehat{C} \end{pmatrix} \quad (\delta \text{ steps Cholesky}) \\
&= \begin{pmatrix} A_l & \\ B_l & H_l \end{pmatrix} \begin{pmatrix} A_l^T & B_l^T \\ & H_l^T \end{pmatrix} + \begin{pmatrix} O & O \\ O & \widehat{C} \end{pmatrix},
\end{aligned}$$

where Π_1 and Π_2 are any permutations matrices. Our low-rank approximation is

$$A_{deep} \overset{def}{=} \begin{pmatrix} A_l \\ B_l \end{pmatrix} (I+H) \begin{pmatrix} A_l^T & B_l^T \end{pmatrix},$$

where

$$H \overset{def}{=} \begin{pmatrix} A_l \\ B_l \end{pmatrix} \left(I_l + \left(A_l^T A_l + B_l^T B_l\right)^{-1} B_l^T H_l H_l^T B_l \left(A_l^T A_l + B_l^T B_l\right)^{-1} \right) \begin{pmatrix} A_l^T & B_l^T \end{pmatrix}.$$

We have the following theorem for the singular values bounds of A_{deep}.

Theorem 4. *Assume we do incomplete Cholesky factorization of a symmetric positive semidefinite matrix*

$$A = \begin{pmatrix} A_l & \\ B_l & H_l \end{pmatrix} \begin{pmatrix} A_l^T & B_l^T \\ & H_l^T \end{pmatrix} + \begin{pmatrix} O & O \\ O & \widehat{C} \end{pmatrix}.$$

A_{deep} is defined as

$$A_{deep} = \begin{pmatrix} A_l \\ B_l \end{pmatrix} (I+H) \begin{pmatrix} A_l^T & B_l^T \end{pmatrix},$$

where $H = \begin{pmatrix} A_l \\ B_l \end{pmatrix} \left(I_l + (A_l^T A_l + B_l^T B_l)^{-1} B_l^T H_l H_l^T B_l (A_l^T A_l + B_l^T B_l)^{-1}\right) \begin{pmatrix} A_l^T & B_l^T \end{pmatrix}$.

$$\sigma_j\left(A_{deep}\right) \geq \frac{\sigma_j(A)}{\frac{\left\|\widehat{C}\right\|_2}{\sigma_j(A_{deep})} + \sqrt{1+\frac{2\left\|H_l H_l^T\right\|_2^2}{\sigma_j^2(A_{deep})}}} \quad (1 \leq j \leq l).$$

Proof of Theorem 4. Using the bounds in Theorem 3, we have

$$\sigma_j^2\left(\begin{pmatrix} A_l & \\ B_l & H_l \end{pmatrix}\begin{pmatrix} A_l^T & B_l^T \\ & H_l^T \end{pmatrix}\right) \leq \sigma_j^2(A_{deep}) + 2\left\|H_l H_l^T\right\|_2^2 \quad (1 \leq j \leq l), \tag{1.19}$$

and by Weyl inequality we have

$$\sigma_j(A) \leq \sigma_j\left(\begin{pmatrix} A_l & \\ B_l & H_l \end{pmatrix}\begin{pmatrix} A_l^T & B_l^T \\ & H_l^T \end{pmatrix}\right) + \left\|\widehat{C}\right\|_2 \quad (1 \leq j \leq l). \tag{1.20}$$

Taking the square root of (1.19) and then combine it with (1.20), we obtain

$$\sigma_j(A) \leq \sqrt{\sigma_j^2(A_{deep}) + 2\left\|H_l H_l^T\right\|_2^2} + \left\|\widehat{C}\right\|_2 \quad (1 \leq j \leq l),$$

and hence

$$\sigma_j(A_{deep}) \geq \frac{\sigma_j(A)}{\frac{\left\|\widehat{C}\right\|_2}{\sigma_j(A_{deep})} + \sqrt{1 + \frac{2\left\|H_l H_l^T\right\|_2^2}{\sigma_j^2(A_{deep})}}} \quad (1 \leq j \leq l).$$

□

Chapter 2

Spectrum-Revealing QR

2.1 Introduction

2.1.1 QR Factorizations with Column Pivoting

A QR factorization of a matrix $A \in \mathbb{R}^{m\times n}$ is $A = QR$ where $Q \in \mathbb{R}^{m\times m}$ is orthogonal and $R \in \mathbb{R}^{m\times n}$ is upper trapezoidal. In LAPACK [2] and ScaLAPACK [11], the QR factorization can be computed by xGEQRF and PxGEQRF, respectively, where x indicates the matrix data type. [1]

In practical situations where either the matrix A is not always known to be of full rank or we want to find representative columns of A, we compute a full or partial QR factorization with column pivoting (QRCP) of the form

$$A\Pi = QR \tag{2.1}$$

for a matrix $A \in \mathbb{R}^{m\times n}$, where $\Pi \in \mathbb{R}^{n\times n}$ is a permutation matrix and $Q \in \mathbb{R}^{m\times m}$ is an orthogonal matrix. A full QRCP, with $R \in \mathbb{R}^{m\times n}$ being upper trapezoidal, can be computed by either xGEQPF or xGEQP3 in LAPACK [2], and either PxGEQPF or PxGEQP3 in ScaLAPACK [11]. xGEQP3 and PxGEQP3 are Level 3 BLAS versions of xGEQPF and PxGEQPF respectively. They are considerably faster while maintaining the same numerical behavior.

Given an approximate rank $1 \le k \le \min(m, n)$ in partial QRCP, equation (2.1) can be written in a 2×2 block form as

$$A\Pi = Q \begin{pmatrix} R_{11} & R_{12} \\ 0 & R_{22} \end{pmatrix} = \begin{pmatrix} Q_1 & Q_2 \end{pmatrix} \begin{pmatrix} R_{11} & R_{12} \\ 0 & R_{22} \end{pmatrix},$$

with upper triangular $R_{11} \in \mathbb{R}^{k\times k}$. If Π is chosen appropriately, the partial QRCP can separate linearly independent columns from dependent ones, yielding a low-rank approximation

[1] Materials in this chapter are mainly from the paper titled Fast Parallel Randomized QR with Column Pivoting Algorithms for Reliable Low-rank Matrix Approximations [121], which was published in 2017 IEEE 24th International Conference on High Performance Computing (HiPC).

$A \approx Q_1 \begin{pmatrix} R_{11} & R_{12} \end{pmatrix} \Pi^T$. Efficient and reliable low-rank approximations are useful in many applications including data mining [120] and image processing [103].

2.1.2 Randomization in Numerical Linear Algebra

Traditional matrix algorithms are prohibitively expensive for many applications where the datasets are very large. Randomization allows us to design provably accurate algorithms for matrix problems that are massive or computationally expensive. Randomized matrix algorithms have been successfully developed in fast least squares [19], sketching algorithms [26], low-rank approximation problems [48], etc.

The computational bottleneck of QRCP is in searching pivots, which requires updating all the column l_2 norms in the trailing matrix. While the number of floating point operations is relatively small, column l_2 norm updating incurs excessive communication costs and is at least as expensive as QR. [34] develops a randomized QRCP (RQRCP), where random sampling is used for column selection, resulting in dramatically reduced communication costs. They also introduce updating formulas to reduce the cost of sampling. With their column selection mechanism they obtain approximations that are comparable to those from the QRCP in quality, but with performance near QR. They demonstrate strong parallel scalability on shared memory multiple core systems using an implementation in Fortran with OpenMP.

2.1.3 Contributions

- **Reliability analysis:** We show, with a rigorous probability analysis, that RQRCP algorithms can be as reliable as QRCP up to failure probabilities that exponentially decay with oversampling size.

- **Distributed memory implementation:** We extend RQRCP shared memory implementation of [34] to distributed memory machines. Based on ScaLAPACK, our implementation is significantly faster than QRCP routines in ScaLAPACK, yet as effective in quality.

- **Spectrum-revealing QR factorization:** We propose a novel variant of the QR factorization for low-rank approximation: spectrum-revealing QR factorization (SRQR). We prove singular value bounds and residual error bounds for SRQR, and develop RQRCP-based efficient algorithms for its computation. We also propose SRQR based CUR and CX matrix decomposition algorithms. SRQR based algorithms are as effective as other state-of-the-art CUR and CX matrix decomposition algorithms, while significantly faster.

Algorithm 5 QR with Column Pivoting (QRCP)

Inputs:
A is $m \times n$ matrix, k is approximate rank, $1 \leq k \leq \min(m, n)$
Outputs:
Q is $m \times m$ orthogonal matrix
R is $m \times n$ upper trapezoidal matrix
Π is $n \times n$ permutation matrix such that $A\Pi = QR$
Algorithm:
Initialize $\Pi^{(0)} = I_n$, $r_s = ||A(1:m, s)||_2 \ \ (1 \leq s \leq n)$
for $i = 1 : k$ **do**
  Find $j = argmax_{i \leq s \leq n} r_s$
  Swap r_i and r_j, $A(1:m, i)$ and $A(1:m, j)$
  Update permutation with last swap $\Pi^{(i)} = \Pi^{(i-1)}\Pi_{i,j}$
  Form Householder reflection H_i from $A(i:m, i)$
  Update $A(i:m, i:n) \leftarrow H_i A(i:m, i:n)$
  Update $r_s = ||A(i+1:m, s)||_2 \quad (i+1 \leq s \leq n)$
end for
$Q = H_1 H_2 \cdots H_k$ is the product of all reflections
$R =$ upper trapezoidal part of A, $\Pi = \Pi^{(k)}$

2.2 Introduction to QRCP

QRCP is introduced in Algorithm 5. In each loop, QRCP swaps the leading column with the column in the trailing matrix with the largest column l_2 norm. It is a very effective practical tool for low-rank matrix approximations, but may require additional column interchanges to reveal the rank for contrived pathological matrices [49, 65].

To find a correct pivot in the $(i+1)^{\text{th}}$ loop, trailing column l_2 norms must be updated to remove contributions from row i. This computation, while relatively minor flop-wise [32], requires accessing the entire trailing matrix, and is a primary cause of significant slow-down of QRCP over QR.

LAPACK subroutines xGEQPF and xGEQP3 are based on Algorithm 5. The difference is that xGEQP3 re-organizes the computations to apply Householder reflections in blocks to the trailing matrix, partly with Level 3 BLAS instead of Level 2 BLAS, for faster execution. ScaLAPACK subroutines PxGEQPF and PxGEQP3 are the parallel versions of xGEQPF and xGEQP3 in distributed memory systems, respectively, with PxGEQP3 being the more efficient.

2.3 Randomized QRCP

2.3.1 Introduction to RQRCP

Algorithm 6 Randomized QRCP (RQRCP)

Inputs:
A is $m \times n$ matrix, k is approximate rank, $1 \leq k \leq \min(m, n)$
Outputs:
Q is $m \times m$ orthogonal matrix
R is $m \times n$ upper trapezoidal matrix
Π is $n \times n$ permutation matrix such that $A\Pi = QR$
Algorithm:
Determine block size b and oversampling size $p \geq 0$
Generate i.i.d Gaussian matrix $\Omega \in \mathcal{N}(0,1)^{(b+p)\times m}$
Compute $B = \Omega A$, initialize $\Pi = I_n$
for $i = 1 : b : k$ **do**
$b = \min(b, k - i + 1)$
Partial QRCP on $B(:, i : n)$ to get b pivots $(j_1, \ldots, j_b)$
Swap $(j_1, \ldots, j_b)$ and $(i : i + b - 1)$ columns in A, Π
Do unpivoted QR on $A(i : m, i : i + b - 1) = \widetilde{Q}\widetilde{R}$
$A(i : m, i + b : n) \leftarrow \widetilde{Q}^T A(i : m, i + b : n)$
$B(1 : b, i + b : n) = B(1 : b, i + b : n) - B(1 : b, i : j + b - 1)(A(i : i + b - 1.i : i + b - 1))^{-1} A(i : i + b - 1, i + b : n)$
end for
Q is the product of $\widetilde{Q}$, R = upper trapezoidal part of A

RQRCP applies a random matrix Ω of independent and identically distributed (i.i.d.) random variables to A to compress A into $B = \Omega A$ with much smaller row dimension, where the block size b and oversampling size p are chosen with $b + p \ll m$. QRCP and RQRCP make pivot decisions on A and B, respectively. Since B has much smaller row dimension than A, RQRCP can choose pivots much more quickly than QRCP. RQRCP repeatedly runs partial QRCP on B to pick b pivots, computes the QR factorization on the pivoted columns, forms a block of Householder reflections, applies them to the trailing matrix of A with Level 3 BLAS, and then updates the remaining columns of B. RQRCP exits this process when it reaches an approximate rank k.

The RQRCP algorithm as described in Algorithm 6 computes a low-rank approximation with an approximate rank k. While it can be modified to compute a low-rank approximation that satisfies an error tolerance, our analysis will be on Algorithm 6. The block size b is a machine-dependent parameter experimentally chosen to optimize performance; whereas oversampling size p is chosen to ensure the reliability of Algorithm 6. In practice, a value of p between 5 and 20 suffices. At the end of each loop, matrix B is updated, via one of

several updating formulas, to become a random projection matrix for the trailing matrix. In Section 2.3.3 we will justify Algorithm 6 with a rigorous probability analysis.

2.3.2 Updating Formulas for B in RQRCP

We discuss updating formulas for B and their efficiency differences. Consider partial QRs on $A\Pi$ and $B\Pi = \Omega A\Pi$,

$$A\Pi = Q \begin{pmatrix} R_{11} & R_{12} \\ & R_{22} \end{pmatrix}, \quad B\Pi = \widehat{Q} \begin{pmatrix} \widehat{R}_{11} & \widehat{R}_{12} \\ & \widehat{R}_{22} \end{pmatrix}, \tag{2.2}$$

where $A \in \mathbb{R}^{m \times n}$. We describe two different updating formulas introduced in [34] and [82], respectively.

For the first updating formula [34], partition $\widehat{\Omega} \stackrel{def}{=} \widehat{Q}^T \Omega Q \stackrel{def}{=} \begin{pmatrix} \widehat{\Omega}_1 & \widehat{\Omega}_2 \end{pmatrix}$, equation (2.2) implies

$$\widehat{\Omega} \begin{pmatrix} R_{11} & R_{12} \\ & R_{22} \end{pmatrix} = \begin{pmatrix} \widehat{R}_{11} & \widehat{R}_{12} \\ & \widehat{R}_{22} \end{pmatrix}, \tag{2.3}$$

leading to the updating formula of Algorithm 6,

$$\begin{pmatrix} \widehat{R}_{12} \\ \widehat{R}_{22} \end{pmatrix} \leftarrow \widehat{\Omega}_2 R_{22} = \begin{pmatrix} \widehat{R}_{12} - \widehat{R}_{11} R_{11}^{-1} R_{12} \\ \widehat{R}_{22} \end{pmatrix}, \tag{2.4}$$

where only the upper part of updating formula (2.4) requires computation, at a total cost of $O(n\,k\,b)$.

For the second updating formula [82], partition

$$\overline{\Omega} \stackrel{def}{=} \Omega\, Q \stackrel{def}{=} \begin{pmatrix} \overline{\Omega}_1 & \overline{\Omega}_2 \end{pmatrix}, \text{ and } B\,\Pi \stackrel{def}{=} \begin{pmatrix} B_1 & B_2 \end{pmatrix},$$

leading to an updating formula,

$$\begin{pmatrix} \widehat{R}_{12} \\ \widehat{R}_{22} \end{pmatrix} \leftarrow \overline{\Omega}_2 R_{22} = B_2 - \overline{\Omega}_1 R_{12}. \tag{2.5}$$

The approach in [82] decomposes Q and is mathematically equivalent to but computationally less efficient than (2.5), which requires totaling $O((b+p)(m+n)k)$ operations.

Both updating formulas are numerically stable in practice. Since $\widehat{\Omega} = \widehat{Q}^T \overline{\Omega}$, the updating formula (2.5) differs from formula (2.4) only by a pre-multiplied orthogonal matrix $\widehat{Q}$. Consequently, both updating formulas produce identical permutations and thus identical low-rank approximations. However, updating formula (2.4) requires 50% fewer operations than (2.5) and is much faster in numerical experiments.

2.3.3 Probability Analysis of RQRCP

In this section, we show that RQRCP is as reliable as QRCP up to negligible failure probabilities. Since QRCP chooses the column with the largest norm as the pivot column in each loop, QRCP satisfies the following pseudo-diagonal dominance property.

Lemma 2 (QRCP pseudo-diagonal dominance)**.** *Let $A \in \mathbb{R}^{m\times n}$. Perform a k-step partial QRCP on A,*

$$A\Pi = QR = Q \begin{pmatrix} R_{11} & R_{12} \\ & R_{22} \end{pmatrix}, \quad \textit{with} \quad R = (r_{ij}),$$

$$\textit{then } |r_{ii}| \geq \sqrt{\sum_{l=i}^{m} |r_{lj}|^2}\, (1 \leq i \leq k, i+1 \leq j \leq n).$$

Actually, the column pivots of A found by QRCP and the symmetric pivots of A^TA found by diagonal pivoted Cholesky would be the same. A similar pseudo-diagonal dominance property holds for RQRCP with approximate rank k.

Theorem 5 (RQRCP pseudo-diagonal dominance)**.** *$A \in \mathbb{R}^{m\times n}, \varepsilon, \Delta \in (0,1)$. Draw $\Omega \in \mathcal{N}(0,1)^{(b+p)\times m}$ with $p \geq \lceil \frac{4}{\varepsilon^2-\varepsilon^3} \boldsymbol{log} \frac{2nk}{\Delta} \rceil - 1$. Perform RQRCP on A given k,*

$$A\Pi = QR = Q \begin{pmatrix} R_{11} & R_{12} \\ & R_{22} \end{pmatrix}, \quad \textit{with} \quad R = (r_{ij}),$$

$$\textit{then } |r_{ii}| \geq \sqrt{\frac{1-\varepsilon}{1+\varepsilon}} \sqrt{\sum_{l=i}^{m} |r_{lj}|^2}\, (1 \leq i \leq k, i+1 \leq j \leq n)$$

with probability at least $1-\Delta$.

We prove Theorem 5 in two stages. Firstly, we prove that the updating formulas for B are indeed products of i.i.d. Gaussian matrices and the trailing matrix of A, conditional on Π. Secondly, we use Johnson-Lindenstrauss Theorem 2 and the law of total probability to establish Theorem 5.

2.3.3.1 Correctness of Updating Formulas for B

For correctness, we show that the matrices $\widehat{\Omega}_2$ and $\overline{\Omega}_2$ in updating formulas (2.4) and (2.5) remain i.i.d. Gaussian given permutation matrix Π. Consider s-step partial QRs of $A\Pi$ and $B\Pi$ in equation (2.2) with $R_{11}, \widehat{R}_{11} \in \mathbb{R}^{s\times s}$. Recall that

$$\overline{\Omega} = \Omega\, Q = \begin{pmatrix} \overline{\Omega}_1 & \overline{\Omega}_2 \end{pmatrix}, \quad \widehat{\Omega} = \widehat{Q}^T \overline{\Omega} = \begin{pmatrix} \widehat{\Omega}_1 & \widehat{\Omega}_2 \end{pmatrix}.$$

Given Π, Q is an orthogonal matrix decorrelated with Ω so $\overline{\Omega} = \Omega Q$ remains i.i.d. Gaussian. From equation (2.2),

$$\widehat{Q} \begin{pmatrix} \widehat{R}_{11} & \widehat{R}_{12} \\ & \widehat{R}_{22} \end{pmatrix} = \overline{\Omega} \begin{pmatrix} R_{11} & R_{12} \\ & R_{22} \end{pmatrix}.$$

The orthogonal matrix $\widehat{Q}$ is completely determined by its first s columns, which in turn are completely determined by $\overline{\Omega}_1$, which is decorrelated with $\overline{\Omega}_2$.

It now follows that $\widehat{\Omega}_2 = \widehat{Q}^T \overline{\Omega}_2$ must remain i.i.d. Gaussian given Π. The matrices $\widehat{\Omega}_2$ and $\overline{\Omega}_2$ in updating formulas (2.4) and (2.5) correspond to the special case $s = b$, and thus must remain i.i.d. given permutation matrix Π.

Applying the above argument on each loop in RQRCP, we conclude that, with both updating formulas, the remaining columns of B in each loop are always a product of an i.i.d. Gaussian matrix and the trailing matrix of A, conditional on Π.

Additionally, equation (2.3) implies that $\widehat{\Omega}$ must be a 2×2 block upper triangular matrix since R_{11} is invertible for any $0 \leq s \leq b-1$:

$$\widehat{\Omega} = \begin{pmatrix} \widehat{\Omega}_1 & \widehat{\Omega}_2 \end{pmatrix} = \begin{pmatrix} \widehat{\Omega}_{11} & \widehat{\Omega}_{12} \\ & \widehat{\Omega}_{22} \end{pmatrix},$$

which allows us to write from equation (2.3)

$$\widehat{R}_{22} = \widehat{\Omega}_{22} R_{22}.$$

Thus every trailing matrix in B is a matrix product of an i.i.d. Gaussian matrix with $b + p - s \geq p + 1$ rows and the trailing matrix of A, given Π. Together with above argument on updating formulas, we conclude that *every* pivot computed by Algorithm 6 is based on choosing the column with the largest column l_2 norm of the matrix product of an i.i.d. Gaussian matrix and a trailing matrix of A, conditional on the corresponding column permutation matrix Π.

2.3.3.2 Probability Analysis of Reliability of RQRCP

Our probability analysis is based on the Johnson-Lindenstrauss Theorem 2. We say a vector $x \in \mathbb{R}^d$ satisfies the Johnson−Lindenstrauss condition for given $\varepsilon > 0$ under i.i.d Gaussian matrix $\Omega \in \mathcal{N}(0,1)^{r \times d}$ if

$$(1-\varepsilon)\|x\|_2^2 \leq \left\| \frac{1}{\sqrt{r}} \Omega x \right\|_2^2 \leq (1+\varepsilon)\|x\|_2^2.$$

The Johnson−Lindenstrauss Theorem states that a vector x satisfies the Johnson−Lindenstrauss condition with high probability

$$P\left((1-\varepsilon)\|x\|_2^2 \leq \left\| \frac{1}{\sqrt{r}} \Omega x \right\|_2^2 \leq (1+\varepsilon)\|x\|_2^2 \right) \geq 1 - 2\exp\left(-\frac{(\varepsilon^2 - \varepsilon^3)\, r}{4} \right).$$

Before we prove Theorem 5, we need Lemma 3.

Lemma 3. *Suppose that RQRCP computes an s-step partial QRCP factorization* $A\Pi = QR$ *with* $R = (r_{i,j})$*, then for any* $\varepsilon \in (0,1)$*,*

$$|r_{s,s}| \geq \sqrt{\frac{1-\varepsilon}{1+\varepsilon}} \sqrt{\sum_{l=s}^{m} |r_{lj}|^2}, \quad (s+1 \leq j \leq n) \tag{2.6}$$

with probability at least $1 - \Delta_s$ *where*

$$\Delta_s = 2(n-s+1) \exp\left(\frac{-(\varepsilon^2 - \varepsilon^3)(p+1)}{4}\right).$$

Proof of Lemma 3. Let Π be a random permutation in equation (2.2) with $R_{11}, \widehat{R}_{11} \in \mathbb{R}^{(s-1)\times(s-1)}$, let E_{sj} be the event where $R_{22}(:,j)$ satisfies the Johnson−Lindenstrauss condition. Therefore $E_s = \bigcap_{j=1}^{n-s+1} E_{sj}$ is the event where all columns of R_{22} satisfy the Johnson−Lindenstrauss condition. By correctness of RQRCP, the event E_{sj} for any given Π satisfies the Johnson−Lindenstrauss condition, therefore

$$\begin{aligned} P(E_s|\Pi) &\geq \sum_{j=1}^{n-s+1} P(E_{sj}|\Pi) - (n-s) \\ &\geq \sum_{j=1}^{n-s+1} \left(1 - 2\exp\left(\frac{-(\varepsilon^2-\varepsilon^3)(p+1)}{4}\right)\right) - (n-s) \\ &= 1 - \Delta_s. \end{aligned}$$

We now remove the condition Π by law of total probability,

$$P(E_s) = \sum_{\Pi} P(E_s|\Pi) P(\Pi) \geq 1 - \Delta_s.$$

Denote columns of R_{22} as $r_s, r_{s+1}, \ldots, r_n$, columns of $\widehat{R}_{22}$ as $\widehat{r}_s, \widehat{r}_{s+1}, \ldots, \widehat{r}_n$ *after* the s^{th} column pivot, then $\|\widehat{r}_s\|_2 \geq \|\widehat{r}_j\|_2$ for $s+1 \leq j \leq n$. With probability at least $1-\Delta_s$, event E_s holds so that

$$\frac{\|r_j\|_2}{\|r_s\|_2} \leq \frac{\sqrt{1+\varepsilon}\|\widehat{r}_j\|_2}{\sqrt{1-\varepsilon}\|\widehat{r}_s\|_2} \leq \sqrt{\frac{1+\varepsilon}{1-\varepsilon}}, \text{ for all } s+1 \leq j \leq n,$$

equivalent to inequality (2.6) after one-step QR. □

Proof of Theorem 5. Denote E as the event where all column lengths used in comparison satisfy the Johnson−Lindenstrauss condition so that $E = \bigcap_{s=1}^{k} E_s$, with E_s being the event

where all columns of R_{22} satisfy the Johnson−Lindenstrauss condition in s^{th} step. By Lemma 3,

$$\begin{aligned} P(E) &\geq \sum_{s=1}^{k} P(E_s) - (k-1) \geq \sum_{s=1}^{k} (1-\Delta_s) - (k-1) \\ &= 1 - \sum_{s=1}^{k} \Delta_s \\ &= 1 - \sum_{s=1}^{k} 2(n-s+1) \exp\left(\frac{-(\varepsilon^2 - \varepsilon^3)(p+1)}{4} \right) \\ &\geq 1 - 2nk \exp\left(\frac{-(\varepsilon^2 - \varepsilon^3)(p+1)}{4} \right), \end{aligned}$$

which is at least $1-\Delta$ for the choice of p in Theorem 5. □

Theorem 5 suggests that p needs only to grow logarithmically with n, k and $1/\Delta$ for reliable column selection. For illustration, if we choose $n = 1000, k = 200, \varepsilon = 0.5, \Delta = 0.05$, then $p \geq 508$. However, in practice, p values like $5 \sim 20$ suffice for RQRCP to obtain high quality low-rank approximations.

2.4 Spectrum-Revealing QR Factorization

2.4.1 Introduction

Before we introduce spectrum-revealing QR factorization (SRQR), we need Lemma 4 for partial QR factorizations with column interchanges. Let $\sigma_j(X)$ be the j^{th} largest singular value of any matrix X, and let $\lambda_j(H)$ be the j^{th} largest eigenvalue of any positive semidefinite matrix H.

Lemma 4. *Let $A \in \mathbb{R}^{m\times n}$, with $1 \leq l \leq \min(m,n)$. For any permutation Π, consider a block QR factorization*

$$A\Pi = Q \begin{pmatrix} R_{11} & R_{12} \\ & R_{22} \end{pmatrix},$$

with $R_{11} \in \mathbb{R}^{l\times l}$. Define $\widetilde{R} = (R_{11}\ R_{12})$. For any $1 \leq k \leq l$, denote $\widetilde{R}_k$ the rank-k truncated SVD of $\widetilde{R}$. Therefore,

$$\sigma_j^2(A) \leq \sigma_j^2\left(\widetilde{R}\right) + \|R_{22}\|_2^2, \quad (1 \leq j \leq k), \tag{2.7}$$

$$\left\| A\Pi - Q \begin{pmatrix} \widetilde{R}_k \\ 0 \end{pmatrix} \right\|_2 \leq \sigma_{k+1}(A) \sqrt{1 + \left(\frac{\|R_{22}\|_2}{\sigma_{k+1}(A)} \right)^2}. \tag{2.8}$$

We introduced a new parameter l, in Lemma 4. It will become clear later on that an l-step partial QR with properly chosen Π for an l somewhat larger than k can lead to a much better rank-k approximation than a k-step partial QR.

Proof of Lemma 4. By definition, for $1 \leq j \leq k$,

$$\begin{aligned}
\sigma_j^2(A) &= \lambda_j\left(\Pi^T A^T A \Pi\right) \\
&= \lambda_j\left(\widetilde{R}^T\widetilde{R} + \begin{pmatrix} 0 & 0 \\ 0 & R_{22}^T R_{22} \end{pmatrix}\right) \\
&\leq \lambda_j\left(\widetilde{R}^T\widetilde{R}\right) + \left\|R_{22}^T R_{22}\right\|_2 \\
&= \sigma_j^2\left(\widetilde{R}\right) + \|R_{22}\|_2^2 .
\end{aligned}$$

$$\begin{aligned}
\left\|A\Pi - Q\begin{pmatrix} \widetilde{R}_k \\ 0 \end{pmatrix}\right\|_2^2 &= \left\|Q\begin{pmatrix} \widetilde{R} - \widetilde{R}_k \\ \overline{R} \end{pmatrix}\right\|_2^2 \\
\leq \left\|\widetilde{R} - \widetilde{R}_k\right\|_2^2 + \left\|\overline{R}\right\|_2^2 &\leq \sigma_{k+1}^2(A) + \|R_{22}\|_2^2 ,
\end{aligned}$$

where $\overline{R} \stackrel{def}{=} \begin{pmatrix} 0 & R_{22} \end{pmatrix}$. □

2.4.2 Spectrum-Revealing QR

We exhibit the properties of equation (2.7) in more detail.

$$\sigma_j^2(A) \leq \sigma_j^2\left(\widetilde{R}\right)\left(1 + \frac{\|R_{22}\|_2^2}{\sigma_j^2\left(\widetilde{R}\right)}\right) \leq \sigma_j^2\left(\widetilde{R}\right)\left(1 + \frac{\|R_{22}\|_2^2}{\sigma_k^2\left(\widetilde{R}\right)}\right), \quad \text{or}$$

$$\sigma_j\left(\widetilde{R}\right) \geq \frac{\sigma_j(A)}{\sqrt{1 + \left(\frac{\|R_{22}\|_2}{\sigma_k\left(\widetilde{R}\right)}\right)^2}}, \quad (1 \leq j \leq k). \tag{2.9}$$

Additionally,

$$\begin{aligned}
\sigma_j^2(A) &\leq \sigma_j^2\left(\widetilde{R}\right)\left(1 + \frac{\sigma_j^2(A)}{\sigma_j^2\left(\widetilde{R}\right)}\frac{\|R_{22}\|_2^2}{\sigma_j^2(A)}\right) \\
&\leq \sigma_j^2\left(\widetilde{R}\right)\left(1 + \frac{\left(\sigma_j^2\left(\widetilde{R}\right) + \|R_{22}\|_2^2\right)}{\sigma_j^2\left(\widetilde{R}\right)}\frac{\|R_{22}\|_2^2}{\sigma_j^2(A)}\right) \\
&\leq \sigma_j^2\left(\widetilde{R}\right)\left(1 + \left(1 + \frac{\|R_{22}\|_2^2}{\sigma_k^2\left(\widetilde{R}\right)}\right)\frac{\|R_{22}\|_2^2}{\sigma_j^2(A)}\right), \text{ or}
\end{aligned}$$

$$\sigma_j\left(\widetilde{R}\right) \geq \frac{\sigma_j(A)}{\sqrt{1+\left(1+\frac{\|R_{22}\|_2^2}{\sigma_k^2\left(\widetilde{R}\right)}\right)\frac{\|R_{22}\|_2^2}{\sigma_j^2(A)}}}, \quad (1 \leq j \leq k). \tag{2.10}$$

By the Cauchy interlacing property,

$$\sigma_l\left(R_{11}\right) \leq \sigma_l\left(\widetilde{R}\right) \leq \sigma_l(A) \quad \text{and} \quad \|R_{22}\|_2 \geq \sigma_{l+1}(A).$$

Relations (2.8)(2.9)(2.10) show that we can reveal the leading singular values of A in $\widetilde{R}$ well by computing a Π that ensures

$$\|R_{22}\|_2 \leq O(\sigma_{l+1}(A)). \tag{2.11}$$

Indeed, equation (2.10) further ensures that such permutation Π would make all the leading singular values of $\widetilde{R}$ very close to those of A if the singular values of A decay relatively quickly. Finally, equation (2.8) guarantees that such a permutation would also lead to a high quality approximation measured in 2-norm. In summary, for a successful SRQR, we only need to find a Π that satisfies equation (2.11).

For most matrices in practice, both QRCP and RQRCP compute high-quality SRQRs, with RQRCP being significantly more efficient. We will develop an algorithm for computing an SRQR in 3 stages:

1. Compute an l-step QRCP or RQRCP.
2. Verify condition (2.11).
3. Compute an SRQR if condition (2.11) does not hold.

In next section, we develop a rather efficient scheme to verify if a partial QR factorization satisfies (2.11).

2.4.3 SRQR Verification

Given a QRCP or RQRCP, we discuss an efficient scheme to check whether it satisfies condition (2.11). We define

$$g_1 \overset{def}{=} \frac{\|R_{22}\|_{1,2}}{|\alpha|} \quad \text{and} \quad g_2 \overset{def}{=} |\alpha| \left\|\widehat{R}^{-T}\right\|_{1,2}, \tag{2.12}$$

where $\|X\|_{1,2}$ is the largest column l_2 norm of any X and

$$\widehat{R} \overset{def}{=} \begin{pmatrix} R_{11} & a \\ & \alpha \end{pmatrix} \text{ is a leading submatrix of } R,$$

where we do one step of QRCP or RQRCP on R_{22}. Then

$$\begin{aligned} \|R_{22}\|_2 &= \frac{\|R_{22}\|_2}{\|R_{22}\|_{1,2}} \|R_{22}\|_{1,2} = \tau \sigma_{l+1}(A), \\ \text{where} \quad \tau &\stackrel{def}{=} g_1 g_2 \frac{\|R_{22}\|_2}{\|R_{22}\|_{1,2}} \frac{\left\|\widehat{R}^{-T}\right\|_{1,2}^{-1}}{\sigma_{l+1}(A)}. \end{aligned} \tag{2.13}$$

While τ depends on A and Π, it can be upper bounded as

$$\tau = g_1 g_2 \frac{\|R_{22}\|_2}{\|R_{22}\|_{1,2}} \frac{\left\|\widehat{R}^{-T}\right\|_{1,2}^{-1}}{\sigma_{l+1}\left(\widehat{R}\right)} \frac{\sigma_{l+1}\left(\widehat{R}\right)}{\sigma_{l+1}(A)} \leq g_1 g_2 \sqrt{(l+1)(n-l)}.$$

On the other hand,

$$\begin{aligned} \|R_{22}\|_2 &= \frac{\|R_{22}\|_2}{\|R_{22}\|_{1,2}} \|R_{22}\|_{1,2} \leq \widehat{\tau} \sigma_l\left(\widetilde{R}\right), \\ \text{where} \quad \widehat{\tau} &\stackrel{def}{=} g_1 g_2 \frac{\|R_{22}\|_2}{\|R_{22}\|_{1,2}} \frac{\left\|R_{11}^{-T}\right\|_{1,2}^{-1}}{\sigma_l\left(\widetilde{R}\right)}. \end{aligned} \tag{2.14}$$

While $\widehat{\tau}$ depends on A and Π, it can be upper bounded as

$$\widehat{\tau} = g_1 g_2 \frac{\|R_{22}\|_2}{\|R_{22}\|_{1,2}} \frac{\left\|R_{11}^{-T}\right\|_{1,2}^{-1}}{\sigma_l(R_{11})} \frac{\sigma_l(R_{11})}{\sigma_l\left(\widetilde{R}\right)} \leq g_1 g_2 \sqrt{l(n-l)}. \tag{2.15}$$

It is typical for the first two ratios in equations (2.14) and (2.15) to be of order $O(1)$, and the last ratio $o(1)$. Thus, even though the last upper bounds in equations (2.14) and (2.15) grow with matrix dimensions, τ is small to modest in practice.

For notational simplicity, we further define $\overline{\tau} \stackrel{def}{=} \widehat{\tau} \frac{\sigma_l\left(\widetilde{R}\right)}{\sigma_k\left(\widetilde{R}\right)}$. Plugging equation (2.14) into equation (2.9), we obtain

$$\sigma_j\left(\widetilde{R}\right) \geq \frac{\sigma_j(A)}{\sqrt{1+\overline{\tau}^2}}, \quad (1 \leq j \leq k). \tag{2.16}$$

Plugging both equations (2.13) and (2.14) into equation (2.10),

$$\sigma_j\left(\widetilde{R}\right) \geq \frac{\sigma_j(A)}{\sqrt{1+\tau^2\left(1+\overline{\tau}^2\right)\left(\frac{\sigma_{l+1}(A)}{\sigma_j(A)}\right)^2}}, \quad (1 \leq j \leq k).$$

Combining the last equation with (2.16), for $1 \leq j \leq k$,

$$\sigma_j\left(\widetilde{R}\right) \geq \frac{\sigma_j(A)}{\sqrt{1+\min\left(\overline{\tau}^2, \tau^2\left(1+\overline{\tau}^2\right)\left(\frac{\sigma_{l+1}(A)}{\sigma_j(A)}\right)^2\right)}}. \tag{2.17}$$

Equation (2.17) shows that, under definition (2.12) and with $\widetilde{R}$, we can reveal at least a dimension dependent fraction of all the leading singular values of A and indeed approximate them very accurately in case they decay relatively quickly.

Finally, plugging equation (2.13) into equation (2.8),

$$\left\| A\Pi - Q\begin{pmatrix} \widetilde{R}_k \\ 0 \end{pmatrix}\right\|_2 \leq \sigma_{k+1}(A)\sqrt{1+\tau^2\left(\frac{\sigma_{l+1}(A)}{\sigma_{k+1}(A)}\right)^2}. \tag{2.18}$$

Equation (2.18) shows that we can compute a rank-k approximation that is up to a factor of $\sqrt{1+\tau^2\left(\frac{\sigma_{l+1}(A)}{\sigma_{k+1}(A)}\right)^2}$ from optimal. In situations where singular values of A decay relatively quickly, our rank-k approximation is about as accurate as the truncated SVD with a choice of l such that

$$\frac{\sigma_{l+1}(A)}{\sigma_{k+1}(A)} = o(1).$$

2.4.4 Spectrum-Revealing Bounds of QRCP and RQRCP

We develop spectrum-revealing bounds of QRCP and RQRCP. It comes down to estimating upper bounds on g_1 and g_2. We need Lemma 5, presented below without a proof:

Lemma 5. *Let $W = (w_{i,j}) \in \mathbb{R}^{n\times n}$ be an upper or lower triangular matrix with $w_{i,i} = 1$ and $|w_{ij}| \leq c$ $(i \neq j)$. Then*

$$\left\|W^{-1}\right\|_{1,2} \leq \left\|W^{-1}\right\|_1 \leq (1+c)^{n-1}.$$

For QRCP, $g_1 = 1$ since $|\alpha|$ is the largest column l_2 norm in the trailing matrix R_{22}. For g_2, decompose $\widehat{R} = DW$ where D is the diagonal of $\widehat{R}$ and W satisfies Lemma 5 with $c = 1$.

$$g_2 = |\alpha|\left\|D^{-T}W^{-T}\right\|_{1,2} \leq \left\|W^{-T}\right\|_{1,2} \leq \left\|W^{-T}\right\|_1 \leq 2^l.$$

For RQRCP, we find upper bounds for g_1 and g_2. According to our probability analysis of RQRCP, the following inequalities are valid with probability $1-\Delta$ for given ε.

$$|\alpha| \geq \sqrt{\frac{1-\varepsilon}{1+\varepsilon}}\left\|R_{22}\right\|_{1,2} \Rightarrow g_1 = \frac{\left\|R_{22}\right\|_{1,2}}{|\alpha|} \leq \sqrt{\frac{1+\varepsilon}{1-\varepsilon}},$$

$$g_2^2 = \alpha^2\left\|\begin{pmatrix} R_{11}^{-T} & \\ -\frac{1}{\alpha}a^T R_{11}^{-T} & \frac{1}{\alpha}\end{pmatrix}\right\|_{1,2}^2 \leq \alpha^2\max\left\{\left|\left|R_{11}^{-T}\right|\right|_{1,2}^2 + \left\|-\frac{1}{\alpha}a^T R_{11}^{-T}\right\|_{1,2}^2, \frac{1}{\alpha^2}\right\}.$$

We decompose $R_{11} = DW$ where D is the diagonal of R_{11} and W satisfies Lemma 5 with $c = \sqrt{\frac{1+\varepsilon}{1-\varepsilon}}$. Assume the smallest entry in D is t, then t satisfies $t \geq \sqrt{\frac{1-\varepsilon}{1+\varepsilon}}\sigma_l(A)$,

$$\left\|R_{11}^{-T}\right\|_{1,2}^2 = \left\|D^{-1}\left(W^T\right)^{-1}\right\|_{1,2}^2 \leq \frac{1}{t^2}\left\|\left(W^T\right)^{-1}\right\|_{1,2}^2 \leq \frac{1}{\sigma_l^2(A)} \cdot \left(\frac{1+\varepsilon}{1-\varepsilon}\right) \cdot \left(1+\sqrt{\frac{1+\varepsilon}{1-\varepsilon}}\right)^{2l-2},$$

$$\left\|-\frac{1}{\alpha}a^T R_{11}^{-T}\right\|_{1,2}^2 = \frac{1}{\alpha^2}\left\|(D^{-1}a)^T\left(W^T\right)^{-1}\right\|_{1,2}^2 \leq \frac{1}{\alpha^2} \cdot \left(\frac{1+\varepsilon}{1-\varepsilon}\right) \cdot \left(1+\sqrt{\frac{1+\varepsilon}{1-\varepsilon}}\right)^{2l-2}.$$

It follows that

$$\begin{aligned} g_2^2 &\leq \frac{1+\varepsilon}{1-\varepsilon} \cdot \left(1+\sqrt{\frac{1+\varepsilon}{1-\varepsilon}}\right)^{2l-2} \cdot \left(\frac{\alpha^2}{\sigma_l^2(A)}+1\right) \\ &\leq \frac{1+\varepsilon}{1-\varepsilon} \cdot \left(1+\sqrt{\frac{1+\varepsilon}{1-\varepsilon}}\right)^{2l-2} \cdot \left(\frac{1+\varepsilon}{1-\varepsilon}+1\right) \\ \Rightarrow\ g_2 &\leq \frac{\sqrt{2(1+\varepsilon)}}{1-\varepsilon}\left(1+\sqrt{\frac{1+\varepsilon}{1-\varepsilon}}\right)^{l-1}. \end{aligned}$$

In summary,

$$g_1 \leq \begin{cases} 1 & \text{for QRCP,} \\ \sqrt{\frac{1+\varepsilon}{1-\varepsilon}} & \text{for RQRCP with probability } 1-\Delta. \end{cases} \tag{2.19}$$

$$g_2 \leq \begin{cases} 2^l & \text{for QRCP,} \\ \frac{\sqrt{2(1+\varepsilon)}}{1-\varepsilon}\left(1+\sqrt{\frac{1+\varepsilon}{1-\varepsilon}}\right)^{l-1} & \text{for RQRCP with probability } 1-\Delta. \end{cases} \tag{2.20}$$

where Δ is definded in Theorem 5.

2.4.5 An Algorithm to Compute SRQR

The parameter g_1 is always modest in equation (2.19). Despite the exponential nature of the upper bound on the parameter g_2 in equation (2.20), g_2 has always been known to be modest with real data matrices. However, it can be exceptionally large for contrived pathological matrices [65]. This motivates Algorithm 7 for computing SRQR. Algorithm 7 computes a partial QR factorization with RQRCP. It then efficiently estimates g_2 and performs additional column interchanges for a guaranteed SRQR only if g_2 is too large.

After RQRCP, Algorithm 7 uses a randomized scheme to quickly and reliably estimate g_2 and compares it to a user-defined tolerance $g > 1$. Algorithm 7 exits if the estimated $g_2 \leq g$. Otherwise, it swaps the $\imath^{\text{th}}$ and $(l+1)^{\text{st}}$ columns of A and R. Each swap will make

Algorithm 7 Spectrum-Revealing QR factorization (SRQR)

Inputs:
A is $m \times n$ matrix,
$l \geq k$, the approximate rank, $1 \leq k \leq \min(m, n)$
Block size b. Oversampling size p.
$g > 1$ is user defined tolerance for g_2
Outputs:
Q is $m \times m$ orthogonal matrix
R is $m \times n$ upper trapezoidal matrix
Π is $n \times n$ permutation matrix such that $A\Pi = QR$
Algorithm:
Compute Q, R, Π with Algorithm 6
Compute squared 2-norm of the columns of $B(:, l+1:n) : \widehat{r}_i \ (l+1 \leq i \leq n)$
Approximate squared 2-norm of the columns of $A(l+1:m, l+1:n) : r_i = \frac{\widehat{r}_i}{b+p} \ (l+1 \leq i \leq n)$
$\imath = \mathbf{argmax}_{l+1 \leq i \leq n}\{r_i\}$
Swap $\imath$-th and $(l+1)$-st columns of A, Π, r
One-step QR factorization of $A(l+1:m, l+1:n)$
$|\alpha| = R_{l+1,l+1}$
$r_i = r_i - A(l+1, i)^2 \ (l+2 \leq i \leq n)$
Generate random matrix $\Omega \in \mathcal{N}(0,1)^{d \times (l+1)} \ (d \ll l)$
Compute $g_2 = |\alpha| \left\| \widehat{R}^{-T} \right\|_{1,2} \approx \frac{|\alpha|}{\sqrt{d}} \left\| \Omega \widehat{R}^{-T} \right\|_{1,2}$

while $g_2 > g$ **do**
$\imath = \mathbf{argmax}_{1 \leq i \leq l+1}\{i\text{th column } l_2 \text{ norm of } \Omega \widehat{R}^{-T}\}$
Swap $\imath$-th and $(l+1)$-st columns of A and Π in a Round Robin rotation
Givens-rotate R back into upper-trapezoidal form
$r_{l+1} = R_{l+1,l+1}^2$
$r_i = r_i + A(l+1, i)^2 \ (l+2 \leq i \leq n)$
$\imath = \mathbf{argmax}_{l+1 \leq i \leq n}\{r_i\}$
Swap $\imath$-th and $(l+1)$-st columns of A, Π, r
One-step QR factorization of $A(l+1:m, l+1:n)$
$|\alpha| = R_{l+1,l+1}$
$r_i = r_i - A(l+1, i)^2 \ (l+2 \leq i \leq n)$
Generate random matrix $\Omega \in \mathcal{N}(0,1)^{d \times (l+1)} (d \ll l)$
Compute $g_2 = |\alpha| \left\| \widehat{R}^{-T} \right\|_{1,2} \approx \frac{|\alpha|}{\sqrt{d}} \left\| \Omega \widehat{R}^{-T} \right\|_{1,2}$
end while

the i^{th} column out of the upper-triangular form in R. A round-robin rotation is applied to the columns of A and R, followed by a quick sequence of Givens rotations left multiplied to R to restore its upper-triangular form. The while loop in Algorithm 7 will stop, after a finite number of swaps, leading to a permutation that ensures $g_2 \leq g$ with high probability.

The cost of estimating g_2 is $O(d\,l^2)$, and the cost for one extra swap is $O(n\,l)$. Matrix R can be SVD-compressed into a rank-k matrix optionally, at cost of $O(n\,l^2)$. There is no practical need for extra swaps for real data matrices, making Algorithm 7 only slightly more expensive than RQRCP in general. At most a few swaps are enough, adding an insignificant amount of computation to the cost of RQRCP, for pathological matrices like the Kahan matrix [65].

2.5 Experimental Performance

Experiments in sections 2.5.1, 2.5.2, and 2.5.4 were performed on a laptop with 2.7 GHz Intel Core i5 CPU and 8GB of RAM. Experiments in section 2.5.3 were performed on multiple nodes of the NERSC machine Edison, where each node has two 12-core Intel processors. Codes in sections 2.5.1, 2.5.2 were in Fortran and based on LAPACK. Codes in section 2.5.3 were in Fortran and C and based on ScaLAPACK. Codes in section 2.5.4 were in Matlab.

Codes are available at https://math.berkeley.edu/∼jwxiao/.

2.5.1 Approximation Quality Comparison on Datasets

In this section, we compare the approximation quality of the low-rank approximations computed by SRQR, QRCP, and QR on practical datasets. We only list the results on two of them: Human Activities and Postural Transitions (HAPT) [92] and the MNIST database (MNIST) [73], while similar performance can be observed in others. We choose block size $b = 64$, oversampling size $p = 10$, tolerance $g = 5.0$ and set $l = k$ in our SRQR implementation. We compare the residual errors in figures 2.1 and 2.3, where QR is not doing a great job, while QRCP and SRQR are doing equally well. We compare the run time in figures 2.2 and 2.4, where SRQR is much faster than QRCP and close to QR.

In other words, SRQR computes low-rank approximations comparable to those computed by QRCP in quality, yet at a performance near that of QR.

2.5.2 Comparison on a Pathological Matrix: the Kahan Matrix

In this section we compare SRQR and QRCP on the Kahan matrix [65]. For the Kahan matrix, QRCP won't do any columns interchanges so it's equivalent to QR. We choose $c = 0.285, s = \sqrt{0.9999 - c^2}, n = 96, 192, 384$ and $k = n - 1$. We choose block size $b = 64$, oversampling size $p = 10$, tolerance $g = 5.0$ and set $l = k$ in our SRQR implementation. From the relative residual errors summarized in table 2.1, we can see that SRQR is able to compute a much better low-rank approximation.

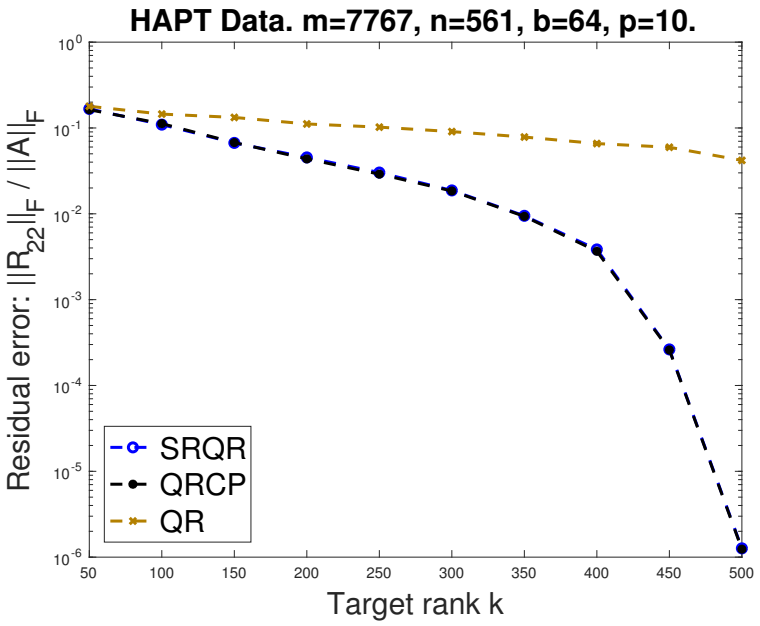

Figure 2.1: Approximation quality comparison on HAPT.

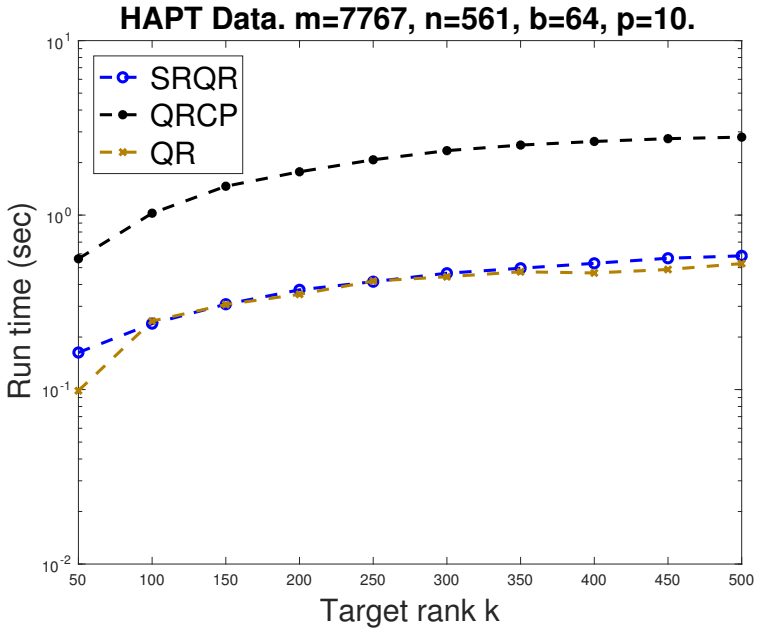

Figure 2.2: Run time comparison on HAPT.

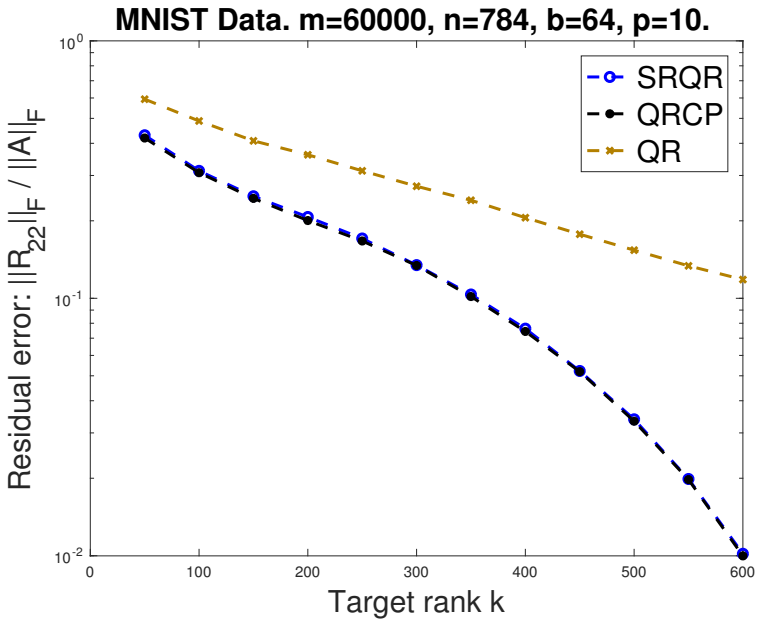

Figure 2.3: Approximation quality comparison on MNIST.

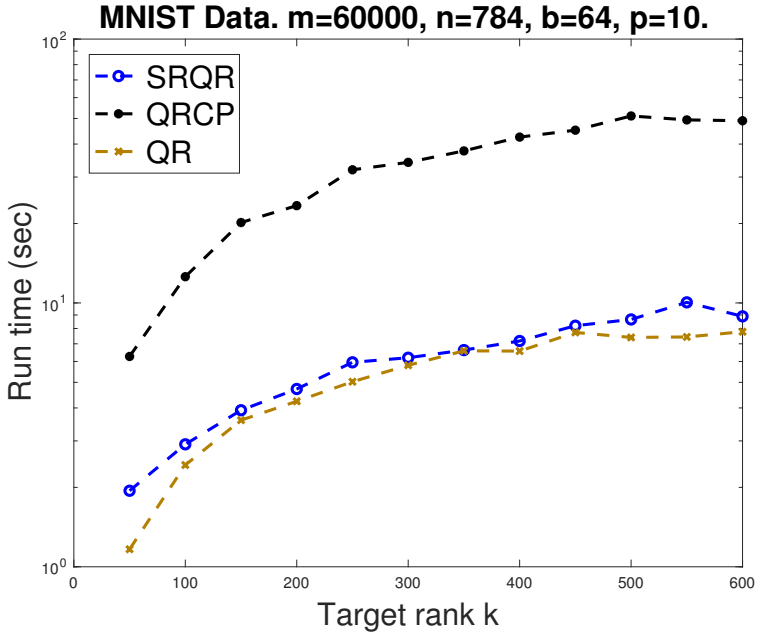

Figure 2.4: Run time comparison on MNIST.

n	k	SRQR $\left(\frac{\|R_{22}\|_F}{\|A\|_F}\right)$	QRCP $\left(\frac{\|R_{22}\|_F}{\|A\|_F}\right)$
96	95	2.449E-13	1.808E-03
192	191	1.031E-25	2.169E-05
384	383	2.585E-50	4.414E-09

Table 2.1: Residual $\frac{\|R_{22}\|_F}{\|A\|_F}$ comparison on the Kahan matrix.

index	SRQR $\left(\frac{\sigma_j(R_{11})}{\sigma_j(A)}\right)$	QRCP $\left(\frac{\sigma_j(R_{11})}{\sigma_j(A)}\right)$
187	1.000	0.9942
188	1.000	0.9932
189	1.000	0.9916
190	1.000	0.9883
191	1.000	0.2806E-17

Table 2.2: Singular value approximation ratio $\frac{\sigma_j(R_{11})}{\sigma_j(A)}$.

The singular value ratios $\frac{\sigma_j(R_{11})}{\sigma_j(A)}$ never exceeds 1 for any approximation, but we would like them to be close to 1 for a reliable spectrum-revealing QR factorization. For the Kahan matrix where $n = 192$ and $k = 191$, table 2.2 demonstrates that QRCP failed to do so for the index 191 singular value, whereas SRQR succeeded for all singular values.

The additional run time required to compute g_2 is negligible. In our extensive computations with practical data in machine learning and other applications, g_2 always remains modest and never triggers subsequent SRQR column swaps. Nonetheless, computing g_2 serves as an insurance policy against potential SRQR mistakes by QRCP or RQRCP.

2.5.3 Run Time Comparison in Distributed Memory Machines

In this section, we compare run time and strong scaling of RQRCP against ScaLAPACK QRCP routines (PDGEQPF and PDGEQP3), ScaLAPACK QR routine PDGEQRF on distributed memory machines. PDGEQP3 [89] is not yet incorporated into ScaLAPACK, but it's usually more efficient than PDGEQPF, so is also included in the comparison.

The way the data is distributed over the memory hierarchy of a computer is of fundamental importance to load balancing and software reuse. ScaLAPACK uses a block cyclic data distribution in order to reduce overhead due to load imbalance and data movement. Block-partitioned algorithms are used to maximize local processor performance and ensure high levels of data reuse.

Now we discuss how we parallelize RQRCP on a distributed memory machine based on ScaLAPACK. After we distribute A to all processors, we use PDGEMM to compute $B = \Omega A$. In each loop, we use our version of PDGEQPF to compute a partial QRCP factorization of B, meanwhile, we swap the columns of A according to the pivots found on B. In our

implementation, A and B share the same column blocking factor NB, therefore we don't introduce much extra communication costs since the same column processors are sending and receiving messages while doing the swaps on both A and B. After we swap the pivoted columns to the leading position of the trailing matrix of A, we use PDGEQRF to perform a panel QR. Next, we use PDLARFT and PDLARFB to apply the transpose of an orthogonal matrix in a block form to the trailing matrix of A. At the end of each loop, we update the remaining columns of B using updating formula (2.4). See algorithm 8.

Algorithm 8 Parallel RQRCP

Inputs:
A is $m \times n$ matrix, k is approximate rank, $1 \leq k \leq \min(m, n)$
Outputs:
Q is $m \times m$ orthogonal matrix
R is $m \times n$ upper trapezoidal matrix
Π is $n \times n$ permutation matrix such that $A\Pi = QR$
Algorithm:
Determine block size b and oversampling size $p \geq 0$
Distribute A to processors using block-cyclic layout
Generate i.i.d Gaussian matrix $\Omega \in \mathcal{N}(0,1)^{(b+p)\times m}$
Compute $B = \Omega A$ using PDGEMM, initialize $\Pi = I_n$
for $i = 1 : b : k$ **do**
$b = \min(b, k - i + 1)$
Run partial version of PDGEQPF (QRCP) on $B(:, i : n)$, meanwhile apply the swaps to $A(:, i : n)$ and Π
PDGEQRF (QR) on $A(i : m, i : i + b - 1) = \widetilde{Q}\widetilde{R}$
Use PDLARFT and PDLARFB to apply $\widetilde{Q}^T$ in a blocked form to $A(i : m, i + b : n)$
Update $B(1 : b, i + b : n) = B(1 : b, i + b : n) - B(1 : b, i : j + b - 1)(A(i : i + b - 1.i : i + b - 1))^{-1}A(i : i + b - 1, i + b : n)$
end for
Q is the product of $\widetilde{Q}$, R = upper trapezoidal part of A

We choose block size $b = 64$ and oversampling size $p = 10$ in our RQRCP implementation. For all routines, we use an efficient data distribution by setting distribution block size MB = NB = 64 and using a square processor grid, i.e., $P_r = P_c$, as recommended in [11]. Since the run time is only dependent on the matrix size but not the actual magnitude of the entries, we do the comparison on random matrices with different sizes, with $n = 20000, 50000$ and 200000. See figures 2.5 through 2.10.

The run time of RQRCP is always much better than that of ScaLAPACK QRCP routines and relatively close to that of ScaLAPACK QR routine. For very large scale low-rank approximations with a limited number of processors, distributed RQRCP is likely the method of choice.

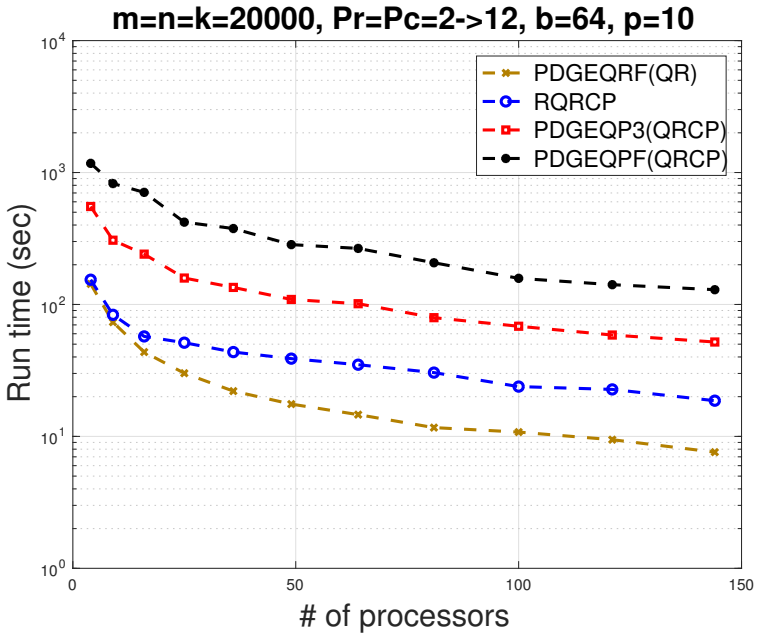

Figure 2.5: Run time comparison on distributed memory machines, n=20000.

However, RQRCP remains less than ideally strong scaled. There are two possible ways to improve our parallel RQRCP algorithm and implementation in our future work.

- One bottleneck of our RQRCP parallel implementation is communication cost incurred by partial QRCP on the compressed matrix B. These communication costs are negligible on share memory machines or in distributed memory machines with a relatively small number of nodes. On distributed memory machines with a large number of nodes, many of them will be idle during partial QRCP computations on B, causing the gap between RQRCP and PDGEQRF (QR) run time lines in figures 2.7 and 2.9. The communication costs on B can be possibly reduced by using QR with tournament pivoting [26] in place of partial QRCP.
- Another possible improvement of our RQRCP parallel implementation is to replace PDGEQRF (QR) with Tall Skinny QR (TSQR) [27] in the panel QR factorization.

The parallelization of SRQR in ScaLAPACK is also in our future work.

2.5.4 SRQR Based CUR and CX Matrix Decomposition

The CUR and CX matrix decompositions are two important low-rank matrix approximation and data analysis techniques, and have been widely discussed in [115, 44, 12]. A CUR matrix decomposition algorithm seeks to find c columns of A to form $C \in \mathbb{R}^{m \times c}$, r rows

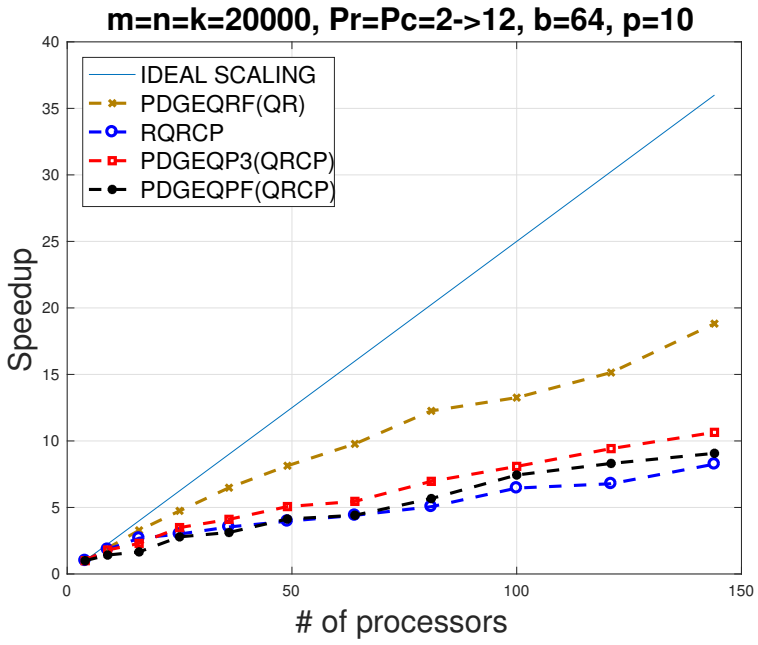

Figure 2.6: Strong scaling comparison on distributed memory machines, n=20000.

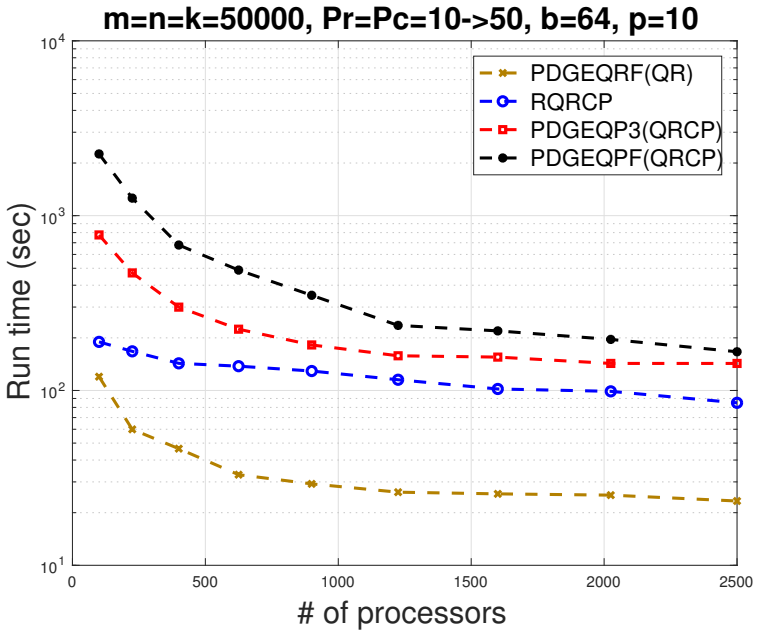

Figure 2.7: Run time comparison on distributed memory machines, n=50000.

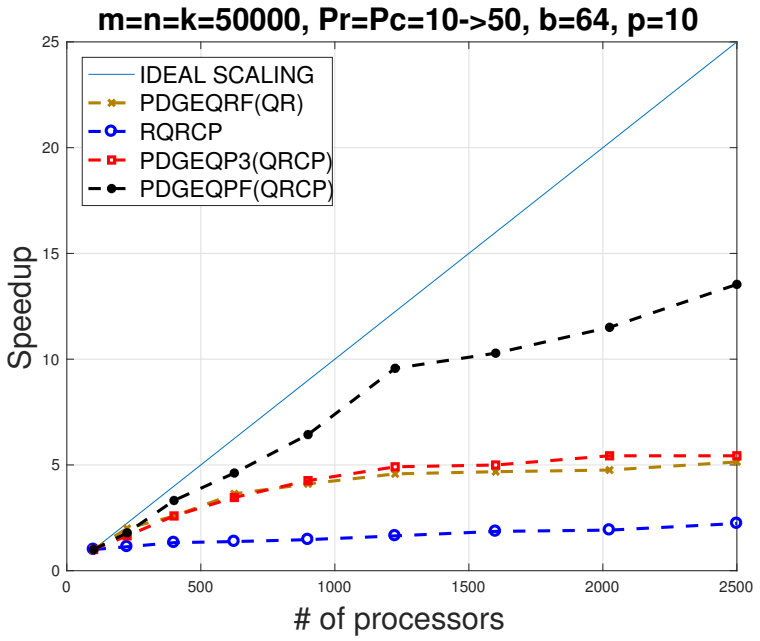

Figure 2.8: Strong scaling comparison on distributed memory machines, n=50000.

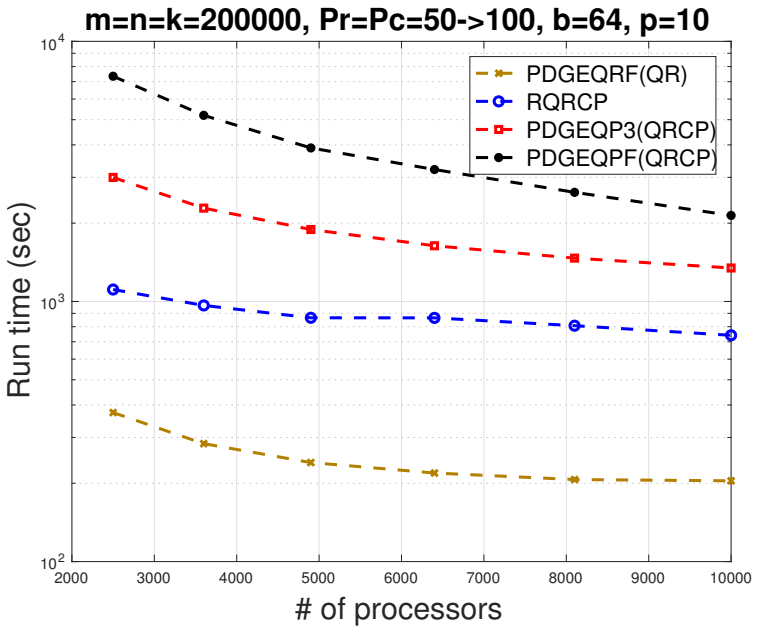

Figure 2.9: Run time comparison on distributed memory machines, n=200000.

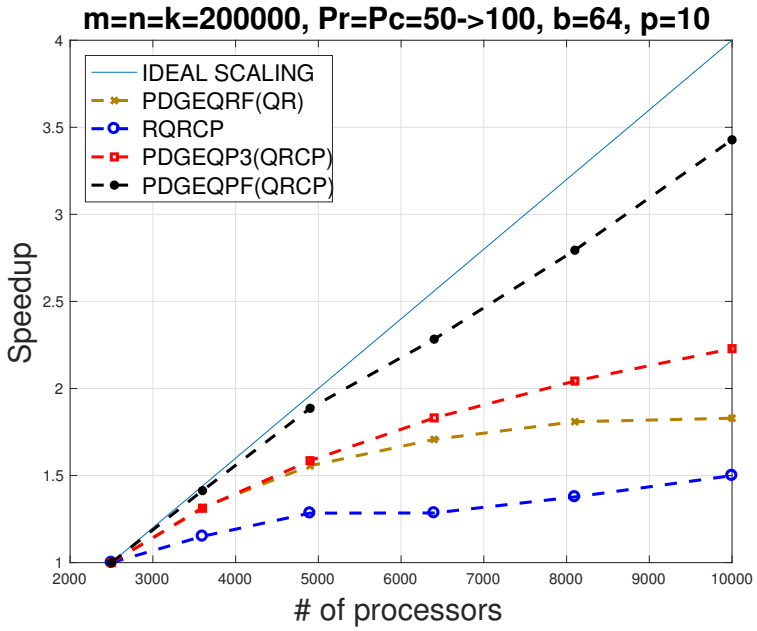

Figure 2.10: Strong scaling comparison on distributed memory machines, n=200000.

of A to form $R \in \mathbb{R}^{r\times n}$, and an intersection matrix $U \in \mathbb{R}^{c\times r}$ such that $\|A - CUR\|_F$ is small. One particular choice of U is $C^\dagger A R^\dagger$, which is the solution to $\min_X \|CXR - A\|_F^2$. A CX decomposition algorithm seeks to find c columns of A to form $C \in \mathbb{R}^{m\times c}$ and a matrix $X \in \mathbb{R}^{c\times n}$ such that $\|A - CX\|_F$ is small. One particular choice of X is $C^\dagger A$, which is the solution to $\min_X \|CX - A\|_F^2$.

GitHub repository [99] provides a Matlab library for CUR matrix decomposition. These CUR matrix decomposition algorithms can be modified to compute a CX matrix decomposition. Since the crucial component of CUR and CX matrix decompositions is column/row selection, we can use SRQR to find the pivots and hence compute these decompositions. In this experiment, we compare SRQR against the state-of-the-art CUR and CX matrix decomposition algorithms.

We compare the approximation quality and run time on a kernel matrix A of size 4177×4177 computed on Abalone Data Set [7], for approximate rank $k = 200 = l$ with different numbers of columns and rows used. In Figures 2.11, 2.12, 2.13, and 2.14, the x-axis stands for the number of columns and rows we choose for the CUR or CX matrix decomposition. The most efficient and effective method in the Matlab library is the near optimal method [115]. The near-optimal algorithm consists of three steps: the approximate SVD via random projection [12, 52], the dual set sparsification algorithm [12], and the adaptive sampling algorithm [28]. We can see that SRQR and the near optimal method are obtaining much better low-rank approximations than the other three methods, while SRQR is much faster than the near optimal method. The other three methods are not competitive, where DetUCS

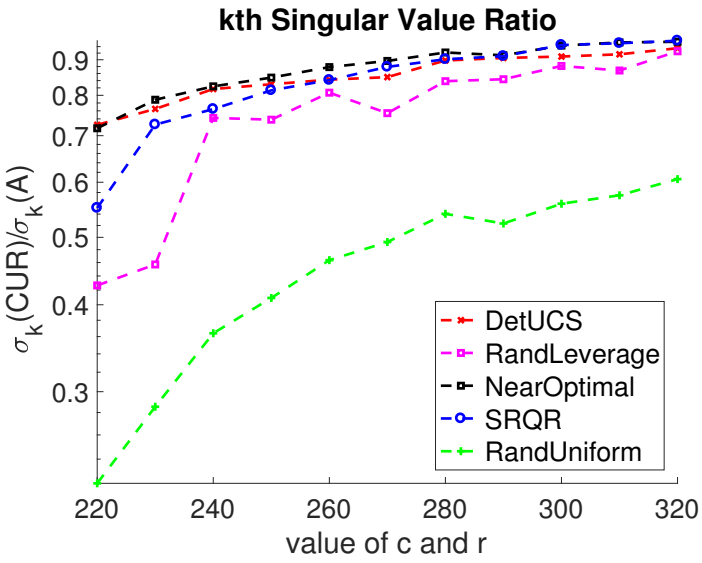

Figure 2.11: Approximation quality comparison, CUR based algorithms.

stands for deterministic unweighted column selection based algorithm; RandLeverage stands for subspace sampling algorithm; RandUniform stands for naive uniform sampling algorithm. For more details about these algorithms, please refer to [44, 115].

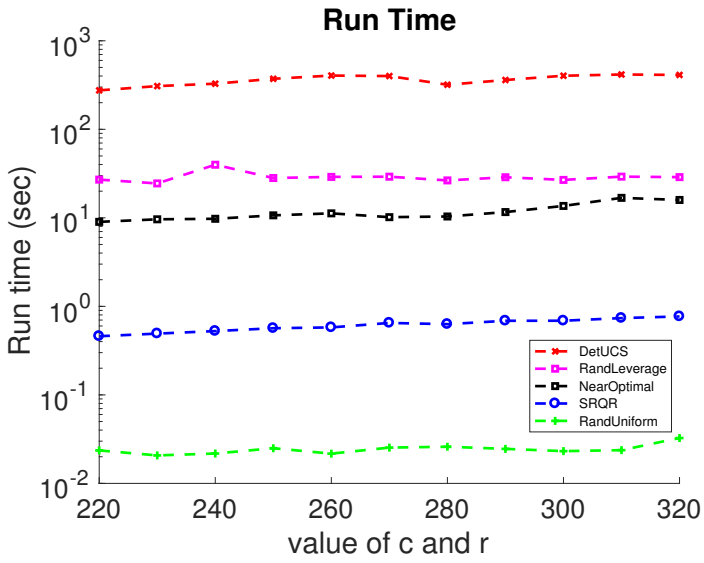

Figure 2.12: Run time comparison, CUR based algorithms.

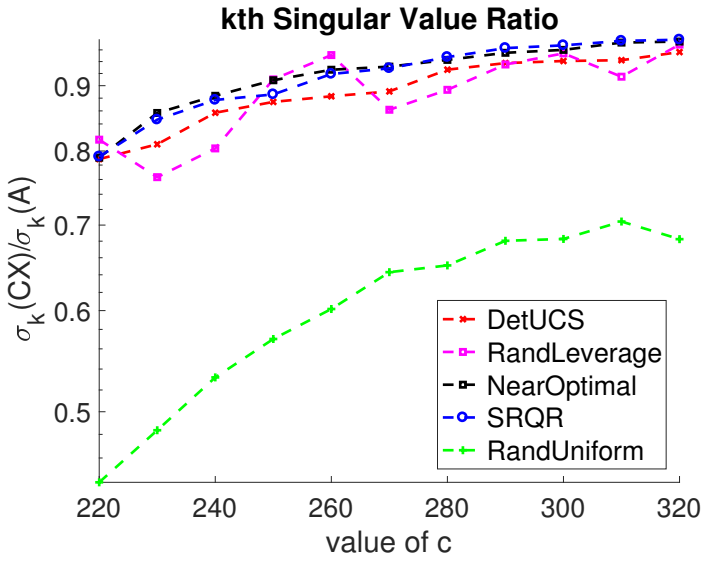

Figure 2.13: Approximation quality comparison, CX based algorithms.

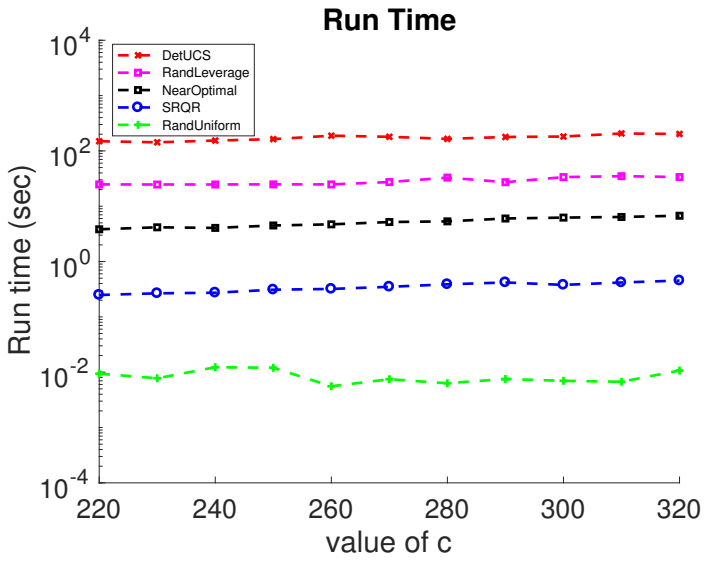

Figure 2.14: Run time comparison, CX based algorithms.

Chapter 3

Flip-Flop SRQR

3.1 Introduction

The singular value decomposition (SVD) of a matrix $A \in \mathbb{R}^{m\times n}$ is the factorization of A into the product of three matrices $A = U\Sigma V^T$ where $U = (u_1, \cdots, u_m) \in \mathbb{R}^{m\times m}$ and $V = (v_1, \cdots, v_n) \in \mathbb{R}^{n\times n}$ are orthogonal singular vector matrices and $\Sigma \in \mathbb{R}^{m\times n}$ is a rectangular diagonal matrix with non-increasing non-negative singular values $\sigma_i \, (1 \leq i \leq \min(m, n))$ on the diagonal. The SVD has become a critical analytic tool in large data analysis and machine learning [1, 36, 86]. [1]

Let $\mathrm{Diag}(x)$ denote the diagonal matrix with vector $x \in \mathbb{R}^n$ on its diagonal. For any $1 \leq k \leq \min(m, n)$, the rank-k truncated SVD of A is defined by

$$A_k \stackrel{def}{=} (u_1, \cdots, u_k) \, \mathrm{Diag}(\sigma_1, \cdots, \sigma_k) \, (v_1, \cdots, v_k)^T .$$

The rank-k truncated SVD turns out to be the best rank-k approximation to A, as explained by Therorem 6.

Theorem 6. *(Eckart–Young–Mirsky Theorem [35, 47]).*

$$\min_{rank(C)\leq k} \|A - C\|_2 = \|A - A_k\|_2 = \sigma_{k+1},$$

$$\min_{rank(C)\leq k} \|A - C\|_F = \|A - A_k\|_F = \sqrt{\sum_{j=k+1}^{\min(m,n)} \sigma_j^2},$$

where $\|\cdot\|_2$ and $\|\cdot\|_F$ denote the l_2 operator norm and the Frobenius norm respectively.

However, due to the prohibitive costs in computing the rank-k truncated SVD, in practical applications one typically computes a rank-k approximate SVD which satisfies some tolerance

[1] Materials in this chapter are mainly from the paper titled Low-Rank Matrix Approximations with Flip-Flop Spectrum-Revealing QR Factorization [39], which is posted on arXiv.

requirements [29, 48, 52, 71, 102]. Rank-k approximate SVD has been applied to many research areas including principal component analysis (PCA) [64, 93], web search models [67], information retrieval [8, 43], and face recognition [85, 108].

Among assorted SVD approximation algorithms, the pivoted QLP decomposition proposed by Stewart [102] is an effective and efficient one. The pivoted QLP decomposition is obtained by computing a QR factorization with column pivoting [13, 46] on A to get an upper triangular factor R and then computing an LQ factorization on R to get a lower triangular factor L. Stewart's key numerical observation is that the diagonal elements of L track the singular values of A with "considerable fidelity" no matter the matrix A. The pivoted QLP decomposition is extensively analyzed in Huckaby and Chan [61, 62]. More recently, Duersch and Gu developed a much more efficient variant of the pivoted QLP decomposition, TUXV, and demonstrated its remarkable quality as a low-rank approximation empirically without a rigid justification of TUXV's success theoretically [33].

In this paper, we present Flip-Flop SRQR, a slightly different variant of TUXV of Duersch and Gu [33]. Like TUXV, Flip-Flop SRQR performs most of its work in computing a partial QR factorization using truncated randomized QRCP (TRQRCP) and a partial LQ factorization. Unlike TUXV, however, Flip-Flop SRQR also performs additional computations to ensure a spectrum-revealing QR factorization (SRQR) [121] before the partial LQ factorization.

We demonstrate the remarkable theoretical quality of this variant as a low-rank approximation, and its high competitiveness with state-of-the-art low-rank approximation methods in real world applications in both low-rank tensor compression [23, 69, 95, 109] and nuclear norm minimization [14, 75, 80, 87, 104].

The rest of this chapter is organized as follows: In Section 3.2 we introduce the TRQRCP algorithm, the spectrum-revealing QR factorization, low-rank tensor compression, and nuclear norm minimization. In Section 3.3, we introduce Flip-Flop SRQR and analyze its computational costs and low-rank approximation properties. In Section 3.4, we present numerical experimental results comparing Flip-Flop SRQR with state-of-the-art low-rank approximation methods. In Section 3.5, we briefly introduce approximate SVD with randomized subspace iteration method and also analyze its complexity.

3.2 Preliminaries and Background

3.2.1 Partial QRCP

The QR factorization of a matrix $A \in \mathbb{R}^{m\times n}$ is $A = QR$ with orthogonal matrix $Q \in \mathbb{R}^{m\times m}$ and upper trapezoidal matrix $R \in \mathbb{R}^{m\times n}$, which can be computed by LAPACK [2] routine xGEQRF, where x stands for the matrix data type. The standard QR factorization is not suitable for some practical situations where either the matrix A is rank deficient or only representative columns of A are of interest. Usually the QR factorization with column pivoting (QRCP) is adequate for the aforementioned situations except a few rare examples

such as the Kahan matrix [47]. Given a matrix $A \in \mathbb{R}^{m\times n}$, the QRCP of matrix A has the form

$$A\Pi = QR,$$

where $\Pi \in \mathbb{R}^{n\times n}$ is a permutation matrix, $Q \in \mathbb{R}^{m\times m}$ is an orthogonal matrix, and $R \in \mathbb{R}^{m\times n}$ is an upper trapezoidal matrix. QRCP can be computed by LAPACK [2] routines xGEQPF and xGEQP3, where xGEQP3 is a more efficient blocked implementation of xGEQPF. For given target rank k $(1 \le k \le \min(m,n))$, the partial QRCP factorization has a 2×2 block form

$$A\Pi = Q\begin{pmatrix} R_{11} & R_{12} \\ & R_{22}\end{pmatrix} = \begin{pmatrix} Q_1 & Q_2\end{pmatrix}\begin{pmatrix} R_{11} & R_{12} \\ & R_{22}\end{pmatrix}, \tag{3.1}$$

where $R_{11} \in \mathbb{R}^{k\times k}$ is upper triangular. The details of partial QRCP are covered in Algorithm 5. The partial QRCP computes an approximate column subspace of A spanned by the leading k columns in $A\Pi$, up to the error term in R_{22}. Equivalently, (3.1) yields a low rank approximation

$$A \approx Q_1\begin{pmatrix} R_{11} & R_{12}\end{pmatrix}\Pi^T, \tag{3.2}$$

with approximation quality closely related to the error term in R_{22}.

Algorithm 9 Truncated Randomized QRCP (TRQRCP)

Inputs:
Matrix $A \in \mathbb{R}^{m\times n}$. Target rank k. Block size b. Oversampling size $p \ge 0$.
Outputs:
Orthogonal matrix $Q \in \mathbb{R}^{m\times m}$.
Upper trapezoidal matrix $R \in \mathbb{R}^{k\times n}$.
Permutation matrix $\Pi \in \mathbb{R}^{n\times n}$ such that $A\Pi \approx Q(:, 1:k)R$.
Algorithm:
Generate i.i.d. Gaussian random matrix $\Omega \in \mathcal{N}(0,1)^{(b+p)\times m}$.
Form the initial sample matrix $B = \Omega A$ and initialize $\Pi = I_n$.
for $j = 1 : b : k$ **do**
  $b = \min(k - j + 1, b)$.
  Do partial QRCP on $B(:, j:n)$ to obtain b pivots.
  Exchange corresponding columns in A, B, Π and W^T.
  Do QR on $A(j:m, j:j+b-1)$ using WY formula without updating the trailing matrix.
  Update $B(:, j+b:n)$.
end for
$Q = Q_1Q_2\cdots Q_{\lceil k/b\rceil}$. $R =$ upper trapezoidal part of the submatrix $A(1:k, 1:n)$.

The Randomized QRCP (RQRCP) algorithm [33, 121] is a more efficient variant of Algorithm 5. RQRCP generates a Gaussian random matrix $\Omega \in \mathcal{N}(0,1)^{(b+p)\times m}$ with $b+p \ll m$, where the entries of Ω are independently sampled from normal distribution, to compress

A into $B = \Omega A$ with much smaller row dimension. In practice, b is the block size and p is the oversampling size. RQRCP repeatedly runs partial QRCP on B to obtain b column pivots, applies them to the matrix A, and then computes QR without pivoting (QRNP) on A and updates the remaining columns of B. RQRCP exits this process when it reaches the target rank k. QRCP and RQRCP choose pivots on A and B respectively. RQRCP is significantly faster than QRCP as B has much smaller row dimension than A. It is shown in [121] that RQRCP is as reliable as QRCP up to failure probabilities that decay exponentially with respect to the oversampling size p.

Since the trailing matrix of A is usually not required for low-rank matrix approximations (see (3.2)), the TRQRCP (truncated RQRCP) algorithm of [33] re-organizes the computations in RQRCP to directly compute the approximation (3.2) without explicitly computing the trailing matrix R_{22}. For more details, both RQRCP and TRQRCP are based on the WY representation of the Householder transformations [9, 89, 96]:

$$Q = Q_1 Q_2 \cdots Q_k = I - YTY^T,$$

where $T \in \mathbb{R}^{k\times k}$ is an upper triangular matrix and $Y \in \mathbb{R}^{m\times k}$ is a trapezoidal matrix consisting of k consecutive Householder vectors. Let $W^T \stackrel{def}{=} T^T Y^T A$, then the trailing matrix update formula becomes $Q^T A = A - YW^T$. The main difference between RQRCP and TRQRCP is that while RQRCP computes the whole trailing matrix update, TRQRCP only computes the part of the update that affects the approximation (3.2). More discussions about RQRCP and TRQRCP can be found in [33]. The main steps of TRQRCP are briefly described in Algorithm 9.

With TRQRCP, the TUXV algorithm (Algorithm 7 in [33]) computes a low-rank approximation with the QLP factorization at a greatly accelerated speed, by computing a partial QR factorization with column pivoting, followed with a partial LQ factorization.

3.2.2 Spectrum-revealing QR Factorization

Algorithm 10 TUXV Algorithm

Inputs:
Matrix $A \in \mathbb{R}^{m\times n}$. Target rank k. Block size b. Oversampling size $p \geq 0$.
Outputs:
Column orthonormal matrices $U \in \mathbb{R}^{m\times k}$, $V \in \mathbb{R}^{n\times k}$, and upper triangular matrix $R \in \mathbb{R}^{k\times k}$ such that $A \approx URV^T$.
Algorithm:
Do TRQRCP on A to obtain $Q \in \mathbb{R}^{m\times k}$, $R \in \mathbb{R}^{k\times n}$, and $\Pi \in \mathbb{R}^{n\times n}$.
$R = R\Pi^T$ and do LQ factorization, i.e., $[V, R] = qr(R^T, 0)$.
Compute $Z = AV$ and do QR factorization, i.e., $[U, R] = qr(Z, 0)$.

Although both RQRCP and TRQRCP are very effective practical tools for low-rank matrix approximations, they are not known to provide reliable low-rank matrix approximations

due to their underlying greediness in column l_2 norm based pivoting strategy. To solve this potential problem of column based QR factorization, Gu and Eisenstat [49] proposed an efficient way to perform additional column interchanges to enhance the quality of the leading k columns in $A\Pi$ as a basis for the approximate column subspace. More recently, a more efficient and effective method, spectrum-revealing QR factorization (SRQR), was introduced and analyzed in [121] to compute the low-rank approximation (3.2). The concept of spectrum-revealing, first introduced in [120], emphasizes the utilization of partial QR factorization (3.2) as a low-rank matrix approximation, as opposed to the more traditional rank-revealing factorization, which emphasizes the utility of the partial QR factorization (3.1) as a tool for numerical rank determination. SRQR algorithm is described in Algorithm 7. SRQR initializes a partial QR factorization using RQRCP or TRQRCP and then verifies an SRQR condition. If the SRQR condition fails, it will perform a pair-wise swap between a pair of leading column (one of the first k columns of $A\Pi$) and trailing column (one of the remaining columns). The SRQR algorithm will always run to completion with a high-quality low-rank matrix approximation (3.2). For real data matrices that usually have fast decaying singular-value spectrum, this approximation is often as good as the truncated SVD. The SRQR algorithm of [121] explicitly updates the partial QR factorization (3.1) while swapping columns, but the SRQR algorithm can actually avoid any explicit computations on the trailing matrix R_{22} using TRQRCP instead of RQRCP to obtain exactly the same partial QR initialization. Below we outline the SRQR algorithm.

In (3.1), let

$$\widetilde{R} \stackrel{def}{=} \begin{pmatrix} R_{11} & a \\ & \alpha \end{pmatrix} \tag{3.3}$$

be the leading $(l+1) \times (l+1)$ submatrix of R. We define

$$g_1 \stackrel{def}{=} \frac{\|R_{22}\|_{1,2}}{|\alpha|} \qquad \text{and} \qquad g_2 \stackrel{def}{=} |\alpha| \left\| \widetilde{R}^{-T} \right\|_{1,2}, \tag{3.4}$$

where $\|X\|_{1,2}$ is the largest column 2-norm of X for any given X. In chapter 2, we proved approximation quality bounds involving g_1, g_2 for the low-rank approximation computed by RQRCP or TRQRCP. RQRCP or TRQRCP will provide a good low-rank matrix approximation if g_1 and g_2 are $O(1)$. We also proved that $g_1 \leq \sqrt{\frac{1+\varepsilon}{1-\varepsilon}}$ and $g_2 \leq \frac{\sqrt{2(1+\varepsilon)}}{1-\varepsilon}\left(1+\sqrt{\frac{1+\varepsilon}{1-\varepsilon}}\right)^{l-1}$ hold with high probability for both RQRCP and TRQRCP, where $0 < \varepsilon < 1$ is a user-defined parameter which guides the choice of the oversampling size p. For reasonably chosen ε like $\varepsilon = \frac{1}{2}$, g_1 is a small constant while g_2 can potentially be a extremely large number, which can lead to poor low-rank approximation quality. To avoid the potential large number of g_2, the SRQR algorithm uses a pair-wise swapping strategy to guarantee that g_2 is below some user defined tolerance $g > 1$ which is usually chosen to be a small number greater than one, like 2.0. In Algorithm 7, we use an approximate formula instead of directly use the definition (3.4) to estimate g_2, i.e., $g_2 \approx \frac{|\alpha|}{\sqrt{d}} \left\| \Omega \widetilde{R}^{-T} \right\|_{1,2}$, based on the Johnson-Lindenstrauss Theorem 2.

3.2.3 Tensor Approximation

In this section we review some basic notations and concepts involving tensors. A more detailed discussion of the properties and applications of tensors can be found in the review [69]. A tensor is a d-dimensional array of numbers denoted by script notation $\mathcal{X} \in \mathbb{R}^{I_1 \times \cdots \times I_d}$ with entries given by

$$x_{j_1,\ldots,j_d}, \quad 1 \le j_1 \le I_1, \ldots, 1 \le j_d \le I_d.$$

We use the matrix $X_{(n)} \in \mathbb{R}^{I_n \times \left(\Pi_{j \neq n} I_j\right)}$ to denote the nth mode unfolding of the tensor $\mathcal{X}$. Since this tensor has d dimensions, there are altogether d-possibilities for unfolding. The n-mode product of a tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times \cdots \times I_d}$ with a matrix $U \in \mathbb{R}^{k \times I_n}$ results in a tensor $\mathcal{Y} \in \mathbb{R}^{I_1 \times \cdots \times I_{n-1} \times k \times I_{n+1} \times \cdots \times I_d}$ such that

$$y_{j_1,\ldots,j_{n-1},j,j_{n+1},\ldots,j_d} = (\mathcal{X} \times_n U)_{j_1,\ldots,j_{n-1},j,j_{n+1},\ldots,j_d} = \sum_{j_n=1}^{I_n} x_{j_1,\ldots,j_d} u_{j,j_n}.$$

Alternatively it can be expressed conveniently in terms of unfolded tensors:

$$\mathcal{Y} = \mathcal{X} \times_n U \Leftrightarrow Y_{(n)} = U X_{(n)}.$$

Decompositions of higher-order tensors have applications in signal processing [22, 98, 25], numerical linear algebra [23, 68, 122], computer vision [110, 97, 114], etc. Two particular tensor decompositions can be considered as higher-order extensions of the matrix SVD: CANDECOMP/PARAFAC (CP) [17, 55] decomposes a tensor as a sum of rank-one tensors, and the Tucker decomposition [106] is a higher-order form of principal component analysis. Given the definitions of mode products and unfolding of tensors, we can define the higher-order SVD (HOSVD) algorithm for producing a rank $(k_1, \ldots, k_d)$ approximation to the tensor based on the Tucker decomposition format. The HOSVD algorithm [23, 69] returns a core tensor $\mathcal{G} \in \mathbb{R}^{k_1 \times \cdots \times k_d}$ and a set of unitary matrices $U_j \in \mathbb{R}^{I_j \times k_j}$ for $j = 1, \ldots, d$ such that

$$\mathcal{X} \approx \mathcal{G} \times_1 U_1 \cdots \times_d U_d,$$

where the right-hand side is called a Tucker decomposition. However, a straightforward generalization to higher-order ($d \ge 3$) tensors of the matrix Eckart–Young–Mirsky Theorem is not possible [24]. The HOSVD algorithm is outlined in Algorithm 11.

Since HOSVD can be prohibitive for large-scale problems, there has been a lot of literature to improve the efficiency of HOSVD computations without a noticeable deterioration in quality. One strategy for truncating the HOSVD, sequentially truncated HOSVD (ST-HOSVD) algorithm, was proposed in [3] and studied by [109]. As was shown by [109], ST-HOSVD retains several of the favorable properties of HOSVD while significantly reducing the computational cost and memory consumption. The ST-HOSVD is outlined in Algorithm 12.

Unlike HOSVD, where the number of entries in tensor unfolding $X_{(j)}$ remains the same after each loop, the number of entries in $G_{(p_j)}$ decreases as j increases in ST-HOSVD. In

Algorithm 11 HOSVD

Inputs:
Tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times \cdots \times I_d}$ and desired rank $(k_1, \ldots, k_d)$.
Outputs:
Tucker decomposition $[\mathcal{G}; U_1, \cdots, U_d]$.
Algorithm:
for $j = 1 : d$ **do**
Compute k_j left singular vectors $U_j \in \mathbb{R}^{I_j \times k_j}$ of unfolding $X_{(j)}$.
end for
Compute core tensor $\mathcal{G} \in \mathbb{R}^{k_1 \times \cdots \times k_d}$ as

$$\mathcal{G} \stackrel{def}{=} \mathcal{X} \times_1 U_1^T \times_2 \cdots \times_d U_d^T.$$

Algorithm 12 ST-HOSVD

Inputs:
Tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times \cdots \times I_d}$, desired rank $(k_1, \ldots, k_d)$, and processing order $p = (p_1, \cdots, p_d)$.
Outputs:
Tucker decomposition $[\mathcal{G}; U_1, \cdots, U_d]$.
Algorithm:
Define tensor $\mathcal{G} \leftarrow \mathcal{X}$.
for $j = 1 : d$ **do**
$r = p_j$.
Compute exact or approximate rank k_r SVD of the tensor unfolding $G_{(r)} \approx \widehat{U}_r \widehat{\Sigma}_r \widehat{V}_r^T$.
$U_r \leftarrow \widehat{U}_r$.
Update $G_{(r)} \leftarrow \widehat{\Sigma}_r \widehat{V}_r^T$, i.e., applying $\widehat{U}_r^T$ to $\mathcal{G}$.
end for

ST-HOSVD, one key step is to compute exact or approximate rank-k_r SVD of the tensor unfolding. Well known efficient ways to compute an exact low-rank SVD include Krylov subspace methods [74]. There are also efficient randomized algorithms to find an approximate low-rank SVD [52]. In Matlab tensorlab toolbox [112], the most efficient method, MLSVD_RSI, is essentially ST-HOSVD with randomized subspace iteration to find approximate SVD of tensor unfolding.

3.2.4 Nuclear Norm Minimization

Matrix rank minimization problem appears ubiquitously in many fields such as Euclidean embedding [38, 76], control [37, 84, 90], collaborative filtering [15, 91, 101], system identification [77, 78], etc. Matrix rank minimization problem has the following form:

$$\min_{X \in \mathcal{C}} \ \operatorname{rank}(X)$$

where $X \in \mathbb{R}^{m\times n}$ is the decision variable, and $\mathcal{C}$ is a convex set. In general, this problem is NP-hard due to the combinatorial nature of the function rank $(\cdot)$. To obtain a convex and more computationally tractable problem, rank (X) is replaced by its convex envelope. In [37], authors proved that the nuclear norm $\|X\|_*$ is the convex envelope of rank (X) on the set $\{X \in \mathbb{R}^{m\times n} : \|X\|_2 \leq 1\}$. The nuclear norm of a matrix $X \in \mathbb{R}^{m\times n}$ is defined as

$$\|X\|_* \stackrel{def}{=} \sum_{i=1}^{q} \sigma_i(X),$$

where $q = \text{rank}(X)$ and $\sigma_i(X)$'s are the singular values of X.

In many applications, the regularized form of nuclear norm minimization problem is considered:

$$\min_{X\in\mathbb{R}^{m\times n}} f(X) + \tau\|X\|_*$$

where $\tau > 0$ is a regularization parameter. The choice of function $f(\cdot)$ is situational: $f(X) = \|M - X\|_1$ in robust principal component analysis (robust PCA) [16], $f(X) = \|\pi_\Omega(M) - \pi_\Omega(X)\|_F^2$ in matrix completion [14], $f(X) = \frac{1}{2}\|AX - B\|_F^2$ in multi-class learning and multivariate regression [80], where M is the measured data, $\|\cdot\|_1$ denotes the l_1 norm, and $\pi_\Omega(\cdot)$ is an orthogonal projection onto the span of matrices vanishing outside of Ω so that $[\pi_\Omega(X)]_{i,j} = X_{ij}$ if $(i,j) \in \Omega$ and zero otherwise.

Many researchers have devoted themselves to solving the above nuclear norm minimization problem and plenty of algorithms have been proposed, including, singular value thresholding (SVT) [14], fixed point continuous (FPC) [80], accelerated proximal gradient (APG) [104], augmented Lagrange multiplier (ALM) [75]. The most expensive part of these algorithms is in the computation of the truncated SVD. Inexact augmented Lagrange multiplier (IALM) [75] has been proved to be one of the most accurate and efficient among them. We now describe IALM for robust PCA and matrix completion problems.

Robust PCA problem can be formalized as a minimization problem of sum of nuclear norm and scaled matrix l_1-norm (sum of matrix entries in absolute value):

$$\min \|X\|_* + \lambda\|E\|_1, \qquad \text{subject to} \qquad M = X + E, \tag{3.5}$$

where M is measured matrix, X has low-rank, E is an error matrix and sufficiently sparse, and λ is a positive weighting parameter. Algorithm 13 describes the details of IALM method to solve robust PCA [75] problem, where $\|\cdot\|_M$ denotes the maximum absolute value of the matrix entries, and $\mathcal{S}_\omega(x) = \text{sgn}(x)\cdot\max(|x| - \omega, 0)$ is the soft shrinkage operator [51] where $x \in \mathbb{R}^n$ and $\omega > 0$.

Matrix completion problem [15, 75] can be written in the form:

$$\min_{X\in\mathbb{R}^{m\times n}} \|X\|_* \qquad \text{subject to} \qquad X + E = M, \quad \pi_\Omega(E) = 0, \tag{3.6}$$

where $\pi_\Omega : \mathbb{R}^{m\times n} \to \mathbb{R}^{m\times n}$ is an orthogonal projection that keeps the entries in Ω unchanged and sets those outside Ω zeros. In [75], authors applied IALM method on the matrix completion problem. We describe this method in Algorithm 14, where $\overline{\Omega}$ is the complement of Ω.

Algorithm 13 Robust PCA Using IALM

Inputs:

Measured matrix $M \in \mathbb{R}^{m \times n}$, positive number $\lambda,\ \mu_0,\ \overline{\mu}$, tolerance tol, $\rho > 1$.

Outputs:

Matrix pair (X_k, E_k).

Algorithm:

$k = 0$; $J(M) = \max\left(\|M\|_2, \lambda^{-1}\|M\|_M\right)$; $Y_0 = M/J(M)$; $E_0 = 0$;

while not converged **do**

$(\mathbf{U}, \mathbf{\Sigma}, \mathbf{V}) = \text{svd}\left(\mathbf{M} - \mathbf{E_k} + \mu_\mathbf{k}^{-1}\mathbf{Y_k}\right)$;

$X_{k+1} = U\mathcal{S}_{\mu_k^{-1}}(\Sigma)V^T$;

$E_{k+1} = \mathcal{S}_{\lambda\mu_k^{-1}}\left(M - X_{k+1} + \mu_k^{-1}Y_k\right)$;

$Y_{k+1} = Y_k + \mu_k(M - X_{k+1} - E_{k+1})$;

Update $\mu_{k+1} = \min(\rho\mu_k, \overline{\mu})$;

$k = k + 1$;

if $\|M - X_k - E_k\|_F / \|M\|_F < tol$ **then**

Break;

end if

end while

Algorithm 14 Matrix Completion Using IALM

Inputs:

Sampled set Ω, sampled entries $\pi_\Omega(M)$, positive number $\lambda,\ \mu_0,\ \overline{\mu}$, tolerance tol, $\rho > 1$.

Outputs:

Matrix pair (X_k, E_k).

Algorithm:

$k = 0$; $Y_0 = 0$; $E_0 = 0$;

while not converged **do**

$(\mathbf{U}, \mathbf{\Sigma}, \mathbf{V}) = \text{svd}\left(\mathbf{M} - \mathbf{E_k} + \mu_\mathbf{k}^{-1}\mathbf{Y_k}\right)$;

$X_{k+1} = U\mathcal{S}_{\mu_k^{-1}}(\Sigma)V^T$;

$E_{k+1} = \pi_{\overline{\Omega}}\left(M - X_{k+1} + \mu_k^{-1}Y_k\right)$;

$Y_{k+1} = Y_k + \mu_k(M - X_{k+1} - E_{k+1})$;

Update $\mu_{k+1} = \min(\rho\mu_k, \overline{\mu})$;

$k = k + 1$;

if $\|M - X_k - E_k\|_F / \|M\|_F < tol$ **then**

Break;

end if

end while

3.3 Flip-Flop SRQR Factorization

3.3.1 Flip-Flop SRQR Factorization

In this section, we introduce our Flip-Flop SRQR factorization, slightly different from TUXV (Algorithm 10), to compute SVD approximation based on QLP factorization. Given integer $l \geq k$, we run SRQR (the version without computing the trailing matrix) to l steps on A,

$$A\Pi = QR = Q \begin{pmatrix} R_{11} & R_{12} \\ & R_{22} \end{pmatrix}, \tag{3.7}$$

where $R_{11} \in \mathbb{R}^{l \times l}$ is upper triangular; $R_{12} \in \mathbb{R}^{l \times (n-l)}$; and $R_{22} \in \mathbb{R}^{(m-l) \times (n-l)}$. Then we run partial QRNP to l steps on R^T,

$$R^T = \begin{pmatrix} R_{11}^T & \\ R_{12}^T & R_{22}^T \end{pmatrix} = \widehat{Q} \begin{pmatrix} \widehat{R}_{11} & \widehat{R}_{12} \\ & \widehat{R}_{22} \end{pmatrix} \approx \widehat{Q}_1 \begin{pmatrix} \widehat{R}_{11} & \widehat{R}_{12} \end{pmatrix}, \tag{3.8}$$

where $\widehat{Q} = \begin{pmatrix} \widehat{Q}_1 & \widehat{Q}_2 \end{pmatrix}$ with $\widehat{Q}_1 \in \mathbb{R}^{n \times l}$. Therefore, combing the fact that $A\Pi\widehat{Q}_1 = Q \begin{pmatrix} \widehat{R}_{11} & \widehat{R}_{12} \end{pmatrix}^T$, we can approximate matrix A by

$$A = QR\Pi^T = Q \left(R^T\right)^T \Pi^T \approx Q \begin{pmatrix} \widehat{R}_{11}^T \\ \widehat{R}_{12}^T \end{pmatrix} \widehat{Q}_1^T \Pi^T = A \left(\Pi\widehat{Q}_1\right) \left(\Pi\widehat{Q}_1\right)^T. \tag{3.9}$$

We denote the rank-k truncated SVD of $A\Pi\widehat{Q}_1$ by $\widetilde{U}_k \Sigma_k \widetilde{V}_k^T$. Let $U_k = \widetilde{U}_k,\ V_k = \Pi\widehat{Q}_1\widetilde{V}_k$, then using (3.9), a rank-k approximate SVD of A is obtained:

$$A \approx U_k \Sigma_k V_k^T, \tag{3.10}$$

where $U_k \in \mathbb{R}^{m \times k},\ V_k \in \mathbb{R}^{n \times k}$ are column orthonormal; and $\Sigma_k = \mathrm{Diag}\left(\sigma_1, \cdots, \sigma_k\right)$ with σ_i's are the leading k singular values of $A\Pi\widehat{Q}_1$. The Flip-Flop SRQR factorization is outlined in Algorithm 15.

3.3.2 Complexity Analysis

In this section, we do a complexity analysis of Flip-Flop SRQR. Since approximate SVD only makes sense when target rank k is small, we assume $k \leq l \ll \min(m, n)$. The complexity analysis of Flip-Flop SRQR is as follows:

1. The main cost of doing SRQR with TRQRCP on A is $2mnl + 2(b+p)mn + (m+n)l^2$.
2. The main cost of QR factorization on $(R_{11}, R_{12})^T$ and forming $\widehat{Q}_1$ is $2nl^2 - \frac{2}{3}l^3$.
3. The main cost of computing $tmp = A\Pi\widehat{Q}_1$ is $2mnl$.

Algorithm 15 Flip-Flop Spectrum-Revealing QR Factorization

Inputs:

Matrix $A \in \mathbb{R}^{m \times n}$. Target rank k. Block size b. Oversampling size $p \geq 0$.
Integer $l \geq k$. Tolerance $g > 1$ for g_2.

Outputs:

$U \in \mathbb{R}^{m \times k}$ contains the approximate top k left singular vectors of A.
$\Sigma \in \mathbb{R}^{k \times k}$ contains the approximate top k singular values of A.
$V \in \mathbb{R}^{n \times k}$ contains the approximate top k right singular vectors of A.

Algorithm:

Run SRQR on A to l steps to obtain (R_{11}, R_{12}).
Run QRNP on $(R_{11}, R_{12})^T$ to obtain $\widehat{Q}_1$, represented by a sequence of Householder vectors.
$tmp = A\Pi\widehat{Q}_1$.
$[U_{tmp}, \Sigma_{tmp}, V_{tmp}] = svd\,(tmp)$.
$U = U_{tmp}\,(:, 1:, k)\,, \Sigma = \Sigma_{tmp}\,(1:k, 1:k)\,, V = \Pi\widehat{Q}_1 V_{tmp}\,(:, 1:k)$.

4. The main cost of computing $[U, \sim, \sim] = svd\,(tmp)$ is $O(ml^2)$.

5. The main cost of forming V_k is $2nlk$.

Since $k \leq l \ll \min(m, n)$, the complexity of Flip-Flop SRQR is $(4l + 2(b + p))mn$ by omitting the lower-order terms.

On the other hand, the complexity of approximate SVD with randomized subspace iteration (RSISVD) [48, 52] is $(4 + 4q)(k + p)mn$, where p is the oversampling size and q is the number of subspace iterations (see detailed analysis in the appendix). In practice, p is chosen to be a small integer like 5 in RSISVD. In Flip-Flop SRQR, l is usually chosen to be a little bit larger than k, like $l = k + 5$. We also found that $l = k$ is sufficient in terms of approximation quality in our numerical experiments. Assume that $l = k + s$ where s is a small positive integer like 5, and block size b is 32 or 64, and q is 1 or 2, and oversampling size p is 5, we claim that Flip-Flop SRQR is more efficient than RSISVD. In fact, we notice that

$$\begin{aligned} & (4l + 2(b + p))mn < (4 + 4q)(k + p)mn \\ \Leftrightarrow \quad & 4l + 2(b + p) < (4 + 4q)(k + p) \\ \Leftrightarrow \quad & 2l + b + p < 2k + 2p + 2qk + 2pq \\ \Leftrightarrow \quad & 2s + b < p + 2qk + 2pq \end{aligned}$$

where the last inequality holds true because the left-hand side is a constant like $(2 * 5 + 64) = 74$ while the right-hand side is a linear function of the target rank k, which is obviously larger. Such over-performance can also be validated by our numerous experiments in Section 3.4.

3.3.3 Quality Analysis of Flip-Flop SRQR

This section is devoted to the quality analysis of Flip-Flop SRQR. We start with Lemma 6.

Lemma 6. *Given any matrix $X = (X_1, X_2)$ with $X_i \in \mathbb{R}^{m\times n_i}$ $(i = 1, 2)$ and $n_1 + n_2 = n$,*

$$\sigma_j(X)^2 \le \sigma_j(X_1)^2 + \|X_2\|_2^2 \quad (1 \le j \le \min(m, n)).$$

Proof. Since $XX^T = X_1X_1^T + X_2X_2^T$, we obtain the above result using [60, Theorem 3.3.16]. □ □

We are now ready to derive bounds on the singular values and approximation error of Flip-Flop SRQR. We need to emphasize that even if the target rank is k, we run Flip-Flop SRQR with an actual target rank l which is a little bit larger than k. The difference between k and l can create a gap between singular values of A so that we can obtain a reliable low-rank approximation.

Theorem 7. *Given matrix $A \in \mathbb{R}^{m\times n}$, target rank k, oversampling size p, and an actual target rank $l \ge k$, U_k, Σ_k, V_k computed by (3.10) of Flip-Flop SRQR satisfies*

$$\sigma_j(\Sigma_k) \ge \frac{\sigma_j(A)}{\sqrt[4]{1 + \frac{2\|R_{22}\|_2^4}{\sigma_j^4(\Sigma_k)}}} \quad (1 \le j \le k), \tag{3.11}$$

and

$$\left\|A - U_k\Sigma_kV_k^T\right\|_2 \le \sigma_{k+1}(A)\sqrt[4]{1 + 2\left(\frac{\|R_{22}\|_2}{\sigma_{k+1}(A)}\right)^4}, \tag{3.12}$$

where $R_{22} \in \mathbb{R}^{(m-l)\times(n-l)}$ is the trailing matrix in (3.7). Using the properties of SRQR, we can further have

$$\sigma_j(\Sigma_k) \ge \frac{\sigma_j(A)}{\sqrt[4]{1 + \min\left(2\widehat{\tau}^4, \tau^4(2 + 4\widehat{\tau}^4)\left(\frac{\sigma_{l+1}(A)}{\sigma_j(A)}\right)^4\right)}} \quad (1 \le j \le k), \tag{3.13}$$

and

$$\left\|A - U_k\Sigma_kV_k^T\right\|_2 \le \sigma_{k+1}(A)\sqrt[4]{1 + 2\tau^4\left(\frac{\sigma_{l+1}(A)}{\sigma_{k+1}(A)}\right)^4}, \tag{3.14}$$

where τ and $\widehat{\tau}$ defined in (3.18) have matrix dimensional dependent upper bounds:

$$\tau \le g_1g_2\sqrt{(l+1)(n-l)}, \quad \text{and} \quad \widehat{\tau} \le g_1g_2\sqrt{l(n-l)},$$

where $g_1 \le \sqrt{\frac{1+\varepsilon}{1-\varepsilon}}$ and $g_2 \le g$ with high probability. $\varepsilon > 0$ and $g > 1$ are user defined parameters.

Proof. In terms of the singular value bounds, observe that

$$\begin{pmatrix} \widehat{R}_{11} & \widehat{R}_{12} \\ & \widehat{R}_{22} \end{pmatrix} \begin{pmatrix} \widehat{R}_{11} & \widehat{R}_{12} \\ & \widehat{R}_{22} \end{pmatrix}^T = \begin{pmatrix} \widehat{R}_{11}\widehat{R}_{11}^T + \widehat{R}_{12}\widehat{R}_{12}^T & \widehat{R}_{12}\widehat{R}_{22}^T \\ \widehat{R}_{22}\widehat{R}_{12}^T & \widehat{R}_{22}\widehat{R}_{22}^T \end{pmatrix},$$

we apply Lemma 6 twice for any $1 \leq j \leq k$,

$$\begin{aligned} &\sigma_j^2 \left(\begin{pmatrix} \widehat{R}_{11} & \widehat{R}_{12} \\ & \widehat{R}_{22} \end{pmatrix} \begin{pmatrix} \widehat{R}_{11} & \widehat{R}_{12} \\ & \widehat{R}_{22} \end{pmatrix}^T \right) \\ \leq &\sigma_j^2 \left(\begin{pmatrix} \widehat{R}_{11}\widehat{R}_{11}^T + \widehat{R}_{12}\widehat{R}_{12}^T & \widehat{R}_{12}\widehat{R}_{22}^T \end{pmatrix} \right) + \left\| \begin{pmatrix} \widehat{R}_{22}\widehat{R}_{12}^T & \widehat{R}_{22}\widehat{R}_{22}^T \end{pmatrix} \right\|_2^2 \\ \leq &\sigma_j^2 \left(\widehat{R}_{11}\widehat{R}_{11}^T + \widehat{R}_{12}\widehat{R}_{12}^T \right) + \left\| \widehat{R}_{12}\widehat{R}_{22}^T \right\|_2^2 + \left\| \begin{pmatrix} \widehat{R}_{22}\widehat{R}_{12}^T & \widehat{R}_{22}\widehat{R}_{22}^T \end{pmatrix} \right\|_2^2 \\ \leq &\sigma_j^2 \left(\widehat{R}_{11}\widehat{R}_{11}^T + \widehat{R}_{12}\widehat{R}_{12}^T \right) + 2 \left\| \begin{pmatrix} \widehat{R}_{12} \\ \widehat{R}_{22} \end{pmatrix} \right\|_2^4 \\ = &\sigma_j^2 \left(\widehat{R}_{11}\widehat{R}_{11}^T + \widehat{R}_{12}\widehat{R}_{12}^T \right) + 2 \left\| R_{22} \right\|_2^4 . \end{aligned} \tag{3.15}$$

The relation (3.15) can be further rewritten as

$$\sigma_j^4 (A) \leq \sigma_j^4 \left(\begin{pmatrix} \widehat{R}_{11} & \widehat{R}_{12} \end{pmatrix} \right) + 2 \left\| R_{22} \right\|_2^4 = \sigma_j^4 (\Sigma_k) + 2 \left\| R_{22} \right\|_2^4 \quad (1 \leq j \leq k),$$

which is equivalent to

$$\sigma_j(\Sigma_k) \geq \frac{\sigma_j(A)}{\sqrt[4]{1 + \frac{2\|R_{22}\|_2^4}{\sigma_j^4(\Sigma_k)}}}.$$

For the residual matrix bound, we let

$$\begin{pmatrix} \widehat{R}_{11} & \widehat{R}_{12} \end{pmatrix} \stackrel{def}{=} \begin{pmatrix} \overline{R}_{11} & \overline{R}_{12} \end{pmatrix} + \begin{pmatrix} \delta\overline{R}_{11} & \delta\overline{R}_{12} \end{pmatrix},$$

where $\begin{pmatrix} \overline{R}_{11} & \overline{R}_{12} \end{pmatrix}$ is the rank-k truncated SVD of $\begin{pmatrix} \widehat{R}_{11} & \widehat{R}_{12} \end{pmatrix}$. Notice that

$$\left\| A - U_k \Sigma_k V_k^T \right\|_2 = \left\| A\Pi - Q \begin{pmatrix} \overline{R}_{11} & \overline{R}_{12} \\ & 0 \end{pmatrix}^T \widehat{Q}^T \right\|_2, \tag{3.16}$$

it follows from the orthogonality of singular vectors that

$$\begin{pmatrix} \overline{R}_{11} & \overline{R}_{12} \end{pmatrix}^T \begin{pmatrix} \delta\overline{R}_{11} & \delta\overline{R}_{12} \end{pmatrix} = 0,$$

and therefore

$$\begin{aligned} & \begin{pmatrix} \widehat{R}_{11} & \widehat{R}_{12} \end{pmatrix}^T \begin{pmatrix} \widehat{R}_{11} & \widehat{R}_{12} \end{pmatrix} \\ = \; & \begin{pmatrix} \overline{R}_{11} & \overline{R}_{12} \end{pmatrix}^T \begin{pmatrix} \overline{R}_{11} & \overline{R}_{12} \end{pmatrix} + \begin{pmatrix} \delta\overline{R}_{11} & \delta\overline{R}_{12} \end{pmatrix}^T \begin{pmatrix} \delta\overline{R}_{11} & \delta\overline{R}_{12} \end{pmatrix}, \end{aligned}$$

which implies

$$\widehat{R}_{12}^T \widehat{R}_{12} = \overline{R}_{12}^T \overline{R}_{12} + \left(\delta\overline{R}_{12}\right)^T \left(\delta\overline{R}_{12}\right). \tag{3.17}$$

Similar to the deduction of (3.15), from (3.17) we can derive

$$\begin{aligned} \left\| \begin{pmatrix} \delta\overline{R}_{11} & \delta\overline{R}_{12} \\ & \widehat{R}_{22} \end{pmatrix} \right\|_2^4 &\leq \left\| \begin{pmatrix} \delta\overline{R}_{11} & \delta\overline{R}_{12} \end{pmatrix} \right\|_2^4 + 2 \left\| \begin{pmatrix} \delta\overline{R}_{12} \\ \widehat{R}_{22} \end{pmatrix} \right\|_2^4 \\ &\leq \sigma_{k+1}^4(A) + 2 \left\| \begin{pmatrix} \widehat{R}_{12} \\ \widehat{R}_{22} \end{pmatrix} \right\|_2^4 = \sigma_{k+1}^4(A) + 2\|R_{22}\|_2^4. \end{aligned}$$

Combining with (3.16), it now follows that

$$\begin{aligned} \left\|A - U_k \Sigma_k V_k^T\right\|_2 &= \left\| A\Pi - Q \begin{pmatrix} \overline{R}_{11} & \overline{R}_{12} \\ 0 \end{pmatrix}^T \widehat{Q}^T \right\|_2 = \left\| \begin{pmatrix} \delta\overline{R}_{11} & \delta\overline{R}_{12} \\ & \widehat{R}_{22} \end{pmatrix} \right\|_2 \\ &\leq \sigma_{k+1}(A) \sqrt[4]{1 + 2\left(\frac{\|R_{22}\|_2}{\sigma_{k+1}(A)}\right)^4}. \end{aligned}$$

To obtain an upper bound of $\|R_{22}\|_2$ in (3.11) and (3.12), we follow the analysis of SRQR in [121].

From the analysis of [121, Section IV], Algorithm 7 ensures that $g_1 \leq \sqrt{\frac{1+\varepsilon}{1-\varepsilon}}$ and $g_2 \leq g$ with high probability (the actual probability guarantee formula can be found in [121, Section IV]), where g_1 and g_2 are defined by (3.4). Here $0 < \varepsilon < 1$ is a user defined parameter to adjust the choice of oversampling size p used in the TRQRCP initialization part in SRQR. $g > 1$ is a user defined parameter in the extra swapping part in SRQR. Let

$$\tau \overset{def}{=} g_1 g_2 \frac{\|R_{22}\|_2}{\|R_{22}\|_{1,2}} \frac{\left\|\widetilde{R}^{-T}\right\|_{1,2}^{-1}}{\sigma_{l+1}(A)} \qquad \text{and} \qquad \widehat{\tau} \overset{def}{=} g_1 g_2 \frac{\|R_{22}\|_2}{\|R_{22}\|_{1,2}} \frac{\left\|R_{11}^{-T}\right\|_{1,2}^{-1}}{\sigma_k(\Sigma_k)}, \tag{3.18}$$

where $\widetilde{R}$ is defined by (3.3). Since $\frac{1}{\sqrt{n}}\|X\|_{1,2} \leq \|X\|_2 \leq \sqrt{n}\|X\|_{1,2}$ and $\sigma_i(X_1) \leq \sigma_i(X)$ $(1 \leq i \leq \min(s,t))$ for any matrix $X \in \mathbb{R}^{m\times n}$ and submatrix $X_1 \in \mathbb{R}^{s\times t}$ of X,

$$\tau = g_1 g_2 \frac{\|R_{22}\|_2}{\|R_{22}\|_{1,2}} \frac{\left\|\widetilde{R}^{-T}\right\|_{1,2}^{-1}}{\sigma_{l+1}\left(\widetilde{R}\right)} \frac{\sigma_{l+1}\left(\widetilde{R}\right)}{\sigma_{l+1}(A)} \leq g_1 g_2 \sqrt{(l+1)(n-l)}.$$

Using the fact that $\sigma_l\left(\begin{pmatrix} R_{11} & R_{12}\end{pmatrix}\right) = \sigma_l\left(\widehat{R}_{11}\right)$ and $\sigma_k\left(\Sigma_k\right) = \sigma_k\left(\begin{pmatrix}\widehat{R}_{11} & \widehat{R}_{12}\end{pmatrix}\right)$ by (3.8) and (3.10),

$$\widehat{\tau} = g_1 g_2 \frac{\|R_{22}\|_2}{\|R_{22}\|_{1,2}} \frac{\left\|R_{11}^{-T}\right\|_{1,2}^{-1}}{\sigma_l\left(R_{11}\right)} \frac{\sigma_l\left(R_{11}\right)}{\sigma_l\left(\begin{pmatrix} R_{11} & R_{12}\end{pmatrix}\right)} \frac{\sigma_l\left(\begin{pmatrix} R_{11} & R_{12}\end{pmatrix}\right)}{\sigma_k\left(\Sigma_k\right)} \leq g_1 g_2 \sqrt{l\left(n-l\right)},$$

By definition of τ,

$$\|R_{22}\|_2 = \tau\, \sigma_{l+1}\left(A\right). \tag{3.19}$$

Plugging this into (3.12) yields (3.14).

By definition of $\widehat{\tau}$, we observe that

$$\|R_{22}\|_2 \leq \widehat{\tau} \sigma_k\left(\Sigma_k\right). \tag{3.20}$$

By (3.11) and (3.20),

$$\sigma_j\left(\Sigma_k\right) \geq \frac{\sigma_j\left(A\right)}{\sqrt[4]{1+2\left(\frac{\|R_{22}\|_2}{\sigma_j\left(\Sigma_k\right)}\right)^4}} \geq \frac{\sigma_j\left(A\right)}{\sqrt[4]{1+2\left(\frac{\|R_{22}\|_2}{\sigma_k\left(\Sigma_k\right)}\right)^4}} \geq \frac{\sigma_j\left(A\right)}{\sqrt[4]{1+2\widehat{\tau}^4}}, \quad \left(1 \leq j \leq k\right). \tag{3.21}$$

On the other hand, using (3.11),

$$\begin{aligned}
\sigma_j^4\left(A\right) &\leq \sigma_j^4\left(\Sigma_k\right)\left(1+2\frac{\sigma_j^4\left(A\right)}{\sigma_j^4\left(\Sigma_k\right)}\frac{\|R_{22}\|_2^4}{\sigma_j^4\left(A\right)}\right) \\
&\leq \sigma_j^4\left(\Sigma_k\right)\left(1+2\frac{\left(\sigma_j^4\left(\Sigma_k\right)+2\|R_{22}\|_2^4\right)}{\sigma_j^4\left(\Sigma_k\right)}\frac{\|R_{22}\|_2^4}{\sigma_j^4\left(A\right)}\right) \\
&\leq \sigma_j^4\left(\Sigma_k\right)\left(1+2\left(1+2\frac{\|R_{22}\|_2^4}{\sigma_k^4\left(\Sigma_k\right)}\right)\frac{\|R_{22}\|_2^4}{\sigma_j^4\left(A\right)}\right),
\end{aligned}$$

that is,

$$\sigma_j\left(\Sigma_k\right) \geq \frac{\sigma_j\left(A\right)}{\sqrt[4]{1+\left(2+4\frac{\|R_{22}\|_2^4}{\sigma_k^4\left(\Sigma_k\right)}\right)\frac{\|R_{22}\|_2^4}{\sigma_j^4\left(A\right)}}}.$$

Plugging (3.19) and (3.20) into this above equation,

$$\sigma_j\left(\Sigma_k\right) \geq \frac{\sigma_j\left(A\right)}{\sqrt[4]{1+\tau^4\left(2+4\widehat{\tau}^4\right)\left(\frac{\sigma_{l+1}(A)}{\sigma_j(A)}\right)^4}}, \qquad \left(1 \leq j \leq k\right). \tag{3.22}$$

Combing (3.21) and (3.22), we arrive at (3.14). □ □

We note that (3.11) and (3.12) still hold true if we replace k by l.

Inequality (3.13) shows that under definitions (3.18) of τ and $\widehat{\tau}$, Flip-Flop SRQR can reveal at least a dimension dependent fraction of all the leading singular values of A and indeed approximate them very accurately in case they decay relatively quickly. Moreover, (3.14) shows that Flip-Flop SRQR can compute a rank-k approximation that is up to a factor of $\sqrt[4]{1+2\tau^4\left(\frac{\sigma_{l+1}(A)}{\sigma_{k+1}(A)}\right)^4}$ from optimal. In situations where singular values of A decay relatively quickly, our rank-k approximation is about as accurate as the truncated SVD with a choice of l such that

$$\frac{\sigma_{l+1}(A)}{\sigma_{k+1}(A)} = o(1).$$

3.4 Numerical Experiments

In this section, we demonstrate the effectiveness and efficiency of Flip-Flop SRQR (FFSRQR) algorithm in several numerical experiments. Firstly, we compare FFSRQR with other approximate SVD algorithms on matrix approximation. Secondly, we compare FFSRQR with other methods on tensor approximation problem using tensorlab toolbox [112]. Thirdly, we compare FFSRQR with other methods on the robust PCA problem and matrix completion problem. All experiments are implemented in Matlab R2016b on a MacBook Pro with a 2.9 GHz i5 processor and 8 GB memory. The underlying routines used in FFSRQR are written in Fortran. For a fair comparison, we turn off multi-threading in Matlab.

3.4.1 Approximate Truncated SVD

In this section, we compare FFSRQR with other four approximate SVD algorithms on low-rank matrices approximation. All tested methods are listed in Table 3.1. The test matrices are:

Type 1: $A \in \mathbb{R}^{m\times n}$ [102] is defined by $A = U\,D\,V^T + 0.1 * D(s,s) * E$, where $U \in \mathbb{R}^{m\times s}, V \in \mathbb{R}^{n\times s}$ are column-orthonormal matrices, and $D \in \mathbb{R}^{s\times s}$ is a diagonal matrix with s geometrically decreasing diagonal entries from 1 to 10^{-3}. $E \in \mathbb{R}^{m\times n}$ is a random matrix where the entries are independently sampled from a normal distribution $\mathcal{N}(0,1)$. In our numerical experiment, we test on three different random matrices. The square matrix has a size of 15000×15000; the short-fat matrix has a size of 1000×15000; the tall-skinny matrix has a size of 15000×1000.

Type 2: $A \in \mathbb{R}^{4929\times 4929}$ is a real data matrix from the University of Florida sparse matrix collection [21]. Its corresponding file name is HB/GEMAT11.

For a given matrix $A \in \mathbb{R}^{m\times n}$, the relative SVD approximation error is measured by $\left\|A - U_k\Sigma_k V_k^T\right\|_F / \|A\|_F$ where Σ_k contains approximate top k singular values, and U_k, V_k

Method	Description
LANSVD	Approximate SVD using Lanczos bidiagonalization with partial reorthogonalization [71]. We use its Matlab implementation in PROPACK [70].
FFSRQR	Flip-Flop Spectrum-revealing QR factorization. We write its implementation using Mex functions that wrapped BLAS and LAPACK routines [2].
RSISVD	Approximate SVD with randomized subspace iteration [52, 48]. We use its Matlab implementation in tensorlab toolbox [112].
LTSVD	Linear Time SVD [29]. We use its Matlab implementation by Ma et al. [80].

Table 3.1: Methods for approximate SVD.

Method	Parameter
FFSRQR	oversampling size $p = 5$, block size $b = \min\{32, k\}$, $l = k,\ d = 10,\ g = 2.0$
RSISVD	oversampling size $p = 5$, subspace iteration $q = 1$
LTSVD	probabilities $p_i = 1/n$

Table 3.2: Parameters used in RSISVD, FFSRQR, and LTSVD.

are corresponding approximate top k singular vectors. The parameters used in FFSRQR, RSISVD, and LTSVD are listed in Table 3.2.

Figure 3.5 through Figure 3.15 show run time, relative approximation error, and top 20 singular values comparison respectively on four different matrices. While LTSVD is faster than the other methods in most cases, the approximation error of LTSVD is significantly larger than all the other methods. In terms of accuracy, FFSRQR is comparable to LANSVD and RSISVD. In terms of speed, FFSRQR is faster than LANSVD. When target rank k is small, FFSRQR is comparable to RSISVD, but FFSRQR is better when k is larger.

3.4.2 Tensor Approximation

This section illustrates the effectiveness and efficiency of FFSRQR for computing approximate tensor decompositions. ST-HOSVD [3, 109] is one of the most efficient algorithms to compute Tucker decomposition of tensors, and the most costly part of this algorithm is to compute SVD or approximate SVD of the tensor unfoldings. Truncated SVD and RSISVD are used in routines MLSVD and MLSVD_RSI respectively in Matlab tensorlab toolbox [112]. Based on this Matlab toolbox, we implement ST-HOSVD using FFSRQR or LTSVD to do the SVD approximation. We name these two new routines MLSVD_FFSRQR and MLSVD_LTSVD respectively. We compare these four routines in this numerical experiment.

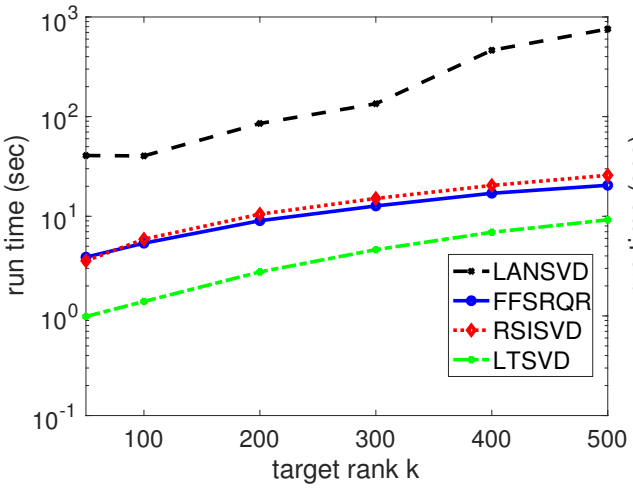

Figure 3.1: Type 1: Random square matrix

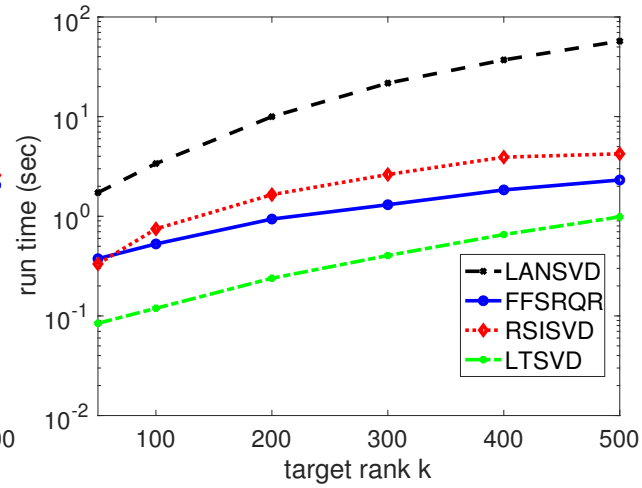

Figure 3.2: Type 1: Random short-fat matrix

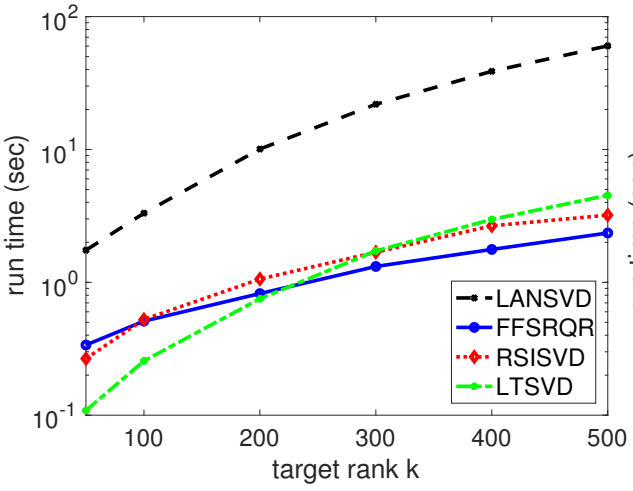

Figure 3.3: Type 1: Random tall-skinny matrix

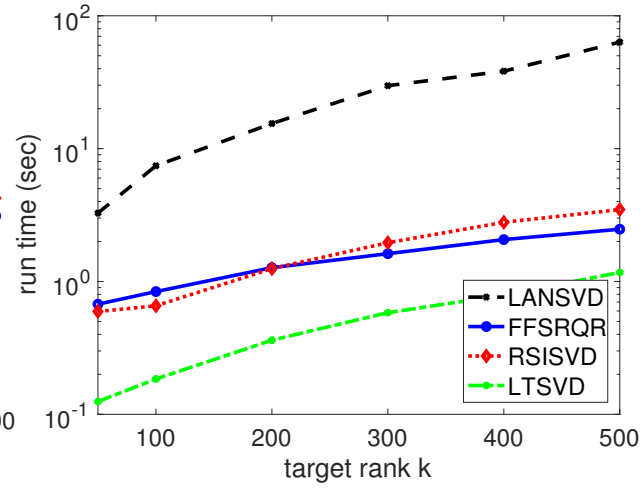

Figure 3.4: Type 2: GEMAT11

Figure 3.5: Run time comparison for approximate SVD algorithms.

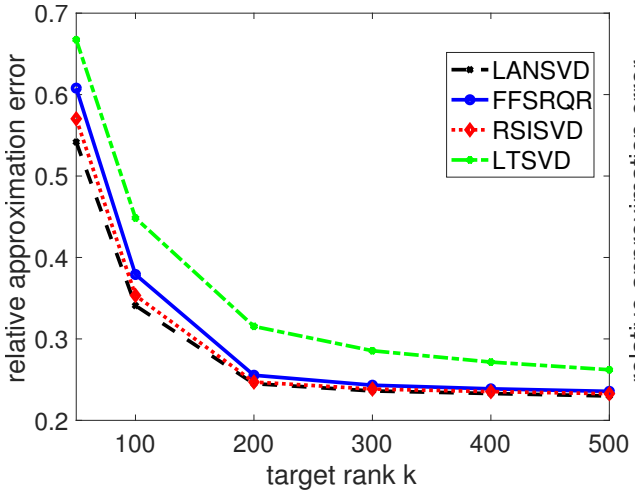

Figure 3.6: Type 1: Random square matrix

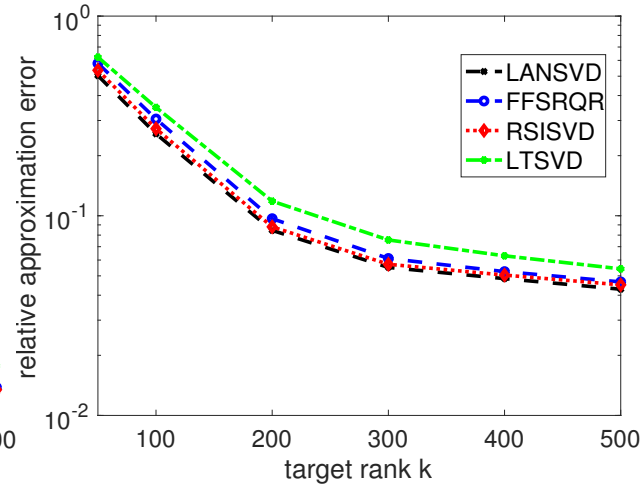

Figure 3.7: Type 1: Random short-fat matrix

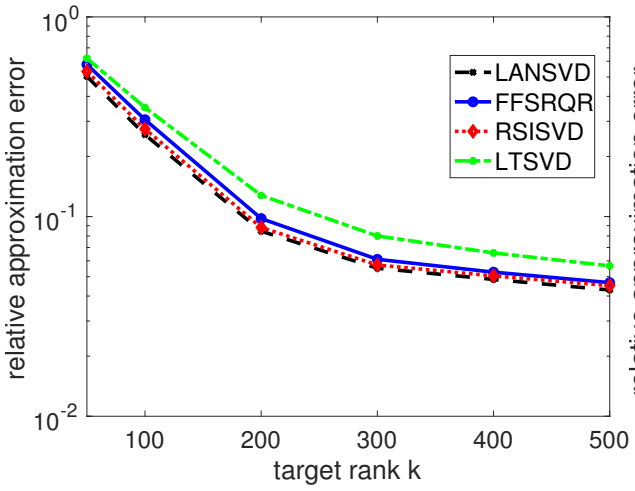

Figure 3.8: Type 1: Random tall-skinny matrix

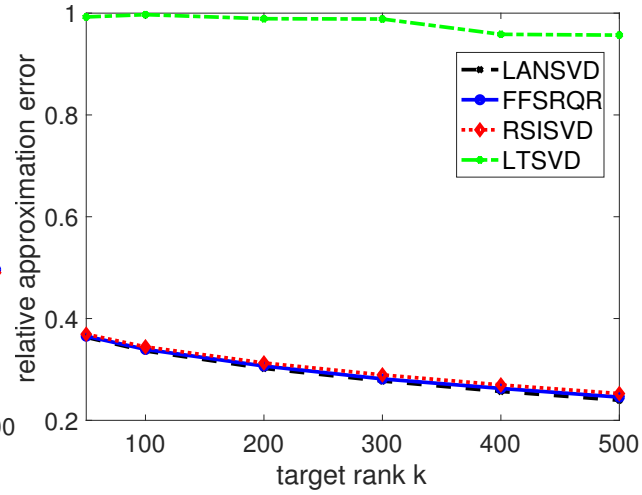

Figure 3.9: Type 2: GEMAT11

Figure 3.10: Relative approximation error comparison for approximate SVD algorithms.

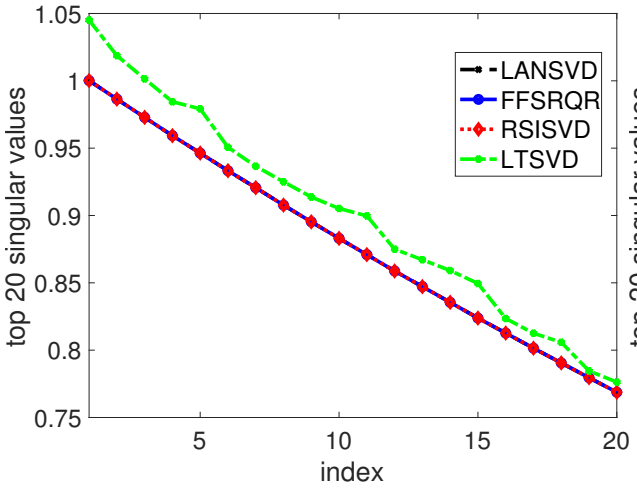

Figure 3.11: Type 1: Random square matrix

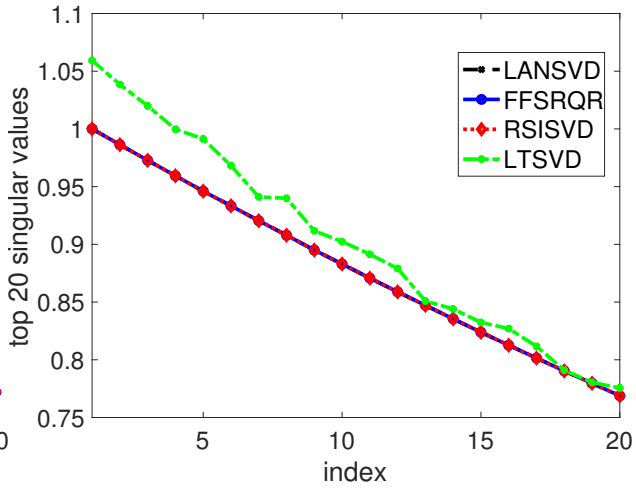

Figure 3.12: Type 1: Random short-fat matrix

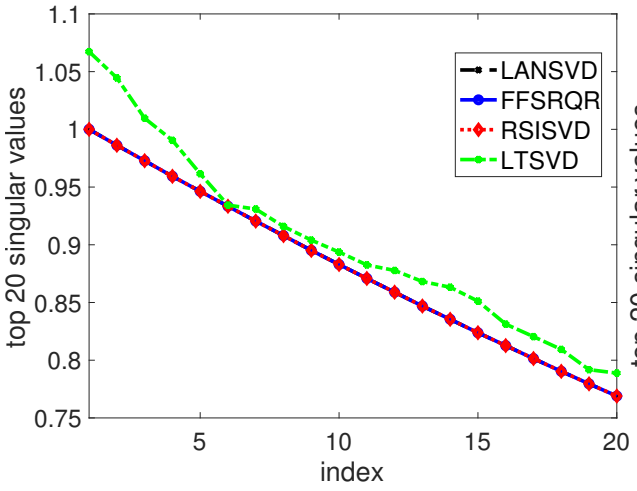

Figure 3.13: Type 1: Random tall-skinny matrix

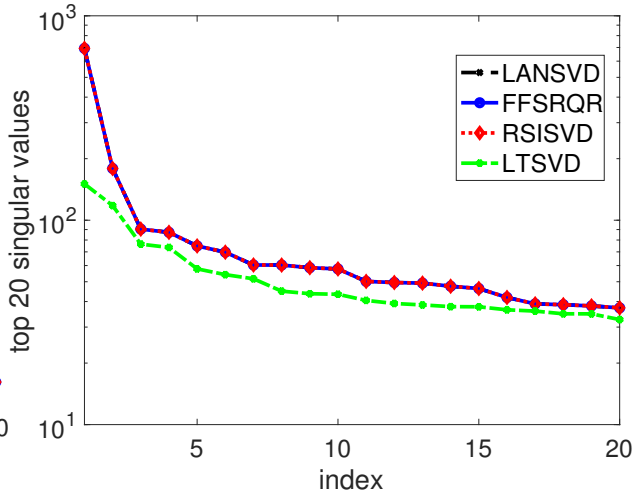

Figure 3.14: Type 2: GEMAT11

Figure 3.15: Top 20 singular values comparison for approximate SVD algorithms.

We also have Python codes for this tensor approximation numerical experiment. We don't list the results of Python here but they are similar to those of Matlab.

3.4.2.1 A Sparse Tensor Example

We test on a sparse tensor $\mathcal{X} \in \mathbb{R}^{n\times n\times n}$ of the following format [100, 94],

$$\mathcal{X} = \sum_{j=1}^{10} \frac{1000}{j} x_j \circ y_j \circ z_j + \sum_{j=11}^{n} \frac{1}{j} x_j \circ y_j \circ z_j,$$

where $x_j, y_j, z_j \in \mathbb{R}^n$ are sparse vectors with nonnegative entries. The symbol "$\circ$" represents the vector outer product. We compute a rank-(k, k, k) Tucker decomposition $[\mathcal{G}; U_1, U_2, U_3]$ using MLSVD, MLSVD_FFSRQR, MLSVD_RSI, and MLSVD_LTSVD respectively. The relative approximation error is measured by $\|\mathcal{X} - \mathcal{X}_k\|_F / \|\mathcal{X}\|_F$ where $\mathcal{X}_k = \mathcal{G} \times_1 U_1 \times_2 U_2 \times_3 U_3$.

Figure 3.16 compares efficiency and accuracy of different methods on a $400 \times 400 \times 400$ sparse tensor approximation problem. MLSVD_LTSVD is the fastest but the least accurate one. The other three methods have similar accuracy while MLSVD_FFSRQR is faster when target rank k is larger.

3.4.2.2 Handwritten Digits Classification

MNIST is a handwritten digits image data set created by Yann LeCun [73]. Every digit is represented by a 28×28 pixel image. Handwritten digits classification is to train a classification model to classify new unlabeled images. A HOSVD algorithm is proposed by Savas and Eldén [95] to classify handwritten digits. To reduce the training time, a more efficient ST-HOSVD algorithm is introduced in [109].

We do handwritten digits classification using MNIST which consists of $60,000$ training images and $10,000$ test images. The number of training images in each class is restricted to 5421 so that the training set are equally distributed over all classes. The training set is represented by a tensor $\mathcal{X}$ of size $786 \times 5421 \times 10$. The classification relies on Algorithm 2 in [95]. We use various algorithms to obtain an approximation $\mathcal{X} \approx \mathcal{G} \times_1 U_1 \times_2 U_2 \times_3 U_3$ where the core tensor $\mathcal{G}$ has size $65 \times 142 \times 10$.

The results are summarized in Table 3.3. In terms of run time, our method MLSVD_FFSRQR is comparable to MLSVD_RSI while MLSVD is the most expensive one and MLSVD_LTSVD is the fastest one. In terms of classification quality, MLSVD, MLSVD_FFSRQR, and MLSVD_RSI are comparable while MLSVD_LTSVD is the least accurate one.

3.4.3 Solving Nuclear Norm Minimization Problem

To show the effectiveness of FFSRQR algorithm in nuclear norm minimization problems, we investigate two scenarios: robust PCA (3.5) and matrix completion (3.6). The test matrix

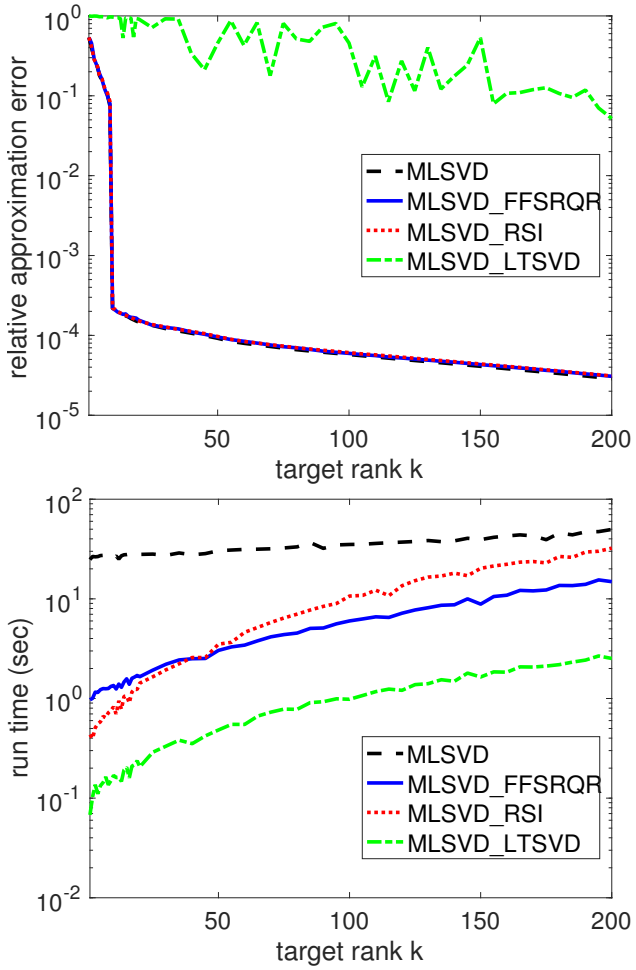

Figure 3.16: Run time and relative approximation error comparison on a sparse tensor.

	MLSVD	MLSVD_FFSRQR	MLSVD_RSI	MLSVD_LTSVD
Training Time [sec]	27.2121	1.5455	1.9343	0.4266
Relative Model Error	0.4099	0.4273	0.4247	0.5162
Classification Accuracy	95.19%	94.98%	95.05%	92.59%

Table 3.3: Comparison on handwritten digits classification.

used in robust PCA is introduced in [75] and the test matrices used in matrix completion are two real data sets. We use IALM method [75] to solve both problems and IALM's code can be downloaded from IALM.

3.4.3.1 Robust PCA

To solve the robust PCA problem, we replace the approximate SVD part in IALM method [75] by various methods. We denote the actual solution to the robust PCA problem by a matrix pair $(X^*, E^*) \in \mathbb{R}^{m\times n} \times \mathbb{R}^{m\times n}$. Matrix $X^* = X_L X_R^T$ where $X_L \in \mathbb{R}^{m\times k},\ X_R \in \mathbb{R}^{n\times k}$ are random matrices where the entries are independently sampled from normal distribution. Sparse matrix E^* is a random matrix where its non-zero entries are independently sampled from a uniform distribution over the interval $[-500, 500]$. The input to the IALM algorithm has the form $M = X^* + E^*$ and the output is denoted by $\left(\widehat{X}, \widehat{E}\right)$. In this numerical experiment, we use the same parameter settings as the IALM code for robust PCA: rank k is $0.1m$ and number of non-zero entries in E is $0.05m^2$. We choose the trade-off parameter $\lambda = 1/\sqrt{\max(m, n)}$ as suggested by Candès et al. [16]. The solution quality is measured by the normalized root mean square error $\|\widehat{X} - X^*\|_F / \|X^*\|_F$.

Table 3.4 includes relative error, run time, the number of non-zero entries in $\widehat{E}$ ($\|\widehat{E}\|_0$), iteration count, and the number of non-zero singular values (#sv) in $\widehat{X}$ of IALM algorithm using different approximate SVD methods. We observe that IALM_FFSRQR is faster than all the other three methods, while its error is comparable to IALM_LANSVD and IALM_RSISVD. IALM_LTSVD is relatively slow and not effective.

3.4.3.2 Matrix Completion

We solve matrix completion problems on two real data sets used in [104]: the Jester joke data set [45] and the MovieLens data set [56]. The Jester joke data set consists of 4.1 million ratings for 100 jokes from $73,421$ users and can be downloaded from the website Jester. We test on the following data matrices:

- jester-1: Data from $24,983$ users who have rated 36 or more jokes;
- jester-2: Data from $23,500$ users who have rated 36 or more jokes;
- jester-3: Data from $24,938$ users who have rated between 15 and 35 jokes;
- jester-all: The combination of jester-1, jester-2, and jester-3.

The MovieLens data set can be downloaded from MovieLens. We test on the following data matrices:

- movie-100K: $100,000$ ratings of 943 users for 1682 movies;
- movie-1M: 1 million ratings of 6040 users for 3900 movies;

Size	Method	Error	Time (sec)	$\|\widehat{E}\|_0$	Iter	#sv
1000×1000	IALM_LANSVD	$3.33e-07$	$5.79e+00$	50000	22	100
	IALM_FFSRQR	$2.79e-07$	$1.02e+00$	50000	25	100
	IALM_RSISVD	$3.36e-07$	$1.09e+00$	49999	22	100
	IALM_LTSVD	$9.92e-02$	$3.11e+00$	999715	100	100
2000×2000	IALM_LANSVD	$2.61e-07$	$5.91e+01$	199999	22	200
	IALM_FFSRQR	$1.82e-07$	$6.93e+00$	199998	25	200
	IALM_RSISVD	$2.63e-07$	$7.38e+00$	199996	22	200
	IALM_LTSVD	$8.42e-02$	$2.20e+01$	3998937	100	200
4000×4000	IALM_LANSVD	$1.38e-07$	$4.65e+02$	799991	23	400
	IALM_FFSRQR	$1.39e-07$	$4.43e+01$	800006	26	400
	IALM_RSISVD	$1.51e-07$	$5.04e+01$	799990	23	400
	IALM_LTSVD	$8.94e-02$	$1.54e+02$	15996623	100	400
6000×6000	IALM_LANSVD	$1.30e-07$	$1.66e+03$	1799982	23	600
	IALM_FFSRQR	$1.02e-07$	$1.42e+02$	1799993	26	600
	IALM_RSISVD	$1.44e-07$	$1.62e+02$	1799985	23	600
	IALM_LTSVD	$8.58e-02$	$5.55e+02$	35992605	100	600

Table 3.4: Comparison on robust PCA.

- movie-latest-small: $100,000$ ratings of 700 users for 9000 movies.

For each data set, we let M be the original data matrix where M_{ij} stands for the rating of joke (movie) j by user i and Γ be the set of indices where M_{ij} is known. The matrix completion algorithm quality is measured by the Normalized Mean Absolute Error (NMAE) which is defined by

$$\text{NMAE} \stackrel{def}{=} \frac{\frac{1}{|\Gamma|}\sum_{(i,j)\in\Gamma}|M_{ij}-X_{ij}|}{r_{\max}-r_{\min}},$$

where X_{ij} is the prediction of the rating of joke (movie) j given by user i, and $r_{\min}, r_{\max}$ are lower and upper bounds of the ratings respectively. For the Jester joke data sets we set $r_{\min} = -10$ and $r_{\max} = 10$. For the MovieLens data sets we set $r_{\min} = 1$ and $r_{\max} = 5$.

Since $|\Gamma|$ is large, we randomly select a subset Ω from Γ and then use Algorithm 14 to solve the problem (3.6). We randomly select 10 ratings for each user in the Jester joke data sets, while we randomly choose about 50% of the ratings for each user in the MovieLens data sets. Table 3.5 includes parameter settings in the algorithms. The maximum iteration number is 100 in IALM, and all other parameters are the same as those used in [75].

The numerical results are included in Table 3.6. We observe that IALM_FFSRQR achieves almost the same recoverability as other methods except for IALM_LTSVD, and is slightly faster than IALM_RSISVD for these two data sets.

Data set	m	n	$\|\Gamma\|$	$\|\Omega\|$
jester-1	24983	100	$1.81e+06$	249830
jester-2	23500	100	$1.71e+06$	235000
jester-3	24938	100	$6.17e+05$	249384
jester-all	73421	100	$4.14e+06$	734210
moive-100K	943	1682	$1.00e+05$	49918
moive-1M	6040	3706	$1.00e+06$	498742
moive-latest-small	671	9066	$1.00e+05$	52551

Table 3.5: Parameters used in the IALM method on matrix completion.

Data set	Method	Iter	Time	NMAE	#sv	$\sigma_{\max}$	σ_{min}
jester-1	IALM-LANSVD	12	$7.06e+00$	$1.84e-01$	100	$2.14e+03$	$1.00e+00$
	IALM-FFSRQR	12	$3.44e+00$	$1.69e-01$	100	$2.28e+03$	$1.00e+00$
	IALM-RSISVD	12	$3.75e+00$	$1.89e-01$	100	$2.12e+03$	$1.00e+00$
	IALM-LTSVD	100	$2.11e+01$	$1.74e-01$	62	$3.00e+03$	$1.00e+00$
jester-2	IALM-LANSVD	12	$6.80e+00$	$1.85e-01$	100	$2.13e+03$	$1.00e+00$
	IALM-FFSRQR	12	$2.79e+00$	$1.70e-01$	100	$2.29e+03$	$1.00e+00$
	IALM-RSISVD	12	$3.59e+00$	$1.91e-01$	100	$2.12e+03$	$1.00e+00$
	IALM-LTSVD	100	$2.03e+01$	$1.75e-01$	58	$2.96e+03$	$1.00e+00$
jester-3	IALM-LANSVD	12	$7.05e+00$	$1.26e-01$	99	$1.79e+03$	$1.00e+00$
	IALM-FFSRQR	12	$3.03e+00$	$1.22e-01$	100	$1.71e+03$	$1.00e+00$
	IALM-RSISVD	12	$3.85e+00$	$1.31e-01$	100	$1.78e+03$	$1.00e+00$
	IALM-LTSVD	100	$2.12e+01$	$1.33e-01$	55	$2.50e+03$	$1.00e+00$
jester-all	IALM-LANSVD	12	$2.39e+01$	$1.72e-01$	100	$3.56e+03$	$1.00e+00$
	IALM-FFSRQR	12	$1.12e+01$	$1.62e-01$	100	$3.63e+03$	$1.00e+00$
	IALM-RSISVD	12	$1.34e+01$	$1.82e-01$	100	$3.47e+03$	$1.00e+00$
	IALM-LTSVD	100	$6.99e+01$	$1.68e-01$	52	$4.92e+03$	$1.00e+00$
moive-100K	IALM-LANSVD	29	$2.86e+01$	$1.83e-01$	285	$1.21e+03$	$1.00e+00$
	IALM-FFSRQR	30	$4.55e+00$	$1.67e-01$	295	$1.53e+03$	$1.00e+00$
	IALM-RSISVD	29	$4.82e+00$	$1.82e-01$	285	$1.29e+03$	$1.00e+00$
	IALM-LTSVD	48	$1.42e+01$	$1.47e-01$	475	$1.91e+03$	$1.00e+00$
moive-1M	IALM-LANSVD	50	$7.40e+02$	$1.58e-01$	495	$4.99e+03$	$1.00e+00$
	IALM-FFSRQR	53	$2.07e+02$	$1.37e-01$	525	$6.63e+03$	$1.00e+00$
	IALM-RSISVD	50	$2.23e+02$	$1.57e-01$	495	$5.35e+03$	$1.00e+00$
	IALM-LTSVD	100	$8.50e+02$	$1.17e-01$	995	$8.97e+03$	$1.00e+00$
moive-latest -small	IALM-LANSVD	31	$1.66e+02$	$1.85e-01$	305	$1.13e+03$	$1.00e+00$
	IALM-FFSRQR	31	$1.96e+01$	$2.00e-01$	305	$1.42e+03$	$1.00e+00$
	IALM-RSISVD	31	$2.85e+01$	$1.91e-01$	305	$1.20e+03$	$1.00e+00$
	IALM-LTSVD	63	$4.02e+01$	$2.08e-01$	298	$1.79e+03$	$1.00e+00$

Table 3.6: Comparison on matrix completion.

3.5 Appendix

3.5.1 Approximate SVD with Randomized Subspace Iteration

Randomized subspace iteration was proposed in [52, Algorithm 4.4] to compute an orthonormal matrix whose range approximates the range of A. An approximate SVD can be computed using the aforementioned orthonormal matrix [52, Algorithm 5.1]. Randomized subspace iteration is used in routine MLSVD_RSI in Matlab toolbox tensorlab [112], and MLSVD_RSI is by far the most efficient function to compute ST-HOSVD we can find in Matlab. We summarize approximate SVD with randomized subspace iteration pseudocode in Algorithm 16.

Algorithm 16 Approximate SVD with Randomized Subspace Iteration

Inputs:
Matrix $A \in \mathbb{R}^{m\times n}$. Target rank k. Oversampling size $p \geq 0$. Number of iterations $q \geq 1$.
Outputs:
$U \in \mathbb{R}^{m\times k}$ contains the approximate top k left singular vectors of A.
$\Sigma \in \mathbb{R}^{k\times k}$ contains the approximate top k singular values of A.
$V \in \mathbb{R}^{n\times k}$ contains the approximate top k right singular vectors of A.
Algorithm:
Generate i.i.d Gaussian matrix $\Omega \in \mathcal{N}(0,1)^{n\times(k+p)}$.
Compute $B = A\Omega$.
$[Q, \sim] = qr(B, 0)$
for $i = 1 : q$ **do**
    $B = A^T * Q$
    $[Q, \sim] = qr(B, 0)$
    $B = A * Q$
    $[Q, \sim] = qr(B, 0)$
end for
$B = Q^T * A$
$[U, \Sigma, V] = svd(B)$
$U = Q * U$
$U = U(:, 1 : k)$
$\Sigma = \Sigma(1 : k, 1 : k)$
$V = V(:, 1 : k)$

Now we perform a complexity analysis on approximate SVD with randomized subspace iteration. We first note that

1. The cost of generating a random matrix is negligible.

2. The cost of computing $B = A\Omega$ is $2mn(k + p)$.

3. In each QR step $[Q, \sim] = qr\,(B, 0)$, the cost of computing the QR factorization of B is $2m\,(k+p)^2 - \frac{2}{3}\,(k+p)^3$ (c.f. [105]), and the cost of forming the first $(k+p)$ columns in the full Q matrix is $m\,(k+p)^2 + \frac{1}{3}\,(k+p)^3$.

Now we count the flops for each i in the for loop:

1. The cost of computing $B = A^T * Q$ is $2mn\,(k+p)$;
2. The cost of computing $[Q, \sim] = qr\,(B, 0)$ is $2n\,(k+p)^2 - \frac{2}{3}\,(k+p)^3$, and the cost of forming the first $(k+p)$ columns in the full Q matrix is $n\,(k+p)^2 + \frac{1}{3}\,(k+p)^3$;
3. The cost of computing $B = A * Q$ is $2mn\,(k+p)$;
4. The cost of computing $[Q, \sim] = qr\,(B, 0)$ is $2m\,(k+p)^2 - \frac{2}{3}\,(k+p)^3$, and the cost of forming the first $(k+p)$ columns in the full Q matrix is $m\,(k+p)^2 + \frac{1}{3}\,(k+p)^3$.

Putting together, the cost of running the for loop q times is

$$q\left(4mn\,(k+p) + 3\,(m+n)\,(k+p)^2 - \frac{2}{3}\,(k+p)^3\right).$$

Additionally, the cost of computing $B = Q^T * A$ is $2mn\,(k+p)$; the cost of doing SVD of B is $O(n\,(k+p)^2)$; and the cost of computing $U = Q * U$ is $2m\,(k+p)^2$.

Now assume $k + p \ll \min\,(m, n)$ and omit the lower-order terms, then we arrive at $(4q+4)\,mn\,(k+p)$ as the complexity of approximate SVD with randomized subspace iteration. In practice, q is usually chosen to be 1 or 2.

Bibliography

[1] Orly Alter, Patrick O Brown, and David Botstein. "Singular value decomposition for genome-wide expression data processing and modeling". In: *Proceedings of the National Academy of Sciences* 97.18 (2000), pp. 10101–10106.

[2] Edward Anderson et al. *LAPACK Users' guide*. Vol. 9. Siam, 1999.

[3] Claus A Andersson and Rasmus Bro. "Improving the speed of multi-way algorithms: Part I. Tucker3". In: *Chemometrics and Intelligent Laboratory Systems* 42.1 (1998), pp. 93–103.

[4] Haim Avron, Petar Maymounkov, and Sivan Toledo. "Blendenpik: Supercharging LAPACK's least-squares solver". In: *SIAM Journal on Scientific Computing* 32.3 (2010), pp. 1217–1236.

[5] Francis R Bach and Michael I Jordan. "Kernel independent component analysis". In: *The Journal of Machine Learning Research* 3 (2003), pp. 1–48.

[6] Francis R Bach and Michael I Jordan. "Predictive low-rank decomposition for kernel methods". In: *Proceedings of the 22nd international conference on Machine learning*. ACM. 2005, pp. 33–40.

[7] K Bache and M Lichman. "UCI Machine Learning Repository [http://archive. ics. uci. edu/ml]. Irvine, CA: University of California, School of Information and Computer Science. Begleiter, H. Neurodynamics Laboratory. State University of New York Health Center at Brooklyn. Ingber, L.(1997). Statistical mechanics of neocortical interactions: Canonical momenta indicatros of electroencephalography". In: *Physical Review E* 55 (2013), pp. 4578–4593.

[8] Michael W Berry, Susan T Dumais, and Gavin W OfffdfffdfffdBrien. "Using linear algebra for intelligent information retrieval". In: *SIAM Review* 37.4 (1995), pp. 573–595.

[9] Christian Bischof and Charles Van Loan. "The WY representation for products of Householder matrices". In: *SIAM Journal on Scientific and Statistical Computing* 8.1 (1987), s2–s13.

[10] L Susan Blackford et al. "An updated set of basic linear algebra subprograms (BLAS)". In: *ACM Transactions on Mathematical Software* 28.2 (2002), pp. 135–151.

[11] L Susan Blackford et al. *ScaLAPACK users' guide*. Vol. 4. siam, 1997.

[12] Christos Boutsidis, Petros Drineas, and Malik Magdon-Ismail. "Near-optimal column-based matrix reconstruction". In: *SIAM Journal on Computing* 43.2 (2014), pp. 687–717.

[13] Peter Businger and Gene H Golub. "Linear least squares solutions by Householder transformations". In: *Numerische Mathematik* 7.3 (1965), pp. 269–276.

[14] Jian-Feng Cai, Emmanuel J Candès, and Zuowei Shen. "A singular value thresholding algorithm for matrix completion". In: *SIAM Journal on Optimization* 20.4 (2010), pp. 1956–1982.

[15] Emmanuel J Candès and Benjamin Recht. "Exact matrix completion via convex optimization". In: *Foundations of Computational Mathematics* 9.6 (2009), pp. 717–772.

[16] Emmanuel J Candès et al. "Robust principal component analysis?" In: *Journal of the ACM (JACM)* 58.3 (2011), 11:1–11:37.

[17] J Douglas Carroll and Jih-Jie Chang. "Analysis of individual differences in multidimensional scaling via an N-way generalization of "Eckart-Young" decomposition". In: *Psychometrika* 35.3 (1970), pp. 283–319.

[18] Chih-Chung Chang and Chih-Jen Lin. "LIBSVM: a library for support vector machines". In: *ACM Transactions on Intelligent Systems and Technology (TIST)* 2.3 (2011), p. 27.

[19] Kenneth L Clarkson and David P Woodruff. "Low rank approximation and regression in input sparsity time". In: *Proceedings of the forty-fifth annual ACM symposium on Theory of computing*. ACM. 2013, pp. 81–90.

[20] Sanjoy Dasgupta and Anupam Gupta. "An elementary proof of a theorem of Johnson and Lindenstrauss". In: *Random structures and algorithms* 22.1 (2003), pp. 60–65.

[21] Timothy A Davis and Yifan Hu. "The University of Florida sparse matrix collection". In: *ACM Transactions on Mathematical Software (TOMS)* 38.1 (2011), 1:1–1:25.

[22] Lieven De Lathauwer and Bart De Moor. "From matrix to tensor: Multilinear algebra and signal processing". In: *Mathematics in Signal Processing IV, J. McWhirter and E. I. Proudler, eds.* Clarendon Press, Oxford, UK. 1998, pp. 1–15.

[23] Lieven De Lathauwer, Bart De Moor, and Joos Vandewalle. "A multilinear singular value decomposition". In: *SIAM journal on Matrix Analysis and Applications* 21.4 (2000), pp. 1253–1278.

[24] Lieven De Lathauwer, Bart De Moor, and Joos Vandewalle. "On the best rank-1 and rank-$(R_1, R_2, \cdots, R_N)$ approximation of higher-order tensors". In: *SIAM journal on Matrix Analysis and Applications* 21.4 (2000), pp. 1324–1342.

[25] Lieven De Lathauwer and Joos Vandewalle. "Dimensionality reduction in higher-order signal processing and rank-$(R_1, R_2, \cdots, R_N)$ reduction in multilinear algebra". In: *Linear Algebra and its Applications* 391 (2004), pp. 31–55.

[26] James W Demmel et al. "Communication avoiding rank revealing QR factorization with column pivoting". In: *SIAM Journal on Matrix Analysis and Applications* 36.1 (2015), pp. 55–89.

[27] James Demmel et al. "Communication-optimal parallel and sequential QR and LU factorizations". In: *SIAM Journal on Scientific Computing* 34.1 (2012), A206–A239.

[28] Amit Deshpande et al. "Matrix approximation and projective clustering via volume sampling". In: *Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm*. Society for Industrial and Applied Mathematics. 2006, pp. 1117–1126.

[29] Petros Drineas, Ravi Kannan, and Michael W Mahoney. "Fast Monte Carlo algorithms for matrices II: Computing a low-rank approximation to a matrix". In: *SIAM Journal on computing* 36.1 (2006), pp. 158–183.

[30] Petros Drineas, Michael W Mahoney, and S Muthukrishnan. "Relative-error CUR matrix decompositions". In: *SIAM Journal on Matrix Analysis and Applications* 30.2 (2008), pp. 844–881.

[31] Petros Drineas et al. "Faster least squares approximation". In: *Numerische Mathematik* 117.2 (2011), pp. 219–249.

[32] Zlatko Drmač and Zvonimir Bujanović. "On the Failure of Rank-Revealing QR Factorization Software–A Case Study". In: *ACM Transactions on Mathematical Software (TOMS)* 35.2 (2008), p. 12.

[33] Jed A Duersch and Ming Gu. "Randomized QR with Column Pivoting". In: *SIAM Journal on Scientific Computing* 39.4 (2017), pp. C263–C291.

[34] Jed A Duersch and Ming Gu. "True BLAS-3 Performance QRCP using Random Sampling". In: *arXiv preprint arXiv:1509.06820* (2015).

[35] Carl Eckart and Gale Young. "The approximation of one matrix by another of lower rank". In: *Psychometrika* 1.3 (1936), pp. 211–218.

[36] Lars Eldén. *Matrix Methods in Data Mining and Pattern Recognition*. SIAM, Philadelphia, 2007.

[37] Maryam Fazel. "Matrix Rank Minimization with Applications". PhD thesis. Stanford University, Stanford, CA, 2002.

[38] Maryam Fazel, Haitham Hindi, and Stephen P Boyd. "Log-det heuristic for matrix rank minimization with applications to Hankel and Euclidean distance matrices". In: *Proceedings of the 2003 American Control Conference*. Vol. 3. 2003, pp. 2156–2162.

[39] Yuehua Feng, Jianwei Xiao, and Ming Gu. "Low-Rank Matrix Approximations with Flip-Flop Spectrum-Revealing QR Factorization." In: *arXiv preprint arXiv:1803.01982* (2018).

[40] Shai Fine and Katya Scheinberg. "Efficient SVM training using low-rank kernel representations". In: *The Journal of Machine Learning Research* 2 (2002), pp. 243–264.

[41] Leslie Foster et al. "Stable and efficient gaussian process calculations". In: *The Journal of Machine Learning Research* 10 (2009), pp. 857–882.

[42] Alan Frieze, Ravi Kannan, and Santosh Vempala. "Fast Monte-Carlo algorithms for finding low-rank approximations". In: *Journal of the ACM (JACM)* 51.6 (2004), pp. 1025–1041.

[43] George W Furnas et al. "Information retrieval using a singular value decomposition model of latent semantic structure". In: *Proceedings of the 11th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*. ACM. 1988, pp. 465–480.

[44] Alex Gittens and Michael W Mahoney. "Revisiting the Nyström method for improved large-scale machine learning". In: *J. Mach. Learn. Res* 28.3 (2013), pp. 567–575.

[45] Ken Goldberg et al. "Eigentaste: A constant time collaborative filtering algorithm". In: *Information Retrieval* 4.2 (2001), pp. 133–151.

[46] Gene Golub. "Numerical methods for solving linear least squares problems". In: *Numerische Mathematik* 7.3 (1965), pp. 206–216.

[47] Gene H Golub and Charles F Van Loan. *Matrix Computations*. 3rd. Johns Hopkins University Press, 2012.

[48] Ming Gu. "Subspace Iteration Randomization and Singular Value Problems". In: *SIAM Journal on Scientific Computing* 37.3 (2015), A1139–A1173.

[49] Ming Gu and Stanley C Eisenstat. "Efficient algorithms for computing a strong rank-revealing QR factorization". In: *SIAM Journal on Scientific Computing* 17.4 (1996), pp. 848–869.

[50] Ming Gu and Luiza Miranian. "Strong rank revealing Cholesky factorization". In: *Electronic Transactions on Numerical Analysis* 17 (2004), pp. 76–92.

[51] Elaine T Hale, Wotao Yin, and Yin Zhang. "Fixed-point continuation for ℓ_1-minimization: Methodology and convergence". In: *SIAM Journal on Optimization* 19.3 (2008), pp. 1107–1130.

[52] Nathan Halko, Per-Gunnar Martinsson, and Joel A Tropp. "Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions". In: *SIAM review* 53.2 (2011), pp. 217–288.

[53] Per Christian Hansen. *Rank-deficient and discrete ill-posed problems: numerical aspects of linear inversion*. Vol. 4. Siam, 1998.

[54] Helmut Harbrecht, Michael Peters, and Reinhold Schneider. "On the low-rank approximation by the pivoted Cholesky decomposition". In: *Applied numerical mathematics* 62.4 (2012), pp. 428–440.

[55] Richard A Harshman. "Foundations of the PARAFAC procedure: Models and conditions for an "explanatory" multimodal factor analysis". In: *UCLA Working Papers in Phonetics* 16 (1970), pp. 1–84. URL: `http://www.psychology.uwo.ca/faculty/harshman/wpppfac0.pdf`.

[56] Jonathan L Herlocker et al. "An algorithmic framework for performing collaborative filtering". In: *Proceedings of the 22nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*. ACM. 1999, pp. 230–237.

[57] Nicholas J Higham. "A survey of condition number estimation for triangular matrices". In: *Siam Review* 29.4 (1987), pp. 575–596.

[58] Nicholas J Higham. *Accuracy and stability of numerical algorithms*. Siam, 2002.

[59] Nicholas J. Higham, Mims Eprint, and Nicholas J. Higham. "Analysis of the Cholesky decomposition of a semi-definite matrix". In: *Reliable Numerical Computation*. University Press, 1990, pp. 161–185.

[60] Roger A Horn and Charles R Johnson. *Topics in Matrix Analysis*. Cambridge University Press, 1991.

[61] D. Huckaby and T. F. Chan. "On the Convergence of Stewart's QLP Algorithm for Approximating the SVD". In: *Numerical Algorithms* 32 (2003), pp. 287–316.

[62] D. Huckaby and T. F. Chan. "Stewart's pivoted QLP decomposition for low-rank matrices". In: *Numerical Linear Algebra with Applications* 12 (2005), pp. 153–159.

[63] William B Johnson and Joram Lindenstrauss. "Extensions of Lipschitz mappings into a Hilbert space". In: *Contemporary Mathematics* 26 (1984), pp. 189–206.

[64] Ian T Jolliffe. *Principal Component Analysis*. Springer-Verlag, New York, 1986.

[65] William Kahan. "Numerical linear algebra". In: *Canadian Math. Bull* 9.6 (1966), pp. 757–801.

[66] Heysem Kaya, Pmar Tüfekci, and Fikret S Gürgen. "Local and global learning methods for predicting power of a combined gas & steam turbine". In: *Proceedings of the International Conference on Emerging Trends in Computer and Electronics Engineering*. 2012, pp. 13–18.

[67] Jon M Kleinberg. "Authoritative sources in a hyperlinked environment". In: *Journal of the ACM (JACM)* 46.5 (1999), pp. 604–632.

[68] Tamara G Kolda. "Orthogonal tensor decompositions". In: *SIAM Journal on Matrix Analysis and Applications* 23.1 (2001), pp. 243–255.

[69] Tamara G Kolda and Brett W Bader. "Tensor decompositions and applications". In: *SIAM Review* 51.3 (2009), pp. 455–500.

[70] R. M. Larsen. *PROPACK - Software for large and sparse SVD calculations*. 1998. URL: `http://sun.stanford.edu/~%20rmunk/PROPACK/`.

[71] Rasmus Munk Larsen. "Lanczos bidiagonalization with partial reorthogonalization". In: *DAIMI Report Series* 27.537 (1998).

[72] Chuck L Lawson et al. "Basic linear algebra subprograms for Fortran usage". In: *ACM Transactions on Mathematical Software (TOMS)* 5.3 (1979), pp. 308–323.

[73] Yann LeCun et al. "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324.

[74] Richard B Lehoucq, Danny C Sorensen, and Chao Yang. *ARPACK Users' Guide: Solution of Large-Scale Eigenvalue Problems with Implicitly Restarted Arnoldi Methods*. SIAM, Philadelphia, 1998.

[75] Zhouchen Lin, Minming Chen, and Yi Ma. *The augmented Lagrange multiplier method for exact recovery of corrupted low-rank matrices*. Sept. 2010. eprint: 1009.5055 (math.NA).

[76] Nathan Linial, Eran London, and Yuri Rabinovich. "The geometry of graphs and some of its algorithmic applications". In: *Combinatorica* 15.2 (1995), pp. 215–245.

[77] Zhang Liu, Anders Hansson, and Lieven Vandenberghe. "Nuclear norm system identification with missing inputs and outputs". In: *Systems & Control Letters* 62.8 (2013), pp. 605–612.

[78] Zhang Liu and Lieven Vandenberghe. "Interior-point method for nuclear norm approximation with application to system identification". In: *SIAM Journal on Matrix Analysis and Applications* 31.3 (2009), pp. 1235–1256.

[79] Craig Lucas. "LAPACK-style codes for level 2 and 3 pivoted Cholesky factorizations". In: *LAPACK Working* (2004).

[80] Shiqian Ma, Donald Goldfarb, and Lifeng Chen. "Fixed point and Bregman iterative methods for matrix rank minimization". In: *Mathematical Programming* 128.1 (2011), pp. 321–353.

[81] Michael W Mahoney. "Randomized algorithms for matrices and data". In: *Foundations and Trends® in Machine Learning* 3.2 (2011), pp. 123–224.

[82] Per-Gunnar Martinsson et al. "Householder QR Factorization With Randomization for Column Pivoting (HQRRP)". In: *SIAM Journal on Scientific Computing* 39.2 (2017), pp. C96–C115.

[83] Xiangrui Meng et al. "Mllib: Machine learning in apache spark". In: *arXiv preprint arXiv:1505.06807* (2015).

[84] Mehran Mesbahi and George P Papavassilopoulos. "On the rank minimization problem over a positive semidefinite linear matrix inequality". In: *IEEE Transactions on Automatic Control* 42.2 (1997), pp. 239–243.

[85] Neil Muller, Lourenço Magaia, and Ben M Herbst. "Singular value decomposition, eigenfaces, and 3D reconstructions". In: *SIAM Review* 46.3 (2004), pp. 518–545.

[86] Manish Narwaria and Weisi Lin. "SVD-based quality metric for image and video using machine learning". In: *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)* 42.2 (2012), pp. 347–364.

[87] Tae-Hyun Oh et al. "Fast randomized singular value thresholding for nuclear norm minimization". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2015, pp. 4484–4493.

[88] Fabian Pedregosa et al. "Scikit-learn: Machine learning in Python". In: *The Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[89] Gregorio Quintana-Ortí, Xiaobai Sun, and Christian H Bischof. "A BLAS-3 version of the QR factorization with column pivoting". In: *SIAM Journal on Scientific Computing* 19.5 (1998), pp. 1486–1494.

[90] Benjamin Recht, Maryam Fazel, and Pablo A Parrilo. "Guaranteed minimum-rank solutions of linear matrix equations via nuclear norm minimization". In: *SIAM Review* 52.3 (2010), pp. 471–501.

[91] Jasson DM Rennie and Nathan Srebro. "Fast maximum margin matrix factorization for collaborative prediction". In: *Proceedings of the 22nd International Conference on Machine Learning*. ACM. 2005, pp. 713–719.

[92] Jorge-L Reyes-Ortiz et al. "Transition-aware human activity recognition using smartphones". In: *Neurocomputing* 171 (2016), pp. 754–767.

[93] Vladimir Rokhlin, Arthur Szlam, and Mark Tygert. "A randomized algorithm for principal component analysis". In: *SIAM Journal on Matrix Analysis and Applications* 31.3 (2009), pp. 1100–1124.

[94] Arvind K Saibaba. "HOID: Higher Order Interpolatory Decomposition for tensors based on Tucker representation". In: *SIAM Journal on Matrix Analysis and Applications* 37.3 (2016), pp. 1223–1249.

[95] Berkant Savas and Lars Eldén. "Handwritten digit classification using higher order singular value decomposition". In: *Pattern recognition* 40.3 (2007), pp. 993–1003.

[96] Robert Schreiber and Charles Van Loan. "A storage-efficient WY representation for products of Householder transformations". In: *SIAM Journal on Scientific and Statistical Computing* 10.1 (1989), pp. 53–57.

[97] Amnon Shashua and Tamir Hazan. "Non-negative tensor factorization with applications to statistics and computer vision". In: *Proceedings of the 22nd International Conference on Machine Learning*. ACM. 2005, pp. 792–799.

[98] Nicholas D Sidiropoulos, Rasmus Bro, and Georgios B Giannakis. "Parallel factor analysis in sensor array processing". In: *IEEE transactions on Signal Processing* 48.8 (2000), pp. 2377–2388.

[99] SimonDu. *CUR-matrix-decomposition*. `https://github.com/SimonDu/CUR-matrix-decomposition`. 2014.

[100] Danny C Sorensen and Mark Embree. "A DEIM induced CUR factorization". In: *SIAM Journal on Scientific Computing* 38.3 (2016), A1454–A1482.

[101] Nathan Srebro, Jason Rennie, and Tommi S Jaakkola. "Maximum-margin matrix factorization". In: *Advances in neural information processing systems*. 2005, pp. 1329–1336.

[102] GW Stewart. "The QLP approximation to the singular value decomposition". In: *SIAM Journal on Scientific Computing* 20.4 (1999), pp. 1336–1348.

[103] Qingtang Su et al. "Color image blind watermarking scheme based on QR decomposition". In: *Signal Processing* 94 (2014), pp. 219–235.

[104] Kim-Chuan Toh and Sangwoon Yun. "An accelerated proximal gradient algorithm for nuclear norm regularized linear least squares problems". In: *Pacific Journal of Optimization* 6.15 (2010), pp. 615–640.

[105] Lloyd N Trefethen and David Bau III. *Numerical Linear Algebra*. Vol. 50. SIAM, Philadelphia, 1997.

[106] Ledyard R Tucker. "Some mathematical notes on three-mode factor analysis". In: *Psychometrika* 31.3 (1966), pp. 279–311.

[107] Pınar Tüfekci. "Prediction of full load electrical power output of a base load operated combined cycle power plant using machine learning methods". In: *International Journal of Electrical Power & Energy Systems* 60 (2014), pp. 126–140.

[108] Matthew Turk and Alex Pentland. "Eigenfaces for recognition". In: *Journal of Cognitive Neuroscience* 3.1 (1991), pp. 71–86.

[109] Nick Vannieuwenhoven, Raf Vandebril, and Karl Meerbergen. "A new truncation strategy for the higher-order singular value decomposition". In: *SIAM Journal on Scientific Computing* 34.2 (2012), A1027–A1052.

[110] M Alex O Vasilescu and Demetri Terzopoulos. "Multilinear analysis of image ensembles: Tensorfaces". In: *European Conference on Computer Vision*. Springer. 2002, pp. 447–460.

[111] Santosh S Vempala. *The random projection method*. Vol. 65. American Mathematical Soc., 2005.

[112] Nico Vervliet et al. *Tensorlab 3.0*. 2016. URL: `http://www.tensorlab.net`.

[113] Endong Wang et al. "Intel math kernel library". In: *High-Performance Computing on the Intel Xeon Phifffdfffdfffd*. Springer, 2014, pp. 167–188.

[114] Hongcheng Wang and Narendra Ahuja. "Facial expression decomposition". In: *Proceedings of the 9th IEEE International Conference on Computer Vision (ICCV)*. 2003, pp. 958–965.

[115] Shusen Wang and Zhihua Zhang. "Improving CUR matrix decomposition and the Nyström approximation via adaptive sampling". In: *The Journal of Machine Learning Research* 14.1 (2013), pp. 2729–2769.

[116] Christopher KI Williams and Carl Edward Rasmussen. "Gaussian processes for machine learning". In: *the MIT Press* 2.3 (2006), p. 4.

[117] Christopher Williams and Matthias Seeger. "The effect of the input density distribution on kernel-based classifiers". In: *Proceedings of the 17th international conference on machine learning*. EPFL-CONF-161323. 2000, pp. 1159–1166.

[118] Svante Wold et al. "The collinearity problem in linear regression. The partial least squares (PLS) approach to generalized inverses". In: *SIAM Journal on Scientific and Statistical Computing* 5.3 (1984), pp. 735–743.

[119] David P Woodruff. "Sketching as a tool for numerical linear algebra". In: *arXiv preprint arXiv:1411.4357* (2014).

[120] Jianwei Xiao and Ming Gu. "Spectrum-Revealing Cholesky Factorization for Kernel Methods". In: *Proceedings of the 16th IEEE International Conference on Data Mining (ICDM)*. 2016, pp. 1293–1298.

[121] Jianwei Xiao, Ming Gu, and Julien Langou. "Fast Parallel Randomized QR with Column Pivoting Algorithms for Reliable Low-rank Matrix Approximations". In: *Proceedings of the 24th IEEE International Conference on High Performance Computing (HiPC)*. 2017, pp. 233–242.

[122] Tong Zhang and Gene H Golub. "Rank-one approximation to high order tensors". In: *SIAM Journal on Matrix Analysis and Applications* 23.2 (2001), pp. 534–550.