Title
An Efficient, Tolerance-Based Algorithm for the Truncated SVD

Author
Yeh, Michael

Publication Date
2022

Peer reviewed|Thesis/dissertation

An Efficient, Tolerance-Based Algorithm for the Truncated SVD

by

Michael Yeh

A dissertation submitted in partial satisfaction of the

requirements for the degree of

Doctor of Philosophy

in

Mathematics

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Professor Ming Gu, Chair
Professor Daniel Tataru
Professor Jon Wilkening

Summer 2022

An Efficient, Tolerance-Based Algorithm for the Truncated SVD

Abstract

An Efficient, Tolerance-Based Algorithm for the Truncated SVD

by

Michael Yeh

Doctor of Philosophy in Mathematics

University of California, Berkeley

Professor Ming Gu, Chair

The truncated singular value decomposition (TSVD) is an important low-rank matrix approximation technique used in data science, machine learning, numerical linear algebra, and many other scientific fields. However, it is quite expensive for large matrices when only a very low-rank approximation is needed. Existing algorithms for TSVD typically assume the user knows the proper rank to use, but in practice, they may not know this rank beforehand. Thus, it is important to have versions of these algorithms that depend on a tolerance parameter, allowing the user to directly control the approximation error. We develop one such algorithm that runs quickly for matrices with rapidly decaying singular values, provide approximation error bounds that are within a constant factor away from optimal, and demonstrate its utility with matrices from a variety of applications.

To my family and friends.

Contents

List of Figures

List of Tables

Acknowledgments

I would like to thank my advisor, Prof. Gu, for his constant support and encouragement. This dissertation would not have been possible without him, and I could not have asked for a better mentor. I would also like to thank Jed Duersch for kindly providing me with his code for RQRCP.

Chapter 1

Introduction and Background

1.1 Low-rank matrix approximation

Low-rank matrix approximation is an important dimensionality reduction technique used in data science and machine learning. It is becoming increasingly important as some applications require larger and larger data that traditional methods cannot adequately handle. Typically, low-rank methods approximate a given matrix as a product of several smaller matrices and thus can be used to reduce memory costs and time needed to train models.

One of the most important low-rank matrix approximations is the truncated singular value decomposition (TSVD). Let A be an $m \times n$ real matrix and $A = U\Sigma V^T$ its SVD. The rank-k TSVD of A is the matrix $A_k := U_k \Sigma_k V_k^T$, where U_k and V_k are the first k columns of U and V, respectively, and Σ_k is the leading $k \times k$ block of Σ. The columns u_j and v_j of U_k and V_k are the top k left and right singular vectors of A, respectively, and the diagonal entries $\sigma_1(A) \geq \cdots \geq \sigma_k(A) \geq 0$ of Σ_k are the top k singular values of A. The importance of TSVD stems from the following theorem:

Theorem 1 (Eckart-Young [20])**.**

$$\min_{\text{rank}(B)\leq k} \|A - B\|_2 = \sigma_{k+1}(A)$$

$$\min_{\text{rank}(B)\leq k} \|A - B\|_F = \sqrt{\sum_{i=k+1}^{\text{rank}(A)} \sigma_i^2(A)}$$

In both cases, the minimum is attained by the rank-k TSVD A_k of A.

In other words, TSVD is, in a sense, the "optimal" low-rank approximation, and this is the reason it occupies a central position among low-rank approximation algorithms.

One classic use of TSVD in data science and machine learning is principal component analysis (PCA) [22]. PCA is really just TSVD, but k is determined using a statistical interpretation. Suppose a dataset consists of m data points in $\mathbb{R}^n$ so that each point is

described by n "features." PCA aims to find linear combinations of these features which account for most of the variation in the data. We may arrange the data points into an $m \times n$ matrix A, each row corresponding to one point. We also assume that the features have been centered so that each column of A has mean 0. Then $A^T A$ is the covariance matrix for the n features, and we can ask in which direction the data vary the most, i.e. find a unit vector v that maximizes $v^T A^T A v$. By replacing A with its SVD, we see that the first right singular vector v_1 of A maximizes $v^T A^T A v$. We can then ask for the direction in which the data vary the second most. To ensure that this direction is independent from v_1, we require that it be orthogonal to v_1. Using the SVD again, we find that this direction is the second right singular vector v_2 of A. Continuing this process, we get v_3, v_4, etc.

When do we stop? Since the v_i are independent, the variance of the data in the first k directions is just

$$\sum_{i=1}^{k} v_i^T A^T A v_i = \sum_{i=1}^{k} \sigma_i(A)^2.$$

Typically, k is chosen so that this variance is a large percentage of the total variance, i.e.

$$\frac{\Sigma_{i=1}^{k} \sigma_i(A)^2}{\sum_{i=1}^{n} \sigma_i(A)^2} \geq 1 - \varepsilon,$$

where ε is small. For example, if the user wishes to capture 99% of the total variance, then we set $\varepsilon = 0.01$. Once we find k, the final approximation that PCA yields is then the rank-k TSVD A_k of A.

A general application of TSVD is in compressing data. For instance, storing an $m \times n$ image directly requires $O(mn)$ space, while storing its rank-k TSVD requires $O((m+n)k)$ space, which is much smaller when $k \ll \min\{m, n\}$. Similarly, training data for machine learning algorithms can be compressed in this way to speed-up training times. Turk and Pentland [42] used TSVD to extract important facial features from a database of facial images (essentially, the "principal components" of a face) and applied this to facial recognition. Deerwester et al. [13] used TSVD to extract semantic information from a collection of documents, a technique known as latent semantic analysis. Given such a collection, they first construct a $t \times d$ term-document matrix A, where each row of A corresponds to a word and each column of A to a document. The entry A_{ij} is then the number of times that word i appears in document j and may be weighted as well. They then compute the rank-k TSVD of A for a suitable number k; in their paper, they use $k \approx 100$. The singular vectors then allow them to measure the semantic similarity of different terms and documents and also retrieve documents that are conceptually related to a queried set of terms.

Besides TSVD, there are a host of other low-rank approximation techniques based on other types of factorizations. A notable one that is popular in data analysis is the CUR decomposition [35], which computes an approximation of the form $A \approx CUR$. Here, C and R contain actual columns and rows, respectively, of A, and usually $U = C^\dagger A R^\dagger$. This

decomposition was inspired by the SVD and was designed to address two of its flaws. Since C and R are actual columns and rows of A (and therefore correspond to actual data points and features), the factorization CUR is more easily interpreted than U and V from the SVD of A. Moreover, the CUR decomposition preserves the sparsity of the original matrix. If A is sparse, then so are C and R, while U and V in general will not be. In [35], Mahoney and Drineas apply this decomposition to analyze term-document matrices and DNA microarray data.

Low-rank approximation has also been used to simplify machine learning models. For instance, Xue, Li, and Gong [46] substantially reduce the size of a large neural network used for speech recognition by approximating its weight matrices with two smaller matrices constructed using TSVD. Despite this substantial reduction in model size, they manage to maintain a level of accuracy similar to that of the original model. Another example is the Nyström method in kernel learning [43]. Kernel-based methods involve an $n \times n$ symmetric positive-semidefinite matrix G, where n is the number of training samples, and cost $O(n^3)$ time to train. By replacing G with a rank-k approximation, this training time can be reduced to $O(k^2 n)$. Williams and Seeger [43] construct their approximation by randomly selecting k columns from G to form an $n \times k$ matrix C and get the resulting approximation $G \approx CW^{-1}C^T$, where W is the "intersection" of C and C^T in G. They train two classifiers with their method and show empirically there is no significant loss in accuracy.

Besides data science and machine learning, low-rank approximation is also used in numerical linear algebra. One classic use case is solving rank-deficient least squares problems: given an $m \times n$ matrix A that is possibly rank-deficient and a vector $b \in \mathbb{R}^m$, we wish to solve $\min_{x \in \mathbb{R}^n} \|Ax - b\|_2$. If A is rank-deficient or close to rank-deficient, the solution x will be sensitive to small perturbations in the data A and b. To regularize the solution, a modified version of the problem is solved instead, with A replaced by a low-rank approximation to A.

The LAPACK [3] routine xGELSY solves rank-deficient least squares problems in this way, where a user-supplied parameter RCOND determines the rank of A. The first part of this routine computes the pivoted QR factorization of $A = QR\Pi^T$ and estimates the condition number of each leading block $R(1:i, 1:i)$, $i = 1, 2, \ldots$. The rank k of A is then the largest i for which the estimated condition number of $R(1:i, 1:i)$ does not exceed 1/RCOND, and the low-rank approximation is obtained by setting $R(k+1:m, k+1:n)$ to 0.

Another approach to solving rank-deficient least squares problems uses TSVD [14]. In floating-point arithmetic, the smallest singular values of a rank-deficient matrix usually will not be exactly 0, but rather $O(\varepsilon_{mach}\|A\|_2)$, where ε_{mach} is machine epsilon. Since these singular values are on the order of the round-off error, we may justifiably set these equal to 0 to obtain a low-rank approximation of A and then solve the least squares problem. More generally, errors in the entries of A may arise not only from round-off error, but also from sources such as measurement error and noise. If the user has some bound tol on the amount of error in A, then they may set any singular values smaller than tol to 0 since this perturbation is no larger than the inherent uncertainty in A. Thus, A is replaced with its rank-k TSVD A_k, where k satisfies $\sigma_k(A) \geq \text{tol} > \sigma_{k+1}(A)$.

1.2 Low-rank approximation algorithms

Rank-revealing QR

Low-rank approximations can be constructed from so-called rank-revealing factorizations. Perhaps one of the simplest examples of such a factorization is QR with column pivoting (QRCP) [6]. Consider a matrix $A \in \mathbb{R}^{m\times n}$. At the ith stage of QRCP, we have

$$A\Pi = QR = Q\begin{pmatrix} R_{11} & R_{12} \\ 0 & R_{22} \end{pmatrix}, \tag{1.1}$$

where Π is a permutation matrix, Q is orthogonal, and $R_{11} \in \mathbb{R}^{i\times i}$ is upper triangular. We then find $j^* = \arg\max_{j=i+1,\dots,n} \|R_{22}(i+1:m, j)\|_2$, swap columns j^* and $i+1$ of R, and apply a Householder reflection to R to zero out $R(i+2:m, i+1)$. We continue until R is upper triangular. This pivoting strategy is greedy in that at each step, it finds the column that will make $\det(R_{11})$ as large as possible.

Now, suppose in exact arithmetic that A has rank k. One important property for the above pivot selection scheme is that $|r_{ii}| \geq \|R(i:m, j)\|_2$ for $j = i+1, \dots, n$. From this, it follows that the trailing block $R(k+1:m, k+1:n)$ of R will be 0. Thus, we obtain a compact representation of A by keeping the first k columns of Q and the first k rows of R.

However, in floating point arithmetic, the situation is more complicated. In general, a rank-deficient matrix will not have singular values *exactly* zero. Rather, they will generally be on the order of $\varepsilon_{mach}\|A\|_2$, where ε_{mach} is machine epsilon. In this case, we would hope that the norm of the trailing block will be $O(\varepsilon_{mach}\|A\|_2)$ so that we could again obtain a compact representation of A, this time with an error on the order of the round-off error.

More generally, suppose there is a large relative gap between $\sigma_k(A)$ and $\sigma_{k+1}(A)$. Partition R as in Equation 1.1 so that R_{11} is $k \times k$. By Theorem 1, $\|R_{22}\|_2 \geq \sigma_{k+1}(A)$, and we also know that $\sigma_k(R_{11}) \leq \sigma_k(A)$ by the Cauchy Interlacing Theorem. Thus, if we have $\|R_{22}\|_2 \approx \sigma_{k+1}(A)$ and $\sigma_k(R_{11}) \approx \sigma_k(A)$, then the R factor will have revealed the gap between $\sigma_k(A)$ and $\sigma_{k+1}(A)$ and thus the (numerical) rank of A. Chan first considered rank-revealing QR factorizations in [8]. Gu and Eisenstat [26] formally define a rank-revealing QR factorization to be one which satisfies

$$\sigma_k(R_{11}) \geq \frac{\sigma_k(A)}{p(k,n)} \quad \text{and} \quad \|R_{22}\|_2 \leq p(k,n)\sigma_{k+1}(A),$$

where $p(k, n)$ is bounded by some low-degree polynomial in k and n. Luckily, for most matrices that arise in practical applications, QRCP does indeed reveal the rank in this way.

However, there are some pathological cases where QRCP fails. A well-known example is the $n \times n$ Kahan matrix [32]:

$$K := K(c, s, n) = \begin{pmatrix} 1 & 0 & \cdots & 0 \\ 0 & s & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & s^{n-1} \end{pmatrix} \begin{pmatrix} 1 & -c & \cdots & -c \\ 0 & 1 & \ddots & \vdots \\ \vdots & \ddots & \ddots & -c \\ 0 & \cdots & 0 & 1 \end{pmatrix}, \quad c^2 + s^2 = 1.$$

Kahan shows that K requires no pivoting, so it remains unchanged after QRCP. Furthermore, he shows that

$$\frac{K_{nn}}{\sigma_n(K)} = O\left((1+c)^{n-1}\right).$$

Thus, QRCP can fail to reveal the rank of K by an arbitrarily large factor if we take $k = n-1$.

To address this failure, Gu and Eisenstat [26] propose extra column swaps. Given $f \geq 1$, we swap one of the first k columns of R with one of its last $n-k$ columns to increase $\det(R_{11})$ by a factor of at least f until no such swap exists. This algorithm can be performed in $O((m+n\log_f n)n^2)$ flops. The resulting R factor satisfies properties stronger than those of a rank-reveal QR factorization. They show that there are functions $q_1(k,n) \leq \sqrt{1+f^2k(n-k)}$ and $q_2(k,n) \leq f$ such that for $1 \leq i \leq k$ and $1 \leq j \leq n-k$:

$$\sigma_i(R_{11}) \geq \frac{\sigma_i(A)}{q_1(k,n)}, \quad \sigma_j(R_{22}) \leq q_1(k,n)\sigma_{k+j}(A), \quad \text{and} \quad \left|(R_{11}^{-1}R_{12})_{ij}\right| \leq q_2(k,n).$$

Thus, not only is the rank of A revealed, but the singular values of the leading and trailing blocks R_{11} and R_{22} are accurate up to a small polynomial factor.

Randomized QR with column pivoting

As we have seen, QRCP is used frequently in rank-revealing algorithms and low-rank matrix approximation. However, the communication costs of pivoting make it considerably more expensive than unpivoted QR. Duersch and Gu [19] developed randomized QR with column pivoting (RQRCP) to reduce these communication costs while maintaining a similar pivot quality as QRCP. Rather than using the columns of the original matrix A to make pivot decisions, they draw a random Gaussian matrix Ω of size $l \times m$, where $l \ll m$, and instead use the columns of the compressed matrix ΩA. The reduction in column size from m to l leads to a significant improvement in performance, nearing that of unpivoted QR.

Spectrum-revealing QR

Another type of RRQR algorithm called spectrum-revealing QR (SRQR) was proposed by Xiao, Gu, and Langou in [45]. It satisfies stronger rank-revealing properties than RRQR in that for matrices with rapidly decaying singular values, the singular values of the approximation are accurate up to a nontrivial number of digits.

To achieve a good approximation, we "oversample" the target rank k by first computing a rank-ℓ ($\ell \geq k$) truncated QR factorization $\tilde{R}$, and then computing the rank-k truncated SVD of $\tilde{R}$. Thus, consider an ℓ-step partial QR factorization (QRCP or RQRCP) of A where $\ell \geq k$:

$$A\Pi = Q\begin{pmatrix} R_{11} & R_{12} \\ 0 & R_{22} \end{pmatrix}, \quad R_{11} \in \mathbb{R}^{\ell\times\ell}$$

Define

$$\tilde{R} := \begin{pmatrix} R_{11} & R_{12} \end{pmatrix}, \quad \hat{R} := \begin{pmatrix} R_{11} & a \\ 0 & \alpha \end{pmatrix},$$

where $\hat{R}$ is the leading $(\ell+1) \times (\ell+1)$ block of R after performing the next step of QRCP or RQRCP. Xiao et al. show that for $1 \le j \le k$,

$$\sigma_j(\tilde{R}) \ge \frac{\sigma_j(A)}{\sqrt{1 + \left(\frac{\|R_{22}\|_2}{\sigma_k(\tilde{R})}\right)^2}} \quad \text{and} \quad \sigma_j(\tilde{R}) \ge \frac{\sigma_j(A)}{\sqrt{1 + \left(1 + \frac{\|R_{22}\|_2^2}{\sigma_k^2(\tilde{R})}\right) \frac{\|R_{22}\|_2^2}{\sigma_j^2(A)}}},$$

and the truncation error satisfies

$$\|A\Pi - Q \begin{pmatrix} \tilde{R}_k \\ 0 \end{pmatrix} \|_2 \le \sigma_{k+1}(A) \sqrt{1 + \left(\frac{\|R_{22}\|_2}{\sigma_{k+1}(A)}\right)^2},$$

where $\tilde{R}_k$ is the rank-k TSVD of $\tilde{R}$. These inequalities show that we can get a good approximation if $\|R_{22}\|_2 \approx \sigma_{\ell+1}(A)$ and A has rapidly decaying singular values. Thus the goal of SRQR is to find a column permutation that ensures $\|R_{22}\|_2 = O(\sigma_{\ell+1}(A))$. Accordingly, SRQR is based on the Hybrid-II(ℓ) RRQR algorithm of Chandrasekaran and Ipsen [9]. This latter algorithm finds a permutation Π that approximately solves $\min_\Pi \|R_{22}\|_2$; on exit it guarantees that $\|R_{22}\|_2 \le \sigma_{\ell+1}(A)\sqrt{(\ell+1)(n-\ell)}$.

For any matrix X, let $\|X\|_{1,2}$ denote the largest column 2-norm of X, and further define the following quantities:

$$g_1 := \frac{\|R_{22}\|_{1,2}}{|\alpha|}, \quad g_2 := |\alpha| \, \|\hat{R}^{-T}\|_{1,2},$$
$$\tau := g_1 g_2 \frac{\|R_{22}\|_2}{\|R_{22}\|_{1,2}} \frac{\|\hat{R}^{-T}\|_{1,2}^{-1}}{\sigma_{\ell+1}(A)}, \quad \bar{\tau} := g_1 g_2 \frac{\|R_{22}\|_2}{\|R_{22}\|_{1,2}} \frac{\|R_{11}^{-T}\|_{1,2}^{-1}}{\sigma_k(\tilde{R})}$$

Xiao et al. show that the singular values of the truncated matrix $\tilde{R}$ satisfy

$$\sigma_j(\tilde{R}) \ge \frac{\sigma_j(A)}{\sqrt{1 + \min\left(\bar{\tau}^2, \tau^2(1+\bar{\tau}^2)\left(\frac{\sigma_{\ell+1}(A)}{\sigma_j(A)}\right)^2\right)}}, \quad 1 \le j \le k,$$

and the truncation error satisfies

$$\|A\Pi - Q \begin{pmatrix} \tilde{R}_k \\ 0 \end{pmatrix} \|_2 \le \sigma_{k+1}(A) \sqrt{1 + \tau^2 \left(\frac{\sigma_{\ell+1}(A)}{\sigma_{k+1}(A)}\right)^2}.$$

Thus, the singular values and truncation error will be accurate up to a small relative error provided τ and $\bar{\tau}$ are not too large and the singular values of A decay quickly. The authors prove that τ and $\bar{\tau}$ are bounded by $g_1 g_2$ times a dimension dependent constant:

$$\tau \leq g_1 g_2 \sqrt{(\ell+1)(n-\ell)}, \quad \bar{\tau} \leq g_1 g_2 \sqrt{\ell(n-\ell)},$$

but note that in practice, the two ratios in τ and $\bar{\tau}$ are modest in size. They further prove that for RQRCP,

$$g_1 \leq \sqrt{\frac{1+\varepsilon}{1-\varepsilon}} \quad \text{and} \quad g_2 \leq \frac{\sqrt{2(1+\varepsilon)}}{1-\varepsilon}\left(1+\sqrt{\frac{1+\varepsilon}{1-\varepsilon}}\right)^{\ell-1}.$$

where ε is typically $\ll 1$. Thus the only potentially large factor in τ and $\bar{\tau}$ is g_2, which could be exponentially large. So, we would like to perform extra column swaps to control the size of g_2. The authors use a user-selected threshold $g > 1$ to achieve this, terminating the algorithm once $g_2 < g$. The way in which extra column swaps are performed in SRQR is similar to Hybrid-II(ℓ), but with some randomization added to make it more efficient. In [9], the authors show that Hybrid-II(ℓ) eventually stops, which is equivalent to having $g_2 = 1$. Thus, SRQR does indeed terminate eventually, but may terminate earlier than Hybrid-II if g is large enough. Xiao et al. show that, compared to QRCP, SRQR is much faster and provides a higher quality low-rank approximation.

The QLP decomposition

The basis of our next spectrum-revealing algorithm is the QLP decomposition [40]. Let A be an $m \times n$ matrix. Perform QRCP on A to obtain $A\Pi = QR$ and then perform QRCP on R^T to get $R^T\Pi_1 = PL^T$, where L is lower triangular. Putting these together yields $A = Q\Pi_1 L P^T \Pi^T$. This is the pivoted QLP decomposition of A. Stewart observed that the diagonal entries l_{ii} of L closely track the singular values of A.

To partially explain this behavior, consider extending the QLP decomposition into an iterative procedure. Compute the QR factorization $A = Q_0 R_0$. Then for $i = 1, 2, \ldots$, we get Q_i and R_i by computing the QR factorization $R_{i-1}^T = Q_i R_i$. Huckaby and Chan [29] show that this iterative procedure corresponds to performing QR iteration on $R_0^T R_0$ and so, the R_i converge to a diagonal matrix whose diagonal elements are the singular values of R_0 (which are the same as those of A) in descending order.

They also perform some analysis on the convergence rate of the singular values of L_{11} and L_{22}, which are the leading $k \times k$ and trailing $(n-k) \times (n-k)$ diagonal blocks of L, respectively, when there is a large gap between $\sigma_k(A)$ and $\sigma_{k+1}(A)$, and conclude that when $\sigma_{k+1}(A)/\sigma_k(A)$ is small, the convergence is quite fast, thus partially explaining the accuracy of the QLP decomposition. Because of this quick convergence, we might expect stronger rank-revealing properties than RRQR or SRQR and a more accurate low-rank approximation by truncating the trailing block L_{22} of L rather than the trailing block R_{22} of R. Indeed, this observation leads to the following algorithm, flip-flop QR.

Flip-flop spectrum-revealing QR

Following SRQR, Feng et al. [23] proposed a spectrum-revealing algorithm based on the QLP decomposition called flip-flop QR (FFQR). For a matrix A and integers $k \leq \ell$, FFQR produces an approximation to the rank-k TSVD A_k in $O(mnl)$ time whose accuracy depends on the ratio $\sigma_{\ell+1}(A)/\sigma_{k+1}(A)$. Thus, if A has rapidly decaying singular values, FFQR will be close to TSVD.

FFQR is computed as follows. Let A be an $m \times n$ matrix ($m \geq n$) and $k \leq \ell$. Perform ℓ steps of RQRCP to get the factorization

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

where Π is an $n \times n$ permutation matrix, Q is an $m \times m$ orthogonal matrix, R is $m \times n$, and R_{11} is an $\ell \times \ell$ upper triangular matrix.

The next phase of FFQR involves performing extra SRQR column swaps on R and using Givens rotations to restore its upper trapezoidal form. Then, perform ℓ steps of QR on R^T to get

$$R^T = PL^T = \begin{pmatrix} P_1 & P_2 \end{pmatrix} \begin{pmatrix} L_{11} & 0 \\ L_{21} & L_{22} \end{pmatrix}^T,$$

where P is an $n \times n$ orthogonal matrix, P_1 is its leading ℓ columns, L is an $m \times n$ matrix, and L_{11} is $\ell \times \ell$ lower triangular. Putting the above together yields

$$A = QR\Pi^T = Q \begin{pmatrix} L_{11} & 0 \\ L_{21} & L_{22} \end{pmatrix} P^T \Pi^T.$$

Observe that unlike the QLP factorization described above, where we pivot when factoring A and R^T, in FFQR we pivot only when factoring A. This is because QRCP on R^T would require us to know all the columns of R^T, not just the first ℓ. Thus, by not pivoting in the second factorization, we can save some computational cost without sacrificing approximation quality. Now, discard L_{22} and approximate $\begin{pmatrix} L_{11}^T & L_{21}^T \end{pmatrix}^T$ with its rank-k TSVD $\hat{U}_k \hat{\Sigma}_k \hat{V}_k^T$:

$$\begin{aligned} Q \begin{pmatrix} L_{11} & 0 \\ L_{21} & L_{22} \end{pmatrix} P^T \Pi^T &\approx Q \begin{pmatrix} L_{11} \\ L_{21} \end{pmatrix} P_1^T \Pi^T \\ &\approx Q(\hat{U}_k \hat{\Sigma}_k \hat{V}_k^T) P_1^T \Pi^T \end{aligned}$$

Setting $\tilde{U}_k := Q\hat{U}_k$, $\tilde{\Sigma}_k := \hat{\Sigma}_k$, $\tilde{V}_k := \Pi P_1 \hat{V}_k$ gives the rank-k approximation $A \approx \tilde{U}_k \tilde{\Sigma}_k \tilde{V}_k^T$.

In [23], the authors prove the following bounds for FFQR that are analogous to the ones for SRQR. The most important difference is the presence of fourth powers rather than second powers. Given $\varepsilon > 0$ and $g > 1$, there are matrix-dependent quantities $g_1 \leq \sqrt{(1+\varepsilon)/(1-\varepsilon)}$, $g_2 \leq g$, $\tau \leq g_1 g_2 \sqrt{(\ell+1)(n-\ell)}$, and $\hat{\tau} \leq g_1 g_2 \sqrt{\ell(n-\ell)}$ such

that for $1 \leq j \leq k$,

$$\sigma_j(\tilde{\Sigma}_k) \geq \frac{\sigma_j(A)}{\sqrt[4]{1 + \min\left(2\hat{\tau}^4, \tau^4(2 + 4\hat{\tau}^4)\left(\frac{\sigma_{\ell+1}(A)}{\sigma_j(A)}\right)^4\right)}} \tag{1.2}$$

and

$$\|A - \tilde{U}_k\tilde{\Sigma}_k\tilde{V}_k^T\|_2 \leq \sigma_{k+1}(A)\sqrt[4]{1 + 2\tau^4\left(\frac{\sigma_{\ell+1}(A)}{\sigma_{k+1}(A)}\right)^4}. \tag{1.3}$$

The quantities g_1, g_2, τ, and $\hat{\tau}$ are defined the same as in SRQR above.

Because of the fourth power on the ratio $\sigma_{\ell+1}(A)/\sigma_{k+1}(A)$, FFQR produces a remarkably accurate approximation to TSVD when the singular values of A decay rapidly and ℓ is chosen to be sufficiently large. It is due to this property that we have chosen to use FFQR as the basis for our algorithm.

Randomized methods

Much of the recent work done on low-rank matrix approximations combines traditional methods with randomization to obtain faster performance while maintaining good approximation quality.

Early randomized algorithms typically built low-rank matrix approximations by randomly sampling columns and/or rows from the original matrix. One of the simplest examples of such algorithms is the LinearTimeSVD algorithm of Drineas, Mahoney, and Kannan [16], shown in Algorithm 1. For a matrix X, we let $X^{(i)}$ and $X_{(i)}$ denote the ith column and ith row of X, respectively.

Algorithm 1 LinearTimeSVD

Input: $A \in \mathbb{R}^{m\times n}$, integers c and k $(1 \leq k \leq c \leq n)$, probability distribution $\{p_i\}_{i=1}^n$
Output: orthogonal $H_k \in \mathbb{R}^{m\times k}$, numbers $\sigma_t(C)$, $1 \leq t \leq k$

1: Select c columns (with indices i_t, $1 \leq t \leq c$) with replacement from A where $A^{(i)}$ is picked with probability p_i.
2: Form the matrix $C \in \mathbb{R}^{m\times c}$, where $C^{(t)} = A^{(i_t)}/\sqrt{cp_{i_t}}$.
3: Form C^TC and compute its SVD $\sum_{t=1}^c \sigma_t^2(C)y^t(y^t)^T$.
4: Compute $h^t = Cy^t/\sigma_t(C)$ for $1 \leq t \leq k$.
5: Return H_k, where $H_k^{(t)} = h^t$, and $\sigma_t(C)$, $1 \leq t \leq k$.

Observe that the h^t are just the left singular vectors of C. They are computed implicitly because performing SVD on C^TC is cheaper than on C. For a suitable probability distribution $\{p_i\}_{i=1}^n$, we can think of C as a set of representative columns of A that approximate the column space of A well. Hence, the h^t are good approximations to the left singular vectors

of A and the $\sigma_t(C)$ good approximations to A's singular values. Writing the projection of A onto H_k as

$$H_k H_k^T A = H_k \Sigma_k (A^T H_k \Sigma_k^{-1})^T,$$

where $\Sigma_k = \text{diag}(\sigma_1(C), \ldots \sigma_k(C))$, as in [34] exhibits an approximate rank-k TSVD of A. For the distribution

$$p_i = \frac{\|A^{(i)}\|_F^2}{\|A\|_F^2},$$

Drineas et al. [16] prove that

$$\|A - H_k H_k A\|_\xi^2 \leq \|A - A_k\|_\xi^2 + \varepsilon \|A\|_\xi^2 \tag{1.4}$$

with high probability if $c \geq O(1/\varepsilon^2)$ for $\xi = 2$ or $c \geq O(k/\varepsilon^2)$ for $\xi = F$. But, note that if $\|A\|_\xi \gg \|A - A_k\|_\xi$, then this bound is unsatisfactory.

In [17], Drineas, Mahoney, and Muthukrishnan improve on the additive-error bound 1.4 by using a more complicated probability distribution $\{p_i\}_{i=1}^n$ and sampling more columns. More specifically, they consider the distribution

$$p_i = \frac{\|(V_k)_{(i)}\|^2}{k}, \quad 1 \leq i \leq n, \tag{1.5}$$

where the columns of V_k are the top k right singular vectors of A. The quantities $\|(V_k)_{(i)}\|^2$ are the leverage scores for A and are related to the notion of *coherence* [7]. Intuitively, columns with high leverage contain the most information about the column space of A and should be included in C to obtain a good approximation. Thus, columns with high leverage should be selected with higher probability. They show that if p_i is defined by 1.5 and $c = O(k^2 \log(1/\delta)/\varepsilon^2)$, then the best approximation error over $O(\log(1/\delta))$ trials satisfies

$$\|A - CC^\dagger A\|_F \leq (1 + \varepsilon)\|A - A_k\|_F$$

with probability at least $1 - \delta$. One issue with this method is that computing the probabilities 1.5 is expensive as it requires computing a partial SVD of A. Some work has been done to compute good approximations to these probabilities quickly. See for instance [18]. Deshpande and Vempala further improve on these algorithms by using a different sampling strategy and sampling only $r := O(k\varepsilon^{-1} + k \log k)$ columns from A to construct a rank-k approximation $\tilde{A}_k$ satisfying the relative error bound

$$\|A - \tilde{A}_k\|_F^2 \leq (1 + \varepsilon)\|A - A_k\|_F^2.$$

Furthermore, $\tilde{A}_k$ can be computed in $O(\texttt{nnz}(A)r + (m + n)r^2)$ time.

The best randomized algorithms typically use random projections instead of randomly sampling columns and/or rows of the original matrix. As Sarlós [39] argues, random projections often result in higher quality approximations in some applications; moreover, the

projections generally can be constructed independently of the data while sampling-based algorithms usually need a data-dependent sampling distribution to obtain good results. The key property needed for a good random projection is that it should approximately preserve the geometry of the data when projecting it to a lower-dimensional space. Such projections are referred to as "low-distortion embeddings" or "sketches."

Johnson and Lindenstrauss [31][2] proved the following lemma that formalizes the idea of a low-distortion embedding:

Lemma 1. *Let $0 < \varepsilon < 1$, and $x_1, \ldots, x_n \in \mathbb{R}^m$ n points in $\mathbb{R}^m$. If $k = O(\varepsilon^{-2} \log n)$ and Φ is $\sqrt{m/k}$ times a $k \times m$ random orthogonal matrix, then with constant probability,*

$$(1-\varepsilon)\|x_i - x_j\|_2 \leq \|\Phi x_i - \Phi x_j\|_2 \leq (1+\varepsilon)\|x_i - x_j\|_2, \quad 1 \leq i, j \leq n.$$

Thus, we can embed n points into $\mathbb{R}^k$ and approximately preserve their pairwise distances provided that k is sufficiently large. However, construction of Φ would require a QR factorization, for instance, which costs $O(mk^2)$ time to compute, and $O(mk)$ time to apply to a vector.

To make low-distortion embeddings viable for practical use, we need to find embeddings that can be generated and applied quickly. Indyk and Motwani [30][39] made the important discovery that one could get the same low-distortion guarantee by taking Φ to be $1/\sqrt{k}$ times a $k \times m$ matrix whose entries are i.i.d. standard normal. This Φ can be generated much more cheaply than a random orthogonal matrix, and moreover has the attractive property that it can be parallelized. Achlioptas [1] proposed an even simpler projection matrix that has the same guarantee. It is a suitable multiple of a matrix whose entries are drawn from a simple distribution over $\{-1, 0, 1\}$.

In [2], Ailon and Chazelle proposed a so-called fast Johnson-Lindenstrauss transform. It is constructed as $\Phi = PHD$. Here, D is an $m \times m$ diagonal matrix where d_{ii} takes on the values ± 1 each with probability $1/2$. H is an $m \times m$ Walsh-Hadamard matrix scaled by $m^{-1/2}$ so that it is orthogonal. In Ailon and Chazelle's original paper, P was a $k \times m$ sampling matrix where p_{ij} takes on the value 0 with probability $1-q$ and otherwise is drawn from a $N(0, q^{-1})$ distribution, where

$$q = \min\left(\Theta\left(\frac{\log^2 n}{m}\right), 1\right)$$

and $k = O(\varepsilon^{-2} \log n)$. In more recent work [4][5], P is $\sqrt{m/k}$ times a matrix whose rows are drawn randomly uniformly (with or without replacement) from the $m \times m$ identity matrix, and so Φ is referred to as a subsampled randomized Hadamard transform (SRHT). P can be thought of as projecting a vector in $\mathbb{R}^m$ by randomly selecting k of its coordinates and then rescaling. The sparsity of P and D, and the fact that the Walsh-Hadamard transform (or more generally any FFT-like transform) can be computed in $O(m \log m)$ time make this faster much than the usual JLT, which requires dense matrix multiplication.

One of the fastest embeddings was proposed by Clarkson and Woodruff in [10]. The basic building block for these embeddings, which they refer to as *generalized sparse embeddings*,

are random matrices called *sparse embedding matrices*. Given t and n ($t \leq n$), a $t \times n$ sparse embedding matrix is constructed as the product ΦD. Each column of the $t \times n$ matrix Φ has exactly one non-zero entry equal to 1, which is selected uniformly at random, and D is a diagonal matrix each of whose diagonal entries is 1 or -1 with equal probability. Then a generalized sparse embedding matrix S has the form

$$S = \begin{pmatrix} B^{(1)} & & & \\ & B^{(2)} & & \\ & & \ddots & \\ & & & B^{(q)} \end{pmatrix} P,$$

where each $B^{(i)}$ consists of a number of sparse embedding matrices that are stacked on top of each other, and P is a certain permutation matrix. The construction involves some more details about the sizes of each $B^{(i)}$ and their number, and how P is constructed, but we will not discuss that here. The main observation is that applying S is very fast due its extreme sparsity. They prove that for a given $\delta > 0$, if we construct a $t \times m$ generalized sparse embedding matrix S, where $t = O(r\varepsilon^{-4}\log(r/\varepsilon\delta)(r + \log(1/\varepsilon\delta)))$, then $(1-\varepsilon)\|y\|_2 \leq \|Sy\| \leq (1+\varepsilon)\|y\|_2$ for all y in the column space of A with probability at least $1-\delta$. Moreover, SA can be computed in $O(\texttt{nnz}(A)\varepsilon^{-1}\log(r/\delta))$ time. They apply this embedding to regression, leverage score estimation, and low-rank matrix approximation, the latter of which we discuss briefly below.

Next, we discuss projection-based randomized low-rank approximation algorithms. Sarlós [39] shows that if $A \in \mathbb{R}^{m\times n}$, $0 < \varepsilon \leq 1$, $\mathcal{S} \in \mathbb{R}^{r\times n}$ is a JLT with i.i.d. Bernoulli entries (that take on values ± 1), and $r = \Theta(k/\varepsilon + k\log k)$ then

$$\|A - \pi_{A\mathcal{S}^T,k}(A)\|_F \leq (1+\varepsilon)\|A - A_k\|_F$$

with probability at least $1/2$, where $\pi_{A\mathcal{S}^T,k}$ is the rank-k truncated SVD of the projection of A onto the range of $A\mathcal{S}^T$. The singular vectors of $\pi_{A\mathcal{S}^T,k}(A)$ can be computed in $O(\texttt{nnz}(A)r + (m+n)r^2)$ time. This algorithm requires only two passes over the data, unlike the sampling-based algorithm by Deshpande and Vempala [15] which requires $\Theta(k\log k)$.

We can break down the computation of the low-rank approximation $\pi_{A\mathcal{S}^T,k}(A)$ into several steps. We will see that later randomized SVD algorithms typically follow a similar pattern. We can think of $\mathcal{S}^T$ as spanning an r-dimensional random space in $\mathbb{R}^n$ so that $A\mathcal{S}^T$ roughly approximates the subspace spanned by the top r left singular vectors of A. Next, we project A onto the subspace spanned by $A\mathcal{S}^T$. The easiest way to do this is to find an orthogonal basis for this subspace by computing the QR factorization $A\mathcal{S} = QR$ and then forming the projection QQ^TA. Finally, we compute the rank-k truncated SVD of QQ^TA by computing the rank-k TSVD $\tilde{U}_k\tilde{\Sigma}_k\tilde{V}_k^T$ of Q^TA and then setting $U_k := Q\tilde{U}_k$, $\Sigma_k := \tilde{\Sigma}_k$, and $V_k := \tilde{V}_k$, giving us the low-rank approximation $\pi_{A\mathcal{S}^T,k}(A) = U_k\Sigma_kV_k^T$.

Rokhlin, Szlam, and Tygert [38] improve on Sarlós's algorithm by including a few power iterations to make the singular values of A decay more quickly and hence increase the accuracy of the final approximation. In their paper, they consider A to be a short, fat matrix. We would like to think of A as a tall, skinny matrix, so we apply their algorithm to our A^T.

Suppose $A \in \mathbb{R}^{m\times n}$, $m \geq n$ and l and k are integers such that $k < l \leq n - k$. Generate $\Omega \in \mathbb{R}^{n\times l}$ and form the $m \times l$ matrix $Y = A(A^TA)^i\Omega$. Compute the rank-k truncated SVD of Y, and let $Q \in \mathbb{R}^{m\times k}$ be the matrix containing the top k left singular vectors of Y. Form $Z = Q^TA \in \mathbb{R}^{k\times n}$ and compute its SVD $Z = \tilde{U}\tilde{\Sigma}\tilde{V}^T$. Finally, set $U := Q\tilde{U}$, $\Sigma := \tilde{\Sigma}$, and $V := \tilde{V}$. Rokhlin et al. show that the approximate SVD resulting from this algorithm satisfies

$$\|A - U\Sigma V^T\|_2 \leq Cn^{1/(4i+2)}\sigma_{k+1}(A)$$

with high probability, where $C = C(k, l)$ is a constant that does not depend on A. While $n^{1/(4i+2)} \to 1$ as $i \to \infty$, there is still the constant C left that is somewhat unsatisfactory.

We can summarize these two algorithms with the following prototype for SVD via randomized subspace iteration:

Algorithm 2 SVD via randomized subspace iteration [25]

Input: a matrix $A \in \mathbb{R}^{m\times n}$, target rank k, integer $l \geq k$, small exponent i
Output: approximate rank-k truncated SVD $U_k\Sigma_kV_k^T$

1: Generate a random matrix $\Omega \in \mathbb{R}^{n\times l}$ (need not be Gaussian)
2: Form $Y = (AA^T)^iA\Omega$
3: Compute orthonormal basis Q for the range of Y
4: Form $B = Q^TA$
5: Compute rank-k truncated SVD $\hat{U}_k\hat{\Sigma}_k\hat{V}_k^T$ of B
6: Return $U_k := Q\hat{U}_k$, $\Sigma_k := \hat{\Sigma}_k$, $V_k = \hat{V}_k$

Halko, Martinsson, and Tropp [27] perform a more refined analysis for this randomized subspace iteration algorithm for the SVD. One can combine the results of Theorem 9.2 and Corollary 10.9 in [27] to obtain the following bound:

Theorem 1. *Let $k \geq 2$ be the target rank, $p \geq 4$ an oversampling parameter such that $k + p \leq \min\{m, n\}$, and $q \geq 0$ an integer. Draw an $n \times (k + p)$ standard Gaussian matrix. Set $Z := (AA^T)^qA\Omega$ and P_Z the projection operator onto Z. Then with failure probability at most $6e^{-p}$,*

$$\begin{aligned}\|(I - P_Z)A\|_2 &\leq \left[\left(1 + 17\sqrt{1 + \frac{k}{p}}\right)\sigma_{k+1}^{2q+1}(A) + \frac{8\sqrt{k+p}}{p+1}\left(\sum_{j>k}\sigma_j^{4q+2}(A)\right)^{1/2}\right]^{1/(2q+1)} \\ &\leq \left(1 + 17\sqrt{1 + \frac{k}{p}} + \frac{8\sqrt{k+p}}{p+1}\right)^{1/(2q+1)}\sigma_{k+1}(A).\end{aligned}$$

They also directly prove bounds on the approximation error *in expectation*, differing only in the value of the constant. However, the moral of the story is the same: as the number of power iterations $q \to \infty$, the approximation error approaches its optimal value of $\sigma_{k+1}(A)$ as

we would expect. The oversampling parameter p is quite small, so $P_Z A$ has rank $k+p \approx k$. As mentioned in [27], using this analysis to further get a bound on $\|A - (P_Z A)_k\|_2$, where $(P_Z A)_k$ is the best rank-k approximation to $P_Z A$ would add an extra $\sigma_{k+1}(A)$ term to the right hand side.

Gu [25] presents a novel error analysis of this algorithm that shows that matrices with rapidly decaying singular values can be approximated very accurately with randomized subspace iteration. The setup is similar to the one above. Let $A \in \mathbb{R}^{m\times n}$, $m \geq n$, $0 < k \leq l < n$, q a nonnegative integer, and $\Omega \in \mathbb{R}^{n\times l}$ a standard Gaussian matrix. Form $Y = (AA^T)^q A\Omega$, compute the QR factorization of $Y = QR$. (However, this way of computing an orthonormal basis for the range of Y could be numerically unstable for large enough q. In a proper implementation, we should compute Q in a more stable way, but since we are mainly interested in the analysis, we will not worry about that here.) Under this setting, Gu proves the following bounds:

Theorem 2. *Let $0 \leq p \leq l-k$, B_k the rank-k truncated SVD of $B := Q^T A$ (so QB_k is a rank-k approximation to A), and $0 < \Delta \ll 1$. Define*

$$\mathcal{C}_\Delta = \frac{e\sqrt{l}}{p+1}\left(\frac{2}{\Delta}\right)^{1/(p+1)}\left(\sqrt{n-l-p} + \sqrt{l} + \sqrt{2\log\frac{2}{\Delta}}\right).$$

Then for $j = 1, \ldots, k$,

$$\sigma_j(QB_k) \geq \frac{\sigma_j(A)}{\sqrt{1 + \mathcal{C}_\Delta^2\left(\frac{\sigma_{l-p+1}(A)}{\sigma_j(A)}\right)^{4q+2}}}$$

and

$$\|(I - QQ^T)A\|_2 \leq \|A - QB_k\|_2 \leq \sqrt{\sigma_{k+1}^2(A) + k\mathcal{C}_\Delta^2\sigma_{l-p+1}^2(A)\left(\frac{\sigma_{l-p+1}(A)}{\sigma_k(A)}\right)^{4q}}.$$

Thus, we see that if the spectrum of A decays rapidly, then the ratio σ_{l-p+1}/σ_k will be small, giving a very accurate approximation. And as in the bounds above, more power iterations lead to quicker convergence of the singular values and truncation error to their optimal values.

Clarkson and Woodruff [10] propose an interesting low-rank approximation algorithm that is based on regression rather than subspace iteration. Given $A \in \mathbb{R}^{n\times n}$, the solution to the regression problem

$$\min_{\text{rank}(X)=k} \|AX - A\|_F$$

is just $X = A_k^\dagger A_k$ since $AA_k^\dagger A_k = A_k$. We will give the basic idea behind the algorithm. In [10], the authors consider an approximation to this problem, namely

$$\min_{\text{rank}(X)=k} \|(AR^T)X - A\|_F, \tag{1.6}$$

where R is a certain embedding matrix. They show that the value of (1.6) is nearly optimal, bounded above $(1+\varepsilon)\|A-A_k\|_F$. They then consider the equivalent problem $\min_{\text{rank}(X)=k}\|UX-A\|_F$, where U is an orthonormal column basis for AR^T, and find an approximate solution (again within a factor of $1+\varepsilon$ of optimal) by solving

$$\min_{\text{rank}(X)=k}\|SUX-SA\|_F, \tag{1.7}$$

where S is another certain embedding matrix. Letting $\tilde{X}$ denote the solution to (1.7), we have

$$\|U\tilde{X}-A\|_F \leq (1+\varepsilon)\min_{\text{rank}(X)=k}\|UX-A\|_F \leq (1+\varepsilon)^2\|A-A_k\|_F.$$

So, all we need to do is find a formula for $\tilde{X}$. Referring back to (1.7), we can compute $\tilde{X}$ essentially by finding the best rank-k approximation of the projection of SA onto SU. Compute the SVD of $SU=\tilde{U}\tilde{\Sigma}\tilde{V}^T$, compute the projection $\tilde{U}^TSA$ of SA onto SU, find its best rank-k approximation $(\tilde{U}^TSA)_k$, and then pull this back to the domain of SU to get $\tilde{X}=\tilde{V}\tilde{\Sigma}^\dagger(\tilde{U}^TSA)_k$. Our final approximation of A is therefore $U\tilde{X}$, from which an approximate truncated SVD LDW^T can be computed. Moreover, they show that LDW^T can be computed quickly, in $O(\texttt{nnz}(A))+\tilde{O}(nk^2\varepsilon^{-4}+k^3\varepsilon^{-5})$, which is faster than previous methods.

Lastly, we would like to briefly mention an iterative randomized SVD algorithm. In [37], the authors present a randomized algorithm based on the blocked Lanczos algorithm to compute an approximate TSVD. This algorithm uses a random starting matrix Π with standard normal entries to build the Krylov subspace of A. For a matrix A, rank k, and tolerance ε, the algorithm produces a matrix Z whose columns approximate the top k left singular vectors of A and such that $\|A-ZZ^TA\|_\xi \leq (1+\varepsilon)\|A-A_k\|_\xi$ for $\xi=2,F$. They also provide an extra guarantee that

$$\left|u_i^TAA^Tu_i - z_i^TAA^Tz_i\right| \leq \varepsilon\sigma_{k+1}^2, \quad 1\leq i\leq k.$$

In other words, the columns of Z are good approximations to the left singular vectors of A. This algorithm is especially suited to sparse matrices, which can be multiplied quickly.

1.3 The fixed-precision problem

Low-rank approximation methods can generally be divided into two types: fixed-rank methods and fixed-precision methods. Fixed-rank methods assume the user provides a target rank k and will then produce a matrix B of rank k so that $\|A-B\|$ is small. Most research has focused on the fixed-rank problem, and indeed, all of the low-rank approximation algorithms mentioned above solve this problem. However, in practice, the user may not know ahead of time what the target rank should be. Instead, they may know how accurate of

an approximation they need, and this is where fixed-precision methods are necessary. For a given matrix A and approximation error $\varepsilon > 0$, fixed-precision methods output a low-rank matrix B so that $\|A - B\| < \varepsilon$. PCA can be recast as a fixed-precision problem. In the PCA example from Section 1.1 above, we want to find a B so that $\|A - B\|_F^2 < \varepsilon \|A\|_F^2$. In the same section, the solution of rank-deficient least squares problems via TSVD can also be recast as a fixed-precision problem: find a low-rank B so that $\|A - B\|_2 < \text{tol}$.

Fixed-precision methods

Most recent research has focused on the fixed-rank problem, however there is some work that addresses the fixed-precision problem. Typically, fixed-precision algorithms use a version of a fixed-rank algorithm that builds the approximation incrementally, stopping when the approximation error is sufficiently small. The crux of the matter is finding a cheap and accurate way to estimate this approximation error after each iteration. And as with recent fixed-rank methods, recent fixed-precision methods typically employ randomization to improve run time.

Martinsson and Voronin [36] proposed a fixed-precision algorithm which, for a given matrix A and error ε, outputs two matrices Q and B so that Q is column-orthogonal, $B = Q^T A$, and $\|A - QB\| < \varepsilon$. This algorithm is prototypical for fixed-precision algorithms. The full algorithm is presented in Algorithm 3.

Algorithm 3 The randQB_b algorithm for the fixed-precision problem [36]

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

1: **for** $i = 1, 2, 3, \ldots$ **do**
2: $\quad \Omega_i = \text{randn}(n, b)$
3: $\quad Q_i = \text{orth}(A\Omega_i)$
4: $\quad Q_i = \text{orth}(Q_i - \sum_{j=1}^{i-1} Q_j Q_j^T Q_i)$
5: $\quad B_i = Q_i^T A$
6: $\quad A = A - Q_i B_i$
7: $\quad$ **if** $\|A\| < \varepsilon$ **then stop**

At iteration i, we have constructed $[Q_1, \ldots, Q_{i-1}]$ and $[B_1; \ldots; B_{i-1}]$, and A holds the residual $A_0 - QB$, where A_0 denotes the original input matrix. To extend Q, the idea is to approximate the part of A_0 that has not yet been captured by evaluating A on a random Gaussian test matrix Ω_i. We then find an orthonormal basis Q_i for the part of $A\Omega_i$ that is orthogonal to the current basis Q, update A, and stop if the approximation error (the norm of the residual) is small enough. Martinsson and Voronin suggest using the Frobenius norm because it is cheap to compute and say that using norms that are more expensive to compute may require modifications to the algorithm.

As noted by Yu et al. [47], the computation of the approximation error $\|A\|$ in line 3.7 can be expensive, especially for very large matrices. They note that for algorithms using the

Frobenius norm, the following theorem (or some variant thereof) may be used to cheaply update the approximation error at each iteration:

Theorem 2 ([47]). *Let $Q_{(i)} := [Q_1, Q_2, \ldots, Q_i]$ be a column orthogonal matrix, $B_j := Q_j^T A$, and $B_{(i)} := Q_{(i)}^T A$, then*

$$\|A - Q_{(i)}Q_{(i)}^T A\|_F^2 = \|A\|_F^2 - \sum_{j=1}^{i} \|B_j\|_F^2.$$

Thus, after computing new basis vectors (the columns of Q_i), the approximation error $\|A - Q_{(i)}B_{(i)}\|_F^2$ can be computed simply by subtracting $\|B_i\|_F^2$ from the previous error $\|A - Q_{(i-1)}B_{(i-1)}\|_F^2$. This is cheaper as B_i is typically much smaller than A. Yu et al.'s randQB_EI algorithm is similar to Algorithm 3, but uses the error update formula in Theorem 2 and optionally can include a few power iterations to improve accuracy. We will compare our algorithm to randQB_EI and use Yu et al.'s Matlab implementation.

Hallman [28] proposes an algorithm that produces a low-rank approximation of the form UBV^T, where U and V are column-orthogonal and B is block bidiagonal, using a blocked version of the Golub-Kahan bidiagonalization procedure [24]. At the kth iteration, we have column-orthogonal matrices $U_{(k-1)} = [U_1, \ldots, U_{k-1}]$ and $V_{(k)} = [V_1, \ldots, V_k]$ and a block bidiagonal matrix

$$B_{k-1} = \begin{pmatrix} R_1 & L_2 & & \\ & R_2 & \ddots & \\ & & \ddots & L_{k-1} \\ & & & R_{k-1} \quad L_k \end{pmatrix}.$$

The algorithm then computes column-orthogonal matrices U_k and V_{k+1} to extend $U_{(k-1)}$ and $V_{(k)}$, respectively, and also new blocks R_k and L_{k+1}. Hallman notes that Theorem 2 implies that

$$\begin{aligned} \|A - U_{(k)}B_k V_{(k+1)}^T\|_F^2 &= \|A\|_F^2 - \|B_k\|_F^2 \\ &= \|A\|_F^2 - \|B_{k-1}\|_F^2 - \|R_k\|_F^2 - \|L_{k+1}\|_F^2 \\ &= \|A - U_{(k-1)}B_{k-1}V_{(k)}^T\|_F^2 - \|R_k\|_F^2 - \|L_{k+1}\|_F^2. \end{aligned}$$

So as before, the approximation error can be updated after each iteration simply by subtracting $\|R_k\|_F^2$ and $\|L_{k+1}\|_F^2$.

Zhang and Mascagni [48] propose a fixed-precision algorithm based on the LU factorization, randomization, and power iteration that also uses this type of Frobenius norm update formula. The goal is to construct a column-orthogonal matrix V so that $\|A - AVV^T\|_F$ is less than a user-prescribed accuracy level. At each iteration, we have $V_{(i-1)} = [V_1, \ldots, V_{i-1}]$ and extend it with new basis vectors V_i. Then using Theorem 2 and transposition, the new approximation error $\|A - AV_{(i)}(V_{(i)})^T\|_F^2 = \|A - AV_{(i-1)}(V_{(i-1)})^T\|_F^2 - \|AV_i\|_F^2$.

Most fixed-precision algorithms work with the Frobenius norm presumably because it is quite cheap to compute compared to the 2-norm. However, Algorithm 4 by Halko et al. [27] is an example of a fixed-precision algorithm based on the 2-norm. They use the following result from [44] to estimate the approximation error:

Theorem 3 ([44]). *Let B be a real matrix, r a positive integer, and $\alpha > 1$ a real number. Draw an independent family $\{\omega^{(i)} : i = 1, 2, \ldots, r\}$ of standard Gaussian vectors. Then*

$$\|B\|_2 \leq \alpha\sqrt{\frac{2}{\pi}} \max_{1\leq i\leq r} \|B\omega^{(i)}\|_2$$

except with probability α^{-r}.

Algorithm 4 Adaptive randomized range finder [27]

Input: an $m \times n$ matrix A; desired accuracy tolerance ε; integer r
Output: column orthogonal Q such that $\|A - QQ^TA\|_2 \leq \varepsilon$ with probability at least $1 - \min\{m, n\}10^{-r}$.
Draw standard Gaussian vectors $\omega^{(1)}, \ldots, \omega^{(r)}$ of length n.
$y^{(i)} = A\omega^{(i)}$ for $1 \leq i \leq r$
$j = 0$.
$Q^{(0)} = [\,]$
while $\max\{\|y^{(j+1)}\|_2, \ldots, \|y^{(j+r)}\|_2\} > \varepsilon/(10\sqrt{2/\pi})$ **do**
    $j = j + 1$
    $y^{(j)} = (I - Q^{(j-1)}(Q^{(j-1)})^T)y^{(j)}$
    $q^{(j)} = y^{(j)}/\|y^{(j)}\|_2$
    $Q^{(j)} = [Q^{(j-1)}, q^{(j)}]$
    Draw a standard Gaussian vector $\omega^{(j+r)}$ of length n.
    $y^{(j+r)} = (I - Q^{(j)}(Q^{(j)})^T)A\omega^{(j+r)}$
    $y^{(i)} = y^{(i)} - q^{(j)}(q^{(j)})^Ty^{(i)}$ for $j + 1 \leq i \leq j + r - 1$
$Q = Q^{(j)}$

The approximation to A is constructed essentially by drawing a sequence of independent standard Gaussian vectors $\omega^{(1)}, \omega^{(2)}, \ldots$ and applying a version of Gram-Schimdt to the sequence $A\omega^{(1)}, A\omega^{(2)}, \ldots$. At step j, we have constructed orthonormal vectors $q^{(1)}, \ldots, q^{(j-1)}$ from $A\omega^{(1)}, \ldots, A\omega^{(j-1)}$ and orthogonalized the next r vectors $A\omega^{(j)}, \ldots, A\omega^{(j+r-1)}$ against them to get $y^{(j)}, \ldots, y^{(j+r-1)}$, i.e.

$$y^{(i)} = (I - Q^{(j-1)}(Q^{(j-1)})^T)A\omega^{(i)}, \quad j \leq i \leq j + r - 1.$$

Since $y^{(j)}$ is already orthogonal to all of the $q^{(i)}$, we can construct $q^{(j)}$ simply by normalizing $y^{(j)}$ to have length 1. Next, orthogonalize all the $y^{(i)}$ against $q^{(j)}$, and construct $y^{(j+r)}$ by orthogonalizing $A\omega^{(j+r)}$ against all the $q^{(i)}$. Observe that at this point,

$$y^{(i)} = (I - Q^{(j)}(Q^{(j)})^T)A\omega^{(i)} = (A - Q^{(j)}(Q^{(j)})^TA)\omega^{(i)}, \quad j + 1 \leq i \leq j + r$$

Thus, as Halko et al. say, we get our norm estimation for free! Using Theorem 3, we conclude that if $\max_{j+1 \leq i \leq j+r} \|y^{(i)}\|_2 \leq \varepsilon/(10\sqrt{2/\pi})$, then $\|A - Q^{(j)}(Q^{(j)})^T A\|_2 \leq \varepsilon$ with high probability.

Chapter 2

A Fast, Approximate TSVD Algorithm

In this section, we introduce our algorithm. We will work with matrices with more rows than columns. For matrices with more columns than rows, apply the algorithm to the transpose.

2.1 Our work

Given the importance of TSVD, we propose a fixed-precision algorithm that outputs an approximate TSVD, using the 2-norm error rather than the Frobenius norm error. For matrices with rapidly decaying singular values, the algorithm runs in $O(mn\ell)$ time, where $\ell \ll \min\{m, n\}$, which is faster than the $O(mn^2)$ needed to compute the full SVD and truncate.

As noted in [37], having a near-optimal approximation error in the Frobenius norm does not necessarily guarantee a high-quality approximation, especially in data science and machine learning. They suggest that the 2-norm is intuitively stronger and often yields better-quality approximations. This is one justification for using the 2-norm rather than the Frobenius norm in our algorithm. They also provide an example showing that even a near-optimal approximation error in the 2-norm does not by itself guarantee a high-quality approximation. This motivates them to consider an algorithm which provides stronger guarantees beyond a near-optimal approximation error.

In this work, we propose an algorithm that, for a matrix A, relative error δ, and singular value tolerance tol, produces an approximate TSVD $\tilde{A}$ satisfying the following properties:

1. The rank $\tilde{k}$ of $\tilde{A}$ does not exceed the true rank of A, which is defined to be $\#\{i : \sigma_i(A) \geq \text{tol}\}$.
2. $\sigma_j(\tilde{A}) \geq (1-\delta)\sigma_j(A)$ for $1 \leq j \leq \tilde{k}$,
3. $\|A - \tilde{A}\|_2 \leq (1+\delta)\sigma_{\tilde{k}+1}(A)$, and

4. $\|A - \tilde{A}\|_2 \leq \frac{1+\delta}{1-\delta}\varepsilon \approx (1 + 2\delta)\text{tol}$.

The third property ensures a near-optimal approximation error while the second provides a stronger guarantee on the quality of the approximation by ensuring that the singular values are computed with relative error not exceeding δ.

Due to FFQR's superior approximation quality, our algorithm is based on FFQR. The main problem then is to determine suitable values of ℓ and $\tilde{k}$ to achieve the properties listed above. Recall the partial ℓ-step QLP factorization used in FFQR:

$$A = Q \begin{pmatrix} R_{11}^{(\ell)} & R_{12}^{(\ell)} \\ 0 & R_{22}^{(\ell)} \end{pmatrix} \Pi^T = Q \begin{pmatrix} L_{11}^{(\ell)} & 0 \\ L_{21}^{(\ell)} & L_{22}^{(\ell)} \end{pmatrix} P^T \Pi^T, \quad R_{11}^{(\ell)}, L_{11}^{(\ell)} \in \mathbb{R}^{\ell \times \ell},$$

where, for clarity, we have added the superscript (ℓ) to indicate how many steps have been performed. We will also write

$$R^{(\ell)} = \begin{pmatrix} R_{11}^{(\ell)} & R_{12}^{(\ell)} \\ 0 & R_{22}^{(\ell)} \end{pmatrix}, \quad L^{(\ell)} = \begin{pmatrix} L_{11}^{(\ell)} & 0 \\ L_{21}^{(\ell)} & L_{22}^{(\ell)} \end{pmatrix}$$

for the R and L factors, respectively, after ℓ steps of QLP. Here is a high-level overview of the algorithm. Since we do not know ℓ ahead of time, we will compute the QLP factorization above a few steps at a time. After performing a few steps, we will check if the approximation we would get at that point is good enough. If it is, then we have found ℓ, and $\tilde{k}$ will be chosen so that

$$\sigma_{\tilde{k}}(L^{(\ell)}(:, 1:\ell)) \geq \text{tol} \geq \sigma_{\tilde{k}+1}(L^{(\ell)}(:, 1:\ell)).$$

The following sections will be organized as follows. We will first describe how to perform QLP incrementally, derive a criterion for finding ℓ, and finally explain how to verify the criterion.

2.2 Blocked QLP

To compute QLP incrementally, select a block size b and perform b steps of RQRCP to get

$$A\Pi_1 = Q_1 \begin{pmatrix} R_{11}^{[b]} & R_{12}^{[b]} \\ 0 & R_{22}^{[b]} \end{pmatrix},$$

where $R_{11}^{[b]}$ is $b \times b$ upper triangular. (Note the superscripts are in brackets so as not to be confused with the notation in the previous section.) The first b rows of R are essentially done since subsequent steps of RQRCP will only permute the columns of $R_{12}^{[b]}$. Perform QR on them (to keep the notation simple, we write this as an LQ factorization):

$$\begin{pmatrix} R_{11}^{[b]} & R_{12}^{[b]} \end{pmatrix} = \begin{pmatrix} L_{11} & 0 \end{pmatrix} P_1^T,$$

where L_{11} is $b \times b$ lower triangular, and P_1 is orthogonal. We have just computed the first b rows of L and know the first b diagonal entries. For the next block, continue RQRCP for another b steps. The permutation matrix Π_2 in this block will affect only columns $b+1$ through n, leaving the first b columns untouched. Thus, Π_2 can be written in block form as $\Pi_2 = \begin{pmatrix} I_b & 0 \\ 0 & \tilde{\Pi}_2 \end{pmatrix}$, where I_b is the $b \times b$ identity matrix and $\tilde{\Pi}_2$ is an $(n-b) \times (n-b)$ permutation matrix. We now have

$$A\Pi_1\Pi_2 = Q_2 Q_1 \left(\begin{array}{c|cc} R_{11}^{[b]} & R_{12}^{[b]}\tilde{\Pi}_2 \\ \hline 0 & R_{11}^{[2b]} & R_{12}^{[2b]} \\ 0 & 0 & R_{22}^{[2b]} \end{array}\right),$$

where $R_{11}^{[2b]}$ is $b \times b$ upper triangular. Since the first b rows have changed, we must account for this in the previous LQ:

$$\begin{pmatrix} R_{11}^{[b]} & R_{12}^{[b]}\tilde{\Pi}_2 \end{pmatrix} = \begin{pmatrix} R_{11}^{[b]} & R_{12}^{[b]} \end{pmatrix} \Pi_2 = \begin{pmatrix} L_{11} & 0 \end{pmatrix} P_1^T \Pi_2.$$

Now apply the matrix $\Pi_2^T P_1$ to the newly completed rows

$$\begin{pmatrix} 0 & R_{11}^{[2b]} & R_{12}^{[2b]} \end{pmatrix}$$

and perform LQ on the last $n-b$ columns to get

$$\begin{pmatrix} 0 & R_{11}^{[2b]} & R_{12}^{[2b]} \end{pmatrix} \Pi_2^T P_1 = \begin{pmatrix} L_{21} & L_{22} & 0 \end{pmatrix} P_2^T,$$

where L_{22} is $b \times b$ lower triangular.

The orthogonal matrix P_2 affects only the last $n-b$ columns and can therefore be written in block form as $P_2 = \begin{pmatrix} I_b & 0 \\ 0 & \tilde{P}_2 \end{pmatrix}$. Hence, $\begin{pmatrix} L_{11} & 0 \end{pmatrix} = \begin{pmatrix} L_{11} & 0 \end{pmatrix} P_2^T$ and

$$\left(\begin{array}{c|cc} R_{11}^{[b]} & R_{12}^{[b]}\tilde{\Pi}_2 \\ \hline 0 & R_{11}^{[2b]} & R_{12}^{[2b]} \end{array}\right) = \begin{pmatrix} L_{11} & 0 & 0 \\ L_{21} & L_{22} & 0 \end{pmatrix} P_2^T P_1^T \Pi_2,$$

showing that we have computed the first $2b$ rows of L. We can continue this procedure, computing b rows of L at a time. Once we decide to stop, we finish the remaining rows of L by applying the orthogonal matrices from all previous LQ factorizations to the last rows of R. For example, if we wanted to stop after 2 blocks, apply $\Pi_2^T P_1 P_2$ to $\begin{pmatrix} 0 & 0 & R_{22}^{[2b]} \end{pmatrix}$ to get

$$\begin{pmatrix} 0 & 0 & R_{22}^{[2b]} \end{pmatrix} \Pi_2^T P_1 P_2 = \begin{pmatrix} L_{31} & L_{32} & L_{33} \end{pmatrix}$$

and the partial QLP decomposition

$$A\Pi_1\Pi_2 = Q_2Q_1 \left(\begin{array}{c|cc} R_{11}^{[b]} & R_{12}^{[b]}\tilde{\Pi}_2 \\ \hline 0 & R_{11}^{[2b]} & R_{12}^{[2b]} \\ 0 & 0 & R_{22}^{[2b]} \end{array}\right)$$
$$= Q_2Q_1 \begin{pmatrix} L_{11} & 0 & 0 \\ L_{21} & L_{22} & 0 \\ L_{31} & L_{32} & L_{33} \end{pmatrix} P_2^T P_1^T \Pi_2.$$

Afterwards, spectrum-revealing swaps can be performed if desired. For each swap and upper-trapezoidal restoration, some nonzero entries will appear above the diagonal in L. These are easily eliminated with Givens rotations.

2.3 Determining ℓ

We will now discuss a criterion to determine ℓ. In principle, we could use the bounds 1.2 and 1.3 derived in [23] to do this. We might be able to use the facts that $\sigma_{k+1}(A)$ is the largest singular value of A less than tol and the diagonal entries of L approximate the singular values of A well to first estimate $\sigma_{k+1}(A)$ and then choose ℓ so that the ratio $\sigma_{\ell+1}(A)/\sigma_{k+1}(A)$ is sufficiently small. However, the dimension-dependent bounds for τ and $\hat{\tau}$ are too large for practical use, so we will use a different bound to determine ℓ.

In [23], the authors prove that $\sigma_j(A)^4 \leq \sigma_j(\tilde{\Sigma}_k)^4 + 2\|R_{22}^{(\ell)}\|_2^4, 1 \leq j \leq k$. Rearranging this inequality gives

$$\sigma_j(\tilde{\Sigma}_k) \geq \sigma_j(A)\sqrt[4]{1 - 2\frac{\|R_{22}^{(\ell)}\|_2^4}{\sigma_j(A)^4}}, \quad 1 \leq j \leq k.$$

They also prove the following bound on the truncation error:

$$\|A - \tilde{U}_k\tilde{\Sigma}_k\tilde{V}_k^T\|_2 \leq \sigma_{k+1}(A)\sqrt[4]{1 + 2\frac{\|R_{22}^{(\ell)}\|_2^4}{\sigma_{k+1}(A)^4}}. \tag{2.1}$$

These bounds hold even without spectrum-revealing swaps. So, if $\|R_{22}^{(\ell)}\|_2/\sigma_{k+1}(A)$ is small, then the leading k singular values of A will be revealed up to a certain number of digits and $\tilde{U}_k\tilde{\Sigma}_k\tilde{V}_k^T$ will be a nearly optimal rank-k approximation. In practice, the above two bounds are sufficient because $\|R_{22}^{(\ell)}\|_2 = O(\sigma_\ell(A))$ already, without extra swaps. The earlier bounds still have theoretical value in that they show the algorithm works well when A has rapidly decaying singular values.

We have the first-order approximations

$$\sqrt[4]{1 - 2\frac{\|R_{22}^{(\ell)}\|_2^4}{\sigma_j(A)^4}} \approx 1 - \frac{1}{2}\frac{\|R_{22}^{(\ell)}\|_2^4}{\sigma_j(A)^4} \quad \text{and} \quad \sqrt[4]{1 + 2\frac{\|R_{22}^{(\ell)}\|_2^4}{\sigma_{k+1}(A)^4}} \approx 1 + \frac{1}{2}\frac{\|R_{22}^{(\ell)}\|_2^4}{\sigma_{k+1}(A)^4}.$$

Introduce a relative error parameter δ, and say we have

$$\frac{1}{2}\frac{\|R_{22}^{(\ell)}\|_2^4}{\sigma_{k+1}(A)^4} \leq \delta.$$

Then up to first order,

$$\sigma_j(\tilde{\Sigma}_k) \geq \sigma_j(A)(1-\delta),\ 1 \leq j \leq k, \quad \text{and} \quad \|A - \tilde{U}_k\tilde{\Sigma}_k\tilde{V}_k^T\|_2 \leq \sigma_{k+1}(A)(1+\delta).$$

This means that $\approx -\log_{10}\delta$ digits of the top k singular values of A and optimal truncation error have been computed correctly. We can rewrite

$$\frac{1}{2}\frac{\|R_{22}^{(\ell)}\|_2^4}{\sigma_{k+1}(A)^4} \leq \delta \implies \|R_{22}^{(\ell)}\|_2 \leq \sigma_{k+1}(A)\sqrt[4]{2\delta}.$$

This is the tolerance-based criterion to determine ℓ.

In general, ℓ will depend on the singular value distribution of A. To get a sense of how big ℓ is compared to k for matrices with decaying singular values, we examine the case when the singular values decay geometrically.

Theorem 3. *Let A be an $m \times n$ matrix, and suppose there is a number c such that $0 < c < 1$ and $\sigma_{i+1}(A) \leq c\sigma_i(A)$ for all i. Let $\varepsilon > 0$ and $g > 1$ be user-defined parameters. Suppose we perform FFQR (with SRQR swaps), choosing ℓ so that*

$$\ell \geq k + \frac{\frac{1}{4}\ln(2\delta) - \ln\left(gn\sqrt{\frac{1+\varepsilon}{1-\varepsilon}}\right)}{\ln c}. \tag{2.2}$$

Then $\|R_{22}^{(\ell)}\|_2 \leq \sigma_{k+1}(A)\sqrt[4]{2\delta}$. Hence, the smallest ℓ that satisfies the tolerance-based criterion is at most

$$\left\lceil k + \frac{\frac{1}{4}\ln(2\delta) - \ln\left(gn\sqrt{\frac{1+\varepsilon}{1-\varepsilon}}\right)}{\ln c} \right\rceil.$$

Proof. Rearranging (2.2), and noting that $\ln c < 0$ and $\sigma_{\ell+1}(A) \leq c^{\ell-k}\sigma_{k+1}(A)$, we get

$$c^{\ell-k} \leq \frac{\sqrt[4]{2\delta}}{gn\sqrt{\frac{1+\varepsilon}{1-\varepsilon}}} \implies gn\sqrt{\frac{1+\varepsilon}{1-\varepsilon}}\frac{\sigma_{\ell+1}(A)}{\sigma_{k+1}(A)} \leq \sqrt[4]{2\delta}. \tag{2.3}$$

By the analysis in [45], the SRQR swaps ensure that

$$\begin{aligned}\|R_{22}^{(\ell)}\|_2 &\leq g\sqrt{\frac{1+\varepsilon}{1-\varepsilon}}\sqrt{(\ell+1)(n-\ell)}\sigma_{\ell+1}(A) \\ &\leq gn\sqrt{\frac{1+\varepsilon}{1-\varepsilon}}\sigma_{\ell+1}(A).\end{aligned} \tag{2.4}$$

Combining (2.3) and (2.4) gives $\|R_{22}^{(\ell)}\|_2 \leq \sigma_{k+1}(A)\sqrt[4]{2\delta}$. □

Note that ε and g are usually chosen to be small numbers, e.g. $\varepsilon = 0.5$ and $g = 2.0$ are typical choices [23]. So, one may treat these two parameters as small constants.

Next, we must figure out a way to cheaply and accurately estimate $\|R_{22}^{(\ell)}\|_2$ and $\sigma_{k+1}(A)$.

Estimating $\|R_{22}^{(\ell)}\|_2$

To estimate $\|R_{22}^{(\ell)}\|_2$, we will consider the more general problem of estimating the 2-norm of a general matrix X. Since $\|X\|_2$ is just the first singular value $\sigma_1(X)$ of X, we can use Stewart's observation about the QLP decomposition to estimate $\|X\|_2$. Recall that the QLP decomposition is computed as follows: perform QRCP on X to get $X = QR\Pi^T$ and then perform QRCP again on R^T to get $R^T = PL^T\Pi_1^T$. According to Stewart's observation, $|l_{11}| \approx \sigma_1(X) = \|X\|_2$. But, $|l_{11}|$ is just the maximum column norm of R^T, or the maximum row norm of R. Hence, we can estimate $\|X\|_2$ by performing QRCP on X and then finding the largest row norm of the R factor.

But Stewart also points out in [40] that we can get a good estimate of $\|X\|_2$ by finding the largest row norm among the first few rows of R rather than all of them. This is more efficient because we do not need to compute R completely, but just a few rows. Letting q denote the number rows, our estimate of $\|X\|_2$ is $\max_{1\le\iota\le q}\|R(\iota,:)\|_2$. For use in the algorithm, we actually want an upper bound on $\|X\|_2$, so we will assume that there is some $\gamma \ge 1$ so that $\|X\|_2 \le \gamma \max_{1\le\iota\le q}\|R(\iota,:)\|_2$. (Note that we always have $\|X\|_2 \ge \max_{1\le\iota\le q}\|R(\iota,:)\|_2$.)

Now going back to $\|R_{22}^{(\ell)}\|_2$, we see that we need to perform q steps of QRCP on it to estimate the norm. But observe that if we continue our QRCP factorization of A and perform the next q steps, then we have automatically performed q steps of QRCP on $R_{22}^{(\ell)}$. So, the easiest way to estimate the norm of $R_{22}^{(\ell)}$ is to wait until more blocks of QRCP have been completed for A. However, we *are* able to estimate $\|R_{22}^{(i)}\|_2$ for $0 \le i \le \ell - q$ because after ℓ steps of QRCP, the first ℓ rows are finished (up to permutation of entries). To sum up, after c steps of QRCP on A, we have

$$\|R_{22}^{(i)}\|_2 \le \gamma \max_{i+1\le\iota\le i+q}\|R^{(c)}(\iota,:)\|_2, \quad 0 \le i \le c - q.$$

Note that subsequent steps of QRCP will not affect the norms of the first c rows, so we need to compute the right hand side only once for each i during the entire algorithm.

Estimating $\sigma_{k+1}(A)$

Now, we consider how to estimate $\sigma_{k+1}(A)$. In light of Stewart's observation, we have $l_{jj} \approx \sigma_j(A)$. However, since we do not pivot when computing L, the diagonal entries of L will not appear in descending order. Thus, we can get an even better estimate for $\sigma_j(A)$ by using the jth largest diagonal entry of L. Let $l^{(j)}$ denote the j-th largest diagonal entry of L in magnitude so that $\left|l^{(1)}\right| \ge \left|l^{(2)}\right| \ge \cdots \ge \left|l^{(n)}\right|$. As with our estimate for $\|R_{22}^{(\ell)}\|_2$, we will assume that there are constants α and β such that $\alpha\left|l^{(j)}\right| \le \sigma_j(A) \le \beta\left|l^{(j)}\right|$, $1 \le j \le n$.

The values of α and β will be estimated empirically below. A simple way to interpret these inequalities is that for each diagonal entry l_{jj}, there is a singular value of A in the interval $\alpha\,|l_{jj}| \le x \le \beta\,|l_{jj}|$. Consider $\{l_{jj} : \beta\,|l_{jj}| \le \text{tol}\}$. For each l_{jj} in this set, there is a singular value $\sigma_i(A)$ such that

$$\alpha\,|l_{jj}| \le \sigma_i(A) \le \beta\,|l_{jj}| \le \text{tol}.$$

Since $\sigma_{k+1}(A)$ is the largest singular value of A less than or equal to tol, we must have $\sigma_i(A) \le \sigma_{k+1}(A)$, which implies $\alpha\,|l_{jj}| \le \sigma_{k+1}(A)$. This yields a lower bound on $\sigma_{k+1}(A)$, namely $\max\{\alpha\,|l_{jj}| : \beta\,|l_{jj}| \le \text{tol}\}$.

However, this estimate requires knowing all the l_{jj}! Unfortunately, we will only know l_{jj}, $1 \le j \le c$, after c steps of QLP, so we have only the suboptimal estimate

$$s_{k+1} := \max\{\alpha\,|l_{jj}| : \beta\,|l_{jj}| \le \text{tol and } j \le c\}.$$

Putting these estimates together gives us the final stopping criterion. Suppose we have completed c steps of QLP. After performing the next b steps, we first update s_{k+1} with the newly computed l_{jj}'s and then check whether

$$\max_{i+1\le\iota\le i+q} \|R^{(c+b)}(\iota,:)\|_2 \le \frac{1}{\gamma} s_{k+1} \sqrt[4]{2\delta}$$

for some $0 \le i \le (c+b)-q$. The smallest i for which this inequality holds is ℓ since

$$\|R_{22}^{(i)}\|_2 \le \gamma \max_{i+1\le\iota\le i+q} \|R^{(c+b)}(\iota,:)\|_2 \le \gamma \frac{1}{\gamma} s_{k+1} \sqrt[4]{2\delta} \le \sigma_{k+1}(A) \sqrt[4]{2\delta}.$$

The full algorithm is presented below in Algorithm 5.

2.4 A simpler version of Algorithm 5

In this section, we discuss a simpler version of Algorithm 5 that in some sense may be considered more reliable because it uses a more conservative estimate for $\|R_{22}^{(i)}\|_2$.

Recall that given a matrix X, we estimated $\|X\|_2$ by performing q steps of QRCP and then obtain the upper bound

$$\|X\|_2 \le \gamma \max_{1\le\iota\le q} \|R(\iota,:)\|_2. \tag{2.5}$$

As we mentioned in that same discussion, a better but more expensive estimate is to perform QRCP completely and then use the upper bound

$$\|X\|_2 \le \gamma \max_{1\le\iota\le n} \|R(\iota,:)\|_2. \tag{2.6}$$

For some matrices, using this bound may actually result in faster performance. Consider running Algorithm 5 on $A \in \mathbb{R}^{m\times n}$, $m \ge n$. We first perform a partial QLP factorization,

Algorithm 5 A fast, approximate algorithm for TSVD

Input: $A \in \mathbb{R}^{m\times n}$ $(m \geq n)$, tol, δ, α, β, γ, q, b
Output: $\tilde{k}$, $\tilde{U}_{\tilde{k}}$, $\tilde{\Sigma}_{\tilde{k}}$, $\tilde{V}_{\tilde{k}}$

1: $c = 0$, $s_{k+1} = 0$
2: **while** $c < n$ **do**
3: Do steps $c+1$ to $c+b$ of RQRCP on A; update Q, R, and Π.
4: Compute rows $c+1$ to $c+b$ of L; update P.
5: **for** $j = c+1 : c+b$ **do**
6: **if** $\beta\,|l_{jj}| \leq \text{tol}$ **and** $\alpha\,|l_{jj}| \geq s_{k+1}$ **then**
7: $s_{k+1} = \alpha\,|l_{jj}|$
8: **for** $i = 0 : c+b-q$ **do**
9: **if** $\max_{i+1\leq\iota\leq i+q} \|R(\iota,:)\|_2 \leq \frac{1}{\gamma} s_{k+1}\sqrt[4]{2\delta}$ **then**
10: $\ell = i$
11: $c = c+b$
12: exit **while** loop
13: $c = c+b$
14: Compute rows $c+b+1$ to m of L.
15: Compute TSVD $\hat{U}_{\tilde{k}}\hat{\Sigma}_{\tilde{k}}\hat{V}_{\tilde{k}}^T$ of $L(:, 1:\ell)$, where $\sigma_{\tilde{k}}(L(:,1:\ell)) \geq \text{tol} \geq \sigma_{\tilde{k}+1}(L(:,1:\ell))$.
16: Return $\tilde{U}_{\tilde{k}} = Q\hat{U}_{\tilde{k}}$, $\tilde{\Sigma}_{\tilde{k}} = \hat{\Sigma}_{\tilde{k}}$, $\tilde{V}_{\tilde{k}} = \Pi P_1 \hat{V}_{\tilde{k}}$

which gives us R and L factors that are $m \times n$. Then we compute the SVD of $L(:, 1:\ell)$, which is $m \times \ell$. But suppose we use the bound 2.6 instead. Performing QRCP completely results in a factorization of the form

$$A = QR = \begin{pmatrix} Q_1 & Q_2 \end{pmatrix} \begin{pmatrix} R_{11}^{(n)} \\ 0 \end{pmatrix} = Q_1 R_{11}^{(n)}, \quad Q_1 \in \mathbb{R}^{m\times n}, R_{11}^{(n)} \in \mathbb{R}^{n\times n}. \tag{2.7}$$

We now compute L by performing QR on $(R_{11}^{(n)})^T$, so L is $n \times n$ and SVD is now performed on $L(:, 1:\ell)$, which is smaller, with size $n \times \ell$. In our experiments, we show an example where the time saved from performing SVD on a smaller $L(:, 1:\ell)$ outweighs the time needed to compute R completely. Another advantage of the bound 2.6 is that the user does not have to worry about the parameter q anymore.

Referring to Equation 2.7, after we compute R completely, we have the estimate

$$\|R_{22}^{(i)}\|_2 \leq \gamma \max_{i+1\leq\iota\leq n} \|R_{11}^{(n)}(\iota,:)\|_2, \quad 0 \leq i \leq n-1.$$

We can compute all of these estimates efficiently with Algorithm 6.

The simpler version of the algorithm is presented below in Algorithm 7. Note that if we check line 7.10 for all i when we have completed only a small number of steps $(c+b \ll n)$, then we might end up with a large ℓ because s_{k+1} is likely to be much smaller than $\sigma_{k+1}(A)$ at that point. Thus, we have restricted checking line 7.10 to $i \leq c+b-1$. We will see in

Algorithm 6 Compute $\max_{i+1\leq\iota\leq n}\|R_{11}^{(n)}(\iota,:)\|_2$ for all i

Input: $R_{11}^{(n)}\in\mathbb{R}^{n\times n}$ from QRCP factorization of A
Output: array nrm_est, where $\text{nrm\_est}[i]=\max_{i+1\leq\iota\leq n}\|R_{11}^{(n)}(\iota,:)\|_2$

Compute row norms of $R_{11}^{(n)}$ and store in row_nrm, where $\text{row\_nrm}[i]=\|R_{11}^{(n)}(i,:)\|_2$
$\text{nrm\_est}[n-1]=\text{row\_nrm}[n]$
for $i=n-2:-1:0$ **do**
    $\text{nrm\_est}[i]=\max\{\text{row\_nrm}[i+1],\text{nrm\_est}[i+1]\}$

Algorithm 7 Simplified version of Algorithm 5

Input: $A\in\mathbb{R}^{m\times n}$ $(m\geq n)$, tol, δ, α, β, γ, b
Output: $\tilde{k}$, $\tilde{U}_{\tilde{k}}$, $\tilde{\Sigma}_{\tilde{k}}$, $\tilde{V}_{\tilde{k}}$

Perform RQRCP on A to get $A=Q_1R_{11}^{(n)}\Pi^T$
Compute trailing norm estimates using Algorithm 6.
$c=0$, $s_{k+1}=0$
while $c<n$ **do**
    Compute rows $c+1$ to $c+b$ of L; update P.
    for $j=c+1:c+b$ **do**
        if $\beta\,|l_{jj}|\leq\text{tol}$ **and** $\alpha\,|l_{jj}|\geq s_{k+1}$ **then**
            $s_{k+1}=\alpha\,|l_{jj}|$
    for $i=0:c+b-1$ **do**
        if $\text{nrm\_est}[i]\leq\frac{1}{\gamma}s_{k+1}\sqrt[4]{2\delta}$ **then**
            $\ell=i$
            $c=c+b$
            exit **while** loop
    $c=c+b$
Compute rows $c+b+1$ to m of L.
Compute TSVD $\hat{U}_{\tilde{k}}\hat{\Sigma}_{\tilde{k}}\hat{V}_{\tilde{k}}^T$ of $L(:,1:\ell)$, where $\sigma_{\tilde{k}}(L(:,1:\ell))\geq\text{tol}\geq\sigma_{\tilde{k}+1}(L(:,1:\ell))$.
Return $\tilde{U}_{\tilde{k}}=Q\hat{U}_{\tilde{k}}$, $\tilde{\Sigma}_{\tilde{k}}=\hat{\Sigma}_{\tilde{k}}$, $\tilde{V}_{\tilde{k}}=\Pi P_1\hat{V}_{\tilde{k}}$

Chapter 3 that generally Algorithm 7 finds a larger ℓ than Algorithm 5 because the former uses a more conservative estimate of $\|R_{22}^{(i)}\|_2$.

2.5 Proof of Properties 1-4

In this section, we prove the properties stated in Section 2.1. We denote the true rank as k, the rank detected by our algorithm as $\tilde{k}$, and the matrix output by our algorithm $\tilde{A}$. Recall k is defined by $\sigma_{k+1}(A)\leq\text{tol}\leq\sigma_k(A)$. Recall also that the detected rank $\tilde{k}$ is determined as follows. We run Blocked QLP until the trailing block of R is small enough and then take

ℓ such that $\|R_{22}^{(\ell)}\|_2 \leq \sigma_{k+1}(A)\sqrt[4]{2\delta}$. Afterwards, compute the SVD of $L_1 := L(:, 1:\ell)$ and define $\tilde{k}$ by $\sigma_{\tilde{k}+1}(L_1) \leq \text{tol} \leq \sigma_{\tilde{k}}(L_1)$.

First, note that $\tilde{k} \leq k$. To see this, first observe that $k = \#\{j : \sigma_j(A) > \text{tol}\}$. By the Cauchy Interlacing Theorem, $\sigma_j(L_1) \leq \sigma_j(A)$, $1 \leq j \leq \ell$. Thus we can only shift the singular values of A downward, which will not increase the size of the above set. This proves Property 1.

For Property 2, it follows from computations in [23] that $\sigma_j(A)^4 \leq \sigma_j^4(L_1) + 2\|R_{22}^{(\ell)}\|_2^4$, or

$$\sigma_j(L_1) \geq \sigma_j(A)\sqrt[4]{1 - 2\frac{\|R_{22}^{(\ell)}\|_2^4}{\sigma_j(A)^4}} \approx \sigma_j(A)\left(1 - \frac{1}{2}\frac{\|R_{22}^{(\ell)}\|_2^4}{\sigma_j(A)^4}\right)$$

for $1 \leq j \leq \ell$. Plugging in $\|R_{22}^{(\ell)}\|_2 \leq \sigma_{k+1}(A)\sqrt[4]{2\delta}$ gives $\sigma_j(L_1) \geq \sigma_j(A)(1-\delta)$ for $1 \leq j \leq k+1$ and in particular for $1 \leq j \leq \tilde{k}$. This is Property 2.

For the last two properties, we refer to Inequality (2.1). In the notation for this section, it reads:

$$\|A - \tilde{A}\|_2 \leq \sigma_{\tilde{k}+1}(A)\sqrt[4]{1 + 2\frac{\|R_{22}^{(\ell)}\|_2^4}{\sigma_{\tilde{k}+1}(A)^4}} \approx \sigma_{\tilde{k}+1}(A)\left(1 + \frac{1}{2}\frac{\|R_{22}^{(\ell)}\|_2^4}{\sigma_{\tilde{k}+1}(A)^4}\right).$$

Again, plugging in $\|R_{22}^{(\ell)}\|_2 \leq \sigma_{k+1}(A)\sqrt[4]{2\delta}$ and using the fact that $\tilde{k}+1 \leq k+1$ gives $\|A - \tilde{A}\|_2 \leq \sigma_{\tilde{k}+1}(A)(1+\delta)$, which is Property 3.

Finally, from the proof of Property 2 above, we have $\sigma_{\tilde{k}+1}(L_1) \geq \sigma_{\tilde{k}+1}(A)(1-\delta)$. Thus, $\sigma_{\tilde{k}+1}(A) \leq \frac{1}{1-\delta}\sigma_{\tilde{k}+1}(L_1)$ and $\|A - \tilde{A}\|_2 \leq \frac{1+\delta}{1-\delta}\sigma_{\tilde{k}+1}(L_1) \leq \frac{1+\delta}{1-\delta}\text{tol}$, which is Property 4.

Note that $\tilde{k} < k$ only when there are singular values slightly above the tolerance. The tolerance-based criterion ensures that up to first order $\sigma_j(L_1) \geq \sigma_j(A)(1-\delta)$. So only singular values satisfying $\sigma_j(A) \geq \text{tol} \geq \sigma_j(A)(1-\delta)$ can be perturbed below tol and decrease the rank.

2.6 Right singular space quality analysis

We include here a brief analysis of the quality of the right singular space produced by FFQR, which was not done in [23]. This is of interest in applications such as PCA. In [11], Kahan and Davis give several theorems bounding trigonometric functions of the angle between the eigenspace of a given Hermitian matrix A and the corresponding eigenspace of a perturbation $A + H$ of A by a Hermitian matrix H. We will introduce some notation used in the paper before stating the relevant theorem.

We are mainly interested in the case when A is a real, symmetric matrix. So, let A be such a matrix and E an orthogonal matrix whose columns are eigenvectors of A. We may partition $E = \begin{pmatrix} E_0 & E_1 \end{pmatrix}$ into its first k columns E_0 and remaining columns E_1. Then

$$A = \begin{pmatrix} E_0 & E_1 \end{pmatrix} \begin{pmatrix} A_0 & 0 \\ 0 & A_1 \end{pmatrix} \begin{pmatrix} E_0^T \\ E_1^T \end{pmatrix}.$$

Similarly, let $F = \begin{pmatrix} F_0 & F_1 \end{pmatrix}$ be an orthogonal matrix whose columns are eigenvectors of $A + H$, partitioned into its first k columns F_0 and remaining columns F_1. Then we may write $A + H$ in either the basis E or the basis F:

$$A + H = \begin{pmatrix} E_0 & E_1 \end{pmatrix} \begin{pmatrix} A_0 + H_0 & B^T \\ B & A_1 + H_1 \end{pmatrix} \begin{pmatrix} E_0^T \\ E_1^T \end{pmatrix} = \begin{pmatrix} F_0 & F_1 \end{pmatrix} \begin{pmatrix} \Lambda_0 & 0 \\ 0 & \Lambda_1 \end{pmatrix} \begin{pmatrix} F_0^T \\ F_1^T \end{pmatrix}.$$

Finally, let Θ_0 be the diagonal matrix containing the principal angles between the subspaces spanned by E_0 and F_0. Davis and Kahan express their bounds in terms of the residual $R := (A + H)E_0 - E_0A_0 = HE_0$. Their so-called $\sin\theta$ theorem is the one we will be most interested in:

Theorem 4 ([11])**.** *Assume there is an interval $[\beta, \alpha]$ and a $\delta > 0$ such that the spectrum of A_0 lies entirely in $[\beta, \alpha]$ while that of Λ_1 lies entirely outside of $(\beta-\delta, \alpha+\delta)$ (or such that the spectrum of Λ_1 lies entirely in $[\beta, \alpha]$ while that of A_0 lies entirely outside of $(\beta - \delta, \alpha + \delta)$). Then for every unitary-invariant norm $\|\cdot\|$, $\delta\|\sin\Theta_0\| \leq \|R\|$.*

To apply the theorem (with $\|\cdot\| = \|\cdot\|_2$), consider the matrix

$$L^TL = \begin{pmatrix} L_{11}^T & L_{21}^T \\ 0 & L_{22}^T \end{pmatrix} \begin{pmatrix} L_{11} & 0 \\ L_{21} & L_{22} \end{pmatrix} = \begin{pmatrix} L_{11}^TL_{11} + L_{21}L_{21}^T & L_{21}^TL_{22} \\ L_{22}^TL_{21} & L_{22}^TL_{22} \end{pmatrix}.$$

We set

$$A = L^TL, \quad H = -\begin{pmatrix} 0 & L_{21}^TL_{22} \\ L_{22}^TL_{21} & L_{22}^TL_{22} \end{pmatrix}, \quad A + H = \begin{pmatrix} L_{11}^TL_{11} + L_{21}^TL_{21} & 0 \\ 0 & 0 \end{pmatrix}.$$

The columns of E_0 are the top k right singular vectors of L, $A_0 = \operatorname{diag}(\lambda_1(A), \ldots, \lambda_k(A))$, and $\Lambda_1 = \operatorname{diag}(\lambda_{k+1}(A + H), \ldots, \lambda_\ell(A + H), 0, \ldots, 0)$. By the Cauchy Interlacing Theorem, $\lambda_{k+1}(A + H) \leq \lambda_{k+1}(A)$, so we can pick $\delta = \lambda_k(A) - \lambda_{k+1}(A)$. Next, we bound $\|R\|_2$:

$$\begin{aligned}
\|R\|_2 &= \|HE_0\|_2 \\
&\leq \|H\|_2\|E_0\|_2 \\
&= \|H\|_2 \\
&= \left\| \begin{pmatrix} 0 & L_{21}^TL_{22} \\ L_{22}^TL_{21} & L_{22}^TL_{22} \end{pmatrix} \right\|_2 \\
&\leq \left\| \begin{pmatrix} 0 & 0 \\ L_{22}^TL_{21} & 0 \end{pmatrix} \right\|_2 + \left\| \begin{pmatrix} 0 & L_{21}^TL_{22} \\ 0 & L_{22}^TL_{22} \end{pmatrix} \right\|_2 \\
&= \left\| L_{22}^TL_{21} \right\|_2 + \left\| \begin{pmatrix} L_{21}^TL_{22} \\ L_{22}^TL_{22} \end{pmatrix} \right\|_2 \\
&\leq \|L_{22}^T\|_2\|L_{21}\|_2 + \left\| \begin{pmatrix} L_{21}^T \\ L_{22}^T \end{pmatrix} \right\|_2 \|L_{22}\|_2 \\
&\leq 2\|R_{22}\|_2^2.
\end{aligned}$$

Finally, we get

$$\|\sin\Theta_0\|_2 \leq \frac{2\|R_{22}\|_2^2}{\lambda_k(A) - \lambda_{k+1}(A)} = \frac{2\|R_{22}\|_2^2}{\sigma_k^2(L) - \sigma_{k+1}^2(L)}.$$

Using spectrum-revealing swaps in FFQR ensures that $\|R_{22}\|_2 = O(\sigma_{\ell+1}(L))$. Even if there is a just small relative gap between $\sigma_k(L)$ and $\sigma_{k+1}(L)$, we will have $\sigma_k^2(L) - \sigma_{k+1}^2(L) = O(\sigma_k^2(L))$. Thus, $\|\sin\Theta_0\|_2 = O\left((\sigma_{\ell+1}(L)/\sigma_k(L))^2\right)$. So, if the singular values of L decay quickly and ℓ is sufficiently large, then $\|\Theta_0\|_2$ is roughly quadratic in the ratio $\sigma_{\ell+1}(L)/\sigma_k(L)$.

Chapter 3

Numerical Experiments

We use the following test matrices for our experiments:

1. A random 3000×3000 matrix with singular values decaying geometrically from 1 down to 10^{-12}. We generate a 3000×3000 matrix with entries from a standard normal distribution, compute its SVD $U\Sigma V^T$, and replace the diagonal of Σ with the desired singular value distribution.

2. A 2003×2003 matrix (bcsstk13) from the SuiteSparse matrix collection [12]. This matrix arises from a computational fluid dynamics problem. Its singular values decay from $\approx 10^{12}$ down to $\approx 10^2$.

3. A matrix arising from discretizing an integral equation of the first kind [21]. The matrix depends on the number of sample points used in the discretization and a parameter κ. We use 1000 sample points (resulting in a 1000×1000 matrix) and $\kappa = 0.1$.

4. A 19200×5322 matrix generated from a video from the UCF-Crime dataset [41]. The original video was a 240×320 RGB video consisting of 5322 frames. We resized the video by half to 120×160, converted it to grayscale, flattened each frame into a 19200×1 column vector, and then stacked these horizontally to form the final matrix.

5. A 5000×5000 kernel matrix generated from 5000 data points from the MNIST handwritten digits dataset [33]. We used the kernel function $k(x, x') = e^{-\gamma\|x-x'\|^2}$, where $\gamma = 1/(\text{median of pairwise distances between data points})^2$.

The singular values for these matrices are plotted in Fig. 3.1.

3.1 Estimating α, β, and γ

For each matrix A, we tested three singular value estimation schemes. See Table 3.1. For the first two columns ("Unpivoted"), we ran RQRCP to get $A\Pi = QR$ and then QR-factored

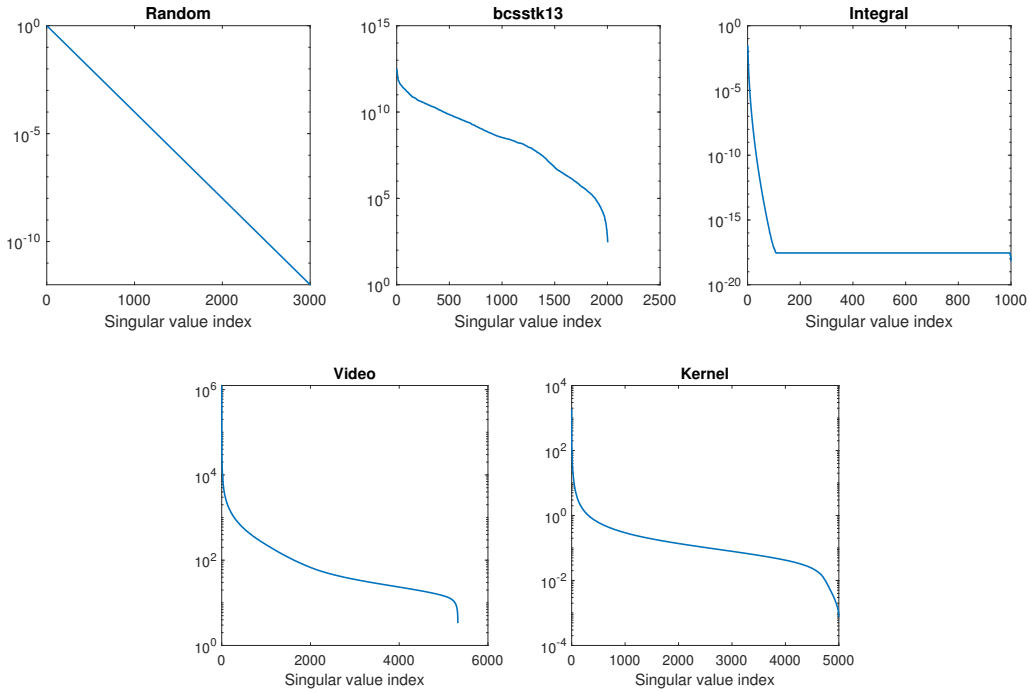

Figure 3.1: Singular value distributions of the test matrices

$R^T = PL$. We recorded the minimum (α) and maximum (β) values of $\sigma_i(A)/|l_{ii}|$. For the second two ("Sorted"), we recorded the minimum and maximum values of $\sigma_i(A)/\left|l^{(i)}\right|$, where $l^{(i)}$ is the ith largest diagonal entry of L in magnitude. For the last two columns ("Pivoted"), we QRCP-factored $R^T\Pi_1 = PL$ and then recorded the minimum and maximum values of $\sigma_i(A)/|l_{ii}|$.

We observed that the tracking behavior can break down when $\sigma_i(A)$ is smaller than machine precision and thus ignored ratios corresponding to such $\sigma_i(A)$ when computing the minimum and maximum values. Therefore, it is recommended that the tolerance tol be set at least a small factor above machine epsilon.

"Sorted" and "Pivoted" have similar α and β values, with the latter slightly better overall, while "Unpivoted" tends to be worse than the other two. Based on the middle two columns, it seems that $\alpha \approx 0.7$ and $\beta \approx 2$ are reasonable values.

We perform a similar experiment to determine the value of γ. Table 3.2 lists γ values for bound 2.5 for various values of q (first five columns) and bound 2.6 (last column, "all"). We

Table 3.1: α and β values for the test matrices under the three schemes.

	Unpivoted		Sorted		Pivoted	
	α	β	α	β	α	β
Random	0.710	1.39	0.742	1.33	0.745	1.32
bcsstk13	0.523	2.75	0.793	1.17	0.817	1.17
Integral	0.572	1.74	0.770	1.35	0.794	1.33
Video	0.321	2.85	0.838	1.85	0.840	1.58
Kernel	0.635	1.60	0.802	1.32	0.817	1.31

Table 3.2: γ values for the test matrices for bounds 2.5 and 2.6

	q					
	5	10	20	50	100	all
Random	1.69	1.64	1.61	1.61	1.61	1.61
bcsstk13	1.86	1.86	1.86	1.45	1.45	1.45
Integral	5.12	4.88	4.02	2.81	1.78	1.78
Video	10.04	7.20	3.41	2.47	2.47	2.47
Kernel	3.97	3.77	3.37	2.73	2.49	1.91

performed RQRCP on each matrix and then computed

$$\max_{0\le i\le n-1} \frac{\|R_{22}^{(i)}\|_2}{\max\limits_{i+1\le \iota\le i+q} \|R_{11}^{(n)}(\iota,:)\|_2},\ q=5,10,20,50,100, \quad \text{and} \quad \max_{0\le i\le n-1} \frac{\|R_{22}^{(i)}\|_2}{\max\limits_{i+1\le \iota\le n} \|R_{11}^{(n)}(\iota,:)\|_2}.$$

For Algorithm 5, $q = 50$ (about %1–5 of n for our matrices) and $\gamma = 3.0$ seem reasonable for all the matrices. And for Algorithm 7, looking at the last column, we take $\gamma = 3.0$ again.

3.2 Comparison to TSVD and randQB_EI

Here we compare the proposed algorithm to TSVD and randQB_EI. Tests were coded in Fortran and run on a laptop with a 2.00 GHz Intel i7-4510U CPU with 16.0 GB of RAM.

First, we compare the proposed algorithm to TSVD. To compute the latter, the LAPACK routine dgesdd is used to compute the full SVD, which is then truncated based on the tolerance tol. For the random, bcsstk13, and kernel matrices, tolerances corresponding to 99% explained variance are chosen. For the video matrix, we choose one corresponding to 99.9% explained variance because the first principal component already accounts for 99% of the variance. These are all Frobenius norm tolerances, and in general it is not so simple to find the corresponding 2-norm tolerance. But for matrices with geometrically decaying singular

values, 99% explained variance roughly corresponds to a tolerance of $0.1\|A\|_2$. The 2-norm of A can be estimated after the first block of FFQR. For experimental purposes, we computed the SVD of each matrix to determine the proper 2-norm tolerance. We treat the integral equation matrix slightly differently, going from a 2-norm tolerance and then determining the equivalent Frobenius norm tolerance. For the 2-norm tolerance, we use tol $\approx \varepsilon_{mach}$ since such matrices are usually rank-deficient and find that the equivalent Frobenius norm tolerance roughly corresponds to $\approx (1 - \varepsilon_{mach}) \times 100\%$ explained variance.

We set $\delta = 10^{-4}$ for all test matrices because machine learning algorithms typically only need a few digits of accuracy. But the larger singular values are computed with more accuracy because the accuracy of the jth singular value is controlled by $\|R_{22}\|_2/\sigma_j(A)$, which is typically much smaller than $\|R_{22}\|_2/\sigma_{k+1}(A)$. Finally, for all matrices, we set $\alpha = 0.7$, $\beta = 2.0$, $\gamma = 3.0$, the number of rows $q = 50$, and the block size $b = 64$.

The results for Algorithm 5 are listed in Table 3.3 and for Algorithm 7 in Table 3.4. Both algorithms detect the rank k correctly for each test matrix and are much faster than dgesdd. As mentioned earlier, Algorithm 7 is more conservative than Algorithm 5; we can see this clearly by comparing the values of ℓ. But despite this, Algorithm 7 is faster for the video matrix since it performs SVD on a smaller matrix.

The column "rel. trunc. err." contains the relative error in the optimal truncation error

$$\left| \frac{\|A - \tilde{A}_{\tilde{k}}\|_2}{\sigma_{\tilde{k}+1}(A)} - 1 \right|.$$

Using the notation in Algorithm 5 or 7, write the SVD of $L(:, 1:\ell)$ as

$$\begin{pmatrix} L_{11}^{(\ell)} \\ L_{21}^{(\ell)} \end{pmatrix} = \hat{U}\hat{\Sigma}\hat{V}^T = \hat{U}_{\tilde{k}}\hat{\Sigma}_{\tilde{k}}\hat{V}_{\tilde{k}}^T + \hat{U}_{\tilde{k}}^-\hat{\Sigma}_{\tilde{k}}^-(\hat{V}_{\tilde{k}}^-)^T,$$

so $\hat{U}_{\tilde{k}}\hat{\Sigma}_{\tilde{k}}\hat{V}_{\tilde{k}}$ corresponds to the top $\tilde{k}$ singular vectors/values and $\hat{U}_{\tilde{k}}^-\hat{\Sigma}_{\tilde{k}}^-\hat{V}_{\tilde{k}}^-$ to the remaining $n - \tilde{k}$. The truncation error $\|A - \tilde{A}_{\tilde{k}}\|_2$ in exact arithmetic is equal to

$$\begin{aligned} \left\| \left(\hat{U}_{\tilde{k}}^-\hat{\Sigma}_{\tilde{k}}^-(\hat{V}_{\tilde{k}}^-)^T \;\middle|\; \begin{matrix} 0 \\ L_{22}^{(\ell)} \end{matrix} \right) \right\|_2 &= \left\| \left(\hat{U}_{\tilde{k}}^-\hat{\Sigma}_{\tilde{k}}^-(\hat{V}_{\tilde{k}}^-)^T \;\middle|\; \begin{matrix} 0 \\ L_{22}^{(\ell)} \end{matrix} \right) \begin{pmatrix} \hat{V} & 0 \\ 0 & I \end{pmatrix} \right\|_2 \\ &= \left\| \left(\hat{U}_{\tilde{k}}^-\hat{\Sigma}_{\tilde{k}}^- \;\middle|\; \begin{matrix} 0 \\ L_{22}^{(\ell)} \end{matrix} \right) \right\|_2. \end{aligned}$$

We compute the truncation error numerically using this last equality rather than forming $\tilde{A}_{\tilde{k}}$ directly. Since our matrices are not well-conditioned, we may not be able to compute the relative truncation error accurately, but we record the results anyway.

Recall that randQB_EI returns matrices Q and $B := Q^T A$ so that $A \approx QB$. The authors of randQB_EI include a power parameter P in their implementation. We set $P = 1$ as in their paper and use a block size of 64. In Table 3.5, we list the tolerances used for each

Table 3.3: Comparison of Algorithm 5 to TSVD.

		dgesdd		Algorithm 5				
Matrix	tol	k	Time(s)	ℓ	$\tilde{k}$	Time (s)	rel. trunc. err.	Speed-up
Random	1.00×10^{-1}	250	14.65	864	250	3.06	2.22×10^{-16}	4.8×
bcsstk13	1.00×10^{11}	128	4.38	834	128	1.60	4.44×10^{-16}	2.7×
Integral	1.00×10^{-15}	74	0.39	101	74	0.09	2.05×10^{-4}	4.3×
Video	5.60×10^{3}	36	155.46	2762	36	92.05	4.44×10^{-16}	1.7×
Kernel	7.68×10^{1}	7	69.36	792	7	6.16	2.78×10^{-15}	11.3×

Table 3.4: Comparison of Algorithm 7 to TSVD.

		dgesdd		Algorithm 7				
Matrix	tol	k	Time(s)	ℓ	$\tilde{k}$	Time (s)	rel. trunc. err.	Speed-up
Random	1.00×10^{-1}	250	14.65	864	250	3.27	4.44×10^{-16}	4.5×
bcsstk13	1.00×10^{11}	128	4.38	834	128	1.62	4.44×10^{-16}	2.7×
Integral	1.00×10^{-15}	74	0.39	101	74	0.15	2.05×10^{-4}	2.6×
Video	5.60×10^{3}	36	155.46	3012	36	74.00	2.88×10^{-15}	2.1×
Kernel	7.68×10^{1}	7	69.36	1367	7	14.54	2.22×10^{-16}	4.8×

Table 3.5: Tolerances and detected ranks for randQB_EI

	E	rank(B)
Random	$\sqrt{0.01}\|A\|_F$	260
bcsstk13	$\sqrt{0.01}\|A\|_F$	136
Integral	$\sqrt{10^{-15}}\|A\|_F$	28
Video	$\sqrt{0.001}\|A\|_F$	40
Kernel	$\sqrt{0.01}\|A\|_F$	7

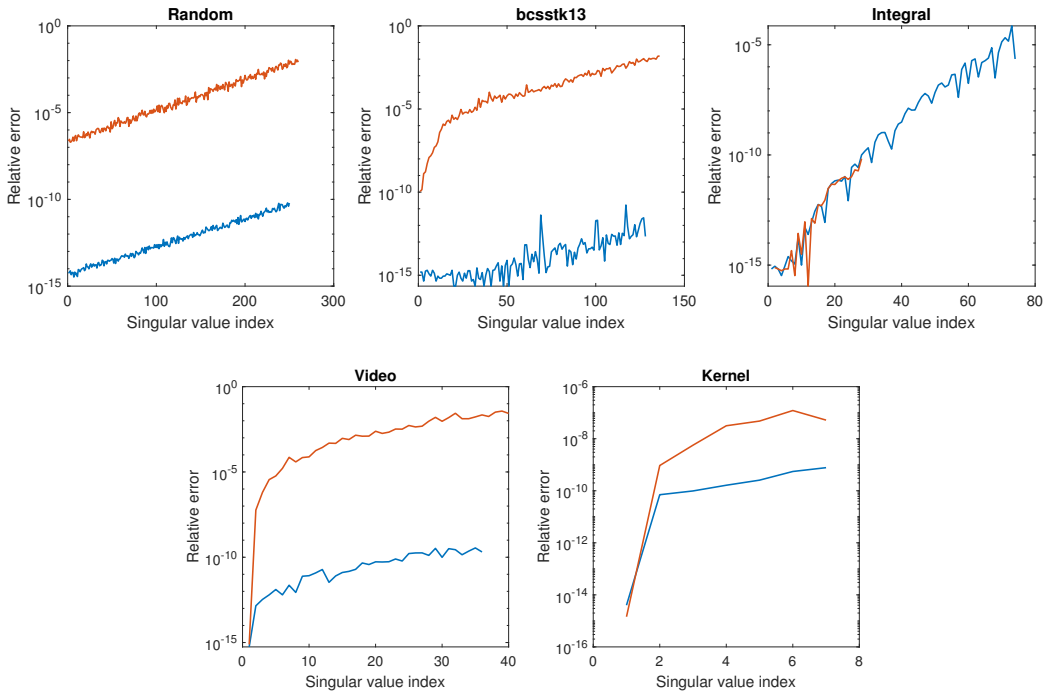

Figure 3.2: Relative singular value errors for the test matrices. Algorithm 5 is the blue line, randQB_EI is the orange one.

matrix as well as the detected rank. As we can see, the algorithm finds nearly the correct rank for all matrices, except for the integral equation matrix.

Fig. 3.2 plots the relative errors in the approximate singular values produced by Algorithm 5

$$\left| 1 - \frac{\sigma_j(\tilde{\Sigma}_{\tilde{k}})}{\sigma_j(A)} \right|, \ 1 \leq j \leq \tilde{k},$$

for each of the test matrices. These are bounded by δ. As expected, the larger singular values are computed more accurately. We also plot the relative errors in the singular values computed by randQB_EI, and we see that generally the singular values are not as accurate as for Algorithm 5.

Next, recall that the principal components for a data matrix are just its right singular vectors. Since this is of practical interest, the principal angles between the leading right singular spaces and their approximations produced by Algorithm 5 and randQB_EI were computed and plotted in Fig. 3.3. The angles are all quite small for Algorithm 5, so it finds high-quality principal components. Our algorithm performs better than randQB_EI,

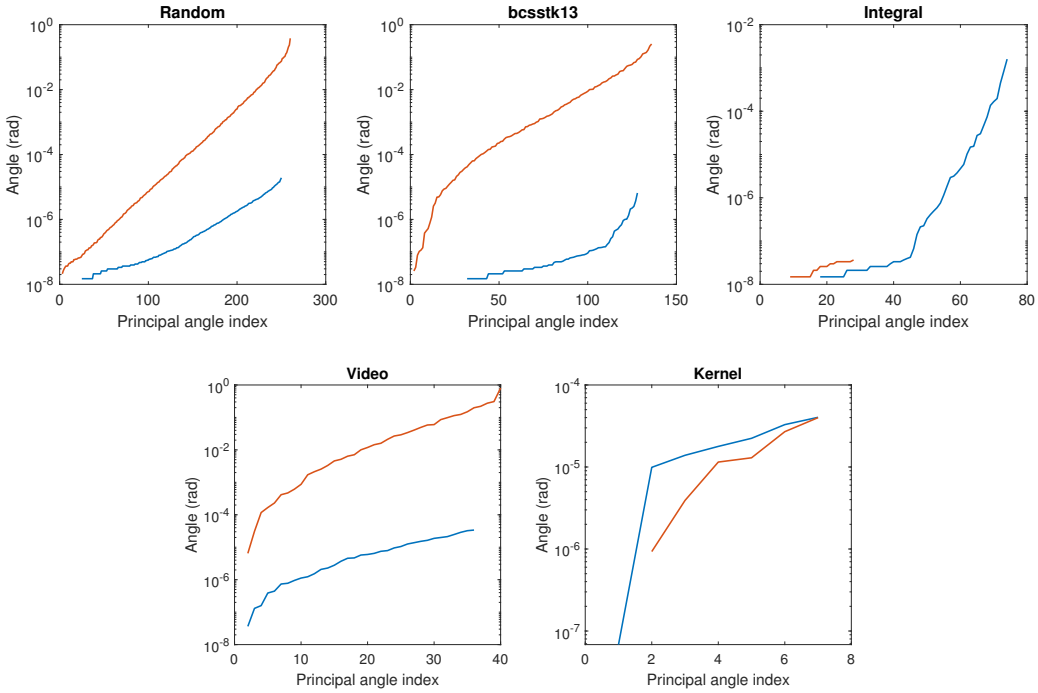

Figure 3.3: Principal angles between the right singular spaces and their approximations for the test matrices. Algorithm 5 is the blue line, randQB_EI is the orange one.

except on the kernel matrix. We found that setting $P = 0$ causes the accuracy of randQB_EI to drop below ours. The first few singular values of the kernel matrix are extremely large compared to the rest; thus, performing even just one power iteration effectively enhances the accuracy of randQB_EI on this matrix.

Bibliography

[1] Dimitris Achlioptas. "Database-friendly random projections: Johnson-Lindenstrauss with binary coins". In: *Journal of Computer and System Sciences* 66.4 (2003). Special Issue on PODS 2001, pp. 671–687. ISSN: 0022-0000. DOI: https://doi.org/10.1016/S0022-0000(03)00025-4. URL: https://www.sciencedirect.com/science/article/pii/S0022000003000254.

[2] Nir Ailon and Bernard Chazelle. "The Fast Johnson–Lindenstrauss Transform and Approximate Nearest Neighbors". In: *SIAM Journal on Computing* 39.1 (2009), pp. 302–322. URL: https://doi.org/10.1137/060673096.

[3] E. Anderson et al. *LAPACK Users' Guide*. Third. Philadelphia, PA: Society for Industrial and Applied Mathematics, 1999. ISBN: 0-89871-447-8 (paperback).

[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. URL: https://doi.org/10.1137/090767911.

[5] Christos Boutsidis and Alex Gittens. "Improved Matrix Algorithms via the Subsampled Randomized Hadamard Transform". In: *SIAM Journal on Matrix Analysis and Applications* 34.3 (2013), pp. 1301–1340. URL: https://doi.org/10.1137/120874540.

[6] Peter Businger and Gene H. Golub. "Linear least squares solutions by householder transformations". In: *Numerische Mathematik* 7.3 (1965), pp. 269–276. URL: https://doi.org/10.1007/BF01436084.

[7] Emmanuel J. Candès and Benjamin Recht. "Exact Matrix Completion via Convex Optimization". In: *Foundations of Computational Mathematics* 9.6 (2009), pp. 717–772. URL: https://doi.org/10.1007/s10208-009-9045-5.

[8] Tony F. Chan. "Rank revealing QR factorizations". In: *Linear Algebra and its Applications* 88-89 (1987), pp. 67–82. ISSN: 0024-3795. DOI: https://doi.org/10.1016/0024-3795(87)90103-0. URL: https://www.sciencedirect.com/science/article/pii/0024379587901030.

[9] Shivkumar Chandrasekaran and Ilse C. F. Ipsen. "On Rank-Revealing Factorisations". In: *SIAM Journal on Matrix Analysis and Applications* 15.2 (1994), pp. 592–622. URL: https://doi.org/10.1137/S0895479891223781.

[10] Kenneth L. Clarkson and David P. Woodruff. "Low-Rank Approximation and Regression in Input Sparsity Time". In: *J. ACM* 63.6 (Jan. 2017). ISSN: 0004-5411. URL: https://doi.org/10.1145/3019134.

[11] Chandler Davis and W. M. Kahan. "The Rotation of Eigenvectors by a Perturbation. III". In: *SIAM Journal on Numerical Analysis* 7.1 (1970), pp. 1–46. URL: https://doi.org/10.1137/0707001.

[12] Timothy A. Davis and Yifan Hu. "The University of Florida Sparse Matrix Collection". In: *ACM Transactions on Mathematical Software* 38.1 (2011), pp. 1–25.

[13] Scott Deerwester et al. "Indexing by latent semantic analysis". In: *Journal of the American Society for Information Science* 41.6 (1990), pp. 391–407. DOI: https://doi.org/10.1002/(SICI)1097-4571(199009)41:6<391::AID-ASI1>3.0.CO;2-9.

[14] J. Demmel. *Applied Numerical Linear Algebra*. Philadelphia, PA: Society for Industrial and Applied Mathematics, 1997. DOI: 10.1137/1.9781611971446.

[15] Amit Deshpande and Santosh Vempala. "Adaptive Sampling and Fast Low-Rank Matrix Approximation". In: *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*. Ed. by Josep Díaz et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 292–303.

[16] 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. URL: https://doi.org/10.1137/S0097539704442696.

[17] Petros Drineas, Michael W. Mahoney, and S. Muthukrishnan. "Subspace Sampling and Relative-Error Matrix Approximation: Column-Based Methods". In: *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*. Ed. by Josep Díaz et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 316–326.

[18] Petros Drineas et al. "Fast Approximation of Matrix Coherence and Statistical Leverage". In: 13.1 (Dec. 2012), pp. 3475–3506. ISSN: 1532-4435.

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

[20] C. Eckart and G. Young. "The Approximation of One Matrix by Another of Lower Rank". In: *Psychometrika* 1.3 (1936), pp. 211–218. URL: https://doi.org/10.1007/BF02288367.

[21] Lars Eldén. "The Numerical Solution of a Noncharacteristic Cauchy Problem for a Parabolic Equation". In: *Numerical Treatment of Inverse Problems in Differential and Integral Equations*. 1983, pp. 246–268.

[22] Karl Pearson F.R.S. "LIII. On lines and planes of closest fit to systems of points in space". In: *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science* 2.11 (1901), pp. 559–572. URL: https://doi.org/10.1080/14786440109462720.

[23] Yuehua Feng, Jianwei Xiao, and Ming Gu. "Low-Rank Matrix Approximations with Flip-Flop Spectrum-Revealing QR Factorization". In: *Electronic Transactions on Numerical Analysis* 51 (2019), pp. 469–494.

[24] G. Golub and W. Kahan. "Calculating the Singular Values and Pseudo-Inverse of a Matrix". In: *Journal of the Society for Industrial and Applied Mathematics Series B Numerical Analysis* 2.2 (1965), pp. 205–224. URL: https://doi.org/10.1137/0702016.

[25] M. Gu. "Subspace Iteration Randomization and Singular Value Problems". In: *SIAM Journal on Scientific Computing* 37.3 (2015), A1139–A1173. URL: https://doi.org/10.1137/130938700.

[26] 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. URL: https://doi.org/10.1137/0917055.

[27] N. Halko, P. G. Martinsson, and J. A. Tropp. "Finding Structure with Randomness: Probabilistic Algorithms for Constructing Approximate Matrix Decompositions". In: *SIAM Review* 53.2 (2011), pp. 217–288. URL: https://doi.org/10.1137/090771806.

[28] Eric Hallman. "A Block Bidiagonalization Method for Fixed-Accuracy Low-Rank Matrix Approximation". In: *SIAM Journal on Matrix Analysis and Applications* 43.2 (2022), pp. 661–680. URL: https://doi.org/10.1137/21M1397866.

[29] David A. Huckaby and Tony F. Chan. "On the Convergence of Stewart's QLP Algorithm for Approximating the SVD". In: *Numerical Algorithms* 32.2 (2003), pp. 287–316. URL: https://doi.org/10.1023/A:1024082314087.

[30] Piotr Indyk and Rajeev Motwani. "Approximate Nearest Neighbors: Towards Removing the Curse of Dimensionality". In: *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing*. STOC '98. Dallas, Texas, USA: Association for Computing Machinery, 1998, pp. 604–613. ISBN: 0897919629. URL: https://doi.org/10.1145/276698.276876.

[31] William Johnson and Joram Lindenstrauss. "Extensions of Lipschitz maps into a Hilbert space". In: *Contemporary Mathematics* 26 (Jan. 1984), pp. 189–206. DOI: 10.1090/conm/026/737400.

[32] W. Kahan. "Numerical Linear Algebra". In: *Canadian Mathematical Bulletin* 9.5 (1966), pp. 757–801. DOI: 10.4153/CMB-1966-083-2.

[33] Yann Lecun et al. "Gradient-based learning applied to document recognition". In: 86.11 (1998), pp. 2278–2324. DOI: 10.1109/5.726791.

[34] 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. URL: https://doi.org/10.1007/s10107-009-0306-5.

[35] Michael W. Mahoney and Petros Drineas. "CUR matrix decompositions for improved data analysis". In: *Proceedings of the National Academy of Sciences* 106.3 (2009), pp. 697–702. DOI: 10.1073/pnas.0803205106. URL: https://www.pnas.org/doi/abs/10.1073/pnas.0803205106.

[36] Per-Gunnar Martinsson and Sergey Voronin. "A Randomized Blocked Algorithm for Efficiently Computing Rank-revealing Factorizations of Matrices". In: *SIAM Journal on Scientific Computing* 38.5 (2016), S485–S507. URL: https://doi.org/10.1137/15M1026080.

[37] Cameron Musco and Christopher Musco. "Randomized Block Krylov Methods for Stronger and Faster Approximate Singular Value Decomposition". In: *Advances in Neural Information Processing Systems 28: Annual Conferencevon Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*. Ed. by Corinna Cortes et al. 2015, pp. 1396–1404.

[38] Vladimir Rokhlin, Arthur Szlam, and Mark Tygert. "A Randomized Algorithm for Principal Component Analysis". In: *SIAM Journal on Matrix Analysis and Applications* 31.3 (2010), pp. 1100–1124.

[39] Tamas Sarlos. "Improved Approximation Algorithms for Large Matrices via Random Projections". In: *2006 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS'06)*. 2006, pp. 143–152. DOI: 10.1109/FOCS.2006.37.

[40] Gilbert W. Stewart. "The QLP Approximation to the Singular Value Decomposition". In: *SIAM Journal on Scientific Computing* 20.4 (1999), pp. 1336–1348.

[41] Waqas Sultani, Chen Chen, and Mubarak Shah. "Real-World Anomaly Detection in Surveillance Videos". In: *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2018, pp. 6479–6488. DOI: 10.1109/CVPR.2018.00678.

[42] M.A. Turk and A.P. Pentland. "Face recognition using eigenfaces". In: *Proceedings. 1991 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. 1991, pp. 586–591. DOI: 10.1109/CVPR.1991.139758.

[43] Christopher Williams and Matthias Seeger. "Using the Nyström Method to Speed Up Kernel Machines". In: *Advances in Neural Information Processing Systems*. Ed. by T. Leen, T. Dietterich, and V. Tresp. Vol. 13. MIT Press, 2000. URL: https://proceedings.neurips.cc/paper/2000/file/19de10adbaa1b2ee13f77f679fa1483a-Paper.pdf.

[44] Franco Woolfe et al. "A fast randomized algorithm for the approximation of matrices". In: *Applied and Computational Harmonic Analysis* 25.3 (2008), pp. 335–366. ISSN: 1063-5203. DOI: https://doi.org/10.1016/j.acha.2007.12.002. URL: https://www.sciencedirect.com/science/article/pii/S1063520307001364.

[45] Jianwei Xiao, Ming Gu, and Julien Langou. "Fast Parallel Randomized QR with Column Pivoting Algorithms for Reliable Low-Rank Matrix Approximations". In: *2017 IEEE 24th International Conference on High Performance Computing (HiPC)*. 2017, pp. 233–242. DOI: 10.1109/HiPC.2017.00035.

[46] Jian Xue, Jinyu Li, and Yifan Gong. "Restructuring of Deep Neural Network Acoustic Models with Singular Value Decomposition". In: *Interspeech*. Jan. 2013.

[47] Wenjian Yu, Yu Gu, and Yaohang Li. "Efficient Randomized Algorithms for the Fixed-Precision Low-Rank Matrix Approximation". In: *SIAM Journal on Matrix Analysis and Applications* 39.3 (2018), pp. 1339–1359.

[48] Bolong Zhang and Michael Mascagni. *Pass-Efficient Randomized LU Algorithms for Computing Low-Rank Matrix Approximation*. 2020. DOI: 10.48550/ARXIV.2002.07138. URL: https://arxiv.org/abs/2002.07138.