On the Accuracy of Randomized Low-Rank Matrix Approximations

Elias J. Severinsen*, Ole L. Elvetun†
Department of Mathematics
Norwegian University of Life Sciences (NMBU)
Ås, Norway
{elias.jegervatn.severinsen, ole.elvetun}@nmbu.no

***Abstract*—Low-rank matrix approximations are essential in modern applications like scientific computing and data analysis. In recent years, the volume of data has seen a rapid increase, resulting in a need for efficient and accurate low-rank approximations. The computation of the singular value decomposition (SVD) is fundamental within low-rank approximations, but is computationally infeasible as the data size increases. Randomized SVD (rSVD) provides an efficient solution, approximating the SVD by first projecting the data onto a low-dimensional space before performing the SVD, significantly reducing the computation time. In this paper, we analyze the computation time on data of wide and tall format, comparing the two cases, and analyze the error on matrices with different properties. Our results show that rSVD substantially reduces computation time for large matrices, especially for small target ranks, while achieving good approximation accuracy that depends on the spectral decay of the matrix. We further show that the relative performance on wide versus tall matrices depends on both matrix size and target rank.**

I. Introduction

The need for efficient and accurate numerical algorithms for data analysis is increasingly relevant. Modern scientific and industrial applications produce data sets of massive scale and complexity. In physics simulations, discretization of partial differential equations can result in linear systems with billions of unknowns [19]. In medical image analysis, large-scale studies produce data sets with hundreds of thousands of images [4]. Similarly, in natural language processing, document-term matrices can have tens of thousands of columns and hundreds of rows [2]. Indeed, large-scale data sets naturally occur in a wide range of fields and applications.

The SVD is a fundamental tool in data analysis and numerical linear algebra. It has numerous applications, including dimensionality reduction and high-dimensional data visualization [9], data compression [25], and various regularization techniques [12]. Given a matrix $A \in \mathbb{R}^{m\times n}$, the SVD decomposes the matrix into its main directions of variation, the singular vectors, and a set of singular values describing the importance of the identified directions. However, the computational cost of the SVD is high: for an $n \times n$ matrix, the cost scales as $\mathcal{O}(n^3)$, which makes it infeasible for many large-scale applications. This has motivated the development of randomized algorithms, notably randomized SVD, which provides an efficient alternative at the cost of somewhat lower accuracy.

The rSVD algorithm computes a low-rank approximation of a matrix A by sampling its column space with random test vectors to construct an approximate orthonormal basis Q. The matrix is projected onto this approximate subspace, producing a matrix B of smaller size, which captures the main components of A. Performing the SVD on this reduced version of A gives an efficient way to compute a rank-k approximation.

In this work, we investigate the trade-offs between the classical SVD and its randomized version. The goal is to compare the computational efficiency and accuracy of the rSVD relative to the SVD on a range of matrix shapes, sizes and spectral decay profiles (the distribution of singular values). We use synthetic test matrices, with a prescribed spectral decay, to investigate the effect on the error of the rSVD against the SVD. In addition, we compare the performance of the rSVD on both tall and wide matrices, evaluating how its efficiency varies with matrix dimensions and target approximation rank.

II. Theory

A. Problem formulation

Consider the goal of finding a rank-k approximation to an $m \times n$ matrix A. We seek a matrix X with $\operatorname{rank}(X) \le k$ which approximates A. The problem formulation is

$$\min_{\operatorname{rank}(X)\le k} \|A - X\|_2, \tag{1}$$

where $\|\cdot\|_2$ is the ℓ_2 operator norm, also known as the spectral norm. Equivalently, every rank-k matrix can be viewed as the projection of A onto a k-dimensional subspace of $\mathbb{R}^m$.

Let $Q \in \mathbb{R}^{m\times k}$ have orthonormal columns, i.e., $Q^TQ = I_k$. Then QQ^TA is the orthogonal projection of A onto the k-dimensional subspace $\operatorname{span}(Q)$. The problem can also be formulated as

$$\min_{Q^TQ=I_k} \|A - QQ^TA\|_2. \tag{2}$$

B. Singular value decomposition

The singular value decomposition of a matrix $A \in \mathbb{R}^{m\times n}$ is a factorization of the form

$$A = U\Sigma V^T, \tag{3}$$

where U is an $m\times m$ orthogonal matrix, Σ is an $m\times n$ diagonal matrix with non-negative entries, called the singular values of

A, and V is an $n \times n$ orthogonal matrix. The singular values are ordered such that $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_n$. Every matrix has such a factorization, where Σ is uniquely determined, but U and V are not [17]. The columns of U and V are called the left and right singular vectors of A, respectively.

C. QR factorization

The QR factorization of $A \in \mathbb{R}^{m\times n}$ is a decomposition

$$A = QR, \tag{4}$$

where $Q \in \mathbb{R}^{m\times n}$ has orthonormal columns and $R \in \mathbb{R}^{n\times n}$ is upper triangular. The matrix Q forms an orthonormal basis for $\mathrm{range}(A)$, as described in Section 6 of Golub and Van Loan [10]. The computation of a QR factorization is done through a series of orthogonal transformations, like Givens rotations or Householder transformations (see Appendix B, C).

D. Randomized SVD

The rSVD is a technique used to compute a rank-k approximation of an $m \times n$ matrix without computing the SVD. The core idea is to use random sampling to find a low-rank matrix Q which approximates the range of A, project A onto $\mathrm{span}(Q)$, and compute the SVD of the projection. The projection $B = Q^T A$ will, with high probability, capture the dominant directions of A, and the SVD of B will yield a good approximation [11]. This procedure is summarized in Algorithm 1. The construction of an orthonormal basis Q of Y is done through a QR factorization. Note that the approximation $A \approx QQ^T A$ and the approximate factorization $A \approx U\Sigma V^T$ produced by rSVD have the same error. This can be seen by observing that $QQ^T A = QB = Q\tilde{U}\Sigma V^T$, and setting $U = Q\tilde{U}$.

Algorithm 1 rSVD

```
Input: $A \in \mathbb{R}^{m\times n}$, target rank $k$, oversampling parameter $p$.
Output: Approximate rank-$k$ factorization $U\Sigma V^T$.
1: Generate a random $n \times (k+p)$ test matrix $\Omega$.
2: Compute $Y = A\Omega$.
3: Form an orthonormal basis $Q$ of $Y$.
4: Set $B = Q^T A$.
5: Compute the SVD $B = \tilde{U}\Sigma V^T$.
6: Set $U = Q\tilde{U}$.
```

E. Error estimates

In 1936, C. Eckart and G. Young showed that the solution to Equation (1) is $X = U_k \Sigma_k V_k^T$, where U_k and V_k consist of the first k left and right singular vectors of A, respectively, and Σ_k contains the first k singular values of A [7]. This is known as the Eckart–Young–Mirsky theorem. Alternatively, the solution Q of Equation (2) consists of the first k left singular vectors of A. This implies that

$$\min_{\mathrm{rank}(X)\leq k} \|A - X\|_2 = \sigma_{k+1}, \tag{5}$$

where σ_{k+1} is the $(k+1)$th singular value of A. Equation (5) says that no rank-k approximation of A can have an error smaller than σ_{k+1}.

Let $A \in \mathbb{R}^{m\times n}$, and choose a target rank $k \geq 2$ and an oversampling parameter $p \geq 2$, where $p + k \leq \min\{m, n\}$. Draw a random matrix $\Omega \in \mathbb{R}^{m\times(k+p)}$, with independent and identically distributed (IID) elements from the standard normal distribution, and use Algorithm 1 to obtain a rank-k approximation $X = U\Sigma V^T$. Then,

$$\mathbb{E}\|A - X\|_2 \leq \left(1 + \sqrt{\frac{k}{p-1}}\right)\sigma_{k+1} + \frac{e\sqrt{k+p}}{p}\sqrt{\sum_{j=k+1}^{\min\{m,n\}} \sigma_j^2}, \tag{6}$$

where σ_j is the jth singular value of A, e is Euler's constant, and $\mathbb{E}$ denotes the expected value. For a proof, see Theorem 10.5 in Halko et al. [11].

Noting that $\sigma_j \leq \sigma_{k+1}$ for all $j > k$, Equation (6) implies that

$$\mathbb{E}\|A - X\|_2 \leq \gamma_k \sigma_{k+1}, \tag{7}$$

where γ_k is a constant given by $\gamma_k = (1 + \frac{4\sqrt{k+p}}{p-1}\min\{m, n\})$ and σ_{k+1} is the $(k+1)$th singular value of A.

F. Computational cost

Assume that $A \in \mathbb{R}^{m\times n}$. Standard implementations of the SVD in linear algebra libraries like LAPACK or ScaLAPACK are based on two-stage algorithms [3]. In the first stage, A is reduced to a bi-diagonal form, most commonly through Householder transformations (see Appendix C). In the second stage, the SVD of the bi-diagonal matrix is computed, typically using a QR-based or a divide-and-conquer approach, as described in Section 2 of Blanchard et al. [3]. The LAPACK routine GESDD, used in this study, where 'GE' denotes a general dense matrix and 'SDD' identifies the SVD driver, implements the divide-and-conquer method [1]. This method typically achieves higher performance compared to the QR-based approach [6]. The cost of this process is of the order $\mathcal{O}(mn\min\{m, n\})$, which simplifies to $\mathcal{O}(m^2 n)$ for wide matrices where $m < n$, or to $\mathcal{O}(mn^2)$ for tall matrices where $m > n$.

The rSVD algorithm consists of two stages. In the first stage, an approximate orthogonal basis Q of A is constructed, and in the second stage, the SVD is approximated using the projection of A onto $\mathrm{span}(Q)$. The first stage of Algorithm 1, which constructs the orthogonal basis Q, has a cost of $\mathcal{O}(mk^2)$ as a result of the QR factorization [5]. Stage two, the projection and construction of an approximate SVD, has a cost of $\mathcal{O}(mnk)$ and $\mathcal{O}(nk^2)$ from the projection $B = Q^T A$ and the SVD $B = \tilde{U}\Sigma V^T$, respectively [11]. The overall cost is therefore on the order of $\mathcal{O}(mnk)$. More efficient but less accurate algorithms for stage 2 are proposed, for instance Algorithm 5.2 in Halko et al., which relies on the interpolative decomposition (ID) (see Appendix D), and has a cost of $\mathcal{O}(mn\log k)$ [11].

III. Methods

A. Test matrix generation

To compare the performance of the SVD and the rSVD on matrices with different spectral decay profiles, we generated random test matrices $A = U\Sigma V^T$, where U and V are random orthogonal matrices, and Σ is a diagonal matrix with prescribed singular values $\{\sigma_1, \dots, \sigma_n\}$.

Each orthogonal matrix was constructed by drawing a random $n \times n$ matrix Ω with IID elements from the standard normal distribution, performing a QR decomposition $\Omega = \tilde{Q}R$, and correcting the signs of the orthogonal columns with $Q = \tilde{Q}\Lambda$, where $\Lambda = \operatorname{diag}(\operatorname{sign}(R_{ii}))$, so that the diagonal elements of R are positive. This procedure generated random orthogonal matrices uniformly distributed over the orthogonal group (Haar measure) [18].

To investigate how spectral decay affects accuracy, three sets of prescribed singular values were used. We used the polynomially decaying sequence of the form

$$\sigma_{k+1} = \frac{10}{(1+\alpha k)^2}, \quad k = 0, \dots, n-1, \tag{8}$$

where the parameter α controls the rate of decay. The condition number is $\kappa = \sigma_1/\sigma_n = (1 + \alpha n - \alpha)^2$, and solving for α gives

$$\alpha = \frac{\sqrt{\kappa(n-1)^2} - n + 1}{(n-1)^2}. \tag{9}$$

In this study, we used the condition numbers $\kappa_1 = 2$, $\kappa_2 = 50$, and $\kappa_3 = 1000$, which corresponds to slow, moderate, and a rapid spectral decay (see Fig. 1). The singular values from Equation (8) are placed on the diagonal of Σ, and the test matrix is the result of $A = U\Sigma V^T$.

The size of the test matrices was set to $n \times 5n$, for the values $n_s = 100$, $n_m = 500$, and $n_l = 1000$, which we refer to as the small, medium, and large cases, respectively. For each combination of spectral decay (slow, moderate, rapid) and matrix size (small, medium, large), we generated r random samples depending on the size, where $r = 30$ for the small case, $r = 15$ for the medium case, and $r = 5$ for the large case. An oversampling parameter of $p = 5$ was used for rSVD in all experiments, as suggested by Halko et al. [11]. The random sampling matrices Ω, used in Algorithm 1, had IID standard normal entries.

B. Computational environment

All experiments were performed on a laptop running Fedora Linux 43 with Linux kernel version 6.17. The system was running on an Intel(R) Core(TM) i7-10510U (4 cores, 8 threads) with 16GB of memory.

Python 3.13 was used for all computations, together with NumPy and SciPy for linear algebra routines [13], [26]. The implementation of Algorithm 1 was done using the QR and SVD routines provided by SciPy, which were specified to use the GESDD solver for SVD computations. All library specifications and code used to generate the results in this paper are openly available at: github.com/EliasSev/randomized-svd-paper.

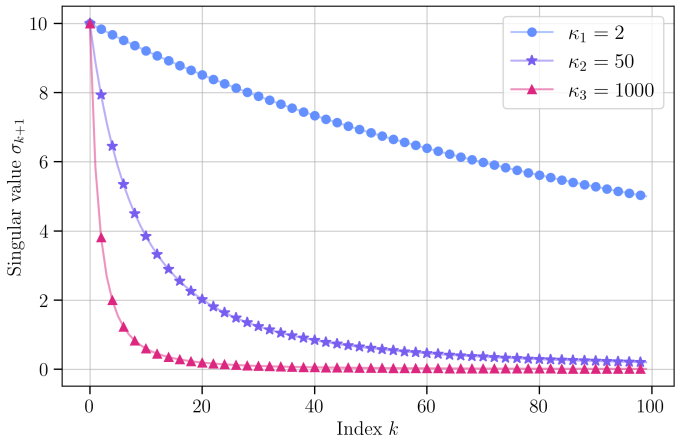

Fig. 1. The slow, moderate, and rapid spectral decay used for the test matrix generation. In this figure, for matrices of small size ($n_s = 100$).

IV. Results

A. Computational performance

When assessing the effect of the matrix size on the computational time, we computed the median value across all test matrices of a given size and across all three spectral decay sets, since the condition number of a dense matrix does not affect the computational cost of the SVD or the QR factorization used in the rSVD [3], [5]. For each matrix size, we reported the median value over 90 (small), 45 (medium), and 15 (large) trials. The results are presented in Fig. 2, and relative runtimes for selected target ranks are listed in Tables I and II.

TABLE I
Computational speedup for rSVD relative to SVD at selected target ranks k for small, medium, and large matrices.

	Small		**Medium**		**Large**	
k	Wide	Tall	Wide	Tall	Wide	Tall
5	13.03	14.87	6.15	19.80	24.79	31.62
10	10.39	13.59	5.82	8.03	17.64	11.53
20	0.21	0.19	2.91	1.80	12.72	8.04
30	0.20	0.13	2.57	1.57	10.89	7.58
40	0.20	0.10	2.45	1.48	10.01	7.27
50	0.19	0.08	2.27	1.46	9.41	6.92

1) Effect of matrix size: Fig. 2 shows the median run times of rSVD as a function of target rank k compared with the median SVD run time, and Table I shows the speedup of rSVD compared to SVD. For small matrices ($n_s = 100$), rSVD achieved a significant speedup compared to SVD for small ranks ($k < 10$), up to 13 times faster. As k increased, the computational time of rSVD increased rapidly, and SVD was faster when k exceeded 10. For medium ($n_m = 500$) and large ($n_l = 1000$) matrices, rSVD consistently performed faster than SVD across all ranks used in this study (5 to 50). Speedups between 7 and 32 were observed for large matrices and between 1.5 and 20 for medium matrices. Overall, the relative advantage of rSVD over SVD generally increased with the matrix size.

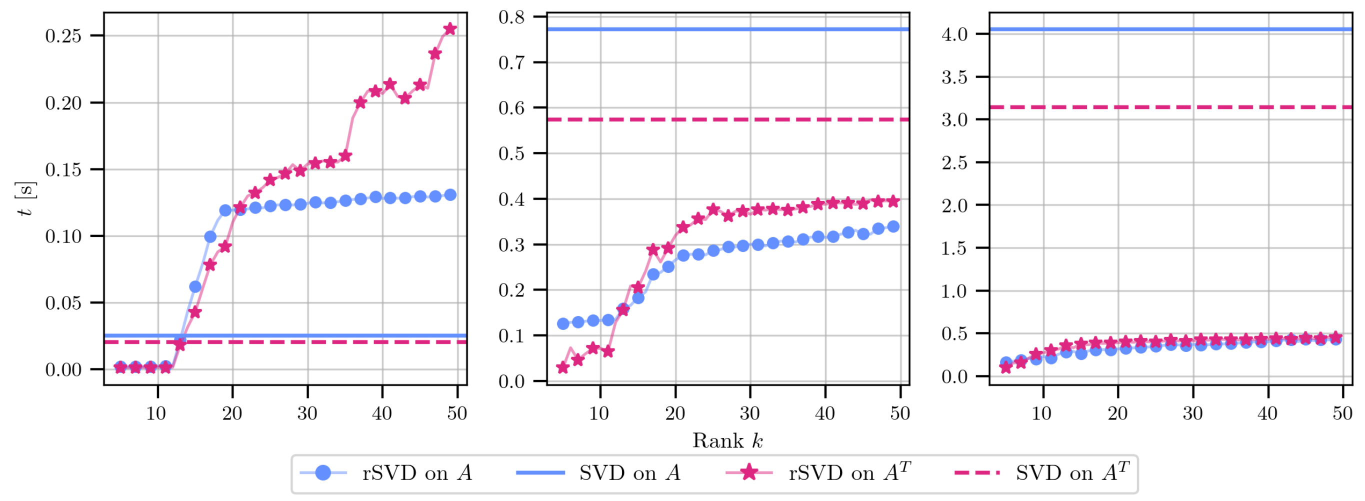

Fig. 2. Median computation time for SVD and rSVD on wide data A and tall data A^T for increasing k and $p = 5$ for three different matrix sizes: Small (left), medium (middle), and large (right).

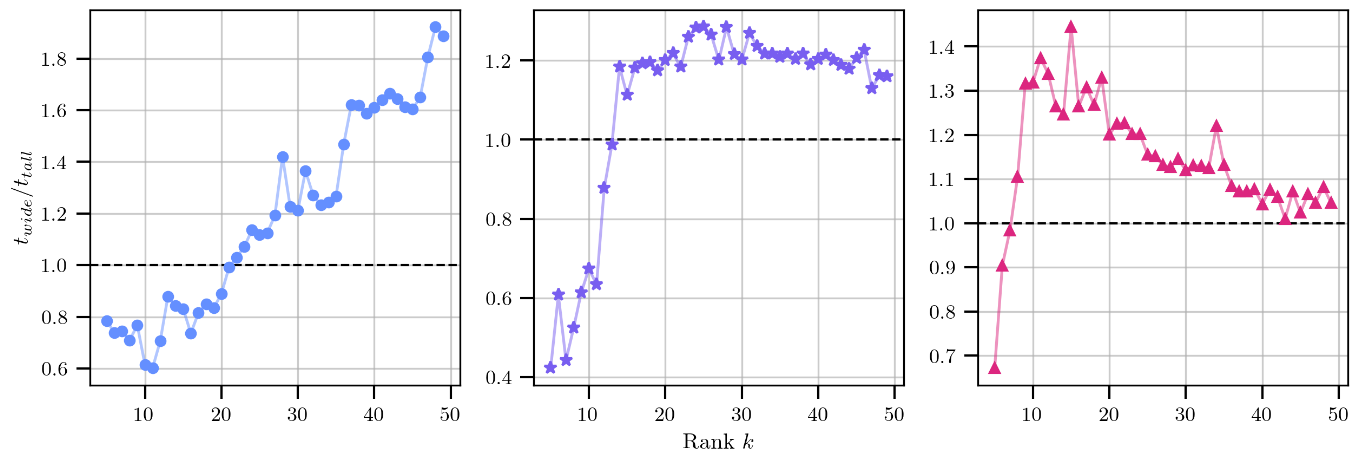

Fig. 3. The median value of t_{wide} divided by t_{tall}, which is the time of rSVD on wide data and on tall data, respectively. Here, three different cases are shown: The small dataset (left), the medium dataset (middle) and the large dataset (right).

2) Effect of matrix shape: To assess the influence of matrix shape, we compared the median computational time of rSVD on wide matrices (t_{wide}) versus tall matrices (t_{tall}) by examining the ratio t_{wide}/t_{tall}, see Fig. 3 and Table II. For small target ranks, rSVD ran faster on wide matrices across all matrix sizes. The crossover rank, at which rSVD became faster on tall matrices, decreased with increasing matrix size, occurring at ranks 22, 14, and 8, for small, medium, and large matrices, respectively. For small matrices, once this crossover rank was exceeded, the relative advantage of tall matrices grew as k approached 50. For medium matrices, the relative speedup on tall matrices remained approximately constant once the rank surpassed the crossover rank. For large matrices, the advantage of tall matrices diminished once k increased beyond the crossover rank.

TABLE II
RELATIVE COMPUTATIONAL TIME FOR WIDE COMPARED TO TALL DATA USING RSVD AT SELECTED TARGET RANKS k FOR SMALL, MEDIUM, AND LARGE MATRICES.

k	**Small**	**Medium**	**Large**
5	0.79	0.42	0.67
10	0.61	0.67	1.32
20	0.89	1.20	1.20
30	1.21	1.20	1.12
40	1.61	1.20	1.04
50	1.89	1.16	1.05

B. Approximation Error

The approximation error was measured using the ℓ_2 operator norm, $\|A - A_k\|_2$, where A_k denotes the rank-k approximation of A. For each combination of matrix size and spectral decay profile, the average error was computed across all

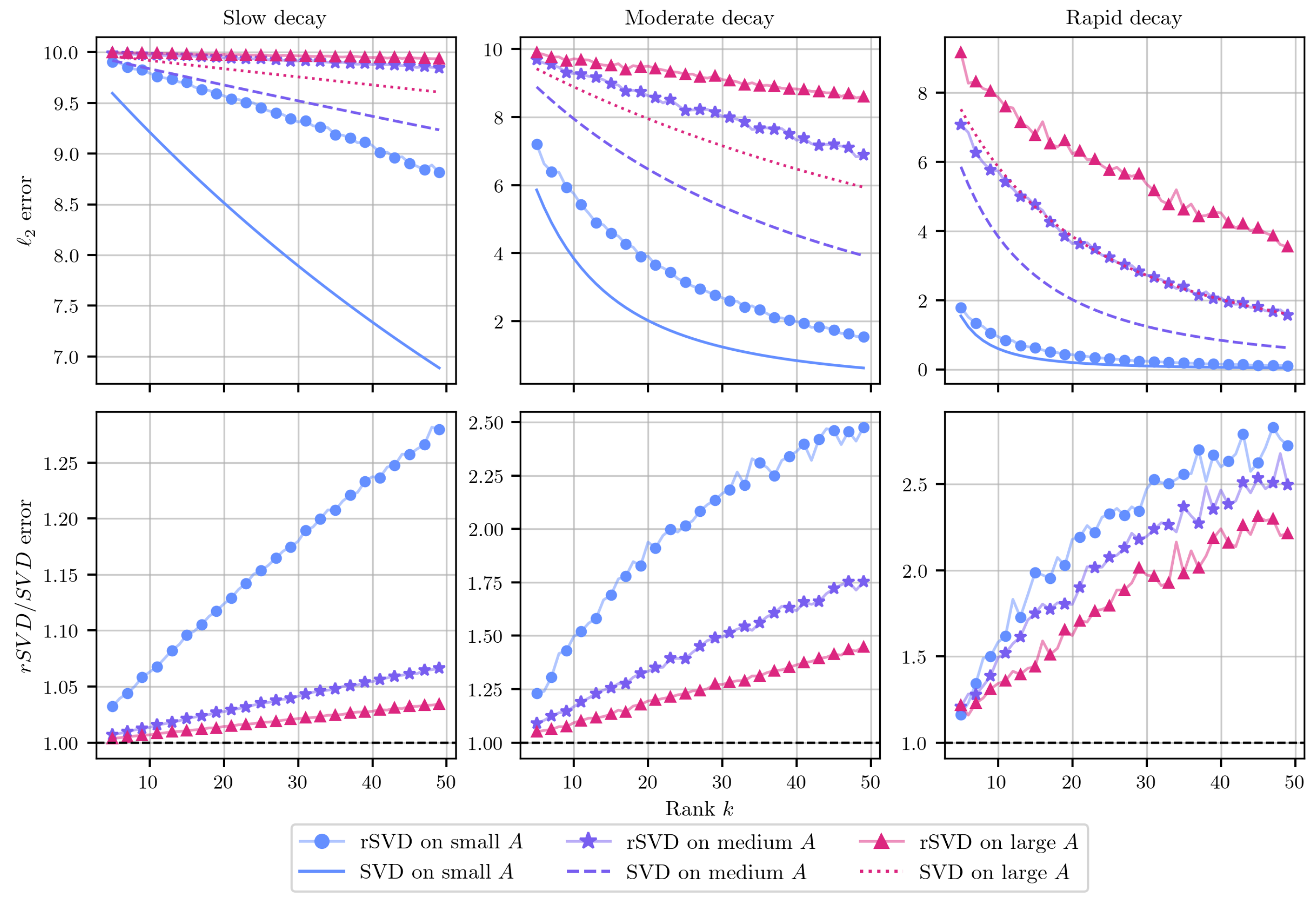

Fig. 4. Average errors for rSVD and the best possible error achieved by the SVD over a range of rank-k approximates, for different spectral decays and matrix sizes. Note that the range on the y-axis is different for each subplot. Top row: The ℓ_2 error for both methods. Bottom row: The relative ℓ_2 error for rSVD compared to SVD.

generated test matrices. Because transposing a matrix does not change its singular values, and since in our experiments the approximation errors obtained by applying rSVD to A and A^T were numerically identical, all error analyses are reported for the wide data only. The results are summarized in Fig. 4 and Table III, showing the three spectral decay profiles (slow, moderate, rapid) and the three matrix sizes (n_s, n_m, n_l).

TABLE III
THE RELATIVE ℓ_2 ERROR FOR RSVD COMPARED TO SVD FOR SELECTED TARGET RANKS k ACROSS DECAY PROFILES AND MATRIX SIZES.

	Slow			**Moderate**			**Rapid**		
k	n_s	n_m	n_l	n_s	n_m	n_l	n_s	n_m	n_l
5	1.0	1.0	1.0	1.2	1.1	1.1	1.2	1.2	1.2
10	1.1	1.0	1.0	1.5	1.2	1.1	1.6	1.5	1.3
20	1.1	1.0	1.0	1.9	1.3	1.2	2.2	1.8	1.6
30	1.2	1.0	1.0	2.2	1.5	1.3	2.5	2.2	2.0
40	1.2	1.1	1.0	2.4	1.6	1.4	2.6	2.5	2.2
50	1.3	1.1	1.0	2.5	1.8	1.4	2.7	2.5	2.2

1) Effect of spectral decay: Fig. 4 shows the absolute approximation error (top row) and the relative error of rSVD compared with SVD (bottom row). As expected, the absolute error was largest for matrices with slowly decaying singular values, both for the rSVD and SVD, which decreased as k increased. For rapidly decaying spectra, the approximation errors decreased much more quickly with increasing rank.

In terms of relative error, rSVD performed worst under rapid spectral decay, although the relative error leveled off as k approached 50. For slow decay, the relative error increased approximately linearly with k. For the moderate decay case, a middle ground was observed: the relative error initially increased roughly linearly and then began to level off as k approached 50.

2) Effect of matrix size: To assess the effect of matrix size, Fig. 4 shows approximation errors for different matrix sizes, and Table III gives relative errors for selected ranks. As expected, the errors increased with both the matrix size and the target rank k. On the contrary, the relative error between the rSVD and SVD was largest for small matrices and decreased with size. Moreover, the difference between slow-, moderate-, and fast-decaying spectra was largest for small matrices: slowly decaying singular values produced significantly higher

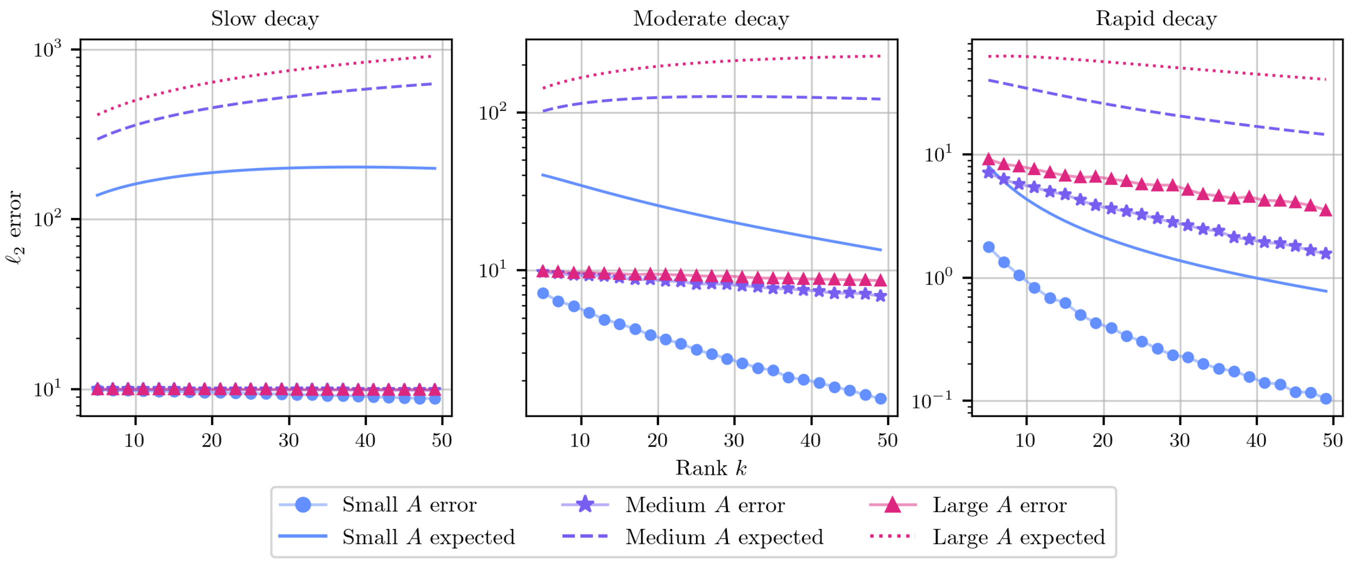

Fig. 5. The observed average rSVD approximation error and the expected upper error bound (Equation (6)) over a range of target ranks. We see that the error bound is severely overestimating the observed error. Notice that the bound is increasing for some estimates, which is seemingly counter-intuitive.

relative errors.

V. Discussion

A. Interpretation of computational performance

The results show that rSVD achieves its greatest speedup compared to the SVD on matrices of large size, and this speedup is greatest for small target ranks k (Fig. 2 and Table I). This can be explained by the computational complexity of the rSVD, $\mathcal{O}(kn^2)$, compared to SVD's complexity of $\mathcal{O}(n^3)$, assuming a matrix of shape $n \times 5n$. As k approaches n, the complexity will be approximately $\mathcal{O}(n^3)$, and the additional computations in the rSVD are no longer beneficial, as was seen for small matrices with target ranks close to 50.

For small matrices, rSVD saw a sudden increase in computational runtime (see Table I). This behavior may be related to implementation details, such as how the QR decomposition or the SVD within the rSVD algorithm is computed. A possible reason is that the matrix size crossed a threshold causing the computations to no longer fit in the fast CPU cache [16]. Further profiling would be required to confirm the exact cause.

Additionally, for small target ranks k, rSVD performs faster on wide matrices than on tall matrices. This can be understood by looking at the dominant costs of Algorithm 1. The first stage constructs an approximate orthonormal basis Q and involves performing a QR-decomposition on the $m \times k$ matrix Y, at a cost of $\mathcal{O}(k^2m)$. The second stage involves forming the projected matrix $B = Q^TA$ with cost $\mathcal{O}(kmn)$, and computing the SVD of B, which costs $\mathcal{O}(k^2n)$. Although the overall cost of Algorithm 1 scales as $\mathcal{O}(kmn)$, the contribution of the quadratic terms $\mathcal{O}(k^2m)$ and $\mathcal{O}(k^2n)$ decides whether tall or wide matrices are more efficient. Specifically, for wide matrices where $m \ll n$, the SVD of B dominates and $\mathcal{O}(k^2n)$ is greater than $\mathcal{O}(k^2m)$. Conversely, for tall matrices where $m \gg n$, the QR decomposition dominates. The observed crossover rank, at which tall matrices become faster, is a balance between these two terms, and depends on both the matrix dimensions and the target rank k.

B. Interpretation of approximation error

As shown in Fig. 4, the approximation error of both the rSVD and SVD depends on spectral decay and target rank k. Rapidly decaying spectra give low-rank approximations with smaller errors, whereas slowly decaying spectra require much larger k to achieve comparable accuracy. For the truncated SVD, this is a direct result of the Eckart–Young–Mirsky theorem, as expressed in Equation (5). For the rSVD, the error bound in Equation (7) tells us the same: when the singular values decay slowly, the σ_{k+1} term remains large, resulting in a larger theoretical upper bound on the expected approximation error.

Fig. 5 further shows that the bound in Equation (6) is highly conservative, especially under slow and moderate spectral decay. The predicted error bound is between one and three orders of magnitude larger than the observed error. For rapidly decaying spectra, the bounds are somewhat tighter, but still overestimate the true error by roughly one order of magnitude.

To understand why the bounds behave this way, it is helpful to investigate the structure of

$$Y = A\Omega, \tag{10}$$

which forms the sample matrix in Algorithm 1. Let $A = U\Sigma V^T$ be the SVD of A, and partition Σ as $\Sigma = \text{diag}(\Sigma_1, \Sigma_2)$, where Σ_1 contains the $k+1$ first singular values and Σ_2 contains the remainder. Correspondingly, partition the right singular vectors as $V = \begin{bmatrix} V_1 & V_2 \end{bmatrix}$. Then A can be expressed as

$$A = U \begin{bmatrix} \Sigma_1 & \\ & \Sigma_2 \end{bmatrix} \begin{bmatrix} V_1^T \\ V_2^T \end{bmatrix}. \tag{11}$$

Multiplying by the random test matrix Ω gives

$$Y = U \begin{bmatrix} \Sigma_1 \Omega_1 \\ \Sigma_2 \Omega_2 \end{bmatrix}, \tag{12}$$

where $\Omega_1 = V_1^T \Omega$ and $\Omega_2 = V_2^T \Omega$. Intuitively, $\Sigma_1 \Omega_1$ contains the dominant actions of A, while $\Sigma_2 \Omega_2$ represents perturbation. When the singular values decay slowly, the perturbation $\Sigma_2 \Omega_2$ remains relatively large, resulting in a larger upper bound on the expected error.

Further, as noted in the Results section, the rSVD approximation errors for A and A^T were numerically identical in our experiments. This can be understood by comparing the random sampling steps $Y = A\Omega$ and $Y' = A^T \tilde{\Omega}$. Similarly to Equation (12), the actions of A^T are captured in $\Sigma_1 \tilde{\Omega}_1$, where $\tilde{\Omega}_1 = U_1^T \tilde{\Omega}$. Hence, for A^T, the quality of the reconstruction depends on the sampling of U, whereas for A, it depends on the sampling of V. In our experiments, both U and V were random orthogonal matrices, and therefore the sampling Y and Y' give numerically identical rank-k approximations using rSVD. For a more comprehensive discussion of rSVD error bounds, see the discussion in Chapter 9 of Halko et al. [11].

C. Interpretation of the relative approximation error

The relative error between rSVD and the SVD approximation (Fig. 4, bottom row) was largest for small matrices and smallest for large ones. Equation (12) explains how the corresponding theoretical bound scales with the problem dimension. In particular, the perturbation term $\Sigma_2 \Omega_2$ grows with the dimension of Ω, increasing the numerator of the relative error. The denominator, however, is the SVD error σ_{k+1}, which does not depend on matrix size. Consequently, as the problem dimension increases, the perturbation grows while the reference error remains fixed, resulting in an increase in the relative error bound, even though the relative error decreases in practice.

Regarding the influence of spectral decay, the experiments showed that a rapid decay gave the highest relative error. This behavior should be interpreted alongside the absolute error: although the relative error is greater for rapid decay, the absolute error is the smallest for such spectra. This illustrates how rSVD and truncated SVD both perform best when the dominant components of A are concentrated in its first few singular vectors, even if small differences appear magnified when expressed relative to the SVD.

D. Practical implications

The observed performance properties of rSVD have several direct implications for practical use in machine learning and scientific computing. The dependence of computational cost on both the matrix shape and the target rank suggests the choice between SVD and rSVD should not be based solely on matrix size, but rather on the interaction between k, m, and n. In particular, the fact that the matrix shape influences performance shows that transposition strategies used by some libraries can meaningfully affect runtime.

These considerations are reflected partially in the heuristics implemented in Scikit-learn. For example, `TruncatedSVD` relies on randomized SVD by default, consistent with the common scenario in which only a small number of components is required [24]. Likewise, Scikit-learn's `PCA` defaults to rSVD when the data are sufficiently large and the target dimension is significantly smaller than the matrix dimension [22]. Both methods rely on Scikit-learn's `randomized_svd` [23]. Notably, this implementation transposes the data when the number of columns exceeds the number of rows. However, our results indicate that this strategy is beneficial only above certain rank thresholds and can be suboptimal for small k.

From a user perspective, our findings suggest several guidelines. Randomized SVD is most effective when (i) the matrix size is large, (ii) the target rank is small relative to the matrix dimensions, and (iii) the singular values decay quickly. Deterministic SVD may be preferable for small matrices or when high approximation accuracy is critical.

E. Limitations and future directions

This study focused on synthetic data without structure in order to isolate the effects of matrix shape, size and spectral decay on the approximation error of rSVD. Real-world data may have additional properties, such as sparsity, symmetry, or noise. For instance, sparse matrices appear in natural language processing [2], graph theory [8], and in discretization of partial differential equations (PDEs) [20]. Specialized algorithms which take advantage of such structures exist for efficiently computing the SVD [21]. Additionally, iterative methods are developed which avoid explicit computation of singular vectors and values entirely, which have important applications in the field of inverse problems [12].

Further, Algorithm 1 may be extended to include power iterations (see Algorithm 4.3 in [11]), which are known to yield substantially more accurate approximations, particularly for matrices with slowly decaying singular spectra. Likewise, the oversampling parameter p, which was fixed to $p = 5$ in this study, may be increased to further improve accuracy of the approximations. Both enhancements, however, increase the computational cost, and their benefits must therefore be weighed against the additional runtime.

VI. Conclusion

This work investigated the computational efficiency and approximation accuracy of rSVD relative to the classical SVD across matrices of varying size, shape, and spectral decay. Our results showed that the computational time for rSVD is substantially lower than that of the classical SVD for large matrices, especially for small target ranks. The accuracy of rSVD is largely determined by the spectral decay, where a rapid decay yields better approximations. We also observed that the crossover rank, at which rSVD becomes faster for tall matrices than for wide ones, depends on both matrix size and target rank. Generally, for small target ranks, matrices of wide format are preferred, where this crossover rank was seen to decrease with the size of the matrices.

Our findings suggest possible improvements to the heuristics in the rSVD implementation in libraries such as Scikit-learn, which should not only depend on matrix shape, but also the target rank and matrix size. From a user perspective, we saw that rSVD is most effective for small target ranks on large matrices, especially with rapid spectral decays.

Limitations of our experiments include the use of synthetic dense matrices, a fixed oversampling parameter, and no power iterations. Future work could explore real-world data, sparse matrices, and alternative algorithms for the rSVD. Overall, rSVD is an effective and computationally efficient alternative to classical SVD in many practical scenarios, especially for low-rank approximations of large matrices.

Acknowledgments

Generative AI tools (ChatGPT, OpenAI) have been used for spelling aid and proofreading. All generated content has been reviewed by the authors. The tools did not contribute to ideas, methodology, or results.

References

[1] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen. *LAPACK Users' Guide*. Society for Industrial and Applied Mathematics, Philadelphia, PA, 3rd edition, 1999.

[2] Ioannis Antonellis and Efstratios Gallopoulos. Exploring term-document matrices from matrix models in text mining. *CoRR*, abs/cs/0602076, 2006.

[3] Pierre Blanchard, Mawussi Zounon, Jack Dongarra, and Nick Higham. Novel SVD Algorithms. Technical Report D2.9, The University of Manchester (UNIMAN), 2019. NLAFET Project Deliverable, Version 2.0.

[4] Aurelia Bustos, Antonio Pertusa, Jose-Maria Salinas, and Maria de la Iglesia-Vayá. Padchest: A large chest x-ray image dataset with multi-label annotated reports. *Medical Image Analysis*, 66:101797, 2020.

[5] Alfredo Buttari, Julien Langou, Jakub Kurzak, and Jack Dongarra. Parallel Tiled QR Factorization for Multicore Architectures. *Concurrency and Computation: Practice and Experience*, 20(13):1573–1590, 2008.

[6] Jack Dongarra, Mark Gates, Azzam Haidar, Jakub Kurzak, Piotr Luszczek, Stanimire Tomov, and Ichitaro Yamazaki. The Singular Value Decomposition: Anatomy of Optimizing an Algorithm for Extreme Scale. *SIAM Review*, 60(4):808–865, 2018.

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

[8] Alan George, John R. Gilbert, and Joseph W. H. Liu. *Graph Theory and Sparse Matrix Computation*. Springer Science & Business Media, 2012.

[9] Felipe L. Gewers, Gustavo R. Ferreira, Henrique F. De Arruda, Filipi N. Silva, Cesar H. Comin, Diego R. Amancio, and Luciano Da F. Costa. Principal Component Analysis: A Natural Approach to Data Exploration. *ACM Comput. Surv.*, 54(4):70:1–70:34, 2021.

[10] Gene H. Golub and Charles F. Van Loan. *Matrix Computations*. JHU Press, 1996.

[11] N. Halko, P. G. Martinsson, and J. A. Tropp. Finding Structure with Randomness: Probabilistic Algorithms for Constructing Approximate Matrix Decompositions. *SIAM Review*, 53(2):217–288, 2011.

[12] Per Christian Hansen. *Discrete Inverse Problems*. Fundamentals of Algorithms. Society for Industrial and Applied Mathematics, 2010.

[13] Charles R. Harris, K. Jarrod Millman, Stéfan J van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585:357–362, 2020.

[14] Alston S. Householder. Unitary Triangularization of a Nonsymmetric Matrix. *Journal of the ACM*, 5(4):339–342, 1958.

[15] Erwin Kreyszig. *Introductory Functional Analysis with Applications*. John Wiley & Sons Inc., 1978.

[16] Monica D. Lam, Edward E. Rothberg, and Michael E. Wolf. The Cache Performance and Optimizations of Blocked Algorithms. In *Proceedings of the Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS IV, page 63–74, New York, NY, USA, 1991. Association for Computing Machinery.

[17] David C. Lay, Steven R. Lay, and Judi J. McDonald. *Linear Algebra and Its Applications*. Pearson, Boston, sixth global edition, 2022.

[18] Francesco Mezzadri. How to generate random matrices from the classical compact groups, 2007. arXiv:math-ph/0609050.

[19] Ruben Ohana, Michael McCabe, Lucas Meyer, Rudy Morel, Fruzsina J. Agocs, Miguel Beneitez, Marsha Berger, Blakesley Burkhart, Keaton Burns, Stuart B. Dalziel, Drummond B. Fielding, Daniel Fortunato, Jared A. Goldberg, Keiya Hirashima, Yan-Fei Jiang, Rich R. Kerswell, Suryanarayana Maddu, Jonah Miller, Payel Mukhopadhyay, Stefan S. Nixon, Jeff Shen, Romain Watteaux, Bruno Régaldo-Saint Blancard, François Rozet, Liam H. Parker, Miles Cranmer, and Shirley Ho. The Well: a Large-Scale Collection of Diverse Physics Simulations for Machine Learning, 2025. arXiv:2412.00568 [cs].

[20] Peter J. Olver. *Introduction to Partial Differential Equations*. Undergraduate Texts in Mathematics. Springer International Publishing, Cham, 2014.

[21] Y. Saad. *Iterative Methods for Sparse Linear Systems*. Society for Industrial and Applied Mathematics, 2003.

[22] scikit-learn developers. PCA - scikit-learn 1.7.2 documentation. https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html, 2025. [Online; accessed 4-Dec-2025].

[23] scikit-learn developers. randomized_svd - scikit-learn 1.7.2 documentation. https://scikit-learn.org/stable/modules/generated/sklearn.utils.extmath.randomized_svd.html, 2025. [Online; accessed 4-Dec-2025].

[24] scikit-learn developers. TruncatedSVD - scikit-learn 1.7.2 documentation. https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.TruncatedSVD.html, 2025. [Online; accessed 4-Dec-2025].

[25] Mr. B. Venkataseshaiah, Ms. Roopadevi K. N., and Stafford Michahial. Image compression using singular value decomposition. *IJARCCE*, 5:208–211, 12 2016.

[26] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C J Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020.

Appendix

A. Operator norm

The ℓ_2 operator norm of $A \in \mathbb{R}^{m\times n}$ is the smallest $c \in \mathbb{R}$ such that $\|Ax\| \le c\|x\|$ for all $x \in \mathbb{R}^n$, denoted $\|A\|_2$. It is a well-known result that

$$\|A\|_2 = \sup_{\|x\|=1} \|Ax\|, \tag{A1}$$

for instance, see Lemma 2.7-2 in Kreyszig [15]. Additionally, it can be shown that

$$\|A\|_2 = \sigma_1, \tag{A2}$$

where σ_1 is the largest singular value of A. See Theorem 6 Section 7.2 in Lay et al. for a proof [17]. Equation (A2) is used when computing $\|A\|_2$ in practice.

B. Givens rotation

A Givens rotation is a rotation in the plane by an angle θ. The rotation is represented by a matrix of the form

$$G = \begin{bmatrix} c & -s \\ s & c \end{bmatrix}, \tag{A3}$$

where $c = \cos\theta$ and $s = \sin\theta$. The method is named after Wallace Givens who introduced it in the 1950s. The matrix can be extended to act on rows i and j of a matrix A, and used repeatedly to convert a matrix into a lower triangular form in order to compute a QR factorization, see Section 5.2 of Golub and Van Loan [10].

C. Householder transformation

Let v be a normal vector defining a hyperplane $\mathcal{H}$. The Householder transformation of a point x is a linear transformation $x \mapsto Px$ representing a reflection about $\mathcal{H}$, where P is the Householder matrix, given by

$$P = I - 2uu^T, \quad u = \frac{v}{\|v\|}. \tag{A4}$$

The transformation was described by Alston S. Householder in a 1958 paper [14]. Householder transformations can be used to reduce matrices to bi-diagonal form, or to lower triangular form as part of a QR factorization, as described in Sections 5.4 and 5.3 of Golub and Van Loan, respectively [10].

D. Interpolative decomposition

Let $A \in \mathbb{R}^{m\times n}$ be of rank k. The ID of A is a factorization of the form

$$A = A_{(:,J)}X, \tag{A5}$$

where $J \subset \{1, \dots, n\}$ is an index set of k indices, $A_{(:,J)}$ a matrix made up of the columns of A indexed by J, and X a matrix where $X_{(:,J)} = I_k$. See Section 3.2.3 in Halko et al. [11].