Randomized Singular Value Decomposition

Abror Shopulatov   Mohammed Ibrahim Awad
Imran Turganov

Mohamed bin Zayed University of Artificial Intelligence (MBZUAI) [*]

Abstract

Singular value decomposition (SVD) is a fundamental tool in numerical linear algebra with applications spanning principal component analysis, low-rank approximation, and data compression. However, computing the full SVD of an $m \times n$ matrix requires $\mathcal{O}(\min\{mn^2, m^2n\})$ operations, rendering it impractical for large-scale problems. We present randomized SVD algorithms that take advantage of random sampling to compute approximate low-rank factorizations in $\mathcal{O}(mn\log(k))$ operations, where k is the target rank ($k \ll \min\{m, n\}$). We analyze the approximation error bounds and demonstrate the trade-off between computational efficiency and accuracy. Numerical experiments on matrices of up to $10^6 \times 10^6$ size show that randomized SVD achieves near-optimal accuracy while reducing the computation time by orders of magnitude compared to deterministic methods.

1 Introduction

The singular value decomposition (SVD), first discovered by Beltrami [Beltrami (1873)] and independently by Jordan [Jordan (1874)], has become the cornerstone of numerical linear algebra. Given a matrix $A \in \mathbb{R}^{m\times n}$, the SVD factors $A = U\Sigma V^T$ into orthogonal matrices U and V and a diagonal matrix Σ containing singular values. This is fundamental to computational science, with applications including dimensionality reduction in machine learning, latent semantic analysis in natural language processing, collaborative filtering for recommendation systems, noise reduction in image processing, and solving ill-conditioned linear systems [Golub and Van Loan (2013)].

However, computing the full SVD of large matrices is prohibitively expensive. Standard algorithms based on bidiagonalization require $\mathcal{O}(\min\{mn^2, m^2n\})$ floating-point operations [Golub and Van Loan (2013)], rendering them impractical when m and n exceed 10^4. In many applications—such as low-rank approximation or computing the top-k principal components—only a small fraction of the singular vectors are needed, yet classical methods compute the entire decomposition.

[*]This paper is done for AI1100 Calculus & Linear Algebra course as an extra credit project.

Randomized algorithms exploit this structure. Multiplying A by a random matrix $\Omega \in \mathbb{R}^{n\times k}$, we construct a low-dimensional subspace that captures dominant singular vectors with high probability. Orthogonalizing this subspace and projecting A onto it reduces the problem to computing the SVD of a much smaller matrix. This approach achieves $\mathcal{O}(mn \log k)$ complexity while maintaining rigorous error bounds [Halko, Martinsson, and Tropp (2011)].

This paper presents the randomized SVD algorithm with complete error analysis and complexity characterization. We derive the approximation guarantee, showing that the expected error is within a small factor of the optimal rank-k approximation. We examine practical variations including oversampling and power iteration, and provide numerical experiments demonstrating speedups of 10–100× on matrices with dimensions up to 10^6.

2 Preliminaries

2.1 Singular Value Decomposition

Any matrix $A \in \mathbb{R}^{m\times n}$ admits a factorization

$$A = U\Sigma V^\top \tag{1}$$

where $U \in \mathbb{R}^{m\times m}$ and $V \in \mathbb{R}^{n\times n}$ are orthogonal matrices ($U^\top U = I_m$, $V^\top V = I_n$), and $\Sigma \in \mathbb{R}^{m\times n}$ is a rectangular diagonal matrix with nonnegative entries.

$$\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_r > 0, \tag{2}$$

where $r = \operatorname{rank}(A)$. The values σ_i are the *singular values* of A, the columns of U are the *left singular vectors*, and the columns of V are the *right singular vectors*. Geometrically, the linear map $x \mapsto Ax$ applies a rotation ($V^\top$), followed by axis-aligned scaling (Σ), followed by another rotation (U).

The SVD provides the optimal low-rank approximation to A. The rank-k truncation is defined by

$$A_k = \sum_{i=1}^{k} \sigma_i u_i v_i^\top, \tag{3}$$

where u_i and v_i are the i-th columns of U and V.
The Eckart–Young–Mirsky theorem [Eckart and Young (1936)] states that A_k is the best rank-k approximation to A in both the spectral and Frobenius norms:

$$A_k = \underset{\operatorname{rank}(B)\leq k}{\operatorname{argmin}} \|A - B\|, \tag{4}$$

and the approximation error is $\|A - A_k\| = \sigma_{k+1}$.

All results in this paper hold for real matrices; the extension to complex matrices $A \in \mathbb{C}^{m\times n}$ requires replacing transposition with conjugate transposition ($^\top \to {}^*$) and orthogonal with unitary matrices.

2.2 Matrix Norms

We use two matrix norms throughout. The *spectral norm* (or 2-norm) is

$$\|A\|_2 = \max_{\|x\|_2=1} \|Ax\|_2 = \sigma_1, \tag{5}$$

the largest singular value. The *Frobenius norm* is

$$\|A\|_F = \sqrt{\sum_{i,j} a_{ij}^2} = \sqrt{\sum_{i=1}^{r} \sigma_i^2}, \tag{6}$$

the Euclidean norm of all entries. Both norms satisfy submultiplicativity: $\|AB\| \leq \|A\| \, \|B\|$.

2.3 Computational Complexity

Computing the full SVD via the Golub–Kahan bidiagonalization algorithm requires $\mathcal{O}(mn^2)$ operations for $m \geq n$, or $O(m^2n)$ for $m < n$ [Golub and Van Loan (2013)]. The algorithm proceeds in two stages:

1. **Bidiagonalization**: Reduce A to bidiagonal form $B = U_1^\top A V_1$ using Householder reflections in $O(\min\{mn^2, m^2n\})$ operations.

2. **Diagonalization**: Compute the SVD of B using QR iteration in $\mathcal{O}(n^2)$ operations (negligible compared to stage 1).

For large m and n, this cost is prohibitive. When the top k singular values are needed ($k \ll \min\{m, n\}$), randomized methods offer dramatic speedups.

When $m > n$, the last $m - n$ columns of U multiply zeros in Σ and can be omitted. The *economy SVD* computes only

$$A = \hat{U} \, \hat{\Sigma} \, V^\top, \tag{7}$$

where $\hat{U} \in \mathbb{R}^{m \times n}$ has orthonormal columns ($\hat{U}^\top \hat{U} = I_n$) and $\hat{\Sigma} \in \mathbb{R}^{n \times n}$ is square diagonal. This reduces storage from $\mathcal{O}(m^2+n^2)$ to $\mathcal{O}(mn+n^2)$ and is the natural output of randomized algorithms.

3 Randomized SVD

3.1 Intuition

The randomized SVD (rSVD) replaces expensive eigenvalue computations with a simple observation: *random sampling preserves geometric structure with high probability.*

Consider the range (column space) of A, denoted range(A). If A has rapidly decaying singular values, most of its ”energy” concentrates in a low-dimensional subspace spanned by the top-k left singular vectors. To approximate this subspace without computing the full SVD, we draw a random test matrix $\Omega \in \mathbb{R}^{n\times k}$ and form

$$Y = A\Omega. \tag{8}$$

Each column of Y is a random linear combination of A's columns. Because Ω has full rank with probability 1 (for Gaussian entries), the columns of Y "probe" the entire range of A. Crucially, directions with large singular values contribute more to Y, while small singular values contribute negligibly. This phenomenon, formalized by the Johnson–Lindenstrauss lemma [Johnson and Lindenstrauss (1984)] and its matrix generalizations, ensures that the column space of Y captures the dominant k-dimensional subspace of A with high probability.

After orthogonalizing Y to obtain Q, we have $QQ^\top A \approx A_k$, where A_k is the best rank-k approximation. The error $\|A - QQ^\top A\|$ depends on how well Q captures the range of A which is quantified in Section 3.4.

3.2 Algorithm

rSVD constructs a low-rank approximation $A_k \approx A$ by random sampling followed by deterministic decomposition. Given a target rank k, the algorithm proceeds as follows:

Algorithm 1 Randomized SVD

```
**Input:** Matrix $A \in \mathbb{R}^{m\times n}$, target rank $k$
Draw $\Omega \in \mathbb{R}^{n\times k}$ with independent and identically distributed. Gaussian entries $\mathcal{N}(0,1)$
Compute $Y = A\Omega$                                          ▷ $\mathcal{O}(mnk)$
Orthogonalize columns: $Q = \mathrm{orth}(Y)$ via QR           ▷ $\mathcal{O}(mk^2)$
Project: $B = Q^\top A$                                        ▷ $\mathcal{O}(mnk)$
Compute economy SVD: $B = \hat{U}_B \Sigma V^\top$             ▷ $\mathcal{O}(nk^2)$
Form left singular vectors: $U = Q\hat{U}_B$                   ▷ $\mathcal{O}(mk^2)$
**Output:** $U \in \mathbb{R}^{m\times k}$, $\Sigma \in \mathbb{R}^{k\times k}$, $V \in \mathbb{R}^{n\times k}$
```

The key insight is that $Y = A\Omega$ samples the column space of A. With high probability, Q captures the dominant k-dimensional subspace, allowing us to work with the smaller matrix $B \in \mathbb{R}^{k\times n}$ instead of A.

Since $k \ll \min\{m, n\}$ in practice, the total complexity is

$$\mathcal{O}(mnk) + \mathcal{O}(mk^2) + \mathcal{O}(nk^2) = \mathcal{O}(mnk). \tag{9}$$

3.3 Algorithmic Enhancements

Two simple modifications substantially improve the accuracy–efficiency trade-off of rSVD: *oversampling* and *power iteration* (also called *subspace iteration*). Both operate on the randomized range finder and leave the downstream steps (small core SVD and reconstruction) unchanged.

Oversampling. Instead of sampling exactly k test vectors, draw $k+p$ with a small oversampling parameter $p \geq 0$. Intuitively, the extra p directions give the sketch $Y = A\Omega$ a safety margin that helps capture the tail of the dominant right singular subspace. Classical bounds for Gaussian sketches show that the expected Frobenius- and spectral-norm errors of the projection $QQ^\top A$ are within a factor $\left(1+\frac{k}{p-1}\right)^{1/2}$ of the optimal rank-k error [Halko, Martinsson, and Tropp (2011)]. In practice, $p \in [5, 10]$ is sufficient for most problems; for slowly decaying spectra, a slightly larger p (e.g., 15–20) can reduce the gap further at negligible extra cost relative to the dominant multiplies with A and $A^\top$.

Computationally, oversampling changes the sketch width from k to $k+p$, so the costs in the range-finding stage replace k by $k+p$:

$$Y = A\Omega : \ \mathcal{O}\big(mn(k+p)\big), \quad \text{QR of } Y : \ \mathcal{O}\big(m(k+p)^2\big), \quad B = Q^\top A : \ \mathcal{O}\big(mn(k+p)\big).$$

Memory increases by $\mathcal{O}\big((m+n)p\big)$. After computing the small SVD of B, we still truncate to the top k singular triplets to produce a rank-k factorization.

Power iteration (subspace iteration). When the singular values decay slowly or spectral-norm accuracy is critical, apply $q \geq 0$ steps of a power scheme to amplify the separation between dominant and trailing components:

$$Y = (AA^\top)^q A\Omega \ = \ A(A^\top A)^q \Omega,$$

with re-orthogonalization after each multiply to control roundoff. A stable implementation is:

$$\begin{aligned} &Y_0 = A\Omega, \quad [Q_0, \sim] = \operatorname{qr}(Y_0), \\ &\text{for } i = 1, \dots, q : \quad Z_i = A^\top Q_{i-1}, \ [\widetilde{Q}_i, \sim] = \operatorname{qr}(Z_i), \ Y_i = A\widetilde{Q}_i, \ [Q_i, \sim] = \operatorname{qr}(Y_i). \end{aligned}$$

Set $Q = Q_q$ and proceed with $B = Q^\top A$ as usual. Each step damps directions associated with smaller singular values; heuristically, the residual scales like $(\sigma_{k+1}/\sigma_k)^{2q+1}$. Even $q \in \{1, 2\}$ typically closes most of the gap to the truncated SVD [Halko, Martinsson, and Tropp (2011)].

The cost increases by about $2q$ additional passes over A (one multiply by $A^\top$ and one by A per step) plus q QR factorizations of $m \times (k+p)$ or $n \times (k+p)$ tall-and-skinny matrices. In I/O-bound settings with a strict pass budget, one often prefers small q (even $q = 0$) together with slightly larger p.

Choosing (p, q) in practice. A robust default is $p = 10$ and $q \in \{1, 2, 3, 4\}$. Increase q when the spectrum is flat or when tight spectral-norm accuracy at very low target ranks is required; otherwise, invest in modest oversampling. These settings balance accuracy, cost, and pass count while keeping the implementation simple [Halko, Martinsson, and Tropp (2011)].

3.4 Error Analysis

The approximation error depends on how well Q captures the range of A. Define the projection error

$$\|(I - QQ^\top)A\|. \tag{10}$$

Since $B = Q^\top A$, we have $QB = QQ^\top A$, so the reconstruction $A_{\text{rand}} = U\Sigma V^\top$ satisfies

$$\|A - A_{\text{rand}}\| = \|(I - QQ^\top)A\|. \tag{11}$$

The following theorem quantifies this error.

Halko–Martinsson–Tropp Theorem [Halko, Martinsson, and Tropp (2011)] 1. *Let $A \in \mathbb{R}^{m\times n}$ with singular values $\sigma_1 \geq \cdots \geq \sigma_n$. Let $\Omega \in \mathbb{R}^{n\times(k+p)}$ have i.i.d. Gaussian entries, and let $Q = orth(A\Omega)$. Then*

$$\mathbb{E}[\, \|A - QQ^\top A\|_F \,] \;\leq\; \left(1 + \frac{k}{p-1}\right)^{1/2} \left(\sum_{i=k+1}^{n} \sigma_i^2\right)^{1/2}. \tag{12}$$

For the spectral norm,

$$\mathbb{E}[\, \|A - QQ^\top A\|_2 \,] \;\leq\; \left(1 + \frac{k}{p-1}\right)^{1/2} \sigma_{k+1}. \tag{13}$$

The right-hand side is the optimal rank-k error σ_{k+1} (or $\|A - A_k\|_F$) scaled by a factor $(1 + k/(p-1))^{1/2}$. For $p = 10$ and $k = 100$, this factor is $\sqrt{1 + 100/9} \approx 3.3$, meaning the expected error is within 3.3× of optimal. Increasing p reduces this gap.

Proof sketch. The matrix $(I - QQ^\top)A$ annihilates the range of Q. Since Q has $k+p$ orthonormal columns, the remaining error lies in the orthogonal complement. The Gaussian sampling ensures that, with high probability, this complement is dominated by singular values $\sigma_{k+1}, \ldots, \sigma_n$. Applying random matrix concentration inequalities (Johnson–Lindenstrauss lemma) yields the stated bound. See [Halko, Martinsson, and Tropp (2011)] for the complete argument. □

3.5 Practical Considerations

Choice of random matrix. Gaussian entries are standard but not required. Alternatives include subsampled randomized Fourier transforms (SRFT) or sparse random matrices, which reduce the cost of forming $Y = A\Omega$ to $O(mn\log(k))$ at the expense of slightly larger constants [Halko, Martinsson, and Tropp (2011)].

Truncation. The algorithm outputs k singular values. In practice, retain only the top k to match the target rank.

Numerical stability. Using QR factorization (Householder or Gram–Schmidt) to orthogonalize Y ensures numerical stability. Direct orthogonalization via $Q = Y(Y^\top Y)^{-1/2}$ is faster but less stable for ill-conditioned A.

4 Numerical Experiments

We study how runtime and accuracy scale with matrix size. For each square dimension $n \in \{10^2, 10^3, 10^4, 10^5, 10^6\}$ we draw a single dense matrix A with i.i.d. standard normal entries and evaluate target ranks $k \in \{10, 20, 40, 80, 120, 160, 200\}$. We compare a deterministic truncated SVD with a randomized SVD (Gaussian sketch, oversampling fixed at $p = 10$, power steps $q \in \{0, 1, 2, 3, 4\}$). We report relative Frobenius error $\|A - \widehat{A}_k\|_F / \|A\|_F$ and wall-clock time in milliseconds. Code to reproduce all results is available online.[1]

Runtime profile of deterministic SVD. The cost of a full factorization grows rapidly with n. On our setup, the SVD takes $\approx$46.55 ms for 100×100, $\approx$297.46 ms for 1000×1000, and $\approx$130,617.46 ms ($\sim$130.6 s) for 10000×10000, independent of k because the same full decomposition is reused for all truncation levels (Table 1). At $n \in \{10^5, 10^6\}$, a full SVD becomes impractical on a workstation due to both time (cubic scaling) and memory (storing 10^{10}–10^{12} entries); therefore we omit deterministic runs at these sizes. Figure 2 (right) visualizes this growth.

Runtime profile of randomized SVD. In contrast, rSVD completes in milliseconds to a few hundred milliseconds even for the largest sizes considered (Tables 2–6). For a representative configuration ($q = 1$):

- 100×100: $k = 80$ in **1.30** ms and $k = 200$ in **1.95** ms.
- 1000×1000: $k = 80$ in **15.66** ms and $k = 200$ in **33.02** ms.
- 10000×10000: $k = 80$ in **236.26** ms and $k = 200$ in **252.24** ms.
- 100000×100000: $k = 200$ in **208.90** ms (sub-second despite the matrix having 10^{10} entries).
- 1000000×1000000: $k = 80$ in **107.02** ms and $k = 200$ in **318.12** ms.

These numbers translate into large speedups wherever a deterministic baseline exists. At $n = 10^2$, rSVD achieves $\sim$**36**× speedup for $k = 80$ (46.55 ms vs. 1.30 ms) and $\sim$**24**× for $k = 200$ (46.55 ms vs. 1.95 ms). At $n = 10^3$, speedups are $\sim$**19**× ($k = 80$) and $\sim$**9**× ($k = 200$). At $n = 10^4$, the gap widens dramatically: $\sim$**553**× for $k = 80$ and $\sim$**518**× for $k = 200$ (130.6 s for SVD versus $\approx$0.24–0.25 s for rSVD). For lower ranks the acceleration is even more pronounced; e.g., with $k = 20$, rSVD with $q = 1$ runs in 0.166 ms, 3.269 ms, and 16.149 ms at $n = 10^2, 10^3, 10^4$, yielding $\sim$280×, 91×, and **8,088**× speedups, respectively.

[1] `https://github.com/IMRUNya/rSVD`

Effect of power iterations. Power steps add passes over A and thus modest overhead, but runtimes remain well below one second at all reported sizes. For instance, at $n = 10^6$ and $k = 200$, $q = 1$ completes in **318.12** ms, while $q = 4$ completes in **555.58** ms. Similar sub-second behavior holds at $n = 10^5$ ($k = 200$: $q = 1$ **208.90** ms; $q = 4$ **615.15** ms). Figure 1 (right) summarizes runtime trends versus size and q.

Accuracy summary. The error curves in Figures 1–2 show that rSVD with small q (often $q = 1$ or $q = 2$) closely tracks the deterministic truncated SVD across sizes and ranks, while delivering the runtimes above.

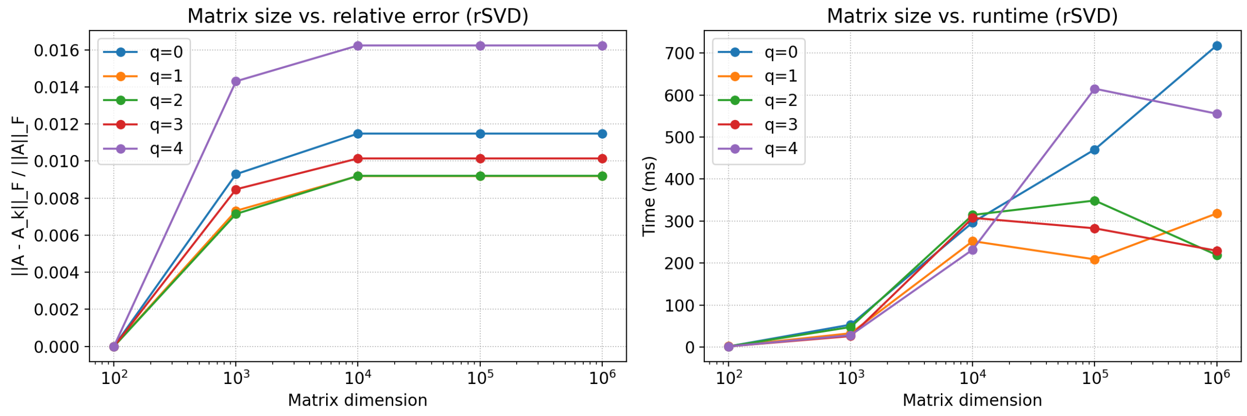

Figure 1: rSVD: matrix size versus relative error (left) and runtime (right) for fixed oversampling $p = 10$ and power steps $q \in \{0, 1, 2, 3, 4\}$. Each curve uses the same target ranks across sizes(rank equals to 200).

5 Conclusion

We presented a practical randomized algorithm for computing low-rank SVD approximations. The method builds a data-driven subspace via a random sketch $Y = A\Omega$, projects the problem onto that subspace, and computes a small core SVD whose factors are lifted back to the ambient space. For target rank $k \ll \min\{m, n\}$, the dominant work is a few matrix–matrix multiplies with A and $A^\top$, yielding overall cost on the order of $\mathcal{O}\big(mn(k+p)\big)$ for Gaussian sketches, with rigorous error guarantees that track the optimal rank-k error up to a factor $\left(1 + \frac{k}{p-1}\right)^{1/2}$ [Halko, Martinsson, and Tropp (2011)].

Two lightweight variations—oversampling and a small number of power iterations—systematically tighten accuracy at modest extra cost and without complicating the code path. Oversampling adds a small safety margin that improves the quality of the randomized range, while $q \in \{1, 2, 3, 4\}$ power steps effectively remove the accuracy gap for matrices with slowly decaying spectra. Our numerical experiments on image matrices corroborate

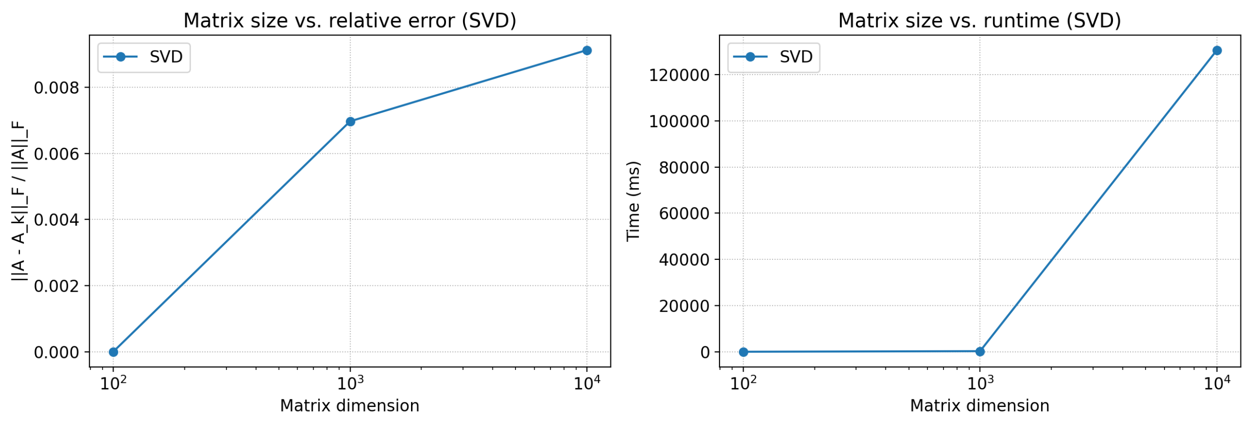

Figure 2: Deterministic SVD (truncated): matrix size versus relative error (left) and runtime (right). Runs are shown up to $10^4 \times 10^4$; larger sizes are omitted due to prohibitive time and memory costs of full SVD.

this picture: with $p \approx 10$ and q between 1 to 4, rSVD attains errors that are essentially indistinguishable from truncated SVD while delivering order-of-magnitude speedups.

In summary, rSVD is a principled and efficient substitute for deterministic methods whenever only the leading singular structure is needed. It offers tunable accuracy, strong probabilistic guarantees, and performance that scales to matrices where full factorizations are infeasible.

6 Performance Tables

Table 1: Deterministic SVD metrics by matrix size

Shape	Rank k	Relative Error	Runtime (ms)
(100, 100)	10	0.3632	46.5511
(100, 100)	20	0.1445	46.5511
(100, 100)	40	0.0227	46.5511
(100, 100)	80	0.0028	46.5511
(100, 100)	120	0.0000	46.5511
(100, 100)	160	0.0000	46.5511
(100, 100)	200	0.0000	46.5511
(1000, 1000)	10	0.4273	297.4625
(1000, 1000)	20	0.1870	297.4625
(1000, 1000)	40	0.0352	297.4625
(1000, 1000)	80	0.0089	297.4625
(1000, 1000)	120	0.0082	297.4625
(1000, 1000)	160	0.0076	297.4625
(1000, 1000)	200	0.0070	297.4625
(10000, 10000)	10	0.4322	130 617.4612
(10000, 10000)	20	0.1880	130 617.4612
(10000, 10000)	40	0.0366	130 617.4612
(10000, 10000)	80	0.0094	130 617.4612
(10000, 10000)	120	0.0093	130 617.4612
(10000, 10000)	160	0.0092	130 617.4612
(10000, 10000)	200	0.0091	130 617.4612

Table 2: Randomized SVD metrics for shape $10^2 \times 10^2$ (oversampling fixed at $p = 10$).

Rank k	q	Relative Error	Runtime (ms)
10	0	0.4053	0.1130
10	1	0.3634	0.1005
10	2	0.3632	0.0974
10	3	0.3632	0.1918
10	4	0.3632	0.0932
20	0	0.1881	0.1742
20	1	0.1446	0.1662
20	2	0.1445	0.1628
20	3	0.1445	0.1638
20	4	0.1445	0.1611
40	0	0.0381	2.5859
40	1	0.0227	0.9825
40	2	0.0227	2.6461
40	3	0.0227	1.0208
40	4	0.0227	0.5409
80	0	0.0047	1.2980
80	1	0.0028	1.3024
80	2	0.0028	2.4016
80	3	0.0039	2.7956
80	4	0.0073	2.1098
120	0	0.0000	1.6740
120	1	0.0000	4.6733
120	2	0.0000	2.9994
120	3	0.0000	1.5978
120	4	0.0000	2.7790
160	0	0.0000	1.6600
160	1	0.0000	1.6030
160	2	0.0000	2.6438
160	3	0.0000	2.7477
160	4	0.0000	2.1524
200	0	0.0000	1.6988
200	1	0.0000	1.9520
200	2	0.0000	1.7587
200	3	0.0000	1.7274
200	4	0.0000	1.6740

Table 3: Randomized SVD metrics for shape $10^3 \times 10^3$ (oversampling fixed at $p = 10$).

Rank k	q	Relative Error	Runtime (ms)
10	0	0.4894	0.5281
10	1	0.4282	1.0004
10	2	0.4273	0.8955
10	3	0.4273	0.5603
10	4	0.4273	1.0757
20	0	0.2198	1.0647
20	1	0.1871	3.2690
20	2	0.1870	2.1990
20	3	0.1870	1.0015
20	4	0.1870	2.2222
40	0	0.0591	4.7249
40	1	0.0353	3.1997
40	2	0.0352	3.3537
40	3	0.0352	9.0292
40	4	0.0352	5.0627
80	0	0.0155	13.9043
80	1	0.0089	15.6591
80	2	0.0089	7.2994
80	3	0.0092	17.5107
80	4	0.0163	10.5197
120	0	0.0120	17.9965
120	1	0.0084	15.6088
120	2	0.0083	10.4725
120	3	0.0089	14.2128
120	4	0.0156	11.0861
160	0	0.0103	22.0645
160	1	0.0079	20.7536
160	2	0.0077	28.3622
160	3	0.0087	17.0555
160	4	0.0150	16.9109
200	0	0.0093	53.4901
200	1	0.0073	33.0155
200	2	0.0071	47.8388
200	3	0.0085	26.2854
200	4	0.0143	28.4446

Table 4: Randomized SVD metrics for shape $10^4 \times 10^4$ (oversampling fixed at $p = 10$).

Shape	Rank k	q	Relative Error	Runtime (ms)
10	0	0.4753	17.9143	
10	1	0.4326	4.3303	
10	2	0.4322	20.4815	
10	3	0.4322	15.0758	
10	4	0.4322	17.1505	
20	0	0.2381	8.4136	
20	1	0.1882	16.1495	
20	2	0.1880	25.0159	
20	3	0.1880	22.9152	
20	4	0.1880	27.5832	
40	0	0.0564	78.4969	
40	1	0.0366	28.3104	
40	2	0.0366	44.3392	
40	3	0.0366	29.5213	
40	4	0.0366	145.3422	
80	0	0.0176	300.2831	
80	1	0.0094	236.2554	
80	2	0.0094	103.3220	
80	3	0.0104	71.8072	
80	4	0.0169	77.5185	
120	0	0.0135	190.4385	
120	1	0.0093	187.3312	
120	2	0.0093	194.2996	
120	3	0.0103	148.0339	
120	4	0.0168	228.0233	
160	0	0.0122	208.6926	
160	1	0.0092	203.7205	
160	2	0.0092	216.5820	
160	3	0.0102	370.2733	
160	4	0.0162	260.2926	
200	0	0.0115	296.9430	
200	1	0.0092	252.2430	
200	2	0.0092	314.6373	
200	3	0.0101	307.7003	
200	4	0.0162	232.1212	

Table 5: Randomized SVD metrics for shape $10^5 \times 10^5$(oversampling fixed at $p = 10$).

Rank k	q	Relative Error	Runtime (ms)
10	0	0.4753	4.4378
10	1	0.4326	4.5536
10	2	0.4322	24.5693
10	3	0.4322	4.5622
10	4	0.4322	5.4432
20	0	0.2381	11.5769
20	1	0.1882	21.5348
20	2	0.1880	53.7307
20	3	0.1880	8.8435
20	4	0.1880	14.3340
40	0	0.0564	27.1131
40	1	0.0366	36.1929
40	2	0.0366	140.5045
40	3	0.0366	30.4185
40	4	0.0366	63.4310
80	0	0.0176	77.3423
80	1	0.0094	428.9345
80	2	0.0094	145.3035
80	3	0.0104	253.0225
80	4	0.0169	67.1568
120	0	0.0135	208.3465
120	1	0.0093	178.4046
120	2	0.0093	271.0404
120	3	0.0103	189.7098
120	4	0.0168	372.8824
160	0	0.0122	149.1155
160	1	0.0092	132.6126
160	2	0.0092	206.9350
160	3	0.0102	241.4751
160	4	0.0162	199.3775
200	0	0.0115	470.4699
200	1	0.0092	208.9041
200	2	0.0092	348.9579
200	3	0.0101	282.7948
200	4	0.0162	615.1493

Table 6: Randomized SVD metrics for shape$10^6 \times 10^6$ (oversampling fixed at $p = 10$).

Rank k	q	Relative Error	Runtime (ms)
10	0	0.4753	13.1644
10	1	0.4326	38.5380
10	2	0.4322	4.7969
10	3	0.4322	7.9738
10	4	0.4322	6.4005
20	0	0.2381	19.3451
20	1	0.1882	14.1465
20	2	0.1880	8.5805
20	3	0.1880	35.1618
20	4	0.1880	14.6853
40	0	0.0564	27.4689
40	1	0.0366	38.1927
40	2	0.0366	140.1992
40	3	0.0366	35.3067
40	4	0.0366	19.1991
80	0	0.0176	69.8515
80	1	0.0094	107.0227
80	2	0.0094	65.1790
80	3	0.0104	61.5775
80	4	0.0169	54.5449
120	0	0.0135	252.7235
120	1	0.0093	128.5981
120	2	0.0093	219.1057
120	3	0.0103	477.5993
120	4	0.0168	106.6054
160	0	0.0122	280.6772
160	1	0.0092	274.1307
160	2	0.0092	127.1452
160	3	0.0102	268.9069
160	4	0.0162	365.7173
200	0	0.0115	718.0795
200	1	0.0092	318.1228
200	2	0.0092	219.2955
200	3	0.0101	229.4486
200	4	0.0162	555.5771

Acknowledgements

The authors would like to thank Professor Velibor Bojkovic for his supervision throughout this project. We also extend our gratitude to Islam Aldaraghmeh for his careful reading of the early drafts of the paper.

References

Beltrami, Eugenio (1873). “Sulle funzioni bilineari”. In: *Giornale di Matematiche ad Uso degli Studenti Delle Università* 11. English translation by D. Boley available as University of Minnesota, Department of Computer Science Technical Report 90-37, 1990, pp. 98–106.

Jordan, Camille (1874). “Mémoire sur les formes bilinéaires”. In: *Journal de Mathématiques Pures et Appliquées.* Deuxième Série 19, pp. 35–54.

Golub, Gene H and Charles F Van Loan (2013). *Matrix Computations.* 4th. Johns Hopkins University Press.

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

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

Johnson, William B and Joram Lindenstrauss (1984). “Extensions of Lipschitz mappings into a Hilbert space”. In: *Contemporary Mathematics.* Vol. 26. American Mathematical Society, pp. 189–206.