CENG 371
HW 4

Aneliya Abdimalik
2547651

December 27, 2024

Question 2 Response

1. Relative Error Analysis

To analyze the relative errors, I calculated the following metrics for both *cameraman.jpg* and *fingerprint.jpg*:

$$\text{Relative Error for } \texttt{approximate\_svd} = \frac{\|(U_k, \Sigma_k, V_k^T) - (U\Sigma V^T)\|_2}{\|U\Sigma V^T\|_2}$$

$$\text{Relative Error for } \texttt{svds} = \frac{\|(U'_k, \Sigma'_k, V_k'^T) - (U\Sigma V^T)\|_2}{\|U\Sigma V^T\|_2}$$

I plotted these relative errors against k for both images. The results indicate that as k increases (in my case up to k = 50), the relative errors for both methods tend to decrease, following the trend of the actual SVD error. However, due to the randomness introduced in the approximation method (via Ω), the relative error for `approximate_svd` fluctuates slightly for smaller k values. Overall, the average relative error aligns closely with the actual SVD's error as k approaches 50.

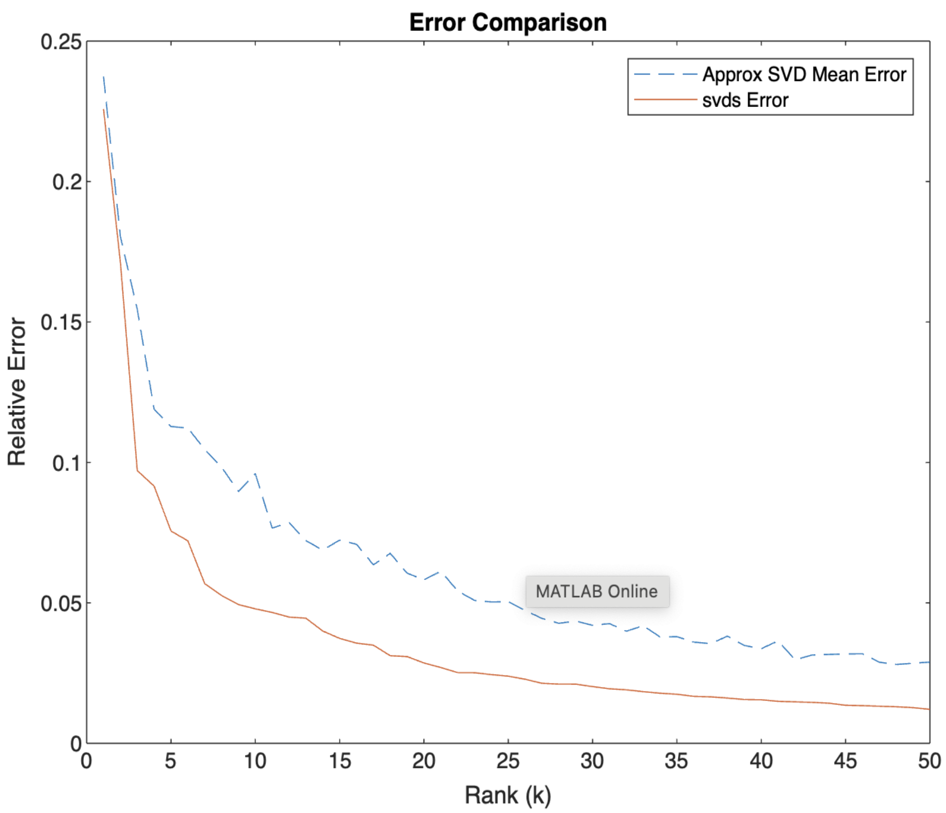

Figure 1: Relative error graph for *cameraman.jpg*

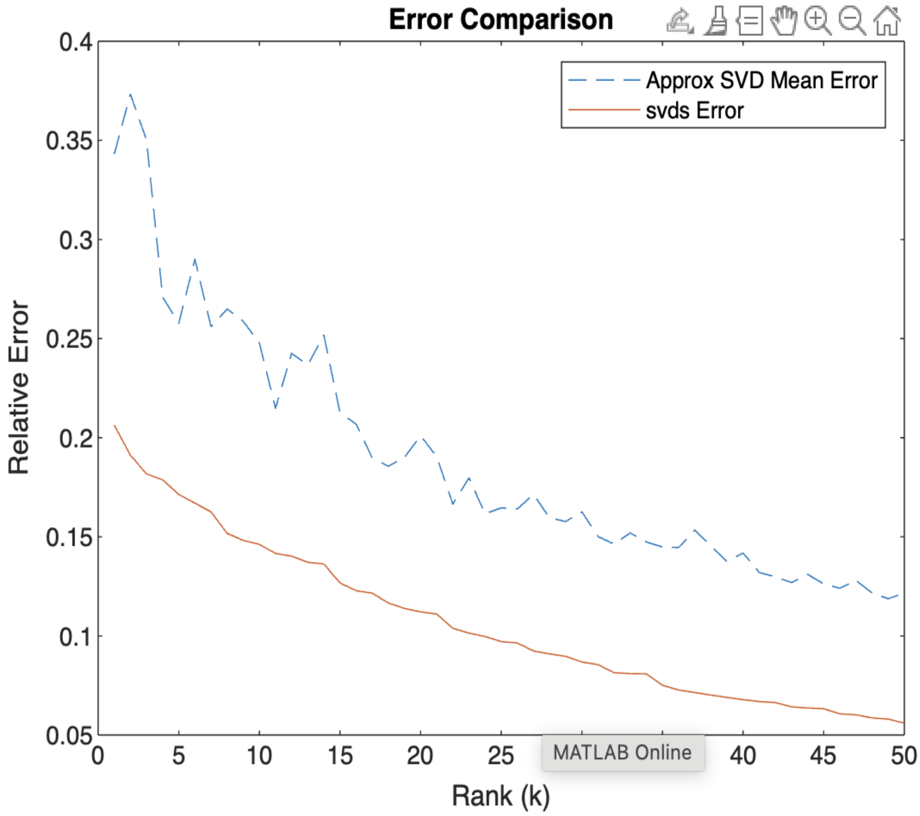

Figure 2: Relative error graph for *fingerprint.jpg*

2. Runtime Analysis

I compared the runtime of `approximate_svd` and `svds` for both images across different values of k (up to 50). The runtime for `approximate_svd` was significantly faster than `svds`, highlighting its efficiency:

- For *cameraman.jpg*, `approximate_svd` was approximately 100 times faster.
- For *fingerprint.jpg*, it was about 10 times faster.

This speed improvement arises because `approximate_svd` operates on a smaller matrix $B = A\Omega$, where $B \in \mathbb{R}^{m\times(k+p)}$, with $k \ll n$. This makes it particularly suitable for scenarios requiring quick computation.

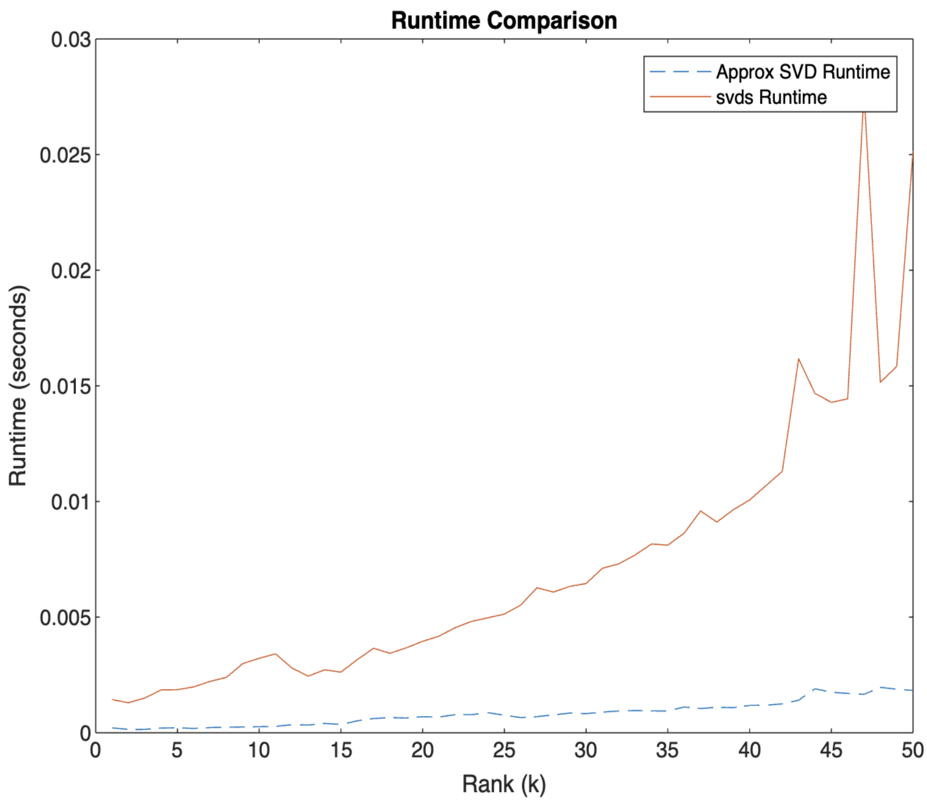

Figure 3: Runtime graph for *cameraman.jpg*

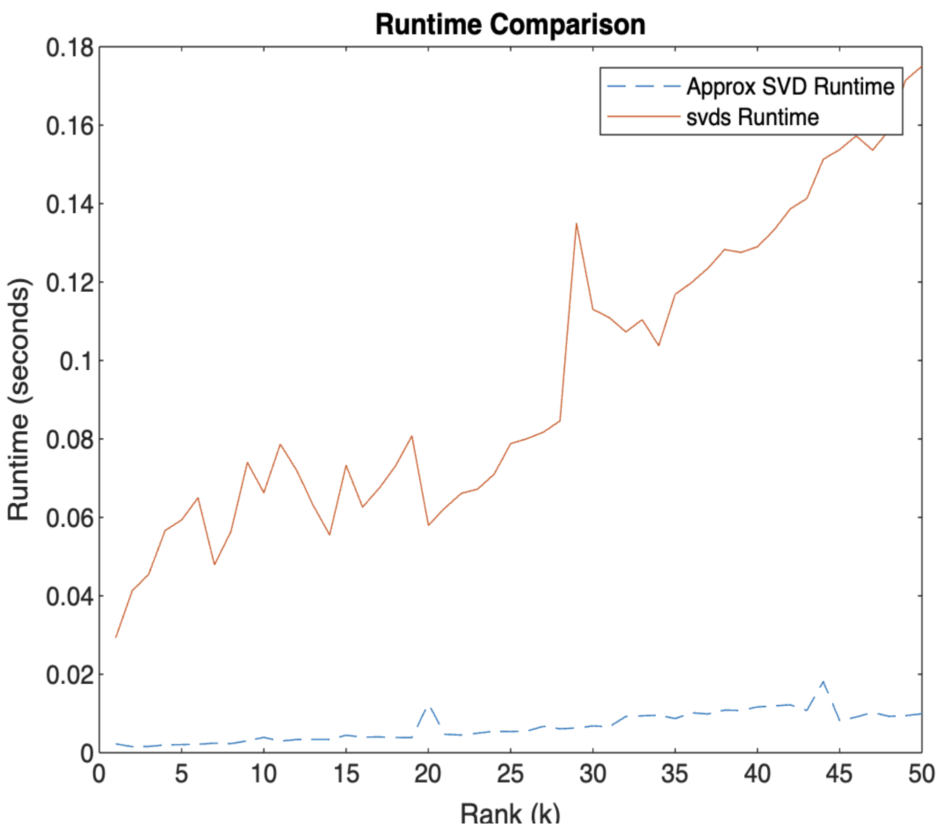

Figure 4: Runtime graph for *fingerprint.jpg*

3. Qualitative Analysis

For a qualitative comparison, I reconstructed the output matrices for selected values of k (e.g., $k = 5$, $k = 25$, and $k = 50$) into images for both datasets.

- **For smaller k (e.g., $k = 5$):** The reconstructed images retained a fair amount of detail but displayed noticeable artifacts and uneven quality. This inconsistency likely stems from the randomness introduced by Ω.

- **For moderate k (e.g., $k = 25$):** The images exhibited significant improvement in quality, with most details being preserved and artifacts being less noticeable.
- **For larger k (e.g., $k = 50$):** The reconstructed images were nearly close to the originals to the human eye. At this point, the approximation achieved high fidelity, with only minimal differences that were visually imperceptible.

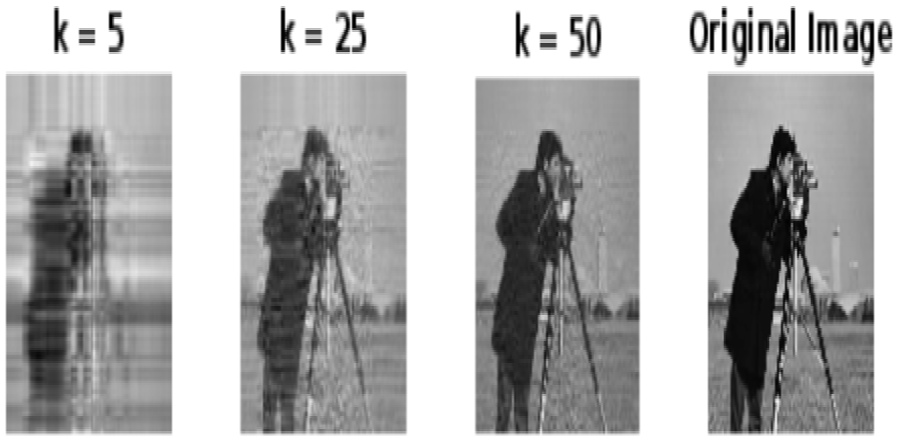

Figure 5: SVD Decomposition of cameraman.jpg for k = 5, 25, 50

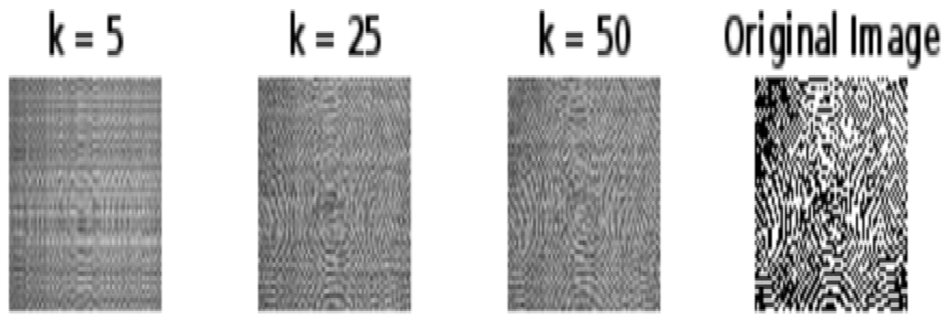

Figure 6: SVD Decomposition of fingerprint.jpg for k = 5, 25, 50

These results demonstrate that as k increases, the quality of reconstructed images improves and converges toward the original.

4. Suggested Use Cases

Based on the analysis, `approximate_svd` is highly effective for scenarios where computational speed is more critical than exact precision. Below are some recommended use cases:

- **Image Compression:** For applications requiring image storage or transmission, `approximate_svd` can reduce data size while maintaining sufficient visual quality, especially for $k \geq 30$.
- **Dimensionality Reduction:** In machine learning or data preprocessing, `approximate_svd` can quickly reduce the dimensionality of large datasets while preserving essential features.
- **Real-Time Applications:** Given its computational efficiency, `approximate_svd` is ideal for tasks requiring real-time processing, such as live video analysis or adaptive streaming.
- **Exploratory Analysis:** For exploratory tasks where exact accuracy is not crucial, such as early-stage data visualization, approximate SVD can provide fast and meaningful results.