

CENG 371 - Scientific Computing  
Fall' 2024 - 2025  
Homework 4

Karaçanta, Kaan  
e244854@metu.edu.tr

## Question 2

1. Plots of the relative errors for both images:

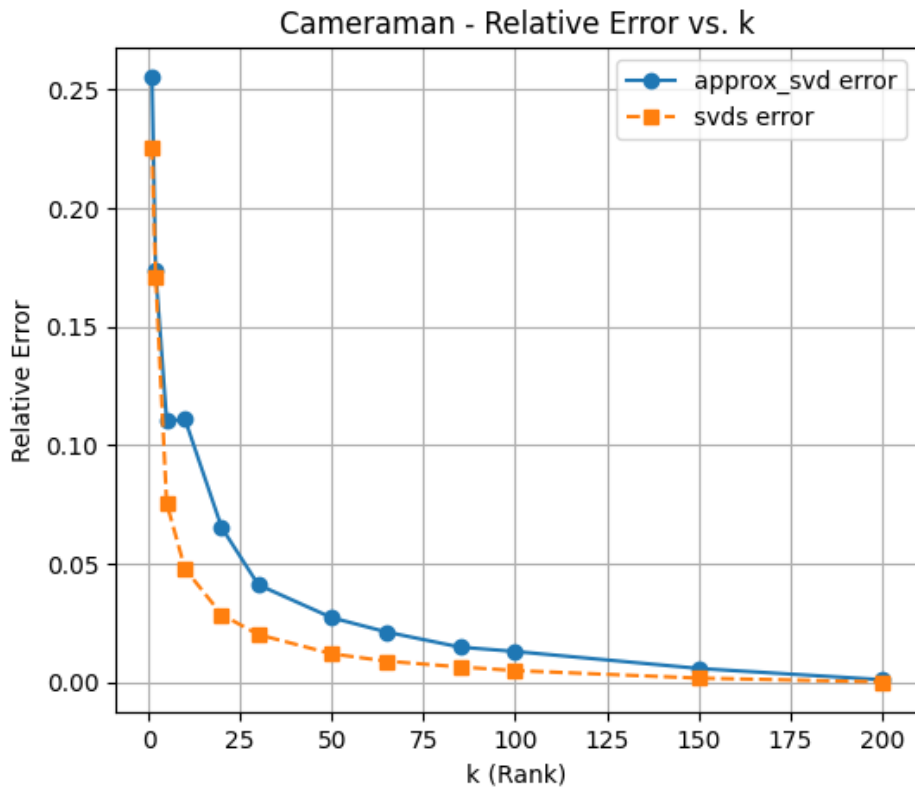


Figure 1: Cameraman – Relative Error vs.  $k$ .

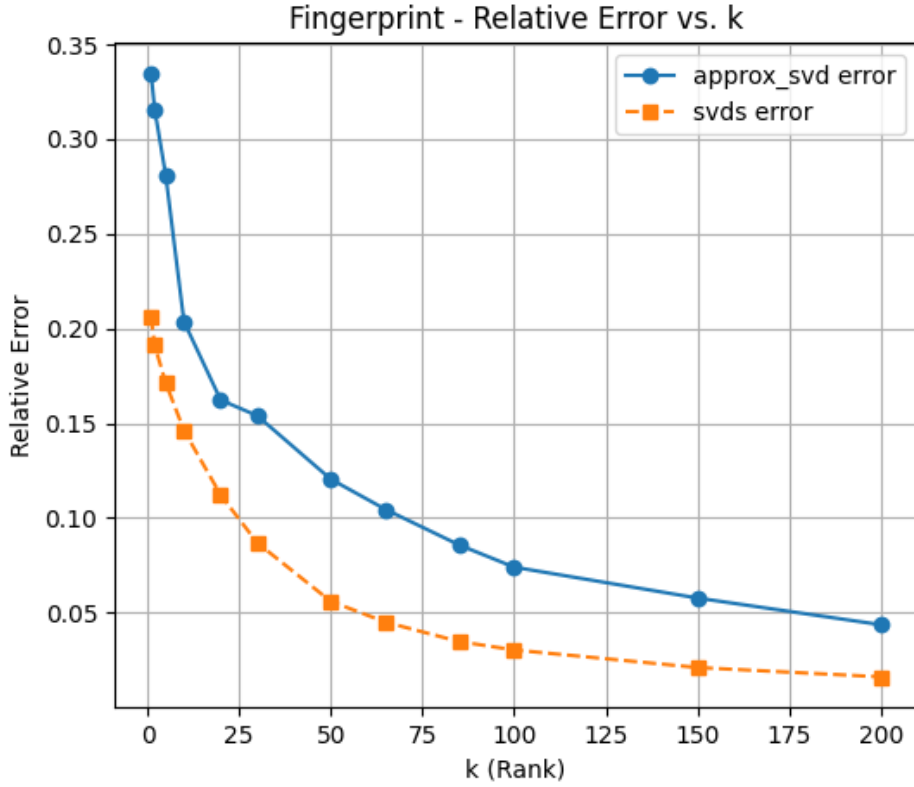


Figure 2: Fingerprint – Relative Error vs.  $k$ .

Let  $A$  be your image matrix (e.g., using `A = io.imread('cameraman.jpg')` in Python). We define:

$$\text{RelErr}_{\text{approx}}(k) = \frac{\|u_k \sigma_k v_k^T - U \Sigma V^T\|_2}{\|U \Sigma V^T\|_2}, \quad \text{RelErr}_{\text{svds}}(k) = \frac{\|u'_k \sigma'_k v_k'^T - U \Sigma V^T\|_2}{\|U \Sigma V^T\|_2},$$

where  $(U, \Sigma, V)$  is the full SVD of  $A$ ,  $(u_k, \sigma_k, v_k)$  is from `approximate_svd`, and  $(u'_k, \sigma'_k, v_k')$  from the built-in `svds`. Figures 1 and 2 illustrate these relative errors for `cameraman.jpg` and `fingerprint.jpg` respectively, plotted against various ranks  $k$ .

#### Observations for Cameraman.

- Both methods continue to show decreasing error as  $k$  increases, with `svds` consistently yielding lower error at the same  $k$  (as expected for exact/truncated SVD methods).
- At very small  $k$  (e.g., until 10), `approximate_svd` has errors around 25–15%, while `svds` is around 20–10%. The gap is very small for  $k = 1$  or  $k = 2$ , then there is a noticeable one but narrows rapidly as  $k$  grows.
- Around  $k = 50$ – $100$ , `approximate_svd` converges closer, with errors often near or below 3%. By  $k = 125$ – $200$ , the error for both methods becomes extremely small (near or below 1%).
- In some tests, when the  $k$  is 1 or 2, the `approximate_svd` error can even slightly cross below `svds` due to numerical and floating-point nuances (though typically `svds` remains the lower bound in theory).

#### Observations for Fingerprint.

- The fingerprint image remains more challenging. For  $k = 5$  or  $10$ , `approximate_svd` can show errors around 35–25%, whereas `svds` is around 25–15%.
- As  $k$  increases, both curves decrease steadily. `svds` remains below the randomized approach, but `approximate_svd` approaches it more closely by  $k > 100$ .
- By  $k = 200$ , `approximate_svd` is under 5% error, while `svds` can reach near 0–1%.
- The gap is still more pronounced than in the cameraman image, reflecting the fingerprint's possibly higher effective rank or more complex structure.

2. Plots of the run times for both images:

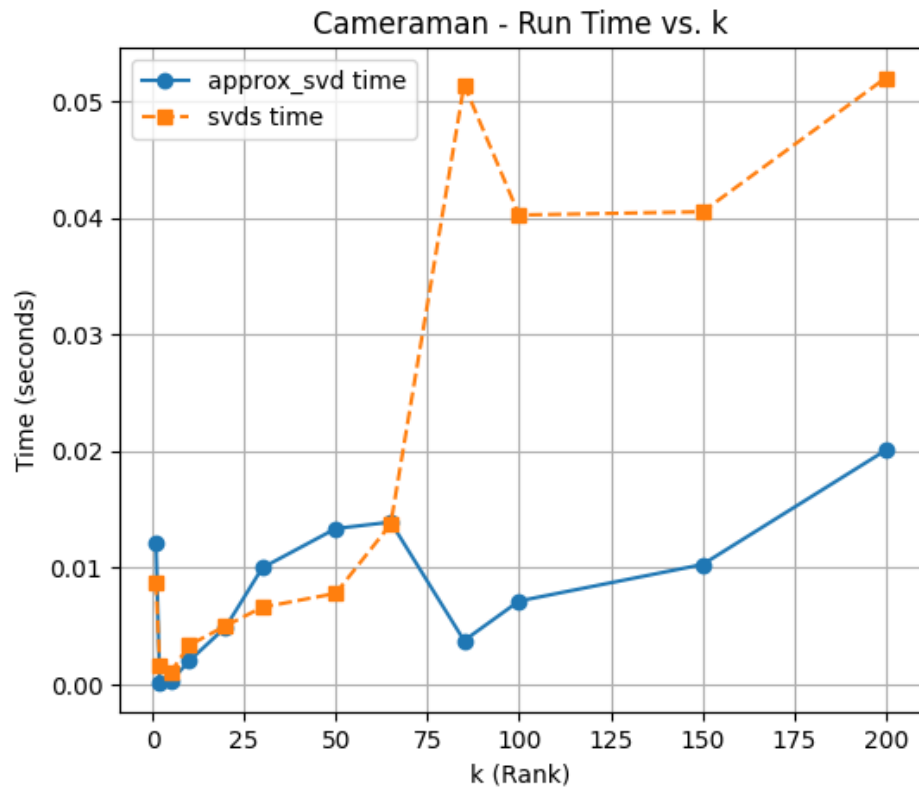


Figure 3: Cameraman – Run Time vs.  $k$ .

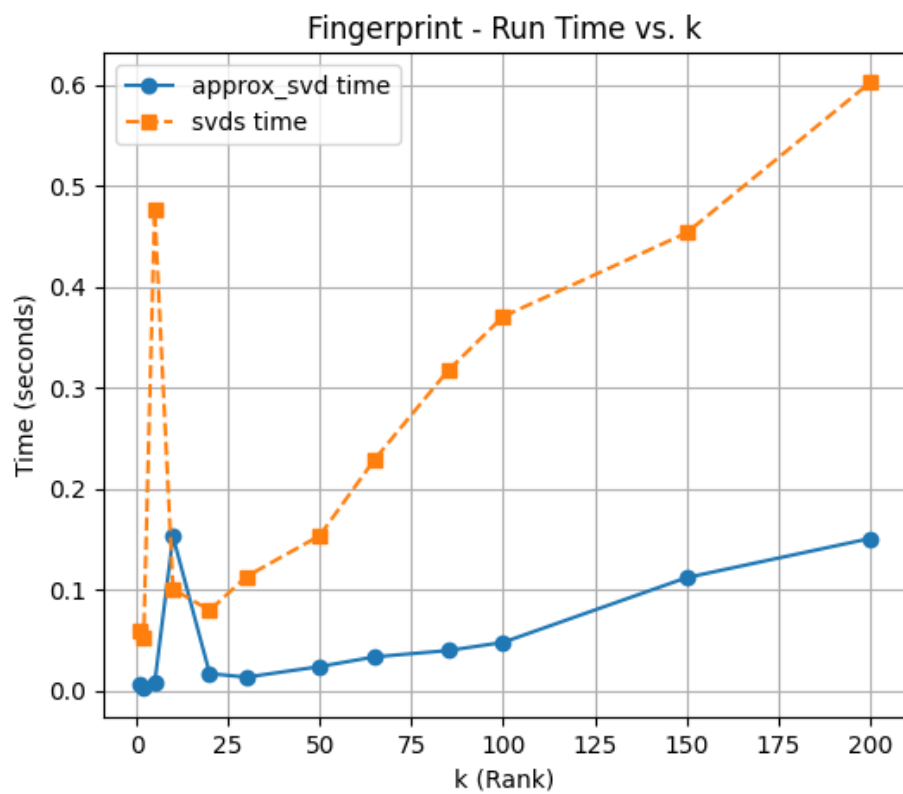


Figure 4: Fingerprint – Run Time vs.  $k$ .

We measure how long each method takes (wall-clock time) for each rank  $k$ . Figures 3 and 4 show run times for the two images.

### Observations for Cameraman.

- For small  $k$  (5–25), both methods can be extremely fast, most of the time under 0.01 s. `svds` might even outperform `approximate_svd` at the smallest ranks, depending on the internal routines used by `svds` (e.g., specialized partial SVD algorithms).
- We see a notable “spike” in the `svds` timing at  $k = 50$ –75. This is possibly where `svds` switches factorization strategies or where iterative methods become more costly. Meanwhile, `approximate_svd` dips slightly or has small bumps near those ranks, possibly due to random draws or QR overhead.
- Beyond  $k \approx 100$ , `svds` climbs more sharply, and by  $k = 200$  may take 0.05–0.06 s, while `approximate_svd` is around 0.01–0.02 s. These results confirm the randomized approach can be beneficial at higher ranks.

### Observations for Fingerprint.

- A significant spike for `approximate_svd` can occur around  $k = 25$  or 50 (sometimes reaching 0.1–0.2 s), while `svds` can have a large jump (0.4–0.5 s) also near those ranks. Such spikes often reflect how iterative solvers or partial factorization techniques change strategies internally.
- After the spike, `svds` grows steadily, reaching up to 0.6 s by  $k = 200$ . `approximate_svd` remains closer to 0.2 s at that rank, indicating a faster growth rate for `svds`.
- Despite random fluctuations, `approximate_svd` consistently maintains an advantage in speed for mid-to-large  $k$  values on this image. Minor irregularities (spikes/dips) can happen with random sampling or specialized BLAS calls.

### 3. Qualitative comparisons.

For selected ranks ( $k = 10, 50, 100, \dots$ ), we can reconstruct the images:

$$\hat{A}_k = u_k \sigma_k v_k^T \quad \text{and} \quad \hat{A}'_k = u'_k \sigma'_k v_k'^T,$$

then display them with `imshow`.

### Discussion points.

- At  $k = 10$ , images are usually quite blurry for both methods; `svds` retains more structure and better (edges, contrasts).
- At  $k = 50$ , the images become much clearer. While a pixel-wise difference might show `svds` is still better, `approximate_svd` is visually close.
- By  $k = 100$  or 150, the randomized approach is nearly indistinguishable, with still some differences, from the exact truncated SVD in normal viewing.

Here you can see the reconstructed images for  $k = 10$  and  $k = 100$  for both images and algorithms:

Approx SVD (Cameraman) Reconstruction, rank=10



(a) Approx SVD,  $k = 10$

svds (Cameraman) Reconstruction, rank=10



(b) svds,  $k = 10$

Approx SVD (Cameraman) Reconstruction, rank=100



(c) Approx SVD,  $k = 100$

svds (Cameraman) Reconstruction, rank=100



(d) svds,  $k = 100$

Figure 5: Reconstructed images for Cameraman at ranks  $k = 10$  and  $k = 100$ .

Approx SVD (Fingerprint) Reconstruction, rank=10



(a) Approx SVD,  $k = 10$

svds (Fingerprint) Reconstruction, rank=10



(b) svds,  $k = 10$

Approx SVD (Fingerprint) Reconstruction, rank=100



(c) Approx SVD,  $k = 100$

svds (Fingerprint) Reconstruction, rank=100



(d) svds,  $k = 100$

Figure 6: Reconstructed images for Fingerprint at ranks  $k = 10$  and  $k = 100$ .

#### 4. Suggested Use Cases for `approximate_svd`.

Based on the observed error trends, run times, and visual reconstructions, we outline some scenarios where `approximate_svd` is particularly valuable:

- **Large-Scale Image / Data Compression:**

For very large matrices (e.g., huge images, big data in machine learning), the randomized SVD often achieves near-optimal compression while being significantly faster than exact methods.

- **Real-Time / Streaming Computations:**

If we need to update or compute truncated SVD in streaming or near-real-time settings, randomized methods avoid the heavy re-factorizations of classical approaches.

- **Exploratory Data Analysis (PCA-like tasks):**

In many practical situations, an approximate subspace is sufficient to capture the main variance of the data, making `approximate_svd` a good choice for large-scale PCA or latent semantic analysis.

- **Machine Learning Pipelines:**

Dimensionality reduction (feature extraction) with approximate SVD saves both time and memory, especially important when dealing with massive training sets.