Software Assignment - Report

EE25BTECH11002 - Achat Parth Kalpesh

Summary of Strang's video

Any real matrix A can can be expressed as the product of three matrices where

$$A \in \mathbb{R}^{m \times n} \tag{0.1}$$

$$A = U\Sigma V^{\top} \tag{0.2}$$

where

$$U \in \mathbb{R}^{m \times m} \tag{0.3}$$

$$\Sigma \in \mathbb{R}^{m \times n} \tag{0.4}$$

$$V \in \mathbb{R}^{n \times n} \tag{0.5}$$

Meaning of each matrix:

- U is an orthogonal matrix whose columns are called the **left singular vectors** of A. These vectors form an orthonormal basis for the column space of A.
  Mathematically,

$$U^{\top}U = I_{mxm} \tag{0.6}$$

- V is an orthogonal matrix whose columns are called the **right singular vectors** of A.
  These vectors form an orthonormal basis for the row space of A.
  Mathematically,

$$V^{\top}V = I_{nxn} \tag{0.7}$$

- Σ is a rectangular diagonal matrix containing the **singular values** $\sigma_1, \sigma_2, \ldots, \sigma_r$ on its diagonal, where

$$r = \text{rank}\,(A) \tag{0.8}$$

  The singular values are always non-negative and arranged in decreasing order:

$$\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_r > 0. \tag{0.9}$$

The SVD is closely related to the eigenvalue decomposition of $A^{\top}A$ and $AA^{\top}$

$$A^{\top}Av_i = \sigma_i^2 v_i \tag{0.10}$$

$$AA^{\top}u_i = \sigma_i^2 u_i \tag{0.11}$$

- The vectors v_i are the eigenvectors of $A^{\top}A$ and form the columns of V.
- The vectors u_i are the eigenvectors of $AA^{\top}$ and form the columns of U.

- The singular values σ_i are the positive square roots of the non-zero eigenvalues of $A^\top A$ or $AA^\top$

Thus

$$Av_i = \sigma_i u_i, \qquad A^\top u_i = \sigma_i v_i. \tag{0.12}$$

If A is square and symmetric,

$$U = V \tag{0.13}$$

and the SVD reduces to the **eigenvalue decomposition**.
The number of non-zero singular values equals the rank of A

Explanation of the implemented algorithm (math + pseudocode)

The algorithm implemented in **C** performs Singular Value Decomposition (SVD) to compress a grayscale image matrix $A \in \mathbb{R}^{m\times n}$.
The algorithm used is **Block Power Iteration** along with **Jacobi Method**
It consists of the following key stages:

- reading the image
- computing dominant singular triplets (U_k, Σ_k, V_k) using an iterative method
- reconstructing the compressed image $A_k = U_k \Sigma_k V_k^\top$

Mathematical steps:

1) Reading the image as a matrix:

$$A = [a_{ij}], \quad a_{ij} \in [0, 255], \quad A \in \mathbb{R}^{m\times n}. \tag{1.1}$$

2) Computing the transpose:

$$A^\top \in \mathbb{R}^{n\times m}. \tag{2.1}$$

3) Initializing $X_0 \in \mathbb{R}^{n\times k}$ as an identity block.
Applying **Block Power Iteration** to estimate the top-k right singular vectors:

$$Y_i = AX_{i-1}, \tag{3.1}$$
$$Z_i = A^\top Y_i, \tag{3.2}$$
$$X_i = \text{Orthonormalize}(Z_i). \tag{3.3}$$

Repeating until convergence or a fixed number of iterations.

4) After convergence, columns of X approximate the eigenvectors of $A^\top A$.
Hence,

$$X \approx V_k \tag{4.1}$$

5) Computing $B = AV_k$ and form the small matrix:

$$C = B^\top B \in \mathbb{R}^{k\times k}. \tag{5.1}$$

The eigen-decomposition of C yields:

$$C = R\Lambda R^\top, \tag{5.2}$$

where $\Lambda = \text{diag}(\sigma_1^2, \ldots, \sigma_k^2)$.

6) Compute

$$U_k = AV_k\Sigma_k^{-1} \tag{6.1}$$

where $\Sigma_k = \text{diag}(\sigma_1, \ldots, \sigma_k)$.

7) Finally, reconstruct the compressed image as:

$$A_k = U_k\Sigma_k V_k^\top \tag{7.1}$$

Pseudocode:
Input: Image A mxn rank k, iterations T
Output: Compressed image A_k
1. Read image A , where A is a matrix with elements of type double
2. Compute AT = transpose(A)
3. Initialize X = identity(n, k)
4. For t = 1 to T:
Y = A * X
Z = AT * Y
Orthonormalize(Z)
X = Z
5. V = X
6. B = A * V
7. C = BT * B
8. Perform Jacobi rotation on C to get eigenvalues and R
9. Sigma matrix = sqrt(diag())
10. U = A * V * inv(Î£)
11. A_k = U * $Sigma$ * $V^\top$
12. Writing A_k as grayscale image

Classical SVD (Golub-Reinsch)

- **Description:** Computes full decomposition using Householder reduction and bidiagonalization.
- **Time Complexity:** $O(mn^2)$ for $m \geq n$
- **Advantages:** Exact, stable, widely used (e.g., LAPACK)
- **Disadvantages:** Computationally expensive for large matrices
- **Accuracy:** Very High

Jacobi SVD

- **Description:** Iteratively diagonalizes A^TA using plane rotations.
- **Time Complexity:** $O(n^3)$

- **Advantages:** Conceptually simple, numerically stable
- **Disadvantages:** Slow convergence for large matrices
- **Accuracy:** Very High

Power Iteration (Single Vector)

- **Description:** Finds only the dominant singular value/vector.
- **Time Complexity:** $O(mn \cdot t)$
- **Advantages:** Easy to implement, memory efficient
- **Disadvantages:** Only finds top singular value
- **Accuracy:** Moderate

Block Power Iteration (Subspace Iteration)

- **Description:** Finds top k singular vectors simultaneously using subspace iteration.
- **Time Complexity:** $O(kmn \cdot t)$
- **Advantages:** Faster convergence, parallelizable, works for multiple singular values
- **Disadvantages:** Requires orthonormalization (extra cost)
- **Accuracy:** High

Randomized SVD

- **Description:** Uses random projections to approximate the dominant subspace.
- **Time Complexity:** $O(mn \log k + k^2(m+n))$
- **Advantages:** Extremely fast for large-scale data
- **Disadvantages:** Accuracy depends on random sampling
- **Accuracy:** Very High

Algorithm choice:
The **Power Block Iteration** method was selected because it efficiently computes the dominant singular vectors without requiring a full SVD decomposition. This is particularly advantageous for large images where direct decomposition is computationally expensive. The Jacobi method was used to diagonalize the small $k \times k$ matrix, ensuring numerical stability and simplicity in implementation.

Key implementation details:

- `stb_image.h` and `stb_image_write.h` were used for reading/writing images.
- Memory was dynamically allocated using `malloc`/`calloc` and freed after use.
- Matrix operations (`transpose`, `multiply`, `orthonormalize`) were implemented from scratch.
- The code ensures numerical safety by checking division thresholds (e.g., $\sigma_i > 10^{-12}$).

Error Analysis: Block Power Iteration + Jacobi

The total error in the computed singular values and vectors comes from two main sources:

- **Block Power Iteration Error:** The block power iteration approximates the top-k singular subspace. The convergence depends on the ratio of singular values:

$$\frac{\sigma_{k+1}}{\sigma_k}$$

The smaller this ratio, the faster the convergence. The error in the computed subspace after t iterations can be bounded as:

$$\| \sin \Theta(X, U_k) \| = O\left(\left(\frac{\sigma_{k+1}}{\sigma_k} \right)^t \right),$$

where X is the computed subspace and U_k is the exact top-k singular vectors.

- **Jacobi Diagonalization Error:** Once the smaller matrix $B = X^T A$ is formed, the Jacobi method is applied to compute its SVD. Jacobi is known to be numerically stable, and the relative error in singular values is bounded by machine precision:

$$\frac{\|\tilde{\Sigma} - \Sigma\|_F}{\|\Sigma\|_F} \approx O(\epsilon_{\text{machine}}).$$

- **Combined Error:** The final error in the reconstructed matrix $A_k = X\tilde{\Sigma}Y^T$ is a combination of the subspace approximation error from the block power iteration and the numerical error from Jacobi:

$$\|A - A_k\|_F \leq \|A - A_k\|_F^{\text{opt}} + O\left(\left(\frac{\sigma_{k+1}}{\sigma_k} \right)^t \|A\|_F \right) + O(\epsilon_{\text{machine}}). \tag{7.2}$$

Here, A_k^{opt} is the best rank-k approximation from exact SVD.

Conclusion: The method achieves high accuracy for the dominant singular values, and errors decrease exponentially with the number of iterations. Jacobi ensures numerical stability in the final step, making the combination reliable for practical computations.

TABLE 7: Error values for different k in three sections

Section	k	Error
Globe	10	15060.93
	50	6185.64
	100	3672.90
Einstein	10	3249.14
	50	880.50
	100	164.78
Greyscale	10	7176.80
	50	1159.88
	100	512.34

Fig. 7.1: Globe

Fig. 7.2: Globe at k = 10

Fig. 7.3: Globe at k = 50

Fig. 7.4: Globe at k = 100

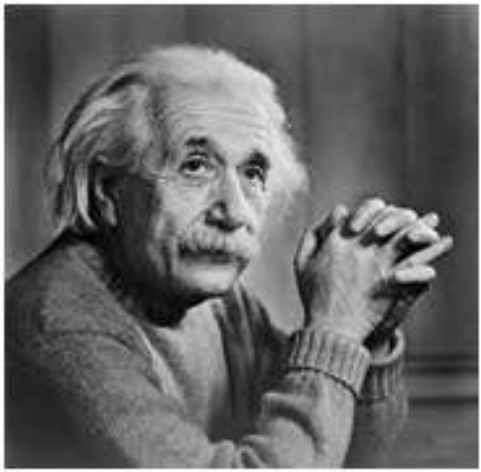

Fig. 7.5: Einstein

Fig. 7.6: Einstein at k = 10

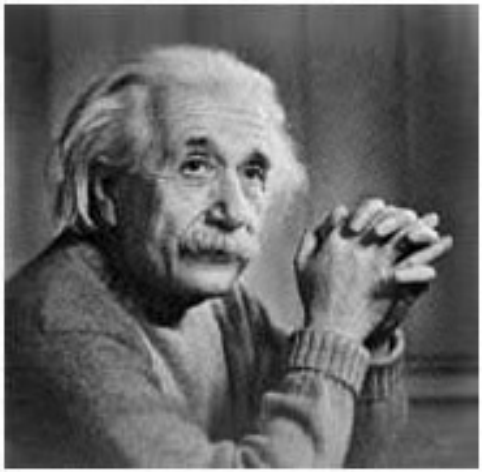

Fig. 7.7: Einstein at k = 50

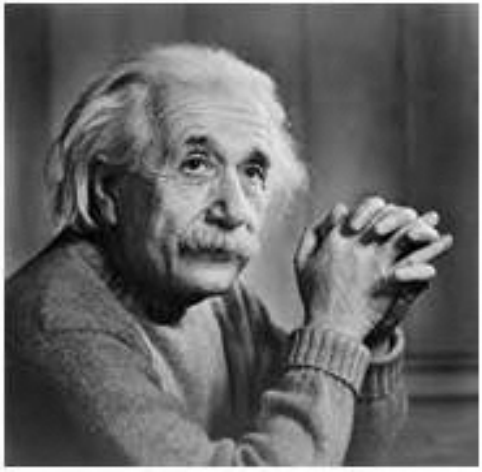

Fig. 7.8: Einstein at k = 100

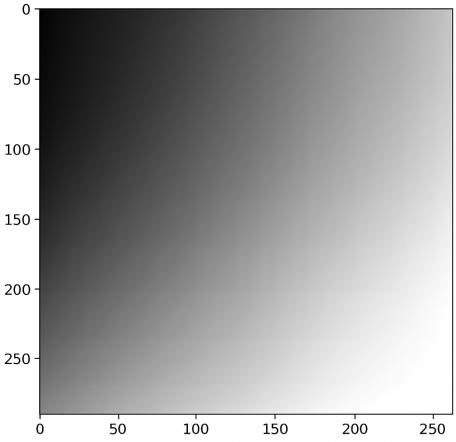

Fig. 7.9: greyscale

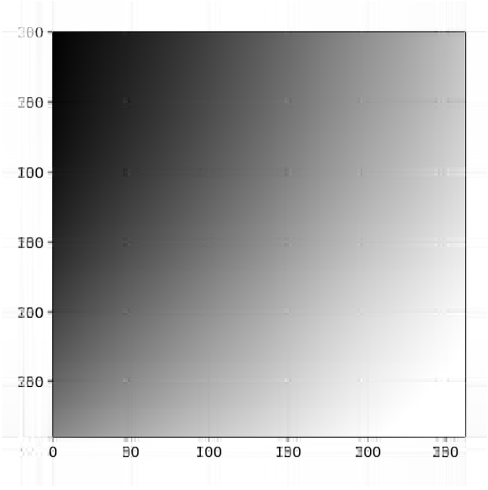

Fig. 7.10: greyscale at k = 10

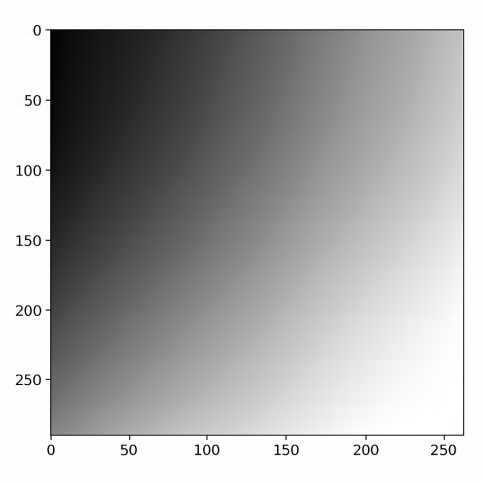

Fig. 7.11: greyscale at k = 50

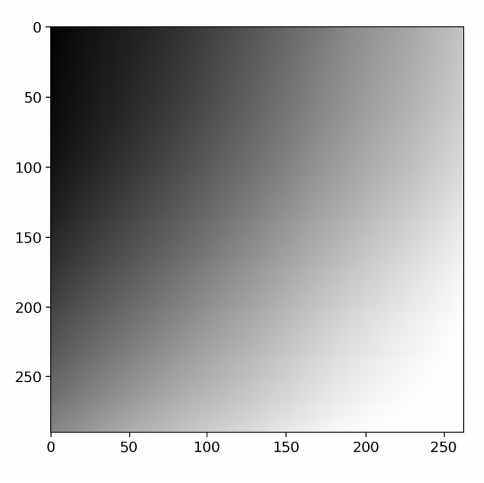

Fig. 7.12: greyscale at k = 100