SOFTWARE ASSIGNMENT-Image Compression using Truncated SVD

ARSH DHOKE
EE25BTECH11010

November 8, 2025

Eigen Values and vectors:

Given a square matrix A, an eigenvalue is a scalar λ such that there exists a nonzero vector v the eigenvector satisfying the equation:

$$Av = \lambda v$$

Singular Value Decomposition:

SVD is a matrix factorization in linear algebra.It states that any real matrix A of size mxn can be decomposed to product of three matrices:

$$A = U\Sigma V^T$$

where U and V are orthogonal matrices containing left and right singular vectors respectively, and Σ is a diagonal matrix containing the singular values arranged in decreasing order.

Jacobi algorithm for SVD:

The Jacobi algorithm is an iterative method used to compute eigenvalues and eigenvectors of a symmetric matrix. It works by repeatedly applying rotations to eliminate off-diagonal elements, gradually transforming the matrix into a diagonal form.It works only for real symmetric matrices.

Time complexity:

The time complexity of the Jacobi algorithm is

$$O(n^3)$$

Limitations:

1. Slow convergence

2. Works only for real and symmetric matrices

3. High computational cost

4. Memory usage is high

Pseudocode:

```
FUNCTION diagonalize(matrixtodiagonalize,eigenvectormatrix,dimension):
matrixtodiagonalize=V
eigenvectormatrix=E
dimension=n in code

SET zerothreshold=1e-12
SET maxpasses=50
Main Loop: Keep sweeping until convergence
FOR passcount from 1 TO maxpasses:

SET rotationsthispass=0   'rotationsdone'
A single "sweep"
Iterate through all upper-triangle pairs (p, q)
FOR p from 0 TO dimension-1:
FOR q from p+1 TO dimension-1:

If the element (p,q) is already small, skip it
IF absolutevalue(matrixtodiagonalize[p][q])<zerothreshold
THEN:
CONTINUE  Go to the next (q) pair
ENDIF

If not small, a rotation is needed
rotationsthispass=rotationsthispass+1

1.Calculate Rotation Parameters(cosine, sine)
SET twoApq=2.0*matrixtodiagonalize[p][q]
SET tau=(matrixtodiagonalize[p][p]-matrixtodiagonalize[q][q])/twoApq

SET tausquaredplusone=1.0+tau*tau
IF tau>=0 THEN:
SET tantheta=1.0/(tau+sqrt(tausquaredplusone))
ELSE:
SET tantheta=1.0/(tau-sqrt(tausquaredplusone))
ENDIF
SET cosine=1.0/sqrt(1.0+tantheta*tantheta)
SET sine=tantheta*cosine

2.Store old pivot values
SET oldApp=matrixtodiagonalize[p][p]
SET oldAqq=matrixtodiagonalize[q][q]
SET oldApq=matrixtodiagonalize[p][q]
```

```
3. Update the matrixtodiagonalize (V)
Update the diagonal pivot elements
matrixtodiagonalize[p][p]=oldApp*cosine*cosine+oldAqq*sine*sine +2*oldApq*sine*cosine

matrixtodiagonalize[q][q]=oldApp*sine*sine+oldAqq*cosine*cosine -2*oldApq*sine*cosine

matrixtodiagonalize[p][q]=0.0  Zero out the off-diagonal element
matrixtodiagonalize[q][p]=0.0

4. Update off-diagonals (V) and eigenvectors (E) in one pass
FOR i from 0 TO dimension-1:  'i' is  'k' in code

Update V
IF i not equal p AND i not equal q THEN:
SET oldAip=matrixtodiagonalize[i][p]   'x'
SET oldAiq=matrixtodiagonalize[i][q]   'y'

matrixtodiagonalize[i][p]=cosine*oldAip+sine*oldAiq
matrixtodiagonalize[p][i]=matrixtodiagonalize[i][p] to maintain symmetry

matrixtodiagonalize[i][q]=-sine*oldAip+cosine*oldAiq
matrixtodiagonalize[q][i]=matrixtodiagonalize[i][q] to maintain symmetry
ENDIF

Update E
SET oldEip=eigenvectormatrix[i][p]   'a1'
SET oldEiq=eigenvectormatrix[i][q]   'a2'

eigenvectormatrix[i][p]=cosine*oldEip+sine*oldEiq
eigenvectormatrix[i][q]=-sine*oldEip+cosine*oldEiq

ENDFOR

ENDFOR  next q
ENDFOR  next p

Convergence Check
If we did a full sweep with no rotations, we are done.
IF rotationsthispass=0 THEN:
BREAK  Exit the main loop
ENDIF

ENDFOR  next pass
END FUNCTION
```

CODE EXPLANATION:

I am using cyclic jacobi algorithm, it parses through every off diagonal element in the upper triangular part of the matrix to diagonalize.Parsing and covering all elements in this manner is a SWEEP.

If an element is found to be smaller than tolerance(1e-12),the iteration is skipped.If not then rotate the matrix using rotation parameters.Modify matrix V(diagonal matrix with entries eigen values(not the one in theory)) and E(matrix with eigenvectors as columns).

V is modified using $V_n = R^T V_o R$ where R is the rotation matrix, E is modified using $E_n = E_o R$

$$R(\theta) = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix}$$

All other entries in rotation matrix are filled in correspondence to the identity matrix.

If a sweep goes without any rotations STOP, the matrix V has been diagonalized and E has the required eigen vectors.

Summary of Strang's Video:

The lecture introduces the Singular Value Decomposition (SVD) as the final and best factorization for any given matrix, A. The SVD breaks any matrix into three components: $A = U\Sigma V^T$, where U and V are orthogonal matrices and σ is a diagonal matrix of singular values.

The central goal of SVD is to find a special, orthonormal basis in the matrix's row space (the vectors in V) that is precisely transformed into a new orthonormal basis in the column space (the vectors in U).The video provides a clear, computational method for finding these components.

The V matrix is found by calculating the eigenvectors of the symmetric matrix $A^T A$. The singular values (σ) in the Σ matrix are the square roots of the eigenvalues of that same $A^T A$ matrix. Similarly, the U matrix is found by calculating the eigenvectors of AA^T.

The SVD is presented as a powerful and fundamental concept because it effectively diagonalizes any linear transformation. It achieves this by identifying the perfect orthonormal bases for all four of a matrix's fundamental subspaces (row space, column space, null space, and left null space), which reveals the true geometry of the transformation.

Test images:

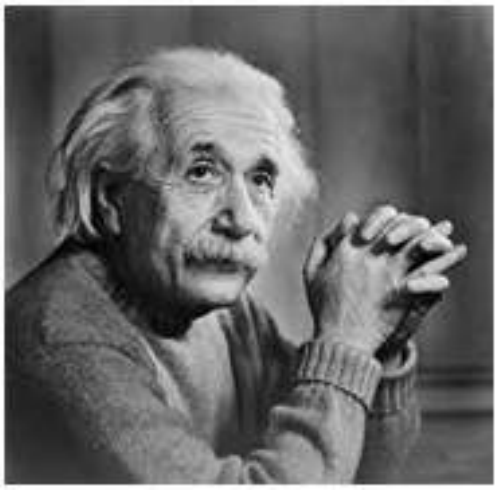
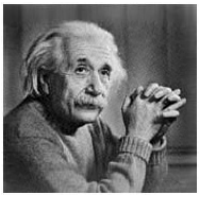


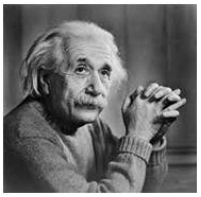
Figure 1: Original Image

k = 5

k = 20



k = 50



k = 100

Figure 2: Original and reconstructed images for different values of k

Figure 3: Original Image

k = 5

k = 20

k = 50

k = 200

Figure 4: Original and reconstructed images for different values of k

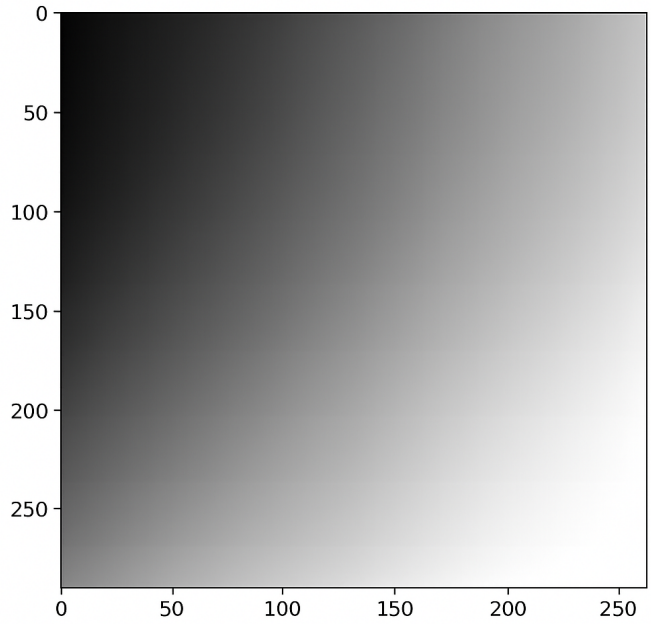
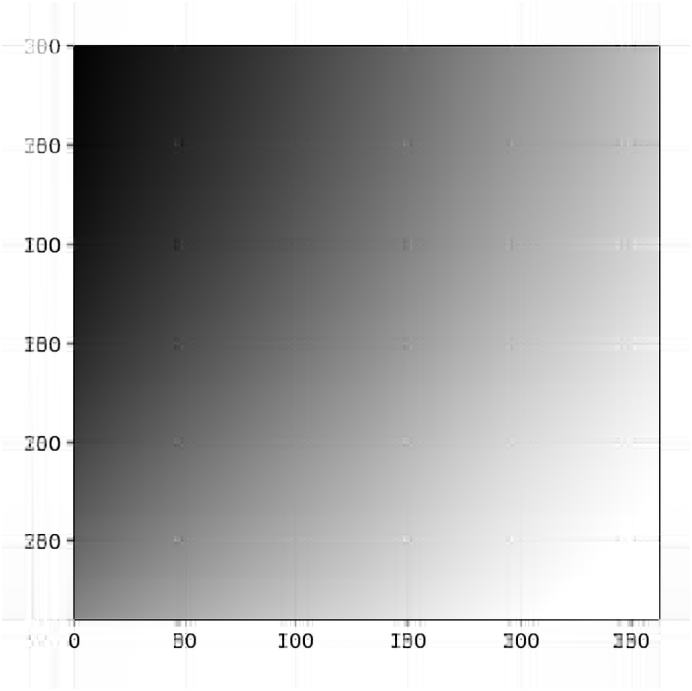
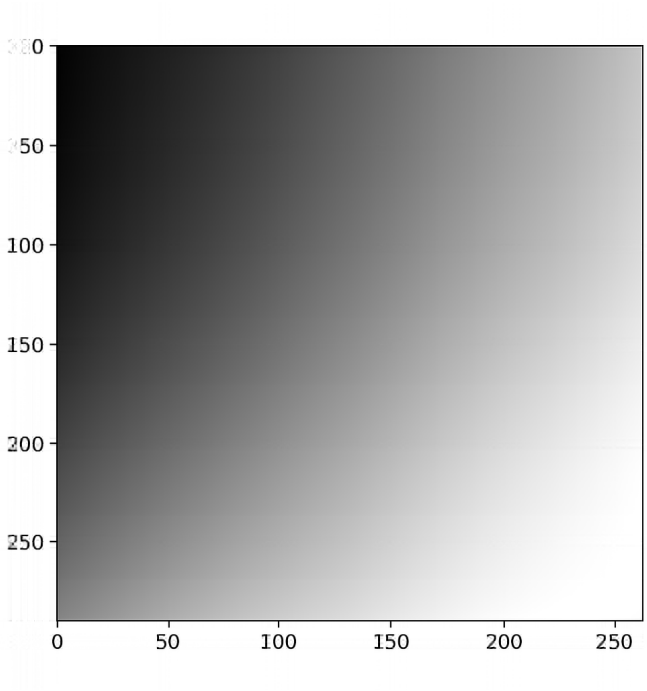


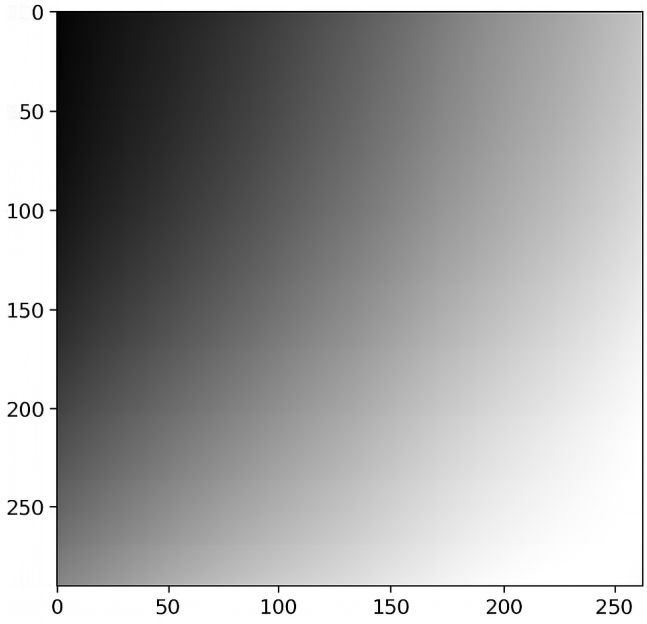
Figure 5: Original Image



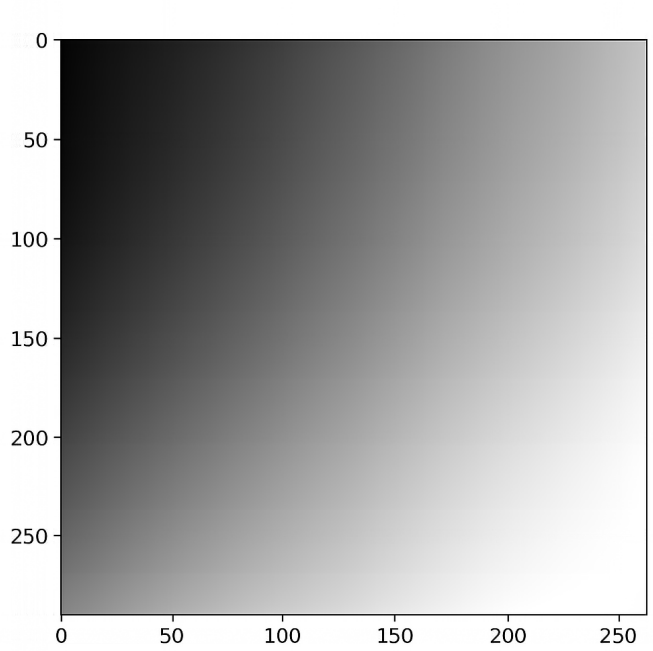
k = 10


0
50
100
150
200
250
0
50
100
150
200
250

k = 20


0
50
100
150
200
250
0
50
100
150
200
250

k = 50



k = 100

Figure 6: Original and reconstructed images for different values of k

Comparison of different algorithms:

1.Jacobi Algorithm

Uses repeated plane rotations to make the off diagonal elements zero. Works for symmetric matrices. Very stable and produces singular values and eigen vectors with high accuracy. Efficient when accuracy is more important than speed.

2.Eigen decomposition method

Computes A^TA and then performs eigenvalue decomposition. Conceptually simple and easy to implement and suitable for beginners. Efficient for small to medium sized matrices.

3.Lancoz method

An iterative technique which works in Krylov subspace. It does not compute full SVD but very efficient when only top k singular values are needed. Ideal for very large and sparse matrices.

4.Power iteration method

An iterative algorithm to find largest eigenvalue and its corresponding eigenvector. Efficient when memory is limited and requirement is only of the top singular value.

5.Golub Reinsch algorithm

Standard SVD algorithm used in MATLAB,NumPy and LAPACK. It reduces the matrix to bidiagonal form and then applies QR iteration. Efficient when full SVD is required and high performance

libraries are allowed.

Algorithm	Time Complexity
Eigen Decomposition	$O(n^3)$
Jacobi algorithm	$O(n^3)$
Golub–Reinsch algorithm	$O(n^3)$
Lanczos Method	$O(k\,n^2)$
Power Iteration	$O(n^2)$ per iteration

Table 1: Time Complexity Comparison of SVD-related Algorithms

Why did I choose Jacobi?
I chose jacobi beacuse its simple, conceptually clear and guarantees convergence for symmetric matrices. It is easy to implement and appropriate for learning purpose and has high accuracy for real symmetric matrices.

Error analysis:

There are two types of error calculated in my code:

1.Frobenius error

$$\|A - A_k\|_F = \sqrt{\sum_{i=1}^{m}\sum_{j=1}^{n}(A_{ij} - (A_k)_{ij})^2}$$

2.Theoretical error

$$\|A - A_k\|_F = \sqrt{\sum_{i=k}^{n}\sigma_i^2}$$

Discussion of Trade-off
A high k results in high image quality but low compression while a low k results in low image quality but high compression.

The Frobenius error will be high for low k values and low for high values of k.

Compression ratio:
The formula for compression ratio for an image is given by:

$$\text{Compression Ratio} = \frac{m \times n}{k(m+n+1)}$$

Conclusion:

In this assignment, I implemented Singular Value Decomposition for image compression using the Jacobi algorithm.

Through experimentation with different values of k, I observed how the reconstructed image quality improves as k is increased, while the compression ratio decreases. Jacobi proved to be accurate and conceptually straightforward, making it suitable for learning-based implementation, even though faster algorithms exist for large-scale applications.

Overall, the project highlights how SVD provides an efficient method for image compression.