Linear Algebra and Applications
Homework #05

Submitted By: Rifat Bin Rashid

RUID: 237000174

Date: Apr 4, 2025

(a)

SVD of A:

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

V: $m \times d$ matrix

Σ: $m \times m$ orthogonal matrix

U: $m \times d$ diagonal matrix

U^T: $d \times d$ orthogonal matrix

So,
$$A^T A = (V \Sigma U^T)^T (V \Sigma U^T)$$
$$= U \Sigma^T V^T V \Sigma U^T$$

But, V is orthogonal, so $V^T V = I$

So,
$$A^T A = U \Sigma^T \Sigma U^T$$

Here,

Σ = ~~dxd~~ $m \times d$ diagonal matrix

Σ^T = $d \times m$ ~~diagonal~~ matrix

So, $\Sigma^T \Sigma$ = $d \times d$ diagonal matrix with diagonal entries σ_i^2 (& zeroes when $i > r$)

let, $\Lambda = \Sigma^T\Sigma = \text{diag}(\underbrace{\sigma_1^2, \sigma_2^2 \ldots \sigma_r^2}_{r \text{ entries}}, \underbrace{0, 0, 0}_{d-r \text{ entries}})$

Thus,

$$A^TA = U \Lambda U^T$$

So, U diagonalizes A^TA, so it's columns are eigenvectors of A^TA (with corresponding eigenvalues σ_i^2)

And since U is also orthogonal, it forms an orthonormal basis for ~~$\mathbb{R}^d$~~ A^TA in $\mathbb{R}^d$

Similarly,

$$AA^T = (V\Sigma U^T)(V\Sigma U^T)^T$$

$$= V\Sigma U^T \cancel{U} \Sigma^T V^T$$

$$= V \Sigma\Sigma^T V^T \qquad (U^TU = I)$$

Here $\Sigma\Sigma^T$ = $m \times m$ diagonal matrix with diagonal entries $\sigma_1^2, \sigma_2^2 \ldots \sigma_r^2$ & additional zeroes

So, $AA^T = V \operatorname{diag}(\underbrace{\sigma_1^2, \sigma_2^2, \dots \sigma_r^2}_{r \text{ entries}}, \underbrace{0, 0, 0}_{m-r \text{ entries}}) V^T$

So, as in case of U, the columns of V are an orthonormal basis of AA^T in $\mathbb{R}^d$.

(b)

from 'a':

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

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

let,

$$U^T A^T A U = \Sigma^2 = \begin{bmatrix} \Sigma_1^2 & 0 \\ 0 & 0 \end{bmatrix} \quad \text{--- (1)}$$

where, $\Sigma_1^2 = \begin{bmatrix} \sigma_1^2 & & 0 \\ & \sigma_2^2 & \\ 0 & & \sigma_r^2 \end{bmatrix}$ is a $r \times r$ diagonal matrix

Now, U is an orthogonal matrix

$$U = [\underline{u}_1, \underline{u}_2 \; .. \; \underline{u}_d]$$

$\underline{u}_i$ is an eigenvector for the i'th eigenvalue of A^TA.

Let,
$$U = [U_1, \; U_2]$$

where, $U_1 = [\underline{u}_1, \underline{u}_2 \; ... \; \underline{u}_r]$ $d \times r$ matrix

$U_2 = [\underline{u}_{r+1} \; ... \; \underline{u}_d]$ $d \times (d-r)$ matrix

So,
$$U^TA^TAU = \begin{bmatrix} U_1^T \\ U_2^T \end{bmatrix} A^TA [U_1 \; U_2] \quad \text{①}$$

$$U^TA^TAU = \begin{bmatrix} U_1^TA^TAU_1 & U_1^TA^TAU_2 \\ U_2^TA^TAU_1 & U_2^TA^TAU_2 \end{bmatrix} \quad .. \text{②}$$

From ① & ②

$$\begin{bmatrix} \Sigma_1^2 & 0 \\ 0 & 0 \end{bmatrix} = \begin{bmatrix} U_1^T A^T A U_1 & U_1^T A^T A U_2 \\ U_2^T A^T A U_1 & U_2^T A^T A U_2 \end{bmatrix}$$

so,

$$\boxed{U_1^T A^T A U_1 = \Sigma_1^2}$$

Now,

since Σ_1 is invertible,

so,

$$\Sigma_1^{-1} U_1^T A^T \; A U_1 \Sigma_1^{-1} = I_{r \times r} \quad \cdots ③$$

let,

$$V_1 = A U_1 \Sigma_1^{-1}$$

so $V_1^T = \Sigma_1^{-1} U_1^T A^T$

where V_1 is an $m \times r$ matrix and ③ asserts that it has orthogonal column.

let, $V_1 = [\underline{v}_1 \quad \underline{v}_2 \quad \cdots \quad \underline{v}_r]$

and define $V = [V_1 \quad V_2]$

with $V_2 = [\underline{v}_{r+1} \quad \cdots \quad \underline{v}_m]$

Using Gram Schmidt, we can find V_2 so that $\{\underline{v}_1 \ \underline{v}_2 \ .. \ \underline{v}_m\}$ form an orthonormal basis in $\mathbb{R}^m$.

Now,

$$V^T A U = \begin{bmatrix} V_1^T \\ V_2^T \end{bmatrix} A [U_1 \quad U_2]$$

$$= \begin{bmatrix} V_1^T A U_1 & V_1^T A U_2 \\ V_2^T A U_1 & V_2^T A U_2 \end{bmatrix}$$

Now, V_2 are the eigenvector corresponding to '0' eigenvalues of $A^T A$

So, $A^T A U_2 = 0$

$A U_2 = 0$

So,

$$V_1^T A U_2 = 0$$

$$V_2^T A U_2 = 0$$

And the columns of V_2 are orthogonal to those of V_1, so the columns of AU_1 are orthogonal to those of V_2, because $\left(V_1 = AU_1\Sigma_1^{-1}\right)$.

So,

$$V_2^T A U_1 = 0$$

r×r matrix

So,

$$V^T A U = \begin{bmatrix} V_1^T A U_1 & 0 \\ 0 & 0 \end{bmatrix}$$

From (3) $\rightarrow$

$$V_1^T A U_1 \Sigma_1^{-1} = I_{r\times r}$$

So,

$$\boxed{\begin{aligned} & V_1^T A U_1 = \Sigma_1 \\ \Rightarrow\ & V_1 = A U_1 \Sigma_1^{-1} \end{aligned}}$$

Problem C:

Strategy : At first, an input matrix A is taken. Then the following steps are followed:

Step 1 : Form A^TA

First, I am forming $A^TA=A^T*A$. This will be a d×d symmetric matrix.

Step 2: Compute spectral decomposition of A^TA

Since A^TA is symmetric, it will have a spectral decomposition. Here, I have used the ‘eigh’ function of the python library to calculate the eigenvalues.

Step 3: Sort eigenvalues (and corresponding eigenvectors) in descending order
Sorting is not mandatory in this problem, it was done as they will be necessary in later steps.
Step 4: Form $\Sigma^T\Sigma$

$\Sigma^T\Sigma$ is the diagonal matrix with eigenvalues.

Step 5: Determine rank r

‘r’ is the number of positive eigenvalues above tolerance. Here, a tolerance value of 10^{-10} was considered, as in numerical computations, due to floating-point round-off errors, eigenvalues that are theoretically zero may appear as very small nonzero numbers. The tolerance helps us decide which eigenvalues are effectively zero.

Step 6: Calculate U_1,V_1 and Σ_1

U_1 is the cropped version of U corresponding to the non-zero eigenvalues.
Σ_1 is the r*r diagonal matrix with all the positive singular values on the diagonal.

V_1 is calculated from :

$$\mathbf{A\ U_1 = V_1\ \Sigma_1}$$

$$\Rightarrow\ \mathbf{V_1 = A\ U_1\ \Sigma_1^{-1}}$$

Full Code:

```
import numpy as np

def svd_via_ata(A, tol=1e-10):
   # Form A^T A
   ATA = A.T @ A
   # Compute spectral decomposition of ATA (ATA is symmetric)
   eigenvalues, U = np.linalg.eigh(ATA)
   # Sort eigenvalues (and corresponding eigenvectors) in descending
order
   idx = np.argsort(eigenvalues)[::-1]
   eigenvalues = eigenvalues[idx]
   U = U[:, idx]

   # Form Σ^TΣ (diagonal matrix with eigenvalues)
   SigmaT_Sigma = np.diag(eigenvalues)

   # Determine rank r (number of positive eigenvalues above
tolerance)
   r = np.sum(eigenvalues > tol)
   U1 = U[:, :r]
   singular_values = np.sqrt(eigenvalues[:r])
   Sigma1 = np.diag(singular_values)

   # Compute V1 using: A U1 = V1 Σ1  =>  V1 = A U1 Σ1^{-1}
   V1 = A @ U1 @ np.linalg.inv(Sigma1)
```

```
   return U, SigmaT_Sigma, U1, Sigma1, V1

def input_matrix():

   m = int(input("Enter the number of rows of matrix A: "))
   n = int(input("Enter the number of columns of matrix A: "))

   A = []
   for i in range(m):
       row_input = input(f"Enter row {i+1} (separate elements by spaces): ")
       # Convert the input string into a list of floats. Change float to int if desired.
       row = list(map(float, row_input.split()))

       # Check if the number of entries matches the expected number of columns
       while len(row) != n:
           print(f"Row {i+1} must have {n} elements. Please try again.")
           row_input = input(f"Enter row {i+1} (separate elements by spaces): ")
           row = list(map(float, row_input.split()))
       A.append(row)

   return np.array(A)



def main():
   # Ask for matrix A row by row.
   print("Enter matrix A row by row.")
   A = input_matrix()
   print("\nMatrix A:")
   print(A)
```

```python
    # Compute the SVD factors via A^T A method
    U, SigmaT_Sigma, U1, Sigma1, V1 = svd_via_ata(A)

    # Set print options for clarity
    np.set_printoptions(precision=4, suppress=True)
    print("\nResults:")
    print("\nU (eigenvectors of A^T A):")
    print(U)
    print("\nΣ^TΣ (diagonal matrix of eigenvalues):")
    print(SigmaT_Sigma)
    print("\nU1 (columns corresponding to positive eigenvalues):")
    print(U1)
    print("\nΣ1 (diagonal matrix of singular values):")
    print(Sigma1)
    print("\nV1 (computed as A U1 Σ1^{-1}):")
    print(V1)

if __name__ == "__main__":
    main()
```

Example output:

Matrix A:

[[1. 2. 3. 4.]
 [5. 6. 7. 8.]
 [9. 4. -8. 9.]]

Results:

U (eigenvectors of A^T A):

```
[[-0.5736 -0.1532 -0.649   0.4757]
 [-0.3992  0.2109 -0.3102 -0.8366]
 [ 0.0831  0.9583 -0.1192  0.2461]
 [-0.7104  0.1172  0.6844  0.1148]]
```

Σ^TΣ (diagonal matrix of eigenvalues):

```
[[314.084   0.      0.      0.   ]
 [  0.    130.465   0.      0.   ]
 [  0.      0.      1.451   0.   ]
 [  0.      0.      0.     -0.   ]]
```

U1 (columns corresponding to positive eigenvalues):

```
[[-0.5736 -0.1532 -0.649 ]
 [-0.3992  0.2109 -0.3102]
 [ 0.0831  0.9583 -0.1192]
 [-0.7104  0.1172  0.6844]]
```

Σ1 (diagonal matrix of singular values):

```
[[17.7224  0.      0.    ]
 [ 0.     11.4221  0.    ]
 [ 0.      0.      1.2046]]
```

V1 (computed as A U1 Σ1^{-1}):

```
[[-0.2237  0.3163  0.9219]
 [-0.5849  0.7131 -0.3865]
 [-0.7797 -0.6257  0.0254]]
```

Problem D:

I approached the problem just like the previous one, only difference was that now A is taken from the image files rather than taking it as an input matrix.

Full code:

```
import numpy as np
import os
from tkinter import Tk, filedialog
from scipy.io import loadmat

def load_matrix(file):
   ext = os.path.splitext(file)[1].lower()
   if ext == '.npy':
       return np.load(file)
   elif ext == '.mat':
       mat_data = loadmat(file)
       for key in mat_data:
           if not key.startswith('__'):
               return mat_data[key]
       raise ValueError("No valid variable found in the .mat file.")
   else:
       raise ValueError("Unsupported file type: " + ext)

def svd_via_ata(A, tol=1e-10):
   # Form A^T A
   ATA = A.T @ A
   # Compute spectral decomposition of ATA (ATA is symmetric)
   eigenvalues, U = np.linalg.eigh(ATA)
   # Sort eigenvalues (and corresponding eigenvectors) in descending order
   idx = np.argsort(eigenvalues)[::-1]
   eigenvalues = eigenvalues[idx]
   U = U[:, idx]

   # Form Σ^TΣ (diagonal matrix with eigenvalues)
   SigmaT_Sigma = np.diag(eigenvalues)
```

```python
    # Determine rank r (number of positive eigenvalues above tolerance)
    r = np.sum(eigenvalues > tol)
    U1 = U[:, :r]
    singular_values = np.sqrt(eigenvalues[:r])
    Sigma1 = np.diag(singular_values)

    # Compute V1 using: A U1 = V1 Σ1  =>  V1 = A U1 Σ1^{-1}
    V1 = A @ U1 @ np.linalg.inv(Sigma1)

    return U, SigmaT_Sigma, U1, Sigma1, V1

def save_results(base_name, U1, Sigma1, V1, save_dir):
    # Construct full paths
    path_U1 = os.path.join(save_dir, base_name + "_U1.txt")
    path_Sigma1 = os.path.join(save_dir, base_name + "_Sigma1.txt")
    path_V1 = os.path.join(save_dir, base_name + "_V1.txt")
    # Save matrices to text files
    np.savetxt(path_U1, U1, fmt="%.4f")
    np.savetxt(path_Sigma1, Sigma1, fmt="%.4f")
    np.savetxt(path_V1, V1, fmt="%.4f")
    print(f"Results saved as:\n  {path_U1}\n  {path_Sigma1}\n  {path_V1}")

def main():
    # Set up Tkinter file dialog to select image files (.npy or .mat)
    root = Tk()
    root.withdraw()  # Hide the root window
    file_paths = filedialog.askopenfilenames(
        title="Select image files (.npy or .mat)",
        filetypes=[("NumPy files", "*.npy"), ("MAT files", "*.mat")]
    )

    if not file_paths:
        print("No files selected.")
        return

    # Ask for folder where to save the results
```

```
    save_dir = filedialog.askdirectory(title="Select folder to save output text
files")
    if not save_dir:
        print("No folder selected for saving files. Exiting.")
        return

    for file in file_paths:
        print("\n----------------------------------")
        print(f"Processing file: {file}")
        try:
            A = load_matrix(file)
        except Exception as e:
            print(f"Error loading file {file}:", e)
            continue

        print(f"Matrix shape: {A.shape}")
        # Compute the SVD factors via the A^T A method
        U, SigmaT_Sigma, U1, Sigma1, V1 = svd_via_ata(A)

        # Print the results (optional)
        np.set_printoptions(precision=4, suppress=True)
        print("\nU1 (columns corresponding to positive eigenvalues):")
        print(U1)
        print("\nΣ1 (diagonal matrix of singular values):")
        print(Sigma1)
        print("\nV1 (computed as A U1 Σ1^{-1}):")
        print(V1)

        # Generate a base name for the output files (remove directory and
extension)
        base_name = os.path.splitext(os.path.basename(file))[0]
        save_results(base_name, U1, Sigma1, V1, save_dir)

    print("\nProcessing complete.")

if __name__ == "__main__":
    main()
```

For the **three-28.mat** file:

U1:

0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
-0.0602 -0.0617 0.1603 0.1049 -0.2713 -0.0905 0.0014 -0.0549 0.2607 -0.1646 0.2075 -0.1438 -0.5762 -0.0895 0.3890 -0.2309 -0.2117 -0.2646 -0.0664 0.2198
-0.0940 -0.0994 0.2638 0.1131 -0.3674 -0.0832 -0.0330 -0.0567 0.0729 -0.1077 0.1357 0.0541 0.2410 -0.1183 -0.2075 -0.1363 -0.0644 0.4907 0.4182 0.4011
-0.0908 -0.0993 0.2669 0.0873 -0.3599 -0.0697 -0.0494 -0.0217 0.1461 -0.0320 0.0318 -0.0682 0.0040 0.0525 -0.2574 -0.2091 0.4614 -0.2423 0.1219 -0.5819
-0.0947 -0.1050 0.2886 0.0705 -0.3544 -0.0543 -0.0827 -0.0308 0.0368 0.1255 -0.0071 0.0125 0.2833 0.1970 -0.1359 0.3507 -0.4920 -0.1483 -0.4581 -0.0605
-0.1344 -0.0592 0.3379 0.1151 -0.1016 0.0904 -0.2736 0.3025 -0.2440 -0.0402 -0.2725 0.1889 -0.1453 -0.0986 0.3292 0.3320 0.4278 0.1344 -0.1506 0.1480
-0.2033 0.0058 0.4384 0.0353 0.3235 0.2580 -0.1999 -0.0382 -0.3410 0.2889 0.1451 -0.0767 0.1721 0.1359 0.1879 -0.3899 -0.1687 -0.1741 0.1775 -0.0136
-0.2259 0.0347 0.4229 -0.0220 0.4004 0.2515 0.2858 -0.2829 0.1990 -0.1898 0.0187 0.0309 -0.2384 -0.1919 -0.3525 0.1340 0.0432 0.1398 -0.2210 0.0048
-0.2187 -0.1094 0.1966 -0.1453 0.1708 -0.2688 0.5343 0.4475 0.2542 -0.0352 -0.0808 0.0907 0.1656 0.1885 0.2323 0.1123 -0.0566 -0.1063 0.2597 -0.0348
-0.2531 -0.1836 0.0096 -0.2763 0.0522 -0.4035 -0.0602 -0.0098 -0.3640 -0.0950 -0.0492 -0.6179 -0.1197 -0.1678 -0.0291 0.1038 -0.0929 0.2055 0.0018 -0.1603
-0.2671 -0.1893 -0.0763 -0.3467 0.0397 -0.2921 -0.1355 -0.1830 -0.0957 -0.2016 -0.2378 0.3411 0.0872 0.0799 -0.1844 -0.3021 0.1254 -0.3187 -0.1771 0.3427
-0.2949 -0.2108 -0.1473 -0.3286 -0.0617 0.1121 -0.1609 -0.1132 0.2552 0.2324 0.0077 0.3599 -0.0393 -0.2294 0.2821 -0.0940 -0.1654 0.3483 -0.0329 -0.3778
-0.3114 -0.2053 -0.2052 -0.2083 -0.1021 0.3250 -0.0453 0.0154 0.1809 0.4142 0.1472 -0.2584 -0.1097 0.1429 -0.1690 0.2968 0.2173 -0.1934 0.1748 0.3164
-0.3657 -0.1507 -0.2879 0.1723 -0.1086 0.3521 0.1359 -0.0336 -0.2950 -0.5675 0.2452 0.0767 0.0703 0.2336 0.1087 0.0817 -0.0513 -0.0009 0.0419 -0.1345
-0.3543 -0.0039 -0.2349 0.4422 -0.0262 0.1203 0.1284 0.3031 0.0722 0.1323 -0.3552 -0.1728 0.1179 -0.3696 -0.1279 -0.3208 -0.0813 -0.0371 -0.2019 0.0539
-0.3181 0.1282 -0.1148 0.4805 0.1148 -0.4098 0.0099 -0.3286 -0.0371 0.2855 -0.0549 0.1612 -0.2439 0.3613 0.0096 0.1068 0.0387 0.1545 0.0951 -0.0356
-0.2512 0.3810 -0.0519 0.0560 0.0960 -0.2426 -0.1660 -0.0473 0.0838 -0.0515 0.4547 0.0602 0.3125 -0.4650 0.1092 0.2403 0.1180 -0.2644 0.0165 0.0186
-0.2035 0.4731 -0.0170 -0.1809 0.0359 -0.0052 -0.2997 0.4129 0.2158 -0.1322 0.1569 -0.0936 -0.0928 0.3886 -0.1544 -0.1960 0.0067 0.2695 -0.2193 0.0058
-0.1421 0.4671 0.0327 -0.1526 -0.1660 0.1630 -0.1141 -0.1113 -0.0131 -0.1397 -0.5044 0.0237 -0.1295 -0.0636 -0.0556 0.1724 -0.2842 -0.2004 0.4485 -0.0863
-0.0851 0.3893 0.0365 -0.2233 -0.3324 0.0795 0.4670 -0.3051 -0.1405 0.1595 -0.0583 -0.1484 0.1790 0.0677 0.2952 -0.1474 0.2463 0.1270 -0.2477 0.0883
-0.0187 0.1213 0.0012 -0.1202 -0.1974 -0.0487 0.2724 0.3054 -0.4662 0.2464 0.2760 0.3601 -0.3546 -0.1752 -0.3189 -0.0316 -0.1203 -0.0524 0.0096 -0.0370
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000

V1:

0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
-0.1678 -0.1405 0.3490 -0.2113 0.0528 0.0930 -0.1408 -0.0280 -0.2948 0.3597 -0.0834 0.1395 0.4550 0.1999 0.1053 0.3256 -0.3792 0.0250 -0.0443 0.0167
-0.3588 -0.2596 0.4158 -0.0294 -0.2926 0.0303 0.0055 0.0845 0.0039 -0.1158 -0.1539 -0.0086 0.2401 -0.0667 -0.4136 -0.2480 0.4491 0.0030 0.0465 -0.0259
-0.3359 -0.1899 0.3385 0.2611 -0.4123 -0.1308 -0.0134 -0.0693 0.2081 0.0136 0.1551 -0.0992 -0.4586 0.0579 0.3263 0.1085 -0.2494 -0.0592 -0.0341 0.0109
-0.1449 0.0278 -0.1354 0.4671 -0.1066 -0.1264 0.1194 -0.1357 -0.1975 -0.2999 0.3078 0.2476 0.4117 -0.3498 0.0349 -0.0115 -0.1957 0.2496 0.0387 0.0377
-0.1397 -0.0046 -0.2226 0.4042 -0.0202 0.1141 0.2631 -0.0876 -0.3273 -0.0469 -0.0382 -0.0531 -0.0692 0.6085 -0.2698 0.1720 0.0838 -0.2153 -0.1387 0.1165
-0.1483 -0.0920 -0.2585 0.1192 -0.1271 0.6330 0.1448 0.2396 0.1767 0.3387 0.0487 -0.2650 0.0413 -0.2275 -0.0950 0.0475 -0.1342 0.1009 0.2448 0.1480
-0.1582 -0.1456 -0.2414 -0.1827 -0.1227 0.4380 -0.1419 -0.2165 0.0386 0.0061 0.2066 0.4708 -0.0511 0.0368 0.2636 -0.0410 0.2968 -0.0186 -0.3661 -0.1575
-0.1918 -0.1914 -0.2261 -0.3492 -0.0758 0.0493 -0.1381 -0.2355 -0.4301 -0.3627 -0.0323 -0.3492 -0.1424 -0.0170 0.1058 0.1332 0.0055 0.1507 0.3753 -0.1350
-0.2788 -0.1263 -0.2956 -0.0982 0.0166 -0.3135 0.0354 0.0226 -0.1038 0.1607 -0.1833 -0.2316 0.1265 -0.2846 0.2442 -0.0362 0.1335 -0.2691 -0.3472 0.4650
-0.3116 0.0739 -0.3275 -0.0865 -0.0307 -0.2220 -0.5460 0.0713 0.3052 -0.0226 0.1040 0.0826 -0.0221 0.0983 -0.4776 0.1705 -0.2032 0.0669 -0.0589 0.0377
-0.2032 0.3053 -0.1954 0.1642 -0.1541 -0.0161 -0.0783 -0.0954 0.2271 0.0229 -0.4138 -0.1318 0.3347 0.1820 0.3007 -0.2235 -0.0506 -0.1466 0.1236 -0.4422
-0.1251 0.3738 -0.0539 -0.0079 -0.1938 -0.2534 0.0796 -0.0949 -0.1363 0.4859 0.0154 0.2140 -0.1387 -0.0060 0.0454 0.2041 0.4049 0.2962 0.3243 0.0630
-0.0746 0.3391 0.0037 -0.2298 -0.2608 0.0147 0.1096 0.2265 -0.2910 0.0057 0.2430 0.1388 -0.0792 -0.2138 -0.1468 -0.0582 -0.1787 -0.6186 0.0798 -0.1849
-0.0678 0.3197 0.0074 -0.2302 -0.2565 0.0662 0.1006 0.1766 -0.1923 -0.0620 -0.0636 -0.0587 -0.0971 0.1963 0.0005 -0.4452 -0.2578 0.4773 -0.3268 0.1819
-0.0972 0.3238 0.1028 -0.2346 -0.0740 0.0816 0.3434 0.0695 0.3052 -0.3664 -0.0150 -0.1284 0.1587 -0.0303 0.0408 0.5907 0.1197 0.0742 -0.2108 0.0125
-0.1511 0.3480 0.2287 -0.0605 0.2152 0.1870 -0.0607 -0.6529 0.1252 0.0347 0.2445 -0.2005 0.0487 -0.0045 -0.1004 -0.2126 -0.0218 -0.1453 0.0678 0.2968
-0.2377 0.2831 0.1874 0.2793 0.3155 0.2664 -0.3037 0.1472 -0.2304 -0.1596 -0.4197 0.1522 -0.2925 -0.2801 0.0400 0.1167 0.0283 0.0115 -0.0697 0.0536
-0.3630 0.0446 0.0527 -0.0175 0.4438 -0.0521 -0.0033 0.4621 0.0237 -0.0771 0.4372 -0.0596 0.0811 0.2706 0.2851 -0.1528 0.1767 0.0164 0.1812 0.0017
-0.3439 -0.1175 -0.0425 -0.0834 0.3779 -0.1312 0.4372 -0.1640 0.0149 0.2133 -0.0414 -0.0129 -0.1919 -0.1822 -0.2233 -0.0579 -0.1570 0.1448 -0.2033 -0.4726
-0.1589 -0.1274 -0.0903 -0.2032 0.0737 -0.0143 0.3121 -0.0449 0.2178 -0.1985 -0.3083 0.5126 -0.0781 0.1106 0.0358 -0.1064 -0.2131 -0.0969 0.3918 0.3466
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000

Σ1:

2167.8393	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
0.0000	1268.1906	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
0.0000	0.0000	819.2278	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
0.0000	0.0000	0.0000	640.7616	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
0.0000	0.0000	0.0000	0.0000	494.5187	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
0.0000	0.0000	0.0000	0.0000	0.0000	308.2740	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	215.6092	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	157.8819	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	137.5699	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	116.5039	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	109.1386	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	88.0830	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	83.0957	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	70.5445	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	44.2014	0.0000	0.0000	0.0000	0.0000	0.0000
0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	29.6665	0.0000	0.0000	0.0000	0.0000
0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	20.2864	0.0000	0.0000	0.0000
0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	13.5537	0.0000	0.0000
0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	3.8957	0.0000
0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	1.6331

Output for the other two image files are too large to be shown here, however, they are provided in the shared folder as .txt files.

Problem E:

Here, the results obtained in (d) to calculate the rank-k approximations using this function :

```
def rank_k_approximation(U1, Sigma1, V1, k):

   U1_k = U1[:, :k]                 # (d x k)
   Sigma1_k = Sigma1[:k, :k]     # (k x k)
   V1_k = V1[:, :k]                 # (m x k)
   A_k = V1_k @ Sigma1_k @ U1_k.T
   return A_k
```

This function 'rank_k_approximation' truncates the SVD to its top k components and reconstructs an approximation of A that captures the most significant features of A.

Then the resultant images were plotted along with the original image using this function :

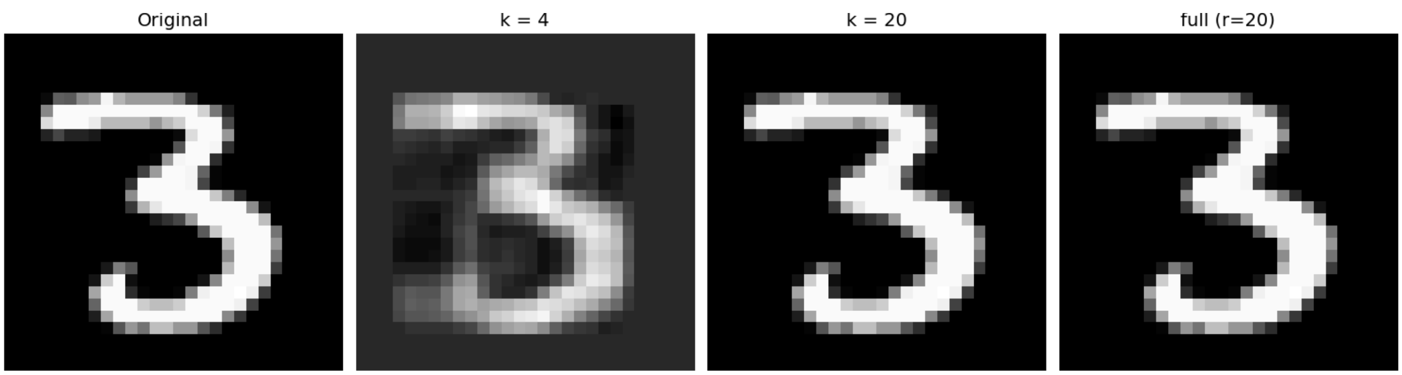
```
def plot_approximations(image_matrices, image_names, ks_list):

   n_images = len(image_matrices)
   n_cols = len(ks_list) + 1

   fig, axes = plt.subplots(n_images, n_cols, figsize=(4*n_cols,
4*n_images))
   if n_images == 1:
       axes = np.expand_dims(axes, axis=0)
```

```
   for i, (A, name) in enumerate(zip(image_matrices,
image_names)):
       # Compute SVD factors via A^T A method for the image A
       U, SigmaT_Sigma, U1, Sigma1, V1 = svd_via_ata(A)
       r = Sigma1.shape[0]  # effective rank
       approximations = {}
       approximations["original"] = A

       for k in ks_list:
           if isinstance(k, str) and k.lower() == "full":
               k_used = r
           else:
               k_used = k if k <= r else r
           approximations[k] = rank_k_approximation(U1, Sigma1,
V1, k_used)

       # Plot: first column is original, then one column for
each k in ks_list.
       col_keys = ["original"] + ks_list
       for j, key in enumerate(col_keys):
           ax = axes[i, j]
           ax.imshow(approximations[key], cmap='gray',
aspect='equal')
           ax.axis('off')
           if i == 0:
               if key == "original":
                   title = "Original"
               elif isinstance(key, str) and key.lower() ==
"full":
                   title = f"full (r={r})"
               else:
                   title = f"k = {key}"
```

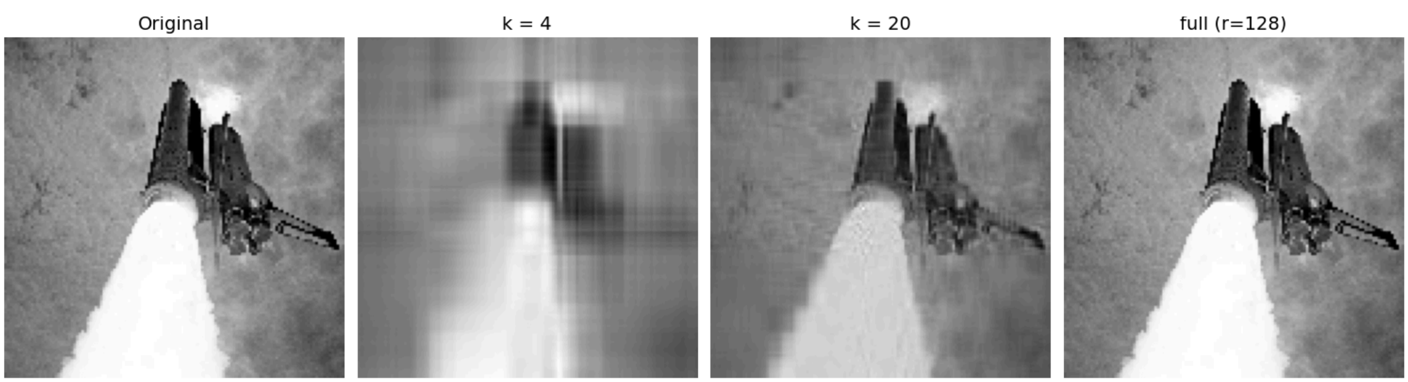
```
                ax.set_title(title, fontsize=14)
            ax.set_ylabel(name, fontsize=12)

plt.tight_layout()
plt.show()
```

Results:

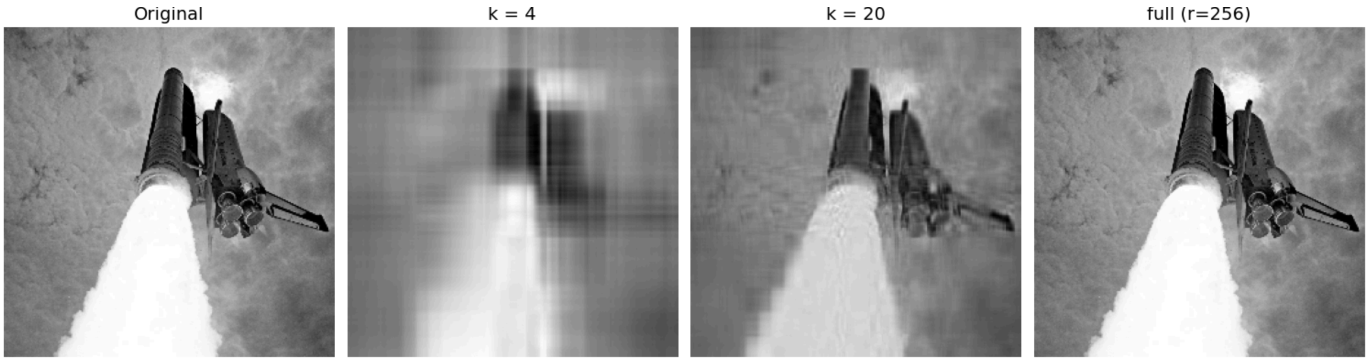
Three-28:



Columbia-128 :



Columbia-256 :



The Files of my solution to the problems can be found at:

HW5_Linear_RBR