Name:	Suhas Kamath	**Assignment No:**	Assignment 2
SR No:	06-18-01-10-51-25-1-25945	**Course Code:**	DS284
Email ID:	suhaskamath@iisc.ac.in	**Course Name:**	Numerical Linear Algebra
Date:	September 26, 2025	**Term:**	AUG 2025

Question 2

Theory

In this question, we have to compress an image using Single Value Decomposition. The original (full-resolution) image is as follows:

This can be done using a concept called a low rank approximation. We know that the SVD of a matrix $\mathbf{A}$ is as follows:

$$\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V^T}$$

In this decomposed format, the components are as follows:

- $\mathbf{U}$: The right singular vectors of the matrix.
- $\mathbf{V^T}$: The left singular vectors of the matrix.
- $\mathbf{\Sigma}$: The singular values of the matrix, in a diagonal format:

$$\mathbf{\Sigma} = \begin{bmatrix} \sigma_1 & 0 & \dots & 0 \\ 0 & \sigma_2 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \sigma_n \end{bmatrix}$$

We also know that $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_n$.

The singular values matrix $\mathbf{\Sigma}$ can also be written as:

$$\mathbf{\Sigma} = \text{sum}_{j=1}^{r} \mathbf{\Sigma_j}$$

Where:

- $r =$ Rank of the matrix $\mathbf{A}$
- $\mathbf{\Sigma_j}[a][b] = \begin{cases} \sigma_j, & a = b = j \\ 0, & \text{otherwise} \end{cases}$, where $[a][b]$ represents the index of the matrix.

We can reconstruct the original matrix $\mathbf{A}$ by multiplying the components of the SVD:

$$\boldsymbol{A} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V^T}$$
$$\boldsymbol{A} = \boldsymbol{U}\left\{\text{sum}_{j=1}^{r} \boldsymbol{\Sigma}_j\right\}\boldsymbol{V^T}$$
$$\boldsymbol{A} = \text{sum}_{j=1}^{r} \left(\boldsymbol{U}\boldsymbol{\Sigma}_j\boldsymbol{V}\right)$$
$$\boldsymbol{A} = \text{sum}_{j=1}^{r} \left(\sigma_j \boldsymbol{u}_j \boldsymbol{v_j^T}\right)$$

Name:	Suhas Kamath	**Assignment No:**	Assignment 2
SR No:	06-18-01-10-51-25-1-25945	**Course Code:**	DS284
Email ID:	suhaskamath@iisc.ac.in	**Course Name:**	Numerical Linear Algebra
Date:	September 26, 2025	**Term:**	AUG 2025

Because we know that $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_n$, we can see that the later σ values (”tail” values) have lesser of an effect on the overall array. Hence, we can ignore the later singular values, and still end up with a decent reconstruction of the image. This can be done by taking a partial sum. The formula for the k^{th} partial sum, $\mathbf{A_k}$ is as follows:

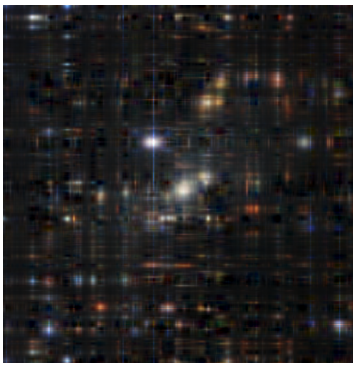
$$\boldsymbol{A_k} = \text{sum}_{j=1}^{k} \left(\sigma_j \boldsymbol{u_j} \boldsymbol{v_j^T}\right)$$

Part (a)

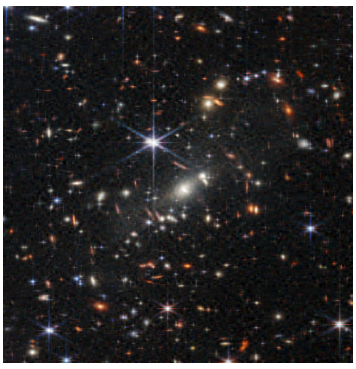
In this section, we have to obtain a value of k where the compressed image will be indistinguishable from the original image. We first split the image into the red, blue and green (RGB) components. The maximum rank of these three arrays is 1960. Hence, we know that the value of k can range from 1 to 1960.

Let us start with $k = 1$. Although this is very less, and will most definitely not give us an indistinguishable image, we can still have a starting point. The image generated at $k = 1$ is as follows:

Let us now slightly increase the value of k. At $k = 10$, we get this image:



We can see that we are getting closer to the original image. Let us try $k = 50$:



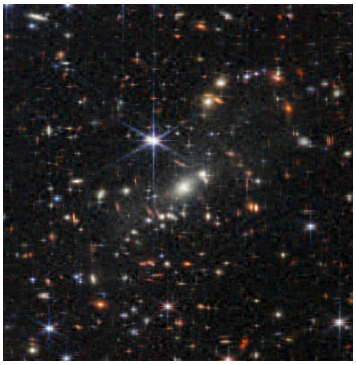
Name:	Suhas Kamath	**Assignment No:**	Assignment 2
SR No:	06-18-01-10-51-25-1-25945	**Course Code:**	DS284
Email ID:	suhaskamath@iisc.ac.in	**Course Name:**	Numerical Linear Algebra
Date:	September 26, 2025	**Term:**	AUG 2025

This image is quite close to the original image.

All the comparisons so far was done using "visual inspection". However, this is not an accurate measure of "indistinguishability". We should be using a more "numeric" system. We can do this by calculating the retained "energy" of the image, and ensuring that this is above an particular ϵ.

$$\text{Energy Retained} = \frac{\sum_{j=1}^{k} \sigma_j^2}{\sum_{j=1}^{r} \sigma_j^2} \geq \epsilon$$

If we take $\epsilon = 90\%$, the minimum value of k is 37. The image generated is as follows:



I believe that a retained energy of 90% is enough for this experiment. However, the code is very flexible and a higher percentage can be chosen at any time.

Part (b)

The total entries in the initial image is as follows:

$$Entries_{initial} = m \times n = 1960 \times 2000 = 3.92 \times 10^6$$

The number of entries in the low-rank approximation of the image is:

$$Entries_{low\_rank} = (m + n + 1) \times k = (1960 + 2000 + 1) \times 37 = 146557 \approx 1.46 \times 10^5$$

Hence, the number of entries that need to be sent back to Earth is $\approx 1.46 \times 10^5$. The number of entries has been reduced by a factor of $\frac{Entries_{initial}}{Entries_{low\_rank}} \approx 27$.

Part (c)

The error for each component is as follows:

Colour	$\|\mathbf{A} - \mathbf{A_k}\|_2$	$\|\mathbf{A} - \mathbf{A_k}\|_F$
Red	4314.38	28680.76
Green	3624.45	24762.16
Blue	3666.71	26069.16

Let us calculate σ_{k+1} and $\sqrt{\sigma_{k+1}^2 + \sigma_{k+2}^2 + \cdots \sigma_r^2}$ for each colour, so that we can verify the theorems given in the question.

Name: Suhas Kamath
SR No: 06-18-01-10-51-25-1-25945
Email ID: suhaskamath@iisc.ac.in
Date: September 26, 2025

Assignment No: Assignment 2
Course Code: DS284
Course Name: Numerical Linear Algebra
Term: AUG 2025

Colour	σ_{k+1}	$\sqrt{\sigma_{k+1}^2 + \sigma_{k+2}^2 + \cdots \sigma_r^2}$
Red	4314.38	28680.76
Green	3624.45	24762.16
Blue	3666.71	26069.16

As can be seen, both theorems hold for the errors in the question.

Algorithm, Code and Output

The algorithm is as follows:

1. Load the image and separate the red, green and blue (RGB) components.
2. Perform a singular vector decomposition on each of the components.
3. Calculate the "energy" retained for every value of k, until we obtain a value of k where $\epsilon = 90\%$ of the energy is retained for all the three components.
4. Perform the partial sum to obtain the low-rank approximation for the three components.
5. Re-construct the image, and then save it to a file on disk.
6. Calculate the norms of the errors (i.e. $\|\mathbf{A} - \mathbf{A_k}\|_2$ and $\|\mathbf{A} - \mathbf{A_k}\|_F$), for each component. Output these norms.
7. To ensure that the theorems are true, calculate the quantities given in the theorem (i.e. σ_{k+1} and $\sqrt{\sigma_{k+1}^2 + \sigma_{k+2}^2 + \cdots \sigma_r^2}$). Output these values.

The Python code which implements the above algorithm is as follows:

```
import cv2
import numpy as np

# Load the image
image = np.asarray(cv2.imread(r"H:\\My Drive\\Numerical Linear
 ↪  Algebra\\Assignments\\Assignment 3\\SMACS_0723.png"))

#Convert BGR to RGB
image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
print("Image loaded successfully.")

#Separate the channels
image_red = image[:,:,0]
image_green = image[:,:,1]
image_blue = image[:,:,2]

#Perform SVD on each channel
print("Performing SVD on each component: ")
U_red, S_red, VT_red = np.linalg.svd(image_red , full_matrices=False)
U_green, S_green, VT_green = np.linalg.svd(image_green , full_matrices=False)
U_blue, S_blue, VT_blue = np.linalg.svd(image_blue , full_matrices=False)

print("SVD Completed.")
print("Starting low rank approximation")

#Calculate energy retained and hence appropriate k
#Energy retained = sum of squares of first k singular values / sum of squares of all
 ↪  singular values
#We need to find the minimum k such that energy retained >= 0.9 for all channels
```

```
epsilon = 0.9

print(f"Calculating energy retained for different values of k to find minimum k for {epsilon*100}% energy retention in all channels.")

#Calculate total energy for each channel
total_energy_red = np.sum(S_red**2)
total_energy_green = np.sum(S_green**2)
total_energy_blue = np.sum(S_blue**2)

for k in range(0, 1961):
    energy_red = np.sum(S_red[:k]**2) / total_energy_red
    energy_green = np.sum(S_green[:k]**2) / total_energy_green
    energy_blue = np.sum(S_blue[:k]**2) / total_energy_blue

    if energy_red >= epsilon and energy_green >= epsilon and energy_blue >= epsilon:
        print(f"Minimum k to retain at least 90% energy in all channels: {k}")
        break

#Low rank approximation
U_red_k = U_red[:, :k]
S_red_k = np.diag(S_red[:k])
VT_red_k = VT_red[:k, :]

U_green_k = U_green[:, :k]
S_green_k = np.diag(S_green[:k])
VT_green_k = VT_green[:k, :]

U_blue_k = U_blue[:, :k]
S_blue_k = np.diag(S_blue[:k])
VT_blue_k = VT_blue[:k, :]

print("Low rank approximation completed.")
print("Reconstructing the image from low rank approximation")
#Reconstruct the image with reduced rank
image_red_k = np.dot(U_red_k, np.dot(S_red_k, VT_red_k))
image_green_k = np.dot(U_green_k, np.dot(S_green_k, VT_green_k))
image_blue_k = np.dot(U_blue_k, np.dot(S_blue_k, VT_blue_k))
image_reconstructed = np.zeros(image.shape)
image_reconstructed[:,:,2] = image_red_k
image_reconstructed[:,:,1] = image_green_k
image_reconstructed[:,:,0] = image_blue_k

image_reconstructed = np.clip(image_reconstructed, 0, 255).astype(np.uint8)
print("Image reconstruction completed.")

#Save the reconstructed image
cv2.imwrite(rf"H:\\My Drive\\Numerical Linear Algebra\\Assignments\\Assignment 3\\SMACS_0723_reconstructed_k{k}.png", image_reconstructed)
print("Reconstructed image saved.")

#Error in image reconstruction for each channel using 2-norm
error_red_2 = np.linalg.norm(image_red - image_red_k, 2)
error_green_2 = np.linalg.norm(image_green - image_green_k, 2)
error_blue_2 = np.linalg.norm(image_blue - image_blue_k, 2)
```

```
print(f"Reconstruction Error (2-Norm) - Red: {error_red_2:.4f}, Green: {error_green_2:.4f}, Blue: {error_blue_2:.4f}")

#Error in image reconstruction for each channel using frobenius norm
error_red_fro = np.linalg.norm(image_red - image_red_k, 'fro')
error_green_fro = np.linalg.norm(image_green - image_green_k, 'fro')
error_blue_fro = np.linalg.norm(image_blue - image_blue_k, 'fro')
print(f"Reconstruction Error (Frobenius Norm) - Red: {error_red_fro:.4f}, Green: {error_green_fro:.4f}, Blue: {error_blue_fro:.4f}")

#Print (k+1)th singular value for each channel
print(f"(k+1)th Singular Value - Red: {S_red[k]:.4f}")
print(f"(k+1)th Singular Value - Green: {S_green[k]:.4f}")
print(f"(k+1)th Singular Value - Blue: {S_blue[k]:.4f}")

red_diff = 0
green_diff = 0
blue_diff = 0

#Sum of squares of singular values from (k+1) to end for each channel
for v in range(k, len(S_red)):
    red_diff += S_red[v] ** 2
    green_diff += S_green[v] ** 2
    blue_diff += S_blue[v] ** 2

print(f"Sq. root of sum of squares of singular values from (k+1) to end - Red: {np.sqrt(red_diff):.4f}")
print(f"Sq. root of sum of squares of singular values from (k+1) to end - Green: {np.sqrt(green_diff):.4f}")
print(f"Sq. root of sum of squares of singular values from (k+1) to end - Blue: {np.sqrt(blue_diff):.4f}")
```

The output of the code is as follows:

```
Image loaded successfully.
Performing SVD on each component:
SVD Completed.
Starting low rank approximation
Calculating energy retained for different values of k to find minimum k for 90.0% energy retention in all channels.
Minimum k to retain at least 90% energy in all channels: 37
Low rank approximation completed.
Reconstructing the image from low rank approximation
Image reconstruction completed.
Reconstructed image saved.
Reconstruction Error (2-Norm) - Red: 4314.3844, Green: 3624.4450, Blue: 3666.7091
Reconstruction Error (Frobenius Norm) - Red: 28680.7619, Green: 24762.1610, Blue: 26069.1641
(k+1)th Singular Value - Red: 4314.3844
(k+1)th Singular Value - Green: 3624.4450
(k+1)th Singular Value - Blue: 3666.7091
Sq. root of sum of squares of singular values from (k+1) to end - Red: 28680.7619
Sq. root of sum of squares of singular values from (k+1) to end - Green: 24762.1610
Sq. root of sum of squares of singular values from (k+1) to end - Blue: 26069.1641
```

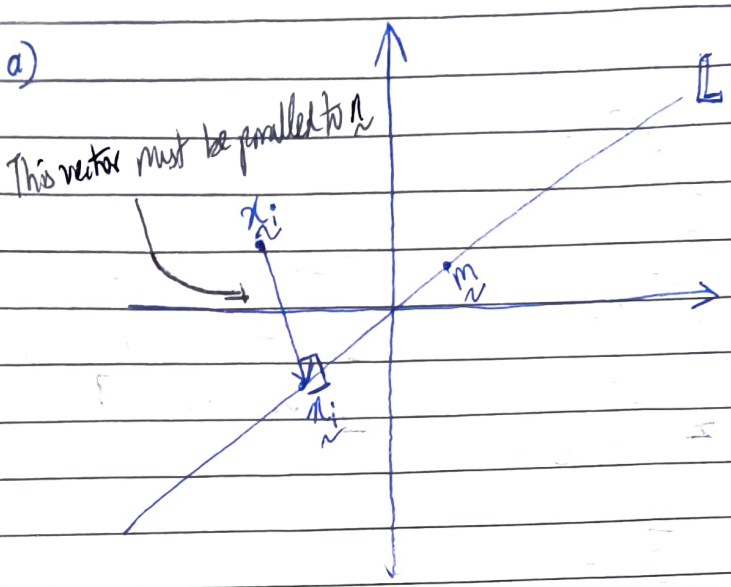
Subhas
Kumuth

Assignment 3

Quick Work
Page No.: 1
Date: 25/4/25
M T W T F S S

25945

3) a)



Because $x_i - \bar{x}_i$ should be parallel to n, we can say that:

$$x_i - \bar{x}_i = ((x_i - m)^T n) n$$

Solving this equation, we get:

$$\bar{x}_i = x_i - ((x_i - m)^T n) n$$

b) We have to minimise the sum of the squares of Euclidean distances between each ~~data point~~ x_i & $\bar{x}_i$. Let us call this sum S.

$$\therefore S = \sum_{i=1}^{N} \| x_i - \bar{x}_i \|_2^2$$

Substituting the expression from part a), we get

$$S = \sum_{i=1}^{N} \| ((\underset{\sim}{x}_i - \underset{\sim}{m})^T \underset{\sim}{n}) \underset{\sim}{n} \|_2^2$$

$$S = \sum_{i=1}^{N} \| [(\underset{\sim}{x}_i - \underset{\sim}{m})^T \underset{\sim}{n}]^2 \cdot \| \underset{\sim}{n} \|_2^2 \quad \left[\text{Because } (\underset{\sim}{x}_i - \underset{\sim}{m})^T \underset{\sim}{n} \text{ is a scaler}\right]$$

$$= \sum_{i=1}^{N} [(\underset{\sim}{x}_i - \underset{\sim}{m})^T \underset{\sim}{n}]^2 \quad \left[\text{Because } \underset{\sim}{n} \text{ is a unit vector}\right]$$

Because we know that the optimal $\underset{\sim}{n}$ is $\underset{\sim}{m}^*$, we can replace $\underset{\sim}{m}$ with $\underset{\sim}{m}^*$.

$$\therefore S = \sum_{i=1}^{N} [(\underset{\sim}{x}_i - \underset{\sim}{m}^*)^T \underset{\sim}{n}]^2$$

c) Let $\underset{\sim}{q}$ be the unit direction vector of L. $\underset{\sim}{q}$ will be orthogonal to $\underset{\sim}{n}$. Hence, $\underset{\sim}{q}$ and $\underset{\sim}{n}$ will form a basis for $\mathbb{R}^2$.

The projection matrix onto the line's direction is $\underset{\sim}{q}\underset{\sim}{q}^T$. Similarly, the projection matrix onto $\underset{\sim}{n}$'s direction is $\underset{\sim}{n}\underset{\sim}{n}^T$.

Because Projection matrix of $\underset{\sim}{q}$ + Projection matrix of $\underset{\sim}{n}$ = Identity matrix,

$$\underset{\sim}{q}\underset{\sim}{q}^T + \underset{\sim}{n}\underset{\sim}{n}^T = \underset{\sim}{I}$$

Let us get down to the minimisation problem: we want to minimise S, where:

$$S = \sum_{i=0}^{N} \left[(x_i - m^*)^T n \right]^2$$

To simplify this expression, let us have $y_i = x_i - m^*$. Then:

$$S = \sum_{i=0}^{N} (y_i^T n)^2$$

We can see that this summation is the squared Frobenius norm of Xnn^T, where:

$$X = \begin{bmatrix} - & y_1^T & - \\ & \vdots & \\ - & y_n^T & - \end{bmatrix}$$

Hence, S can be further simplified: $S = \| X n n^T \|_F^2$

Because we know that $nn^T = I - qq^T$,

$$S = \| X (I - qq^T) \|_F^2$$

$$= \| X \|_F^2 - \| Xq \|_F^2$$

Because $\| X \|_F^2$ is a constant we have to maximise $\| Xq \|_F^2$ so that S is minimised. For this we shall have to find a q s.t. $\| Xq \|_F^2$ is maximised.

d) Because we know that the maximum value of $\|\underset{\sim}{X}\underset{\sim}{q}\|_2^2$ occurs when $\underset{\sim}{q}$ is the eigenvector corresponding to the largest eigenvalue of $\underset{\sim}{X}$.

Furthermore, the maximum value of $\|\underset{\sim}{X}\underset{\sim}{q}\|_2^2$ will be equal to this largest eigenvalue.

If we perform an SVD on $\underset{\sim}{X}$ we know that:

→ The largest eigenvalue will be σ_1^2 (square of largest singular value)

→ $\underset{\sim}{q} = \underset{\sim}{v}_1$, where $\underset{\sim}{v}_1$ is the largest principal component

Hence, the problem is now reduced to finding the best 1-rank approximation for $\underset{\sim}{X}$. We know, from the Eckhart-Young theorem, that the best way to do this is to perform a low rank approximation.

The one-rank approximation of $\underset{\sim}{X} = \underset{\sim}{u}_1 \sigma_1 \underset{\sim}{v}_1^T$

Because σ_1 is a scalar, $\underset{\sim}{X} = \sigma_1 \underset{\sim}{u}_1 \underset{\sim}{v}_1^T$

We can also see that the unit vector of the line, $\underset{\sim}{q} = \underset{\sim}{v}_1$.

Because $\underset{\sim}{V}^T$ is an orthogonal matrix, $\underset{\sim}{v}_1$ & $\underset{\sim}{v}_2$ are orthonormal to each other. Also, because we know that $\underset{\sim}{q}_k = \underset{\sim}{v}_1$, $\underset{\sim}{v}_k$ must be equal to $\underset{\sim}{n}$.

In other words, $\underset{\sim}{V}^T = \begin{bmatrix} | & | \\ \underset{\sim}{q} & \underset{\sim}{n} \\ | & | \end{bmatrix}$

e) To get the equation of the best fit ~~line~~ line, we need a point on the line. We also shall require a unit normal vector to the line

→ We have $\underset{\sim}{m}^*$ as the point on the line.
→ We have $\underset{\sim}{n}$ is the unit normal vector. We may use $\underset{\sim}{V}_2^T$ also.

Hence, the equation for the best-fit line is:

$$\underset{\sim}{L} = \underset{\sim}{n}\left(\underset{\sim}{x}_i - \underset{\sim}{m}^*\right)$$