Jan 2024 Paper

2. Section B (40 marks)

```
import numpy as np
import numpy.linalg as la
from numpy.linalg import inv ,det
from numpy import linalg as LA
import pandas as pd
from sympy import Symbol, Derivative
```

(a) Consider the following matrix and answer the below 2 questions

$$A = \begin{bmatrix} 3 \\ 2 \\ 1 \end{bmatrix}$$
$$B = \begin{bmatrix} -1 \\ 3 \\ 4 \end{bmatrix}$$
$$C = \begin{bmatrix} 5 \\ 0 \\ 1 \end{bmatrix}$$

(i) Check whether the vectors are linearly independent and find the rank of the matrix. (4 marks)

(ii) Check if this matrix is orthogonal. (3 marks)

```
a = np.array([3, 1, 2])
b = np.array([-1, 3, 4])
c = np.array([5, 0, 1])

# Form the matrix using the vectors as columns
matrix = np.column_stack((a, b, c))

# Calculate the determinant of the matrix
determinant = np.linalg.det(matrix)

# Output the determinant
print("Determinant of the matrix:", determinant)

# Check if the determinant is non-zero
if determinant != 0:
    print("The vectors are linearly independent.")
else:
```

```
    print("The vectors are linearly dependent.")

# Calculate the rank of the matrix
rank = np.linalg.matrix_rank(matrix)
print("Rank of the matrix:", rank)

# Check if the matrix is orthogonal
if rank == len(matrix):
    print("The matrix is orthogonal.")
else:
    print("The matrix is not orthogonal.")
```

```
Determinant of the matrix: 0.0
The vectors are linearly dependent.
Rank of the matrix: 2
The matrix is not orthogonal.
```

(b) Find Eigen Values of A, A^2 and A^{-1} for

$$A = \begin{bmatrix} 4 & 3 \\ 2 & 1 \end{bmatrix}$$

(7 marks)

```
# Define the matrix A
A = np.array([[4, 2],
              [1, 3]])

# Calculate A^2
A_squared = np.dot(A, A)

# Calculate the inverse of A
A_inverse = np.linalg.inv(A)

# Find the eigenvalues of A
eigenvalues_A = np.linalg.eigvals(A)

# Find the eigenvalues of A^2
eigenvalues_A_squared = np.linalg.eigvals(A_squared)

# Find the eigenvalues of A^-1
eigenvalues_A_inverse = np.linalg.eigvals(A_inverse)

# Output the results
print("Eigenvalues of A:", eigenvalues_A)
print("Eigenvalues of A^2:", eigenvalues_A_squared)
print("Eigenvalues of A^-1:", eigenvalues_A_inverse)
```

```
Eigenvalues of A: [5. 2.]
Eigenvalues of A^2: [25.  4.]
Eigenvalues of A^-1: [0.2 0.5]
```

(c) Find the scalar and vector projections of the vector u = 8i + 6j on vector v = i - 7j

(6 marks)

```
# Scalar Projection of u on v
u = np.array([3, 4])
v = np.array([3, -2])

sc_proj = np.dot(u, v) / np.linalg.norm(v)

print(sc_proj)


# Vector projection of u on v

vec_proj = (v * sc_proj)/ np.linalg.norm(v)   # (v * (np.dot(u,v)/np.linalg.norm(v)) / np.linalg.norm(v)

vec_proj


# Vector Projection (vector * scalar projection) / v norm

u = np.array([4, 5, 6])   # vector u
v = np.array([1, 2, 2])   # vector v:

vec_proj = v * (np.dot(u,v)/np.linalg.norm(v)) / np.linalg.norm(v)

print("Projection of Vector u on Vector v is: ", vec_proj)
```

```
Projection of Vector u on Vector v is:  [2.88888889 5.77777778 5.77777778]
```

(d) Find singular value decomposition of

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}$$

(6 marks)

```
import numpy as np

# Define the matrix A
A = np.array([[1, 2],
              [3, 4],
              [5, 6]])
```

```
# Calculate the singular value decomposition of A
U, S, Vt = np.linalg.svd(A)

# Output the results
print("Matrix U:")
print(U)
print("\nDiagonal matrix Sigma:")
print(np.diag(S))
print("\nTranspose of matrix V:")
print(Vt)
```

```
Matrix U:
[[-0.2298477   0.88346102  0.40824829]
 [-0.52474482  0.24078249 -0.81649658]
 [-0.81964194 -0.40189603  0.40824829]]

Diagonal matrix Sigma:
[[9.52551809 0.        ]
 [0.         0.51430058]]

Transpose of matrix V:
[[-0.61962948 -0.78489445]
 [-0.78489445  0.61962948]]
```

(e) Convert the following matrix to an orthogonal matrix using Gram Schmidt Process?

$$B = \begin{bmatrix} 1 & 1 & 1 \\ -1 & 0 & 1 \\ 1 & 1 & 2 \end{bmatrix}$$

(6 marks)

```
import numpy as np
u1= np.array([1,-1,1])
u2= np.array([1,0,1])
u3= np.array([1,1,2])
v1=np.copy(u1)
#v1=np.array(v1.append(u1))
print(v1)
v2= (u2-np.dot(u2,v1)/(np.linalg.norm(v1))**2)*v1
print(v2)
v3=u3-(np.dot(u3,v1)/(np.linalg.norm(v1))**2)*v1 -(np.dot(u3,v2)/(np.linalg.norm(v2))**2)*v2
print(v3)
```

```
[ 1 -1  1]
[0.33333333 0.66666667 0.33333333]
[-5.00000000e-01 -2.22044605e-16  5.00000000e-01]
```

(f) A=1111 B=01−11 C=−2100 Check if the following are true or false:

1. (BA + A) = (B + **I**)A, where **I** is the identity matrix.
2. (A + B)C = AC + BC

```
# Define the matrices
A = np.array([[1, 1],
              [1, 1]])

B = np.array([[0, 1],
              [-1, 1]])

C = np.array([[-2, 1],
              [0, 0]])

# Define the identity matrix
I = np.eye(A.shape[0])

# Calculate the left-hand side of the first equation (BA + A)
lhs_first_equation = np.dot(B, A) + A

# Calculate the right-hand side of the first equation ((B + I)A)
rhs_first_equation = np.dot(B + I, A)

# Check if the first equation is true
is_true_first_equation = np.array_equal(lhs_first_equation, rhs_first_equation)

# Calculate the left-hand side of the second equation (A + B)C
lhs_second_equation = np.dot(A + B, C)

# Calculate the right-hand side of the second equation (AC + BC)
rhs_second_equation = np.dot(A, C) + np.dot(B, C)

# Check if the second equation is true
is_true_second_equation = np.array_equal(lhs_second_equation, rhs_second_equation)

# Output the results
print("(BA + A) == (B + I)A is", is_true_first_equation)
print("(A + B)C == AC + BC is", is_true_second_equation)
```

Extra questions (5 to 6 marks)

(1) Find out the minima of the following function for the interval (-5, -2) $f(x) = x3 + 2x$

(2) Find the critical points of the function f(x)=x5−5x4+5x3−1

(3) The revenue generation function of an IT company is 3000x - 20x2 + 200 rupees where x is the number of employees. Find out the marginal revenue generation when 10 employees are hired.

(4) Calculate the angle between two given vectors. The two vectors are, a = i + 2j and b = 9 i + 3j

(5) Verify the following for the matrix A and B and C -

- $(AB)^T = B^T A^T$;
- $(AB)^{-1} = B^{-1} A^{-1}$;
- $A(B+C) = AB + AC$;
- $(A^T)^{-1} = (A^{-1})^T$;
- $(A^T A)^T = A^T A$
- $|C^{-1}| = \frac{1}{|C|}$,

$$A = \begin{bmatrix} 5 & 6 & 2 \\ 4 & 7 & 1 \\ 0 & 3 & 1 \end{bmatrix}$$

,

$$B = \begin{bmatrix} 1 & -2 & 1 \\ 4 & 4 & 5 \\ 5 & 5 & 1 \end{bmatrix}$$

$$C = \begin{bmatrix} 1 & 0 & 1 \\ 4 & 1 & 0 \\ 2 & 0 & 1 \end{bmatrix}$$

```
#(2 ans)
import sympy as sp

# Define the variable
x = sp.symbols('x')

# Define the function
f = x**5 - 5*x**4 + 5*x**3 - 1

# Calculate the derivative of the function
f_prime = sp.diff(f, x)

# Find the critical points by solving f'(x) = 0
critical_points = sp.solve(f_prime, x)

# Output the critical points
print("Critical points:", critical_points)
```

```
Critical points: [0, 1, 3]
```

```
#3 ans

# Define the revenue generation function
revenue_function = 3000*x - 20*x**2 + 200

# Calculate the derivative of the revenue generation function
marginal_revenue_function = sp.diff(revenue_function, x)

# Evaluate the marginal revenue function at x = 10
marginal_revenue_at_10_employees = marginal_revenue_function.subs(x, 10)

# Output the result
print("Marginal revenue generation when 10 employees are hired:", marginal_revenue_at_10_employees, "rupees")
```

```
Marginal revenue generation when 10 employees are hired: 2600 rupees
```

```
#(4 ans)
import numpy as np

# Define the vectors
a = np.array([1, 2])  # Vector a = i + 2j
b = np.array([9, 3])  # Vector b = 9i + 3j

# Calculate the dot product of the vectors
dot_product = np.dot(a, b)

# Calculate the magnitudes of the vectors
magnitude_a = np.linalg.norm(a)
magnitude_b = np.linalg.norm(b)

# Calculate the cosine of the angle between the vectors
cosine_theta = dot_product / (magnitude_a * magnitude_b)

# Calculate the angle in radians
angle_radians = np.arccos(cosine_theta)

# Convert radians to degrees
angle_degrees = np.degrees(angle_radians)

# Output the result
print("Angle between vectors a and b (in degrees):", angle_degrees)
```

```
Angle between vectors a and b (in degrees): 45.00000000000001
```

```
# 5 ans
A = np.array([[5,6,2],[4,7,1],[0,3,1]])
B = np.array([[1,-2,1],[4,4,5],[5,5,1]])
C = np.array([[1,0,1],[4,1,0],[2,0,1]])

print('A=',A)
```

```
print('B=',B)
print('C=',C)
print(100*'=')
print(np.allclose((np.dot(A,B)).T,(np.dot(B.T,A.T))))
print(np.allclose(inv(np.dot(A,B)),np.dot(inv(B),inv(A))))
print(np.allclose(np.dot(A,B+C),np.dot(A,B)+np.dot(A,C)))
print(np.allclose(inv(A.T),(inv(A)).T))
print(np.allclose((np.dot(A.T,A)).T,np.dot(A.T,A)))
print(det(inv(C))== 1.0/det(C))
print(100*'=')
```

```
1
```

3. Section C (40 marks)

3 questions max (15,15, 10) or 2 20 marks questions

(a) image processing problem (convolution, transformation,)

10 marks

```
from PIL import Image, ImageFilter, ImageOps
import numpy as np
import matplotlib.pyplot as plt

# Step 1: Load the image
image_path = "/content/drive/MyDrive/MF_sagarika_changes/MF ESA Jan24 model paper/Lena.jpg"
image = Image.open(image_path)

# Step 2: Display the original image
plt.figure(figsize=(10, 10))
plt.subplot(3, 3, 1)
plt.imshow(image)
plt.title("Original Image")
plt.axis("off")

# Step 3: Apply grayscale transformation
gray_image = ImageOps.grayscale(image)
plt.subplot(3, 3, 2)
plt.imshow(gray_image, cmap="gray")
plt.title("Grayscale Image")
plt.axis("off")

# Step 4: Apply Gaussian blur filter
blurred_image = gray_image.filter(ImageFilter.GaussianBlur(radius=5))
plt.subplot(3, 3, 3)
plt.imshow(blurred_image, cmap="gray")
plt.title("Blurred Image")
plt.axis("off")
```

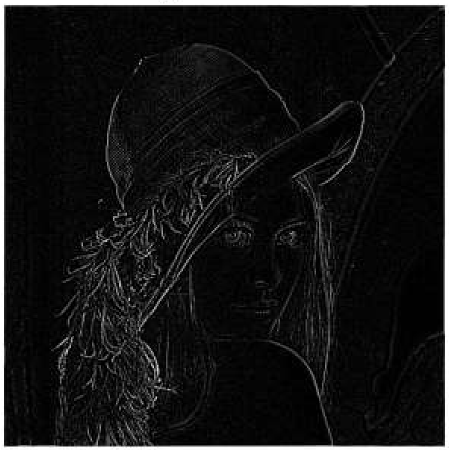
```
# Step 5: Apply edge detection (Sobel filter)
sobel_image = gray_image.filter(ImageFilter.FIND_EDGES)
plt.subplot(3, 3, 4)
plt.imshow(sobel_image, cmap="gray")
plt.title("Edge Detection")
plt.axis("off")

# Step 6: Rotate the image by 45 degrees
rotated_image = image.rotate(45)
plt.subplot(3, 3, 5)
plt.imshow(rotated_image)
plt.title("Rotated Image")
plt.axis("off")

# Step 7: Scale the image
scaled_image = image.resize((int(image.width * 1.5), int(image.height * 1.5)))
plt.subplot(3, 3, 6)
plt.imshow(scaled_image)
plt.title("Scaled Image")
plt.axis("off")

# Step 8: Translate the image
translated_image = image.transpose(Image.FLIP_LEFT_RIGHT)
plt.subplot(3, 3, 7)
plt.imshow(translated_image)
plt.title("Translated Image")
plt.axis("off")

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

Original Image

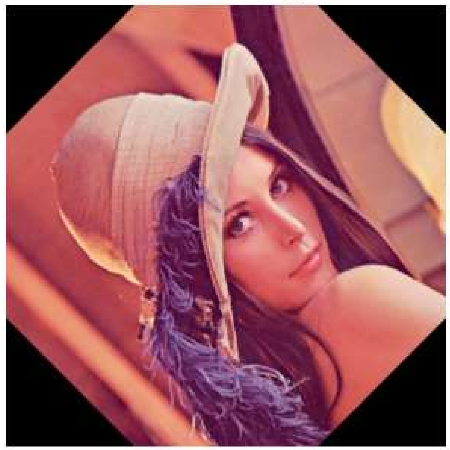
Grayscale Image

Blurred Image

Edge Detection

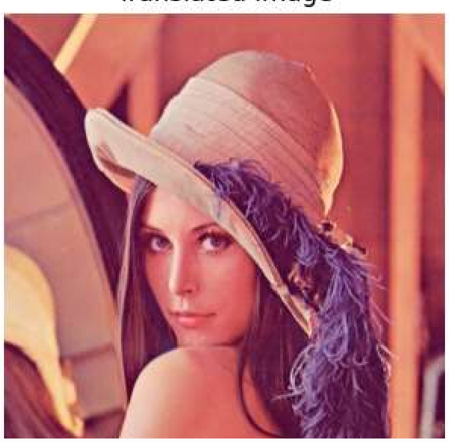


Rotated Image



Scaled Image

Translated Image



Start coding or generate with AI.

Start coding or generate with AI.

(b) function optimization problem or

PCA step by step on a iris dataset and capture 95 % variance(15 marks)

```
import numpy as np
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.preprocessing import StandardScaler

# Step 1: Load the Iris dataset
iris = load_iris()
X = iris.data
y = iris.target
feature_names = iris.feature_names

# Step 2: Standardize the Data
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Step 3: Compute the Covariance Matrix
cov_matrix = np.cov(X_scaled.T)

# Step 4: Compute the Eigenvectors and Eigenvalues
eigenvalues, eigenvectors = np.linalg.eig(cov_matrix)

# Step 5: Select Principal Components
# Sort eigenvalues and eigenvectors in descending order
sorted_indices = np.argsort(eigenvalues)[::-1]
eigenvalues = eigenvalues[sorted_indices]
eigenvectors = eigenvectors[:, sorted_indices]

# Select the top k eigenvectors based on the explained variance ratio
total_variance = np.sum(eigenvalues)
explained_variance_ratio = eigenvalues / total_variance
cumulative_explained_variance_ratio = np.cumsum(explained_variance_ratio)
k = np.argmax(cumulative_explained_variance_ratio >= 0.95) + 1

# Step 6: Project Data onto Principal Components
projection_matrix = eigenvectors[:, :k]
X_pca = X_scaled.dot(projection_matrix)

# Display the explained variance ratio and cumulative explained variance ratio
print("Explained Variance Ratio:", explained_variance_ratio[:k])
```

```
print("Cumulative Explained Variance Ratio:", cumulative_explained_variance_ratio[:k])
```

```
Explained Variance Ratio: [0.72962445 0.22850762]
Cumulative Explained Variance Ratio: [0.72962445 0.95813207]
```

3. Consider the data given below and fit a linear regression line y=ax+b using gradient descent.

X 0 0.4 0.6 1

Y 0 1 0.48 0.95

Initialize the weights a and b to 0.8, 0.2 respectively.

Update the weights such that the error is minimum using gradient descent.

Use the function sum of squared errors $y - y^2$ where $\boxed{\text{y^}}$ is the y-predicted value and y is the actual given y.

Plot the linear regression line after updating the values of a and b in two iterations.

(15 marks)

```
import numpy as np
import matplotlib.pyplot as plt

# Define the dataset
X = np.array([0, 0.4, 0.6, 1])
Y = np.array([0, 1, 0.48, 0.95])

# Define initial weights
a = 0.8
b = 0.2

# Define learning rate and number of iterations
learning_rate = 0.1
iterations = 2

# Define function to calculate sum of squared errors
def sum_squared_errors(Y, Y_pred):
    return np.sum((Y - Y_pred) ** 2)

# Perform gradient descent
for i in range(iterations):
    # Calculate predicted values of y
    Y_pred = a * X + b

    # Calculate gradients
    gradient_a = -2 * np.sum((Y - Y_pred) * X)
    gradient_b = -2 * np.sum(Y - Y_pred)
```

```
    # Update weights
    a -= learning_rate * gradient_a
    b -= learning_rate * gradient_b

    # Print the updated weights and sum of squared errors
    print(f"Iteration {i+1}: a = {a}, b = {b}, Sum of Squared Errors = {sum_squared_errors(Y, Y_pred)}")

# Plot the original data points
plt.scatter(X, Y, color='blue', label='Original data points')

# Plot the linear regression line
plt.plot(X, a*X + b, color='red', label='Linear regression line')

# Add labels and legend
plt.xlabel('X')
plt.ylabel('Y')
plt.title('Linear Regression using Gradient Descent')
plt.legend()

# Show plot
plt.grid(True)
plt.show()
```

```
Iteration 1: a = 0.8044, b = 0.20600000000000002, Sum of Squared Errors = 0.31289999999999996
Iteration 2: a = 0.8050624, b = 0.20544, Sum of Squared Errors = 0.3126254272
```