Q 1 solution

Calculate the following for the matrix A,B and C where I is 3 by 3 Identity matrix

- $|A + I|$;
- BC;
- AB;
- A^2
- Value of $D^2 - 5D - 2I$

$$A = \begin{bmatrix} 5 & 6 & 2 \\ 4 & 7 & 19 \\ 0 & 3 & 12 \end{bmatrix}$$

$$B = \begin{bmatrix} 14 \\ 4 \\ 5 \end{bmatrix}$$

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

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

In [4]:
```
import numpy as np

A = np.array([[5, 6, 2], [4, 7, 19],[0, 3, 12]])
B = np.array([14, 4, 5])
C= np.array([1,2,3])
D=np.array([[1,2],[3,4]])
I= np.identity(3)
I2= np.identity(2)
det= np.linalg.det(A+I)
print("|A+I|= :" , det)

BC= np.dot(B,C)
print("BC is :" , BC)
M= np.dot(A,B)
print("The Multiplication of  A and B is :" , M)
P= np.dot(A,A)
print("The A square is :" , P)
print('D^2-5D-2I=',np.dot(D,D)-5*D-2*I2)
```

```
|A+I|= : -5.999999999999998
BC is : 37
The Multiplication of  A and B is : [104 179  72]
The A square is : [[ 49  78 148]
 [ 48 130 369]
 [ 12  57 201]]
D^2-5D-2I= [[0. 0.]
 [0. 0.]]
```

Q 2 Solution

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

$$B = \begin{bmatrix} 1 & 7 & 9 \\ 3 & 4 & 5 \\ 1 & 2 & 1 \end{bmatrix}$$

In [2]:
```
import numpy as np
u1= np.array([1,7,9])
u2= np.array([3,4,5])
u3= np.array([1,2,1])
v1=np.copy(u1)
#v1=np.array(v1.append(u1))
print(v1/(np.linalg.norm(v1)))
v2= u2-(np.dot(u2,v1)/(np.linalg.norm(v1))**2)*v1
print(v2/(np.linalg.norm(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/(np.linalg.norm(v3)))
```

```
[0.08737041 0.61159284 0.78633365]
[ 0.99552721 -0.02512372 -0.09107347]
[-0.03594426  0.79077367 -0.61105238]
```

In [ ]:

Q 3 Solution

- Find Scalar and Vector Projection of u on v

u = [6,7,8] v= [2,3,-1]

- Verify Q is orthogonal

$$Q = \frac{1}{3}\begin{bmatrix} 2 & -2 & 1 \\ 1 & 2 & 2 \\ 2 & -1 & -2 \end{bmatrix}$$

In [25]:

```
# Scalar Projection of u on v
u = np.array([6,7,8])
v = np.array([2,3,-1])
sc_proj = np.dot(u, v) / np.linalg.norm(v)   # u.v/||v||
sc_proj
```

Out[25]: 6.681531047810609

In [26]:

```
# 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
```

Out[26]: array([3.57142857, 5.35714286, -1.78571429])

In [29]:

```
from numpy import array
from numpy.linalg import inv
# define orthogonal matrix
Q =1/3* array([
[2,-2, 1],
[1, 2, 2],
[2,-1, -2]])
print(Q)
# inverse equivalence
V = inv(Q)
#print(Q.T)
print(V)
```

```
[[ 0.66666667 -0.66666667  0.33333333]
 [ 0.33333333  0.66666667  0.66666667]
 [ 0.66666667 -0.33333333 -0.66666667]]
[[ 0.28571429  0.71428571  0.85714286]
 [-0.85714286  0.85714286  0.42857143]
 [ 0.71428571  0.28571429 -0.85714286]]
```

In [30]:

```
# identity equivalence
I = Q.dot(Q.T)
print(I)
```

```
[[ 1.00000000e+00  1.54197642e-17  4.44444444e-01]
 [ 1.54197642e-17  1.00000000e+00 -4.44444444e-01]
 [ 4.44444444e-01 -4.44444444e-01  1.00000000e+00]]
```

Q 4 solution

The Following table lists the weight and heights of 5 boys Find the covariance matrix for the data.

Boy	1	2	3	4	5
Weight(lb)	120	125	125	135	145
Height(in.)	61	60	64	68	72

In [6]:

```
x = [120,125,125,135,145]
y = [61,60,64,68,72]
X = np.stack((x, y), axis=0)
np.cov(X)
```

Out[6]:
```
array([[100. ,  47.5],
       [ 47.5,  25. ]])
```

Question 5 solution

Find eigen values and eigen vectors for

$$A = \begin{bmatrix} 5 & 6 & 2 \\ 4 & 7 & 19 \\ 0 & 3 & 12 \end{bmatrix}$$

In [3]:
```
from numpy import linalg as LA
import numpy as np

a = np.array([[5, 6, 2],[4,7,19],[0,3,12]])
w, v = np.linalg.eig(a)

print(w)
print(v)
```

```
[-1.03997841  6.80080283 18.23917558]
[[ 0.6664281   0.91899137  0.42824655]
 [-0.72658863  0.34150445  0.81440731]
 [ 0.16716024 -0.19705222  0.39159371]]
```

Question 6

Find singular value decomposition of

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

In [16]:
```
# singular-value decomposition
from numpy import array
from scipy.linalg import svd
# define a matrix
A = array([
[1, 2],
[3, 4],
[5, 6]])
print(A)
# factorize
U, s, V = svd(A)
print(U)
print(s)
print(V)
```

```
[[1 2]
 [3 4]
 [5 6]]
[[-0.2298477   0.88346102  0.40824829]
 [-0.52474482  0.24078249 -0.81649658]
 [-0.81964194 -0.40189603  0.40824829]]
[9.52551809 0.51430058]
[[-0.61962948 -0.78489445]
 [-0.78489445  0.61962948]]
```

Question 7 (8 Marks)

1. From sklearn.dataset import iris data
2. Preprocess the data Preprocess the data by subtracting the mean and dividing by the standard deviation of each attribute value. The resulting data should be zero-mean with variance 1.
3. Compute the covriance matrix
4. Factorize the covariance matrix using singular value decomposition (SVD) and obtain the eigenvalues and eigenvectors.
5. Find principle components.

In [3]:
```
import numpy as np
from sklearn.datasets import load_iris
from sklearn.preprocessing import StandardScaler
```

```python
# Step 1: Load the Iris dataset
iris = load_iris()
X = iris.data

# Step 2: Standardize the data
scaler = StandardScaler()
X_std = scaler.fit_transform(X)

# Step 3: Compute the covariance matrix
cov_matrix = np.cov(X_std.T)

# Step 4: Factorize the covariance matrix using SVD
U, S, Vt = np.linalg.svd(cov_matrix)

# Eigenvalues and eigenvectors from SVD
eigenvalues = S
eigenvectors = Vt.T

# Step 5: Find the principal components
# Project the data onto the eigenvectors
principal_components = X_std.dot(eigenvectors)

print("Eigenvalues:\n", eigenvalues)
print("Eigenvectors:\n", eigenvectors)
print("Principal Components:\n", principal_components.shape)
```

```
Eigenvalues:
 [2.93808505 0.9201649  0.14774182 0.02085386]
Eigenvectors:
 [[-0.52106591 -0.37741762  0.71956635  0.26128628]
 [ 0.26934744 -0.92329566 -0.24438178 -0.12350962]
 [-0.5804131  -0.02449161 -0.14212637 -0.80144925]
 [-0.56485654 -0.06694199 -0.63427274  0.52359713]]
Principal Components:
 (150, 4)
```

In [39]:
```python
# mean Centering the data
X_meaned = X - np.mean(X , axis = 0)
```

In [40]:
```python
# calculating the covariance matrix of the mean-centered data.
cov_mat = np.cov(X_meaned , rowvar = False)
```

In [41]:
```python
#Calculating Eigenvalues and Eigenvectors of the covariance matrix
eigen_values , eigen_vectors = np.linalg.eigh(cov_mat)
```

In [42]:
```python
#sort the eigenvalues in descending order
sorted_index = np.argsort(eigen_values)[::-1]

sorted_eigenvalue = eigen_values[sorted_index]
#similarly sort the eigenvectors
sorted_eigenvectors = eigen_vectors[:,sorted_index]
```

In [43]:
```python
# select the first n eigenvectors, n is desired dimension
# of our final reduced data.

n_components = 2 #you can select any number of components.
eigenvector_subset = sorted_eigenvectors[:,0:n_components]
```

In [44]:
```python
#Transform the data
X_reduced = np.dot(eigenvector_subset.transpose(),X_meaned.transpose()).transpose()
```

In [46]:
```python
#sort the eigenvalues in descending order
sorted_index = np.argsort(eigen_values)[::-1]

sorted_eigenvalue = eigen_values[sorted_index]
#similarly sort the eigenvectors
sorted_eigenvectors = eigen_vectors[:,sorted_index]
```

In [47]:
```python
def PCA(X , num_components):

    #Step-1
    X_meaned = X - np.mean(X , axis = 0)

    #Step-2
    cov_mat = np.cov(X_meaned , rowvar = False)

    #Step-3
    eigen_values , eigen_vectors = np.linalg.eigh(cov_mat)

    #Step-4
    sorted_index = np.argsort(eigen_values)[::-1]
```

```
    sorted_eigenvalue = eigen_values[sorted_index]
    sorted_eigenvectors = eigen_vectors[:,sorted_index]

    #Step-5
    eigenvector_subset = sorted_eigenvectors[:,0:num_components]

    #Step-6
    X_reduced = np.dot(eigenvector_subset.transpose() , X_meaned.transpose() ).transpose()

    return X_reduced
```

In [48]:

```
import pandas as pd

#Get the IRIS dataset
url = "https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data"
data = pd.read_csv(url, names=['sepal length','sepal width','petal length','petal width','target'])

#prepare the data
x = data.iloc[:,0:4]

#prepare the target
target = data.iloc[:,4]

#Applying it to PCA function
mat_reduced = PCA(x , 2)

#Creating a Pandas DataFrame of reduced Dataset
principal_df = pd.DataFrame(mat_reduced , columns = ['PC1','PC2'])

#Concat it with target variable to create a complete Dataset
principal_df = pd.concat([principal_df , pd.DataFrame(target)] , axis = 1)
```

Question 8

Solve the following system

$$\begin{pmatrix} 1 & -1 & 1 \\ 2 & 1 & -3 \\ 1 & 1 & 1 \end{pmatrix}\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} 4 \\ 0 \\ 2 \end{pmatrix}$$

In [20]:

```
import numpy as np

a = np.array([[1,-1,1], [2,1,-3],[1,1,1]])
b = np.array([4,0,2])
x = np.linalg.solve(a, b)

print(x)
```

```
[ 2. -1.  1.]
```

In [1]:

```
import sympy as sym
sym.init_printing()
x,y,z = sym.symbols('x,y,z')
sol = sym.solve([x + (-1)*y + z - 4,
2*x +y +(-3)*z - 0,
x+y+z - 2],[x,y,z])
sol
```

Out[1]: $\{x: 2,\ y: -1,\ z: 1\}$

In [ ]:

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js