Homework4 - Wooseok Kim

In [1]:
```
import numpy as np
```

A given matrix is like below:

$$X = \begin{bmatrix} -2 & 1 & 4 & 6 & 5 & 3 & 6 & 2 \\ 9 & 3 & 2 & -1 & -4 & -2 & -4 & 5 \\ 0 & 7 & -5 & 3 & 2 & -3 & 4 & 6 \end{bmatrix}$$

In [2]:
```
X = np.array([
        [-2, 1, 4, 6, 5, 3, 6, 2],
        [9, 3, 2, -1, -4, -2, -4, 5],
        [0, 7, -5, 3, 2, -3, 4, 6]
         ])
```

In [3]:
```
print(X)
```

```
[[-2  1  4  6  5  3  6  2]
 [ 9  3  2 -1 -4 -2 -4  5]
 [ 0  7 -5  3  2 -3  4  6]]
```

1. write command to compute the mean of the data matrix X use function mean. Your code have to return the mean in terms of a bf column vector.
2. Use your code, compute the mean of matrix X as given in the problem setting

$$mean = \frac{1}{n} \sum_{k=1}^{n} X_k$$

In [4]:
```
X_mean = np.mean(X, axis=1)
```

In [5]:
```
print(X_mean)
```

```
[3.125 1.    1.75 ]
```

1. Write code to center data matrix X, you can't use any loop command Use variable X1 for the resulting centered matrix.
2. Use your code, compute the centered data matrix X as given in the problem setting.

Center Data Matrix

$$X = \sum_{k=1}^{n} (X_k - mean)$$

In [6]:
```
X1 = np.zeros( (len(X), len(X[0]) ))
for i in range(len(X[0])):
        X1[:,i] = X[:,i] - X_mean
```

In [7]:
```
print(X1)
```

```
[[-5.125 -2.125  0.875  2.875  1.875 -0.125  2.875 -1.125]
 [ 8.     2.     1.    -2.    -5.    -3.    -5.     4.   ]
 [-1.75   5.25  -6.75   1.25   0.25  -4.75   2.25   4.25 ]]
```

1. write code to compute unnormalized covariance matrix of the centered data matrix X1. Use variable C for the resulting covariance matrix.
2. use your code, compute the covariance matrix of matrix X as given in the problem setting.

Covariance Matrix

$$C = XX^T = \sum_{k=1}^{n} (X_k - mean)(X_k - mean)^T$$

In [8]:
```
C = X1.dot(X1.T)
print("covariance matrix\n", C)
```

```
covariance matrix
 [[ 52.875 -78.     -1.75 ]
 [-78.    148.      6.   ]
 [ -1.75    6.    123.5  ]]
```

1. write code to compute the first principal component. (corresponding to the maximum eigenvalue of C).
2. use your code, compute the first principal component and its corresponding principal value for matrix X as given in the problem setting.

$$(XX^T)v = \lambda v$$
$$v = Eigenvector$$

$$\lambda = Eigenvalue$$

In [9]:
```
eigenvalue_C, eigenvector_C = np.linalg.eig(C)
print("Principal Component\n", eigenvector_C)
```

```
Principal Component
 [[-0.87168926 -0.48708504  0.05390734]
 [-0.48990322  0.86889736 -0.07079702]
 [ 0.01235577  0.08812238  0.99603302]]
```

1. write code to compute the best 1D representation of data matrix X. [hint: don't forget to add back the data mean].
2. use your code, compute the best 1D representation of matrix X as given in the problem setting.

In [11]:
```
arrNum = np.argsort(eigenvalue_C)[::-1]
eigenvector = eigenvector_C[:,arrNum]
eigenvalue = eigenvalue_C[arrNum]


bestRep = eigenvalue[0] / np.sum(eigenvalue_C)
bestRepVector = eigenvector[:,:1]
bestRepMatrix = np.dot(bestRepVector.T, X)
```

Sort eigenvalues and eigenvectors from highest to lowest

The best 1D principal value

$$\frac{\sum_{i=1}^{r} \lambda_i}{\sum_{i=1}^{d} \lambda_i}$$

Eigenvector

$$v = \begin{bmatrix} v_{11} & v_{12} & \dots & v_{1r} \\ v_{21} & v_{22} & \dots & v_{2r} \\ \vdots & \vdots & \ddots & \vdots \\ v_{n1} & v_{n2} & \dots & v_{nr} \end{bmatrix}$$

The best 1D representation

$$v^T X$$

In [14]:
```
print("The best 1D representation of matrix X\n", bestRepMatrix)
print("\nThe best 1D principal component\n", bestRepVector)
print("\nThe best 1D principal value: ", bestRep)
```

```
The best 1D representation of matrix X
 [[ 8.79424627  2.73646372 -0.65115735 -3.52704041 -5.73476983 -3.46
341697
  -6.04561009  3.89905101]]

The best 1D principal component
 [[-0.48708504]
 [ 0.86889736]
 [ 0.08812238]]

The best 1D principal value:  0.5929360221022756
```

1. collect all previous steps, write a function with name mypca, which take inputs of a data matrix assuming column data, and return the best k-dimensional representation. Your function should use the following declaration: function [rep, pc, pv] = mypca(X, k), where rep contains the optimal k dimensional representation, pc contains k top principal components, and pv contains the top k principal values.

In [15]:
```
def mypca(X, k):
        X_mean = np.mean(X, axis=1)
        X1 = np.zeros( (len(X), len(X[0]) ))
        for i in range(len(X[0])):
                X1[:,i] = X[:,i] - X_mean
        C = X1.dot(X1.T)
        eigenvalue_C, eigenvector_C = np.linalg.eig(C)
        arrNum = np.argsort(eigenvalue_C)[::-1]
        eigenvector = eigenvector_C[:,arrNum]
        eigenvalue = eigenvalue_C[arrNum]

        sum = 0
        for i in range(k):
                sum += eigenvalue[i]

        pv = sum / np.sum(eigenvalue_C)
        pc = eigenvector[:,:k]
        rep = np.dot(pc.T, X)

        return np.array([rep, pc, pv])
```

I made 2 python files, hw4.py, and hw4_instruction.py. All the information above is included in hw4_instruction.py. hw4.py includes below code, which is whole code regarding homework4.

Actual Code

In [16]:

```
def mypca(X, k):
        X_mean = np.mean(X, axis=1)
        X1 = np.zeros( (len(X), len(X[0]) ))
        for i in range(len(X[0])):
                X1[:,i] = X[:,i] - X_mean
        C = X1.dot(X1.T)
        eigenvalue_C, eigenvector_C = np.linalg.eig(C)
        arrNum = np.argsort(eigenvalue_C)[::-1]
        eigenvector = eigenvector_C[:,arrNum]
        eigenvalue = eigenvalue_C[arrNum]

        sum = 0
        for i in range(k):
                sum += eigenvalue[i]

        pv = sum / np.sum(eigenvalue_C)
        pc = eigenvector[:,:k]
        rep = np.dot(pc.T, X)

        return np.array([rep, pc, pv])



def main():
        X = np.array([
        [-2, 1, 4, 6, 5, 3, 6, 2],
        [9, 3, 2, -1, -4, -2, -4, 5],
        [0, 7, -5, 3, 2, -3, 4, 6]
        ])
        rep1, pc1, pv1 = mypca(X, 1)
        rep2, pc2, pv2 = mypca(X, 2)
        rep3, pc3, pv3 = mypca(X, 3)

        print("Optimal 1 dimensional representation\n", rep1)
        print("1 top principal component\n", pc1)
        print("top 1 principal value: ", pv1)

        print("\n\nOptimal 2 dimensional representation\n", rep2)
        print("2 top principal component\n", pc2)
        print("top 2 principal value: ", pv2)

        print("\n\nOptimal 3 dimensional representation\n", rep3)
        print("3 top principal component\n", pc3)
        print("top 3 principal value: ", pv3)

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

```
Optimal 1 dimensional representation
 [[ 8.79424627  2.73646372 -0.65115735 -3.52704041 -5.73476983 -3.46
341697
  -6.04561009  3.89905101]]
1 top principal component
 [[-0.48708504]
 [ 0.86889736]
 [ 0.08812238]]
top 1 principal value:  0.5929360221022756


Optimal 2 dimensional representation
 [[ 8.79424627  2.73646372 -0.65115735 -3.52704041 -5.73476983 -3.46
341697
  -6.04561009  3.89905101]
 [-0.74498789  6.81374742 -4.9061298   3.38234013  2.54479084 -2.684
783
   4.59076422  5.73002769]]
2 top principal component
 [[-0.48708504  0.05390734]
 [ 0.86889736 -0.07079702]
 [ 0.08812238  0.99603302]]
top 2 principal value:  0.9720614555458488


Optimal 3 dimensional representation
 [[ 8.79424627  2.73646372 -0.65115735 -3.52704041 -5.73476983 -3.46
341697
  -6.04561009  3.89905101]
 [-0.74498789  6.81374742 -4.9061298   3.38234013  2.54479084 -2.684
783
   4.59076422  5.73002769]
 [-2.66575048 -2.25490851 -4.52834236 -4.70316502 -2.37412186 -1.672
32866
  -3.22109958 -4.11875999]]
3 top principal component
 [[-0.48708504  0.05390734 -0.87168926]
 [ 0.86889736 -0.07079702 -0.48990322]
 [ 0.08812238  0.99603302  0.01235577]]
top 3 principal value:  1.0
```

In [ ]: