Assignment - I

Name:	**Gavali Deshabhakt Nagnath**
Subject:	**Machine Learning**
Specialization :	**Computational and Data Sciences**
Department:	**Mathematical and Computational Sciences**
Course Instructor:	**Dr. Jidesh P.**
Data:	**14/04/2021**

Q1. Write codes to perform, LU, LDU, QR, and SV Decomposition.

A. LU - Decomposition:

Program:

```
import numpy as np
import copy

def inputMatrix():                  # Function to take matrix input from user
    print("Enter the size of matrix: ")
    n= int(input())

    A= np.zeros((n,n),dtype=float)
    print("Now enter elements of matrix 'A':")
    for i in range(n):
        print("Enter elements for row:",i+1)
        for j in range(n):
            A[i][j]=int(input())
    return A

def printMatrix(V):                 # Function to print Matrix
    for i in range(n):
        for j in range(n):
            print(f'{V[i][j]:15.08f}' ,  end="  ")
        print()
    print()

def LUDecomposition(A):     #LU-Decomposition function definition

    n = len(A)

    L= np.zeros((n,n),dtype=float)
    for i in range(len(L)):
        L[i][i]=1

    U = copy.copy(A)    # copying matrix A into U

    for i in range(0,n-1):
        for j in range(i+1,n):
            L[j][i]= (U[j][i]/U[i][i])
            U[j][:]=U[j][:]-L[j][i]*U[i][:]

    return L,U


# Default input
n = 3

A = np.array([
```

```
    [1,2,4],
    [3,8,14],
    [2,6,13]
])

# Uncomment below line to take input from user
# A = inputMatrix()

# Calling Function of matrix A
L,U = LUDecomposition(A)

# Printing Results
print("A = ")
printMatrix(A)

print("L = ")
printMatrix(L)

print("U = ")
printMatrix(U)
```

Output:

```
A =
    1.00000000    2.00000000     4.00000000
    3.00000000    8.00000000    14.00000000
    2.00000000    6.00000000    13.00000000

L =
    1.00000000    0.00000000     0.00000000
    3.00000000    1.00000000     0.00000000
    2.00000000    1.00000000     1.00000000

U =
    1.00000000    2.00000000     4.00000000
    0.00000000    2.00000000     2.00000000
    0.00000000    0.00000000     3.00000000
```

B. LDU – Decomposition

Program:

```
import numpy as np
import copy

def inputMatrix():                # Function to take matrix input from user
    print("Enter the size of matrix: ")
    n= int(input())

    A= np.zeros((n,n),dtype=float)
    print("Now enter elements of matrix 'A':")
```

```
    for i in range(n):
        print("Enter elements for row:",i+1)
        for j in range(n):
            A[i][j]=int(input())
    return A

def printMatrix(V):                # Function to print matrix
    for i in range(n):
        for j in range(n):

            print(f'{V[i][j]:15.08f}' ,  end="  ")
        print()
    print()


def LUDecomposition(A):                # LDU – Decomposition Function Definition

    n = len(A)

    L= np.zeros((n,n),dtype=float)
    D = np.zeros((n,n),dtype=float)

    for i in range(len(L)):
        L[i][i]=1

    U = copy.copy(A)    # copying matrix A into U



    for i in range(0,n-1):
        for j in range(i+1,n):
            L[j][i]= (U[j][i]/U[i][i])
            U[j][:]=U[j][:]-L[j][i]*U[i][:]

    for i in range(n):
        if(U[i][i]!=0):
            D[i][i] = copy.copy(U[i][i])
            U[i,:] = copy.copy(U[i,:]/U[i][i])

    return L,D,U

# Default Input
n = 3

A = np.array([
    [1,2,4],
    [3,8,14],
    [2,6,13]
])
```

```
# Calling LDU decomposition function
L, D, U = LUDecomposition(A)

print("A = ")
printMatrix(A)

print("L = ")
printMatrix(L)

print("D = ")
printMatrix(D)

print("U = ")
printMatrix(U)
```

Output:

```
A =
    1.00000000      2.00000000      4.00000000
    3.00000000      8.00000000     14.00000000
    2.00000000      6.00000000     13.00000000

L =
    1.00000000      0.00000000      0.00000000
    3.00000000      1.00000000      0.00000000
    2.00000000      1.00000000      1.00000000

D =
    1.00000000      0.00000000      0.00000000
    0.00000000      2.00000000      0.00000000
    0.00000000      0.00000000      3.00000000

U =
    1.00000000      2.00000000      4.00000000
    0.00000000      1.00000000      1.00000000
    0.00000000      0.00000000      1.00000000
```

C. QR – Decomposition

Program:

```
import numpy as np

def matrixInput():
    m = int(input("Enter row size :"))
    n = int(input("Enter column size :"))

    A = np.zeros((m,n), dtype=float)
```

```
    print("Enter elements of matrix: ")
    for i in range(m):
        for j in range(n):
            A[i][j] = float(input())

    return A


def Normalize(v):
    sum = 0.0
    for i in v:
        sum+=i**2
    v=v/(sum**0.5)
    return v

def QRDecomp(A):

    n = len(A[0])   # Columns/Vectors
    m = len(A)      # Rows/Components
    q = []

    q.append(Normalize(A[:,0].reshape(m,1)))

    for i in range(1,n):

        vec = A[:,i].astype('float64').reshape(m,1)
        temp = np.zeros((m,1),dtype=float)

        for j in range(i):

            multiplier = (((vec.transpose()).dot((q[j]))))/(q[j].transpose().dot(q[j]))
            temp -= (multiplier)*q[j]

        vec = vec + temp
        normalizedvec = Normalize(vec)
        q.append(normalizedvec)

    Q = np.array(q).transpose().reshape(m,n)    # typecasting python list to numpy array and taking
np.array's transpose

    # Calculating R
    R = np.zeros((n,n))
    for i in range(n):
        for j in range(n):
            if i<=j:
                R[i][j] = A[:,j].transpose().dot(Q[:,i])

    return Q,R
```

```
def printMatrix(V):
    m = len(V)
    n = len(V[0])
    for i in range(m):
        for j in range(n):
                print(f'{V[i][j]:10.05f}' ,  end="  ")
        print()
    print()

# Default Input

A = np.array(((
    (1, -1, 4),
    (1, 4, -2),
    (1, 4, 2),
    (1, -1, 0)
)))

# Uncomment following lines for custom input
# A = matrixInput()

# Calling QR-decomposition function
Q,R = QRDecomp(A)

print("A = ")
printMatrix(A)

print("Q =")
printMatrix(Q)

print("R =")
printMatrix(R)
```

Output:

```
A =
   1.00000    -1.00000     4.00000
   1.00000     4.00000    -2.00000
   1.00000     4.00000     2.00000
   1.00000    -1.00000     0.00000

Q =
   0.50000    -0.50000     0.50000
   0.50000     0.50000    -0.50000
   0.50000     0.50000     0.50000
   0.50000    -0.50000    -0.50000

R =
   2.00000     3.00000     2.00000
```

```
0.00000     5.00000    -2.00000
0.00000     0.00000     4.00000
```

D. SVD

Program:

```
# # SVD Implementation
# ## Importing libraries

# In[1]:

import numpy as np
import copy

# ## SVD function Definition

# In[2]:


def printMatrix(V):
    m = len(V)
    n = len(V[0])

    for i in range(m):
        for j in range(n):
            print(f'{V[i][j]:10.05f}' ,  end="  ")
        print()
    print()


def SVD(A):
    m = len(A)
    n = len(A[0])

    At = A.transpose()

    AtA = np.matmul(At,A)

    AAt = np.matmul(A,At)

    # Finding Eigen Values and Vectors of AAt and AtA
    eigValuesAAt, eigVectorsAAt = np.linalg.eig(AAt)

    eigValuesAtA, eigVectorsAtA = np.linalg.eig(AtA)

    # Forming U, D and VT
    U = eigVectorsAAt
```

```
# Sorting eigen values in descending order and also changing position of corresponding eigen vectors
    idx = eigValuesAAt.argsort()[::-1]
    eigValuesAAt = eigValuesAAt[idx]
    eigVectorsAAt = eigVectorsAAt[:,idx]
    eigVectorsAtA = eigVectorsAtA[:,idx]


    D = np.zeros((m,n))
    for i in range(m):
        for j in range(n):
            if i==j:
                D[i][j] = (eigValuesAAt[i])**(1/2)
            else:
                D[i][j] = 0

    Vt = eigVectorsAtA.transpose()

    return U,D,Vt


# In[3]:


A = np.array(((
    (1,2,3),
    (4,5,6),
    (7,8,9)
)))

# A = np.array(((
#     (1, -1, 4),
#     (1, 4, -2),
#     (1, 4, 2),
#     (1, -1, 0)
# )))

# Calling SVD-decomposition function
U,D,Vt = SVD(A)


# In[4]:

print("A = ")
printMatrix(A)
print("U = ")
printMatrix(U)
print("D = ")
printMatrix(D)
```

```
print("VT = ")
printMatrix(Vt)
```

Output:

```
A =
   1.00000     2.00000     3.00000
   4.00000     5.00000     6.00000
   7.00000     8.00000     9.00000

U =
  -0.21484    -0.88723     0.40825
  -0.52059    -0.24964    -0.81650
  -0.82634     0.38794     0.40825

D =
  16.84810     0.00000     0.00000
   0.00000     1.06837     0.00000
   0.00000     0.00000     0.00000

VT =
  -0.47967    -0.57237    -0.66506
  -0.77669    -0.07569     0.62532
   0.40825    -0.81650     0.40825
```