Homework - 1

Srinidhi Bharadwaj Kalgundi Srinivas

A59010584

In [40]:
```
import numpy as np
```

Problem 1

In [41]:
```
# Matrix multiplication funciton
def multiply_matrix(a, b):
    c = np.zeros((a.shape[1], b.shape[0]))
    if a.shape[1] != b.shape[0]:
        return 0
    for i in range(a.shape[0]):
        row = a[i]
        for j in range(b.shape[1]):
            col = b[:,j]
            sum = 0
            for k in range(row.shape[0]):
                sum += row[k]*col[k]
            c[i, j] = sum
    return c
```

Algorithm used for LDU decomposition

- Goal of the algorithm is to start with A = LDU + B (where L, D, U are zero matrices to start with) and iteratively find L, D, U such that B turns out to be a zero matrix
- Column 'c' of the L matrix is calculated by the column of B that contains the element
- Row 'r' of the U matrix is calculated by the row of B that contains the element
- Diagonal entry is calculated as (1/element)
- New B is calculated as B = A - LDU
- Above steps are repeated for all the elements

Note: As a verification step, L, D and U matrices are computed using the above steps and the product LDU is subtracted from the original matrix A to obtain and resultant zero matrix

In [42]:

```
def LDU_Decomposition(A):
    '''This method implements the LDU decomposition of
    the input matrix A'''

    #Creating a temporary copy as python list copy is referenced
    B = A.copy()
    #print(A.shape)

    matrix_size = A.shape[0]

    #Initializing L, D, U matrices with size of the input matrix
    L = np.zeros((matrix_size, matrix_size))
    U = np.zeros((matrix_size, matrix_size))
    D = np.zeros((matrix_size, matrix_size))

    #Looping through individual elements of the matrix
    for i in range(A.shape[0]):
        for j in range(A.shape[1]):
            if B[i, j] == 0: #No update needed if the value is 0, essential t
                continue
            L[:,j] = B[:,j]  #Assign the column of B to L
            U[i] = B[i]      #Assign the row of B to R
            D[i, i] = 1/B[i, j] #Get the diagonal element
            C = multiply_matrix(D, U) #Intermediate step to update B as B = A
            B = A - multiply_matrix(L, C)

    return L, D, U
```

In [43]:
```
#Example vector of size 3x3
A = np.array([[-7,4,-1],[-1,-6,1],[1,0,0]])
lower, diag, upper = LDU_Decomposition(A)

#Verfication to prove A = LDU -> Purely for verification purposes

#Calculate LDU
C = multiply_matrix(diag, upper)
LDU = multiply_matrix(lower, C)

#Subtract LDU from A to obtain a matrix with zero elements
result = A - LDU
print(result)
```

```
[[0. 0. 0.]
 [0. 0. 0.]
 [0. 0. 0.]]
```

In [ ]:
```
```

Problem 2

In [44]:
```
# a)

A1 = np.array([[4, 7, 0], [3, 2, 1], [2, 2, -6]])

lower, diag, upper = LDU_Decomposition(A1)

u, s, vd = np.linalg.svd(A1, full_matrices=True)

print("Lower diagonal matrix from LDU: ")
print(lower)
print("Diagonal matrix from LDU: ")
print(diag)
print("Upper diagonal matrix from LDU: ")
print(upper)

print("Matrices from SVD Decomposition are: ")
print(u)
print("")
print(s)
print("")
print(vd)
```

```
Lower diagonal matrix from LDU:
[[ 4.          0.          0.        ]
 [ 3.         -3.25        0.        ]
 [ 2.         -1.5        -6.46153846]]
Diagonal matrix from LDU:
[[ 0.25        0.          0.        ]
 [ 0.         -0.30769231  0.        ]
 [ 0.          0.         -0.1547619 ]]
Upper diagonal matrix from LDU:
[[ 4.          7.          0.        ]
 [ 0.         -3.25        1.        ]
 [ 0.          0.         -6.46153846]]
Matrices from SVD Decomposition are:
[[ 0.83108771  0.36392168 -0.42054041]
 [ 0.3206405   0.30429769  0.89699085]
 [ 0.45440389 -0.88032036  0.13621004]]

[9.33057832 5.78987054 1.55489788]

[[ 0.55678008  0.7896298  -0.25783856]
 [ 0.10500046  0.24100823  0.96482638]
 [ 0.82399688 -0.56426927  0.05127709]]
```

In [45]:

```
# b)

A2 = np.array([[1,0,0,0,1],
               [0,0,1,0,0],
               [0,1,0,1,0],
               [0,1,0,0,0],
               [1,0,0,0,0]])
lower, diag, upper = LDU_Decomposition(A2)

u, s, vd = np.linalg.svd(A2, full_matrices=True)

print("Lower diagonal matrix from LDU: ")
print(lower)
print("Diagonal matrix from LDU: ")
print(diag)
print("Upper diagonal matrix from LDU: ")
print(upper)

print("Matrices from SVD Decomposition are: ")
print(u)
print("")
print(s)
print("")
print(vd)
```

```
Lower diagonal matrix from LDU:
[[ 1.  0.  0.  0.  0.]
 [ 0.  0.  1.  1.  0.]
 [ 0.  1. -1.  0.  0.]
 [ 0.  1. -1. -1.  0.]
 [ 1.  0.  0.  0. -1.]]
Diagonal matrix from LDU:
[[ 1.  0.  0.  0.  0.]
 [ 0.  1.  0.  0.  0.]
 [ 0.  0. -1.  0.  0.]
 [ 0.  0.  0. -1.  0.]
 [ 0.  0.  0.  0. -1.]]
Upper diagonal matrix from LDU:
[[ 1.  0.  0.  0.  1.]
 [ 0.  0.  1.  0.  0.]
 [ 0.  1. -1.  1.  0.]
 [ 0.  0.  0. -1.  0.]
 [ 0.  0.  0.  0. -1.]]
Matrices from SVD Decomposition are:
[[-0.85065081  0.          0.         -0.52573111  0.        ]
 [ 0.          0.         -1.          0.          0.        ]
 [ 0.         -0.85065081  0.          0.         -0.52573111]
 [ 0.         -0.52573111  0.          0.          0.85065081]
 [-0.52573111  0.          0.          0.85065081  0.        ]]

[1.61803399 1.61803399 1.         0.61803399 0.61803399]

[[-0.85065081  0.          0.          0.         -0.52573111]
 [-0.         -0.85065081 -0.         -0.52573111  0.        ]
 [-0.         -0.         -1.         -0.          0.        ]
 [ 0.52573111  0.          0.          0.         -0.85065081]
 [ 0.          0.52573111  0.         -0.85065081  0.        ]]
```

In [46]:

```
# c)

A3 = np.array([[2,2,5],
               [3,2,5],
               [1,1,5]])

lower, diag, upper = LDU_Decomposition(A3)

u, s, vd = np.linalg.svd(A3, full_matrices=True)

print("Lower diagonal matrix from LDU: ")
print(lower)
print("Diagonal matrix from LDU: ")
print(diag)
print("Upper diagonal matrix from LDU: ")
print(upper)

print("Matrices from SVD Decomposition are: ")
print(u)
print("")
print(s)
print("")
print(vd)
```

```
Lower diagonal matrix from LDU:
[[ 2.   0.   0. ]
 [ 3.  -1.   0. ]
 [ 1.   0.   2.5]]
Diagonal matrix from LDU:
[[ 0.5  0.   0. ]
 [ 0.  -1.   0. ]
 [ 0.   0.   0.4]]
Upper diagonal matrix from LDU:
[[ 2.   2.   5. ]
 [ 0.  -1.  -2.5]
 [ 0.   0.   2.5]]
Matrices from SVD Decomposition are:
[[-0.58592436 -0.04442838 -0.80914693]
 [-0.62305157 -0.61376658  0.48486836]
 [-0.51816926  0.78823645  0.33193962]]

[9.79103061 1.4162264  0.36058604]

[[-0.36351359 -0.29987866 -0.88200378]
 [-0.8063118  -0.37293011  0.45911264]
 [ 0.46660385 -0.87806373  0.10623054]]
```

In [ ]:

Problem 3 : SVD interpretation of system of equations with no solutions

Equations b and c do not have any solutions

SVD can be used to analyze the system of equations that have no solutions. The eigen values in the 's' matrix below code represent the rank of the matrix. If any of these eigen values are zero or very small (~0), inverse of the matrix A will not exist and hence the system will not be solvable.

Note: Row reduction method has been handwritten and the PDF is attached in gradescope

In [115…

```
# Problem 3a, to show that a system with solution is full rank (no zero or ve
A = np.array([[2, 1, 3],
              [2, 1, 2],
              [5, 5, 5]])
b = np.array([[10], [-10], [0]])
u, s, vd = np.linalg.svd(A, full_matrices=False)
soln = vd.T.dot(np.diag(1/s)).dot(u.T).dot(b)
print("Solution is : \n", soln)
```

```
Solution is :
 [[-30.]
 [ 10.]
 [ 20.]]
```

In [111…

```
# Problem 3b
A = np.array([[8, 14, 0],
              [2, 2, -6],
              [1, 2, 1]])
b = np.array([[6], [5], [1]])
u, s, vd = np.linalg.svd(A, full_matrices=True)

#The value in the third index of the s matrix is very small (6.88359513e-17)
print(u, s, vd)
np.linalg.inv(u @ np.diag(s) @ vd) @ b
soln = vd.T.dot(np.diag(1/s)).dot(u.T).dot(b)
print("Solution is : \n", soln)
```

```
[[-0.9736016   0.16057442 -0.16222142]
 [-0.18682932 -0.96890609  0.16222142]
 [-0.13112871  0.18824675  0.97332853]] [1.65315460e+01 6.05871178e+00 1.53596
605e-16] [[-0.50168327 -0.86297667  0.05987626]
 [-0.07674405  0.11334475  0.99058736]
 [-0.86164044  0.49236596 -0.12309149]]
```

Out[111…

```
array([[-4.55011804e+15],
       [ 2.60006745e+15],
       [-6.50016863e+14]])
```

In [116…

```
# Problem 3c
A = np.array([[4, 7, 0],
              [2, 2, -6],
              [1, 2, 1]])
b = np.array([[18], [-12], [8]])

u, s, vd = np.linalg.svd(A, full_matrices=True)

#The value in the third index of the s matrix is very small (6.88359513e-17)
#infinitely many solutions
print(u, s, vd)
soln = vd.T.dot(np.diag(1/s)).dot(u.T).dot(b)
print("Solution is : \n", soln)
```

```
[[-0.84780042  0.42857143 -0.31234752]
 [-0.49083182 -0.85714286  0.15617376]
 [-0.20079484  0.28571429  0.93704257]] [9.05538514e+00 5.74456265e+00 6.88359
513e-17] [[-0.50507627 -0.80812204  0.30304576]
 [ 0.04973647  0.32328708  0.94499299]
 [ 0.86164044 -0.49236596  0.12309149]]
Solution is :
 [[-4. ]
 [ 0. ]
 [ 3.5]]
```

In [ ]: