LAA Assignment

Himanshu, MDS202327

In [ ]:
```
from helper_functions import luGEPP, luSolve
import scipy.linalg as sp
import numpy as np
import time
import pandas as pd
```

In [ ]:
```
st = time.perf_counter()
time.sleep(5)
en = time.perf_counter()
en-st
```

Out[ ]:
```
5.000906399975065
```

Helper functions

In [ ]:
```
def luGEPP(A):
    '''
    Input: A square matrix (A)
    Function: Performs the LU decompositon of the input matrix using
    Gaussian Elimination with Partial Pivoting such that PA = LU.
    Output: Permutation Matrix (P)
    Unit Lower Triangular Matrix (L)
    Upper Triangular Matrix (U)
    Time Taken for the factorisation process (t)
    '''
    n = len(A)
    A = A.astype(float)
    P = np.identity(n, dtype=float)
    L = np.zeros((n,n),dtype=float)
    U = np.copy(A)
    st = time.perf_counter()
    for i in range(n):
        max_pivot = abs(U[i:,i])
        if max(max_pivot) == 0:
            print("unable to find nonzero pivot")
            break
        max_row = np.argmax(abs(U[i:,i])) + i

        U[[i, max_row]] = U[[max_row, i]]
        P[[i, max_row]] = P[[max_row, i]]
        L[[i, max_row]] = L[[max_row, i]]

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

    np.fill_diagonal(L, 1.0)
    en = time.perf_counter()
    t = (en-st)*1000

    return P, L, U, t

def forward_sub(L, Pb):
    '''
    Forward Substitution to find y such that Ly = Pb
    '''
    n = len(L)
    y = np.zeros(n, dtype=float)
    for i in range(n):
```

```
        y[i] = Pb[i] - sum(L[i,:i] * y[:i])
    return y

def backward_sub(U, y):
    '''
    Backward Substitution to find x such that Ux = y
    '''
    n = len(U)
    x = np.zeros(n, dtype=float)
    for i in range(n-1, -1, -1):
        x[i] = (y[i] - sum(U[i,i+1:] * x[i+1:])) / U[i,i]
    return x

def luSolve(A, b):
    '''
    Input: A square matrix (A) and a column vector (b) of size same as that of A.
    Function: Computes a column vector (x) using the luGEPP() to factorise A,
    and then, doing backward and forward substitution to solve for x such that, Ax = b.
    Output: Column vector (x) with same size as b.
    Time Taken (in ms) to get the solution (t).
    '''

    P, L, U, _ = luGEPP(A)
    Pb = np.dot(P, b)
    st = time.perf_counter()
    y = forward_sub(L, Pb)
    x = backward_sub(U, y)
    en = time.perf_counter()
    t = (en-st)*1000
    return x,t
```

Driver Code

In [ ]:
```
N = [10, 50, 100, 400, 800, 1000]
factorTime = {
    'Size':N,
    "SciPy factoriser (ms)":[],
    "Self factoriser (ms)": []
    }

solveTime = {
    'Size':N,
    "SciPy solver (ms)":[],
    "Self solver (ms)": []
    }

matNorm = {
    'Size':N,
    "Matrix Norm Scipy":[],
    "Matrix Norm Self":[],
    "Vector Norm Scipy":[],
    "Vector Norm Self":[]
    }
```

In [ ]:
```
for n in N:
    A = 200*np.random.random_sample(size=(n,n))-100
    b = 20*np.random.random_sample(size=n)-20

    # Scipy factoriser
    factorST = time.perf_counter()
    P, L, U = sp.lu(A)
    factorEND = time.perf_counter()
    factorT = (factorEND - factorST)*1000
    factorTime["SciPy factoriser (ms)"].append(factorT)
```

```
    # Self factoriser
    p, l, u, ft = luGEPP(A)
    factorTime["Self factoriser (ms)"].append(ft)

    # Scipy solver
    solveST = time.perf_counter()
    X = sp.lu_solve(sp.lu_factor(A), b)
    solveEND = time.perf_counter()
    solveT = (solveEND - solveST)*1000
    solveTime["SciPy solver (ms)"].append(solveT)

    # Self solver
    x, st = luSolve(A,b)
    solveTime["Self solver (ms)"].append(st)

    # Norm of (PA-LU) for Scipy factoriser
    matNorm["Matrix Norm Scipy"].append(np.linalg.norm(P.T@A - L@U))

    # Norm of (PA-LU) for Self factoriser
    matNorm["Matrix Norm Self"].append(np.linalg.norm(p@A-l@u))

    # Norm of (Ax-b) for Scipy solver
    matNorm["Vector Norm Scipy"].append(np.linalg.norm(A@X-b))

    # Norm of (Ax-b) for Self solver
    matNorm["Vector Norm Self"].append(np.linalg.norm(A@x-b))
```

Factorization Time Comparison

In [ ]:

```
factorDF = pd.DataFrame(factorTime)
factorDF["Time Difference (ms)"] = np.subtract(factorDF["Self factoriser (ms)"],
                                               factorDF["SciPy factoriser (ms)"])
factorDF
```

Out[ ]:

	Size	SciPy factoriser (ms)	Self factoriser (ms)	Time Difference (ms)
0	10	0.5586	1.3131	0.7545
1	50	0.2288	12.8834	12.6546
2	100	40.2254	29.2221	-11.0033
3	400	9.6145	334.6275	325.0130
4	800	27.2704	1415.8016	1388.5312
5	1000	44.6676	2332.9476	2288.2800

Solver Time Comparison

In [ ]:

```
solveDF = pd.DataFrame(solveTime)
solveDF['Time Difference (ms)'] = np.subtract(solveDF["Self solver (ms)"],
                                              solveDF["SciPy solver (ms)"])
solveDF
```

Out[ ]:

	Size	SciPy solver (ms)	Self solver (ms)	Time Difference (ms)
0	10	0.2965	0.0749	-0.2216
1	50	0.1621	0.4464	0.2843
2	100	9.1784	2.3594	-6.8190
3	400	8.2130	15.1024	6.8894
4	800	23.5115	57.1029	33.5914
5	1000	49.2274	96.5267	47.2993

Matrix (PA-LU) and Vector (Ax-b) Norm Comparison

In [ ]:

```
normtempDF = pd.DataFrame(matNorm)
normsDF = pd.DataFrame()
normsDF['Size'] = normtempDF['Size']
normsDF['Matrix Norm Difference'] = np.subtract(normtempDF['Matrix Norm Self'],
                                            normtempDF['Matrix Norm Scipy'])
normsDF['Vector Norm Difference'] = np.subtract(normtempDF['Matrix Norm Self'],
                                            normtempDF['Vector Norm Scipy'])
normsDF
```

Out[ ]:

	Size	Matrix Norm Difference	Vector Norm Difference
0	10	1.524314e-14	-1.472951e-14
1	50	4.518089e-13	1.128427e-12
2	100	1.657303e-12	3.569810e-12
3	400	1.887651e-11	3.161320e-11
4	800	3.553470e-11	9.776235e-11
5	1000	6.105484e-11	-2.943371e-10