Q1) Implementing Gaussian Elimination Method

Assignment - 1

Q1 (i)

Operation	Addition (s)	Multiplication (s)	Division (s)
Approximate Time taken	2.2328×10^{-7}	0.0003076	3.7836×10^{-7}

(iii)

N	Without pivoting			With pivoting		
	Addition	Multiplication	Division.	Addition	Multiplication	Division.
100	343,300	343,300	5050	343300	343300	5050
200	2,706,600	2,706,600	20,100	2,706,600	2,706,600	20,100
300	9,089,900	9,089,900	45,150	9,089,900	9,089,900	45,150
400	21,493,200	21,493,200	80,200	21,493,200	21,493,200	80,200
500	41,916,500	41,916,500	125,250	41,916,500	41,916,500	125,250
600	72,359,800	72,359,800	180,300	72,359,800	72,359,800	180,300
700	114,823,100	114,823,100	245,350	114,823,100	114,823,100	245,350
800	171,306,400	171,306,400	320,400	171,306,400	171,306,400	320,400
900	243,809,700	243,809,700	405,450	243,809,700	243,809,700	405,450
1000	334,333,000	334,333,000	500,500	334,333,000	334,333,000	500,500

(ii)

```python
import numpy as np
import pandas as pd
import time
import random
import math
# Function to convert given number to dS arithmetic
def to_dS(N,d=5):
    if N==0:
        return 0
    else:
        # no of places to the left of the decimal point
        l = int(math.floor(math.log10(abs(N)))) + 1
        # no of floating points to be rounded
        f = d-l
        return round(N,f)
def get_rank(A):
    rank = len(A)
    for row in A:
        if sum(row) == 0:
            rank -= 1
    return rank
```

```python
def gauss_ellimination(A,b,pivot_enable=True,d=5):
    nA = len(A)
    nb = len(b)
    add_count = 0
    mul_count = 0
    div_count = 0

    if nA != nb:
        print("Incompatible A matrix and b vector.")
```

```
        return 0

    # Create Augmented matrix
    A_b = A
    for i in range(nA):
        A_b[i].append(b[i])
    # print(A_b)

    ## Forward ellimination process -------------------
    # Do below for each row
    for row in range(nA):
        # Partial pivotting (if enabled)
        if pivot_enable:
            # Current pivot value
            max_pivot = abs(A_b[row][row])
            max_pivot_index = row
            # Iterate through pivot column to find the maximum pivot value
            for i in range(row+1,nA):
                if max_pivot < abs(A_b[i][row]):
                    max_pivot = abs(A_b[i][row])
                    max_pivot_index = i

            # Do partial pivotting
            if row < max_pivot_index:
                A_b[row],A_b[max_pivot_index] = A_b[max_pivot_index],A_b[row]

        else:
            if A_b[row][row] == 0:
                print("Pivot value is zero. Please enable partial pivotting to do
the calculations...")
                return 0

        ## Apply Row transformation for all the rows below current row
        pivot_element = A_b[row][row]
        for row2 in range(row+1, nA):
            # Interested element: The element that we set to zero
            interested_element = A_b[row2][row]

            # Check if interested element if zero and skip
            if interested_element == 0:
                continue

            # Calculate the row multiple value
            row_multiple = to_dS(interested_element/pivot_element,d)
            div_count += 1
```

```
            # Update the interested element to zero
            A_b[row2][row] = 0

            # Iterated through other columns to update the rest of the values in
row
            # nA+1 since Augmented value at the end of row
            for col in range(row+1, nA+1):
                tmp = A_b[row2][col] - to_dS(row_multiple*A_b[row][col],d)
                A_b[row2][col]= to_dS(tmp,d)
                mul_count += 1
                add_count += 1
    # print("Augmented matrix after forward ellimination: \n", A_b)
    ## End of Forward ellimination process -------------------
    #list to keep results of x
    x = [None for _ in range(nA)]

    # Get the row echolon form from augmented matrix by removing the last element
from each row
    # ref_A = list(A_b)
    ref_A = [None for _ in range(nA)]
    for i in range(nA):
        ref_A[i] = A_b[i][:]
        # print("Before pop: ", ref_A[i])
        # ref_A[i].pop()
        del ref_A[i][-1]
        # print("After pop: ", ref_A[i])

    # print("refA: ",ref_A)
    # print("AugA: ",A_b)

    # Check if the linear system has solutions
    # Case1: Finite solutions
    if get_rank(ref_A) == nA:
        ## Backward substitution process -------------------
        # Backward iteration loop for rows
        for row in range(nA-1,-1,-1):
            # Keep sum of product in a row
            row_sop = 0
            # Backward iteration loop for cols
            for col in range(nA-1,row,-1):
                tmp = to_dS(A_b[row][col]*x[col],d)
                row_sop = to_dS(row_sop+tmp,d)
                mul_count += 1
                add_count += 1
            # Obtain the x value: x = (b - row_sop)/a
```

```
            # print(A_b)
            tmp = to_dS(A_b[row][nA] - row_sop,d)
            x[row] = to_dS(tmp/A_b[row][row],d)
            add_count += 1
            div_count += 1

        # Return x solutions and operation counts
        print("Solution x = \n",x,"No of additions: ",add_count,"\n No of 
multiplications: ", mul_count, "\n No of divisions: ", div_count)
        return x,add_count,mul_count,div_count


    # Case2: No solutions
    elif get_rank(ref_A) != get_rank(A_b):
        print("Inconsistent linear system...")

    # Case3: Infinite solutions
    elif  get_rank(ref_A) < nA:
        print("Infinite solutions....")

    return 0
```

(iv)

(iv) [a]

No:	N	Actual time with pivoting (s)	Actual time w/o pivoting (s)	Theoritical total time (s)
01	100	0.43850	0.46733	691,650
02	200	3.4231	3.3564	5,433,300
03	300	11.2893	12.1024	18,224,950
04	400	27.4325	27.6521	43,066,600
05	500	50.1145	52.4325	83958250
06	600	87.7543	90.2619	144,899,900
07.	700	143.6721	154.7321	229,891,550
08	800	213.0025	238.6710	342,933,200
09	900	301.8956	326.1436	488,024,850
10	1000	416.2617	441.3970	669,166,500

[b]

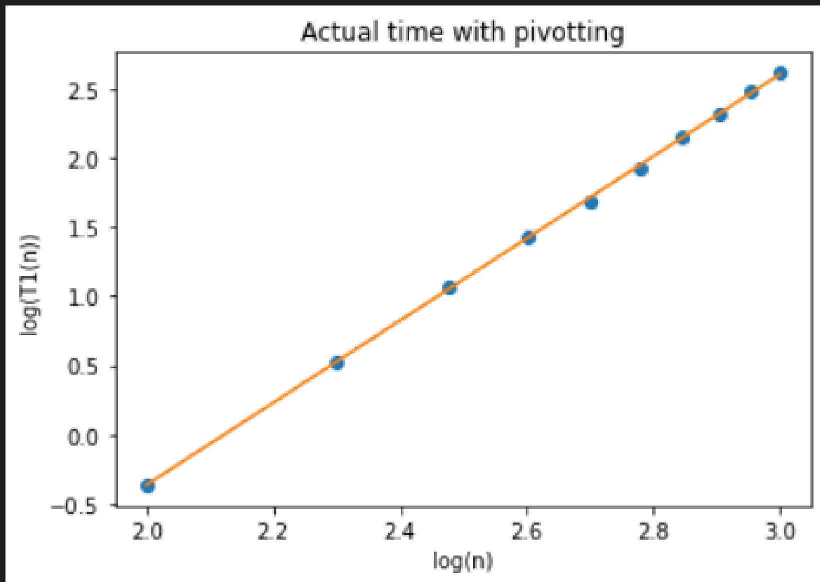
~~slope with pivoting = 2.97~~

slope with pivoting = 2.97

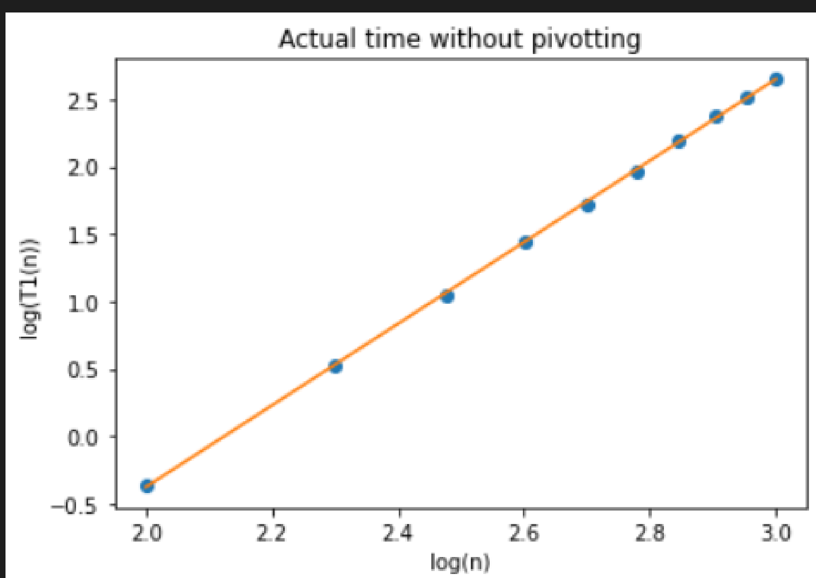
slope w/o pivoting = 3.02

(b)

```
Slope with pivotting:  2.973677653475038
```




```
Slope without pivotting:  3.0237825029839267
```



Q2) Implementing Gauss Seidel and Gauss Jacobi Methods

(i) Write a function to check whether a given square matrix is diagonally dominant or not. If not, the function should indicate if the matrix can be made diagonally dominant by interchanging the rows? Code to be written and submitted. (1)

Deliverable(s): The code

```
def is_diag_dom(A):
    #traverse rows
    for i in range(len(A)):
        row_sum = 0
        #traverse each column value in row
        for j in range(len(A[i])):
            row_sum += abs(A[i][j])
        if abs(A[i][i]) <= row_sum - abs(A[i][i]):
            return False
    return True

def can_diag_dom(A):
    permutation_lst = list(permutations(range(len(A))))
#     print(permutation_lst)
    for order in permutation_lst:
#         print(list(order))
        new_A = A[list(order)]
#         print(new_A)
        # Check row interchanged matrix is diagonally dominant
        if is_diag_dom(new_A):
            print("The matrix can be made diagonally dominant by interchanging 
the rows")
            print("Diagonally dominant matrix : {}".format(new_A))
            return True

    print("The matrix cannot be made diagonally dominant by interchanging the 
rows")
    return False
```

(ii) Write a function to generate Gauss Seidel iteration for a given square matrix. The function should also return the values of $1, \infty$ and Frobenius norms of the iteration matrix. Generate a random 4×4 matrix.
Report the iteration matrix and its norm values returned by the function along with the input matrix. (1)

Deliverable(s): The input matrix, iteration matrix and the three norms Obtained

Question 2

(ii) Random 4×4 Matrix

$$\begin{bmatrix} 0.47468969 & 0.07766808 & 0.36152096 & 0.68413888 \\ 0.98403711 & 0.77247109 & 0.38462241 & 0.06245002 \\ 0.74677540 & 0.58656961 & 0.82082262 & 0.20289251 \\ 0.55028426 & 0.0098881 & 0.69711275 & 0.55256012 \end{bmatrix}_{4\times4}$$

GS iteration matrix:

$$\begin{bmatrix} 0.00000000\times10^{0} & -1.63618638\times10^{-1} & -7.61594296\times10^{-1} & -1.44123392\times10^{0} \\ 0.00000000\times10^{0} & 2.08430858\times10^{-1} & 4.72269634\times10^{-1} & 1.75511766\times10^{0} \\ 0.00000000\times10^{0} & -8.87318616\times10^{-5} & 3.55400622\times10^{-1} & -1.90191086\times10^{-1} \\ 0.00000000\times10^{0} & 1.59326795\times10^{-1} & 3.01630897\times10^{-1} & 1.64383598\times10^{0} \end{bmatrix}_{4\times4}$$

Norm 1 = 5.0303786447342 3.

Frobenius Norm = 3.0019855728 97997

Norm 3 = 2.4358181507814507

(iii) Random 4×4 matrix

$$\begin{bmatrix} 0.18555686 & 0.69298259 & 0.64736743 & 0.04216195 \\ 0.41123198 & 0.99989315 & 0.39108323 & 0.30185870 \\ 0.13223849 & 0.23603213 & 0.92394961 & 0.17158608 \\ 0.0230949 & 0.24984723 & 0.07034604 & 0.36144684 \end{bmatrix}_{4\times4}$$

GJ iteration matrix:

$$\begin{bmatrix} 0.00000000 & -3.73461056 & -3.48878204 & -0.22721850 \\ -0.41127585 & 0.00000000 & -0.39112502 & -0.30189096 \\ -0.14312305 & -0.25545996 & 0.00000000 & -0.18570935 \\ -0.06389569 & -0.69124200 & -0.19462348 & 0.00000000 \end{bmatrix}_{4\times4}$$

Norm 1 = 4.6813125182 86023

Frobenius Norm = 5.2176448351 1 9604

Norm 3 = 7.4506111092 80567.

(iv) Write a function that perform Gauss Seidel iterations. Generate a Random 4×4 matrix A and a suitable random vector $b \in \mathbb{R}^4$ and report the results of passing this matrix to the functions written above.
Write down the first ten iterates of Gauss Seidel algorithm. Does it converge? Generate a plot of $\|x_{k+1} - x_k\|_2$ for the first 10 iterations.
Take a screenshot and paste it in the assignment document. (1)
Deliverable(s): The input matrix and the vector, the 10 successive iterates and the plot

(iv) Random 4x4 matrix

$$\begin{bmatrix} 0.19324773 & 0.45980187 & 0.11964718 & 0.69836178 \\ 0.25918718 & 0.64047834 & 0.24723603 & 0.23240717 \\ 0.40642848 & 0.93426151 & 0.84734846 & 0.134252\text{[illegible]}86 \\ 0.37988218 & 0.05264220 & 0.32869123 & 0.34077902 \end{bmatrix}_{4\times 4}$$

Random vector

$$\begin{bmatrix} 0.98146814 & 0.71036472 & 0.75660719 & 0.14515545 \end{bmatrix}$$

Iteration 1

$$\begin{bmatrix} 0.98146814 \\ 0.31318663 \\ -0.05946181 \\ -0.93995980 \end{bmatrix}$$

Iteration 2

$$\begin{bmatrix} 3.66994806 \\ -0.41074892 \\ -0.40186805 \\ -3.49484167 \end{bmatrix}$$

Iteration 3

$$\begin{bmatrix} 14.83730685 \\ -3.87067585 \\ -1.53865644 \\ -14.31267340 \end{bmatrix}$$

Iteration 4

$$\begin{bmatrix} 62.85713225 \\ -18.94303334 \\ -6.24373446 \\ -60.98724569 \end{bmatrix}$$

Iteration 5

$$\begin{bmatrix} 270.31580614 \\ -84.14003536 \\ -26.46658919 \\ -262.66300194 \end{bmatrix}$$

Iteration 6

$$\begin{bmatrix} 1166.78146242 \\ -365.93207732 \\ -113.80511842 \\ -1134.22419501 \end{bmatrix}$$

Iteration 7

$$\begin{bmatrix} 5040.99696486 \\ -1583.76671349 \\ -491.22470845 \\ -4900.83353721 \end{bmatrix}$$

Iteration 8

$$\begin{bmatrix} 21784.14771129 \\ -6846.88202953 \\ -2122.30003599 \\ -21178.9557591 \end{bmatrix}$$

Iteration 9

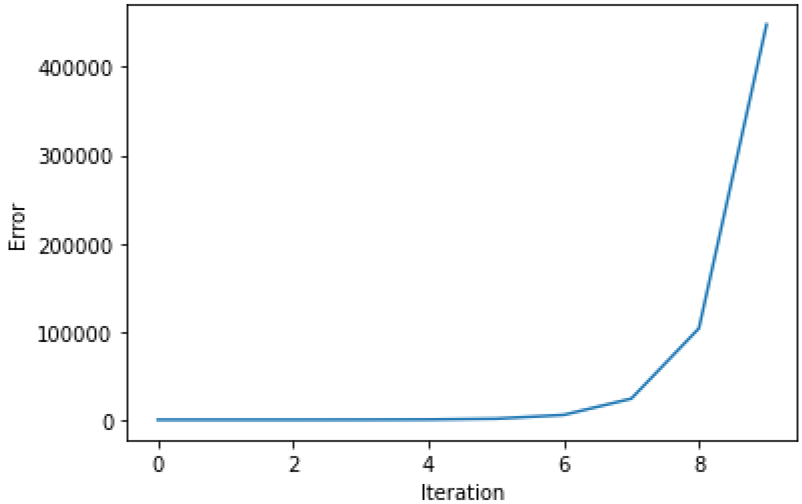
$$\begin{bmatrix} 94142.88847822 \\ -29592.45275044 \\ -9071.29768677 \\ -91527.99262526 \end{bmatrix}$$

Iteration 10

$$\begin{bmatrix} 406855.11514495 \\ -127891.81879393 \\ -39634.89394023 \\ -395554.90026586 \end{bmatrix}$$

In [82]:

```
import matplotlib.pyplot as plt
plt.plot(diff_x)
plt.ylabel('Error')
plt.xlabel('Iteration')
plt.show()
```



Solution does not converge

(v) Repeat part (iv) for the Gauss Jacobi method. (1)

Deliverable(s): The input matrix and the vector, the 10 successive iterates and the plot

(i) Random 4x4 matrix

$$\begin{bmatrix} 5.64386749\times10^{-1} & 7.81231937\times10^{-1} & 4.24221442\times10^{-1} & 6.44968758\times10^{-1} \\ 6.28425792\times10^{-1} & 8.94888036\times10^{-1} & 6.03613438\times10^{-1} & 9.72497710\times10^{-1} \\ 9.31398577\times10^{-2} & 9.82079265\times10^{-1} & 3.49818186\times10^{-1} & 5.35886066\times10^{-1} \\ 1.04518654\times10^{-1} & 4.37689467\times10^{-1} & 3.7022487\times10^{-4} & 6.26642814\times10^{-1} \end{bmatrix}$$

Random vector.

$$\begin{bmatrix} 0.74407137 & 0.55459531 & 0.60745097 & 0.94404768 \end{bmatrix}$$

Iteration 1

$$\begin{bmatrix} 0.74407137 \\ 0.55459531 \\ 0.60745097 \\ 0.94404768 \end{bmatrix}$$

Iteration 2

$$\begin{bmatrix} -1.55903471 \\ -1.40357525 \\ -2.59381556 \\ 0.43221741 \end{bmatrix}$$

Iteration 3

$$\begin{bmatrix} 7.1426332 \\ 2.92927127 \\ 4.30082995 \\ 2.18596507 \end{bmatrix}$$

Iteration 4

$$\begin{bmatrix} -9.04145909 \\ -7.63103465 \\ -12.06784506 \\ -1.79544899 \end{bmatrix}$$

Iteration 5

$$\begin{bmatrix} 22.42965334 \\ 16.9949420 \\ 27.18856535 \\ 7.78924296 \end{bmatrix}$$

Iteration 7

$$\begin{bmatrix} 124.52489651 \\ 96.96001818 \\ 154.88900293 \\ 39.01109232 \end{bmatrix}$$

Iteration 8

$$\begin{bmatrix} -294.47270138 \\ -233.76068175 \\ -364.51409075 \\ -87.64054268 \end{bmatrix}$$

Iteration 9.

$$\begin{bmatrix} 498.4596286 \\ 548.45561684 \\ 869.52742712 \\ 213.54910388 \end{bmatrix}$$

Iteration 6

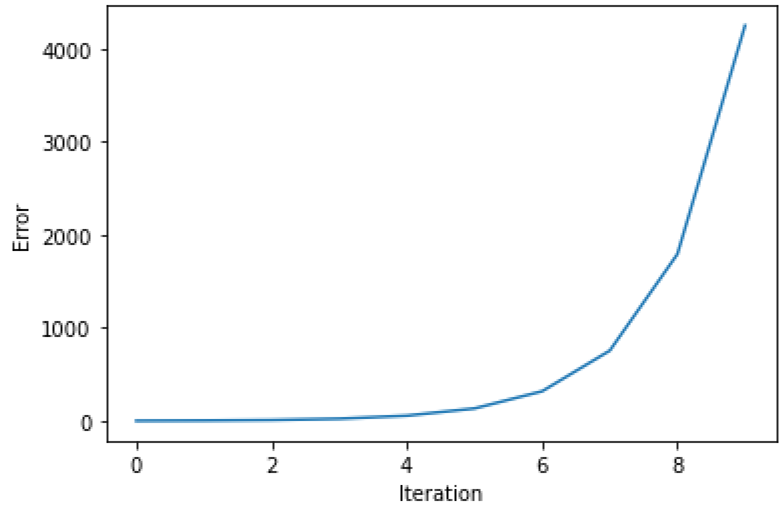
$$\begin{bmatrix} -52.11822961 \\ -42.00119977 \\ -65.00841055 \\ -14.68349959 \end{bmatrix}$$

Iteration 10

$$\begin{bmatrix} -1656.05550041 \\ -1308.50804903 \\ -2052.22839283 \\ -499.14503213 \end{bmatrix}$$

In [87]:

```
import matplotlib.pyplot as plt
plt.plot(diff_x)
plt.ylabel('Error')
plt.xlabel('Iteration')
plt.show()
```



Solution does not converge