MFDS Assignment 1

December 8, 2021

Student Name: Vinayak Nayak

Student ID: 2021FC04135

Student Section: Section 2

Q1.i Find the approximate time your computer takes for a single addition by adding first \$ 10^6 \$ positive integers using a for loop and dividing the time taken by \$ 10 ^ 6 \$. Similarly find the approximate time taken for a single multiplication and division. Report the result obtained in the form of a table.

Performing the operations exactly as mentioned above, the following results are obtained

Operation	Time Taken(s)	Time Taken(ns)
Avg. time per addition	3.4e-8	34
Avg. time per division	5.1e-5	51000
Avg. time per multiplication	10.1e-7	1010

If we were to add/multiply/divide the first 1e6 natural numbers sequentially, we get the above table. One caveat here is that multiplication becomes very tricky when multiplying first 1e6 numbers because we're essentially computing the factorial of one million which is a humongous number. It will be several thousand orders of magnitude of 10 and computing that is going to be extremely time consuming considering multiplication is just repeated addition.

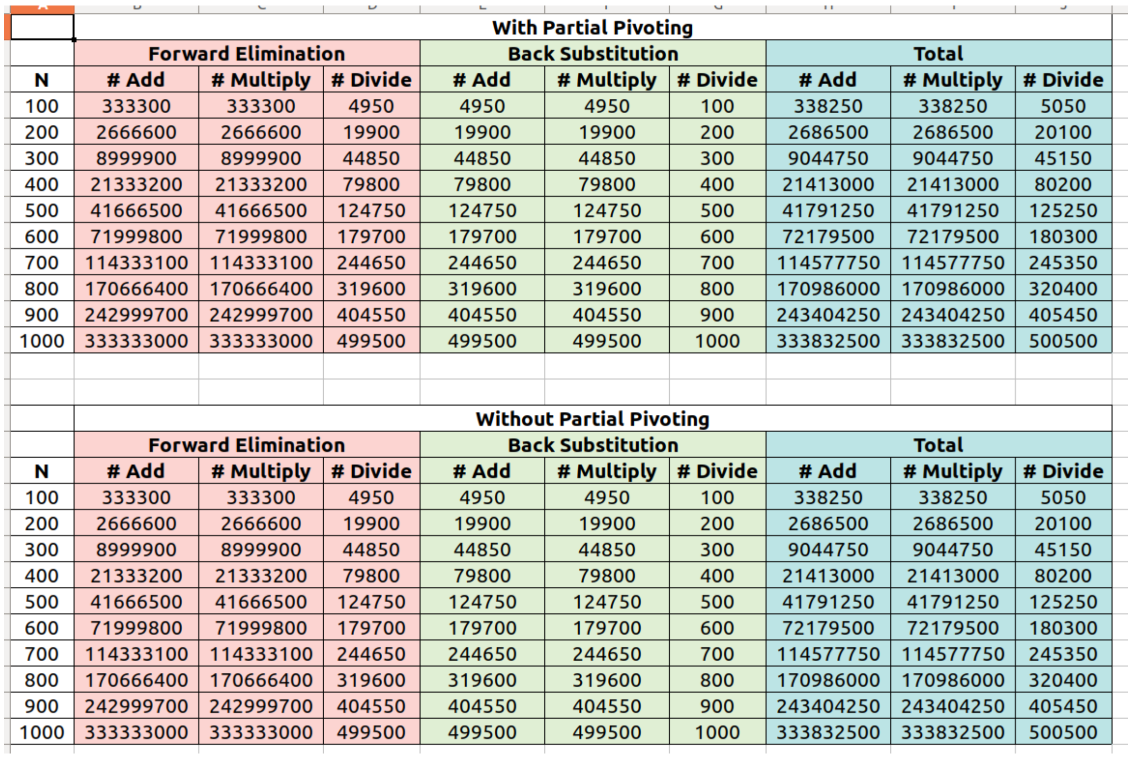
So, I don't think this is a very just estimate of the average time taken for multiplication for any two numbers which are individually within 6 orders of magnitude of 10. So, alternatively, I tried to simulate the process of randomly picking pairs of numbers between 1 and 1e6 for 1e6 times and then computed the time taken per operation which comes out to be as follows. I think this practically makes more sense as it is performing the operations on natural numbers which might actually be used in everyday calculations (to the best of my knowledge that is and, I am open to stand corrected; this is just an aside) rather than performing these operations on operands with huge orders of magnitude (of 10) and then averaging that time out.

Operation	Time Taken(s)	Time Taken(ns)
Avg. time per addition	1.8e-8	18
Avg. time per division	9.9e-8	99
Avg. time per multiplication	11.1e-8	111

Q1.ii Write a function to implement Gauss elimination with and without pivoting. Also write the code to count the number of additions, multiplications and divisions performed during Gaussian elimination.Ensure that the Gauss elimination performs 5S arithmetic which necessitates 5S arithmetic rounding for every addition, multiplication and division performed in the algorithm. If this is not implemented correctly, the rest of the answers will be considered invalid.Note that this is not same as simple 5 digit rounding at the end of the computation. Do not hardwire 5S arithmetic in the code and use dS instead. The code can then be run with various values of d.

```python
import math
from typing import List

def pivot(a: List, k:int) -> List:
    """A function to perform partial pivoting for an array

    Args:
        a (List): [The input matrix as a list of lists]
        k (int): [Which row index to pivot]

    Returns:
        List: [Matrix after partial pivoting]
    """

    interested_col = []

    n = len(a) # Number of rows
    for idx in range(k, n):
        interested_col.append(a[idx][k])

    pivot_row_position = interested_col.index(max(interested_col)) + k

    a[k], a[pivot_row_position] = a[pivot_row_position], a[k]
    return a

def custom_round(a: float, sig_dig:int = 5) -> float:
    """[Takes an array and keeps only sig_dig digits in the array elements]

    Args:
        a (float): [A floating point number]
        sig_dig (int, optional): [Number of significant digits to
                                  which to round to]. Defaults to 5.

    Returns:
        float: [Number rounded to sig_dig significant digits]
    """

    if a != 0.:
        to_round_to = int(math.floor(math.log10(abs(a))))
        rounded_number =  round(a, sig_dig - to_round_to - 1)
    else:
        rounded_number = 0.
    return rounded_number

def forward_elimination(a: List,  sig_dig:int = 10, print_progress:bool = True,
                        pivot_flag: bool = False) -> List:
    """Takes in a matrix and reduces it to an upper triangular matrix
           using forward elimination

    Args:
        a (List): [Input matrix]
        sig_dig (int): [How many significant digits to consider]
        pivot_flag (bool): [Whether or not to pivot the rows when reducing them]
        print_progress (bool): [Whether or not to print the number of primitive operations]

    Returns:
        List: [REF of the given matrix]
    """
    # Create a copy of a and get it's dimensionality
    nr = len(a)
    nc = nr + 1

    # Pivot if the argument for pivoting is True
    if pivot_flag: a = pivot(a, 0)

    divisions, multiplications, additions = 0, 0, 0

    # Iterate for all rows
    for r in range(1, nr):

        # Make all the elements below leading diagonal zeros
        for r_inner in range(r, nr):
            # Create a substitute row and initialize it to all zeros as a placeholder
            # for the row transformation operation
            substitute_row = [0.0] * (nc)
            scale_factor = a[r_inner][r - 1] / a[r - 1][r - 1]
            divisions += 1

            for idx in range(r, nc):
                substitute_row[idx] = a[r_inner][idx] -  scale_factor * a[r - 1][idx]

                # Round each entry to sig_dig number of significant digits
                substitute_row[idx] = custom_round(substitute_row[idx], sig_dig)

                # Add one multiplication and addition each for every element update
                multiplications += 1
                additions += 1
            a[r_inner] = substitute_row
        if pivot_flag: a = pivot(a, r)

    if print_progress:
        print(f"Forward Elimination\n#Additions: {additions:10d}\
              \t#Multiplications: {multiplications:10d}\
              \t#Divisions{divisions:10d}")
    return a


# Backward substitution
def back_substitution(a: List, print_progress:bool = True, sig_dig:int = 5) -> List:
    """Takes in an array in row reduced echelon form and performs back
       substitution for getting the solution to the system of linear equations

    Args:
        fw_a (List): [Row reduced array (REF) of augmented matrix of
                      a system of linear equations]
        print_progress (bool): [Whether to print the number of additions,
                                multiplications, divisions or not]
        sig_dig (bool): [The number of significant digits to consider
                         when carrying out the arithmetic operations]

    Returns:
        List: [Solution]
    """
    nr = len(a)

    # Initialize all the solutions to be zeros
    solutions = [0.] * nr
    divisions, additions, multiplications = 0, 0, 0

    # Start from last row and go upto the first
    for r in range(nr - 1, -1, -1):
        # Compute the sum of product of columns on LHS
        dp = 0.0
        for idx in range(r + 1, nr):
            dp = dp + custom_round(a[r][idx] * solutions[idx], sig_dig)
            additions = additions + 1
            multiplications = multiplications + 1

        # Compute the rhs
        rhs = custom_round(a[r][nr] - dp, sig_dig)
        # Find the rth element of the solutions array
        solutions[r] = custom_round(rhs / a[r][r], sig_dig)
        divisions += 1

    if print_progress:
        print(f"Backward Substitution\n#Additions: {additions:10d}\
                \t#Multiplications: {multiplications:10d}\
                \t#Divisions{divisions:10d}")
    return solutions
```

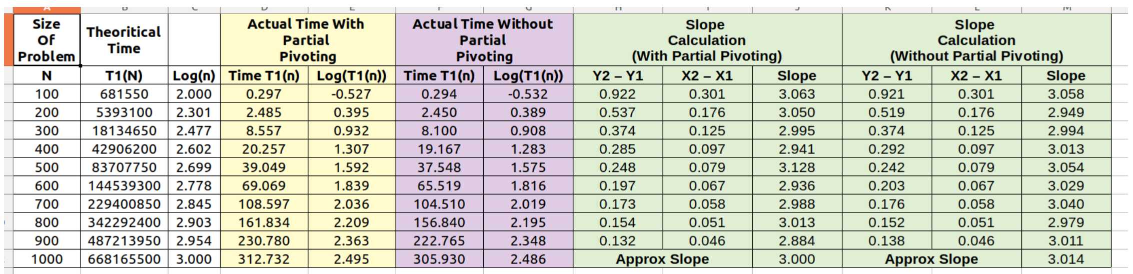
Q1.iii: Generate random matrices of size n×n where \$ n= 100,200, . . . ,1000. \$ Also generate a random b R^n for each case. Each number must be of the form m.dddd(Example : 4.5444) which means it has 5 Significant digits in total. Perform Gaussian elimination with and without partial pivoting for each n value (10 cases) above. Report the number of additions, divisions and multiplications for each case in the form of a table. No need of the code and the matrices / vectors.



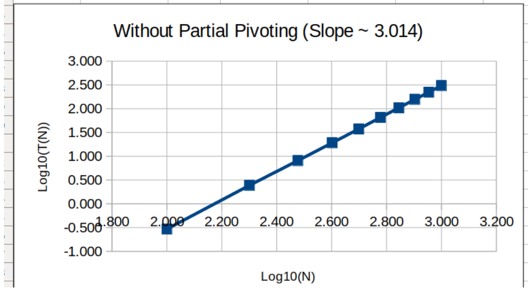
	With Partial Pivoting								
	Forward Elimination			Back Substitution			Total		
N	# Add	# Multiply	# Divide	# Add	# Multiply	# Divide	# Add	# Multiply	# Divide
100	333300	333300	4950	4950	4950	100	338250	338250	5050
200	2666600	2666600	19900	19900	19900	200	2686500	2686500	20100
300	8999900	8999900	44850	44850	44850	300	9044750	9044750	45150
400	21333200	21333200	79800	79800	79800	400	21413000	21413000	80200
500	41666500	41666500	124750	124750	124750	500	41791250	41791250	125250
600	71999800	71999800	179700	179700	179700	600	72179500	72179500	180300
700	114333100	114333100	244650	244650	244650	700	114577750	114577750	245350
800	170666400	170666400	319600	319600	319600	800	170986000	170986000	320400
900	242999700	242999700	404550	404550	404550	900	243404250	243404250	405450
1000	333333000	333333000	499500	499500	499500	1000	333832500	333832500	500500

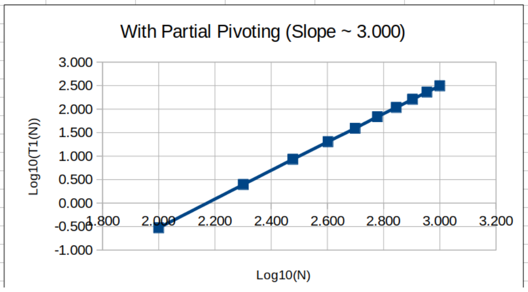
	Without Partial Pivoting								
	Forward Elimination			Back Substitution			Total		
N	# Add	# Multiply	# Divide	# Add	# Multiply	# Divide	# Add	# Multiply	# Divide
100	333300	333300	4950	4950	4950	100	338250	338250	5050
200	2666600	2666600	19900	19900	19900	200	2686500	2686500	20100
300	8999900	8999900	44850	44850	44850	300	9044750	9044750	45150
400	21333200	21333200	79800	79800	79800	400	21413000	21413000	80200
500	41666500	41666500	124750	124750	124750	500	41791250	41791250	125250
600	71999800	71999800	179700	179700	179700	600	72179500	72179500	180300
700	114333100	114333100	244650	244650	244650	700	114577750	114577750	245350
800	170666400	170666400	319600	319600	319600	800	170986000	170986000	320400
900	242999700	242999700	404550	404550	404550	900	243404250	243404250	405450
1000	333333000	333333000	499500	499500	499500	1000	333832500	333832500	500500

Q1.iv Using the code determine the actual time taken for Gaussian elimination with and without partial pivoting for the 10 cases and compare this with the theoretical time. Present this data in a tabular form. Assuming T1(n) is the actual time calculated for an n×n matrix, plot a graphs of log(T1(n)) vs log(n) (for the 10 cases) and fit a straight line to the observed curve and report the slope of the lines. Ensure that separate graphs are to be plotted for the method with and without partial pivoting.



Size Of Problem	Theoritical Time		Actual Time With Partial Pivoting		Actual Time Without Partial Pivoting		Slope Calculation (With Partial Pivoting)			Slope Calculation (Without Partial Pivoting)		
N	T1(N)	Log(n)	Time T1(n)	Log(T1(n))	Time T1(n)	Log(T1(n))	Y2 – Y1	X2 – X1	Slope	Y2 – Y1	X2 – X1	Slope
100	681550	2.000	0.297	-0.527	0.294	-0.532	0.922	0.301	3.063	0.921	0.301	3.058
200	5393100	2.301	2.485	0.395	2.450	0.389	0.537	0.176	3.050	0.519	0.176	2.949
300	18134650	2.477	8.557	0.932	8.100	0.908	0.374	0.125	2.995	0.374	0.125	2.994
400	42906200	2.602	20.257	1.307	19.167	1.283	0.285	0.097	2.941	0.292	0.097	3.013
500	83707750	2.699	39.049	1.592	37.548	1.575	0.248	0.079	3.128	0.242	0.079	3.054
600	144539300	2.778	69.069	1.839	65.519	1.816	0.197	0.067	2.936	0.203	0.067	3.029
700	229400850	2.845	108.597	2.036	104.510	2.019	0.173	0.058	2.988	0.176	0.058	3.040
800	342292400	2.903	161.834	2.209	156.840	2.195	0.154	0.051	3.013	0.152	0.051	2.979
900	487213950	2.954	230.780	2.363	222.765	2.348	0.132	0.046	2.884	0.138	0.046	3.011
1000	668165500	3.000	312.732	2.495	305.930	2.486	Approx Slope		3.000	Approx Slope		3.014





Q2.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.

Algorithm:

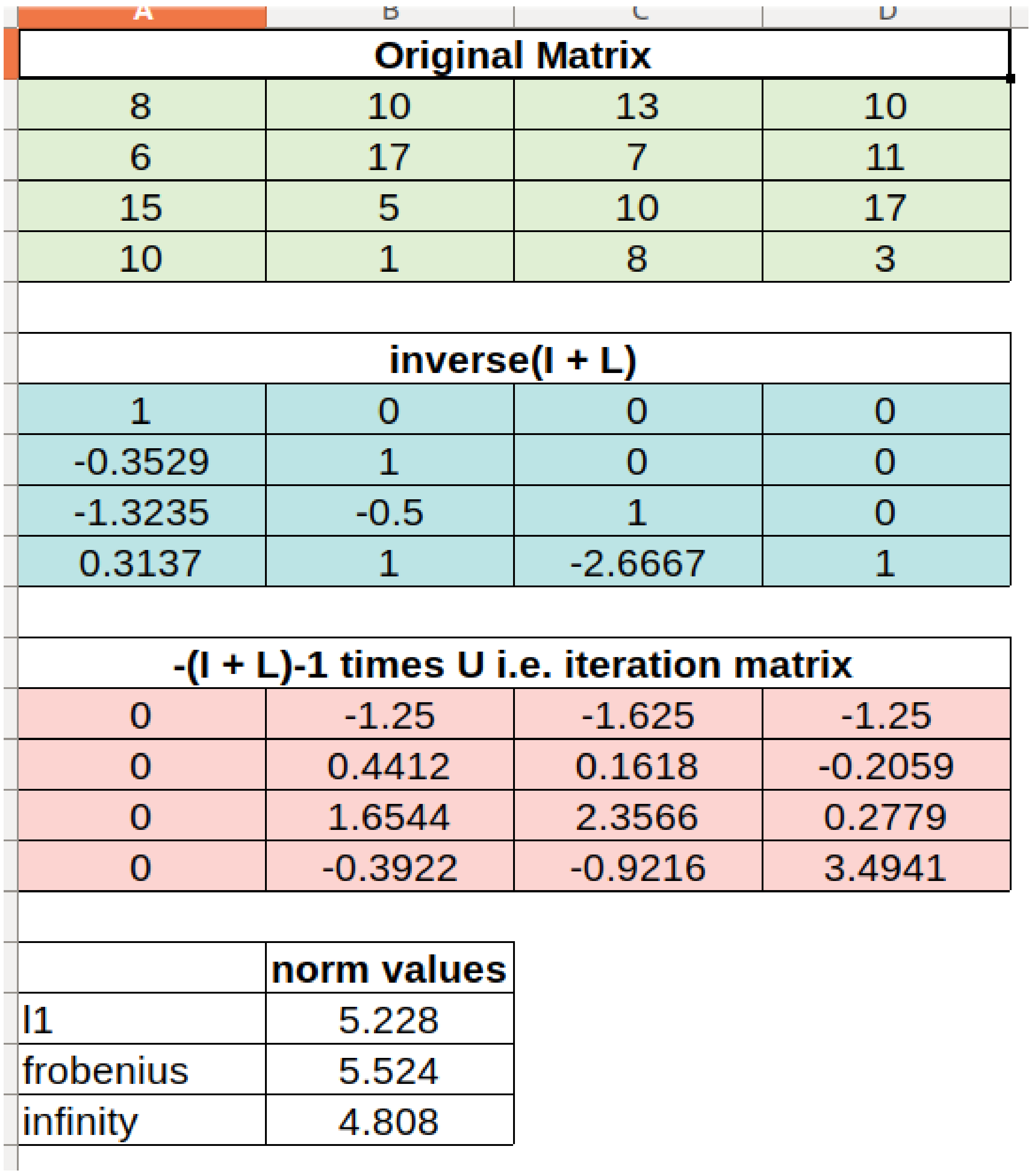
- Find the maximum in each row.
- If any row has multiple maxes, it can never be DD
- If the maxima in any row are in the same col, it can never be DD
- If the sum of all elements except the maxima elements in any one row exceeds the maxima element, then it can never be DD.
- If all checks have passed above, and it is not DD, then we can make the matrix DD

Algorithm Credit https://www.mathworks.com/matlabcentral/answers/511902-making-a-matrix-strictly-diagonally-dominant#answer_421082

```python
from typing import List, Tuple

def check_row(matrix: List) -> Tuple:
    """[Given a square matrix, check the rowwise sums, if any row
        has multiple maxes, it can never be a DD matrix]

    Args:
        matrix (List): [A list of lists]

    Returns:
        Tuple: [A tuple of two elements - A list of max indices,
                A boolean which has checked if any row has two maxes]
    """

    # Initialize an empty container for the max indices
    max_indices = []

    # Initialize a Flag which indicates if the matrix could be diagonally dominant
    DD_flag = True

    for row in matrix:
        # Get the max element of the row
        r_max = max(row)

        # Check for the columns which have the max element
        temp = []
        for idx, element in enumerate(row):
            if element == r_max:
                temp.append(idx)

        # If there's more than one max per row, then it can never be a DD matrix
        if len(temp) > 1:
            DD_flag = False
            break
        max_indices.append(temp[0])

    return (max_indices, DD_flag)


def check_row_sum(matrix: List, indices: List) -> bool:
    """[This function checks the possibility of a diagonal dominance in each row]

    Args:
        matrix (List): [A square matrix]
        indices (List): [A list of index for checking which
                         row in the matrix has which diagonal dominance]

    Returns:
        bool: [If the list of provided rows has diagonal dominance]
    """
    n = len(matrix)

    # Iterate over the index and matrix together
    for i, r in zip(indices, matrix):
        # Get a slice of the row without the diagonal element
        sub_row = r[:i] + r[(i+1):]
        # If sum of absolute values of all other elements is more
        # than the diagonal element, then return False
        if r[i] <= sum([abs(x) for x in sub_row]):
            return False

    return True

def check_diagonal_dominance(matrix: List):
    """[Whether or not it is diagonally dominant or could be made diagonally dominant]

    Args:
        matrix (List): [A list of lists i.e. Matrix]
    """
    n = len(matrix)
    indices, decision = check_row(matrix)

    if not decision:
        print("This matrix can never be made diagonally dominant. \
               One or more rows has repeated maximae")
    elif indices == list(range(0, n)):
        print("This matrix is already diagonally dominant")
    elif len(set(indices).difference(set(range(0, n)))) != 0:
        print("This matrix can never be made diagonally dominant. \
               Multiple rows have diagonal dominance in the \
               same column position")
```

```python
elif check_row_sum(matrix, indices):
    print(f"This matrix is not diagonally dominant, but it \
            could be made diagonally dominant in the following \
            row ordering {indices}")
else:
    print(f"This matrix cannot be made diagonally dominant. \
            One or more rows don't satisfy \
            `Diagonal Element > Sum of all non-diagonal elements`")
```

Q2.ii Write a function to generate Gauss Seidel iteration for a given square matrix. The function should also return the values of 1, inf 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.



Original Matrix

8	10	13	10
6	17	7	11
15	5	10	17
10	1	8	3

inverse(I + L)

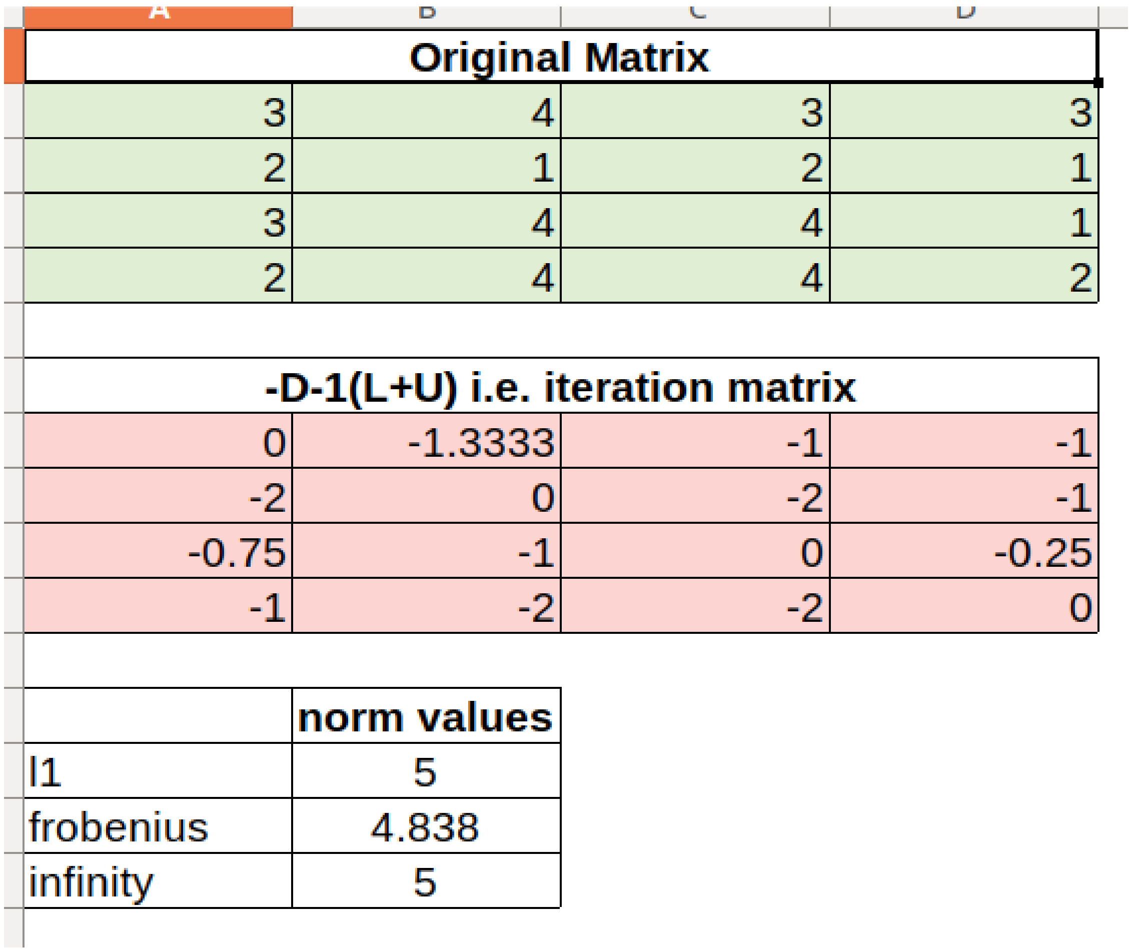
1	0	0	0
-0.3529	1	0	0
-1.3235	-0.5	1	0
0.3137	1	-2.6667	1

-(I + L)-1 times U i.e. iteration matrix

0	-1.25	-1.625	-1.25
0	0.4412	0.1618	-0.2059
0	1.6544	2.3566	0.2779
0	-0.3922	-0.9216	3.4941

	norm values
l1	5.228
frobenius	5.524
infinity	4.808

Q2.iii: Repeat part (ii) for the Gauss Jacobi iteration

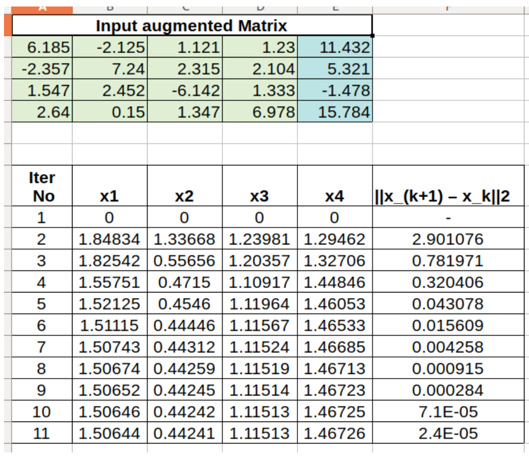


Original Matrix			
3	4	3	3
2	1	2	1
3	4	4	1
2	4	4	2

-D-1(L+U) i.e. iteration matrix			
0	-1.3333	-1	-1
-2	0	-2	-1
-0.75	-1	0	-0.25
-1	-2	-2	0

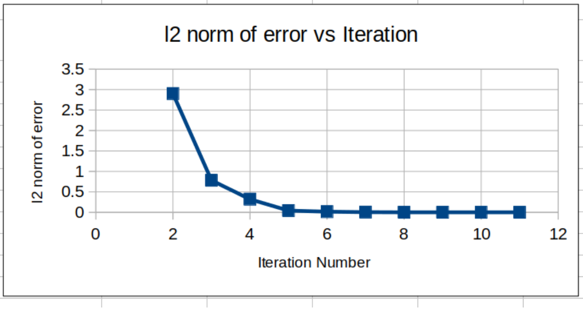
	norm values
l1	5
frobenius	4.838
infinity	5

Q2.iv: Write a function that perform Gauss Seidel iterations. Generate arandom 4×4 matrix A and a suitable random vector $ b 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.

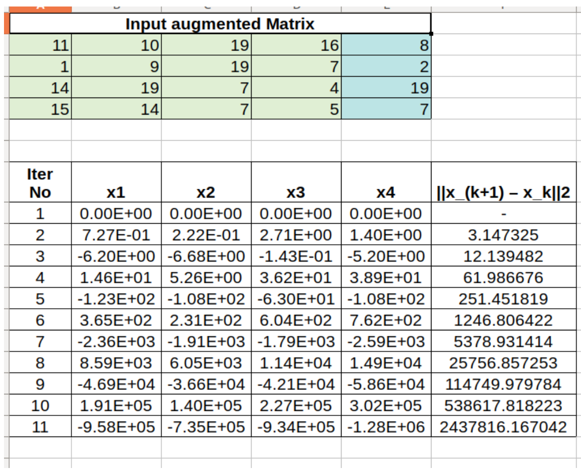


Input augmented Matrix				
6.185	-2.125	1.121	1.23	11.432
-2.357	7.24	2.315	2.104	5.321
1.547	2.452	-6.142	1.333	-1.478
2.64	0.15	1.347	6.978	15.784

Iter No	x1	x2	x3	x4	||x_(k+1) – x_k||2
1	0	0	0	0	-
2	1.84834	1.33668	1.23981	1.29462	2.901076
3	1.82542	0.55656	1.20357	1.32706	0.781971
4	1.55751	0.4715	1.10917	1.44846	0.320406
5	1.52125	0.4546	1.11964	1.46053	0.043078
6	1.51115	0.44446	1.11567	1.46533	0.015609
7	1.50743	0.44312	1.11524	1.46685	0.004258
8	1.50674	0.44259	1.11519	1.46713	0.000915
9	1.50652	0.44245	1.11514	1.46723	0.000284
10	1.50646	0.44242	1.11513	1.46725	7.1E-05
11	1.50644	0.44241	1.11513	1.46726	2.4E-05

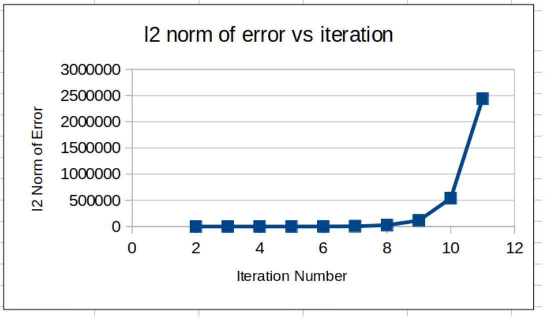


Q2.v: Repeat part (iv) for the Gauss Jacobi method.



Input augmented Matrix				
11	10	19	16	8
1	9	19	7	2
14	19	7	4	19
15	14	7	5	7

Iter No	x1	x2	x3	x4	||x_(k+1) – x_k||2
1	0.00E+00	0.00E+00	0.00E+00	0.00E+00	-
2	7.27E-01	2.22E-01	2.71E+00	1.40E+00	3.147325
3	-6.20E+00	-6.68E+00	-1.43E-01	-5.20E+00	12.139482
4	1.46E+01	5.26E+00	3.62E+01	3.89E+01	61.986676
5	-1.23E+02	-1.08E+02	-6.30E+01	-1.08E+02	251.451819
6	3.65E+02	2.31E+02	6.04E+02	7.62E+02	1246.806422
7	-2.36E+03	-1.91E+03	-1.79E+03	-2.59E+03	5378.931414
8	8.59E+03	6.05E+03	1.14E+04	1.49E+04	25756.857253
9	-4.69E+04	-3.66E+04	-4.21E+04	-5.86E+04	114749.979784
10	1.91E+05	1.40E+05	2.27E+05	3.02E+05	538617.818223
11	-9.58E+05	-7.35E+05	-9.34E+05	-1.28E+06	2437816.167042


l2 norm of error vs iteration
l2 Norm of Error
Iteration Number