solutionWeek2

Aman Kumar EE21B013 <ee21b013@smail.iitm.ac.in>

February 8, 2023

1 Guide to run the program :

- I have written programs splitwise into different functions serially, so run the programs serially -It may cause some error if the program is not run in a serial manner because some functions are using previously written function to do some calculation.
- This is a simple jupyter notebook file anyone can run it in their local machine(python installed as well as jupyter notebook) as well on jupyter notebook server.
- For running the program Just run this file in jupyter notebook and run the program serially.
- Some library used : **math, numpy, cmath**

1.1 Problem statement :

- Write a function to find the factorial of N (N being an input) and find the time taken to compute it. This will obviously depend on where you run the code and which approach you use to implement the factorial. Explain your observations briefly.

[38]:
```
#iterative approach to finding the factorial of a natural no
import math as mt
import numpy as np
n=1
#Iterative function to Calculate Factorial
def IterFact(n):
    try:
        no=int(n)
        if(no<0):
            return("ERROR! can't find the factorail of a negative number")

        if(no==0):
            return 1
        if(no==1):
            return 1
        ans=1
        cnt=1
        while(cnt<=no):
            ans=ans*cnt
            cnt=cnt+1

        return ans
```

```
    except ValueError:
        print("ERROR! Please enter a valid input")

#recursive method to find the factorial of a number
def RecFact(n):
    try:
        no=int(n)
        if(no<0):
            return("ERROR! can't find the factorail of a negative number")
        if(no==0 or no==1):
            return 1
        return (no)*RecFact(no-1)
    except ValueError:
        print("ERROR! Enter a valid input")


#taking the input in fact_No
fact_No=input('Enter no : ')
try:
    n=int(fact_No)
    if(n<0):
        print("ERROR! can't find the factorail of a negative number")
    print(f'factrial of {n} by iterative : ',IterFact(n))
    print(f'factrial of {n} by recursive : ',RecFact(n))
except ValueError:
    print('ERROR ! Enter a valid input')
```

[38]:
```
"ERROR! can't find the factorail of a negative number"
```

1.2 Explanation of above code:

Taking the input and after catching the error (if any) the above line of code print the factorial of a given number by two method iterative and recursive.

[25]:
```
print(f'Factorial of 10 by Recursive approach is : ',RecFact(10))
print(f'Time taken by Recursive Method of finding factorial of 10 is = ␣
 ↪',end='')
%timeit RecFact(10)
print(f'Factorial of 10 by Iterative approach is : ',IterFact(10))
print(f'Time taken by Iterative Method of finding factorial of 10 is = ␣
 ↪',end='')
%timeit IterFact(10)
print(f'Factorial of 10 by Inbuilt numpy apprach is : ',np.math.factorial(10))
print(f'Time taken by numpy function for finding factorail of 10 : is = ␣
 ↪',end='')
%timeit np.math.factorial(10)
```

```
Factorial of 10 by Recursive approach is :  3628800
Time taken by Recursive Method of finding factorial of 10 is =  1.01 µs ± 3.48
ns per loop (mean ± std. dev. of 7 runs, 1,000,000 loops each)
Factorial of 10 by Iterative approach is :  3628800
Time taken by Iterative Method of finding factorial of 10 is =  473 ns ± 3.17 ns
per loop (mean ± std. dev. of 7 runs, 1,000,000 loops each)
Factorial of 10 by Inbuilt numpy apprach is :  3628800
Time taken by numpy function for finding factorail of 10 : is =  58.3 ns ±
0.0765 ns per loop (mean ± std. dev. of 7 runs, 10,000,000 loops each)
```

1.3 Explanation and observation of above code :

we can see by running the above predefined function and the inbuilt library function in numpy the time taken to calculate the factorail of a given no (here it it 10 to test) * The time taken by by recursive method of finding factorial is larger than both the iterative and inbuilt mthod in numpy * The time taken by inbuilt mehtod of numpy is least

2 Problem 2(part a):

Write a linear equation solver that will take in matrices A and b as inputs, and return the vector x that solves the equation $Ax = b$. Your function should catch errors in the inputs and return suitable error messages for different possible problems.

2.1 Explanation of functions and how to run the function to solve Ax=b :

I have written three(3) Function (`swap,gaussian,solve`) to solve this problem of solving linear equation.

- Function swap :- It takes parameter as (`row i , row j ,length of row m , matrix A,Matrix b`) and swap row i and row j of both the matrix A and Matrix b.
- function gaussian :- It takes two Matrix(`matrix , solmatrix`) as Parameter and change the matirix into row echlon form and the corresponding solmatrix.
- Function solve :- It takes two matrix as **parameter (`matrix,solmatrix`),where `Ax=b matrix=A and solmatrix=B`** and solve the equation and return the `vector solution` .
- For running the function gaussian swap is needed and for running the function solve gaussian is needed so firstly run the function swap than gaussian than use solve to find solution of equation AX=B , otherwise it may gives some error that some function is not defined.
- Flow of function be like :-
  - solve -> gaussian -> swap ->return from gaussian -> return solution from solve.

[13]:
```
#Function to interchange row i and row j of matrix and solmatrix given as␣
 ↪argument
#First run this function before running the next function
def swap(i,j,m,matrix,solmatrix):
    for ele in range(m):
        cc=matrix[i][ele]
        matrix[i][ele]=matrix[j][ele]
        matrix[j][ele]=cc
```

```
    temp=solmatrix[i]
    solmatrix[i]=solmatrix[j]
    solmatrix[j]=temp
```

2.2 Explanation of swap function :

I am using this functioin inside the gaussian function, whenever I found any pivot is zero, I will call this function to interchange the row so my pivot will be non-zero.

[14]:
```
# Function to convert the given two matrix (matrix and solmatrix) into gaussian␣
 ↪form(uppper traingular matix and the respective value of solmatrix)
def gaussian(matrix,solmatrix):
    n=len(matrix)
    m=len(matrix[0])  #n and m are the rows and columms of the given matrix
    # print('nodex',n,m)
    minele=min(n,m)
    for i in range (minele):
        if(abs(matrix[i][i])<1e-15): # I have used tolerence level as 1e-15 i.e.
 ↪ considering the value of less than 1e-15 as 0.
            matrix[i][i]=0
            for row in range(i+1,n):
                if(abs(matrix[row][i])>1e-15):

                    swap(i,row,m,matrix,solmatrix) # If digonal entry is zero␣
 ↪than using swap function built earlier swap this row with some other␣
 ↪non-zero digonal element rows

      # If digonal element is non-zero than change the matrix by doing forward␣
 ↪ellimination
       if(abs(matrix[i][i])>1e-15):
            divisorno=matrix[i][i]
            solmatrix[i]=solmatrix[i]/divisorno  #making the pivot element␣
 ↪1(digonal element).

            for j in range(i,m):
                matrix[i][j]=matrix[i][j]/divisorno  # normalizing every other␣
 ↪element of row correspondingly.

            for row in range(i+1,n):
                if(abs(matrix[row][i])>1e-15):
                    multiplier=matrix[row][i]
                    solmatrix[row]=solmatrix[row]-multiplier*solmatrix[i] ␣
 ↪#Making corresponding changes in solmatrix

                    #Now normalizing the other row below the pivot row
                    matrix[row][i]=0
                    for k in range(i+1,m):
```

```
                        matrix[row][k]=matrix[row][k]-multiplier*matrix[i][k]
                # elif(matrix[row][i]<=1e-15):
                #     matrix[row][i]=0
        else :
            return

# a=[[1,1,2,1],[2,2,-2,3],[7,8,9,17],[6,1,1,1]]
# b=[14,6,44,16]
```

2.3 Explanation of above gaussian function :

- This function is used to convert `matrix(A)` into row echelon form and changes the `solmatrix(B)` correspondingly.
- I am using the algorithm of checking the `pivot element(digonal element)` and using that i am normalizing the other element of row so that the pivot element will be 1 and it will chanages the other element of the row correspondingly.
- Now moving forward (that is going down in the row) **forward ellimination** and making the element below the pivot to zero and making the corresponding changes to element of that row.
- After these operation the given matrix will be changed into **echlon form**`(upper triangular matrix with pivot element =1)`.

[15]:
```
import numpy as np
# Function to solve two linar equation of form Ax=B , where A=matrix and␣
 ↪B=solmatrix which are passed inside the function as an argument
def solve(matrix,solmatrix):
    solution=[]  # Matrix to store the final result(solution) of variables
    solution.clear()

    gaussian(matrix,solmatrix)  #calling the gaussian function to tranform the␣
 ↪given matrix into echleon form (upper trinagular matrix)
                                # and the respective solmatrix
    cnt=0
    n=len(matrix)
    m=len(matrix[0])
    index=0
    for row in matrix:
        index+=1
        cntNonZeroElement=0
        for ele in row:
            if(ele!=0):
                cntNonZeroElement+=1
        if(cntNonZeroElement>0):
            cnt+=1
        if(cntNonZeroElement==0):
            if(solmatrix[index-1]!=0):
                # print("No solution Exits")
```

```
                return "No solution Exits"

    # checkint condition for infinite solution
    if(cnt<m):
        # print('Infinte solution')
        return "Infinite solution"

    #checking condition for no solution
    if(cnt>m):
        # print('No solution')
        return "NO solution"

    # if sol exits then finding the solution using BackPropagation Method from Down to Up
    for row in range(m-1,-1,-1):
        ans=0
        l=len(solution)
        l=l-1

        for col in range(m-1,row,-1):

            if(l<0):
                continue
            ans+=solution[l]*matrix[row][col]
            l-=1
        ans=solmatrix[row]-ans
        solution.insert(0,ans)
    # print(solution)
    return solution  #returning the solution matrix
```

```
[-1.666666666666677, -6.666666666666657, 8.166666666666666]
[-1.66666667 -6.66666667  8.16666667]
```

2.4 Explanation and observation of above code :

- This function is basically finding the solution of equatioin `Ax=b` where A,x,b are matrices uisng **Backward_Substitution** method of solving.
- This functionis using the gaussian function to convert the given matrix into row echelon form then it will back propagates to find the solution of matrix.
- As I have used the gaussian ellimination technique to solve the linear equation of form AX=B where A,X and B are matrices.
- Using this technique it may fail in some cases as i am dividing the row by pivot element if the pivot element is too small it may go unbounded and may will give some unusual output one of the example is :
- a=[[7,9,10],[4,5,6],[11,14,16]] b=[10,9,19]

– it is giving some unusual solution but the solution of this matrix is Infinite because the given matrix is singular.

3 Problem 2(Part b):

- Time your solver to solve a random 10×10 system of equations. Compare the time taken against the `numpy.linalg.solve` function for the same inputs.

3.1 Explanation :

- I have generated matrix a(10x10) ->size=10x10 and matrix b(10)->size=10 using numpy function (np.random.rand) and printed the solution solved by function **solve(made by me)** and by Inbuilt function **np.linagl.solve from numpy** and also Printed the time taken by both the function .

[39]:
```
low1=-100000
high1=100000
a=np.random.rand(10,10)
b=np.random.rand(10)
print('solution by my function solve : ',solve(a,b))
print('Time taken by my written function solve is : ',end='')
%timeit solve(a,b)
print('Solution by Inbuilt function np.linalg.solve is : ',end='')
print(np.linalg.solve(a,b))
print('Time taken by Inbuilt numpy function np.linalg.solve is :',end='')
%timeit np.linalg.solve(a,b)
```

```
solution by my function solve :  [-0.3209516733692859, 0.22661891283885893,
-0.7393175449582505, 0.4661170018549267, -1.7108230163572316, 0.896411871154196,
0.4976775405335111, 1.601840915495122, 0.11987420842865129, -0.8232685763259588]
Time taken by my written function solve is : 43.8 µs ± 379 ns per loop (mean ±
std. dev. of 7 runs, 10,000 loops each)
Solution by Inbuilt function np.linalg.solve is : [-0.32095167  0.22661891
-0.73931754  0.466117   -1.71082302  0.89641187
  0.49767754  1.60184092  0.11987421 -0.82326858]
Time taken by Inbuilt numpy function np.linalg.solve is :11.7 µs ± 1.25 µs per
loop (mean ± std. dev. of 7 runs, 100,000 loops each)
```

3.2 Conclusion from above result :

- As we can see the result given by both the solver is approximately equal but the time taken by the Inbuilt function **np.linalg.solve** is less than the time taken by **function solve** to solve the **same (10 x 10) matrix**.But there is not much difference in time taken between the inbuilt function and my function.The inbuilt function is approximately 4 times faster than my function solve in solving 10 x 10 Matrix.

4 Problem Statement 3:

- Given a circuit netlist in the form described above, read it in from a file, construct the appropriate matrices, and use the solver you have written above to obtain the voltages and currents in the circuit. If you find AC circuits hard to handle, first do this for pure DC circuits, but you should be able to handle both voltage and current sources.

5 Brief Explanation of Function I have written to solve this problem.

- I have made the function **Ultimate_MNA(filename)** -> take a file having circuit related data and process the data and gives the output as the circuit unknown variables like (voltages and current through each voltage source).
- I have commented in the required line wherever i think some comment is required for more readablity.
- I have done analysis of ac and dc circuit seperatly using if and else statement and by checking some ac and dc flags.
- I have done analysis for only single frequency circuit(as sir told us).
- If there are multiple frequency, I am giving out error as multiple frequency present.
- for dc analysis I have considered only resistive and capacitive circuit.
- I have also tested my program on various test cases and i am also attaching those test cases file and output.
- I have used simple algorithm of forming the MNA matrix and solve the circuit.
- **Imported library: cmath**
- **Convention and assumption taken** :
  - In .netlist file **first_Node (+ve terminal) , Second_Node (-ve terminal)**
  - For current source direction : **from first_Node -> to second_Node**
  - angles given assuming in - **Degree**
  - Frequency given in **Hz**
  - The impedence of capacior is 1/jwC.
  - The impedence of Inductor is jwL.
  - The equations are solved in complex form.
  - The final solution is in the form of **Phasors** magnitude and theta
  - Printing the Node_voltages and current through each voltage source
- **Function flow** :-
  - call Ultimate_MNA {process data} => call solve() => call gaussian {convert mnamatrix into echelon form} =>call swap() - return from gaussian-> solve matrix ->return solution vector to UltimateMNA {take the solution vector and print the voltages and current}.
- **OUTPUT** :-
  - The function doesnot return anything it will print the value of node_voltages and current through each voltage source.

[19]:
```
# This function will read the given file and calculate the node voltages and␣
 ↪current through the voltages.It will take only the file name as
# input
import cmath
```

```
def Ultimate_MNA(filename):
    import math as mt
    CIRCUIT = '.circuit' # Defining these to easy to handle
    END = '.end'
    AC='.ac'
    start=-1 #start and end index will store the starting and ending index of␣
 ↪line of required data.
    end=-1
    is_ac=False
    with open(filename,'r') as mainfile:

        content_of_mainfile=mainfile.readlines() #storing the content of file␣
 ↪in content_of_mainfile
        for each_line in content_of_mainfile:

            if(CIRCUIT==each_line[:len(CIRCUIT)]):
                start=content_of_mainfile.index(each_line)

            if(END==each_line[:len(END)]):
                end=content_of_mainfile.index(each_line)

        #check for invalid file declearation
        if(start>end):
            print("Invalid declaration of circuit")
            exit()
        if(start==0 and end==0):
            print('No circuit found')
            exit()
        if(start==-1 or end==-1):
            print('Invalid declearation of circuit')
            exit()


        cnt_freq=set()
        womega=0    #frequency of ac source
        for each_line in content_of_mainfile:
            if(len(each_line.split())==0):
                continue
            if(each_line.split()[0]==AC):
                womega=float(each_line.split()[2])  # if there is ac source␣
 ↪than assigning frequency to womega
                cnt_freq.add(each_line.split()[2])

        #check for if there is more than one frequency source in the file given
        womega=womega*mt.pi*2 #Multiplying through 2*pi.
```

```
        # If more than one frequency return because we are not dealing with␣
↪multiple frequency.
        if(len(cnt_freq)>1):
            print("More than one frequency")
            return
        #for counting the no of nodes in the circuit=len(st)
        # print(womega)
        # return

        st=set()  # for counting of different nodes in the file
        cnt_dc=0  #these cnt_dc and cnt_ac are flags that are checking if there␣
↪is ac or dc or both source together.
        cnt_ac=0
        for index in range(start+1,end):
            list=[word for word in content_of_mainfile[index].split('#')[0].
↪split()]
            if(list[1]!='GND'):
                st.add(list[1])
            if(list[2]!='GND'):
                st.add(list[2])
            if(list[3]=='ac'):
                cnt_ac=1
            if(list[3]=='dc'):
                cnt_dc=1


        # If cnt_ac and cnt_dc both are 1 means It's is also a multiple␣
↪frequency than return from here.
        if(cnt_ac and cnt_dc):
            print('Ac and Dc source together (Multiple Frequency)')
            return
        #If the circuit is ac than execute this if statement it contains the␣
↪ac_analysis.
        if(cnt_ac):

            no_of_node=len(st) #storing the no of unique nodes.

            storage={}  #storage is used to store different types of key value␣
↪pair where key->different element of ckt and value-> is the data related to␣
↪that element
            storage.clear()
            voltage=[] # Storage for different voltage source
            voltage.clear()
            cnt_of_voltage_source=0
            for index in range(start+1,end):
```

```
            list=[word for word in content_of_mainfile[index].split('#')[0].
↪split()] #reading the main_file line by line and storing it in list by␣
↪ignoring if any comment is there in that line
            if(list[0][0]=='V'):
                cnt_of_voltage_source+=1
                voltage.append(list[0])
            if(list[3]!='ac' and list[3]!='dc'):
                list[3]=float(list[3])
            storage[list[0]]=list[1:]

        for ele in storage:
            if(storage[ele][0][0]=='n'):
                storage[ele][0]=storage[ele][0][1:]
            if(storage[ele][1][0]=='n'):
                storage[ele][1]=storage[ele][1][1:]

        for ele in storage:
            if(ele[0]=='L'):
                value=womega*storage[ele][2]#womega is the frequency of ac␣
↪source
                storage[ele][2]=complex(0,value)
            elif(ele[0] =='C'):
                value=1/(womega*storage[ele][2])
                storage[ele][2]=complex(0,-value)

        #making MNAmatrix for solving problem
        size=no_of_node+cnt_of_voltage_source

        cnt=size-cnt_of_voltage_source

        MNAmatrix=[]
        AnsMatrix=[]
        # Initialization of MNAmatrix and AnsMatrix by 0.
        for row in range(size):
            lt=[]
            for col in range(size):
                lt.append(0)
            MNAmatrix.append(lt)
            AnsMatrix.append(0)
        #  Traversing through each of the element in storage and filling up␣
↪the MNAmatrix and AnsMatrix according to MNA algorithm
        for ele in storage:
            if(storage[ele][0]=='GND'):
                storage[ele][0]='0'

            if(storage[ele][1]=='GND'):
                storage[ele][1]='0'
```

```
                if(ele[0]=='R'):
                    a=int(storage[ele][0])
                    b=int(storage[ele][1])
                    if(a!=0 and b!=0):
                        MNAmatrix[a-1][a-1]+=1/float(storage[ele][2])
                        MNAmatrix[a-1][b-1]-=1/float(storage[ele][2])
                        MNAmatrix[b-1][b-1]+=1/float(storage[ele][2])
                        MNAmatrix[b-1][a-1]-=1/float(storage[ele][2])
                    elif(a==0 and b!=0):
                        MNAmatrix[b-1][b-1]+=1/float(storage[ele][2])
                    elif(a!=0 and b==0):
                        MNAmatrix[a-1][a-1]+=1/float(storage[ele][2])
                    else :
                        continue

                elif(ele[0]=='V'):

                    a=int(storage[ele][0])
                    b=int(storage[ele][1])
                    voltageValue=0

                    voltageValue=-1*complex(float(storage[ele][3])*mt.
↪cos(float(storage[ele][4])*(mt.pi/180)),float(storage[ele][3])*mt.
↪sin(float(storage[ele][4])*(mt.pi/180)))
                    AnsMatrix[cnt]+=voltageValue
                    if(a!=0 and b!=0):

                        MNAmatrix[a-1][cnt]+=1
                        MNAmatrix[cnt][a-1]-=1
                        MNAmatrix[b-1][cnt]-=1
                        MNAmatrix[cnt][b-1]+=1
                    elif(a==0 and b!=0):

                        MNAmatrix[b-1][cnt]-=1
                        MNAmatrix[cnt][b-1]+=1

                    elif(a!=0 and b==0):

                        MNAmatrix[a-1][cnt]+=1
                        MNAmatrix[cnt][a-1]-=1
                    else :
                        continue
                    cnt+=1
                elif(ele[0]=='I'):
                    a=int(storage[ele][0])
                    b=int(storage[ele][1])
```

```
                    CurrentValue=0
                    CurrentValue=complex(float(storage[ele][3])*mt.
↪cos(float(storage[ele][4])*(mt.pi/180)),float(storage[ele][3])*mt.
↪sin(float(storage[ele][4])*(mt.pi/180)))
                    if(a>0):
                        AnsMatrix[a-1]=AnsMatrix[a-1]-CurrentValue
                    if(b>0):
                        AnsMatrix[b-1]=AnsMatrix[b-1]+CurrentValue
                elif(ele[0]=='L'):
                    a=int(storage[ele][0])
                    b=int(storage[ele][1])
                    if(a!=0 and b!=0):
                        MNAmatrix[a-1][a-1]+=1/storage[ele][2]
                        MNAmatrix[a-1][b-1]-=1/storage[ele][2]
                        MNAmatrix[b-1][b-1]+=1/storage[ele][2]
                        MNAmatrix[b-1][a-1]-=1/storage[ele][2]
                    elif(a==0 and b!=0):
                        MNAmatrix[b-1][b-1]+=1/storage[ele][2]
                    elif(a!=0 and b==0):
                        MNAmatrix[a-1][a-1]+=1/storage[ele][2]
                    else :
                        continue
                elif(ele[0]=='C'):
                    a=int(storage[ele][0])
                    b=int(storage[ele][1])
                    if(a!=0 and b!=0):
                        MNAmatrix[a-1][a-1]+=1/storage[ele][2]
                        MNAmatrix[a-1][b-1]-=1/storage[ele][2]
                        MNAmatrix[b-1][b-1]+=1/storage[ele][2]
                        MNAmatrix[b-1][a-1]-=1/storage[ele][2]
                    elif(a==0 and b!=0):
                        MNAmatrix[b-1][b-1]+=1/storage[ele][2]
                    elif(a!=0 and b==0):
                        MNAmatrix[a-1][a-1]+=1/storage[ele][2]
                    else :
                        continue

            Final_Solution=solve(MNAmatrix,AnsMatrix)  #solution is stored␣
↪inside Final_Solution array.

            # Now Steps for printing the values of voltages through different␣
↪Nodes and Current through Voltages sources.
            vol_iter=0
            i=0
            for ele in Final_Solution:
                if i<no_of_node:
```

```
                    print(f'Voltage_Node{i+1} :magnitude:  {abs(ele):>10}␣
↪,phase : {cmath.phase(ele)*(180/cmath.pi):>10}')
                    i+=1
                else:
                    print(f'Current_through_Voltage_source {voltage[vol_iter]} :
↪magnitude:  {abs(ele):>10} , phase : <{cmath.phase(ele)*(180/cmath.pi):>10}␣
↪degree>')
                    vol_iter+=1


        #From here Below codes is for DC_Analysis.
        # If cnt_dc is 1 and cnt_ac is 0 that is only dc sources are present so␣
↪do the dc analysis.

        elif(cnt_dc==1 and cnt_ac==0):
            storage={} #It's a dictionary storing different elements with their␣
↪given data.
            resistence=[] #for storing resistences
            voltage=[] #For storing voltages
            storage.clear()
            resistence.clear()
            voltage.clear()
            for index in range (start+1,end):
                list=[word for word in content_of_mainfile[index].split('#')[0].
↪split()] #reading the file and storing inside list and with ignoring the␣
↪comments in line
                # print(list)
                if(list[0][0]=='R'):
                    resistence.append(list[0])
                elif(list[0][0]=='V'):
                    voltage.append((list[0]))
                storage[list[0]]=list[1:]

            for ele in storage:
                if(storage[ele][0][0]=='n'):
                    storage[ele][0]=storage[ele][0][1:] #checking for given␣
↪format of nodes if node is 'n12' than ignore 'n' and store '12'
                if(storage[ele][1][0]=='n'):
                    storage[ele][1]=storage[ele][1][1:]

            st_node=set() #store no of nodes uniquly.
            for ele in storage:
                st_node.add(storage[ele][0])
                st_node.add(storage[ele][1])
            size=len(st_node)-1
            n1=0
```

```
            for ele in storage:
                if(ele[0]=='V'):
                    size+=1
                    n1+=1

            cnt=size-n1
            MNAmatrix=[]  # MNA matrix is the mna matrix ans AnsMatrix is the
↪corresponding the B matrix in Ax=B equation.
            AnsMatrix=[]

            # Initialization of MNAmatrix and AnsMatrix
            for row in range(size):
                lt=[]
                for col in range(size):
                    lt.append(0)
                MNAmatrix.append(lt)
                AnsMatrix.append(0)

            # Traversing through each of the element in storage and filling up
↪the MNAmatrix and AnsMatrix according to MNA algorithm.
            for ele in storage:
                if(storage[ele][0]=='GND'):
                    storage[ele][0]='0'

                if(storage[ele][1]=='GND'):
                    storage[ele][1]='0'
                # If ele in storage is 'R' than filling the MNAmatrix
↪accordingly
                if(ele[0]=='R'):
                    a=int(storage[ele][0])
                    b=int(storage[ele][1])
                    if(a!=0 and b!=0):
                        MNAmatrix[a-1][a-1]+=1/float(storage[ele][2])
                        MNAmatrix[a-1][b-1]-=1/float(storage[ele][2])
                        MNAmatrix[b-1][b-1]+=1/float(storage[ele][2])
                        MNAmatrix[b-1][a-1]-=1/float(storage[ele][2])
                    elif(a==0 and b!=0):
                        MNAmatrix[b-1][b-1]+=1/float(storage[ele][2])
                    elif(a!=0 and b==0):
                        MNAmatrix[a-1][a-1]+=1/float(storage[ele][2])
                    else :
                        continue
                elif(ele[0]=='V'):
                    a=int(storage[ele][0])
                    b=int(storage[ele][1])
                    voltageValue=-1*float(storage[ele][3])
                    AnsMatrix[cnt]=float(voltageValue)
```

```
                    if(a!=0 and b!=0):

                        MNAmatrix[a-1][cnt]+=1
                        MNAmatrix[cnt][a-1]-=1
                        MNAmatrix[b-1][cnt]-=1
                        MNAmatrix[cnt][b-1]+=1
                    elif(a==0 and b!=0):

                        MNAmatrix[b-1][cnt]-=1
                        MNAmatrix[cnt][b-1]+=1

                    elif(a!=0 and b==0):

                        MNAmatrix[a-1][cnt]+=1
                        MNAmatrix[cnt][a-1]-=1
                    else :
                        continue
                    cnt+=1
                elif(ele[0]=='I'):
                    #for current source assumption is current is going from␣
↪nodea to node b i.e a-->b  first_node to second_node in the given .netlist␣
↪file .
                    a=int(storage[ele][0])
                    b=int(storage[ele][1])
                    CurrentValue=float(storage[ele][3])
                    if(a>0):
                        AnsMatrix[a-1]=AnsMatrix[a-1]-CurrentValue
                    if(b>0):
                        AnsMatrix[b-1]=AnsMatrix[b-1]+CurrentValue

            Final_Solution=solve(MNAmatrix,AnsMatrix)  #solution is stored␣
↪inside Final_Solution array.

            # Now Steps for printing the values of voltages through different␣
↪Nodes and Current through Voltages sources.
            vol_iter=0
            node_size=len(st_node)-1
            i=0
            for ele in Final_Solution:
                if i<node_size:
                    print(f'Voltage_Node{i+1} : {ele:>20}')
                    i+=1
                else:
                    print(f'Current_through_Voltage_source {voltage[vol_iter]} :
↪ {ele:>20}')
                    vol_iter+=1
```

```
Ultimate_MNA('Junk_Example.netlist')  #Function calling for reading of file␣
 ↪('Junk Example.netlist') and solving the circuit based on the data

# we can call function like Ultimate_MNA('some.netlist') to read and solve the␣
 ↪circuit.
```

```
Voltage_Node1 :                   4.0
Voltage_Node2 :   6.0000000000001705
Voltage_Node3 :   172.84615384615375
Voltage_Node4 :   -39.07692307692332
Voltage_Node5 :  -29.076923076923237
Voltage_Node6 :                   9.0
Voltage_Node7 :    570.9230769230765
Current_through_Voltage_source V1 :  -0.6615384615384556
Current_through_Voltage_source V2 :  -5.1000000000000085
Current_through_Voltage_source V3 :  -0.7615384615384642
Current_through_Voltage_source V4 :    20.76153846153846
```