EE2703 - Week 2

Ata Karim Subhani <EE21B025>

February 8, 2023

1 Directions

- I have prepared my documentation on LaTeXusing the .tex file that I exported from the Jupyter Notebook. I wrote very little in the markdown section of the notebook.
- To run the code, upload the .ipynb file on Jupyter server, and run as directed
- With this document, I have submitted all the .netlist files that I use, also I have share the .asc file of the two DC circuits and one AC circuit that I used as a test case apart from those uploaded on the moodle.

2 Factorial Computation

I have used **two** methods for factorial computation:

- **Iterative** In this method, I have created a list name `fact`, such `fact[n]` will store factorial of n. For each $n \geq 2$, I have appended the list w with $n \times fact[n-1]$.
- **Recursive** In this approach, I used the recursive relation. In other words, to calculate $factorial(n)$, I return the $n \times factorial(n-1)$. I have given the base condition to be $factorial(0) = 1$ and $factorial(1) = 1$

[37]:
```
import numpy as np
def factorial(N):
    fact = []
    fact.append(1)
    fact.append(1)
    for i in range(2, N+1):
        fact.append(i*fact[i-1])
    return fact[N]

def rec_fact(N):
    if(N==1 or N==0):
        return 1
    else:
        return N*rec_fact(N-1)

%timeit rec_fact(20)
%timeit factorial(20)
%timeit np.math.factorial(10)
```

```
2.02 µs ± 38.9 ns per loop (mean ± std. dev. of 7 runs, 100,000 loops each)
1.92 µs ± 85 ns per loop (mean ± std. dev. of 7 runs, 1,000,000 loops each)
80.7 ns ± 1.34 ns per loop (mean ± std. dev. of 7 runs, 10,000,000 loops each)
```

3 Solving System of Linear Equations

In this problem, our aim is to find the solution of given set of linear equations.

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + a_{13}x_3..... + a_{1n}x_n &= b_1 \\ a_{21}x_1 + a_{22}x_2 + a_{23}x_3..... + a_{2n}x_n &= b_2 \\ a_{31}x_1 + a_{32}x_2 + a_{33}x_3..... + a_{3n}x_n &= b_3 \\ &. \\ &. \\ &. \\ a_{m1}x_1 + a_{m2}x_2 + a_{m3}x_3..... + a_{mn}x_n &= b_m \end{aligned} \tag{1}$$

To solve the above set of linear equations, I have to create matrix a mxn matrix A with elements a_{ij} and b vector with elements b_i. Then solution vector x (if exist) of the given set of linear equation will satisfy the following equation

$$Ax = b \tag{2}$$

There are two possibilities for the solution of above equation:

- **m<n**: Less number of linearly independent equations than number of variables. So, we there can be **infinitely many** solutions possible which can satisfy given system of equations.
- **m>n or m = n**: In this case given system of equations may or may not have solution.

Approach to solve for the 2^{nd} case:

In this, I have converted the augmented matrix [A|b] to **Reduced Echelon Form** using the elementary row transformation. Because of partial pivoting, all the zero rows comes at the bottom of Augmented matrix. Now from the REF of the augmented matrix, I have deduced the nature of solution and then calculated the solution (if exist).

- **Inconsistent System**: If there exist at least one row in matrix A whose all elements are zeros, and corresponding element in b vector is non-zero then the given system of equations are inconsistent.
- **Infinite solution**: If corresponding to all zero rows in A matrix, element in b vector is zero, then count the number of non-zero rows in A matrix. If the number of non-zero rows in A matrix is less than n (i.e number of variables). It means given system has less number of linearly independent equations, than the number of variables, hence there will be infinite solution.
- **Unique Solution**: In this case number of non-zero rows in REF of A must be n, then only we can have n linearly independent equations to uniquely solve the system of equations. We obtain the solution vector x using back substitution.

3.1 Pivoting Technique

Pivoting Technique is basically used to avoid division by zero during elementary row operation. Each time I swap the row with that row which has maximum value in the pivotal column below the pivot element. It could have been done other way also, whenever pivot element is zero, swap the given row with that row which has non-zero element in pivotal column below the pivot element.

I am actually doing partial pivoting, where I have only interchange the row.

[36]:
```
def Partial_Pivot(A, b, k):
    big = A[k][k]
    index = k
    for i in range(k+1, len(A)):
        if (abs(A[i][k]) > abs(big)):
            big = - A[i][k]
            index = i
    if (index != k):
        # swap row number "index" with row number "k"
        for i in range(k, len(A[0])):
            temp = A[k][i]
            A[k][i] = A[index][i]
            A[index][i] = temp
        # swap element of 'b' vector
        temp = b[k]
        b[k] = b[index]
        b[index] = temp
```

3.2 Function to Convert given mxn matrix to Reduced Echelon Form (REF)

This is the routine code for the forward elimination after applying pivoting technique to the pivot element.

[34]:
```
def REF(A, b):
    m = len(A)  # m is no. of rows.
    n = len(A[0])  # n is no. of columns.
    for k in range(n):
        # Pivoting the element.
        Partial_Pivot(A, b, k)
        for i in range(k+1, m):
            factor = A[i][k]/A[k][k]
            b[i] = b[i] - factor*b[k]
            for j in range(k, n):
                A[i][j] = A[i][j] - factor*A[k][j]
```

3.3 Main Solver Function:

This is code where I have consolidated **REF** and **partial_pivoting** function to solve the given system of equations and do the back substitution (if solution exist) or raise the relevant information regarding the givne system of equations.

[4]:
```
def solve(A, b):
    m = len(A)  # m is no. of rows (equivalent to no. of equation)
    n = len(A[0])  # n is no. of columns (equivalent to no. of variable.)

    if (m < n):
        # We have less no. of equations than number of variable.
        return "Insufficient number of linearly independent equations, So there␣
↪are infinite solutions"
    # Convert coefficient matrix to Row Echelon Form.
    REF(A, b)
    # now count the number rows in Row echelon form of coefficient matrix with␣
↪all zero's element.
    NumOfNonZerosRow = 0
    for i in range(m):
        for j in range(n):
            cnt = 0  # no. of non zeros element in a row.
            if (A[i][j] != 0):
                cnt = 1
                break
        if (cnt == 1):  # there is atleast one non zero element in that row
            NumOfNonZerosRow += 1
            continue
        if (cnt == 0):  # all element is zero
            if (b[i] != 0):  # It means that
                return "Inconsistent system, Hence No Solution"
    # Num of non zeros row is the number of linearly independent equations in␣
↪given system of equations
    if (NumOfNonZerosRow < n):
        return "Insufficient number of linearly independent equations, So there␣
↪are infinite solutions"

    # Note: Number of non zeros row can't be less than n provided(m>=n)

    # Back Substitution.
    if (NumOfNonZerosRow == n):
        x = []  # x is final solution vector
        x.clear()
        for i in range(n):
            x.append(0)
        for row in range(n-1, -1, -1):
            sum = 0
            for col in range(row+1, n):
                sum += A[row][col]*x[col]
            x[row] = (b[row]-sum)/A[row][row]
        return x
```

3.4 Solving system with 10 equations and 10 unknowns using own linear equation solver and numpy

[33]:
```
# Test 5 randomly generated 10x10 matrix using solve() and numpy.linalg.solve()

import numpy as np
# np.random.seed(0)
n = 10
for _ in range(1):
    A = np.random.rand(n, n)
    b = np.random.rand(n)
    print(np.linalg.solve(A, b))
    print(solve(A, b))
    print(100*"=")


print(f"Time taken for my solver function is ", end="")
%timeit solve(A, b)
print(f"Time taken for numpy.linalg.solve() function is ", end="")
%timeit np.linalg.solve(A, b)
```

```
[ 2.89885842 -0.4785833  -0.47526755 -1.19773257 -1.68907064 -0.45485954
  1.21197607 -1.71776273  1.40998669  1.31718901]
[2.8988584209161723, -0.47858330194500065, -0.47526755353471484,
-1.197732568854598, -1.6890706428424125, -0.45485954119330585,
1.2119760675203488, -1.717762725161453, 1.4099866934702776, 1.317189014870706]
====================================================================================
====================
Time taken for my solver function is 257 µs ± 4.95 µs per loop (mean ± std. dev.
of 7 runs, 1,000 loops each)
Time taken for numpy.linalg.solve() function is 15.9 µs ± 133 ns per loop (mean
± std. dev. of 7 runs, 100,000 loops each)
```

As we can see time taken by my solve() is 257μs but `numpy.linalg.solve()` is taking only 15.9μs. I think this much discrepency in time is due to the fact that most of the part of `numpy` is written in `C` language which is indeed much faster than `Python`

4 Solving Electrical circuits

AIM: Read netlist file and solve the circuit using matrix solver coded in the last problem.

Note: My code for this begins in sub-section 4.3

First of all, I have checked the nature of sources present in netlist. If I found sources to be DC then I called function **DC_MNA** (sub-section 4.1), If I found source to be only AC then I called function **AC_MNA** (sub-section 4.2). If circuit contains both AC and DC sources, then I flag that circuit contains multiple frequencies.

I have introduced a unknown current I_i through voltage source V_i and nodes voltages as variable. So, total variables are number of nodes + number of voltage sources.

Convention

- Current is flowing from node 1 ($<n_1>$) to node 2 ($<n_2>$) in the following line of netlist $I_1 < n1 > < n2 > < value >$

- Current I_i is flowing from positive terminal to negative terminal of voltage source V_i.

4.1 When only DC sources are present

Parameter of DC_MNA function:

- **content_of_netlist** It is a list which stores all the different lines of netlist files.
- **Start** and **end** are index of .circuit and .end respectively in the netlist file.

In this function, I iterated through all the lines of netlist file and update my conductance matrix and source vector according to the circuit component found in that particular line. After Iterating through all the lines, I solve the conductance matrix, with source vector using my solve() function.

Note: I didn't handle Capacitor or Inductor in the DC circuit.

[19]:
```
def DC_MNA(content_of_netlist, start, end):

    st = set() # creating a list to count no. of nodes.
    Nv = 0  #no. of voltage source
    for line in content_of_netlist[start+1:end]:
        if(len(line.split())==0): # when line is blank
            continue
        elif(line[0]=='#'): # when line is comment
            continue
        else:
            st.add(line.split()[1])
            st.add(line.split()[2])
            if(line[0]=='V'):
                Nv+=1
    #Total no. of variables in circuit i.e size of conductance matrix.
    # Nv = no. of voltage sources, n = no. of nodes
    n = len(st)-1
    size = Nv + n
    A = [] # conductance matrix
    B = []
    #initialising conductance matrix (A) and B vector.
    for row in range(size):
        lt = []
        for col in range(size):
            lt.append(0)
        A.append(lt)
        B.append(0)
        a, b = 0, 0
    for line in content_of_netlist[start+1:end]:
```

```
        if(len(line.split())==0):
            continue
        discription = [word for word in line.split('#')[0].split()]
        # Operations to store node number in a and b.
        if(discription[1]=='GND'):
            discription[1]='n0'
        elif(discription[2]=='GND'):
            discription[2]='n0'
        a, b = discription[1], discription[2] # a and b store nodes information
        if(a[0]=='n'): # when nodes are in form of n1, n2, .....
            a = int(a[1:])
        if(b[0]=='n'):
             b = int(b[1:])
        a, b = int(a), int(b)  # here finally a and b stores node number.

        if(len(line.split())==0): # when line is blank
            continue
        elif(line[0]=='#'): # when line start with comment
            continue
        elif(line[0]=='R'):
            R = float(discription[-1])
            if(a==0 and b!=0):
                A[b-1][b-1] += 1/R
            elif(b==0 and a!=0):
                A[a-1][a-1] += 1/R
            else:
                A[b-1][b-1] += 1/R
                A[a-1][a-1] += 1/R
                A[a-1][b-1] -= 1/R
                A[b-1][a-1] -= 1/R
            continue
        elif(line[0]=='V'):
            x = n + int(discription[0][1:])-1
            B[x] = float(discription[-1]) # updating B vector with value of␣
↪Voltage of source.
            if(a==0 and b!=0):
                A[x][b-1] = -1
                A[b-1][x] = -1
            elif(b==0 and a!=0):
                A[x][a-1] = 1
                A[a-1][x] = 1
            else:
                A[x][b-1] = -1
                A[x][a-1] = 1
                A[a-1][x] = 1
                A[b-1][x] = -1
            continue
```

```
        elif(line[0]=='I'):
            if(a==0 and b!=0):
                B[b-1] += float(discription[-1])
            elif(b==0 and a!=0):
                B[a-1] -= float(discription[-1])
            else:
                B[a-1] -= float(discription[-1])
                B[b-1] += float(discription[-1])
            continue

    x = solve(A,B)
    for i in range(n):
        print(f"Node {i+1} Voltage = {x[i]} V")
    for i in range(Nv):
        print(f"Current through Voltage source V{i+1} = {x[n+i]} A")

    return ""
```

4.2 When only AC sources are present

Parameter of AC_MNA function:

- **content_of_netlist** It is a list which stores all the different lines of netlist files.
- **Start** and **end** are index of .circuit and .end respectively in the netlist file.

In this section, I have first find the number of different frequency source in the circuit. If the number of frequency is more than 1 than I throw the flag "Circuit contains more than one frequency source" and terminate the program.

Else if I find one frequency, then I iterate through all the lines of netlist file and update my conductance matrix. Now conductance matrix is a complex valued matrix. I have incorporated the phase of source using complex number. The result obtained (node voltage and current through the voltage sources) are complex in nature. I have shown then in phasor form in the output.

[20]:
```
from math import pi, sin, cos
import cmath
def AC_MNA(content_of_netlist, start, end):

    freq = set()
    f=0
    for line in content_of_netlist[end+1:]:
        if(line.split()[0]=='.ac'):
            f = float(line.split()[2])
            freq.add(line.split()[2]) #putting frequencies in set "freq"
        if(len(freq)>1):
            return "Circiut Contains more than one frequency source."
    w = 2*pi*f
    # Counting number of nodes and Voltage source.
```

```
    st = set()
    n = 0
    Nv = 0
    for line in content_of_netlist[start+1:end]:
        if(len(line.split())==0): # when line is blank
            continue
        elif(line[0]=='#'): # when line is comment
            continue
        else:
            st.add(line.split()[1])
            st.add(line.split()[2])
            if(line[0]=='V'):
                Nv+=1
    n = len(st)-1 # no. of nodes
    size = n + Nv # size of conductance matrix.
    A = [] # conductance matrix
    B = []
    #initialising conductance matrix (A) and B vector.
    for row in range(size):
        lt = []
        for col in range(size):
            lt.append(0)
        A.append(lt)
        B.append(0)
    a, b = 0, 0
    for line in content_of_netlist[start+1:end]:
        if(len(line.split())==0):
            continue
        discription = [word for word in line.split('#')[0].split()]
        # Operations to store node number in a and b.
        if(discription[1]=='GND'):
            discription[1]='n0'
        elif(discription[2]=='GND'):
            discription[2]='n0'
        a, b = discription[1], discription[2] # a and b store nodes information
        if(a[0]=='n'): # when nodes are in form of n1, n2, .....
            a = int(a[1:])
        if(b[0]=='n'):
             b = int(b[1:])
        a, b = int(a), int(b)  # here finally a and b stores node number.

        if(len(line.split())==0): # when line is blank
            continue
        elif(line[0]=='#'): # when line start with comment
            continue

        elif(line[0]=='R'):
```

```
            if(a==0 and b!=0):
                A[b-1][b-1] += 1/float(discription[-1])
            elif(b==0 and a!=0):
                A[a-1][a-1] += 1/float(discription[-1])
            else:
                A[b-1][b-1] += 1/float(discription[-1])
                A[a-1][a-1] += 1/float(discription[-1])
                A[a-1][b-1] -= 1/float(discription[-1])
                A[b-1][a-1] -= 1/float(discription[-1])
            continue

        elif(line[0]=="L"):
            Z = complex(0, float(discription[-1])*w)

            if(a==0 and b!=0):
                A[b-1][b-1] += 1/Z
            elif(b==0 and a!=0):
                A[a-1][a-1] += 1/Z
            else:
                A[b-1][b-1] += 1/Z
                A[a-1][a-1] += 1/Z
                A[a-1][b-1] -= 1/Z
                A[b-1][a-1] -= 1/Z
            continue
        elif(line[0]=="C"):
            Z = complex(0, 1/(float(discription[-1])*w))

            if(a==0 and b!=0):
                A[b-1][b-1] += 1/Z
            elif(b==0 and a!=0):
                A[a-1][a-1] += 1/Z
            else:
                A[b-1][b-1] += 1/Z
                A[a-1][a-1] += 1/Z
                A[a-1][b-1] -= 1/Z
                A[b-1][a-1] -= 1/Z
            continue
        elif(line[0]=='V'):
            x = n + int(discription[0][1:])-1
            V_mag = float(discription[-2])
            phase = float(discription[-1])
            B[x] = complex(V_mag*cos(phase*(pi/180)), V_mag*sin(phase*(pi/180)))␣
↪# updating B vector with value of Voltage of source.
            if(a==0 and b!=0):
                A[x][b-1] = -1
                A[b-1][x] = -1
            elif(b==0 and a!=0):
```

```
                A[x][a-1] = 1
                A[a-1][x] = 1
            else:
                A[x][b-1] = -1
                A[x][a-1] = 1
                A[a-1][x] = 1
                A[b-1][x] = -1
            continue
        elif(line[0]=='I'):
            I_mag = float(discription[-2])
            phase = float(discription[-1])
            I =  complex(I_mag*cos(phase*(pi/180)), I_mag*sin(phase*(pi/180)))
            if(a==0 and b!=0):
                B[b-1] += I
            elif(b==0 and a!=0):
                B[a-1] -=I
            else:
                B[a-1] -=I
                B[b-1] += I

    x = solve(A, B)
    for i in range(n):
        print(f"Node {i+1} Voltage : magnitude = {abs(x[i])} V, phase = {cmath.
↪phase(x[i])*(180/pi)} degrees")
    for i in range(Nv):
        print(f"Current through Voltage source V{i+1} : magnitude = {␣
↪abs(x[n+i])} A, phase =  {cmath.phase(x[n+i])*(180/pi)} degrees")

    return ""
```

4.3 Read Netlist file and solve the circuit

From here, I start reading netlist and then take decision to call DC_MNA function or AC_MNA function or call none of them.

[21]:
```
def Solve_Circuit(filename):
    with open(filename, 'r') as netlist:
        content_of_netlist = netlist.readlines()
        start, end = 0, 0
        for i in range(len(content_of_netlist)):
            if(len(content_of_netlist[i].split())==0):
                continue
            if(content_of_netlist[i].split()[0] == '.circuit'):
                start = i
            elif(content_of_netlist[i].split()[0]=='.end'):
                end = i
        if(end < start):
```

```
            return "Invalid Circuit Declaration"
        #Check dc or ac source(s).
        dc = False
        ac = False
        for lines in content_of_netlist:
            if(lines[0]=='V' or lines[0]=='I'):
                if(lines.split()[3]=='dc'):
                    dc = True
                elif(lines.split()[3]=='ac'):
                    ac = True
        if(ac and dc):
            return "Given circuit has more than one frequency.(since frequency␣
↪of DC source is zero)"
        elif(dc==True):
            return DC_MNA(content_of_netlist, start, end)
        elif(ac==True):
            return AC_MNA(content_of_netlist, start, end)
```

Output of test cases uploaded on moodle

[22]:
```
print(Solve_Circuit("ckt1.netlist"))
print("="*30)
print(Solve_Circuit("ckt2.netlist"))
print("="*30)
print(Solve_Circuit("ckt3.netlist"))
print("="*30)
print(Solve_Circuit("ckt4.netlist"))
print("="*30)
print(Solve_Circuit("ckt5.netlist"))
print("="*30)
print(Solve_Circuit("ckt6.netlist"))
print("="*30)
print(Solve_Circuit("ckt7.netlist"))
```

```
Node 1 Voltage = 0.0 V
Node 2 Voltage = 0.0 V
Node 3 Voltage = 0.0 V
Node 4 Voltage = -5.0 V
Current through Voltage source V1 = -0.0005 A

==============================
Given circuit has more than one frequency.(since frequency of DC source is zero)
==============================
Node 1 Voltage = -10.0 V
Node 2 Voltage = -5.029239766081871 V
Node 3 Voltage = -2.5730994152046787 V
Node 4 Voltage = -1.403508771929825 V
Node 5 Voltage = -0.9356725146198834 V
```

```
Current through Voltage source V1 = -0.004970760233918128 A

==============================
Node 1 Voltage = -10.0 V
Node 2 Voltage = -5.555555555555556 V
Node 3 Voltage = -3.7037037037037033 V
Current through Voltage source V1 = -2.2222222222222214 A

==============================
Node 1 Voltage = -10.0 V
Current through Voltage source V1 = -1.0 A

==============================
Node 1 Voltage : magnitude = 3.141592653524598e-05 V, phase = -90.00036911890652
degrees
Node 2 Voltage : magnitude = 3.221170125068894e-05 V, phase = -90.00036911890652
degrees
Node 3 Voltage : magnitude = 5.0 V, phase = -180.0 degrees
Current through Voltage source V1 : magnitude = 0.00499999999989624 A, phase =
179.99963088109348 degrees

==============================
Node 1 Voltage : magnitude = 0.0007761154806732287 V, phase = 89.99999110637171
degrees
```

Some of my own test cases

[23]:
```
print("AC Source(s)\n==============================")
print(Solve_Circuit('AC1.netlist'))
```

```
AC Source(s)
==============================
Node 1 Voltage : magnitude = 4.0 V, phase = 29.999999999999996 degrees
Node 2 Voltage : magnitude = 6.000000000000001 V, phase = 59.99999999999999
degrees
Node 3 Voltage : magnitude = 168.4333450781387 V, phase = 39.27476850443678
degrees
Node 4 Voltage : magnitude = 91.96228125341496 V, phase = 158.0780671886268
degrees
Node 5 Voltage : magnitude = 96.14432133702611 V, phase = 152.5410015060631
degrees
Node 6 Voltage : magnitude = 8.9239583734777 V, phase = 38.33806605740891
degrees
Node 7 Voltage : magnitude = 533.034650816758 V, phase = 21.99380938872583
degrees
Current through Voltage source V1 : magnitude = 2.090577954005548 A, phase =
153.4094702189483 degrees
Current through Voltage source V2 : magnitude = 5.1277700946880405 A, phase =
```

```
-118.8825666749316 degrees
Current through Voltage source V3 : magnitude = 2.002684775001888 A, phase =
157.20350346613174 degrees
Current through Voltage source V4 : magnitude = 20.250017332467316 A, phase =
21.54500746452838 degrees
```

[24]:
```
print("DC circuit 1\n====================================")
print(Solve_Circuit('DC1.netlist'))
```

```
DC circuit 1
====================================
Node 1 Voltage = 4.0 V
Node 2 Voltage = 6.0 V
Node 3 Voltage = 172.8461538461539 V
Node 4 Voltage = -39.07692307692299 V
Node 5 Voltage = -29.076923076922988 V
Node 6 Voltage = 9.0 V
Node 7 Voltage = 570.923076923077 V
Current through Voltage source V1 = -0.6615384615384607 A
Current through Voltage source V2 = -5.1 A
Current through Voltage source V3 = -0.7615384615384607 A
Current through Voltage source V4 = 20.76153846153846 A
```

[25]:
```
print("DC circuit 2\n====================================")
print(Solve_Circuit('DC2.netlist'))
```

```
DC circuit 2
====================================
Node 1 Voltage = 4.0 V
Node 2 Voltage = 6.0 V
Node 3 Voltage = 50.23061013443635 V
Node 4 Voltage = 129.96173733195434 V
Node 5 Voltage = 139.96173733195434 V
Node 6 Voltage = 9.0 V
Node 7 Voltage = 739.9617373319542 V
Node 8 Voltage = -1870.0382626680455 V
Node 9 Voltage = 34.0 V
Current through Voltage source V1 = 3.425646328852118 A
Current through Voltage source V2 = -5.1 A
Current through Voltage source V3 = 3.325646328852118 A
Current through Voltage source V4 = 16.67435367114788 A
Current through Voltage source V5 = 0.706411582213029 A
```