EE2703

Srikar Babu Gadipudi EE21B138

February 6, 2023

1 Prerequisites to run this notebook

A working jupyter lab environment is needed, to access files and run python cells. Libraries numpy, sys and cmath are expected. Custom testcases are provided in the same directory.

2 Problem 1

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.

[1]:
```
def factorial(n):
    if n < 0:
        print("Error: The number is negative.")
    elif n == 1:
        return n
    else:
        return n*factorial(n-1)
```

[2]:
```
print(factorial(100))
%timeit factorial(100)
```

```
9332621544394415268169923885626670049071596826438162146859296389521759999322991 5
608941463976156518286253697920827223758251185210916864000000000000000000000000
14.8 µs ± 518 ns per loop (mean ± std. dev. of 7 runs, 100000 loops each)
```

2.1 Explanation

We use recursion to calculate the factorial of the given number. The function is called at every instant decrementing the argument till it reaches the exit case `n=1`.

3 Problem 2

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

[3]:
```
import numpy as np
import sys
```

3.1 Explanation

Import numpy library as np, we use numpy for computation and defining arrays.

[4]:
```
def gaussian(A, B):

    if A.shape[0] != A.shape[1] or A.shape[0] != B.shape[0]:
        print("Check your matrix.")
        sys.exit()
    n = A.shape[0]

    for i in range(n):

        #row swapping and checking pivot elements
        if A[i][i] == 0:
            for j in range(i, n):
                if A[j][j] !=0:
                    ls = A[i]
                    A[i] = A[j]
                    A[j] = ls
                    dum = B[i]
                    B[i] = B[j]
                    B[j] = dum

        #checking the valuse in the last row of the augmented matrix
        #if all zero then infinetly many solutions
        if all(l == 0 for l in A[n-1]) and B[n-1] == 0:
            return "Infinitely many solutions"
        #else if all zero in matrix A and non-zero value in B then no solution
        elif all(l == 0 for l in A[n-1]) and B[n-1] != 0:
            return "No solutions"

        #pivot element
        norm = A[i][i]
        #forward substitution for upper triangular matrix
        for j in range(n):
            A[i][j] = A[i][j]/norm
        B[i] = B[i]/norm

        for j in range(i+1, n):
            norm = A[j][i]
            for k in range(n): A[j][k] = A[j][k] - norm*A[i][k]
            B[j] = B[j] - B[i]*norm
```

```
    #back substitution for row reduced echelon form
    for i in range(n-1, -1, -1):
        for j in range(i-1, -1, -1):
            norm = A[j][i]
            for k in range(n):
                A[j][k] = A[j][k] - norm*A[i][k]
            B[j] = B[j] - norm*B[i]
    #returns the matrices A in row reduced echelon form and B gives the values␣
↪of the variables
    return A, B
```

[5]:
```
A = np.array([[1 ,3 ,1 ,2 ,6 ,6 ,0 ,1 ,3 ,5 ],
[7 ,0 ,2 ,0 ,5 ,5 ,6 ,3 ,3 ,3 ],
[6 ,0 ,6 ,0 ,0 ,8 ,4 ,5 ,3 ,7 ],
[8 ,5 ,4 ,9 ,3 ,5 ,3 ,5 ,8 ,7 ],
[7 ,6 ,3 ,8 ,9 ,2 ,3 ,8 ,7 ,8 ],
[9 ,5 ,7 ,0 ,7 ,7 ,0 ,1 ,8 ,6 ],
[3 ,9 ,7 ,9 ,2 ,1 ,7 ,6 ,7 ,1 ],
[8 ,5 ,6 ,4 ,4 ,0 ,3 ,7 ,2 ,5 ],
[1 ,2 ,7 ,6 ,1 ,5 ,2 ,0 ,8 ,1 ],
[6 ,4 ,4 ,3 ,6 ,2 ,7 ,8 ,5 ,2 ]], dtype='float')
B = np.array([2, 3, 4, 1, 2, 2, 2, 9, 7, 5], dtype='float')

print(gaussian(A, B))

%timeit gaussian(A, B)
```

```
(array([[ 1.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.],
       [ 0.,  1.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.],
       [ 0.,  0.,  1.,  0.,  0.,  0.,  0.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  1.,  0.,  0.,  0.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.,  1.,  0.,  0.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.,  0.,  1.,  0.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.,  0.,  0.,  1.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.,  0.,  0.,  0.,  1.,  0.,  0.],
       [ 0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  1.,  0.],
       [-0., -0., -0., -0., -0., -0., -0., -0., -0.,  1.]]), array([ 0.57592865,
-1.1434718 ,  1.700912  ,  1.56273285,  1.1733649 ,
        1.36712171, -1.35775437,  1.04556496, -1.97475077, -2.06722465]))
886 µs ± 28.7 µs per loop (mean ± std. dev. of 7 runs, 1000 loops each)
```

3.2 Explanation

We use Gaussian Elimination technique to solve this set of equations. We use pivoting and row swapping to check for '0' pivot elements and to detect "infinitely many solutions" or "no solution" cases.

The array B gives the values of the variables.

[6]:
```
np.linalg.solve(A, B)

print(A, B)
%timeit np.linalg.solve(A, B)
```

```
[[1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 1. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 1. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 1. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 1. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 1. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 1. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 1. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 1. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 1.]] [ 0.57592865 -1.1434718   1.700912
1.56273285  1.1733649   1.36712171
 -1.35775437  1.04556496 -1.97475077 -2.06722465]
5.73 µs ± 349 ns per loop (mean ± std. dev. of 7 runs, 100000 loops each)
```

3.3 Explanation

Our gaussian elimination solver takes more time to compute the solution to this 10x10 system of equations, while numpy's inbuilt function `numpy.linalg.solve()` takes lesser time to run.

We see that numpy is faster but it cannot handle complex numbers, while our eliminator can.

4 Problem 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.

[7]:
```
def chan(x):
    if x == "GND":
        x = '0'
    return x
```

4.1 Explanation

This is custom defined function to check and assign the `GND` node to `0`.

4.2 Use this cell when the circuit purely DC

When the sources are purely DC and the elements are resistor use this cell to get the output.

In the custom testcases, files: **1, 3 and 5** are run using this cell.

[12]:

```
f = open("ckt5.netlist", "r")

ls = f.readlines()

#look for start and end of circuit
for i in range(len(ls)):
    if ls[i] == ".circuit\n" or ls[i] == ".circuit": start = i+1
    if ls[i] == ".end\n" or ls[i] == ".end": end = i

ls = ls[start:end]

cnt=0

arr = [[]]

#splitting the terms in each line
for i in range(len(ls)):
    arr.append(ls[i].split())

arr = arr[1:]
maxi = 0
nv = 0

#to calculate number of nodes and number of voltage sources
for i in range(len(arr)):
    if arr[i][0][0] == 'V': nv += 1
    if maxi < int(chan(arr[i][1])):
        maxi = int(chan(arr[i][1]))
    if maxi < int(chan(arr[i][2])):
        maxi = int(chan(arr[i][2]))

#initialize A and B amtrices accordingly
A = np.zeros((maxi+nv, maxi+nv))
B = np.zeros(maxi+nv)

f.close()

#go through each node and fill the matrix A and B
for i in range(maxi):
    for j in range(len(arr)):
        #for resistive element
        if arr[j][0][0] == 'R':
            if int(chan(arr[j][1])) == i+1:
                A[i][i] += 1/float(chan(arr[j][3]))
                if int(chan(arr[j][2])) != 0:
                    A[i][int(chan(arr[j][2]))-1] += -1/float(chan(arr[j][3]))
            if int(chan(arr[j][2])) == i+1:
```

```
                A[i][i] += 1/float(chan(arr[j][3]))
                if int(chan(arr[j][1])) != 0:
                    A[i][int(chan(arr[j][1]))-1] += -1/float(chan(arr[j][3]))
        #for voltage source
        if arr[j][0][0] == 'V':
            if int(chan(arr[j][1])) == i+1:
                A[i][maxi - 1 + int(arr[j][0][1])] += 1
            if int(chan(arr[j][2])) == i+1:
                A[i][maxi - 1 + int(arr[j][0][1])] -= 1
        #for current source
        if arr[j][0][0] == 'I':
            if int(chan(arr[j][1])) == i+1:
                B[i] -= int(chan(arr[j][4]))
            if int(chan(arr[j][2])) == i+1:
                B[i] += int(chan(arr[j][4]))

#auxillary equations
for j in range(len(arr)):
    if arr[j][0][0] == 'V':
        if int(chan(arr[j][1])) != 0:
            A[maxi - 1 + int(arr[j][0][1])][int(chan(arr[j][1])) - 1] = -1
        if int(chan(arr[j][2])) != 0:
            A[maxi - 1 + int(arr[j][0][1])][int(chan(arr[j][2])) - 1] = 1
        B[maxi - 1 + int(arr[j][0][1])] = int(chan(arr[j][4]))

print("The modified nodal equations in the matrix form")
print(A)
print(B)
```

```
The modified nodal equations in the matrix form
[[ 0.1 -1. ]
 [ 1.   0. ]]
[ 0. 10.]
```

[13]:
```
gaussian(A, B)
print("Nodal Voltages and Auxillary Currents")
for i in range(maxi):
    print(f"V{i+1:} = %.5f" % B[i])
for i in range(nv):
    print(f"I{i+1:} = %.5f" % B[maxi + i])
```

```
Nodal Voltages and Auxillary Currents
V1 = 10.00000
I1 = 1.00000
```

4.3 Explanation

We use list manipulation and modified nodal analysis techniques to build our A and B matrices with nodal voltages and auxillary currents as its variables. Then use our previously defined gaussian solver to solve the constructed matrix.

4.4 Use this cell for purely AC circuit with or without Inductors and Capacitors

When the sources are purely AC use this cell to get the output.

In the custom testcases, files: **6 and 7** are run using this cell.

[16]:
```
f = open("ckt7.netlist", "r")

ls = f.readlines()

fr = []

#look for start and end of circuit and identify frequency
for i in range(len(ls)):
    if ls[i] == ".circuit\n" or ls[i] == ".circuit": start = i+1
    if ls[i] == ".end\n" or ls[i] == ".end": end = i
    if ls[i][0:3] == ".ac":
        fr.append(i)
        chk = len(ls[i])
        for j in range(len(ls[i])):
            if ls[i][j] == '#':
                chk = j
        ls[i] = ls[i][:chk]

#to check for multiple frequencies
nl = [float(ls[fr[i]][7:]) - float(ls[fr[0]][7:]) for i in range(len(fr))]

#if not multiple frequencies
if all(i == 0.0 for i in nl):
    freq = float(ls[fr[0]][7:])
    #define omega
    omega = 2*np.pi*freq

    ls = ls[start:end]

    cnt=0

    arr = [[]]

    for i in range(len(ls)):
        arr.append(ls[i].split())

    arr = arr[1:]
    maxi = 0
    nv = 0
    #slice the array ad convert nodes accordingly
```

```
for i in range(len(arr)):
    if arr[i][1][0] == 'n': arr[i][1] = arr[i][1][1]
    if arr[i][2][0] == 'n': arr[i][2] = arr[i][2][1]
#finding the number of nodes and voltage sources
for i in range(len(arr)):
    if arr[i][0][0] == 'V': nv += 1
    if maxi < int(chan(arr[i][1])):
        maxi = int(chan(arr[i][1]))
    if maxi < int(chan(arr[i][2])):
        maxi = int(chan(arr[i][2]))
#initializing A and B matrices
A = np.zeros((maxi+nv, maxi+nv), dtype = 'complex')
B = np.zeros(maxi+nv, dtype = 'complex')

f.close()

#for each node update A and B matrices accordingly
for i in range(maxi):
    for j in range(len(arr)):
        #if found resistor
        if arr[j][0][0] == 'R':
            if int(chan(arr[j][1])) == i+1:
                A[i][i] += 1/float(chan(arr[j][3]))
                if int(chan(arr[j][2])) != 0:
                    A[i][int(chan(arr[j][2]))-1] += -1/float(chan(arr[j][3]))
            if int(chan(arr[j][2])) == i+1:
                A[i][i] += 1/float(chan(arr[j][3]))
                if int(chan(arr[j][1])) != 0:
                    A[i][int(chan(arr[j][1]))-1] += -1/float(chan(arr[j][3]))
        #if found inductor
        if arr[j][0][0] == 'L':
            xl = float(chan(arr[j][3]))*complex(0, 1)*omega
            if int(chan(arr[j][1])) == i+1:
                A[i][i] += 1/xl
                if int(chan(arr[j][2])) != 0:
                    A[i][int(chan(arr[j][2]))-1] += -1/xl
            if int(chan(arr[j][2])) == i+1:
                A[i][i] += 1/xl
                if int(chan(arr[j][1])) != 0:
                    A[i][int(chan(arr[j][1]))-1] += -1/xl
        #if found capacitor
        if arr[j][0][0] == 'C':
            xc = 1/(float(chan(arr[j][3]))*complex(0, 1)*omega)
            if int(chan(arr[j][1])) == i+1:
                A[i][i] += 1/xc
                if int(chan(arr[j][2])) != 0:
                    A[i][int(chan(arr[j][2]))-1] += -1/xc
```

```
                if int(chan(arr[j][2])) == i+1:
                    A[i][i] += 1/xc
                    if int(chan(arr[j][1])) != 0:
                        A[i][int(chan(arr[j][1]))-1] += -1/xc
            #if found voltage source
            if arr[j][0][0] == 'V':
                if int(chan(arr[j][1])) == i+1:
                    A[i][maxi - 1 + int(arr[j][0][1])] += 1
                if int(chan(arr[j][2])) == i+1:
                    A[i][maxi - 1 + int(arr[j][0][1])] -= 1
            #if found current source
            if arr[j][0][0] == 'I':
                if int(chan(arr[j][1])) == i+1:
                    B[i] -= int(chan(arr[j][4]))
                if int(chan(arr[j][2])) == i+1:
                    B[i] += int(chan(arr[j][4]))
    #auxillary equations
    for j in range(len(arr)):
        if arr[j][0][0] == 'V':
            if int(chan(arr[j][1])) != 0:
                A[maxi - 1 + int(arr[j][0][1])][int(chan(arr[j][1])) - 1] = -1
            if int(chan(arr[j][2])) != 0:
                A[maxi - 1 + int(arr[j][0][1])][int(chan(arr[j][2])) - 1] = 1
            if arr[j][3] == 'ac':
                B[maxi - 1 + int(arr[j][0][1])] = int(chan(arr[j][4]))

    print("The modified nodal analysis matrix")
    print(A)
    print(B)

#for multiple frequencies
else:
    print("Skill issue : Multiple Frequencies")
```

```
The modified nodal analysis matrix
[[0.001+6124.03036409j]]
[5.+0.j]
```

[17]:
```
import cmath
gaussian(A, B)

print("Nodal Voltages and Auxillary Currents")

for i in range(maxi):
    print(f"V{i+1:}: Magnitude = %.5f" % cmath.polar(B[i])[0], "\t", "Phase = %.
 ↪5f" % cmath.polar(B[i])[1])
for i in range(nv):
```

```
    print(f"I{i+1:}: Magnitude = %.5f" % cmath.polar(B[maxi + i])[0], "\t",␣
↪"Phase = %.5f" % cmath.polar(B[maxi + i])[1])
```

```
Nodal Voltages and Auxillary Currents
V1: Magnitude = 0.00082          Phase = -1.57080
```

4.5 Explanation

We use similar techniques as DC, but in complex domain where the corresponding impedances are considered.

The output given is to be undertood as the

$$V = Magnitude \times e^{j\omega t+phase}$$

5 Use this cell to run cicuits with a mixed set of sources AC and DC

When the sources are a superposition of DC and AC sources use this cell to get the output.

In the custom testcases, file: **2** is run using this cell.

5.0.1 This cell is considering only DC, shorting all other AC sources

[18]:
```
f = open("ckt2.netlist", "r")

ls = f.readlines()

for i in range(len(ls)):
    if ls[i] == ".circuit\n" or ls[i] == ".circuit": start = i+1
    if ls[i] == ".end\n" or ls[i] == ".end": end = i

ls = ls[start:end]

cnt=0

arr = [[]]


for i in range(len(ls)):
    arr.append(ls[i].split())

arr = arr[1:]
maxi = 0
nv = 0

for i in range(len(arr)):
    if arr[i][0][0] == 'V': nv += 1
    if maxi < int(chan(arr[i][1])):
```

```
        maxi = int(chan(arr[i][1]))
    if maxi < int(chan(arr[i][2])):
        maxi = int(chan(arr[i][2]))

A = np.zeros((maxi+nv, maxi+nv))
B = np.zeros(maxi+nv)

f.close()

for i in range(maxi):
    for j in range(len(arr)):
        if arr[j][0][0] == 'R':
            if int(chan(arr[j][1])) == i+1:
                A[i][i] += 1/float(chan(arr[j][3]))
                if int(chan(arr[j][2])) != 0:
                    A[i][int(chan(arr[j][2]))-1] += -1/float(chan(arr[j][3]))
            if int(chan(arr[j][2])) == i+1:
                A[i][i] += 1/float(chan(arr[j][3]))
                if int(chan(arr[j][1])) != 0:
                    A[i][int(chan(arr[j][1]))-1] += -1/float(chan(arr[j][3]))
        if arr[j][0][0] == 'V':
            if int(chan(arr[j][1])) == i+1:
                A[i][maxi - 1 + int(arr[j][0][1])] += 1
            if int(chan(arr[j][2])) == i+1:
                A[i][maxi - 1 + int(arr[j][0][1])] -= 1
        if arr[j][0][0] == 'I' and arr[j][3] == "dc":
            if int(chan(arr[j][1])) == i+1:
                B[i] -= int(chan(arr[j][4]))
            if int(chan(arr[j][2])) == i+1:
                B[i] += int(chan(arr[j][4]))


for j in range(len(arr)):
    if arr[j][0][0] == 'V':
        if int(chan(arr[j][1])) != 0:
            A[maxi - 1 + int(arr[j][0][1])][int(chan(arr[j][1])) - 1] = -1
        if int(chan(arr[j][2])) != 0:
            A[maxi - 1 + int(arr[j][0][1])][int(chan(arr[j][2])) - 1] = 1
        if arr[j][3] == "dc":
            B[maxi - 1 + int(arr[j][0][1])] = int(chan(arr[j][4]))

print("The modified nodal equations in the matrix form for DC sources alone by shorting AC sources.")
print(A)
print(B)
```

```
The modified nodal equations in the matrix form for DC sources alone by shorting
AC sources.
```

```
[[ 1.50e-03 -5.00e-04  0.00e+00  0.00e+00  0.00e+00  0.00e+00  0.00e+00
   0.00e+00]
 [-5.00e-04  7.50e-04 -2.50e-04  0.00e+00  0.00e+00  0.00e+00  0.00e+00
   1.00e+00]
 [ 0.00e+00 -2.50e-04  2.50e-04  0.00e+00  0.00e+00  0.00e+00  1.00e+00
   0.00e+00]
 [ 0.00e+00  0.00e+00  0.00e+00  3.25e-04 -2.00e-04 -1.25e-04  0.00e+00
  -1.00e+00]
 [ 0.00e+00  0.00e+00  0.00e+00 -2.00e-04  2.00e-04  0.00e+00  0.00e+00
   0.00e+00]
 [ 0.00e+00  0.00e+00  0.00e+00 -1.25e-04  0.00e+00  1.25e-04  0.00e+00
   0.00e+00]
 [ 0.00e+00  0.00e+00 -1.00e+00  0.00e+00  0.00e+00  0.00e+00  0.00e+00
   0.00e+00]
 [ 0.00e+00 -1.00e+00  0.00e+00  1.00e+00  0.00e+00  0.00e+00  0.00e+00
   0.00e+00]]
[ 0.   0.   0.   0. -1. 10.   0.   5.]
```

[19]:
```
gaussian(A, B)
print("Nodal Voltages and Auxillary Currents only DC")
for i in range(maxi):
    print(f"V{i+1:} = %.5f" % B[i])
for i in range(nv):
    print(f"I{i+1:} = %.5f" % B[maxi + i])
```

```
Nodal Voltages and Auxillary Currents only DC
V1 = 5142.85714
V2 = 15428.57143
V3 = 0.00000
V4 = -14464.00000
V5 = -19464.00000
V6 = 65536.00000
I1 = 3.85714
I2 = -9.00000
```

5.0.2 This cell is considering only AC, shorting all other DC sources

[21]:
```
f = open("ckt2.netlist", "r")

ls = f.readlines()

fr = []

for i in range(len(ls)):
    if ls[i] == ".circuit\n" or ls[i] == ".circuit": start = i+1
    if ls[i] == ".end\n" or ls[i] == ".end": end = i
    if ls[i][0:3] == ".ac":
        fr.append(i)
```

```
        chk = len(ls[i])
        for j in range(len(ls[i])):
            if ls[i][j] == '#':
                chk = j
        ls[i] = ls[i][:chk]

nl = [float(ls[fr[i]][7:]) - float(ls[fr[0]][7:]) for i in range(len(fr))]

if all(i == 0.0 for i in nl):
    freq = float(ls[fr[0]][7:])

    omega = 2*np.pi*freq

    ls = ls[start:end]

    cnt=0

    arr = [[]]

    for i in range(len(ls)):
        arr.append(ls[i].split())

    arr = arr[1:]
    maxi = 0
    nv = 0

    for i in range(len(arr)):
        if arr[i][1][0] == 'n': arr[i][1] = arr[i][1][1]
        if arr[i][2][0] == 'n': arr[i][2] = arr[i][2][1]

    for i in range(len(arr)):
        if arr[i][0][0] == 'V': nv += 1
        if maxi < int(chan(arr[i][1])):
            maxi = int(chan(arr[i][1]))
        if maxi < int(chan(arr[i][2])):
            maxi = int(chan(arr[i][2]))

    A = np.zeros((maxi+nv, maxi+nv), dtype = 'complex')
    B = np.zeros(maxi+nv, dtype = 'complex')

    f.close()

    for i in range(maxi):
        for j in range(len(arr)):
            if arr[j][0][0] == 'R':
                if int(chan(arr[j][1])) == i+1:
                    A[i][i] += 1/float(chan(arr[j][3]))
```

```
                if int(chan(arr[j][2])) != 0:
                    A[i][int(chan(arr[j][2]))-1] += -1/float(chan(arr[j][3]))
            if int(chan(arr[j][2])) == i+1:
                A[i][i] += 1/float(chan(arr[j][3]))
                if int(chan(arr[j][1])) != 0:
                    A[i][int(chan(arr[j][1]))-1] += -1/float(chan(arr[j][3]))
        if arr[j][0][0] == 'L':
            xl = float(chan(arr[j][3]))*complex(0, 1)*omega
            if int(chan(arr[j][1])) == i+1:
                A[i][i] += 1/xl
                if int(chan(arr[j][2])) != 0:
                    A[i][int(chan(arr[j][2]))-1] += -1/xl
            if int(chan(arr[j][2])) == i+1:
                A[i][i] += 1/xl
                if int(chan(arr[j][1])) != 0:
                    A[i][int(chan(arr[j][1]))-1] += -1/xl
        if arr[j][0][0] == 'C':
            xc = 1/(float(chan(arr[j][3]))*complex(0, 1)*omega)
            if int(chan(arr[j][1])) == i+1:
                A[i][i] += 1/xc
                if int(chan(arr[j][2])) != 0:
                    A[i][int(chan(arr[j][2]))-1] += -1/xc
            if int(chan(arr[j][2])) == i+1:
                A[i][i] += 1/xc
                if int(chan(arr[j][1])) != 0:
                    A[i][int(chan(arr[j][1]))-1] += -1/xc
        if arr[j][0][0] == 'V':
            if int(chan(arr[j][1])) == i+1:
                A[i][maxi - 1 + int(arr[j][0][1])] += 1
            if int(chan(arr[j][2])) == i+1:
                A[i][maxi - 1 + int(arr[j][0][1])] -= 1
        if arr[j][0][0] == 'I' and arr[j][3] == "ac":
            if int(chan(arr[j][1])) == i+1:
                B[i] -= int(chan(arr[j][4]))
            if int(chan(arr[j][2])) == i+1:
                B[i] += int(chan(arr[j][4]))

for j in range(len(arr)):
    if arr[j][0][0] == 'V':
        if int(chan(arr[j][1])) != 0:
            A[maxi - 1 + int(arr[j][0][1])][int(chan(arr[j][1])) - 1] = -1
        if int(chan(arr[j][2])) != 0:
            A[maxi - 1 + int(arr[j][0][1])][int(chan(arr[j][2])) - 1] = 1
        if arr[j][3] == 'ac':
            B[maxi - 1 + int(arr[j][0][1])] = int(chan(arr[j][4]))
```

```
    print("The modified nodal equations in the matrix form for AC sources alone␣
 ↪by shorting DC sources.")
    print(A)
    print(B)

else:
    print("Multiple Frequencies")
```

```
The modified nodal equations in the matrix form for AC sources alone by shorting
DC sources.
[[ 1.50e-03+0.j -5.00e-04+0.j  0.00e+00+0.j  0.00e+00+0.j  0.00e+00+0.j
   0.00e+00+0.j  0.00e+00+0.j  0.00e+00+0.j]
 [-5.00e-04+0.j  7.50e-04+0.j -2.50e-04+0.j  0.00e+00+0.j  0.00e+00+0.j
   0.00e+00+0.j  0.00e+00+0.j  1.00e+00+0.j]
 [ 0.00e+00+0.j -2.50e-04+0.j  2.50e-04+0.j  0.00e+00+0.j  0.00e+00+0.j
   0.00e+00+0.j  1.00e+00+0.j  0.00e+00+0.j]
 [ 0.00e+00+0.j  0.00e+00+0.j  0.00e+00+0.j  3.25e-04+0.j -2.00e-04+0.j
  -1.25e-04+0.j  0.00e+00+0.j -1.00e+00+0.j]
 [ 0.00e+00+0.j  0.00e+00+0.j  0.00e+00+0.j -2.00e-04+0.j  2.00e-04+0.j
   0.00e+00+0.j  0.00e+00+0.j  0.00e+00+0.j]
 [ 0.00e+00+0.j  0.00e+00+0.j  0.00e+00+0.j -1.25e-04+0.j  0.00e+00+0.j
   1.25e-04+0.j  0.00e+00+0.j  0.00e+00+0.j]
 [ 0.00e+00+0.j  0.00e+00+0.j -1.00e+00+0.j  0.00e+00+0.j  0.00e+00+0.j
   0.00e+00+0.j  0.00e+00+0.j  0.00e+00+0.j]
 [ 0.00e+00+0.j -1.00e+00+0.j  0.00e+00+0.j  1.00e+00+0.j  0.00e+00+0.j
   0.00e+00+0.j  0.00e+00+0.j  0.00e+00+0.j]]
[0.+0.j 0.+0.j 0.+0.j 0.+0.j 0.+0.j 0.+0.j 2.+0.j 0.+0.j]
```

[22]:
```
import cmath
gaussian(A, B)

print("Nodal Voltages and Auxillary Currents for only AC")

for i in range(maxi):
    print(f"V{i+1:}: Magnitude = %.5f" % cmath.polar(B[i])[0], "\t", f"Phase = %.
 ↪5f" % cmath.polar(B[i])[1])
for i in range(nv):
    print(f"I{i+1:}: Magnitude = %.5f" % cmath.polar(B[maxi + i])[0], "\t",␣
 ↪"Phase = %.5f" % cmath.polar(B[maxi + i])[1])
```

```
Nodal Voltages and Auxillary Currents for only AC
V1: Magnitude = 0.28571         Phase = 3.14159
V2: Magnitude = 0.85714         Phase = 3.14159
V3: Magnitude = 2.00000         Phase = 3.14159
V4: Magnitude = 0.85714         Phase = 3.14159
V5: Magnitude = 0.85714         Phase = 3.14159
V6: Magnitude = 0.85714         Phase = 3.14159
I1: Magnitude = 0.00029         Phase = 0.00000
```

```
I2: Magnitude = 0.00000          Phase = -0.00000
```

5.1 Explanation

The output should be interpreted as

$$V_{dc} = Output_{dc}$$

$$V_{ac} = Output_{ac} = Magnitude \times e^{j\omega t + phase}$$

$$V_{total} = V_{dc} + V_{ac}$$