EE2703 Assignment 2

Name: Hemanth Ram
Roll Number: EE18B132

January 30, 2020

1 Assignment 2 -

In this assignment, we get as input a netlist file, extract useful parts of it, and solve the circuit using the Matrix method.

Importing Required Libraries for the process:
- sys - system operations
- numpy - matrix manipulations
- math & cmath - general math computations

```
import sys
import numpy as np
import math
import cmath
```

Function to return value given in some other notation

```
def parse_val(x):
    y = len(x)
    if(not x[y-1].isalpha()):
        return float(x)
    if(x[y-1]=='p'):
        return float(x[0:y-1])* 1e-12
    if(x[y-1]=='n'):
        return float(x[0:y-1])* 1e-9
    if(x[y-1]=='u'):
        return float(x[0:y-1])* 1e-6
    if(x[y-1]=='m'):
        return float(x[0:y-1])* 1e-3
    if(x[y-1]=='k'):
        return float(x[0:y-1])* 1e3
    if(x[y-1]=='M'):
        return float(x[0:y-1])* 1e6
    if(x[y-1]=='G'):
        return float(x[0:y-1])* 1e9
```

2 Part 1 - Class Definition

Component class which takes the information about the component as input and creates object. The *name* contains name of node, the *nodes* contain the nodes the element is connected to, and the *value* contains the corresponding value of the node. The global variable *node* is a dictionary which has the list of all nodes present in the circuit, and is updated whenever a *comp* object is initialized.

```
class comp:
        name = ''
        nodes = []
        value = 0
        global node,w
        def __init__(self, info):
                l = len(info)
                self.name = info[0]
                self.nodes = info[1:]
                self.nodes.pop()
                if(l == 4):
                        if(self.name[0] == 'R' or self.name[0] == 'V' or self.name[0] == 'I'):
                                self.value = parse_val(info[-1])
                        elif(self.name[0] == 'L'):
                                if(AC == 1):
                                        self.value = complex(0,parse_val(info[-1])*w)
                                else:
                                        self.value = 10**(-9)
                        elif(self.name[0] == 'C'):
                                if(AC == 1):
                                        self.value = complex(0,-1/(parse_val(info[-1])*w))
                                else:
                                        self.value = 10**(9)
                elif(l == 5):
                        self.nodes.pop()
                        self.value = parse_val(info[-1])
                else:
                        _ = [self.nodes.pop() for i in range(2)]
                        phi = parse_val(info[-1])
                        v = (parse_val(info[-2]))/2
                        self.value = v*complex(math.cos(phi),math.sin(phi))
                for n in self.nodes:
                        node[n] = True
```

3 Part 2 - Reading file

Getting *filename* and trying to open it, if not, exiting.

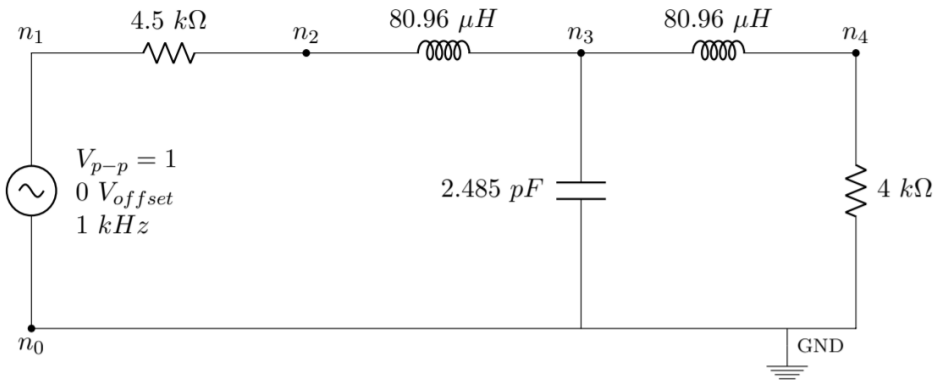
```
# if(len(sys.argv) < 2):
#         print('Enter file name')
#         exit()
```

```
# filename = sys.argv[1]
filename = input('Enter File Name : ')
try:
        f = open(filename, 'r')
except Exception:
        print('File not found')
        sys.exit(0)

Enter File Name : ckt4.netlist
```

Finding '.circuit' and '.end' in the file to extract the useful part and checking if '.ac' is specified, if yes, then reading the frequency also for further calculation in w. If the circuit is AC, then the *AC* flag is also set.

```
netlist = list(map(lambda x : x.strip('\n'),f.readlines()))
f.close()
size = len(netlist)
start = 0
end = 0
AC = 0
w = 0
for i in range(size):
        if(netlist[i] == '.circuit'):
                start = i
        if(netlist[i] == '.end'):
                end = i
        if(netlist[i].split(' ')[0] == '.ac'):
                AC = 1
                w = parse_val(netlist[i].split(' ')[2])
                w = w*(math.pi)*2
                break
if(start >= end):
        print('Invalid .netlist file')
        sys.exit(0)
netlist = netlist[start+1:end]
netlist = list(map(lambda x : (x.split('#')[0].strip(' ')).split(' '), netlist))
print('The netlist contains :\n[')
for nn in netlist:
    print(nn)
print(']')

The netlist contains :
[
['V1', 'GND', '1', 'ac', '1', '0']
['R1', '1', '2', '4.5e3']
['L1', '2', '3', '80.96e-6']
['C1', '3', 'GND', '2.485e-12']
['L1', '3', '4', '80.96e-6']
```

```
['R2', '4', 'GND', '4e3']
]
The corresponding circuit is shown below :
```



4 Part 3 - Initializing necessary variables

'n' is the number of nodes, 'vs' is the number of voltage sources present. Each node is given a specific number starting from 0 and will be referenced further using the dictionary. Similarly each voltage source is also given a number using the 'Vs' dict. Finally, the 'ckt' list contains all the 'comp' objects created for each element.

```
node = {}
Vs = {}
vs = 0
ckt = [comp(x) for x in netlist]
n = 0
for k in node:
        node[k] = n; n+=1

for ele in ckt:
        if(ele.name[0] == 'V'):
                Vs[ele.name] = vs
                vs += 1
```

If the circuit is AC, then declaring the M and b matrices to contain complex values, else, leaving to be default type.

```
if(AC != 1):
        M = np.zeros((n+vs,n+vs))
        b = np.zeros(n+vs)
else:
        M = np.zeros((n+vs,n+vs),dtype = np.complex)
        b = np.zeros(n+vs,dtype = np.complex)
```

5 Part 4 - Updating the M matrix

Each element in 'ckt' is checked for type and M is updated accordingly.

```
for ele in ckt:
        if(ele.name[0] == 'V'):
                high = node[ele.nodes[0]]
                low = node[ele.nodes[1]]
                M[n+Vs[ele.name]][high] = 1
                M[n+Vs[ele.name]][low] = -1
                b[n+Vs[ele.name]] = ele.value
                if(high != 0):
                        M[high][n+Vs[ele.name]] = 1
                if(low != 0):
                        M[low][n+Vs[ele.name]] = -1
        elif(ele.name[0] == 'R' or ele.name[0] == 'L' or ele.name[0] == 'C'):
                n1 = node[ele.nodes[0]]
                n2 = node[ele.nodes[1]]
                if(n1 != 0):
                        M[n1][n1] = M[n1][n1] + 1/(ele.value)
                        M[n1][n2] = M[n1][n2] - 1/(ele.value)
                if(n2 != 0):
                        M[n2][n1] = M[n2][n1] - 1/(ele.value)
                        M[n2][n2] = M[n2][n2] + 1/(ele.value)
        elif(ele.name[0] == 'I'):
                fr = node[ele.nodes[0]]
                to = node[ele.nodes[1]]
                b[fr] = b[fr] + ele.value
                b[to] = b[to] - ele.value
M[0][0] = 1
print('\nThe M Matrix :')
print(M)
print('\nThe b Matrix :')
print(b)
```

```
The M Matrix :
[[ 1.00000000e+00+0.00000000e+00j  0.00000000e+00+0.00000000e+00j
   0.00000000e+00+0.00000000e+00j  0.00000000e+00+0.00000000e+00j
   0.00000000e+00+0.00000000e+00j  0.00000000e+00+0.00000000e+00j]
 [ 0.00000000e+00+0.00000000e+00j  2.22222222e-04+0.00000000e+00j
  -2.22222222e-04+0.00000000e+00j  0.00000000e+00+0.00000000e+00j
   0.00000000e+00+0.00000000e+00j -1.00000000e+00+0.00000000e+00j]
 [ 0.00000000e+00+0.00000000e+00j -2.22222222e-04+0.00000000e+00j
   2.22222222e-04-1.96584663e-03j  0.00000000e+00+1.96584663e-03j
   0.00000000e+00+0.00000000e+00j  0.00000000e+00+0.00000000e+00j]
 [ 0.00000000e+00-1.56137155e-05j  0.00000000e+00+0.00000000e+00j
   0.00000000e+00+1.96584663e-03j  0.00000000e+00-3.91607954e-03j
```

```
   0.00000000e+00+1.96584663e-03j  0.00000000e+00+0.00000000e+00j]
 [-2.50000000e-04+0.00000000e+00j  0.00000000e+00+0.00000000e+00j
   0.00000000e+00+0.00000000e+00j  0.00000000e+00+1.96584663e-03j
   2.50000000e-04-1.96584663e-03j  0.00000000e+00+0.00000000e+00j]
 [ 1.00000000e+00+0.00000000e+00j -1.00000000e+00+0.00000000e+00j
   0.00000000e+00+0.00000000e+00j  0.00000000e+00+0.00000000e+00j
   0.00000000e+00+0.00000000e+00j  0.00000000e+00+0.00000000e+00j]]

The b Matrix :
[0. +0.j 0. +0.j 0. +0.j 0. +0.j 0. +0.j 0.5+0.j]
```

6 Part 5 - Solving the matrices to get solutions

If node 'GND' is present, setting that to 0 and getting the offset to be added. Then, the equations are solved using the *numpy.linalg.solve* function. If not possible, then exiting.

```
try:
    x = np.linalg.solve(M,b)
except:
    print('Matrix unsolvable')
    sys.exit(0)
for k in node:
    if(k == 'GND'):
        offset = x[node[k]]
print('Solved Matrix')
print('\nResults :')
print(x)
```

```
Solved Matrix

Results :
[ 0.00000000e+00+0.00000000e+00j -5.00000000e-01-0.00000000e+00j
 -2.38888436e-01-2.34158802e-02j -2.36241470e-01+6.10055877e-03j
 -2.31718159e-01+3.55685441e-02j -5.80247921e-05+5.20352893e-06j]
```

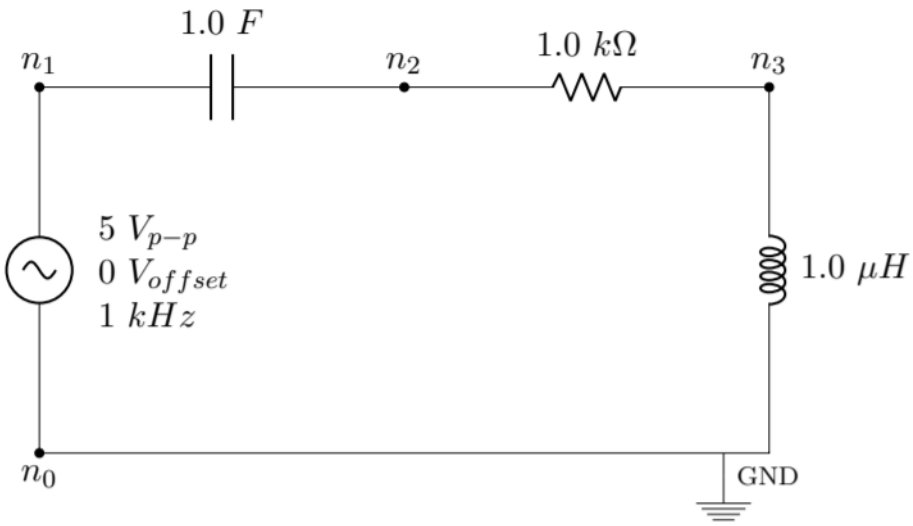
Printing the results using the offset value. For AC circuits, using the *cmath* library to compute phase and amp, rounding them and printing them accordingly.

```
if(AC != 1):
    for k in node:
        print('V('+k+') = '+str(round(x[node[k]]-offset,3)))
    for V in Vs:
        print('I in '+ V +' = ' + str(round(x[n+Vs[V]],3)))
else:
    for k in node:
        cmp = x[node[k]]-offset
```

```
        phase = round(cmath.phase(cmp)*(180/math.pi))
        amp = round(abs(cmp),3)
        print('V('+k+') = '+str(amp)+' < '+str(phase))
    for V in Vs:
        cmp = x[n+Vs[V]]
        phase = round(cmath.phase(cmp)*(180/math.pi))
        amp = round(abs(cmp),6)
        print('I in '+ V +' = ' + str(amp)+' < '+str(phase))
```

```
V(GND) = 0.0 < 0
V(1) = 0.5 < -180
V(2) = 0.24 < -174
V(3) = 0.236 < 179
V(4) = 0.234 < 171
I in V1 = 5.8e-05 < 175
```

For the circuit shown below, the outputs were :




```
The netlist contains :
[
['V1', 'n1', 'GND', 'ac', '5', '0']
['C1', 'n1', 'n2', '1']
['R1', 'n2', 'n3', '1000']
['L1', 'n3', 'GND', '1e-6']
]

The M Matrix :
```

```
[[ 1.      +0.j          0.      +0.j          0.      +0.j
   0.      +0.j          0.      +0.j        ]
 [ 0.      +0.j          0.    -159.15494309j  0.      +0.j
   0.    +159.15494309j -1.      +0.j        ]
 [ 0.   -6283.18530718j  0.      +0.j          0.001+6283.18530718j
  -0.001   +0.j          0.      +0.j        ]
 [ 0.      +0.j          0.    +159.15494309j -0.001   +0.j
   0.001 -159.15494309j  0.      +0.j        ]
 [ 1.      +0.j         -1.      +0.j          0.      +0.j
   0.      +0.j          0.      +0.j        ]]

The b Matrix :
[0. +0.j 0. +0.j 0. +0.j 0. +0.j 2.5+0.j]

Solved Matrix

Results :
[-5.26599533e-28+0.00000000e+00j -2.50000000e+00+0.00000000e+00j
  2.43667426e-12+3.97887358e-07j -2.50000000e+00+1.57079633e-05j
 -2.50000000e-03+1.53100759e-08j]

V(n1) = 2.5 < 0
V(GND) = 0.0 < 0
V(n2) = 2.5 < 0
V(n3) = 0.0 < 90
I in V1 = 0.0025 < 180
```