Applied Programming Lab - week 2

Nachiket Mahesh Dighe EE21B093

February 8, 2023

1 Speed of computation

Python, being an interpreted language, tends to be slower than compiled languages like C or Fortran. Some other languages like Java and Julia tend to use Just-in-Time compilation which can give speedups, but Python also has the problem of being dynamically typed, which eliminates the possibility of many optimizations.

The `timeit` library provides functions to estimate the time taken to run a piece of code. It can automatically run the code multiple times to get better average results, and can be used to identify bottlenecks in your program. However, it should be used with care as it is not a detailed function-call-level profiler.

It can either be `imported` as a module where you can then explicitly called `timeit.timeit(func)` to estimate time for a function, or you can use the *magic syntax* in Python notebooks as shown below.

[1]:
```
import numpy as np
x = np.random.rand(10000,1)
```

[2]:
```
def sumarr(x):
    sum = 0
    for i in range(len(x)):
    # for i in x:
        sum += x[i]
    return sum
print(sumarr(x))
%timeit sumarr(x)
```

```
[4963.41330484]
18.7 ms ± 650 µs per loop (mean ± std. dev. of 7 runs, 100 loops each)
```

[3]:
```
import numpy as np
def npsumarr(x):
    return np.sum(x)
print(npsumarr(x))
%timeit npsumarr(x)
```

```
4963.413304840611
13.6 µs ± 251 ns per loop (mean ± std. dev. of 7 runs, 100,000 loops each)
```

2 Solving equations by Gaussian elimination

Once you have constructed two matrices A and B to represent the system of linear equations

$$Ax = b$$

you can then proceed to solve the equations using the process known as Gaussian elimination.

It is assumed you already know how the process works, but to refresh your memory, you could use the reference material at LibreTexts.

Basically it involves making the A matrix *triangular* and ultimately into the shape of an identity matrix.

[4]:
```
# Input matrices - the set of equations - 2 variables x1 and x2
A = [ [2,3], [1,-1] ]
B = [6,1/2]
print(A)
print(B)
```

```
[[2, 3], [1, -1]]
[6, 0.5]
```

[5]:
```
# Normalize row 1
norm = A[0][0]
for i in range(len(A[0])): A[0][i] /= norm
B[0] = B[0]/norm
print(A)
print(B)
```

```
[[1.0, 1.5], [1, -1]]
[3.0, 0.5]
```

[6]:
```
for i in range(len(A[1])):
    A[1][i] = A[1][i] - A[0][i]
B[1] -= B[0]
print(A)
print(B)
```

```
[[1.0, 1.5], [0.0, -2.5]]
[3.0, -2.5]
```

[7]:
```
# Eliminate row 2 - A[1] - need to check and ensure div-by-zero etc doesnt happen
norm = A[1][0] / A[0][0]
for i in range(len(A[1])): A[1][i] = A[1][i] - A[0][i] * norm
B[1] = B[1] - B[0] * norm
print(A)
print(B)
```

```
[[1.0, 1.5], [0.0, -2.5]]
[3.0, -2.5]
```

[8]:
```
# Normalize row 2 - B[1] will now contain the solution for x2
norm = A[1][1]
for i in range(len(A[1])): A[1][i] = A[1][i] / norm
B[1] = B[1] / norm
print(A)
print(B)
```

```
[[1.0, 1.5], [-0.0, 1.0]]
[3.0, 1.0]
```

[9]:
```
# Sub back and solve for B[0] <-> x1
# This can be seen as eliminating A[0][1]
norm = A[0][1] / A[0][0]
# note that len(A) will give number of rows
for i in range(len(A)):
    A[i][1] = A[i][1] - A[i][0] * norm
    B[i] = B[i] - A[i][0] * norm
print(A)
print(B)
```

```
[[1.0, 0.0], [-0.0, 1.0]]
[1.5, 1.0]
```

2.1 Problems with Gaussian elimination

There are several obvious problems with the method outlined here. These include:

- Performance - Python lists are not the most efficient way to store matrices
- Zeros: the simple example does not consider a scenario where one of the values on the diagonal may be 0. Then some shuffling of rows is required.
- Numerical stability: there are several *normalization* steps involved, where it is quite possible for the values to blow up out of control if not managed properly. Usually some kind of pivoting techniques are used to get around these issues.

[10]:
```
import numpy as np
A1 = np.array( [ [2,3], [1,-1] ] )
B1 = np.array( [6, 1/2] )
np.linalg.solve(A1, B1)
```

[10]:
```
array([1.5, 1. ])
```

3 Library functions

Numeric Python or `numpy` is a library that allows Python code to directly call highly efficient implementations of several linear algebra routines (that have been written and optimized using C or Fortran and generally offer very high performance).

Although you can use the same `list` based approach to create matrices, it is better to declare them as the `array` data type:

- the numeric `type` (float, int etc.) can be specified for arrays
- memory allocation is done more efficiently by assuming they are not meant to be arbitrarily extended or changed

4 SPICE basics

Our goal is to implement a SPICE simulator. In order to do this, we first need to read in the circuit description from a text file. To start with, we will only consider the basic elements of SPICE: Voltage sources, Current sources, and Resistors. A typical SPICE netlist would look like this:

```
.circuit
R1 GND 1 1
R2 1 2 1
V1 GND 2 dc 2
.end
```

This is basically a *netlist* with 3 *nodes* - one of which is Ground (GND) which is assumed to be have a voltage of 0V. We can write down Kirchhoff's current law (KCL) equations at each node, to account for current balance. In addition, we will have some equations that specify the voltages between nodes having a direct voltage source, since there is no resistance there to provide an equation.

For the above example, the equations will be

$$\begin{aligned} \frac{V1-0}{R1}+ \quad \frac{V1-V2}{R2} &= 0 \\ \frac{V2-V1}{R2}+ \qquad I1 &= 0 \\ V2- \qquad 0 &= 2 \end{aligned}$$

which can be written in Matrix form as:

$$\begin{bmatrix} \frac{1}{R1}+\frac{1}{R2} & \frac{-1}{R2} & 0 \\ \frac{-1}{R2} & \frac{1}{R2} & 1 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} V1 \\ V2 \\ I1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 2 \end{bmatrix}$$

At this point, you have reduced the solving of the SPICE equations to a known problem (linear equation solving) that you already know how to do.

4.1 AC sources

The assumption above is that the system consists only of Voltage or Current sources and resistors. What about capacitors, inductors, and AC sources? These can be handled in exactly the same way as long as the circuit is operating at a single frequency. We then replace the elements with their corresponding *impedance* values, which are frequency dependent complex numbers, but since there is only a single frequency they will still be constants.

4.2 Problem scenarios

- Voltage source loops
- Current sources at a node
- Circuit defined with both DC and AC sources
- Syntax errors

5 String and File manipulation

Given a SPICE netlist like the one above, you need to *read* it and construct an internal matrix as described above. For string manipulation, there are a few helpful utility functions that we can see here.

[11]:
```
circ = """.circuit
R1 GND 1 1
R2 1 2 1
V1 GND 2 dc 2
.end
""".splitlines()
for l in circ:
    if l[0] == 'R':
        print("Found a resistor")
    elif l[0] == 'V':
        print("Found a voltage source with value: ", float(l.split()[4]))
```

```
Found a resistor
Found a resistor
Found a voltage source with value:  2.0
```

5.1 Files

You can read from a file using the `readlines()` method of file objects. One thing to keep in mind is how you open and close file objects. In particular, it is strongly recommended to use the pattern `with open("filename") as f:` to ensure that the file is closed once you are done with reading it.

6 Assignment

The following are the problems you need to solve for this assignment. You need to submit your code (either as standalone Python script or a Python notebook), a PDF document explaining your solution (either generated from the notebook or a separate LaTeX document), and any supporting files you may have (such as circuit netlists you used for testing your code).

- 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.
- 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.

  - 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.
- 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.

6.1 Bonus assignments

- (Small bonus): after reading in the netlist, allow some or all sources and impedances to be controlled interactively - either using widgets or other mechanisms. On each change you should recompute the currents and voltages and display them.
- (Large bonus): make a solver that can do real-time transient simulations of a SPICE netlist and update the currents and voltages dynamically. They should also be plotted as a function of time and react to changes. This is something along the lines of https://www.falstad.com/circuit/. Ideally you should be able to do a real-time demo of some experiments you might conduct as part of a basic electronics lab, and simulate the behaviour of an oscilloscope and signal generator.

7 1) Function to find Factorial of a Number

7.1 Method 1

[12]:
```
#function to find Factorial (method 1)
def fact(x):
    try:
        if type(x) is not int:
            raise TypeError
        elif x<0:
            raise ValueError
        else:
            fact = 1
            for i in range(x):
                fact *= i+1
            return fact
    except ValueError:
        print("The number should be a non negative integer")
    except TypeError:
        print("The number is not an integer")


num = 15
print(fact(num))
%timeit fact(num)
```

```
1307674368000
1.63 µs ± 14.6 ns per loop (mean ± std. dev. of 7 runs, 1,000,000 loops each)
```

7.2 Method 2

[13]:
```
#function to find Factorial (method 2)
def fact2(x):
    if x == 1:
        return 1
    else:
        return x*fact2(x-1)

num = 15
print(fact2(num))
%timeit fact(num)
```

```
1307674368000
1.65 µs ± 24.6 ns per loop (mean ± std. dev. of 7 runs, 1,000,000 loops each)
```

7.3 Method 3

[14]:
```
#function to Find Factorial(method 3)
import math
num = 15
print(math.factorial(num))
%timeit math.factorial(num)
```

```
1307674368000
128 ns ± 6.46 ns per loop (mean ± std. dev. of 7 runs, 10,000,000 loops each)
```

7.4 Explaination for Factorial

1. In method 1 and 2 we use the iterative approach and recursive approach.
2. Method 1 is more efficient (time complexity) compared to the recursive one because each call to the recursive function requires additional memory allocation on the call stack, which leads to stack overflow for large numbers.
3. Method 3 is the best of all three as it makes use of a pre-written function from the `math` module as its short and pretty effective.

8 2) Linear Equation Solver

[15]:
```
def swap(A,b,n):
    for i in range(n):
        pivot = i
        for j in range(i + 1, n):
            if abs(A[j][i]) > abs(A[pivot][i]):
                pivot = j
        A[i], A[pivot] = A[pivot], A[i]
        b[i], b[pivot] = b[pivot], b[i]
    return A,b
```

8.0.1 swap():

- This Function is used if we have any zeros present in our input matrices
- What this function simply does is , swaps all the zeros at higher row to the lowest row

[16]:
```
def upper_tri(A,b,n):
    for j in range(n):
        for i in range(j+1,n):
            try: #check whether we have zerodivision error or not
                c=A[i][j]/A[j][j] # the normalizing coefficient
                for k in range(1,n):
                    # we use the Ri = Ri - c*Rj
                    A[i][k]=A[i][k]-c*A[j][k]
                b[i] = b[i] - c*b[j]

            except:
                if b[i] == 0:
                    print("The given input has infinite solutions!!")
                    return None
                else:
                    print("The given input has no solutions!!")
                    return None
    return A,b
```

8.0.2 upper_tri():

- This function converts the matrix into a Upper Triangular Matrix

[17]:
```
def back_sub(A,b,n):
    res = [0 for i in range(n)]  # will contain the result of linear equations
    try:
        res[n-1] = b[n-1]/A[n-1][n-1]
        for i in range(n-2,-1,-1):
            sum=0
            for j in range(i+1,n):
                sum=sum+A[i][j]*res[j]
            res[i]=(b[i]-sum)/A[i][i]

        return res

    except:
        if b[n-1] == 0:
            print("The given input has infinite solutions!!")
            return None
        else:
            print("The given input has no solutions!!")
            return None
```

8.0.3 back_sub():

- Since, we got the upper triangular matrix we have * A[n-1][n-1]*x[n-1] = b[n-1]*
- We substitute this value in the above and calculate x[n-2] and this process goes on which is known as Back Substitution

[18]:
```
import random
def gauss_elim(A,b,n):
    swap(A,b,n)
    upper_tri(A,b,n)
    result = back_sub(A,b,n)
    return result


A1 = A = np.random.rand(10,10)
b1 = b = np.random.rand(10)
n = len(A)
A=A.tolist()
b=b.tolist()
print(A,'\n')
print(b)

try:
    for i in range(len(A)): #check for sqaure matrix
        if len(A) != len(A[i]):
            raise TypeError

    if len(A) != len(b): #check for same dimensions of both matrices
        raise TypeError

    for i in range(len(A)):
        for j in range(len(A)):
            float(A[i][j])
    print()
    print(gauss_elim(A,b,n))

except TypeError:
    print("The matrix does not have proper dimensions!!")
except ValueError:
    print("The input given is a string!!")

%timeit gauss_elim(A,b,n)
```

```
[[0.31731421718126185, 0.40506387939605437, 0.4298864241500483,
0.9534055432470077, 0.06770073986068259, 0.14713781134812842,
0.024341774688874773, 0.4034217328171993, 0.9587958476018255,
0.436080944677802], [0.47599814687597797, 0.3315051045611779,
0.48787086089340614, 0.9180214806160385, 0.1622580353293933, 0.5214796372648657,
0.09990388943105333, 0.03419992577410236, 0.3455074308354519,
```

```
0.6495074702517256], [0.6793023664373097, 0.888149498116146, 0.6034651682604211,
0.7285321284332179, 0.8681317146019799, 0.7800733008331681, 0.8045965637050055,
0.5178100957047329, 0.9667958331106321, 0.7579965279560702],
[0.6203661834072057, 0.5762162981043647, 0.04889594887745896,
0.13853943240770972, 0.035508512068110565, 0.5569696821019162,
0.6137717012728534, 0.7257484231849756, 0.8842053421845916,
0.17085480010312204], [0.5887666711154531, 0.8110937525717868,
0.48371739628742016, 0.31859130011020986, 0.892887485179279,
0.04220035747467388, 0.4158113819598803, 0.6611159655958062, 0.7214236530871815,
0.5675198579195487], [0.17035577579643146, 0.6937134415359962,
0.5649730756550035, 0.09850891167110054, 0.6128593478836132, 0.3192713841159358,
0.747012869265766, 0.2193004264901357, 0.8683486230258041, 0.8777786766106827],
[0.735344467125471, 0.2233802701756744, 0.5499469635113969, 0.848709700576779,
0.6275396723981799, 0.919205832664707, 0.7471684486894448, 0.9416266934675778,
0.679245522456006, 0.2829989285066482], [0.8295277657383112, 0.6723411742412999,
0.9903856287198879, 0.8678452787822597, 0.3753936067100142, 0.8138092557169287,
0.7125314427521522, 0.11325268114970866, 0.48414561421729774,
0.629541566369653], [0.7228018305304997, 0.7069968739786133,
0.14045509345404328, 0.7000982378597816, 0.15860671868826426,
0.08603672243103833, 0.13379638960318152, 0.2843665574029005,
0.5104657058347228, 0.5043954581114998], [0.922549785590874, 0.7758037279213476,
0.5041904775761482, 0.9218121638444087, 0.36432616912860694, 0.2506011006184202,
0.31597119408729124, 0.6537908230805717, 0.4223485325716092,
0.5893086164118468]]

[0.6463864136249334, 0.37836028906099783, 0.2573561562242821,
0.0774253957138723, 0.16306579046553438, 0.8270910099413519, 0.7220230031487045,
0.794310248467582, 0.12733405536034248, 0.011514169008597919]

[1.6304452407506573, -2.8183320328549586, 0.4669882561163116,
0.045851310091878476, -0.04858209740726221, -1.5357594958860905,
1.5255104061379667, -1.04361216152642, 1.369857605817966, 0.7473071477642833]
153 µs ± 13.9 µs per loop (mean ± std. dev. of 7 runs, 10,000 loops each)
```

[19]:
```
print(np.linalg.solve(A1,b1))
%timeit np.linalg.solve(A1,b1)
```

```
[ 1.63044524 -2.81833203  0.46698826  0.04585131 -0.0485821  -1.5357595
  1.52551041 -1.04361216  1.36985761  0.74730715]
10.9 µs ± 897 ns per loop (mean ± std. dev. of 7 runs, 100,000 loops each)
```

9 3) Solving a Circuit Netlist

[32]:
```
import math
def finding_elements(lines):
    elements = []
    for i in lines:
```

```
        if i.startswith('.current'):
            #For not going through any junk if present in the file
            continue
        if i.startswith('.end'):
            #Only checking between '.circuit' and '.end' of the file
            break
        if i.startswith('I'):  #to check for Current Source
            items = i.strip().split(" ")
            if items[3] == 'ac':
                elements.
↪append(['I',(items[1]),(items[2]),(items[3]),float(items[4]), float(items[5])])
            else:
                elements.
↪append(['I',(items[1]),(items[2]),(items[3]),float(items[4])])

        if i.startswith('V'): #to check for Voltage source
            items = i.strip().split(" ")
            if items[3] == 'ac':
                elements.
↪append(['V',(items[1]),(items[2]),(items[3]),float(items[4]) ,␣
↪float(items[5])])
            else:
                elements.
↪append(['V',(items[1]),(items[2]),(items[3]),float(items[4])])

        if i.startswith('R'): #to check for Resistor
            items = i.strip().split(" ")
            elements.append(['R',(items[1]),(items[2]),float(items[3])])

        if i.startswith('L'): #to check for Inductor
            items = i.strip().split(" ")
            elements.append(['L',(items[1]),(items[2]),float(items[3])])

        if i.startswith('C'): #to check for Capacitor
            items = i.strip().split(" ")
            elements.append(['C',(items[1]),(items[2]),float(items[3])])

    return elements
```

9.0.1 finding_elements():

- This function takes the lines read from the file as argument.
- We first create a empty list of elements where we will append all the information about the circuit.
- To check for each componnent we use the `.startswith()` method to check whether the line

has info about resistor, capacitor, etc.

- By using the `.strip.split()` method we append all the information of the components and `return elements`.

[33]:
```
def nodes_mapping(elements):
    n=len(elements)
    map1={}
    map2={}
    p=1
    for i in range(n):
        #Since the nodes are present at element[i][1] and element[i][2]
        for j in range(1,3):
            element=elements[i][j]
            if(element=='GND'):
                elements[i][j] = 0
                continue
            if(element in map1.keys()):
                elements[i][j]=map1[element]
            if(element not in map1.keys()):
                #If node is not in map1 we have element as key and give int(p)␣
↪as its value
                map1[element]=int(p)
                #Same as above, only difference is the key and value pairs are␣
↪changed
                map2[int(p)]=element
                elements[i][j]=int(p)
                p+=1
    return(map2)
```

9.0.2 nodes_mapping():

- This Function is used to write all the nodes in a sequence
- We use a dictionary to map all the nodes we have by passing `elements` as the argument to the function

[34]:
```
def readfile(file_name):
    lines = file_name.readlines()
    for i in lines:
        print(i)
    return lines
```

9.0.3 readfile():

- readfile() function takes file_name(a file) as an argument and print each line of the file

[35]:
```
def ac_freq(lines):
    freq = 0
```

```
    for i in lines:
        items = i.strip().split(" ")
        if i.startswith('.ac'):
            freq = 2*math.pi*float(items[2])
    return freq

freq = ac_freq(lines)
```

9.0.4 ac_freq():

- This function is used for checking whether we have a dc or ac source .
- And if we have, then it checks the `3rd string` of the line which starts with '.ac' .
- Finally we find the `angular frequency` which is 2π*frequency and return it.

[36]:
```
def total_nodes(elements):
    nodes = []
    voltage_sources = 0
    for element in elements:
        nodes.append(element[1])
        nodes.append(element[2])
        if element[0].startswith("V"):
            voltage_sources+= 1

    max_node = max(nodes)

    dim_matrix = max_node + voltage_sources
    return dim_matrix , voltage_sources
```

9.0.5 total_nodes():

- By iterating each element of the list elements which we got using `finding_elements()` function.
- We append each node from the elements list in a list called nodes.
- Then, we check for the voltage sources to take into account the current which goes through independent voltage sources.
- Finally, we find the maximum value of the node using the max() function and consider the dimensions of the matrix to be max_node + voltage_sources.

[37]:
```
def matrix_formation(rank_matrix):
    A = [[0 for j in range(rank_matrix)] for i in range(rank_matrix)]
    b = [0 for j in range(rank_matrix)]
    return A,b
```

9.0.6 matrix_formation():

- This function simply creates a `n x n` matrix and a `n x 1` matrix by finding n using the total_nodes() function which has all its elements initialised to zero

[38]:
```
def equation_maker(elements , dim , vs):
    for element in elements:
        if element[0] == 'V': #check for voltage source
            if len(element) == 6:
                # This means we have a AC source and thus change the value of voltage
                volt = abs(element[4])*(math.cos(freq+element[5])+math.sin(freq+element[5]))
            else:
                # This means we have a DC source
                volt = element[4]

            i = element[1] - 1
            j = element[2] - 1
            # i and j are defined to use in adding values to specified row and column
            if i == -1:   A[j][dim] -= 1
            #if ith node is ground we take -1 as coefficient of I(ind. source current)
            elif j == -1:  A[i][dim] += 1
            #if jth node is ground we take +1 as coefficient of I(ind. source current)
            else:
                A[i][dim] += 1
                A[j][dim] -= 1
                b[j] += volt
            for n in range(vs):
            #iterate through all voltage sources and add its values to the rows containing sources
                if i == -1:
            #if ith node is ground we take +1 as for the coefficient of node j
                    A[dim + n][j] = 1
                    b[dim + n] += volt
                elif j == -1:
            #if jth node is ground we take +1 as for the coefficient of node i
                    A[dim + n][i] = 1
                    b[dim + n] += volt
                else:
                    A[dim + n][i] -= 1
                    A[dim + n][j] += 1
                    b[dim + n] += volt
```

```
        if element[0] == 'I':
            if len(element) == 6:
                amp = abs(element[4])*(math.cos(freq+element[5])+math.
↪sin(freq+element[5]))
            else:
                amp = element[4]
            i = element[1] - 1
            j = element[2] - 1
            # assuming current is leaving i node we add -amp to the b matrix
            if i == -1:   b[j] -= amp
            # assuming current is entering j node we add +amp to the b matrix
            elif j == -1:  b[i] += amp
            else:
                b[j] -= amp
                b[i] += amp


        if element[0] == 'R':
            imp = element[3]
            i = element[1] - 1
            j = element[2] - 1
        #if ith node is ground we take +1/imp as for the coefficient of node j
            if i == -1:   A[j][j]+=1/imp
        #if jth node is ground we take +1/imp as for the coefficient of node i
            elif j == -1: A[i][i]+=1/imp
            else:
                A[i][i] += 1/imp
                A[i][j] -= 1/imp
                A[j][j] += 1/imp
                A[j][i] -= 1/imp

        if element[0] == 'C':
            imp = 1/(1j*freq*element[3])
            # Since the impedennce of capacitance is 1/jWC
            i = element[1] - 1
            j = element[2] - 1
            if i == -1:  A[j][j] += 1/imp
            elif j == -1: A[i][i] += 1/imp
            else:
                A[i][i] += 1/imp
                A[i][j] -= 1/imp
                A[j][j] += 1/imp
                A[j][i] -= 1/imp

        if element[0] == 'L':
            imp = 1j*element[3]*freq
            # Since the impedence of inductor is jWL
```

```
            i = element[1] - 1
            j = element[2] - 1
            if i == -1:  A[j][j]+= 1/imp
            elif j == -1: A[i][i]+= 1/imp
            else:
                A[i][i] += 1/imp
                A[i][j] -= 1/imp
                A[j][j] += 1/imp
                A[j][i] -= 1/imp


    return A,b
```

9.0.7 equation_maker():

- We first take elements, dim , vs as arguments.
- `dim , vs` are the total nodes and voltage sources of the circuit.
- We add the coefficients of the eqautions formed by the voltage sources and also add the coefficients of currents in the corresponding nodes.
- Finally for the impedence, we check for ground and check for each node and write the coefficients of the equations in the matrix for those corresponding nodes.
- Finally, we return the matrix `A, b`

[39]:
```
netlist = open('ckt7.netlist', 'r')
lines = readfile(netlist)
```

```
Parallel RLC circuit with Current Source



.circuit

I1 GND n1 ac 5 0 # Current source with Magnitude 5 V, phase 0

C1 GND n1 1

R1 GND n1 1000

L1 GND n1 1e-6

.end

.ac I1 1000      # Frequency of operation is 1000 Hz
```

[40]:
```
elements = finding_elements(lines)
for element in elements:   print(element)
print()

nodes_mapping(elements)
dim , voltage_sources = total_nodes(elements)
nodes = dim - voltage_sources

print()
A,b = matrix_formation(dim)
A,b = equation_maker(elements, nodes , voltage_sources)
for a in A: print(a)
print()
print(f'{b} \n')
```

```
['I', 'GND', 'n1', 'ac', 5.0, 0.0]
['C', 'GND', 'n1', 1.0]
['R', 'GND', 'n1', 1000.0]
['L', 'GND', 'n1', 1e-06]


[(0.001+6124.03036408769j)]

[-4.999999999996786]
```

[41]:
```
print(gauss_elim( A, b , len(A)))
%timeit gauss_elim(A,b,len(A))
```

```
[(-1.33320008797332e-10+0.0008164557820152991j)]
2.68 µs ± 99.3 ns per loop (mean ± std. dev. of 7 runs, 100,000 loops each)
```

9.0.8 Note:

- Here, I have assumed if we go from node 1 to node 2, node 1 is assumed to have higher voltage and node 2 to have lower voltage.
- Hence , +1/imp is given to node 1 and -1/imp is given to node 2
- Similarly for Current , current going out of node 1 is assumed to be positive and current entering node 2 is assumed to be of negative sign