Assignment 2
Spice - Part 2

Applied Programming Lab (EE2703)

January 29, 2019

1 Introduction

In the last assignment, we have seen the basics of python. In this assignment, you will get introduced to scientific python. Make sure you have completed all the previous homework as well as the assignment before getting started with this assignment.

2 OOP and Python

Just like C++, classes help you implement OOP in Python. Classes are the blueprint of objects. They specify what the object can have. For example, say you have a class for Player. The class will tell you that your object has a property named "position". But does not say what the "position" is. This is useful since a class can parent as many different objects as you want. Each object can have a different "position".

```
In [1]: class Player:
   ...:     position_x = 0
   ...:     position_y = 0
   ...:
   ...:     def __init__(self, color):
   ...:         self.color = color
   ...:
   ...:     def move(self, distance, direction):
   ...:         if direction == 'right':
   ...:             self.position_x += distance
   ...:         elif ...
   ...:             ...
```

Here, we defined a class Player. Let's go line by line.

1. The first line says the it's a definition of a class and the name of the class is Player
2. The next two lines define features of the class. The player has x and y positions. These values are set to 0 by default.
3. `__init__()` is the constructor. The first argument to it is always `self`. `self` refers to the object itself!
4. The constructor takes in one extra argument `color`. The interesting thing here is that, the constructor is declaring a new feature for the class at run-time! In python, you can add features to an object dynamically!
5. `move()` here is a member function of the class. Note that the first argument to a member function should be `self`.

```
In [2]: p1 = Player('red')   # Creating a player object

In [3]: p1.position_x
Out[3]: 0

In [4]: p1.move(2, 'right')  # Calling a member function

In [5]: p1.position_x
Out[5]: 2
```

Note that, we did not provide the parameter `self` here. Since first argument to member function is always `self`, Python passes it automatically.

Though we are seeing Python classes for the first time, you have been using the power of classes even without realizing it. For example, lists that you have been creating are actually objects of the class "List".

```
In [6]: l = [1, 4, 5, 2, 3]

In [7]: l.sort()
```

Here, you are creating an object of the class list in the first line (`[...]` is the constructor here) and calling a member function of the object in the second line.

3 List unpacking

One peculiar thing about Python is that you can return multiple values from a function!

```
In [8]: def f():
   ...:     return 1, 2, 3, 4
```

```
In [9]: r = f(); r
Out[9]: (1, 2, 3, 4)

In [10]: w, x, y, z = f()
```

`f()` is returning four values here. How the returned values get stored is heavily depended on the calling statement. In the first call to `f()`, Python will compile all the return values and give to us a tuple. But in the next call, return values are assigned to separate variables in the order mentioned. What will happen if you try `x, y = f()`? What about `x, *y, z = f()` ? This is called extended unpacking.

Python goes even further:

```
In [11]: a, b, c = [1, 2, 3]

In [12]: i = 1; j = 2

In [13]: (i, j )= (j, i)
```

What are the values of `i` and `j` now?

3.0.1 Homework

Given a list

```
l = [ [1, 2], 'hello' ]
```

How can you extract the first character of `'hello'` only by using list unpacking? Do it in single statement.

a,(b,c,d,e,f) = l;b

first letter of 'hello' is stored in b

4 Scientific Python

What makes python amazing is the huge collection of modules that come with it. For example, Modules like scipy, numpy, matplotlib, sympy and panda make it suitable for scientific computing. numpy helps with numerical computations, scipy provides special functions that will help you with scientific computing and also adds complex number support to all the functions. matplotlib provides sophisticated plotting capabilities.

Most of the underlying implementations of these libraries are in C. This ensures the efficiency of these implementations. But to make your life easy, these C implementations are wrapped in Python and given as python functions which you can directly call from python. Which means they are efficient at the same time easy to use.

4.1 Arrays vs Lists

While Python lists are amazing by themselves, they are not always the best data type to depend. The flexibility they provide makes them slow. To make things faster, numpy provides array data type. numpy arrays are lot similar to C arrays. As I already mentioned, the speed come from the fact that the underlying implementation is in C.

Let's see some examples of arrays:

```
In [14]: from numpy import *

In [15]: array(range(10))
Out[15]: array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9]) # Note the way it is printed.
# The output is explicitly telling you that this is an array and not a list.

In [16]: A = array([[1,2,3],[4,5,6],[7,8,9]]); A
Out[16]:
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])

In [17]: A * 10              # Multiplication with a scalar
Out[17]:
array([[10, 20, 30],
       [40, 50, 60],
       [70, 80, 90]])

In [18]: A * A               # Element by element multiplication
Out[18]:
array([[ 1,  4,  9],
       [16, 25, 36],
       [49, 64, 81]])

In [19]: A + 10              # Addition with a scalar
Out[19]:
array([[11, 12, 13],
       [14, 15, 16],
       [17, 18, 19]])

In [20]: A + A              # Addition with a vector
Out[20]:
array([[ 2,  4,  6],
       [ 8, 10, 12],
       [14, 16, 18]])

In [21]: ones(10)           # Try "ones?" and see what does it do
Out[21]: array([1., 1., 1., 1., 1., 1., 1., 1., 1., 1.])

In [10]: B = array([ones(3) for i in range(3)]); B
Out[10]:
```

```
array([[1., 1., 1.],
       [1., 1., 1.],
       [1., 1., 1.]])

In [22]: dot(A,B)
Out[22]:
array([[ 6.,  6.,  6.],
       [15., 15., 15.],
       [24., 24., 24.]])
```

Now, try out the same operations with lists and see what happens. Here is an example:

```
In [23]: a = [list(i) for i in A]; a
Out[23]: [[1, 2, 3], [4, 5, 6], [7, 8, 9]]

In [24]: a + a
Out[24]: [[1, 2, 3], [4, 5, 6], [7, 8, 9], [1, 2, 3], [4, 5, 6], [7, 8, 9]]
```

List uses '+' for concatenation. maybe "__add__()" function is used for concatenation?

arrays will give you what a mathematician would expect, while list is more for a general programming use. Arrays are quite fast compared to lists. Here are some important facts about arrays:

- Array elements are all of one type, unlike lists. This is precisely to improve the speed of computation.
- An array of integers is different from an array of reals or an array of doubles. So you can also use the second argument to create an array of the correct type.
  Eg:

  ```
  x=array([[1,2],[3,4]], dtype=complex)
  ```

- Arrays are stored row wize by default. This can be changed by setting some arguments in numpy functions. This storage is consistent with C.
- The size and shape methods give information about arrays. In above examples,

  ```
  x.size   # returns 4
  x.shape  # returns (2, 2)
  len(x)   # returns 2
  ```

  `size` gives the number of elements in the array. `shape` gives the dimensions while `len` gives only the number of rows.
- Arrays can be more than two dimensional. This is a big advantage over Matlab and its tribe. Scilab has hypermatrices, but those are slow. Here arrays are intrinsically multi-dimensional

- The `dot` operator does tensor contraction. The sum is over the last dimension of the first argument and the first dimension of the second argument. In the case of matrices and vectors, this is exactly matrix multiplication.

Note that `numpy` also has a `matrix` data type. But this is not recommended to use. In fact, the community is planning to remove it in future.

4.2 `where()`

Sometimes we want to know the indices in a matrix that satisfy some condition. The method to do that in Python is to use the `where` command. To find the even elements in the above matrix we can do:

```
In [25]: A=array([[1,2,3],[4,5,6],[7,8,9]]);A
Out[25]:
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])

In [26]: coords = where(A%2==0) # Returns the coords of even elements

In [27]: coords                 # This is a tuple
Out[27]: (array([0, 1, 1, 2]), array([1, 0, 2, 1]))

In [28]: i, j = where(A%2==0)

In [29]: B=array([[6,6,6],[4,4,4],[2,2,2]])

In [30]: i, j = where((A>3)&(B<5)>0)
```

What does the last line here do? Break it down to pieces and understand what each piece does. Can you replace the `&` with `and`? Why not?

Here is another peculiar thing about arrays.

```
In [31]: A
Out[31]:
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])

In [32]: i, j = where(A%2==0); i
Out[32]: array([0, 1, 1, 2])

In [33]: j
Out[33]: array([1, 0, 2, 1])
```

```
In [34]: A[i, j]
Out[34]: array([2, 4, 6, 8])
```

Notice what is happening in the last command. Can you do the same with lists?

4.2.1 Homework

- Explain what is happening in the last command here to your TA.

```
In [35]: A
Out[35]:
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])

In [36]: A[i][j]
Out[36]:
array([[4, 5, 6],
       [1, 2, 3],
       [4, 5, 6],
       [4, 5, 6]])
```

- Generate an `array` (of shape (3,3)) of random numbers.

5 Solving Linear Equations

Solving linear equations are quite simple using numpy. Let's try and solve:

$$3x_0 + x_1 = 9 \tag{1}$$

$$x_0 + 2x_1 = 8 \tag{2}$$

The above equation can be written as:

$$Ax = b \tag{3}$$

Where,

$$A = \begin{bmatrix} 3 & 1 \\ 1 & 2 \end{bmatrix} \tag{4}$$

$$x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \tag{5}$$

$$b = \begin{bmatrix} 9 \\ 8 \end{bmatrix} \tag{6}$$

All you need to do is to create the matrices:

```
In [37]: import numpy as np

In [38]: A = np.array([[3,1], [1,2]]); A
Out[38]:
array([[3, 1],
       [1, 2]])

In [39]: b = np.array([9,8]); b
Out[39]: array([9, 8])

In [40]: x = np.linalg.solve(A, b); x
Out[40]: array([2., 3.])
```

The `numpy.linalg.solve()` solved the set of equations for us.

Note that the first line is same as "`import numpy`". But "`as np`" here allows you to write `np.array()` instead of `numpy.array()`. This is simply to save some typing efforts.

6 Spice - Part 2

In the last assignment, we read a netlist file and parsed it. In this assignment, we will solve the circuits. We have n node voltages (say V_i ; $i = 0$ *to* $n-1$) and k currents (say I_{ij}) (the currents through the k voltage sources). The system of equations are:

1. n KCL equations at the n nodes.
2. k equations of the type $V_i - V_j = \epsilon_{ij}$ for nodes connected by voltage sources.

As long as a loop consisting purely of voltage sources or a node connected purely to current sources does not exist, a unique solution is possible. Write the equations in the form

$$\boldsymbol{A} \cdot \vec{V} + \boldsymbol{B} \cdot \vec{I} = \vec{b} \tag{7}$$

The node equations take the form

$$\sum \frac{V_i - V_j}{Z_{ij}} + \sum I_{ij} = -\sum \mathscr{I}_{ij} \tag{8}$$

where the first sum is over the passive branches, the second sum over voltage sources and the third sum is over current sources.

The auxiliary equations take the form

$$V_i - V_j = \mathscr{E}_{ij} \tag{9}$$

The dependent sources have similar equations that you can easily derive for yourself.

- Explain how a loop consisting purely of voltage sources will result in the system of equations becoming inconsistent.
- Explain how a node connected purely to current sources will result in the system of equations becoming inconsistent.

Now, write the equation 7 in the following form:

$$Mx = b \tag{10}$$

i.e.,

$$\begin{pmatrix} a_{11} & \dots & a_{1n} & b_{11} & \dots & b_{1k} \\ \dots & \dots & \dots & \dots & \dots & \dots \\ a_{n1} & \dots & a_{nn} & b_{n1} & \dots & b_{nk} \\ \dots & \dots & \dots & 0 & 0 & 0 \\ \dots & \dots & \dots & 0 & 0 & 0 \\ \dots & \dots & \dots & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} V_1 \\ \dots \\ V_n \\ I_1 \\ \dots \\ I_k \end{pmatrix} = \begin{pmatrix} \mathscr{I}_1 \\ \dots \\ \mathscr{I}_n \\ \mathscr{E}_1 \\ \dots \\ \mathscr{E}_k \end{pmatrix} \tag{11}$$

This matrix needs to be solved to obtain currents and voltages.

Note that this can be readily solved using the method specified in section 5.

7 Assignment

1. Parse the netlist and create list(s) of different components. You may define class(es) for components and create objects for each component to store them nicely. Component name, connected nodes, value etc... can be the features of this(these) class(es). Remember that list elements can be class objects too.
2. Create a table of distinct nodes present in the circuit. Assign numbers to the nodes so that they correspond to the rows (columns) of the incidence matrix. You may use a dictionary for doing this. You could then store the node number as the value with the node name as the key.

   Note: You can assume that ground node is always named as `GND`. The ground node will add an extra equation:

   $$V_k = 0 \tag{12}$$

   where k is the node number of `GND`. You may keep it as V_0 always.
3. Construct matrices M and b using numpy arrays.
4. Solve the equation $Mx = b$.
5. Print all the unknowns. (All node volatges and current through voltage sources)
6. Solve the following circuits with your program:

(a) A simple resistive circuit (use $R_L = 1, 10$ and 100Ω)

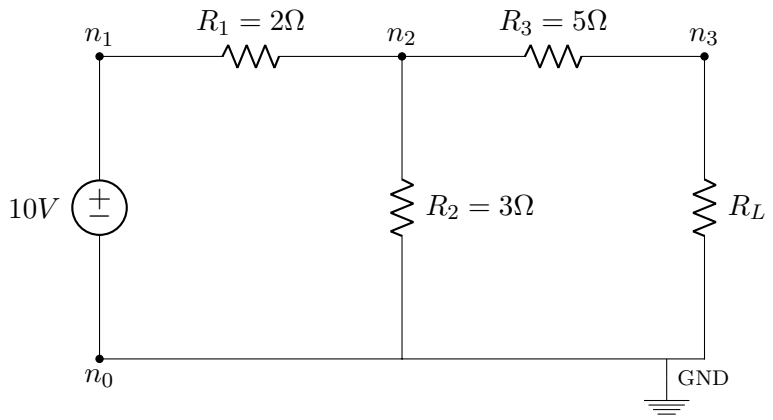

Figure 1: Resistive circuit

7. Can you use the same techniques to solve for ac circuits? We only have to interpret the impedance as complex numbers and the solution will follow. We will only work with circuits with single frequency at steady state.

   Till now, we only had a single command in the netlist (`.circuit ... .end`). We now allow a new command `.ac`.

```
.ac  V...  frequency
```

   This is a single line command. It will appear after the `.circuit ... .end` block and specify the frequency of a voltage source.

   We will also modify the way voltage source and current values are given. We will use the following representations:

```
V... n1 n2 ac  V_{p-p}  phase  # ac voltage source
V... n1 n2 dc  v              # dc voltage source
```

   Similar representations will follow for current sources.

   Solve the following circuits:

   (a) A band-pass filter for the current in the resistor

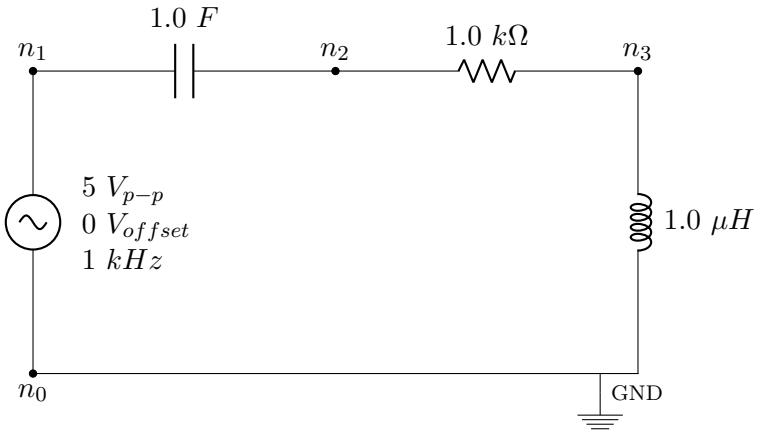

Figure 2: Band-pass filter

(b) A low-pass filter, for the voltage across the load resistor

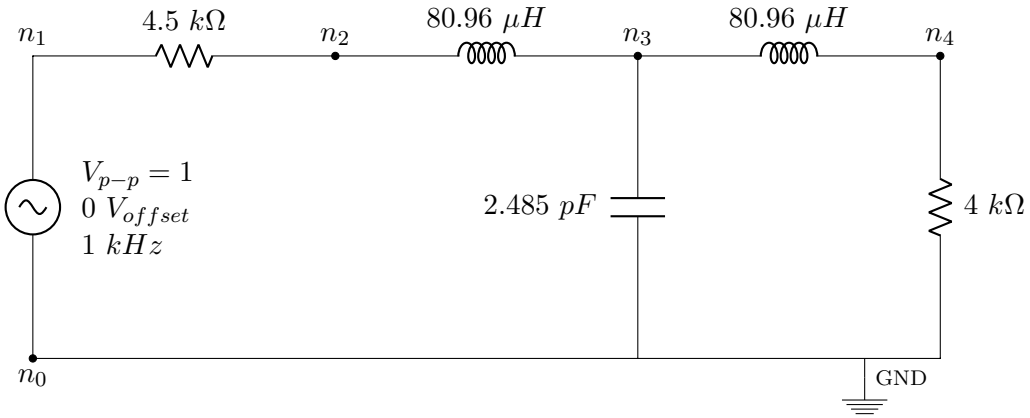

Figure 3: Low-pass filter

Submit a single program on Moodle. It should be able to solve ac as well as dc circuits.