Spice Simulation

Keshav Saravanan, EE23B035

September 2024

In this assignment, I write a program to solve for the voltage at each node and current through voltage branches in a given circuit.

1 .ckt Files

A general .ckt file looks like the following:

```
.circuit
Vs n1 GND dc 10
Is n2 GND dc 1
R1 n1 n2 2
.end
```

In this case, the voltage at `n1` is 10V higher than the `GND` voltage. The `Is` current source allows a 1A current from `n2` to `GND`.

2 Approach

Straight-forward nodal analysis would generate a conductance matrix, which we would typically apply supernode analysis on, to solve quickly. Supernode analysis is computationally faster - We break down the system of equations into two smaller sets, and since linear equations are solved in `O(N^3)` time, this results in a speedup.

However, supernode analysis involves a lot of edge cases where it fails (such as a current source connected across the terminals of a voltage source). Moreover, the circuits we are working with are quite small and to actually see gains in performance, the number of elements in the circuit would have to be of the order of atleast 10ˆ3 (which is quite unreasonable). Hence, I decided to scrap this approach and instead implement the recommend algorithm in the assignment instructions.

The number of equations in this approach is `N` + `V`, where `N` is the number of nodes and `V` is the number of voltage sources. To generate the equations we apply KCL at each node: I first generate the conductance matrix and insert the I (current source values) into the last column of the augmented matrix. Then, for every voltage source, we include a term to account for the current through the voltage source in that node's KCL equation, and in another equation, write the voltage difference across the voltage source.

I now have an equation of the form `Gx = I`. I first check if the `G` matrix is singular, and if not, invert it and multiply with `I` to obtain x

I approached the code with the object-oriented paradigm - this makes it very easy to scale to include different elements and features. For example, all the current testcases involve only DC sources. It is possible to extend this simulator to solve for circuits that involve both DC and AC circuits, using super-position principle (which I did not implement in this submission because it fails the unit tests) and return the output as a tuple of (DC, AC) components of the voltages and currents.

My code consists of an `ElementType(Enum)`, an `Element`, and a `Circuit` class. The `ElementType` class is used to keep track of the type of a circuit element (resistor/current source/voltage source). The `Element` class is a generalisation of circuit elements. The `Circuit` class represents the whole circuit.

The .ckt file is first parsed with `Circuit.read_circuit()`, and a graph of all connections is generated. The augmented matrix is then generated with `Circuit.generate_eqns()` - I iterate over every circuit element and accordingly populate the matrix entries. Finally, the system of equations is solved with `Circuit.solve()` and the solutions stored in `Circuit.voltage_solns` and `Circuit.current_solns`.

3 Special Cases Implemented

My submission supports only exclusively DC circuits or exclusively AC circuits. If a circuit containing both AC and DC sources, an error is thrown.

If a 0-valued resistance is provided, and error is raised. However, 0-valued current and voltage sources are allowed, as are negative resistances.