Simplified SPICE Simulator

Sahithi.P – EE22B080

This Python script allows you to simulate electrical circuits described in the SPICE format and solve for node voltages and current through branches containing voltage sources using matrix methods. This README provides instructions on how to use the script and details about its functionality.

Input Format

Your SPICE circuit file with an extension `.ckt`should adhere to the following format:

```
.circuit

Voltage source (V): V<(name)> <(node1)> <(node2)> dc <(value)>

Current source (I): I<(name)> <(node1)> <(node2)> dc <(value)>

Resistor (R): R<(name)> <(node1)> <(node2)> <(value)>

.end
```

- Node labels can be anything and GND for reference node (Ground) (e.g., "1", "2", "GND").
- Comments are allowed and should start with the "#" character.
- Only the elements which are present within the `.circuit` and `.end` blocks will be considered as input and are evaluated by this script. The text outside the `.circuit` and `.end` block will be ignored.
- For voltage sources, the first node (node1) is treated as the higher potential node and the second node (node2) will be the lower potential one.
- For current sources, the direction of current is from the second node (node2) to first node (node1).

Working of code

1. Importing `numpy` library to solve system of equations.
2. Defining a function to create matrix A (**matrixA** function):

- This function calculates and returns matrix A by deleting the row and column corresponds to the GND label. Matrix A is essential for solving the system of equations in circuit analysis.

3. Defining a function to create matrix B (**matrixB** function):

- This function calculates and returns matrix B based on the circuit's voltage sources and external current sources. Matrix B is also used in solving the system of equations.

4. Defining the main function **evalSpice(filename)**:

- This function takes the filename of a SPICE circuit description file as input and performs the circuit analysis.

5. Initializing empty lists and dictionaries to store circuit information:

- **vsources**, **isources**, and **resistors** store information about voltage sources, current sources, and resistors in the circuit.
- **nodesckt** and **nodenums** are dictionaries used to keep track of unique nodes and assign node numbers.

6. Reading the circuit description from the input file:

- The code reads the lines from the input file, ignoring comments, and extracts the circuit description between the `.circuit` and `.end` markers.

7. Validating the circuit description:

- Checking whether the circuit description is well-formed, including the format of each line.

8. Processing circuit elements:

- Loop through each line in the circuit description, extract information about circuit elements (voltage sources, current sources, and resistors), and populate the respective lists.
- Unique nodes are tracked in `nodesckt`, and each node is assigned a node number in `nodenums`.

9. Creating matrices A and B:

- Call **matrixA** and **matrixB** functions to calculate matrices A and B based on the circuit elements and node information.

10. Solving the system of equations:

- Using NumPy's `np.linalg.solve()` to solve the system of linear equations represented by matrices A and B.
- If a solution exists, it inserts the ground node value (0V) into the result.

11. Organizing and return results:

- Store the node voltages and currents through voltage sources in dictionaries (**voltages_nodes** and **currents_vsources**).
- Return these dictionaries as the output of the function.

12. Error handling:

- The code includes error handling for cases such as malformed circuit files, missing input files, invalid circuits, or circuits with no solutions.

Matrix Formation

Equations necessary to frame the matrix A are obtained by using Kirchoff's Laws. The elements in matrix A will be in the form of

For $i, j = range(nodenums)$; $a_{ii} = \sum \frac{1}{R_{i*}}$; $a_{ij} = -\sum \frac{1}{R_{ij}}$

For $p, q = range(len(nodenums), len(vsources))$; $a_{pk} = 1.0$ if node1 is having the index `p`, and $a_{qk} = -1.0$ if node2 is having the index `q`, where 'k' is the count of (len(nodnums) + index of the source).

For $m = range(len(nodenums))$; b_m = External current entering to the node with label 'm'.

For $n = range(len(nodenums), len(vsources))$; b_n = Value of the voltage source with positive node labelled as 'n'.

Error Handling

The script handles errors and exceptions as follows:

- If the script encounters issues with the input or circuit description, it will raise relevant exceptions, such as **'ValueError'** for malformed circuit files or **'FileNotFoundError'** if the specified file does not exist.
- If the circuit has no solution (e.g., due to a short circuit or conflicting sources), it will raise a **'LinAlgError'**.

Other Errors:

- Checking for malformed circuits i.e., with no markers, invalid components.
- Checking for GND Node.
- LinAlg.solve() couldn't able to find the solution for the circuits having Parallel Voltage sources and series current sources.