SPICE Simulation - Code Explanation

Lodha Soumya Sachin

EE23B140

0.1 Loading relevant data from the file

```
def evalSpice(filename):
if not filename or not os.path.isfile(filename): #check for valid file
    raise FileNotFoundError("Please give the name of a valid SPICE file as input")

components = []
nodes = set()
check_components={}
within_circuit = False # set flags

with open(filename, "r") as file: #read the file
    for line in file:
        line = line.strip()

        if line.startswith(".circuit"): #set the flags if data is within the .circuit block
            within_circuit = True
            continue
        elif line.startswith(".end"): #break if you hit .end
            break
```

1. We make use of a flag (within circuit) to make sure that only the data within the .circuit and .end block is considered.

2. Also raises error if it is an invalid SPICE file and creates empty dictionary,list and set which we will use later in the code.

0.2 Creating a dictionary

```
if within_circuit and line: #execute code only for circuit elements within the circuit block
            parts = line.split()
            # Extract component details based on the number of parts
            if len(parts) >= 4:
                name = parts[0]
                node1 = parts[1]
                node2 = parts[2]
                value = parts[3]

```

```
                    # Determine the type of the component (V for voltage source, R for resistor, etc.)
                    component_type = name[0].upper()
                    if component_type == "V" or component_type == "I":
                        if parts[3] != "dc" and parts[3] != "ac":
                            raise ValueError("Malformed circuit file") #raise error if type of source not defined
                        value = parts[4]
                    if (
                        component_type not in ('V', 'I', 'R')
                    ):
                        raise ValueError("Only V, I, R elements are permitted") # raise error if other components detected

                nodes.add(node1) #make a set of all the nodes
                nodes.add(node2)
                #create the dictionary
                component = {
                    "type": component_type,
                    "name": name,
                    "nodes": tuple(([node1, node2])),
                    "value": value,
                }

                components.append(component)
                components = sorted(components, key=lambda component: (component["type"] != "V", component["type"]))

                node_mapping = {"GND": 0} # GND is always assigned to 0
                node_number = 1

                for node in sorted(nodes): #map all the nodes to specific numbers
                    if node != "GND": # Skip GND since it's already assigned
                        node_mapping[node] = node_number
                        node_number += 1
        else:
            if not within_circuit:
                raise ValueError("Malformed circuit file")
```

1. If the within circuit flag is set, split the line into parts to gather description of all components.

2. Assign the parts to relevant keys in the dictionary and change the part for value if the component is a voltage source or current source as it will have dc/ac specification in the file as well.

3. Give error if dc/ac is not specified for sources.

4. Give error if any component type other than V,I and R is found.

5. Make a set of all the nodes and map them to numbers in the node mapping dictionary.

6. Raise error if the circuit file has circuit elements outside the .circuit block.

7. Also sort the component dictionary so that all the voltage sources are at the top followed by resistors as this might mess up with output return later if voltage source comes after resistances(as in test 4.ckt)

0.3 Setting up matrix B

```
# Count the number of voltage source components
        v_components = sum(1 for component in components if component["type"] == "V")

        # Determine the number of nodes
        num_nodes = len(node_mapping)

        # Create the matrix A
        A_dim = num_nodes - 1 + v_components
        A = np.zeros((A_dim, A_dim), dtype=float)
        B = np.zeros(A_dim, dtype=float) # Create a null vector B with dimensions A_dim
        # Fill B with the values of the voltage sources
        if v_components != 0:
            v_index = num_nodes - 1 # Start index for voltage sources in the B vector

            for component in components:
                if component["type"] == "V":
                    B[v_index] = float(component["value"]) # Add the value of the voltage source to B
                    v_index += 1 # Move to the next index for the next voltage source
                if component["type"] == "I":
                    node_pos, node_neg = component["nodes"]

                    i = node_mapping[node_pos]
                    j = node_mapping[node_neg]

                    # Set up the rows and columns corresponding to the voltage source
                    if i != 0:
                        B[i - 1] = -float(component["value"]) # Positive terminal
                    if j != 0:
                        B[j - 1] = float(component["value"]) # Negative terminal
```

1. Determine dimensions of matrix A and B based on the number of nodes and voltage sources in the given circuit.

2. Create null arrays of given dimensions using numpy.

3. B will be a null matrix except for the values of voltage sources starting from the index after the number of nodes are taken into account.

4. If there are current sources in the circuit, they will also be reflected in B but with opposite polarity to that of voltage sources.

5. Also if one of the nodes for the given sources is GND, it would not be reflected in our matrix so check if i or j is not equal to zero and then only fill the matrix.

0.4 Setting up matrix A

```
if component["type"] == "R": #Fill A with the values of conductance
    node1, node2 = component["nodes"]
    resistance = float(component["value"])
    conductance = 1 / resistance

    i = node_mapping[node1]
    j = node_mapping[node2]

    # Update diagonal elements (self-node conductance sums)
    if i != 0:
        A[i - 1, i - 1] += conductance # Node i conductance
    if j != 0:
        A[j - 1, j - 1] += conductance # Node j conductance

    # Update off-diagonal elements (between nodes i and j)
    if i != 0 and j != 0:
        A[i - 1, j - 1] -= conductance
        A[j - 1, i - 1] -= conductance

if component["type"] == "V":
    v_index = num_nodes - 1 # Start index for voltage sources in A
    node_pos, node_neg = component["nodes"]

    i = node_mapping[node_pos]
    j = node_mapping[node_neg]

    # Set up the rows and columns corresponding to the voltage source
    if i != 0:
        A[v_index, i - 1] = 1 # Positive terminal
        A[i - 1, v_index] = 1 # Positive terminal
    if j != 0:
        A[v_index, j - 1] = -1 # Negative terminal
        A[j - 1, v_index] = -1 # Negative terminal

    v_index += 1
```

1. Update the values in the matrix with conductance (1/resistance) for the nodes connected to the respective resistors. Depending on the position in the matrix, some will be positive while some will be negative values.

2. Matrix need not be updated if one of the nodes is GND for that node.

3. For a voltage source again find the relevant nodes and map them according to convention as positive and negative.

0.5 Solve and return the solution

```
# Solve the system of equations
       if np.linalg.det(A)==0:
          raise ValueError('Circuit error: no solution') #error if there are two
               different current/voltage sources connected across two same
              nodes

       solution = np.linalg.solve(A, B)
       V = {node: solution[i - 1] for node, i in node_mapping.items() if node !=
            "GND"} #create dictionary to fit the format of final answer
       V["GND"] = 0.0
       I = {
          component["name"]: solution[num_nodes - 1 + i]
          for i, component in enumerate(components)
          if component["type"] == "V"
       }

   return (V, I)
```

1. Raise error if A is a singular matrix which happens in cases when two voltage sources with different values are connected between the same nodes or two current sources with different values flow through one branch.

2. Solve the matrix equation Ax=B for matrix x using the np.linalg.solve() function.

3. For returning solution create 2 dictionaries - for node voltages, map every node to the first n values in the matrix x(n=number of nodes-1) and assign GND separately.

4. For current through the voltage sources, map the name of the source to the remaining part of the solution matrix and return the two dictionaries.