Assignment 2: SPICE Simulation

Problem Statement

In this assignment, I developed a function to analyze and solve electrical circuits described using SPICE (Simulation Program with Integrated Circuit Emphasis) format. The goal was to create a Python function that processes a SPICE file to compute node voltages and branch currents for given circuits.

Requirements

The function `evalSpice(filename)` was designed to fulfill the following requirements:

1. **Input Handling:**

    - The function accepts a filename containing the SPICE circuit description.
2. **File Parsing:**

    - The file is read and parsed to extract various circuit components, such as resistors (R), voltage sources (V), and current sources (I).
    - Each component's details, including component name and the nodes it connects, are stored for further processing.
3. **Node Mapping:**

- Nodes in the circuit are listed and assigned unique consecutive numbers.
- The node named `GND` is assigned the number 0, and other nodes are numbered sequentially starting from 1.
- A dictionary is used to map node names to these numerical identifiers.

4. **Matrix Formation:**

   - A matrix representation for the circuit equations is constructed using `numpy` arrays.
   - The matrix is populated based on the parsed component data and their connections in the circuit.

5. **Equation Solving:**

   - The system of equations is solved using `numpy.linalg.solve` to determine the voltages at each node and the currents through each branch.

6. **Return Values:**

   - The function returns two dictionaries:
     - `V`: Voltages at each node (e.g., `V['GND']` or `V['n1']`).
     - `I`: Currents through each voltage source (e.g., `I['Vs']`).

7. **Error Handling:**

   - The function includes error handling for various issues, such as:
     - Malformed circuit descriptions.
     - Invalid circuit configurations like loops of voltage sources or nodes with all current

  - sources entering.
  - Parsing errors due to incorrect or missing parameters.

Output Format

- **Voltages Dictionary (`V`):**
  - Key: Node name (string)
  - Value: Voltage at the node (float)
- **Currents Dictionary (`I`):**
  - Key: Voltage source name (string)
  - Value: Current through the voltage source (float)

Submission

- The solution is implemented in the file `evalSpice.py`, which includes detailed comments explaining the approach and implementation.
- Provided `pytest` test cases are used to validate the implementation. The test cases should not be modified, as they are used for grading.
- Additional helper functions were included as necessary, but the function `evalSpice` remains the primary focus.
- A `README.pdf` file is submitted alongside the code, detailing the solution approach, references, and any special cases or additional tests implemented.

Theoretical Background

Circuit Representation and Analysis

In circuit analysis, a circuit is often represented as a graph consisting of nodes and edges. Each node in this graph represents a junction where two or more circuit elements meet.

Figure 1: Simple Circuit

Consider a simple circuit with three nodes labeled 1, 2, and GND. In this context, the GND node is a special reference point with zero potential. The choice of GND is crucial because voltages are always relative to a reference point, and selecting one node as the ground simplifies the analysis.

This circuit includes:

- A fixed independent voltage source (V_s)
- Two resistors (R_1) and (R_2)

The objective is to determine the voltages at nodes 1 and 2 and the current through the voltage source (I_s).

Unknown Values and Equations

When analyzing a circuit with (N) nodes, we need to solve for (N) unknown voltages. Although the voltage at the GND node is known to be 0 V, it is still included in the equations for completeness.

Challenges:

- Kirchhoff's Current Law (KCL) equations at all (N) nodes provide (N) equations. However, these equations are redundant as they effectively provide

only ($N-1$) independent equations. This is because the KCL equations for nodes 1 through ($N-1$) implicitly satisfy the GND node equation.
- The current through a voltage source is not directly determined by the voltage value alone, creating an additional unknown. To resolve this, auxiliary equations are added for each independent voltage source, which relate the voltages at the end nodes.

Example:

For a circuit with resistors (R_1) and (R_2) and a voltage source (V_s), the relevant equations are:

1. **Current Balance Equation:**
   $$\begin{matrix} I_s + \frac{V_1 - V_2}{R_1} = 0 \end{matrix}$$
   $$\begin{matrix} \frac{V_2 - 0}{R_2} + \frac{V_2 - V_1}{R_1} = 0 \end{matrix}$$

2. **Voltage Equation:**
   $$\begin{matrix} V_1 - 0 = V_s \end{matrix}$$

These equations can be expressed in matrix form as:
$$\begin{array}{ccc} \left[\begin{matrix} \frac{1}{R_1} & -\frac{1}{R_1} & 0 \\ -\frac{1}{R_1} & \frac{1}{R_1} + \frac{1}{R_2} & 0 \\ 0 & 0 & 1 \end{matrix}\right] & \left[\begin{matrix} V_1 \\ V_2 \\ I_s \end{matrix}\right] & = \left[\begin{matrix} 0 \\ 0 \\ V_s \end{matrix}\right] \end{array}$$

Circuit Representation in SPICE Format

Circuits are described in a text-based format, which specifies the components and their connections. For instance:

Figure 2: Voltage and Current Sources

A circuit with the following description:

```
plaintext
.circuit
Vs 1 GND dc 2
R1 1 2 1
R2 2 GND 1
.end
```

Describes:

- A voltage source (Vs) connected between nodes 1 and GND with a voltage of 2 V.
- A resistor (R1) between nodes 1 and 2 with resistance 1 Ω.
- A resistor (R2) between nodes 2 and GND with resistance 1 Ω.

Figure 1: Simple circuit

Another example circuit:

```
plaintext
.circuit
Vsource n1 GND dc 10
Isource n3 GND dc 1
R1 n1 n2 2
R2 n2 n3 5
R3 n2 GND 3
.end
```

This describes:

- A voltage source (Vsource) between nodes (n1) and GND with a voltage of 10 V.
- A current source (Isource) between nodes (n3) and GND with a current of 1 A.
- Resistors (R1), (R2), and (R3) with their respective values and connections.

Figure 1: Voltage and Current sources

This theoretical background provides the foundation for understanding how circuits are represented and analyzed using SPICE format.

My First Approach

The following describes my first method used to solve the SPICE assignment by performing nodal analysis on a circuit description file.

1. File Parsing and Component Identification

- **Function Implementation**: A function named `evalSpice` is implemented to read and process the circuit description from a file.

- **Circuit Definition Identification**: The function identifies the boundaries of the circuit definition using `.circuit` and `.end` markers.

- **Component Parsing**: Components in the circuit are parsed into lists based on their types:

        - Resistors
        - Voltage sources
        - Current sources

    Nodes in the circuit are identified and collected.

2. Node Mapping

- **Unique Identifiers**: Nodes are assigned unique numerical identifiers. Specifically:
    - The node labeled `GND` is mapped to zero.
    - Other nodes receive consecutive numbers starting from one.

3. Matrix Setup for Nodal Analysis

- **Matrix Construction**: A matrix `A` and a vector `b` are constructed to represent the system of equations for nodal analysis.

- **Matrix Population**:

    - **Resistors**: Contribute to the diagonal and off-diagonal elements of matrix `A`.
    - **Voltage Sources**: Introduce additional rows and columns to the matrix `A`, and add values of voltage sources to vector `b`.
    - **Current Sources**: Are incorporated into vector `b`.

4. Handling the Ground Node

- **Ground Node Adjustment**: The row and column corresponding to the `GND` node are removed from matrix `A` and vector `b` since the ground node's voltage is fixed at zero.

5. Solving the System of Equations

- **Equation Solving**: The system of linear equations is solved using `numpy.linalg.solve` to determine:
    - The voltages at the nodes
    - The currents through the voltage sources

6. Returning Results

- **Results**:
    - Voltages are returned for all nodes, with the `GND` node's voltage fixed at zero.
    - Currents through voltage sources are also returned.

```
# Version 1 Code
import numpy as np


def evalSpice(filename):
    try:
        with open(filename, "r") as file:
            lines = file.readlines()

        circuit_start = None
        circuit_end = None

        # Finding .circuit and .end
boundaries
        for i, line in enumerate(lines):
            stripped_line = line.strip()
            if stripped_line == ".circuit":
                circuit_start = i
            elif stripped_line == ".end":
```

```
                circuit_end = i

        if circuit_start is None or
circuit_end is None or circuit_start >=
circuit_end:
            raise ValueError("Invalid circuit
definition. Missing .circuit or .end")

        # Parse the circuit components
        component_list = lines[circuit_start
+ 1 : circuit_end]
        voltage_sources = []
        current_sources = []
        resistors = []
        nodes = set()

        for line in component_list:
            tokens = line.split()
            if len(tokens) < 4:
                raise ValueError("Invalid
component definition")

            component_name = tokens[0]
            node1 = tokens[1]
            node2 = tokens[2]
            nodes.update([node1, node2])

            if
component_name.startswith("R"):  # Resistor
                resistors.append(
                    {
                        "name":
component_name,
                        "node1": node1,
                        "node2": node2,
                        "value":
float(tokens[3]),
                    }
                )
            elif
```

```
component_name.startswith("V"):  # Voltage
Source
                voltage_sources.append(
                    {
                        "name":
component_name,
                        "node1": node1,
                        "node2": node2,
                        "value":
float(tokens[4]),
                    }
                )
            elif
component_name.startswith("I"):  # Current
Source
                current_sources.append(
                    {
                        "name":
component_name,
                        "node1": node1,
                        "node2": node2,
                        "value":
float(tokens[3]),
                    }
                )
            else:
                raise ValueError(f"Unknown
component: {component_name}")

        # Step 2: Node Mapping
        node_map = {}
        node_count = 0
        for node in nodes:
            if node == "GND":
                node_map["GND"] = 0
            else:
                node_count += 1
                node_map[node] = node_count

        num_nodes = node_count + 1  #
```

```
Including GND (0th node)
        num_voltage_sources =
len(voltage_sources)

        # Step 3: Set up matrices for nodal
analysis
        matrix_size = num_nodes +
num_voltage_sources
        A = np.zeros((matrix_size,
matrix_size))
        b = np.zeros(matrix_size)

        # Fill in resistors in the matrix
        for resistor in resistors:
            n1 = node_map[resistor["node1"]]
            n2 = node_map[resistor["node2"]]
            value = resistor["value"]
            if n1 != 0:
                A[n1 - 1, n1 - 1] += 1 /
value
            if n2 != 0:
                A[n2 - 1, n2 - 1] += 1 /
value
            if n1 != 0 and n2 != 0:
                A[n1 - 1, n2 - 1] -= 1 /
value
                A[n2 - 1, n1 - 1] -= 1 /
value

        # Fill in voltage sources in the
matrix
        for i, vsource in
enumerate(voltage_sources):
            n1 = node_map[vsource["node1"]]
            n2 = node_map[vsource["node2"]]
            row = num_nodes + i
            if n1 != 0:
                A[n1 - 1, row] = 1
                A[row, n1 - 1] = 1
            if n2 != 0:
```

```
                A[n2 - 1, row] = -1
                A[row, n2 - 1] = -1
            b[row] = vsource["value"]

        # Fill in current sources into b
vector
        for current_source in
current_sources:
            n1 =
node_map[current_source["node1"]]
            n2 =
node_map[current_source["node2"]]
            value = current_source["value"]
            if n1 != 0:
                b[n1 - 1] -= value
            if n2 != 0:
                b[n2 - 1] += value

        # Debugging: Print A and b before
removal of GND
        print("Matrix A before GND removal:")
        print(A)
        print("Vector b before GND removal:")
        print(b)

        # Step 4: Remove the row and column
corresponding to the GND node
        gnd_row_col = node_map["1"]

        # Remove GND row and column from A
        A = np.delete(A, gnd_row_col, axis=0)
# Remove GND row
        A = np.delete(A, gnd_row_col, axis=1)
# Remove GND column

        # Remove GND entry from b
        b = np.delete(b, gnd_row_col, axis=0)

        # Debugging: Print A and b after
removal of GND
```

```
        print("Matrix A after GND removal:")
        print(A)
        print("Vector b after GND removal:")
        print(b)

        # Step 5: Solve the system of
equations
        try:
            x = np.linalg.solve(A, b)
        except np.linalg.LinAlgError:
            raise ValueError(
                "Circuit is unsolvable,
possibly due to dependent loops or invalid
inputs"
            )

        # Step 6: Return the voltages and
currents
        voltages = {}
        for node, idx in node_map.items():
            if idx != 0:
                voltages[node] = x[idx - 1]

        voltages["GND"] = 0.0

        currents = {}
        for i, vsource in
enumerate(voltage_sources):
            currents[vsource["name"]] =
x[num_nodes + i - 1]

        return voltages, currents

    except FileNotFoundError:
        raise ValueError("File not found")


if __name__ == "__main__":
    filename = input("Enter the .ckt file
name: ")
```

```
    result = evalSpice(filename)
    print(result)
```

What Went Wrong

Issue Description

The primary issue arose from the random assignment of nodes to indices in the `node_map`. This randomization led to inconsistencies in the results. Specifically:

- **Inconsistent Outputs**: The same circuit input would yield different outputs depending on the random node assignments. This problem indicated that the approach was not correctly managing the mapping of nodes to indices.

Details of the Issue

- **Random Node Assignment**: Since node identifiers were assigned randomly, there was no consistent mapping between nodes and their corresponding indices in the matrix and vector. This inconsistency resulted in different voltage and current calculations for the same circuit configuration.

- **Output Discrepancies**: This randomness caused the results to be unreliable, with different outputs for the same input, demonstrating that the system was not correctly solving the circuit equations due to the lack of a stable node mapping.

Figure 3: Issue with first approach

Next Steps

To address this issue, a revised approach is needed where:

- Node assignments are consistent and deterministic.
- The mapping between nodes and matrix indices is fixed and reliable.

My Second Approach

On applying the steps mentioned above, the code worked and passed all test cases given in the test_evalSpice.py script.

```python
# evalSpice.py

"""
Roll No: EE23B110
Name: Ishaan Seth
Date: 08 Sept 2024
Version: 2
Description: To write a function (evalSpice)
that will read the given SPICE circuit, parse
it and solve the circuit by finding all node
voltages and branch currents of branches with
a voltage source, and raise specific errors
for invalid definitions.
Inputs: filename (Name of the file containing
the SPICE circuit)
Outputs: voltages (Dictionary containing all
node voltages), currents (Dictionary
containing branch currents through each
voltage source)
"""

import numpy as np
```

```
def evalSpice(filename):
    try:
        # Open the SPICE file and read the
circuit
        with open(filename, "r") as file:
            lines = file.readlines()

        circuit_start = None
        circuit_end = None

        # Finding .circuit and .end
boundaries
        start_flag = False
        end_flag = False
        for i, line in enumerate(lines):
            stripped_line = line.strip()
            if stripped_line == ".circuit":
                if start_flag == False:
                    circuit_start = i
                    start_flag = True
                elif start_flag == True:
                    raise ValueError("Invalid
circuit definition")
            elif stripped_line == ".end":
                if end_flag == False:
                    circuit_end = i
                    end_flag = True
                elif end_flag == True:
                    raise ValueError("Invalid
circuit definition")

        # Check if the file has valid
.circuit and .end markers
        if circuit_start is None or
circuit_end is None or circuit_start >=
circuit_end:
            raise ValueError("Malformed
circuit file")
```

```
        # Extract the component list between
.circuit and .end markers
        component_list = lines[circuit_start
+ 1 : circuit_end]
        voltage_sources = []
        current_sources = []
        resistors = []
        nodes = set()

        # Parse the components and categorize
them
        for line in component_list:
            tokens = line.split()
            if len(tokens) < 4:
                raise ValueError("Invalid
component definition")

            component_name = tokens[0]
            node1 = tokens[1]
            node2 = tokens[2]
            if node1 == node2:
                raise ValueError("Invalid
node definition")
            nodes.update([node1, node2])

            # Categorize components based on
their type
            if
component_name.startswith("R"):  # Resistor
                resistors.append(
                    {
                        "name":
component_name,
                        "node1": node1,
                        "node2": node2,
                        "value":
float(tokens[3]),
                    }
                )
```

```
            elif
component_name.startswith("V"):  # Voltage
Source
                if len(tokens) < 5:
                    raise ValueError("Invalid
component definition")

                voltage_sources.append(
                    {
                        "name":
component_name,
                        "node1": node1,  #
Positive node of voltage source
                        "node2": node2,
                        "value":
float(tokens[4]),
                    }
                )
            elif
component_name.startswith("I"):  # Current
Source
                if len(tokens) < 5:
                    raise ValueError("Invalid
component definition")

                current_sources.append(
                    {
                        "name":
component_name,
                        "node1": node1,
                        "node2": node2,  #
Current flows out through this node
                        "value":
float(tokens[4]),
                    }
                )
            else:
                raise ValueError("Only V, I,
R elements are permitted")
```

```
        # Node Mapping
        node_map = {}
        node_count = 0

        # Ensure GND node is present and
assign it index 0
        if "GND" in nodes:
            node_map["GND"] = 0
        else:
            raise ValueError(
                "GND node is missing from the
circuit. Please include a GND node."
            )

        # Remove GND from the node set and
map remaining nodes
        nodes_without_gnd = sorted(node for
node in nodes if node != "GND")

        for idx, node in
enumerate(nodes_without_gnd, start=1):
            node_map[node] = idx
            node_count += 1

        num_voltage_sources =
len(voltage_sources)
        matrix_num = (
            node_count + num_voltage_sources
        )  # Total number of rows/columns in
the matrix

        # Initialize matrices for nodal
analysis
        G = np.zeros(
            (matrix_num, matrix_num)
        )  # Conductance matrix, containing
conductances and coefficients for voltage
source equations
        b = np.zeros(
            (matrix_num, 1)
```

```
        )  # Column vector containing voltage
and current source values

        # Ensure current sources are not in
series with different values
        for i in range(len(current_sources)):
            for j in range(i + 1,
len(current_sources)):
                i1_node1 = current_sources[i]
["node1"]
                i1_node2 = current_sources[i]
["node2"]
                i2_node1 = current_sources[j]
["node1"]
                i2_node2 = current_sources[j]
["node2"]

                if (
                    (i1_node1 == i2_node1 and
i1_node2 == i2_node2)
                    or (i1_node1 == i2_node2
and i1_node2 == i2_node1)
                ) and (current_sources[i]
["value"] != current_sources[j]["value"]):
                    raise ValueError("Circuit
error: no solution")

        # Ensure voltage sources are not in
parallel in between the same pair of nodes
with different values
        for i in range(len(voltage_sources)):
            for j in range(i + 1,
len(voltage_sources)):
                v1_node1 = voltage_sources[i]
["node1"]
                v1_node2 = voltage_sources[i]
["node2"]
                v2_node1 = voltage_sources[j]
["node1"]
                v2_node2 = voltage_sources[j]
```

```
["node2"]

                if (
                    (v1_node1 == v2_node1 and
v1_node2 == v2_node2)
                    or (v1_node1 == v2_node2
and v1_node2 == v2_node1)
                ) and (voltage_sources[i]
["value"] != voltage_sources[j]["value"]):
                    raise ValueError("Circuit
error: no solution")

        # Fill in conductances in the matrix
        for resistor in resistors:
            n1 = node_map[resistor["node1"]]
            n2 = node_map[resistor["node2"]]
            value = resistor["value"]

            if n1 != 0:  # Only update if n1
is not GND
                G[n1 - 1, n1 - 1] += 1 /
value  # Self-conductance at n1

            if n2 != 0:  # Only update if n2
is not GND
                G[n2 - 1, n2 - 1] += 1 /
value  # Self-conductance at n2

            if n1 != 0 and n2 != 0:  # Mutual
conductance between n1 and n2
                G[n1 - 1, n2 - 1] -= 1 /
value
                G[n2 - 1, n1 - 1] -= 1 /
value

        # Fill in voltage sources in the
matrix
        for i, vsource in
enumerate(voltage_sources):
            n1 = node_map[vsource["node1"]]
```

```
            n2 = node_map[vsource["node2"]]
            row = node_count + i  # Voltage sources are added after all node equations

            if n1 != 0:  # n1 is not GND
                G[n1 - 1, row] = 1  # Voltage source adds 1 at n1
                G[row, n1 - 1] = 1  # Symmetric entry

            if n2 != 0:  # n2 is not GND
                G[n2 - 1, row] = -1  # Voltage source subtracts 1 at n2
                G[row, n2 - 1] = -1  # Symmetric entry

            b[row, 0] = vsource["value"]

        # Fill in current sources into b vector
        for current_source in current_sources:
            n1 = node_map[current_source["node1"]]
            n2 = node_map[current_source["node2"]]
            value = current_source["value"]

            if n1 != 0:
                b[n1 - 1, 0] -= value

            if n2 != 0:
                b[n2 - 1, 0] += value

        # Solve the system of equations
        try:
            x = np.linalg.solve(G, b)
        except np.linalg.LinAlgError:
            raise ValueError("Circuit error: no solution")
```

```
        # Return the voltages and currents
        voltages = {}
        for node, idx in node_map.items():
            if idx != 0:
                voltages[node] = x[idx - 1,
0].item()

        voltages["GND"] = 0.0  # Set the GND
node voltage

        currents = {}
        for i, vsource in
enumerate(voltage_sources):
            currents[vsource["name"]] =
x[node_count + i, 0].item()

        return voltages, currents

    except FileNotFoundError:
        raise FileNotFoundError("Please give
the name of a valid SPICE file as input")
```

Successful Resolution

Overview

After addressing the issue with random node assignments, the SPICE assignment solution has been successfully validated. The updated approach now consistently produces accurate results.

Validation

Screenshot below show the code passing all tests from the `test_evalSpice.py` script, demonstrating that the solution is now reliable and effective.

Figure 4: Test passed by the second approach

Test Cases

These are the given test cases that the script test_evalSpice.py checks for.

1. **Wrong Filename**
   - **Description**: Test with a file that has an incorrect extension or a non-existent file.
   - **Expected Outcome**: The program should handle the error gracefully, typically reporting that the file could not be found or opened.
2. **No .end or .circuit**
   - **Description**: Test with a file that lacks either the `.end` directive or the `.circuit` directive.
   - **Expected Outcome**: The program should detect the missing directive and report an appropriate error.
3. **Component Other Than V, I, and R Defined**
   - **Description**: Test with a circuit definition that includes components other than voltage sources (V), current sources (I), and resistors (R), such as capacitors (C) or inductors (L).
   - **Expected Outcome**: The program should report an unknown component error.
4. **Current Sources in Series with Different Current**

- **Description**: Test with current sources connected in series with different currents.
- **Expected Outcome**: The program should report that the circuit configuration is invalid or unsolvable.

5. **Voltage Sources in Parallel with Different Voltage**

   - **Description**: Test with voltage sources connected in parallel with different voltages.
   - **Expected Outcome**: The program should report that the circuit configuration is invalid or unsolvable.

6. **Circuit Not Solvable**

   - **Description**: Test with a circuit configuration that is inherently unsolvable or ambiguous.
   - **Expected Outcome**: The program should report that the circuit cannot be solved.

Extra Test Cases

These are some extra test cases that I check for in my code, to make it more secure from erroneous files.

1. **Two .circuits or .ends**

   - **Description**: Test with multiple `.circuit` or `.end` directives in the same file.
   - **Expected Outcome**: The program should report an error for having multiple `.circuit` or `.end` directives.

2. **GND Missing**

- **Description**: Test with a circuit definition that does not include a ground node.
- **Expected Outcome**: The program should report an error indicating the missing ground node.

3. **.end Before .circuit**

    - **Description**: Test with the `.end` directive appearing before the `.circuit` directive.
    - **Expected Outcome**: The program should report an error indicating that `.end` appears out of order.

4. **Invalid Component Definition (Contains Less Than Required Info)**

    - **Description**: Test with component definitions that are missing required information, such as nodes or values.
    - **Expected Outcome**: The program should report an error indicating that the component definitions are incomplete.

In [ ]:

```

```