SPICE Circuit Simulator Implementation Report

Arivoli Ramamoorthy

1 Introduction

This report details the implementation of a SPICE (Simulation Program with Integrated Circuit Emphasis) circuit simulator. The simulator is designed to analyze DC circuits containing resistors, voltage sources, and current sources using the Modified Nodal Analysis (MNA) method.

2 Implementation

2.1 Main Function: evalSpice

The core of the simulator is the `evalSpice` function:

```python
def evalSpice(filename: str) -> tuple[dict[str, float], dict[str,
    float]]:
    """
    Evaluate a SPICE circuit file and return node voltages and
    branch currents.

    Parameters:
    filename (str): The name of the SPICE circuit file to evaluate.

    Returns:
    tuple[dict[str, float], dict[str, float]]: A tuple containing
    two dictionaries:
        - The first dictionary maps node names to their voltages.
        - The second dictionary maps voltage source names to their
    currents.
    """
    # Function implementation...
```

This function takes a filename as input and returns a tuple of two dictionaries containing node voltages and branch currents.

2.2 Circuit Parsing

The circuit is parsed line by line using the `parse_line` function:

```python
def parse_line(line: str) -> None:
    """
    Parse a single line of the SPICE circuit file.

    Parameters:
    line (str): A line from the SPICE circuit file.

    Returns:
    None
    """
    # Function implementation...
```

This function processes each line of the circuit file, extracting component information and validating the input.

2.3 Matrix Setup and Solving

The simulator sets up the MNA matrix and solves it using NumPy:

```
# Initialize matrix and RHS vector
matrix = np.zeros((N + M, N + M))
b = np.zeros(N + M)

# Fill matrix for resistors, voltage sources, and current sources
# ...

# Solve the matrix
solution = np.linalg.solve(matrix, b)
```

2.4 Result Extraction

The solution is extracted into voltage and current dictionaries:

```
voltages = {"GND": 0.0}
for node, idx in nodes.items():
    if node != "GND":
        voltages[node] = float(solution[idx - 1])

currents = {
    comp[0]: float(solution[N + i]) for i, comp in enumerate(
    components["V"])
}
```

3 Error Handling

The simulator implements comprehensive error handling. Here are the key error checks:

3.1 File Not Found

```
try:
    with open(filename, "r") as file:
        lines = file.readlines()
except FileNotFoundError:
    raise FileNotFoundError("Please give the name of a valid SPICE
    file as input")
```

3.2 Removes Comments

```python
def parse_line(line: str) -> None:
    # Remove comments
    if "#" in line:
        line = line[: line.index("#")]

    line = line.strip()
    if not line or line == ".circuit" or line == ".end":
        return

    # Rest of the parsing logic...
```

3.3 Invalid Element

```python
if component_type not in "VIR":
    raise ValueError("Only V, I, R elements are permitted")
```

3.4 Malformed Circuit File

```python
if len(parts) < 4:
    raise ValueError("Malformed circuit file")

if not circuit_started or not circuit_ended:
    raise ValueError("Malformed circuit file")
```

3.5 Negative Resistance

```python
if R < 0:
    raise ValueError(f"Negative resistance value: {R}")
```

3.6 Invalid Component Specifications

```python
try:
    R = float(R)
    # ...
except ValueError:
    raise ValueError(f"Invalid resistance specification: {comp}")

# Same check for current source and voltage too
```

3.7 Disconnected Subcircuits

```python
if np.linalg.matrix_rank(matrix[:N, :N]) < N - 1:
    raise ValueError("Circuit contains disconnected subcircuits")
```

example:

```
.circuit
V1 1 0 DC 5
R1 1 2 1k
R2 2 0 2k

R3 3 4 1k  // This resistor is part of a disconnected subcircuit
R4 4 5 2k  //also part of the disconnected subcircuit
.end
```

3.8 Unsolvable Circuit

This will cover most of the error cases on its own

```python
try:
    solution = np.linalg.solve(matrix, b)
except np.linalg.LinAlgError:
    raise ValueError("Circuit error: no solution")
```

4 Additional special cases handled

Listing the additional errors handled that were not part of the given pytests below:

- Zero Resistance: Prints a warning and the resistor is treated as a wire
- Zero current source: Ignored and a warning is printed
- Validation of component specifications (e.g., ensuring resistance, voltage, and current values are valid floats)
- Disconnected subcircuits detection
- Checking for negative resistance values

5 Extra testcases added (to the extra folder)

- All Current circuit (raises error: no solution)
- Disconnected subcircuits (raises error: no solution)
- Invalid value of component (raises value error)
- Negative resistance (raises value error)
- Parallel Voltage (invalid = no solution)
- Zero current (ignored, output is printed)
- Zero resistance (resistor is treated as a wire, and output is printed)

6 Conclusion

This program helps solve DC circuits and is helpful for electrical engineering. It can handle different circuit parts like resistors, voltage sources, and current sources. It checks for mistakes in the circuit description and warns about special cases. The code is easy to read and can be improved later if we want to improve it.

7 References

1. to understand lambda function `https://www.geeksforgeeks.org/python-lambda/`
2. for basic understanding of numpy `https://numpy.org/doc/2.1/user/absolute_beginners.html`
3. googled around to learn more about pytest

8 Discussions

Discussed for a few minutes randomly in my hostel corridor about the edge cases my friends had implemented. I had already covered most of it. So did not bother changing the code.