

EE2703: Assignment-2

Automated SPICE Circuit Analysis Using Python

By Siva Sundar (EE23B151)

Date: 8/9/2024

Indian Institute of Technology Madras

Contents

1	Overview	1
1.1	Abstract	1
1.2	Introduction	1
1.3	Methodology	1
2	Circuit File (.ckt) Parsing	2
3	Matrix Construction	2
3.1	Using KCL and KVL to make the MNA matrices	2
3.2	Structure of M and N matrices	3
3.3	Example	3
3.4	Discussion on Structure	3
4	Solving the Matrix	4
5	Error Handling	4
5.1	File Not Found (<code>FileNotFoundError</code>)	4
5.2	Missing <code>.circuit</code> or <code>.end</code> Tags	5
5.3	Insufficient or Incorrect Data for Voltage/Current Sources and Resistors .	5
5.4	GND node not found	5
5.5	Invalid Component Type	5
5.6	Floating Nodes	6
5.7	Linear Algebra Errors (<code>np.linalg.LinAlgError</code>)	6
5.8	Summary of Error Handling	6
6	Assumptions	7
7	Results	7
8	Testing	7
9	Conclusion	8
10	References	8

1 Overview

1.1 Abstract

This experiment demonstrates the development of a Python-based program to perform SPICE circuit analysis. The implementation utilizes Modified Nodal Analysis (MNA) to solve electrical circuits, extract data from '.ckt' files which follow [SPICE standard format](#), and compute the voltages and currents in the circuit.

1.2 Introduction

Circuit analysis is a fundamental task in electrical engineering. Tools like SPICE are widely used for simulating and analyzing circuits. This experiment implements an automated system to handle circuits described in SPICE-like files using Python, matrix computation and [Modified Nodal Analysis](#) (MNA).

The goal is to create a program that extracts circuit data, builds matrix equations, and computes solutions for “node voltages” and “currents flowing via voltage sources”.

1.3 Methodology

The analysis is broken down into several stages:

- **Component Extraction:** Circuit details from the '.ckt' file are extracted and stored in a dictionary format. This task is done by **comp_extract** function. The function also checks whether the .ckt file follows SPICE standards, or else it **raises appropriate errors**.
- **Initialization:**
 - ★ Nodes and voltage sources are identified, and appropriate data structures are initialized using **initialize_VI** function.
 - ★ Matrices are set up for solving the system of equations using MNA. The size of the matrices is determined by the number of unknowns in the circuit. This is carried out by **matrix_initialize** function.
- **Splitting Components:** Voltage sources, current sources, and resistors are separated into distinct categories using **VI_split** function. This is done to make the task of building MNA matrices easier.
- **Building Matrices:** Based on *Kirchhoff's Current Law* (KCL) and *Kirchhoff's Voltage Law* (KVL), matrices are populated to reflect the relationships between components. This is done by **matrix_build** function.
- **Solving:** Using [linalg.solve](#) method in **numpy** library, the matrices are solved to find the unknown voltages and currents in the circuit.

evalSpice function takes in the .ckt file and by following the above steps, returns two dictionaries: one for node voltages (V) and another for currents flowing through voltage sources (I).

2 Circuit File (.ckt) Parsing

The circuit information is extracted from a ‘.ckt’ file. The file must follow SPICE conventions or else the function raises errors as and when needed.

```

1 def comp_extract(ckt_file):
2     # Extracting components from the .ckt file
3     try:
4         with open(ckt_file, "r") as file:
5             lines = file.readlines()
6             # Code continues to process the lines between .circuit and
7             .end
8     except FileNotFoundError:
9         raise FileNotFoundError("Please give a valid SPICE file")

```

Code Snippet 1: Circuit File Extraction

comp_extract function ‘tries’ to take in the ‘.ckt’ file and stores each line between .circuit and .end in a list defined as ‘lines’. If it fails to open the file, it raises “FileNotFoundError”. (See Error handling section for more details)

```

1 # Inside comp_extract()
2 # Simplified code (without error handling)
3 for line in lines: # Iterating through the lines
4     words = line.split() # Splitting the line into words
5
6     if words[0][0] == "V" or words[0][0] == "I":
7         comp[words[0]] = [words[1], words[2], words[3], words[4]]
8
9     elif words[0][0] == "R":
10        comp[words[0]] = [words[1], words[2], words[3]]

```

Code Snippet 2: Making the Component dictionary

The circuit components are then extracted into a dictionary with keys representing component names and values containing details of the connections (nodes and values) as seen from the code snippet above. The components handled include resistors (‘R’), *independent* voltage (‘V’) and current sources (‘I’).

3 Matrix Construction

The MNA technique is employed to build the matrix equation $M \cdot X = N$, where M represents the system matrix, X contains unknown voltages and currents, and N holds the input sources.

matrix_build function is used for building the MNA matrices M and N using the list of voltage sources, current sources, resistors (three of these are obtained from component dictionary using **VI_split** function) and dictionary with node names (which is ‘V’).

3.1 Using KCL and KVL to make the MNA matrices

By KCL, “Sum of currents leaving a node is zero”. For making the matrices, we use the statement as: “Sum of currents leaving a node via resistor and voltage branches *equals* Sum of currents entering the node through current source branches entering the node”

Mathematically, for a node n_1 :

$$\frac{V_{n_1} - V_{n_2}}{R_{n_1-n_2}} + \dots + \frac{V_{n_1} - V_{n_k}}{R_{n_1-n_k}} + I_{V_{s_1}} + I_{V_{s_2}} + \dots + I_{V_{s_m}} = I_{s_1} + I_{s_2} + \dots + I_{s_l}$$

where,

- $n_1, n_2, ..n_k$ are 'k' nodes whose voltage is unknown.
- V_{n_i} represents node Voltage of node n_i .
- $I_{V_{s_i}}$ is the current flowing out of the node V_{n_1} via the i 'th voltage source V_{s_i} .
- I_{s_i} is the current flowing into the node V_{n_1} via the i 'th current source I_{s_i} .

Similar rearrangement in KVL statement is done, for making the matrices: “*Voltage difference across the nodes of voltage source equals Voltage difference produced by the voltage source.*”

Mathematically, for a voltage source V_{s_1} whose positive terminal is node n_i and negative at node n_j :

$$V_{n_i} - V_{n_j} = V_{s_1}$$

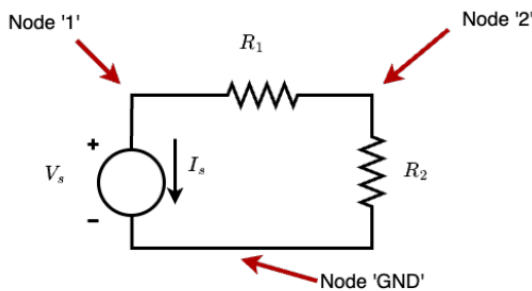
3.2 Structure of M and N matrices

Consider a network with resistors, voltage sources and current sources connected in some arbitrary way. Say it has 'k' nodes and 'm' voltage sources. The matrices have a particular structure. For the matrix M :

- First k rows of the matrix represent the LHS of KCL equations for the k nodes (neglect “GND”).
- Row ' i ' ($\leq k$) corresponds to the KCL equation for node n_i . Row ' i ' ($> k$) corresponds to the KVL equations for $(i - k)$ 'th voltage source. (For $i = k + 1$, the row is KVL for voltage source V_{s_1})
- Columns from 1 to k represent nodes n_1 to n_k , and columns from $k + 1$ to $k + m$ represent the currents in the m voltage sources $I_{V_{s_1}}$ to $I_{V_{s_m}}$ respectively.

The corresponding RHS for the equations represented by each row of matrix M is the row element of matrix N .

3.3 Example



$$\begin{bmatrix} \frac{1}{R_1} & \frac{-1}{R_1} & 1 \\ \frac{-1}{R_1} & \frac{1}{R_1} + \frac{1}{R_2} & 0 \\ 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} V_1 \\ V_2 \\ I_s \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ V_s \end{bmatrix}$$

MNA matrix equation

Circuit shown in the *INSTRUCTIONS.pdf*

3.4 Discussion on Structure

I tried to make the code without considering this structure, after many tries, I realised making the matrix without a systematic way would make the code unreadable and also makes it too difficult. The idea to structure the matrix was from a friend of mine.

More detailed explanation on the **matrix_build** function is given as its '*doc string*' in the python code.

4 Solving the Matrix

Once the matrices are built, the linear system is solved using the `linalg.solve` method in `numpy` library as shown in the below code snippet:

```

1  # Inside evalSpice()
2  try:
3      X = np.linalg.solve(M, N)
4  except np.linalg.LinAlgError:
5      raise ValueError("Circuit error: no solution")

```

Code Snippet 3: Solving the MNA equation

The `try-except` block is used to check whether there are any problematic loops in the circuit (parallel voltage sources or serial current sources of different values).

The vector `X` has its first k elements as node voltages and remaining elements as currents. This can be split into 'V' and 'I' dictionaries as shown in the code snippet below:

```

1  # Inside evalSpice()
2  # Writing the solution from X to V and I dicts
3  index = 0 # For traversing the X vector
4  del V["GND"] # Removing it for now
5  for key in V:
6      V[key] = X[index]
7      index += 1
8  V["GND"] = 0 # Adding "GND"
9  for key in I:
10     I[key] = X[index]
11     index += 1

```

Code Snippet 4: Splitting the 'X' vector to 'V' and 'I' dictionaries

The dictionaries 'V' and 'I' now contain the voltage at each node and the currents through the voltage sources respectively.

5 Error Handling

Error handling enhances program robustness by managing unexpected inputs, ensuring data integrity, and preventing crashes. It improves user experience with clear error messages, aids debugging, and helps maintain reliable outputs.

5.1 File Not Found (FileNotFoundError)

```

1  try:
2      with open(ckt_file, 'r') as file:
3          lines = file.readlines()
4  except FileNotFoundError:
5      raise FileNotFoundError("Please give the name of a valid SPICE file as input")

```

Purpose: This block handles the scenario where the specified circuit file does not exist or cannot be opened.

Explanation: If the file does not exist or can't be opened, the program raises the error `FileNotFoundError` with the message: *"Please give the name of a valid SPICE file as input."*

5.2 Missing .circuit or .end Tags

```

1 if start == -1 or stop == -1 or stop <= start:
2     raise ValueError(f"{ckt_file} should contain a .circuit and .end
   with .end after .circuit")

```

Purpose: This block checks if the SPICE file is properly formatted and contains both .circuit and .end tags.

Explanation: If the .circuit and .end tags are missing, or .end is placed before .circuit, a ValueError is raised with the message: “{ckt_file} should contain a .circuit and .end with .end after .circuit.”

5.3 Insufficient or Incorrect Data for Voltage/Current Sources and Resistors

```

1 if len(words) < 5: # "len(words) < 4" for resistors
2     raise ValueError(f" '{words[0]}' should contain 4 data")
3 if not is_float(words[4]): # "not is_float(words[3])" for resistors
4     raise ValueError(f"{words[0]} contains non-numeric source value")
5 if words[0] in comp:
6     raise ValueError(f"{words[0]} defined multiple times in the circuit
   ")

```

Purpose: These statements handle errors when processing resistors (R) or voltage (V) or current (I) sources in the circuit description.

Explanation:

- If fewer than 5 pieces of data (4 for resistors) are provided for the source, the program raises a ValueError with the message: “ '{words[0]}' should contain 4 data.”
- If the source value is non-numeric, a ValueError is raised with the message: “{words[0]} contains non-numeric source value.”
- If the same component is defined more than once, a ValueError is raised with the message: “{words[0]} defined multiple times in the circuit.”

5.4 GND node not found

```

1 if "GND" not in V_temp:
2     raise ValueError("Circuit must have a GND node")

```

Purpose: This block checks whether there is GND node assigned to any component because only then can the solutions be found.

Explanation: The code checks if GND is present in the node list (in **initialize_VI**). If GND is missing, a ValueError is raised, indicating that the circuit description is incomplete.

5.5 Invalid Component Type

```

1 # If V or I, elif R,...else:
2 else:
3     raise ValueError(f" '{words[0][0]}' is not a valid element")

```

Purpose: This line ensures that only valid circuit elements (like resistors, voltage, and current sources) are processed.

Explanation: If an unsupported component type is encountered, a ValueError is raised with the message: “ '{words[0][0]}' is not a valid element.”

5.6 Floating Nodes

```
1 if is_floating_node(comp):
2     raise ValueError(f"There are floating nodes in the circuit
   described in '{ckt_file}' file")
```

Purpose: This block checks for the presence of floating nodes (nodes that are not connected to any other node in the circuit).

Explanation: If floating nodes are detected, a `ValueError` is raised with the message: *"There are floating nodes in the circuit described in '{ckt_file}' file."*

5.7 Linear Algebra Errors (`np.linalg.LinAlgError`)

```
1 try:
2     X = np.linalg.solve(M,N)
3 except np.linalg.LinAlgError:
4     raise ValueError("Parallel voltage sources or current sources in
   series.")
```

Purpose: This block handles errors related to solving the matrix equations, which is part of the Modified Nodal Analysis (MNA) process.

Explanation: If a singular matrix or linear dependency is detected while solving the system (e.g., parallel voltage sources or current sources in series), a `LinAlgError` is raised by NumPy. This error is caught, and a `ValueError` is raised with the message: *"Parallel voltage sources or current sources in series."*

5.8 Summary of Error Handling

- **File Handling:** The program ensures that the input file exists and it follows SPICE standard format.
- **Component Validation:** The program checks each component's data (resistors, voltage sources, current sources) for proper formatting, valid numerical values, and duplicate definitions.
- **Ground Node Requirement:** The program verifies that the circuit description includes a GND node, which serves as the reference point for all voltages.
- **Invalid Circuit Structures:** Floating nodes and parallel/series errors in circuit configuration are also handled.
- **Linear Algebra Errors:** Errors during matrix solving (due to invalid configurations like parallel voltage sources) are captured and explained to the user.

6 Assumptions

There are some assumptions that I made which constraints the node names for resistors in the '.ckt' file:

- It can be a string of the type: “string without numbers” + “number” + “string without numbers”.
- Directly taking big numbers without starting from the number 1 is not supported.

eg: A file can contain nodes as:

- 1, 2, 3.... or
- abc1def, abc2def, abc3def....
- names like A, B, C.... are **not supported**.
- 13, 14, 15... without using 1, 2... is **not supported**.

The last two cases are valid as per SPICE format. Hence, the code fails to solve such circuits directly. (If I could change the node names which satisfy my code's assumption, then its solvable.)

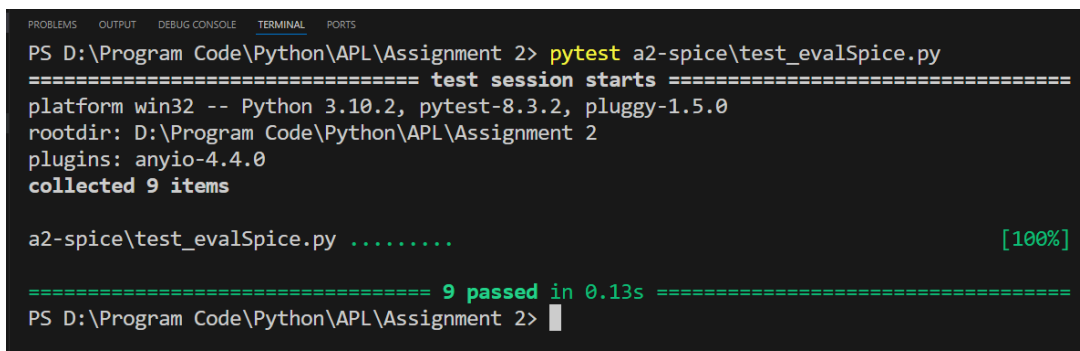
7 Results

The `evalSpice()` returns two dictionaries: one for the node voltages (V) and another for the currents through voltage sources (I).

- **Voltages:** The potential difference between nodes, including the ground node (0V).
- **Currents:** The current values for each voltage source.

8 Testing

Used the `test_evalSpice.py` python file with `pytest` to test the python code using the test cases given in `testdata` folder.



```
PS D:\Program Code\Python\APL\Assignment 2> pytest a2-spice\test_evalSpice.py
===== test session starts =====
platform win32 -- Python 3.10.2, pytest-8.3.2, pluggy-1.5.0
rootdir: D:\Program Code\Python\APL\Assignment 2
plugins: anyio-4.4.0
collected 9 items

a2-spice\test_evalSpice.py ..... [100%]

===== 9 passed in 0.13s =====
PS D:\Program Code\Python\APL\Assignment 2> |
```

All test cases passed

9 Conclusion

This experiment demonstrates how a Python program can automate the analysis of electrical circuits using the Modified Nodal Analysis (MNA) method. By parsing SPICE-like '.ckt' files, building matrix equations based on circuit components, and solving the system using numerical methods, the program successfully computes node voltages and current values, providing an efficient tool for circuit simulation.

10 References

- [1] Modified Nodal Analysis
- [2] SPICE: A brief overview