Spice Simulation - EE23B060

September 8, 2024

1 Overview:

This assignment implements a simplified SPICE circuit simulator in Python for purely resistive circuits without dependent sources. The program parses a circuit description from a text-based file and builds a matrix representation of the circuit using Kirchhoff's Current and Voltage Laws. This matrix is then solved to determine node voltages and branch currents. The code also includes error checking for malformed files, such as missing .circuit or .end statements, duplicate component names, and invalid element types. The program outputs nodal voltages and current through the voltage sources.

2 Problem Approach:

The code is majorly divided into three parts - Parsing through the file, forming the matrix and solving it.

3 Parsing through the circuit file:

The `parsing()` function reads a SPICE circuit file, accesses the circuit components (resistors, voltage sources, and current sources), and maps the nodes in the circuit to numbers starting from 0(for GND) using another function called `mapping_nodes()`. It processes the file line-by-line, checks for malformed file case or invalid elements, and stores the parsed information in different list of dictionaries assigned for each element type.

3.1 Steps in the function:

1. **File Reading:** The function accepts the name of the file containing the circuit as input and opens it in read mode. The try block of the function tries to open the file, and reads it line by line into a list named `lines`. If the file does not exist or the file path is incorrect,the except block catches this and raises a `FileNotFoundError`.
2. **Initialising:**
   - Dictionary `node_mapping` is initialized with `'GND'` (the ground node) mapped to 0.
   - Three lists are initialized to store resistors (`Resistors`), voltage sources (`Voltage_sources`), and current sources (`Current_sources`).
   - A `node_counter` is initialized to 1 to track and assign unique numbers to each node in the circuit using the `mapping_nodes()` function.
   - A flag `Junk` is used to determine which part of the text file is to be considered as the circuit.

- Three sets: `resistor_names`, `voltage_source_names` and `current_source_names` keep track of the names of the components, and checks if a component is repeated.

3. **Checking if the file is malformed:**
   - **Check 1: Presence of `.circuit`:** `.circuit` marks the beginning of the circuit. The program scans for this to determine when the important part of the circuit file starts. If the keyword is not found, it raises a `ValueError` indicating a malformed circuit file. The first `.circuit` encountered is the valid beginning when there are multiple `.circuit`
   - **Check 2: Presence of `.end`:** `.end` marks the end of the circuit definition. Similiar to the previous check, if `.end` is missing, a `ValueError` is raised and indicated the same error. The first `.end` encountered is the valid in the case of multiple `.end`.
4. **Parsing Circuit Elements:** The lines between the `.circuit` and `.end` are considered valid circuit information:
   - Each line is split into individual words and the first character of the first word in each line is scanned to identify the type of circuit element. The letters scanned are coverted to uppercase and are then compared to 'V', 'I','R'
     - `R`: Resistor
     - `V`: Voltage source
     - `I`: Current source
   - The three sets `resistor_names`, `voltage_source_names` and `current_source_names` keep track of the components, if a component is already present in these sets, then the code raises a `ValueError` for duplicate name entries.
   - In case of resistors, the line must have at least 4 words, and for voltage and current sources, the line must have at least 5 words.
   - The subsequent words in the line specify the nodes between which they are connected (`n1`, `n2`) and the values, in case of the sources, they conatin an additional information of ac/dc. The node names are mapped to integers using the `mapping_nodes()`.
   - The function stores the name, node connections, and value of the element in a dictionary and appends it to the appropriate element list.
5. **`mapping_nodes()` function** This function assigns a unique integer index to each node in the circuit. It checks if the node has been encountered before, if it hasn't, it adds the node to the dictionary `node_mapping` with a new index and increments `node_counter`. If the node already exists in the dictionary, it returns its previously assigned index.
6. **Checking for invalid elements** The function raises a `ValueError` if an invalid element type is encountered, invalid elements being any element other than 'V', 'I', or 'R'.
7. **Returns:**
   - `Resistors`: A list of dictionaries representing the resistors in the circuit.
   - `Voltage_sources`: A list of dictionaries representing the voltage sources.
   - `Current_sources`: A list of dictionaries representing the current sources.
   - `node_mapping`: A dictionary mapping node names to numbers, including `'GND'`.
   - `node_counter`: The total number of nodes in the given circuit.

4 Matrix Formation

The `matrix_formation` function is responsible for generating two matrices - `Matrix` and `Matrix_right`- of the circuit using Kirchhoff's Voltage and Current Laws. The matrix is used to obtain the nodal voltages and branch currents for resistors, voltage sources, and current sources in the circuit. The matrices are - `Matrix`: Stores the coefficients of the unknowns of the nodal equations and voltage source equations(which explain the contribution of current through it). - `Matrix_right`: Stores the constants of the equations in their corresponding rows.

4.1 Input Parameters:

1. **Resistors, Voltage_sources, Current_sources**: A list of dictionaries, where each dictionary contains:
   - `'name'`: The name of the element.
   - `'n1'`: The first node the element is connected to.
   - `'n2'`: The second node the element is connected to.
   - `'value'`: The value of the element
2. **node_mapping**: A dictionary that maps each node name to a unique number using the `mapping_nodes()` function, these are then accessed and used to assign the indices of the matrices.
3. **node_count**: The total number of nodes in the circuit including ground (GND:0).

4.2 Steps in the Function:

1. **Initialization:**
   - The function calculates the number of non-ground nodes in the circuit and assigns the value to `total_nodes`
   - It also counts the number of voltage sources and assigns it to `total_voltage_nodes`.
   - The matrix `'value'` is initialized as a zero matrix of size `(total_nodes + total_voltage_sources, total_nodes + total_voltage_sources)`. This matrix will represent the nodal equations, by storing the coefficients of the unknowns of the said equations
   - The matrix `Matrix_right` is initialized as a zero column vector of size `(total_nodes + total_voltage_sources, 1)`, which will store the source contributions, that is the constant values of the nodal equations
2. **Handling Resistors in the matrix:**
   - For each resistance/impedance, the admittance `Y` is calculated initially. This is done by type casting a string into float, if this is not possible due to invalid type of Value, the program raises a `TypeError`.
   - The matrix entries for nodes `n1` and `n2` are updated according to the following logic:
     - Diagonal entries (`Matrix[n1-1][n1-1]` and `Matrix[n2-1][n2-1]`) are incremented by `Y`.
     - Off-diagonal entries (`Matrix[n1-1][n2-1]` and `Matrix[n2-1][n1-1]`) are decremented by `Y`, representing the interaction between nodes `n1` and `n2`.
   - If the resistor connects a node to itself, only the diagonal entry is updated, as this short circuits the mentioned resistor.
3. **Handling Current Sources in the matrix:**

- For each current source, the current value `I` is retrieved from the dictionary and is type casted into a float, if this is not possible due to invalid type of Value, the program raises a `TypeError`.
- The contribution of the current source is added to `Matrix_right` according to the logic:
  - `Matrix_right[n1-1][0]` is decreased by `I` (current flows out of node `n1`).
  - `Matrix_right[n2-1][0]` is increased by `I` (current flows into node `n2`).

4. **Handling Voltage Sources in the matrix:**
   - For each voltage source, additional rows and columns are added to `Matrix` to impose the constraint of unknown current through the voltage source in the nodal equations.
     - A `1` is added to the entry corresponding to `n1`, and `-1` is added for `n2`.
     - The corresponding row of `Matrix_right` is set to the voltage value `V`, the value of the voltage source considered..
5. **Returns:**
   - The function returns the contsructed`Matrix` and `Matrix_right` which form the system of linear equations that can be used to find the nodal voltages and branch currents.

4.3 Important Considerations

- **Handling of GND**: The ground node is always treated as node 0, and it is not included in the equations unless it has an effect due to a current or voltage source.
- **Voltage Source Handling and Matrix size**: The size of the matrix grows as the number of nodes and voltage sources increases, with each voltage source adding one extra row and column. These extra set of rows or columns take care of the branch currents passing through the voltage sources.
- **Current Source Handling** Each current source directly affects the `Matrix_right` contributing to the current entering and leaving the pair of nodes of the source. This is where we introduce auxillary equations.

5 `evalSpice()` function : Solving the system of equations and storing them

The `evalSpice()` function takes the name of the file containing the circuit information, parses the components, forms the necessary matrices by calling the previously defined fucntions `parsing()` , `matrix_formation` and `mapping_nodes`. The function then solves the system of linear equations to obtain the nodal voltages and currents through voltage sources. It also handles any errors that may arise during the solving process.

5.1 Function Parameters:

- **filename**: The name of the input file that contains the circuit information.

5.2 Steps in the Function:

1. **Parsing the Circuit File**: The function first calls the `parsing` function to get information about resistors, voltage sources, current sources, and node mapping from the input file.
2. **Matrix Formation**: The function calls `matrix_formation`, passing the parsed data, to construct the matrices necessary to solve the circuit:

3. **Solving the Matrix Equation**: Using NumPy's `linalg.solve()` function, the system of equations is solved for nodal voltages and voltage source currents. If the equations cannot be solved due to singularity or other issues, a `ValueError` is raised with the message `'Circuit error: no solution'`.
4. **Storing Node Voltages**: After obtaining the solution, the nodal voltages are stored in a dictionary called `voltages`. The Ground node (`GND`) is set to 0V, and the other node voltages are extracted from the `solution` array
5. **Storing Source Currents**: The function then computes the currents through the voltage sources, which are stored in the `currents` dictionary. These currents are extracted from the additional rows in the solution corresponding to the voltage sources.
6. **Returning Results**: The function returns two dictionaries:
   - **voltages**: Contains the computed voltage for each node.
   - **currents**: Contains the computed current for each voltage source.

6 Special Cases and Error Handling

- **File Handling Errors:**
  - **FileNotFoundError:** This error is raised if the specified SPICE file cannot be found. It asks the user to provide a valid file name.
- **Malformed Circuit File Errors:**
  - **ValueError: Missing `.circuit` or `.end` Line:** This error is raised if the SPICE file does not contain a `.circuit` line to start the circuit description or an `.end` line to end it. According to my code, the valid part of the circuit file is in between the the first `.circuit` and the first `.end` encountered, this is in the presence of multiple `.circuit` and `.end` in the circuit file.
- **Data Validity Errors:**
  - **Duplicate Component Names:**
    - **Resistor:** If a resistor with the same name is encountered more than once, a `ValueError` is raised to prevent duplicate entries.
    - **Voltage Source:** Similarly, if a voltage source with the same name is duplicated, an error is raised.
    - **Current Source:** The same check applies to current sources.
  - **Insufficient Data:**
    - **Resistor:** If the line describing a resistor has fewer than four components (name, node 1, node 2 and value), a `ValueError` is raised indicating insufficient data to access the resistor's value.
    - **Voltage Source:** If a voltage source line contains fewer than five components (name, node 1, node 2 and value), an error is raised for insufficient data.
    - **Current Source:** Similar to voltage sources, a `ValueError` is raised if a current source line has fewer than five components.
  - **Unnecessary Data**
    - If the lines describing the component have more words than necessary, my code just takes takes the first 4 words into consideration for resistors, and the first 5 words for voltage and current sources. It ignores the rest of the words in the line.
  - **Type of value for different components**

    - For each component, while extracting a Value, its datatype is string, while using the same Value in the matrix, it is typecasted into Floats. If this is not possible due to invalid datatype of Value, `TypeError` is raised
- **Component Type Validation:**
  - **Invalid Component Type:** If a line starts with a component type other than `R` for resistors, `V` for voltage sources, or `I` for current sources, a `ValueError` is raised to ensure only valid component types are processed.
- **Matrix Solution Errors:**
  - **ValueError: Singular Matrix:** If the matrix solver (`np.linalg.solve`) encounters a singular matrix, a `ValueError` is raised. This indicates that there is no solution to the system of equations
- **Case Sensitivity:**
  - A resistor could have it's name starting with either 'r' or 'R', while a Voltage Source with either 'V' or 'v' and a Current Source could start with 'I' or 'i'. This is done by checking the first letter, converting it to upper case and checking it with 'V', 'I' or 'R' appropriately.
- **Physical Redundant cases**
  - **Two current sources with the same value in series:** For this case, even though in the physical sense, this could be replaced with just one source, the inability of finding the nodal voltage between the two sources using Kirchhoff's laws results in a singular matrix.
  - **Two voltage sources with the same value in parallel:** For this case, in the physical sense, this can be replaced with just one voltage source, the inability of finding the branch currents through each of them using Kirchhoff's laws results in a singular matrix.
- **Resistance=0** -When the value of resistor is zero, calculating admittance raises `ZeroDivisionError`

7 Reference:

1. https://cheever.domains.swarthmore.edu/Ref/mna/MNA3.html : Used this to verify if my matrix formation logic made sense.
2. https://numpy.org/doc/stable/reference/generated/numpy.linalg.solve.html : Used this to understand what is returned from the function.
3. https://numpy.org/doc/stable/reference/generated/numpy.zeros.html : referred to this to initialise zero matrices using numpy.
4. My classmate (Navin Kumar, EE23B047) helped me set up pytest locally in my laptop. We also discussed cases to be considered for malformed files. The following are:
   - Multiple '.circuit' and '.end' blocks, my code considers the first '.circuit' and the first '.end' that it encounters as the valid start and end, respectively. This is regardless of the number of '.circuit' being equal to or not equal to the number of '.end'. Missing of either of the two is considered to be a malformed file.
   - If the circuit element values cannot be typecasted into float(not valid value datatype), this is considered as a malformed file in my code.
5. My other classmate (Aprajithan S, EE23B007) and I had discussions regarding two special cases, which would result in singular matrix using the code, but is solvable in reality:
   - Two current sources of same value being in series in a network.
   - Two voltage sources of same value veing in Parallel in a network.