University of Bath

Design Document

EE20084 Structured Programming

George Madeley
3-12-2021

INTRODUCTION

The task is to design a circuit analysis software where connected series and shunt resistors of any value can be in any order between source and load. The program must accept an input file, in XML format, containing three sections describing the circuit, the terms and the requested outputs. The program must identify the given resistance and conductance values between nodes given in the input file's circuit description section. The program must also identify the input parameters, Thevenin's voltage and resistance or Norton's current and resistance. From this information, the code should be able to calculate the requested information, stipulated in the 'output' section on the input file, using the ABCD matrix analysis scheme. A formatted table should contain the calculated data. The top row should contain the names of each piece of information, then their units on the second row and the data collected after. Figure 3 shows an example of this table.

In addition to this, there are numerous extension tasks that the code may process. The input parameters may include exponent prefixes that the program should use to alter that component value. The output section may also include exponent prefixes. The program should convert the requested information into the requested exponent. The output value may also be requested to be in decibels. From this information and the unit, the program should convert the output value into decibels. The output table should change to accompany these unit alterations. If specified by the input file, the table should layout frequencies in a logarithmic format.

METHOD

By analysing the problem, the program can is into three distinct sections: read the file, perform calculations, format data. Each section will utilise data collected from the read file section of the code. Due to the amount of data and the different types, a custom class will store circuit subsections information.

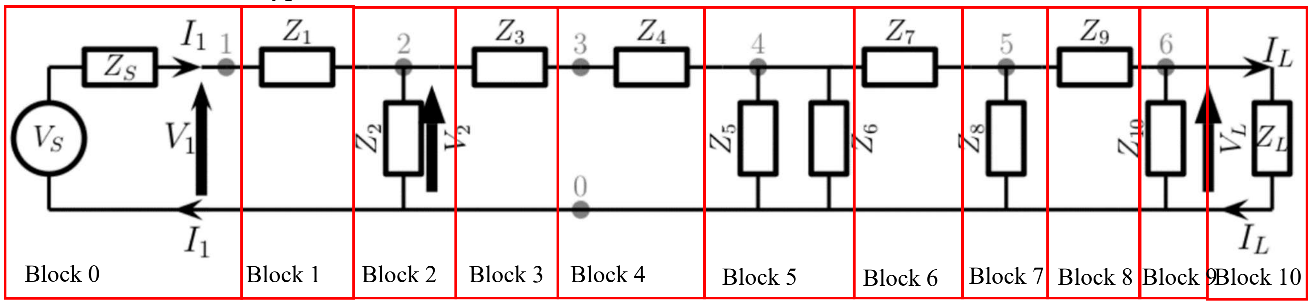

Figure 1 Example Circuit with circuit blocks

i. Class

The class, named 'Block', will represent a block of the circuit between two nodes. For instance, in the example circuit seen in Figure 1, an instance of the class can represent the resistor Z3 situated between nodes two and three (block 3). These classes can then calculate input and output voltage, current, power, impedance, and gain at that part of the circuit and the whole circuit.

Attributes

The object will contain attributes relating to the voltage and current in and out, along with the ABCD matrix.

Name	Type	Description
inputVoltage	Float List	This is a list the input voltage to a block in the format of a float list of length bigger than zero. If the list has one element, the voltage is DC; else, it is AC.
inputCurrent	Float List	This is a list the input current to a block in the format of a float list of length larger than zero. If the list has one element, the current is DC; else, it is AC.
outputVoltage	Float List	This is a list the calculated output voltage from the block.
outputCurrent	Float List	This is a list the calculated output current from the block.
matrixABCD	Float Matrix	This is a matrix containing four elements corresponding to A, B, C, and D in the matrix in that order. This matric represents either series impedance (equation four from laboratory script) or shunt admittance (equation five from laboratory script).

Methods

The class methods will calculate the voltage, current, impedance, gain, and power in and out of the resistor.

$$\begin{bmatrix} A & B \\ C & D \end{bmatrix} = \begin{bmatrix} 1 & Z \\ 0 & 1 \end{bmatrix} \qquad \begin{bmatrix} A & B \\ C & D \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ Y_{sh} & 1 \end{bmatrix}$$

Equation 2 How the matrix will store resistance and conductance

- **Calculate Output Voltage –** The method calculates the output voltage from the given "inputVoltage", "inputCurrent", and "matrixABCD". This method will utilise the inverted voltage equation (Equation 3) from chain matrix analysis, which returns a list of the output voltages from that circuit block. The method raises an exception if the values of A*D and B*C are the same as nothing can be divided by zero. Three tests will be conducted. The first will provide the method with a series of different predetermined values of the class attributes and see if the methods calculate the correct output voltage. The second test will provide the method with only some of the class attributes and see if the method can still calculate the correct output voltage or raise an exception if there is not enough data. The third will provide the method with data that is the incorrect data type that the method accepts. For example, passing in a string when the method only accepts integers, floats, and doubles. The method is accepted to raise the invalid data type exception.

$$\begin{bmatrix} V_1 \\ I_1 \end{bmatrix} = \begin{bmatrix} A\,B \\ C\,D \end{bmatrix} \begin{bmatrix} V_2 \\ I_2 \end{bmatrix} = [T] \begin{bmatrix} V_2 \\ I_2 \end{bmatrix}$$

Equation 1 Input current and voltage equation

- **Calculate Output Current –** This method calculates the output current from the given data in "inputVoltage", "inputCurrent", and "matrixABCD". The method will utilise the inverted voltage equation (Equation 3)from chain matrix analysis. This calculation returns a float list of the output voltages from that circuit block. The method raises an exception if the values of A*D and B*C are the same as nothing can be divided by zero. Three tests will be conducted. The first will provide the method with a series of different predetermined values of the class attributes and see if they calculate the correct output current. The second test will provide the method with only some of the class attributes and see if the method can still calculate the correct output current or raise an exception if there is not enough data. The third will provide the method with data that is the incorrect data type that the method accepts. For example, passing in a string when the method only accepts integers, floats, and doubles. The method is accepted to raise the invalid data type exception.

$$\begin{bmatrix} V_2 \\ I_2 \end{bmatrix} = [T]^{-1} \begin{bmatrix} V_1 \\ I_1 \end{bmatrix} = \frac{1}{AD - BC} \begin{bmatrix} D & -B \\ -C & A \end{bmatrix} \begin{bmatrix} V_1 \\ I_1 \end{bmatrix}$$

Equation 3 Output current and voltage equation

- **Calculate Power –** This calculates the input and output power from the data in "inputVoltage", "inputCurrent", "outputVoltage", and "outputCurrent". The method will utilise the chain matrix analysis. If an attribute is missing, the code will raise an exception or call the "CalculateOutputVoltage/Current' method to get the required data. This calculation returns a list of the circuit block's input power and a list of the block's output power. Three tests will be conducted. The first will provide the method with a series of different predetermined values of the class attributes and see if the methods calculate the correct input and output power. The second test will provide the method with only some of the class attributes and see if the method can still calculate the correct input and output power or raise an exception if there is not enough data. The third will provide the method with data that is the incorrect data type that the method accepts. For example, passing in a string when the method only accepts integers, floats, and doubles. The method is accepted to raise the invalid data type exception.
- **Calculate impedance –** calculates the input and output impedances from the data given in class attributes. This method will utilise the chain matrix analysis impedance equation (Equation 4). If an attribute is missing, the code will raise an exception or call the "CalculateOutputVoltage/Current' method. This calculation returns two float lists; one contains the input impedances of that block, and the other contains the output impedances of that block. If the value for input current or any denominator in Equation 4 are 0, the method raises an exception. Three tests will be conducted. This first will provide the method with a series of different predetermined values of the class attributes and see if the methods calculate the correct input and output impedance. The second test will provide the method with only

some of the class attributes and see if it can still calculate the correct input and output impedance. Call the corresponding methods for missing required data or raise an exception if there is not enough data. The third will provide the method with data that is the incorrect data type that the method accepts. For example, passing in a string when the method only accepts integers, floats, and doubles. The method is accepted to raise the invalid data type exception.

$$Z_{in} = \frac{V_1}{I_1} = \frac{A_n V_4 + B_n I_4}{C_n V_4 + D_n I_4} = \frac{A_n \frac{V_4}{I_4} + B_n}{C_n \frac{V_4}{I_4} + D_n} = \frac{A_n Z_L + B_n}{C_n Z_L + D_n} \quad Z_{out} = \frac{D_n Z_S + B_n}{C_n Z_S + A_n}$$

Equation 4 Equation for input and output impedance

- **Calculate Gain –** This method calculates the block's voltage and current gain using the given class attributes. This method will utilise the chain matrix analysis current and voltage gain equation (Equation 5). If an attribute is missing, the code will raise an exception or call the "CalculateOutputVoltage/Current' method. This calculation returns two floats; one containing the voltage gain from the input to the block's output voltage, and the other will contain the current gain from the input to output voltage of the block. If the value for input current, input voltage, or any denominator in Equation 5 are 0, the method raises an exception. Three tests will be conducted. This first will provide the method with a series of different predetermined values of the class attributes and see if the methods calculate the correct voltage and current gain. The second test will provide the method with only some of the class attributes and see if the method can still calculate the correct voltage and current gain or raise an exception if there is not enough data. The third will provide the method with data that is the incorrect data type that the method accepts. For example, passing in a string when the method only accepts integers, floats, and doubles. The method is accepted to raise the invalid data type exception.

$$A_V = \frac{V_4}{V_1} = \frac{1}{A_n + B_n Y_L} \qquad A_I = \frac{I_4}{I_1} = \frac{1}{C_n Z_L + D_n}$$

Equation 5 Current and voltage gain equations

From these attributes and methods, multiple circuit blocks can be created and called upon when needed to identify voltage, current, or power in different parts of the circuit. Each of these blocks would be linked together so that the output voltage and current of one block would be the input voltage and current to another block. Lists store the values of the voltages and currents. The lists passed from one block to the next block will not be copied but only create a new pointer to the memory address. This allows us to change the information in block two and have all data automatically updated once called via recursion. (i.e. The voltage, of block seven points to the output voltage of block six, which requires the same block's input voltage, which points to the output voltage of block five et cetera). This uses the cascade of several circuit elements from Equation 6.

$$\begin{aligned} \begin{bmatrix} V_1 \\ I_1 \end{bmatrix} &= [T_1] \begin{bmatrix} V_2 \\ I_2 \end{bmatrix} \\ &= [T_1][T_2] \begin{bmatrix} V_3 \\ I_3 \end{bmatrix} \\ &= [T_1][T_2][T_3] \begin{bmatrix} V_4 \\ I_4 \end{bmatrix} = \begin{bmatrix} A_n & B_n \\ C_n & D_n \end{bmatrix} \begin{bmatrix} V_4 \\ I_4 \end{bmatrix} \end{aligned}$$

Equation 6 Cascade of several circuit elements equation

ii. Functions

The following are the three parts that make up the code: reading the file, calculating requested values, formatting the data. Each has a series of various functions which flow in a specific order, as seen in Figure 2. Three functions are not included in the flow as they will be called numerous times throughout the code.

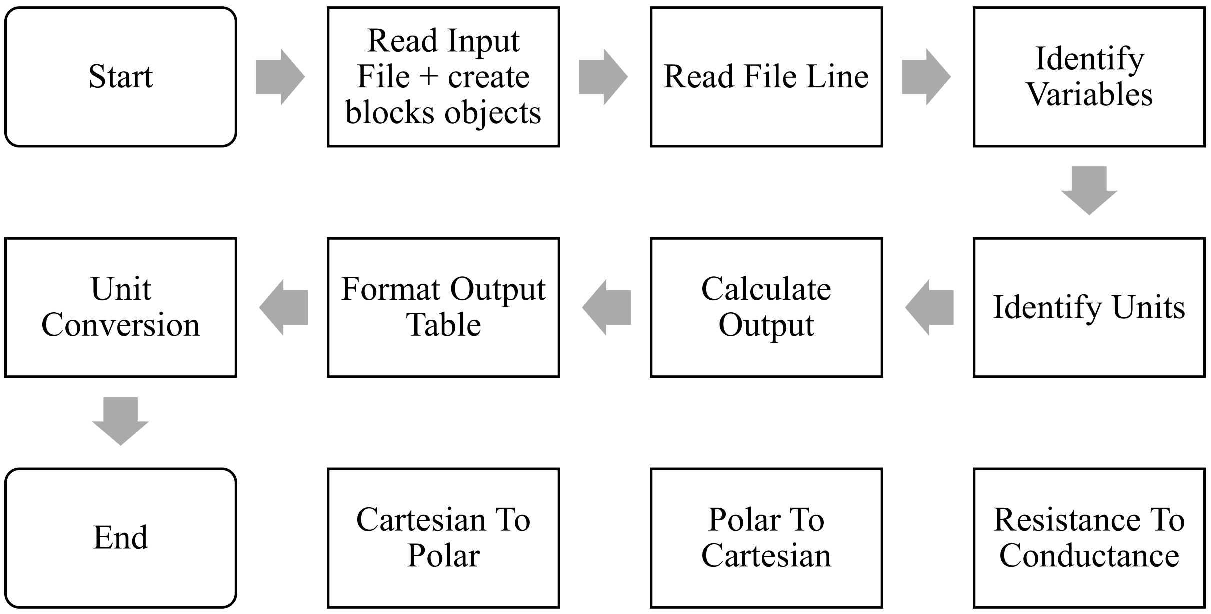

Figure 2 Function Flowchart

- **Main –** The program calls this function first. The method calls the Read Input File function and passes the circuit file's name into that function. That function returns a 2D-array containing information about the required outputs, their units and their exponents, a list of the created blocks, and a list of frequency information. From this, it calls the Calculate Outputs function and passes the returned data. That function returns a 2D-Array containing all the calculated values. The Format Output Table function formats that 2D-array into the desired format.
  Arguments:
  - (string) The filename of the circuit file to be analysed

  Returns:
  - (2D-Array) the table containing the desired information of the circuit.

  Testing
  - Big Bang integration testing to identify how each component interacts with each other.
  - System testing to evaluate the systems compliance with the specification.
  - Both these test will utilise the provided test files on Moodle.
- **Read Input File –** This reads the input test file to identify the required outputs and the given inputs. Loop over each line of the input and call the Read File Line function. If the line is in the <CIRCUIT> section, call create an instance of the 'Block' class and store the object in a list. If the line is in the <TERMS> section, create a new 'block' object with Thevenin's Voltage and resistance or Norton's current and resistance, and another for the load resistance. The function adds these two objects to the object list. The function orders the object list, so the source block goes first, and the load block goes last. Figure 1 shows the order. The function defines the order by the concatenation of the first and second nodes name (n1 + n2). i.e. nodes '20' will be before nodes '23'. Additionally, create a list of information regarding the frequencies. Find the <OUTPUT> section and identify the required outputs by calling the Identify Units function. Store the required outputs, units, and power in a 2D-array. The function uses a while loop to read each line in each section as the number of lines is unknown and a linear search to search for each sections' headers and footers.
  Arguments:
  - (string) The filename of the circuit file to be analysed

  Returns:
  - (2D-array) the required outputs, their units, and their exponents.
  - (List) the list of circuit blocks created in order.
  - (List) frequency information

  Testing:
  - Pass in a filename that exists and see if the desired output is correct.

  - Pass in a filename that does not exist, or the data is of the wrong type to see if the function raises an exception without breaking the program.
  - Pass in a filename that does exist but does not contain one of the three sections, i.e. <CIRCUIT>, <OUTPUT>, or <TERMS>, and see if it raised an exception and points to the missing section.
  - Reorder the sections in the file and determine if the program can still identify each section.
- **Read File Line -** Read a given line from the input file. Split up the given string using spaces input into multiple lists containing each variable and its value. Pass each list to the Identify Variables function using a loop. The function uses an if clause to identify if the line beings with '#' denoting that line as a comment and irrelevant to the function.
  Arguments:
  (String) A string of the given line in the file.
  Returns:
  - (List) A list containing the name of the variables and their value

  Testing:
  - The programmer will pass a string of data from a line in the test file to the function and check if it has split up the string correctly.
  - The programmer will pass a string of data from a line in the test file to the function without any spaces and check if it has not split up the string.
  - The programmer will pass in a value of the incorrect type to see if the function raises an exception.
- **Identify Variables -** Read the line and identify the given variables in that line and assign them with the correct value by looping over each element in the list. The function will need to convert the value into the correct type. The function utilises try clauses to identify if a character is a float or a string without causing error.
  Arguments:
  - (String List) a list of strings to identify the variable and value of that variable

  Returns:
  - (float) returns the value of the variable
  - (string) name of the variable

  Testing:
  - Pass a list of strings containing just the variable name and see if it can identify that variable
  - Pass a list of strings containing just the values and see if it can convert those values into the correct data type
  - Pass a list of strings containing both the variable name and value separated by a space and see if it can do the above
  - Pass a list of strings containing both the variable name and value separated by an equal's sign and see if it can do the above
- **Identify Units -** Identifies the units of the given string (V, Amps, Ohms). Also identifies the power (milli, kilo, mega, et cetera). The function uses if clauses to differentiate between the name and the unit of the output. The function uses a linear search to find the exponents corresponding float value from a lookup table..
  Arguments:
  - (String List) A list of strings to be used to identify the outputs, the units, and the power

  Returns:
  - (String) name of the output, i.e. VoltageIn, GainCurrent, Power, et cetera.
  - (float) the exponent of that output i.e. (0.0001, 1000, 1*10^6)
  - (string) the unit, i.e. V, dB, Ohms

  Testing:
  - Pass in a prefix for a unit of measure, i.e. K, m, n, M (kilo, milli, nano, mega) and identify if it returned the correct value for that prefix, i.e. 1000, 0.001, -1*10^-9, 1*10^6 et cetera. If the programmer passes an unidentified prefix, raise an exception without breaking the program.
  - Pass in a string list to see if it can identify the name of the variable
  - Pass in a string list to see if it can identify the unit of the value

- **Calculate Outputs -** Loops through each block in the list to calculate the output voltage and output current by using each blocks' method. This is how the equation of cascade of several circuit elements (Equation 6) operates. From that information, it will call the block methods to calculate the requested information. The function utilises if clauses to identify which data needs to be required from the 2D-array of output elements then calls the blocks corresponding method. The function uses a while loop until all the required data has been calculated.
  Arguments:
    - (List) the list of created blocks of the circuit to be used to calculate the required outputs
    - (2D-array) the array of output elements required their units and their power.

  Returns:
    - (2D-array) an array of all the values calculated.

  Testing:
    - Test to see if power (in/out), voltage(in/out), current(in/out), gain(voltage/current), and impedance(in/out) were calculated correctly with given set values.
    - Test to see if the code knows which variables (power, voltage, gain) are required by the output.
    - Test to see if the code can calculate only the required variable information.
    - Test to see if the information is in a list in the correct order.
    - Pass over fake/missing data to see if the function can raise an exception without breaking the program.
- **Format Output Table -** Format the information into the correct format as desired by the lab script. Calls the Unit Conversion function to convert the calculated values into their desired format (decibels, milli, kilo). An if clause is used by the function to identify the required data's unit. This function may also need to call the Polar to Cartesian and Cartesian to Polar functions. The function uses an if statement to identify if a logarithmic or linear frequency is required. The first column of the table will list the frequencies linearly or logarithmically.
  Arguments:
    - (2D-array) the calculated values.
    - (2D-array) the desired output values, their units, and their exponent.
    - (List) the list of frequency information.

  Returns:
    - (2D-array) the formatted desired 2D-array

  Testing
    - Test to see if the table is in the correct format.
    - Pass fake/missing data to see if the function can raise an exception without breaking the program.
- **Unit Conversion -** Converts a value to its desired power. i.e. 5V = 5000mV if the given input is false. Else, it will convert it into Decibels. This function may also need to call the Polar to Cartesian and Cartesian to Polar functions.
  Arguments:
    - (float) the value to be converted
    - (float) the power
    - (string) unit of the value to be converted
    - (bool) whether it is in decibels

  Returns:
    - (float) the converted value

  Testing:
    - Test to see if the input value can be multiplied correctly by the power by passing in predetermined values for the converted value and exponent and comparing the results to hand calculated results.
    - Test to see if the input value can be converted into decibels bypassing predetermined values for the converted value and unit and comparing the results to hand calculated results.
    - Test to see if it can identify whether it should convert the value into decibels or multiply it by the power by providing predetermined values for all inputs and comparing the results to hand calculated results.

  - Pass in fake data to see if it can raise an exception without breaking the program.

- **Cartesian to Polar -** Converts the given inputs from Cartesian form to polar form. The function uses an if clause to identify if the real value is 0 and raises an exception as you cannot divide a number by 0.
  Arguments:
  - (float) the real value.
  - (float) the imaginary value.

  Returns:
  - (float) the value for magnitude.
  - (float) the value for phase.

  Testing
  - Pass in values to see if their polar counterparts can be calculated correctly. The magnitude will always be equal to or larger than zero. If the imaginary value is zero, the phase will be zero.
  - Pass in fake/missing data to see if the program can raise an exception without breaking the program. If the real value is zero, the function should raise an exception
- **Polar to Cartesian -** Converts the given inputs from polar to Cartesian form.
  Arguments:
  - (float) the value for magnitude.
  - (float) the value for phase.

  Returns:
  - (float) the real value.
  - (float) the imaginary value.

  Testing
  - Check if the Cartesian form is correct, given a set of polar values. If the phase is 0, π, 2π, or any multiple of π, the imaginary value will return 0 and the real value will be indistinguishable from the magnitude value. If the phase is π/2, 3π/2, 5π/2, or any other multiple of π plus π/2, the real value will return 0 and the imaginary value will be indistinguishable from the magnitude value.. If the magnitude is 0, both values will return 0.
  - Pass in fake/missing data to see if the program can raise an exception without breaking the program.
- **Resistance to Conductance -** Converts resistance into conductance or vice versa as the conversion equation is the same. The function uses an if clauses to check if the input is a zero. If so, the function calls an exception as nothing can be divided by zero. The function uses an if clauses to check if the input is a negative as you cannot have negative resistance or conductance.
  Arguments:
  - (float) the resistance or conductance to be converted.

  Returns:
  - (float) the calculated conductance or resistance.

  Testing
  - Check if the value calculated is correct, given a set of inputs. Any input larger than one will result in a value between zero and one, and vice versa.
  - Pass in fake/missing data to see if the function can raise an exception without causing the program to break. A negative value should raise an exception as only positive integers, doubles, and float are allowed.
  - Pass in a zero to see if an exception was raised.

CONCLUSION

By utilising these functions, the Main function can accept a filename for a given circuit file and read the file's data to create small chunks of the circuit by using a custom class. Each block's methods can then calculate the circuits output and input voltages, current, power, impedance, and gain. These values can be converted into the desired units and stored within a formatted table. The Main function can then return this table of results.

The program also aims to complete certain extended features like handling unit exponent prefixes, conversion to decibels, and logarithmic frequency as seen in the Unit Conversion function and Format Output function.

```
     Freq,      |Vin|,      /_Vin,     |Vout|,     /_Vout,      |Iin|,      /_Iin,
       Hz,        dBV,       rads,        dBV,       rads,        dBA,       rads,
1.000e+01,  1.116e+01,  0.000e+00, -2.716e+01,  0.000e+00, -3.115e+01,  0.000e+00,
1.111e+06,  1.116e+01,  0.000e+00, -2.716e+01,  0.000e+00, -3.115e+01,  0.000e+00,
2.222e+06,  1.116e+01,  0.000e+00, -2.716e+01,  0.000e+00, -3.115e+01,  0.000e+00,
.
.
.
```

Figure 3 An example formatted results table