Ryerson University

Department of Electrical, Computer, & Biomedical Engineering
Faculty of Engineering & Architectural Science

Course Title:	Capstone Design
Course Number:	ELE70B
Semester/Year (e.g.F2016)	W2022

Instructor:	Dr. Balasubramanian Venkatesh

Assignment/Lab Number:	N/A
Assignment/Lab Title:	EDP: Theory and Design Section

Submission Date:	11 FEB 2022
Due Date:	11 FEB 2022

Student LAST Name	Student FIRST Name	Student Number	Signature*
Fairoj	Rehnuba	500750254	RF
Ahsan	Abrar	500722182	AA
Shirazi	Muhammad	500753756	MS
Habibi	Parham	500781855	PH

*By signing above you attest that you have contributed to this written lab report and confirm that all work you have contributed to this lab report is your own work. Any suspicion of copying or plagiarism in this work will result in an investigation of Academic Misconduct and may result in a "0" on the work, an "F" in the course, or possibly more severe penalties, as well as a Disciplinary Notice on your academic record under the Student Code of Academic Conduct, which can be found online at: http://www.ryerson.ca/senate/current/pol60.pdf

Table of Contents

Theory and Design

Ladder iterative load flow calculation techniques are carried out to obtain node voltage and current in a transmission line while minimizing the error within a preset tolerance. A detailed description of the ladder iterative load flow was completed in the EDP Fall Report [1]. This report shall add or correct the initial theory and design process.

Since the EDP Fall Report [1] submission, changes were made to the design and implementation to better optimize the software's performance as well as account for new scenarios. Each component of the software was upgraded to make it more modular, and error cases were accounted for.

Theory

Ladder iterative technique is applied to RDS as opposed to other techniques as the others often lead to poor convergence.[2][3] Executing this algorithm involves a cycle of forward sweep calculations to acquire voltage, error calculation and analysis as per the tolerance requested and finally a backward sweep to calculate the current along each bus. This is only viable when the line and load impedances, and source voltage at the RDS is defined initially. [1]

Forward and Backward Sweeps

To begin, the initial conditions are set in which all the nodes are declared to have zero volts and amperes. The error tolerance is also specified here. Firstly, the forward sweep is conducted but where no load conditions are assumed. As there is no load, there is no current hence the voltages at each node are equal to the source voltage.

$$V_5, V_4, V_3, V_2 = V_S \tag{1}$$

Error calculations are carried out where the initial voltage which in this case is zero is compared to the first forward sweep voltages. Given that the nominal voltage is equal to the source voltage, the calculation is as follows:

$$\frac{|V_5|-|V_{Old}|}{V_{Nominal}} = 1 \tag{2}$$

As the error is greater than the tolerance, the first backward sweep is run. The backward sweep calculates the current across each bus.

$$I_5 = \frac{S_5}{V_5} = I_{45} \tag{3}$$

$$I_{34} = I_4 + I_{45} \tag{4}$$

With the new currents, a second forward sweep is run. At this time, the voltages at each node are no longer the same as the voltage at the source. Therefore, the must be recalculated using the following equations:

$$V_2 = V_S - Z_{12}I_{12} \tag{5}$$

$$V_3 = V_2 - Z_{23}I_{23} \tag{6}$$

This is repeated until the final node, after which, the error calculation is run again. If this time, the value is less than the tolerance, the iterative calculations can be concluded and the results are saved. Otherwise, the entire process above is recycled until the error conditions have been satisfied.[1]

Software Platform

Ladder Iterative algorithm implementation is not only limited to Python. MATLAB is the ideal choice for these calculations as this programming language is highly time efficient at carrying out iterative calculation. However, MATLAB is a costly programming language. Alternatively, VBA and Excel can also be used, but are very time inefficient. Therefore, Python was chosen as the software platform due to the low cost, beginner friendly and versatile environment. Processing speed can be further optimized by including free-to-use libraries such as NumPy [4] and Pandas [5] for manipulating large datasets.[1]

Design

Data Parser

The data parser is successfully able to identify and extract the bus and branch data into separate arrays. Previously, they were fed into the system using separate excel files, but it has been changed to account for a single excel file. It identifies the individual table by searching for the exact string (BUS DATA FOLLOWS, BRANCH DATA FOLLOWS) as provided on the CDF file. Once the index is identified, it searches for the next "-999" which identifies the end of the table. The two indices are the start and end of each table, and the needed columns are extracted into two separate arrays. The values are in per-unit.

Per-unit notation is used to express the units in reference to a base value. Carrying out the calculations without establishing the base values would result in an incomplete calculation, as such, they are also provided. The voltage base is part of the BUS TABLE and is extracted alongside the other information. The resistance and reactance values are provided in the BRANCH table, and they are extracted and combined to create the impedance values. However, the power base is not part of the tables, but instead part of a string header. To carry out this extraction, an additional function is added to parse through the string. This is done by separating each item in the string by their respective white-space into a list. The base value will be in the 3rd index of the list, and is extracted as such and then converted to a floating-point value.

Calculation Engine

This component represents the core of the software as this performs the forward and backward sweep calculations. The calculation engine is reliant on the GUI and Data Parser for parameters used in this process. The GUI helps the user interact by adding the excel file with the input data and specifying the error tolerance. The Data parser sorts the information from the excel file and converts it for the calculation engine to use.

Up until now, the calculator engine was used on a linear network but radial distribution networks often consist of branches of data. Each node may have one or more nodes in line which means an increased complexity. Each branch would have to be calculated separately as a unique branch in order for ladder iterative load flow calculations to work. This means that the calculation engine must be able to identify where in the array branching occurs and be able to refer back to the branching node to carry out backward and forward operations on the new branch.

The current calculation engine consists of three main functions; namely forward sweep, backward sweep and convergence test with reference to the tolerance. As the calculation engine receives data from the parser, the bus data arrays are used to establish bus characteristics while the branch data array is used to establish the connectivity between each node. Within this array, where there is a branch the subsequent node will show more than one connection to the same node. Therefore, an additional function to the design is required.

The duplicate finder function in the calculation engine would store the location of the new branches by recording them in a separate array. Once the branch calculations have been completed, the program is to refer to the new array and point back to the origin node in order to carry out ladder iterative load flow calculations on the primary branch. This is an instance where modularity in code is extremely helpful.

GUI

The GUI represents the user's interaction with this software. Using the PySimpleGUI library in Python, a GUI was created to open the excel file. The Browse button upon clicking would open a directory window to open the input file to be relayed to the Data Parser. If the input file type was not an excel file, there would be an error message displayed which would have requested the user to enter the correct file type. If the file could not be converged, the GUI would display an error message.

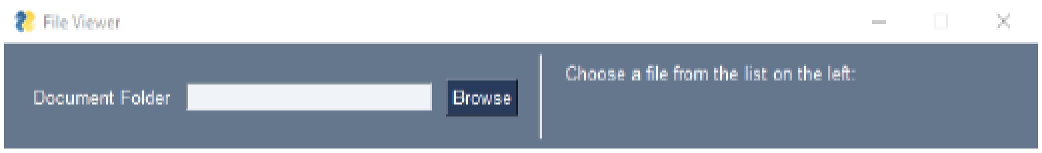
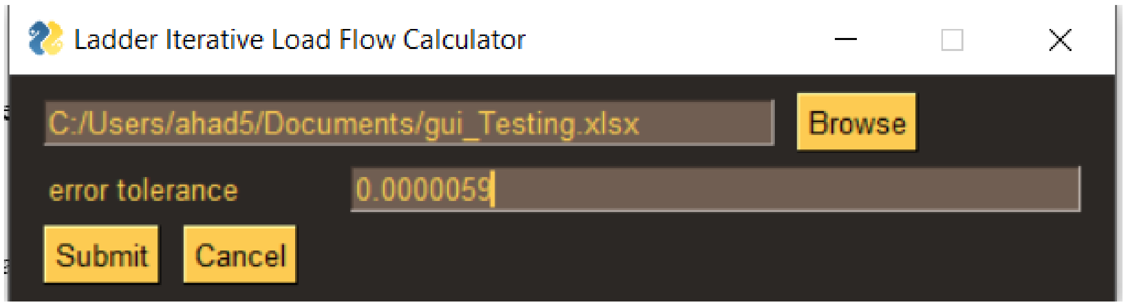


Figure 1: Previous Design of the GUI

As shown in Figure 1, the design of this GUI is very simple and limited. It does not allow error tolerance to be entered and the solutions cannot be interacted with at all. Therefore the GUI had to be extended to allow more interactions.

Firstly, the browser logic was changed to only open files that are type .xls. This removed the need to use an error message regarding the wrong file format as the directory already filters out the unwanted file types for us. After the user has selected a file from the directory a submit button was also included. In case there has been an error in file entry, the user can click browse once more more redirect the file. This way there is a buffer between when the file is uploaded to when the code is run. Furthermore a tolerance block was incorporated in which the user can specify the error tolerance thus making this program more versatile. The default tolerance, if the user does not specify, has been limited to 0.0001 as per the requirements.

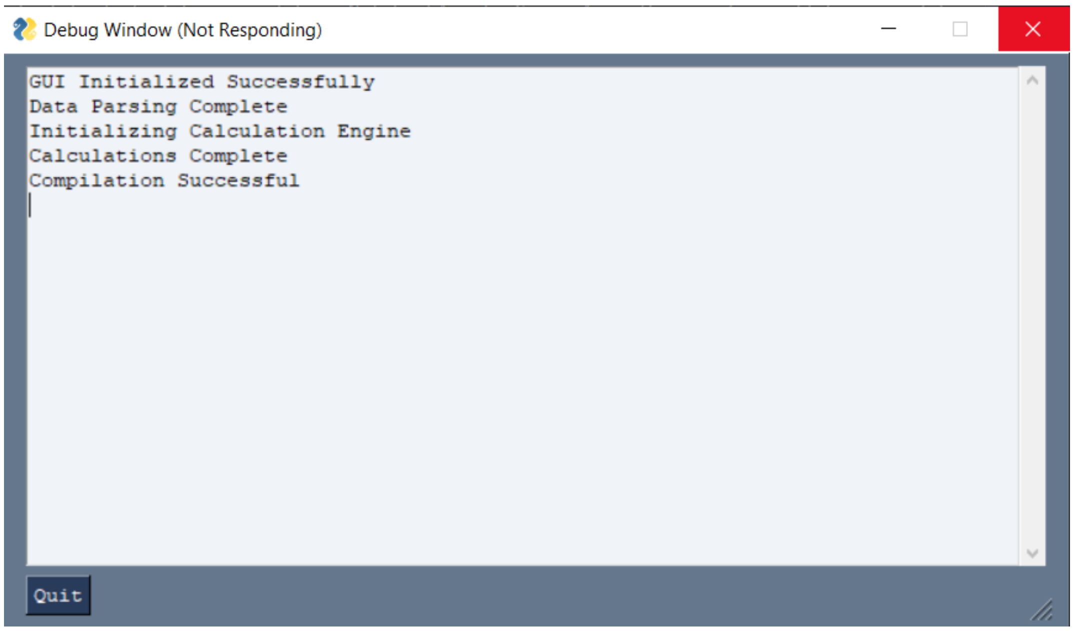


C:/Users/ahad5/Documents/gui_Testing.xlsx

error tolerance 0.0000059

Figure 2: Directory path, Submit button and error tolerance input

Another change is the addition of two GUI’s. The first is the runtime status GUI. This is prompted by the submit button and will update the user on which code has been run and at what point. The GUI will list the status updates as the code is run via a debug window. Within this parameter, any failed or passed instruction will be displayed. Once the code is completed, a final status will be displayed indicating that the program has completed running.



```
GUI Initialized Successfully
Data Parsing Complete
Initializing Calculation Engine
Calculations Complete
Compilation Successful
```

Figure 3: Runtime Status GUI

Finally, the GUI will also include an option for the user to save the output data once the calculations have been completed allowing the user to view the initial data as well as the final results output from the calculation engine.

References

[1] A. Ahsan, M. Shirazi, R. Fairoj, P. Habibi, “EDP Fall Report”, Capstone Fall Report, Ryerson University, Toronto, ON, 2021

[2] W. H. Kersting, Distribution System Modeling and Analysis, CRC Press, 2018

[3] W. Kersting, "A Method to Teach the Design and Operation of a Distribution System," IEEE Transactions on Power Apparatus and Systems, Vols. PAS-103, pp. 1945-1952, 1984.

[4] NumPy, [Online]. Available: https://numpy.org/

[5] "pandas - Python Data Analysis Library," [Online]. Available: https://pandas.pydata.org/.