Ryerson University

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

Course Title:	Capstone Design
Course Number:	ELE70B
Semester/Year (e.g. F2017)	F2021

Instructor	Dr. Balasubramanian Venkatesh

Assignment/Lab Title:	EDP Fall Report

Submission Date:	December 03, 2021
Due Date:	December 03, 2021

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

Department of Electrical and Computer Engineering
Faculty of Engineering and Architectural Science

Python Program for Ladder Iterative Load-Flow

EDP Project Report – Fall Semester

Authors: Abrar Ahsan, Muhammad Shirazi, Rehnuba Fairoj, Parham Habibi
Faculty Lab Coordinator: Dr.Balasubramanian Venkatesh
December 3, 2021

Table of Content

Abstract

At the core of any power System distribution analysis is a comprehensive, efficient approach to calculating load flow accurately. The method used in this paper relies on sending end voltage, line impedances, and tolerance to reach convergence. Calculations involve multiple iterations of backward and forward sweeps. Ultimately, currents and voltages of each bus shall be obtained.

In this research, the calculations are to be performed using Python interactive environment with power and complex calculation-based libraries including Panda and NumPy. So far, the program contains five major parts, the GUI, data parser, sorter, and verification tool, and finally the calculation engine. The software accepts an IEEE CDF input file. This shall be received by the GUI and validated with the data parser. The data verification tools search content for errors before execution while the data sorter searches branch identification. The calculation engine then carries out backward and forward calculations, referencing the error with the tolerance at each iteration until convergence is achieved. The advantages of this iterative method truly shine in the case of radial distributive systems with non-linear components.

Introduction

Load-flow is a simulation of the power system, trying to predict the amount of power flowing on each transmission line connecting the generation to the load. Load studies are important as they are essential in planning future development projects. As these projects need to be planned for 10 years or more, the designers need to be able to run simulations and design systems quickly and accurately. Hence, the fast, and accurate operation of these simulations has been a topic of research for decades.

While different approaches have been taken to perform load-flow simulations, a good approach requires low memory storage, high accuracy as well as high speed, while remaining a robust tool. Many matrix-based approaches exist, such as Newton-Raphson, Gauss-Seidal, as well as iterative techniques for radial distribution systems (RDS). However, iterative techniques are not used for RDS often as they fail to converge on various instances.

Kersting [1] developed a load-flow technique that uses ladder-network algorithm, to solve radial networks. Ellis [2], Al-Awadi [3] extends this to different systems. Stevens et al. [4]'s implementation demonstrated that the ladder iterative approach is the fastest approach, but experiments failed to converge in 5 out of 12 cases studied.

Among other approaches, Musti et al. [5] implemented load-flow analysis using four different algorithms on Microsoft Excel using VBA. Even Artificial Neural Network approaches have been tried to find the most efficient way to calculate load-flow, as demonstrated by Arunagiri [6].

In this work, we present a Python-based ladder-iterative solution for radial networks, that takes inputs from Excel files and can run the algorithm and generate output as an Excel file as well.

Objectives

The main objective of this project is to develop a software solution in Python to implement the ladder-iterative technique to calculate load-flow for various power system networks. The end goal is to be able to implement a system that is optimized enough to compete, if not surpass, past works involving iterative techniques.

The Python program can run calculations in 3-phase circuits, based on the data-input via excel file, and outputs the results after convergence in an excel file as well.

Theory

As RDS systems typically do not employ iterative techniques due to poor convergence, the ladder-iterative algorithm is used [1] [7].

For a ladder network, line, load impedances, and the voltage at the source (V_S) is known. The algorithm involves a forward sweep to calculate the voltages at each node under, followed by an error calculation which is compared to the preset tolerance, and finally, a backward sweep to calculate the current along each bus.

The following sections will use Figure 1 as an example.

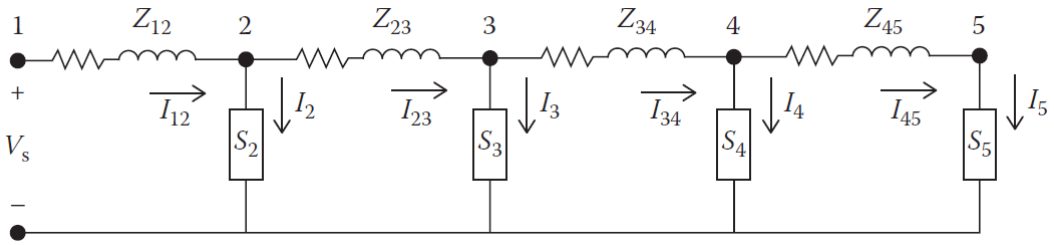

Figure 1: Non-linear Ladder Network [7]

Forward and Backward Sweeps

The initial conditions are first set before the sweeps begin. This sets the initial voltage at each node as 0, current as 0, and the preset tolerance of the algorithm.

The first forward sweep assumes no-load conditions and calculate the voltage at each node. As current is 0 under no-load, the voltage at each node is equal to the voltage at the source.

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

Error is calculated at the final node by comparing the new value with the old value.

as $V_{Old} = 0$, the error is found to be 1

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

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

$$I_5 = \left(\frac{S_5}{V_5}\right)^* = I_{45} \tag{3}$$

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

The current is calculated across each bus, and then a second forward sweep is run. At this time, we have current, the voltages at each node are no longer the same as the voltage at the source.

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

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

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 final values are found and sent as the output. If not, the entire process above is repeated until the error conditions have been satisfied.

Software Platform

The ladder-iterative algorithm can be implemented on various platforms. The simplest method is to use MATLAB as it can perform large calculations at very fast speeds, however, the limitation present here is that MATLAB is not readily available and has a very high license cost. Alternatively, VBA and Excel can also be used, as demonstrated by [5], but it is very slow as a programming language. Thus, Python was chosen as the software platform due to its beginner-friendly programming style, large number of readily available libraries to optimize performance, as well as the fast processing speed of NumPy [8] and Pandas [9] for manipulating large datasets.

Preliminary Design

The Python program has been divided into 3 modules, which are discussed in the following sections. Each module had its own development stage to demonstrate viability, before being improved for optimized implementation. They will be further expanded before the integration of the full system.

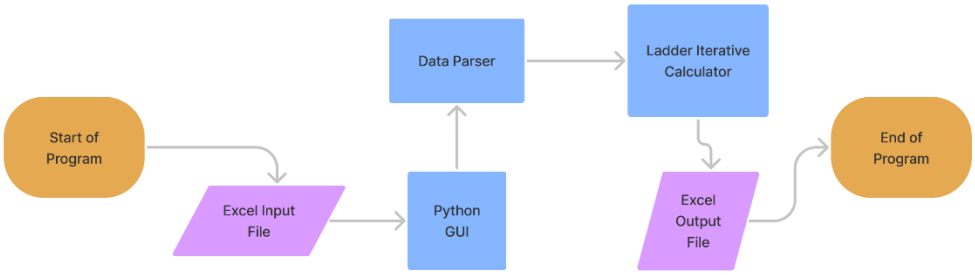

Figure 2: High-Level Software Flowchart

Graphical User Interface

The Graphical User Interface (GUI) allows the user to interact with the software. The user can select the input data file, send error prompts when problems are encountered, as well as give the user to option to select the output type, as an Excel file output, or a viewable XML output directly on the GUI.

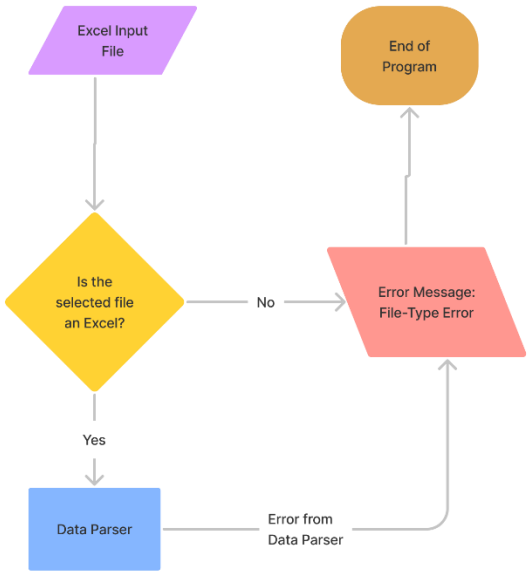

Figure 3: Graphical User Interface Flowchart

The first iteration of its development was to set up a small pop-up window with a button to close the said window. The second iteration allowed the user to browse and select the data file to be inputted. The third iteration expanded on this, adding a file-type verification, limiting the user to only select Excel file types. Finally, the fourth iteration expanded to display error messages. There are three primary error messages:

1. Incorrect File-Type: If the program is unable to recognize the file type, it will send a file-type error.

2. Data is not in CDF Format: If the data in Excel does not follow the IEEE Common Data Format, it will send a data-format error.
3. Failure to Converge: If the calculation engine is unable to converge, i.e., unable to reduce the error percentage low enough, it will send a convergence error.

Only the file-type error is checked in this module. The data-format error is checked in the Data Parser module and the convergence error is checked in the Calculation Engine.

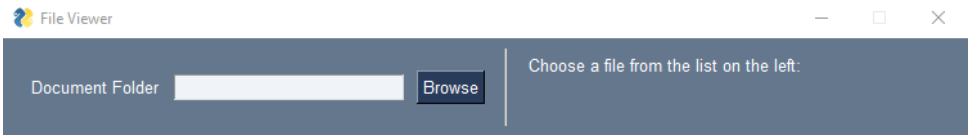

Figure 4: Third Iteration of the Graphical User Interface

Data Parser

The Data Parser's primary objective is to extract the relevant information from the input Excel File. To accomplish this, Python's Pandas library is used [9]. This allows the program to extract the information quickly and efficiently, while maintaining the worksheet structure of Excel. Once extracted, the Pandas table is verified, to check whether it follows the Common Data Format (CDF) as well as checked to see if the system is linear or non-linear.

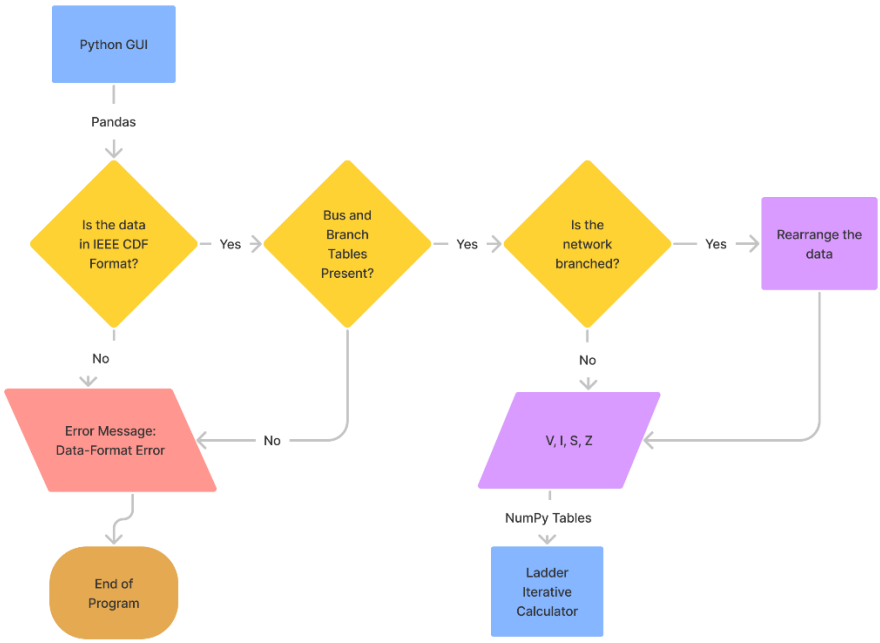

Figure 5: Data Parser Flowchart

Data-Format Verification

The extracted Panda table is verified to ensure that it follows IEEE CDF, and then it ensures that the correct number of tables are present. The Pandas table is split along the rows that contain "-999" in the first column. These rows are identifiers for the end of the table. If the table count is wrong, that is a source of error. The two main tables of concern are the Bus Data and the Branch Data tables. If either is missing or has the incorrect number of columns, that is another source of error.

If either error arises, a Data-Format error is sent back to the GUI. Otherwise, the data is converted into NumPy tables and then passed to the Data Sorter.

Data Sorter

The Data Sorter checks whether the system is linear or non-linear. A non-linear system has branches that separate at specified nodes. To determine whether the system has branches, the

Branch Data table is used and the count of unique values [10] is run using NumPy on the first column which represents the "From" column. If any value has more than 1 occurrence, that "From" node represents where branching occurs.

If the system has branches, the data is rearranged. The first iteration of the sorting algorithm consisted primarily of For-loops for moving data around. The second iteration took advantage of NumPy's sort functions [11] [12] to rearrange such that the branch's rows are directly under the branching node and above the rows containing data for the rest of the system. The second iteration still requires further testing to ensure perfect performance.

Calculation Engine

The Calculation Engine is the final module of the Python Program. It is the main component of the entire software solution. The module takes the NumPy arrays as input from the Data Parser and initializes its own NumPy array with initial conditions before starting the process. The tolerance for debugging purposes has been set at 0.0001, but the program can be made flexible to take the tolerance from the user as well.

The calculation engine consists of the forward sweep function for the voltage calculations, the backward sweep function for current calculation, and the error calculation function.

The first forward sweep is not performed as the initial conditions for it is no-load, as such, all the voltage values are the same as V_S. We start with the first backward sweep. This calculates the current across each bus using the voltages at the nodes, the line, and load impedances (Equation 3, 4). After the backward sweep is complete, the forward sweep starts, using the calculated currents to get the voltages at each node (Equation 5, 6). Once complete, the error is calculated by comparing the new voltage with the old (Equation 2). If the calculated error is not smaller

than the tolerance, the loop runs again. This process continues indefinitely until the error condition is satisfied.

During these runs, if the error value plateaus and stops growing smaller, it is said that the calculation has failed to converge, creating an error case. If allowed to continue running, the system will remain frozen inside the loop, unable to escape its local minima. To account for this, the old error and the new error values are compared in each loop, and if they are nearly identical, a convergence failure is prompted, and the program exits with an error message. This is a fail-safe to ensure the program is robust enough to know when it has failed.

If convergence failure does not occur, and the error does indeed get smaller than the tolerance set, the resulting NumPy array is sent back to the GUI, allowing the user to decide whether to select the output as an Excel file or view it as an XML on the GUI.

The first iteration of the calculation engine was verified by running a 3-bus linear system from [7] and the calculations were matched with hand-written solutions. However, for larger networks, this was not possible, so PandaPower [13] was used instead. The second iteration of the calculation engine was verified by creating a 10-bus system. The results were verified using PandaPower and the module was found to be accurate.

Further testing and optimization are still required for larger systems, as well as for non-linear systems

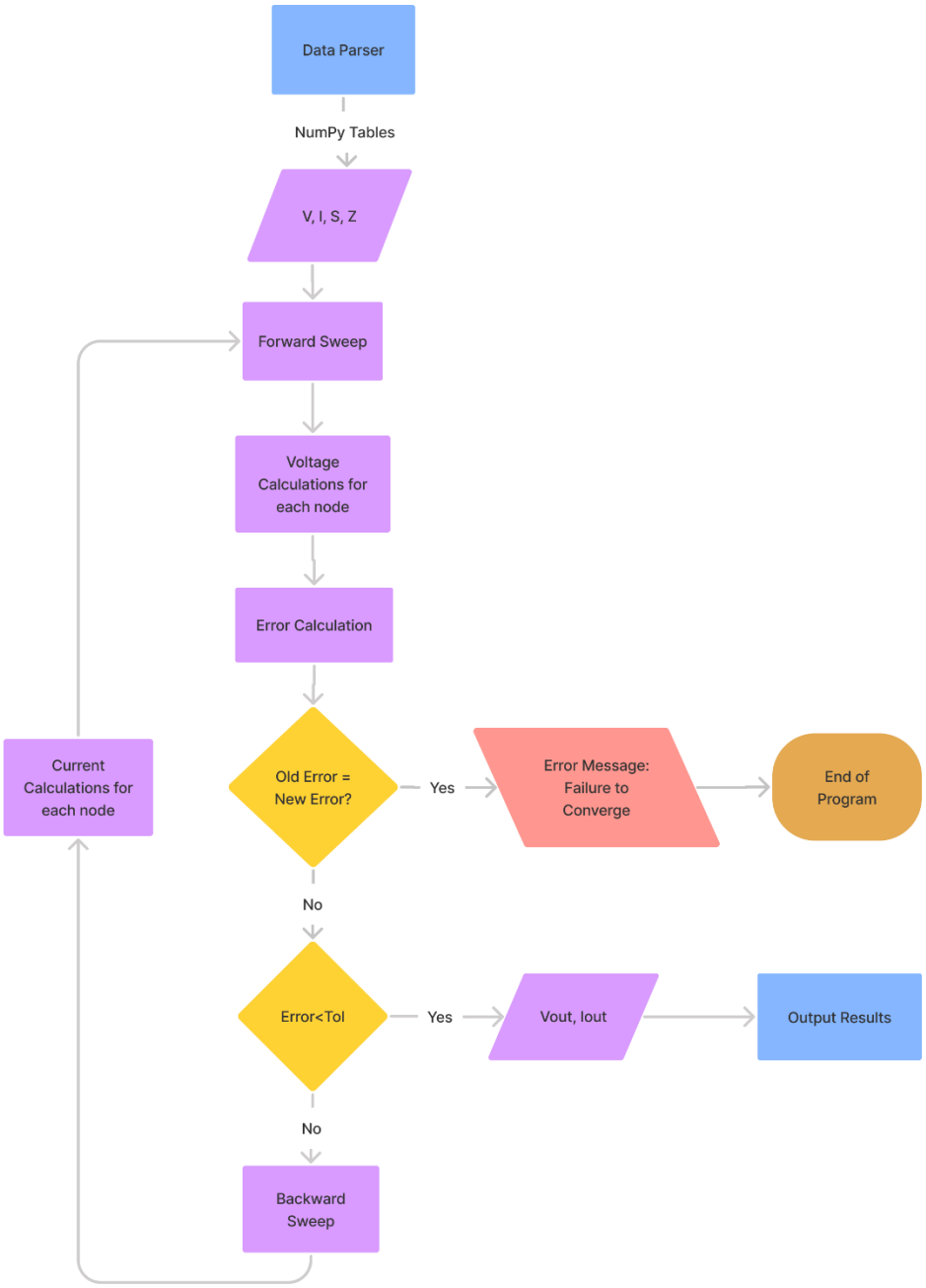

Figure 6: Second Iteration Calculation Engine Flowchart

Gantt Chart

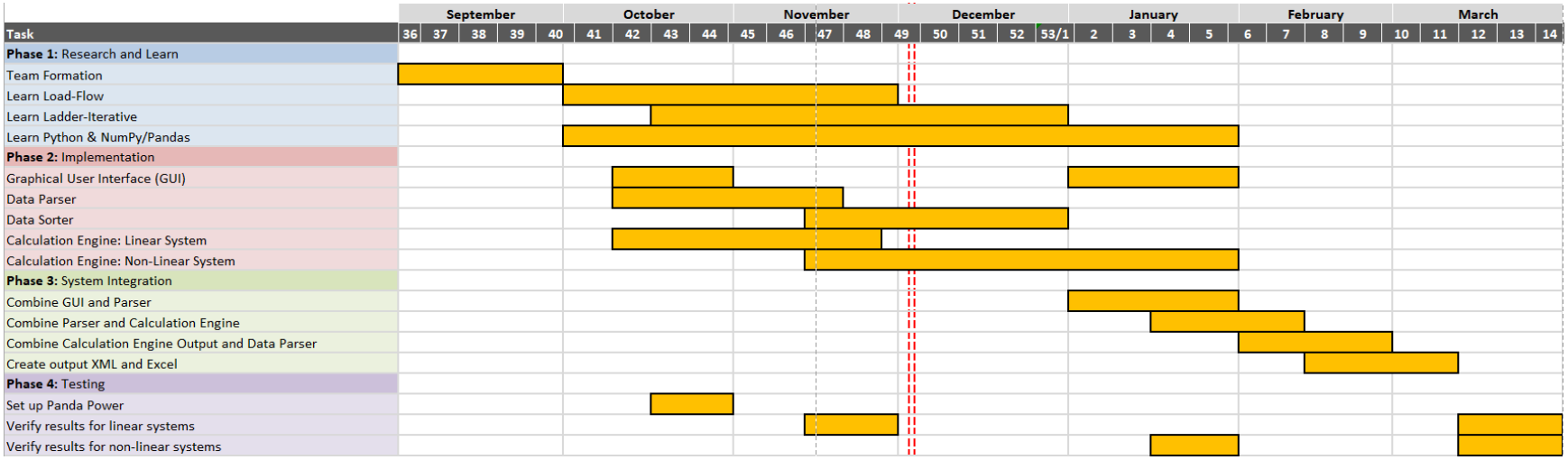

Figure 7: Project Timeline with Major Tasks

The timeline provided has been set while accounting for everyone's course load and time availability. While some basic elements have already been completed, further optimization and full system integration is the main challenge for the team during the Winter semester.

Appendices

Bibliography

[1] 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.

[2] M. V. Ellis, "The ladder load-flow method extended to distribution networks," 1994.

[3] A. K. Al-Awadi, "Load Flow Method for Radial Systems with Distributing Generation," 2008.

[4] R. A. Stevens, D. T. Rizy and S. L. Purucker, "Performance of conventional power flow routines for real-time distribution automation applications," 01 01 1986.

[5] K. S. S. Musti and R. B. Ramkhelawan, "Power System Load Flow Analysis Using Microsoft Excel," *Spreadsheets in Education*, vol. 6, p. 1, 2012.

[6] A. Arunagiri, B. Venkatesh and K. Ramasamy, "Artificial neural network approach-an application to radial loadflow algorithm," *IEICE Electronics Express,* vol. 3, no. 14, pp. 353-360, 2006.

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

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

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

[10] "NumPy.Unique," [Online]. Available: https://numpy.org/doc/stable/reference/generated/numpy.unique.html.

[11] "NumPy.searchsort," [Online]. Available: https://numpy.org/doc/stable/reference/generated/numpy.searchsorted.html#numpy.searchsorted.

[12] "NumPy.where," [Online]. Available: https://numpy.org/doc/stable/reference/generated/numpy.where.html#numpy.where.

[13] "Panda Power," [Online]. Available: http://www.pandapower.org/.

[14] U. Singla and R. Bala, "A Simple Method for Load Flow Solution of Radial Distribution Systems," 2016.

[15] S. Gosh and D. Das, "Method for load-flow solution of radial distribution networks," 1999.

[16] G. Chang, S. Chu and H. Wang, "A Simplified Forward and Backward Sweep Approach for Distribution System Load Flow Analysis," in *2006 International Conference on Power System Technology*, 2006.

[17] T. Alinjak, I. Pavić and M. Stojkov, "Improvement of backward/forward sweep power flow method by using modified breadth-first search strategy," *IET Generation, Transmission & Distribution*, vol. 11, no. 1, pp. 102-109, 2017.