Development of a Network Design Tool for the Herman-Beta Extended Transform

Prepared by:

Isma-eel Khan

KHNISM003

Department of Electrical Engineering
University of Cape Town

Prepared for:

Dr. D. Oyedokun

&

Professor C.T. Gaunt

Department of Electrical Engineering
University of Cape Town

May 2023

Submitted to the Department of Electrical Engineering at the University of Cape Town in fulfilment of the academic requirements for a Master's degree in Electrical Engineering

Key Words: Herman-Beta, Probabilistic Load Flow, Stochastic Loads, Network Planning Tool

Declaration

I know the meaning of plagiarism and declare that all the work in the document, save for that which is properly acknowledged, is my own. This thesis/dissertation has been submitted to the Turnitin module (or equivalent similarity and originality checking software) and I confirm that my supervisor has seen my report and any concerns revealed by such have been resolved with my supervisor.

Name: **Isma-eel Khan**

Signature: Signed by candidate **Date:** 12 May 2023

Acknowledgements

Firstly, and most importantly, I would like to thank God for granting me this opportunity to further my studies. None of this would have been possible without Your blessing.

Secondly, I would like to thank my supervisors, Dr. David Oyedokun and Professor Trevor Gaunt, for their continuous guidance and for giving me the opportunity to receive funding for this research. I would not have taken this step without your support.

Thank you to my employer, the City of Cape Town, for granting me the ability to work in an environment where DIgSILENT and other network planning tools are frequently used. The experience I have gained through my profession has helped me tremendously with this dissertation.

I am extremely grateful to Dr. M.J. Chihota for his assistance regarding the operation of his MATLAB program. Without your help, I would not have been able to develop my program accurately.

Finally, a special thank you goes to my wife, Rushda, for her continued love, encouragement and kind words when I was struggling. To my son, who was always my ray of sunshine and my main motivation. Thank you to all my family and friends for believing in me when I needed it most. Your support means the world to me.

Abstract

The Herman-Beta method has been one of the most important network planning tools used in South Africa. Over decades, it has transformed from being able to perform probabilistic load flow studies for LV passive feeders to an algorithm capable of performing PLF studies for passive/active feeders of any voltage. Furthermore, the new algorithm reformulated the equations used to dispose of the underlying assumptions used in the original method. This reformulation came to be known as the Herman-Beta Extended Transform (HBET), a powerful network planning tool which could prove vital to network planners as the uncertainty in power systems increase.

Previously, the HBET was implemented using MATLAB programming software. In this dissertation, a set of user requirements and data structure to enable efficient handling of input and output data in the HBET was developed for an open-source platform.

Python programming language was chosen as the most suitable programming language to implement the program, due to its array manipulation capabilities and the plethora of information and help available online.

Four scenarios were used to test the accuracy with which the tool was created. The four scenarios included a 12-bus passive feeder, a 12-bus active feeder, a 33-bus passive feeder with laterals and a 33-bus active feeder with laterals. These results were tested against the results produced by the MATLAB tool, where it was previously proven to be accurate.

It was found that the percentile voltages, mean voltages and voltage standard deviations for all systems simulated in Python were identical to those simulated using the MATLAB tool, indicating that the tool had been implemented accurately, thereby validating the hypothesis.

Table of Contents

List of Figures

List of Tables

Acronyms and Abbreviations

The acronyms and abbreviations below are often used in the report:

1C – Single Core

3C – Three Core

ABC – Aerial Bundle Conductor

ADMD – After Diversity Maximum Demand

Al – Aluminium

Cu – Copper

DG – Distributed Generation

DLF – Deterministic Load Flow

GUI – Graphical User Interface

HBET – Herman – Beta Extended Transform

LV – Low Voltage

MCS – Monte Carlo Simulation

MS – Microsoft

MV – Medium Voltage

PDF – Probability Density Function

PF – Power Factor

PLF – Probabilistic Load Flow

SLD – Single Line Diagram

XLPE – Cross-linked Polyethylene

1. Introduction

The purpose of this study is to produce an easily accessible, user-friendly design tool for network planners to accurately size radial feeders within power systems containing uncertainty. The program will use the extended transformation of the Herman-Beta algorithm, allowing for the determination of voltage variations at all voltage levels. The voltage variations calculated will be assessed against the South African National Standard for voltage variations and will state whether the designed network configuration, with user-defined inputs, complies with the specified limits. Thus, the network planning tool will help network planners to avoid under designing or overdesigning systems.

This chapter provides a background to the study and outlines the scope and limitations of the work to be undertaken. The research problem is stated, from which a hypothesis is formulated and finally, research questions are posed.

1.1 Background to the study

In power systems, one of the most important factors to consider is the quality of supply. Three main factors cause a reduction in the quality of supply, namely voltage drops, transients and harmonics [1]. Voltage drops are focused on in this research, since they can be accounted for in the design stage of network planning. The power quality is considered to be satisfactory if the voltage limits specified in the South African National Standards are complied with. Complying with these limits will prevent consumers' appliances from being damaged or malfunctioning.

To determine the voltage variation on network configurations, it is necessary to solve the load flow problem. Previously, deterministic methods were used to solve the load flow problem of low voltage feeders, whereby the number and type of households as well as the after diversity maximum demand per household was often used to determine the total load demand. Deterministic methods require inputs for an instant in time to determine a unique load flow result. These methods do not account for the variability of loads and variation in the production of distributed generation such as solar power. Since uncertainties exist in power systems, it has been found that deterministic methods are insufficient to accurately solve the load flow problem. Therefore, probabilistic methods have been used in recent years, since they account for stochastic loads and uncertainties in power systems.

Probabilistic methods are characterised into numerical and analytical methods. The main numerical method is the Monte Carlo simulation, which essentially repeats deterministic load flows numerous times to account for uncertainty. However, calculations done using Monte Carlo Simulations have a long processing time, since the number of repeated runs is typically in the order of 5000. Nevertheless, the results provided by this method are extremely accurate, making it ideal as a base case with which to compare other probabilistic methods.

The Herman-Beta method is an analytical probabilistic load flow technique that has been tested extensively and has been proven to provide accurate results in a relatively short period of time, compared to other probabilistic methods. The method has been accepted as the optimal way to determine voltage drops in South Africa, to such an extent that it has been incorporated into the South African National Standards.

The Herman-Beta method originally used the assumptions of unity power factor and consideration of feeders as purely resistive to allow for the easy calculation of voltage drops on Low Voltage networks. Thereafter, the model was adapted to allow for the inclusion of distributed generation. Finally, the

assumptions stated above were removed to develop the Herman-Beta Extended Transform (HBET), which allows for the calculation of voltage drops on Low, Medium and High Voltage radial networks [2].

A program was developed for the HBET using MATLAB programming language. However, the program is not easily accessible by network planners. This is due to the fact that MATLAB has limited accessibility, since it is only readily available to certain higher education institutions and is only available for commercial or industrial purchase at a tremendous cost. Therefore, the development of an easily accessible network planning tool based on the HBET is required.

1.2 Objectives of this study

The main objective of this study is to create a set of user requirements and data structure to enable efficient handling of input and output data in the HBET for an open-source platform. A further objective involves the analysis of network planners' requirements for assessing compliance of radial systems with voltage variation limits. Additionally, the way network planners handle uncertainty in power systems will be investigated.

1.2.1 Hypothesis

The hypothesis for this research is as follows:

"An open-source platform can be developed for efficient handling of network data and implementation of the voltage assessment capabilities of the HBET."

1.2.2 Research Questions

The Herman-Beta Extended Transform is an accurate method for calculating probabilistic voltage variations in a relatively short time. A program will therefore be developed using this method to allow network planners to easily determine whether their radial system designs will comply with specified voltage limits. This study attempts to find answers to the following research questions:

- What other probabilistic load flow techniques have been developed?
- How do network designers deal with uncertainty in power systems?
- What approaches were used to develop other network planning tools and what features were implemented in these programs?
- What is the ideal programming software to implement the Herman-Beta Extended Transform based on ease of learning, functionality and online support?
- What form of input is the most user-friendly and efficient for network planners?
- What output information will be useful to network planners and what results would they require to make informed decisions?
- How can the developed software program be broken down into simpler functions?
- What features should be implemented in the program to promote ease of use?

1.3 Scope and Limitations

The Herman-Beta Extended Transform has been developed for 3-phase, 4-wire systems and 3-phase, 3-wire systems. However, only 3-phase, 4-wire systems will be catered for in the developed program. Although the algorithm for 3-phase, 3-wire systems is presented at the end of this dissertation, this system type was not implemented in the developed program due to time constraints.

The scope of the work includes the investigation of other probabilistic load flow techniques, the determination of information that is available to network planners during the planning phase and the

information they require as outputs to ensure radial systems are being designed adequately. Once this information is obtained, the critical component of this study is to create an easily accessible, user-friendly program to assist network planners to determine whether their radial system designs are adhering to voltage variation limits. Finally, once the program has been implemented in the preferred software package, the results will be tested against those obtained from existing programs using the Herman-Beta method and the Herman-Beta Extended Transform.

Although the HBET allows for loads to be modelled as various probability density function types, this study restricts the modelling of loads as beta-probability density functions for simplicity. Therefore, other probability density functions will not be considered for the modelling of loads. Meshed networks will not be considered in this study because the HBET only applies to radial feeders. Finally, only uncorrelated systems will be considered in this study. Therefore, all equations of the HBET relating to correlated systems will not be presented.

1.4 Plan of development

Chapter 1 provides background information to the study. This includes brief discussions on quality of supply, voltage drops, probabilistic load flows and the Herman-Beta method. The hypothesis, research questions, scope and limitations are also provided.

Chapter 2 discusses literature relevant to the topic in an attempt to determine answers to the research questions. Firstly, various probabilistic load flow methods are investigated. Software packages are then investigated to determine which package is best suited to cater for the needs of the program to be developed. Literature is also analysed to determine the information available to network planners during the design stage and the information desired as outputs for decision making purposes.

Chapter 3 provides an in-depth look at the Herman-Beta method and the modifications made to include the effects of distributed generation. Thereafter, the Herman-Beta Extended Transform is presented. The underlying principles and assumptions are stated and the process of the algorithm is provided using the relevant equations and explanations.

Chapter 4 provides the methodology followed to develop the two network planning tools. Thereafter, methods used to evaluate the accuracy and correct translation of the Herman-Beta Extended Transform to Python code are discussed. Finally, the research methodology is analysed and limitations are stated.

Chapter 5 provides the process of developing the HBE-Excel and HBE-Python tools. Detailed designs for developing the two tools are presented. The inputs and outputs are discussed, as well as the importance and relevance of the outputs. Portions of code and pseudo code related to certain functions are also presented.

Chapter 6 involves the validation of the results produced by the developed tools. In this chapter, the tools are checked for errors which may have occurred in the translation of the algorithm to the two platforms by comparing the results of each tool to those obtained using the HBE-MATLAB program.

Chapter 7 concludes the study by providing conclusions and answers to the research questions. The validity of the hypothesis is also evaluated.

Appendices contain all work relevant to the study that has not been included in the main report.

2. Literature Review

This chapter discusses and analyses the relevant literature reviewed in relation to the research topic. The literature review is conducted to gain greater insight to the research topic and to determine answers to some of the research questions posed. Firstly, an overview of various probabilistic load flow techniques is provided. Thereafter, various software packages are investigated to determine the optimal platform to develop the network planning tool. Furthermore, the information available to network planners during the design stage is determined. Thereafter, the way network planners usually take uncertainty into account in their designs are discussed. Then, the output information required for network planners to make informed decisions is discussed. Finally, previously developed network planning tools are analysed to determine the approaches developers used to create their programs. The layout, functionality, appearance and drawbacks of these programs are also discussed.

2.1 Load Flow Calculation Techniques

Load flow calculations are used to determine the performance of a power system. The results from load flow calculations provide the active, reactive and apparent power flowing along feeders as well as the voltages present at sources and busbars. The current flowing along feeders are also calculated, providing the planner with an indication of how heavily the feeder is loaded. The solutions to load flows are also used to determine the network performance under (N-1) contingency conditions, which is essentially the load flow solution that will be obtained when a feeder or transformer is disconnected from the system.

Since voltage drops have a significant effect on the sizing of feeders, and it can be accounted for in the design stage of network planning, it will be the main factor considered when evaluating the effectiveness of the load flow calculation techniques discussed in this section.

Load flow calculation methods are broadly classified into two main groups, namely deterministic load flow techniques and probabilistic load flow techniques [5]. Using either method, voltage drops can be calculated by using the results obtained from the load flow solutions. Deterministic and probabilistic load flow techniques are discussed in this section.

2.1.1 Deterministic Load Flow

In deterministic load flow, inputs are characterised by exact values, such as the amount of power or apparent power a load will utilise. Methods using deterministic load flows for power systems include the Gauss-Seidel, Newton-Raphson and backward-forward sweep methods, whereby the power flow results obtained changes with every alteration to the input values. Deterministic load flows are often utilised by computer simulation packages such as DIgSILENT PowerFactory and PowerWorld.

The major drawback of deterministic load flow is that it does not account for uncertainties, such as the variability of power production from distributed generation as well as the variations in load demand at different periods of time [5].

In network design, specifically at utilities such as the City of Cape Town and Eskom, deterministic approaches are still being utilised when planning for future developments at the MV level. In these instances, the After Diversity Maximum Demand (ADMD) is used per household and multiplied by the number of proposed households to determine the total load that will be required by the development. These ADMD values depend on factors such as the area of development, property price and types of

appliances expected to be utilised in the proposed development. These factors all contribute towards a Living Standards Measure (LSM) of between 1 and 10, where LSM 1 generally refers to informal settlements and LSM 10 refers to the most affluent consumers.

Despite providing accurate results based on the inputs entered, it has been noted that deterministic load flow results vary due to the variable nature of generators and loads. Furthermore, in practice, power systems are subject to additional uncertainty due to the following [5]:

- Measurement errors.
- Load distribution variations between phases.
- Incorrect predictions of load demand required.
- Unplanned power outages.
- Incorrect equipment ratings or types.

Due to the abovementioned reasons, deterministic calculation methods are not sufficient to accurately determine the power flow and hence, the voltage drops in power systems. Therefore, they cannot be used to accurately determine conductor sizes in power systems.

2.1.2 Probabilistic Load Flow

Unlike deterministic load flow techniques, probabilistic load flow techniques account for stochastic loads as well as variations and uncertainties in power systems. Therefore, more accurate load flow results are obtained. Probabilistic load flow techniques use probability density functions as inputs and produce outputs in the form of probability density functions of the same type. A confidence level or design risk level is then assigned to determine a specific output value [5].

The concept of probabilistic load flow was first investigated by Borkowska [6] in 1974. Borkowska noted that deterministic methods are impractical since a tremendous amount of calculations are required to account for the variability of loads. Furthermore, it was noted that the analysis of such a large number of load flows is complex. Borkowska also stated that, whilst it may prove practical to randomly select a few sample points of the variable load, the results obtained from this selection would often prove to be inaccurate since the sample points are chosen arbitrarily. Hence, the misleading results may lead to an under or overdesign [6].

The purpose of the research done by Borkowska was to determine the probability density function of load flows across feeders. The assumptions below were necessary to obtain a solution to the problem being investigated [6]:

- The flow of active power is not dependent on the flow of reactive power.
- There is a linear relationship between the load flow along a feeder and the total load at the node connected to that feeder.

Borkowska utilised probability calculus to obtain a mathematical model which provided more accurate load flow results by taking uncertainties into account. The algorithm created allowed for the solution of networks containing a maximum of 100 feeders and 100 loads [6]. It was noted that, for a case utilising maximum feeders and loads, the calculation time is equivalent to that of the deterministic case.

Since Borkowska's investigation, other probabilistic approaches have been developed in an attempt to improve the accuracy of the results obtained. Probabilistic load flow techniques are grouped into numerical and analytical techniques based on the method used to obtain solutions [7]. These methods are discussed in the subsections which follow.

i. Numerical Techniques

Numerical techniques utilise multiple deterministic load flows to determine the probability that a certain load flow will occur on a feeder in the tested system [8]. Since the deterministic load flow provides a snapshot of the power system at a given instant, executing the load flow multiple times at different instants in a system containing uncertainty provides a wider range of possible results.

The main numerical technique used for probabilistic load flow is the Monte Carlo Simulation. Monte Carlo Simulations utilise a random selection of an input containing uncertainty to determine the resulting load flow solution. The random sampling and determination of results relating to that specific random input is repeated many times to obtain accurate results [7].

Due to the extensive number of simulations required to obtain accurate results, this method is relatively slow and requires significant computational processing power. Nevertheless, due to its accuracy, Monte Carlo Simulations are often used as a base case when testing other probabilistic load flow methods. The probabilistic load flow methods are deemed accurate if the load flow results obtained are similar to those obtained from Monte Carlo Simulations.

In essence, Monte Carlo Simulations repeat a deterministic load flow numerous times using different input values to account for uncertainty in power systems. Thus, the usual non-linear load flow equations are used in the calculation of results. The capability of the Monte Carlo Simulations to use the unaltered non-linear load flow equations allows very accurate results to be obtained [9].

ii. Analytical Techniques

Analytical probabilistic load flow techniques use mathematical equations to obtain load flow results from the system inputs. Analytical techniques are faster than numerical techniques such as Monte Carlo Simulations but require complex mathematical calculations. Furthermore, the accuracy of these methods vary significantly based on the approximation used [9].

Assumptions are often made to simplify the calculations used in analytical probabilistic load flow methods [9]. These include:

- Linearizing the non-linear load flow equations.
- Input variables at different nodes are independent.
- Normal distributions are commonly used for loads and discreet distributions are often used for generation.
- The configuration of the network and its parameters remain constant.

Due to the assumptions stated above, analytical techniques are not as accurate as numerical techniques. However, the computational time is greatly reduced. Many analytical techniques have been developed to increase the accuracy of the outputs whilst retaining the reduced computational time. Some of these methods are discussed below.

a) Multi-linearisation

As opposed to numerical probabilistic load flow methods, analytical methods often use linearisation techniques to reduce the computational time required to obtain load flow solutions. The major drawback with linearisation techniques is the transfer of errors from the input to the output. Whilst the technique only requires a single linearisation point to obtain values close to the expected value when the uncertainty in the loads is low, deviations of up to 100% of the expected value is obtained when the

uncertainty level is high [10]. A technique using multiple linearisation points may provide a solution to this problem, allowing for the forecasting of long-term loads which have a high level of uncertainty.

The multi-linearisation technique is a combination of the Monte Carlo Simulation technique and load flow equations which have been multi-linearised. The multi-linearisation technique presented in [10] determines when the system load meets a certain criterion to obtain various linearisation points. Since the boundary load flow technique is not used, the results obtained via this method are more accurate and efficient [10].

The method was tested in [10] using an IEEE 14-bus test system containing 14 busbars and 20 feeders. The simulation was repeated 5000 times to obtain the results discussed. The voltage and angle at bus 14 was assessed using three methods, namely single linearisation, multi-linearisation (five linearisations) and a probabilistic load flow method utilising non-linear load flow equations. The minimum, maximum, standard deviation and expected value was determined for each method. Furthermore, each method was assessed for low, medium and high levels of uncertainty.

The results indicate that the multi-linearisation technique is more accurate than the conventional linearisation technique, since the values obtained using the proposed method are closer to the expected values obtained from the non-linear PLF method. In a review of probabilistic load flow techniques [9], it has been noted that this method is simple yet efficient.

b) Quadratic Probabilistic Load Flow

Since analytical load flow methods often linearise non-linear load flow equations, a degree of error is introduced into the results. However, non-linearisation of the load flow equations results in complex mathematical equations requiring excessive computational time. Therefore, in probabilistic load flow, a choice must often be made between the speed offered by analytical techniques and the accuracy offered by numerical techniques.

In an attempt to resolve the respective negative aspects of analytical and numerical techniques, the lower order non-linear load flow terms are retained in the quadratic probabilistic load flow technique presented in [11], resulting in more accurate results being obtained. Furthermore, the higher order terms are linearised, resulting in less complex mathematical equations and hence, a reduction in computational time [11]. Thus, the method intends to combine the speed of analytical techniques and the accuracy of numerical techniques.

The method was tested on an IEEE 14-busbar test system and compared with the results obtained through Monte Carlo simulations. The results indicated that the absolute error obtained when calculating the expected value and variance of voltage and power was minimal [11]. However, significant deviations were obtained in the variance of active and reactive power flows on the assessed feeders. Thus, the method should not be used for the forecasting of long-term loads, since the probability density functions drawn from the variance and expected values may be misleading.

c) Point Estimate Method

The point estimate method utilises random values of the random variable to obtain the statistical moments of the output. These statistical moments are then used to determine a probability density function.

In [12], a two-point estimate method was developed. This method utilises two random values per iteration, resulting in two estimates of the solution. Weighting functions are then applied to the two

estimated solutions to account for the skewness of the probability density function. From this, the expected value and standard deviation of the probability density function can be obtained [12].

The method was tested on multiple IEEE test networks. Monte Carlo simulations with 5000 runs was used as a base case for comparison. The multi-linearisation technique presented in [10] was also used as a method for comparison. The results obtained from the multi-linearisation technique was for 3 linearisations and 5 linearisations, using 1000 samples.

The three performance indices used for comparison was the errors in power angle, bus voltage and line flows [12]. For the IEEE 6-bus test system, the results indicated that the performance of the proposed method is similar to those obtained using 5 linearisations and better than the results obtained using 3 linearisations. However, the total computational time for the two-point estimate method is 0.219s while the multi-linearisation technique using 5 linearisations requires 0.363s to obtain results. However, it was noted that the results from the two-point estimate method deviates noticeably from the MCS method when the probability is close to either 0.5 or 0.75. This deviation can be reduced using three-point or four-point estimate methods [12].

The proposed method was also compared to the MCS method on larger IEEE test systems. As expected, the execution time of the proposed method becomes exponentially faster than that of the MCS method with an increase in system size. Furthermore, it is also noted that the performance indices mentioned previously remains similar to those obtained on the 6-bus test system, indicating that the error does not increase with an increase in system size.

d) Least Square Method

A method which uses the Taylor Series expansion is presented in [13]. The method requires a solved load flow problem to determine the error deviation of outputs containing uncertainty. Thereafter, the standard deviation and variance can be determined.

The method uses a least square estimation to convert the deterministic problem to a stochastic one. The Taylor Series expansion is utilised to linearise the non-linear load flow equations. The input values are represented by any probability density function. Since the outputs are linear combinations of the inputs, the Central Limit Theorem is applied to express the outputs as normally distributed random variables [13]. From this distribution function, the expected value and variance can be obtained.

The least square estimation method was tested on the system utilised in [14]. The results in [14] were obtained for a conventional deterministic load flow run 50 times and was thus used as a base case. Six cases were considered for the proposed least square estimation method. These are as follows [13]:

1. Fixed voltages
2. Variable voltages
3. 5% error limitation in load and generation areas
4. 2% error limitation in load and generation areas
5. 5% error limitation in load and generation areas and 40% normally distributed error limitation of the average value for each quantity
6. 3% error limitation in total load area

Line flows, voltages and generation at buses were tested for all cases. It was noted that case 1 and 2 provided results which were very similar, although both sets of results contained a considerable margin of error when compared to the base case. The results for cases 3 to 6 produced large deviations from

the base case. Therefore, this method does not provide the accuracy required for network design planning.

e) Convolution Methods

Often, the output probability density functions of a probabilistic load flow problem are assumed to be normally distributed. However, this is not always the case. It is therefore necessary to determine if the Central Limit Theorem applies to the output probability density function. The evaluation of the output probability density function is performed by linearising the load flow equations around the expected value. Thereafter, a convolution method is utilised to transform the input values to output values [15].

In [15], it is noted that the Laplace Transform, which was initially used as a convolution technique to conduct probabilistic load flows, requires a significant amount of time to obtain reasonably accurate results. Therefore, a new method was presented, which utilises the Fast Fourier Transform. The convolution by Fast Fourier Transformation was tested on a 14-bus system comprising 12 normal distributions and 12 discrete distributions. Tests were also performed on a 32-bus system comprising 56 normal distributions and 7 discrete distributions [15]. The conventional method utilising the Laplace Transform was also tested on the same systems and its performance was compared to the proposed method.

The execution time for each system depends on the number of points utilised to represent the discrete probability density functions. The results indicate that the method utilising the Laplace Transform has a shorter execution time for a smaller number of points, without necessarily providing more accurate results. However, for discrete density functions represented by many points, the Fast Fourier Transform method requires much less computation time than the Laplace Transform method [15].

For the conventional Laplace Transformation method, the execution time increases exponentially as the number of representation points increase, whereas the proposed Fast Fourier Transform method retains a linear relationship between execution time and number of representation points. The crossover point, after which the proposed method is always faster than the conventional method, is at approximately 100 representation points [15]. Furthermore, it was found that the Fast Fourier Transform produced smaller deviations from the expected value in all instances.

f) Cumulants and Gram-Charlier Expansion Method

The convolution methods presented in the previous section provides reasonably accurate results, with the computational time dependent on the convolution method used. However, convolution calculations are complex and thus require significant computational effort [16].

The method in [16] uses Cumulants combined with Gram-Charlier expansion theory to avoid complex convolution calculations, thereby reducing the time and computational effort required to obtain probability density functions. Furthermore, only a single run is required to obtain the probability density function.

The proposed method was tested on a WSCC 179-bus test system. The results indicated that the proposed method compared well with the base case obtained using Monte Carlo Simulations, proving the precision of the method [16]. The major drawback of the method is that it provides inaccurate results when discrete functions are used.

g) Herman-Beta Method

The Herman-Beta method uses statistical moments of loads modelled as beta probability density functions to determine voltage probability density functions, from which a confidence level is assigned to obtain a single solution [17]. Thereafter, the difference between sending end voltage and the solution obtained will provide the voltage variation for the given network configuration and conditions.

Since the Herman-Beta method obtains a solution after only one iteration, the computational effort required to obtain solutions is greatly reduced compared to other methods [7]. Furthermore, extensive testing has proven that the Herman-Beta method is more accurate than existing methods which do not account for the variation of load magnitudes [17]. This combination of simplicity, speed and accuracy makes it an attractive method for determining solutions in power systems containing uncertainty.

In [18], the Herman-Beta method was tested against results obtained through Monte Carlo Simulations for two areas. The results from the Herman-Beta method were similar to the results obtained through Monte Carlo Simulations in both cases, especially when there were at least consumers 10 consumers in an area. However, it was found that relatively poor results are obtained when there are minimal consumers present [18].

Apart from being simple, fast and accurate, the Herman-Beta method has been proven to have the following advantages [19]:

- Ability to handle probability density functions which are negatively or positively skewed.
- Ability to account for neutral resistance and unbalanced feeder loading.
- Ability to model power system configurations consisting of one, two or three phases and allows for easy comparison between various configurations.

In [19], using the Herman-Beta's phase allocation capability to shift some connections, a lower voltage drop was achieved across the radial feeder. This functionality could allow for the use of smaller feeders on radial systems [19]. This is an important feature of the Herman-Beta algorithm that may mitigate the overdesign of power systems.

The Herman-Beta method has been developed over many years and has been tested extensively. The original method contained underlying assumptions and was only capable of determining the performance of passive radial feeders. Thereafter, the Herman-Beta method was modified to include the effect of distributed generation in radial systems, whilst retaining the underlying assumptions. This modified HB method was later incorporated into a network planning tool using the C# programming language.

More recently, an approximation method was developed, which extended the approach to MV radial networks by removing the underlying assumptions of loads operating at unity power factor and feeders which are purely resistive. Although the approximation method provided more accurate results, a degree of error still existed [2].

Finally, the HBET was developed, which reformulated the algorithm's equations by removing all assumptions applied in previous versions of the method. In addition to this, the method was modified even further to account for the representation of loads as various models, shunt capacitance and correlation, among others. The HBET was developed into a tool using MATLAB programming language and tested thoroughly, where extremely accurate results were obtained.

2.2 Programming Languages

Since the HBET will be implemented in a software package, it is necessary to determine the optimal programming language to use based on ease of accessibility, complexity, functionality and the ability of the software to interface with existing software simulation tools often used by network planners, such as DIgSILENT PowerFactory. This section investigates the different software packages available and assesses each based on the criteria stated above.

2.2.1 MATLAB

MATLAB is ideal for users wanting to analyse data, create algorithms and develop models to simplify processes and easily compute complex formulations. The program uses its own programming language and is ideal for dealing with arrays and matrices. All toolboxes in MATLAB are well-documented and have been tested extensively to ensure optimal functionality [20].

MATLAB software contains interactive applications which allows the user to test the compatibility of various algorithms with their data. Iterations can be run until the required results are obtained. Once this is done, a MATLAB program can be generated to enable automation of a process [20].

The integration with programming languages such as C and C++ is easy with MATLAB, since the program has the ability to automatically convert algorithms from MATLAB code to the desired programming language. Furthermore, MATLAB utilises Simulink to enable the user to easily model algorithms using predefined models. Finally, the MATLAB programming language is known to be easily understandable and quick to learn.

The major drawback with using MATLAB to create the Herman-Beta tool is that it is not easily accessible. MATLAB is only readily available to students from certain higher education institutions. It is available for commercial and industrial purchase at a tremendous cost. Thus, not all network planners will be able to use the created program if it is implemented using MATLAB software.

2.2.2 JavaScript

JavaScript has been specially designed for the scripting of web pages. It usually forms part of a browser or a plug-in. JavaScript is ideal for writing light web scripts for both client and server sides. However, there is very little use for JavaScript outside web programming. It should also be noted that JavaScript and Java are completely different languages with different syntax [21].

JavaScript is often the number one choice for software developers. It was voted as the number one programming language among developers for six consecutive years [22]. Certain major social media platforms utilise JavaScript since it provides the ability to easily develop interactive web pages. JavaScript is compatible with all major browsers, making it even more attractive. It is often used to develop front-ends of programs [22].

2.2.3 Java

Java is an object-oriented programming language that is related to the C and C++ languages, but not compatible with them. It should be noted that Java is considerably easier to learn that C++. Java is similar to JavaScript in that it is ideal for web programming [21]. Java is often used to write small web applications.

Java can be used on basically all devices, making it an easily accessible programming language. It is often used for the creation of back-end applications. Furthermore, Java automatically allocates memory and uses garbage collection for optimal memory management [22].

2.2.4 Visual Basic .NET

Microsoft Visual Basic .NET is generally used for web and local applications. Visual Basic. NET allows for the compilation of Visual Basic language to the .NET language. Therefore, programmers can use Microsoft Visual Basic to develop .NET applications [23].

Although applications using Visual Basic .NET can be developed with similar functionality to applications developed in other programming languages, it is not very popular. Software developers tend to prefer C, C++ and C# for business applications. Most applications developed using Visual Basic .NET are older applications that are likely to be redeveloped using modern programming languages [23].

2.2.5 C#

C#, pronounced C-Sharp, was developed by Microsoft. It is essentially the Java language that has been modified to utilise the .NET framework [21]. C# is used to develop desktop applications and needs a .NET framework to operate [22].

C# is relatively easy to learn for new programmers. Error-checking is made easy since the code is checked for errors before converting it into an application. The C# programming language is perfect for the development of web and desktop applications. It has also been useful in the development of Virtual Reality applications, as well as 2D and 3D games [22]. The major drawback with the C# programming language is that it is only supported by Microsoft [21].

2.2.6 C and C++

C and C++ are very similar in function and are both extremely popular. C is however, much simpler to learn than C++ since the syntax used in C++ can be difficult to grasp [21]. The major advantage of these programming languages is that it can be freely downloaded from the web. However, C and C++ cannot be used to create web applications. The code can also be very difficult to understand [21].

Since these two languages facilitate fast execution of code, they are frequently utilised in applications requiring significant computational power. These two languages are often preferred when creating games and virtual reality applications [22].

2.2.7 Python

Python is a simple programming language to learn. Code written by other programmers is easily understandable, even when comments have not been provided as part of the code. Python uses indentation to easily identify portions of code that are run together [21].

Another useful feature of Python is the capability of using wheels to develop programs. Wheels are portions of pre-written code used to perform powerful functions, such as the manipulation of data and plotting of graphs. Additionally, Python has a built-in library specifically for the creation of easily customisable GUIs.

Python has the capability to interface with various existing programs, including software often used by utilities, such as DIgSILENT PowerFactory and OpenDSS. It also contains many built-in libraries and functions which makes it a simple yet powerful tool [22]. The popularity of the program will make

finding solutions to problems encountered while programming an easier task, since many online forums exist which help users find solutions to problems they are having with their codes.

2.3 Information Available to Network Planners in the Design Stage

Network designers try to increase the accuracy of their designs by obtaining as much information about the system they are designing and the customers to be supplied. This section explores the information readily available to network designers at the design stage.

2.3.1 Equipment specifications

When designing projects, network planners are required to consider the continuous and short circuit ratings of various equipment, including but not limited to transformers, cables, overhead lines, busbars, circuit breakers and surge arrestors. A database of all equipment installed is usually kept, which can be used to verify the size, type and date of manufacture of the equipment. The date is often captured for maintenance requirements. Therefore network designers usually know the age, type and size of equipment contained within the existing network.

On certain equipment, such as transformers and circuit breakers, a nameplate is installed, which indicates all the information required by network planners. These nameplates are extremely important for planning purposes. Therefore, a hard copy is often kept on file with the equipment drawings. Typical information found on nameplates include the serial number, manufacturer, date of manufacture, rated voltage and current as well as short circuit current rating.

Manufacturers generally have datasheets available on their websites. Thus, if the size and type of equipment is known, the datasheet can be used to easily find all required information. The equipment specifications, specifically for feeders in this case, will be included when selecting feeder types in the designed program. The rated current is of particular interest, since this will indicate whether the cable types to be used for a proposed project is adequately rated to supply the anticipated load.

In investigating the impact of rooftop PV installations on the network in [24], the author stated the cable length, type, impedance per kilometre and rated current of each feeder. This information was used at the design stage and was thus readily available to the network planner. Furthermore, [17] states that one of the design parameters is cable characteristics such as resistance, reactance and rated current. Although the values of these parameters will depend on the design, the actual specifications are readily available to the network planner, often via datasheets.

2.3.2 Site Development Plan

When applying for power supply from municipalities, developers are urged to submit a site development plan to help network planners assess the impact the new development will have on their network. Thus, the site development plan is usually available to network planners at the design stage.

The site development plan indicates the commercial, industrial and residential components of the property and where they will be located [25]. Roads, public open space and institutions, such as schools, are also indicated.

Municipalities often have standards in place indicating the ADMD to be used for certain categories of consumers. The values for different types of residential units, schools, churches, commercial, light/heavy industrial and any other consumer types are generally stated in kVA or VA/m^2. These standard values are multiplied by either the total area of the consumer category or the total number of

units of a certain category, depending on what the standard dictates. The summation of load demands for the various consumer types will provide the planner with the total load demand required by the development.

Once the total load demand is calculated, the network planner can investigate the number of feeders required and the manner of connecting the development to the existing network. Once the network planner has designed the portion of the network which will feed the development, a replica of the system will be modelled in the created program. At this stage, it will be possible to assign the equipment types to the respective parts of the modelled network.

A design flow chart of the design process is shown in Figure **2.1** below.

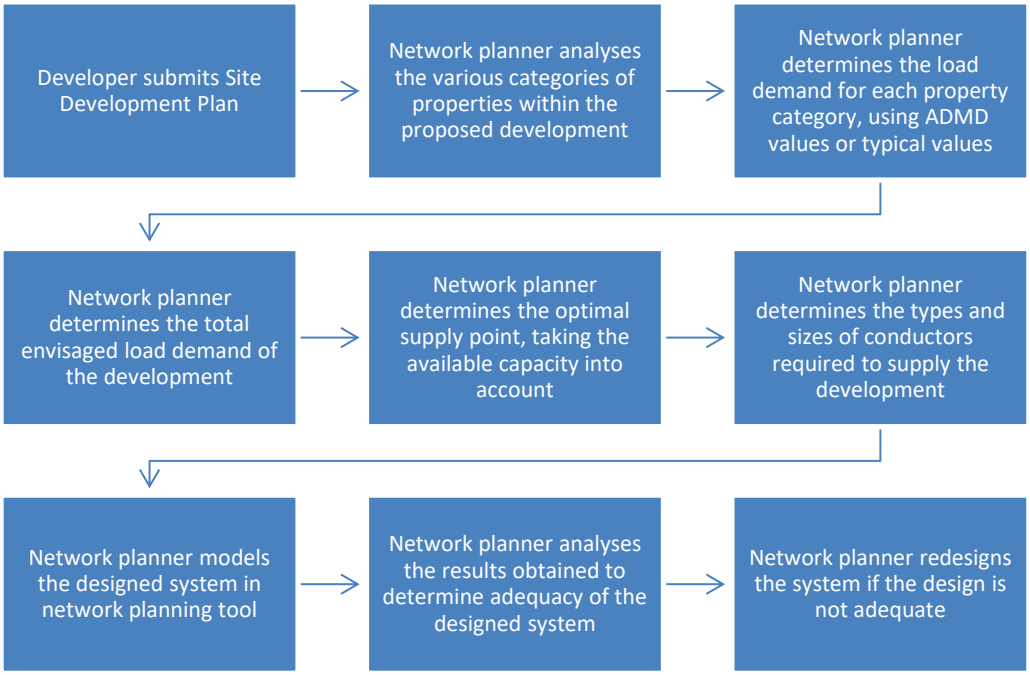


Figure 2.1: **Network Planning Design Flow Chart**

In most instances, Supervisory Control and Data Acquisition (SCADA) services allow for the recording of measurands at certain portions of the network, especially at the MV level and upstream. The measurands typically recorded are active power (MW), reactive power (MVar), current (Amps) and voltage. Power factor (PF) and apparent power (MVA) is then calculated from the recorded measurands.

The load data is generally sampled at short intervals of a few seconds. From these samples, average load data over certain periods are calculated. In South Africa, the City of Cape Town and Eskom utilise 30-minute average load data for planning purposes. For network upgrades and analysis of existing connections, annual load profiles are used.

The load data is extremely helpful when network upgrades are required for existing connections. However, for new developments, load data is not available. If, on analysis of the site development plan, it is found that the mix of consumers proposed is similar to the mix of consumers at an existing portion of the network, the existing network portion's load data may be used and scaled to a new peak demand to provide a predicted load profile. This method provides a more realistic scenario of what is expected when the new development materialises, since utilising a single average load value throughout the year is unrealistic.

In the reliability analysis in [26], the actual load interrupted is determined by comparing four different load models. All four load models utilise the original load data in some way. Thus, the load data must have been readily available to the network planner initially.

To determine the parameters of the probability density function in probabilistic load flow procedures, load data of the consumers are required [17]. Since probabilistic load flow forms the basis of this research, it is assumed that load data will always be available for the network being tested, either directly from measured data or indirectly by manipulation of existing data.

2.3.3 Load Forecasts

In addition to load data, load forecasts are also typically available to network planners. The load forecasts are particularly useful for the analysis of the network in future years. Load forecasts can be obtained by applying a percentage of load growth per annum. This is, however, a very simplistic method and is not the recommended approach.

Another method of obtaining a future load value is by utilising the load density (VA/m^2). The existing supply area can be multiplied by the higher load density of a similar, more energy intensive area, to obtain an estimated future load demand. It should be noted that this only provides the end state load demand. Hence, the load demand for each year in the future cannot be obtained, unless a linear or exponential relationship is assumed to link the existing and forecasted load demand. Furthermore, the year in which the end state occurs has to be estimated, introducing even more uncertainty in the load forecast.

Load forecasts may also be drawn by utilising factors such as proposed development and electrification load demands, shifts in load between substations, expected natural load growth per annum and changes in the source substation. Although this method requires much more information as inputs compared to the other two methods discussed, it provides more accurate load forecasts.

In addition to the forecasting methods described, various other methods have been developed, such as load forecasting using fuzzy logic or artificial intelligence. Irrespective of the method used for load forecasting, it plays a crucial role in network planning. Due to the importance of load forecasting, it is assumed that network planners will have this information available in instances where the network needs to be analysed for future periods.

2.4 Dealing with Uncertainty

Since the loads of customers vary with time, it is often difficult to predict the overall performance of a power system for extended periods of time, especially during the design stage where a load profile is not available. The problem is simplified slightly when past load data is available, wherefrom trends can be drawn to determine the predicted load value and hence, the corresponding system performance. The uncertainty associated with loads is often not considered, since most conventional design processes are deterministic [17].

Deterministic methods often utilise the average value of loads to account for uncertainty. The average value is obtained by considering a large group of consumers within specific consumer classes [17]. At the time the system peak occurs, the average load demand is determined by dividing the system peak by the number of consumers in that consumer class.

Section 2.3 discusses the possibility of using load data and forecasts to account for uncertainty. Although trends can be drawn from previous load data and load forecasts can be obtained by using a variety of inputs to obtain a more realistic estimation, the analysis of the results obtained are based on the network planner's opinion of what will happen in the future.

The uncertainty associated with the increased penetration of renewable energy and the uptake of electric vehicles [8] can be taken into account by network planners using popular network planning programs such as DIgSILENT PowerFactory and ReticMaster. These programs have taken cognisance of the importance of taking uncertainty into account when designing systems and have incorporated PLF analysis in their toolboxes.

2.5 Information Required by Network Planners

The developed program will utilise the Herman-Beta Extended Transform to calculate voltage variations on radial networks created by the user. This section looks at the input information required by the program to determine the necessary outputs. Furthermore, the output information that will be useful to network planners and the results network planners require to make informed decisions are discussed.

The Herman-Beta Extended Transform, as well as the original Herman-Beta method, models input loads as probability density functions and determines output voltages along radial feeders as probability density functions [7]. The voltage variation at a user-defined confidence level is determined by calculating the difference between the source voltage and the consumer voltage.

To determine the required outputs from the HBET program, the user needs to specify the loads and/or generators per phase, cable type, cable length, cable operating temperature, Beta PDF parameters (α and β), circuit breaker rating and power factor. It should be noted that the selectable cable types would vary depending on whether a LV or MV system is being designed.

Since the voltage variation changes at each node, it would be useful for the network planner to see the voltage per phase at each node. This can be represented graphically, with the calculated values per node shown against the minimum and maximum voltage drop limits, as in [3]. Each phase will have their own plot on the graph, since the voltage variation across each phase will differ for feeders with unbalanced phase loading.

2.6 Application Development

To determine the optimal approach in developing a user-friendly network planning tool, previously developed applications were consulted and analysed to identify key elements and drawbacks. Additionally, a guideline document for software development was consulted. The relevant information extracted from these sources are presented below.

2.6.1 Previously Developed Tools

In this section, three previously developed network planning tools are investigated and analysed to determine the approaches used to develop these programs. The features, functionality, programming language and graphical user interface of each program is discussed.

i. Tool to Calculate Induced Current and Voltage on PV Systems Due to Lightning Strikes

In [27], a network analysis tool was developed to determine the current and voltage induced on PV systems by the electromagnetic field emitted by lightning strikes. The software tool was programmed

using Microsoft Visual Basic. The program considered both direct and nearby lightning strikes to recommend lightning arresters with adequate ratings to protect the PV systems [27].

The developed software tool utilises a graphical user interface to accept inputs from the user, after which a 'Calculate' button needs to be pressed to determine the induced voltages and currents. The graphical user interface contains radio buttons, push buttons, dropdown lists and text boxes requiring input information. There are also various tabs which the user can switch between. Although the GUI and button types used promotes ease of use, the GUI seems cluttered, since the blank results sheet forms part of the main window. The main window would not be as cluttered if only the input parameters were displayed. The use of a pop-up window to display results after the 'Calculate' button has been pressed may have been a wiser choice. The program main window is indicated in Figure 2.2 below.

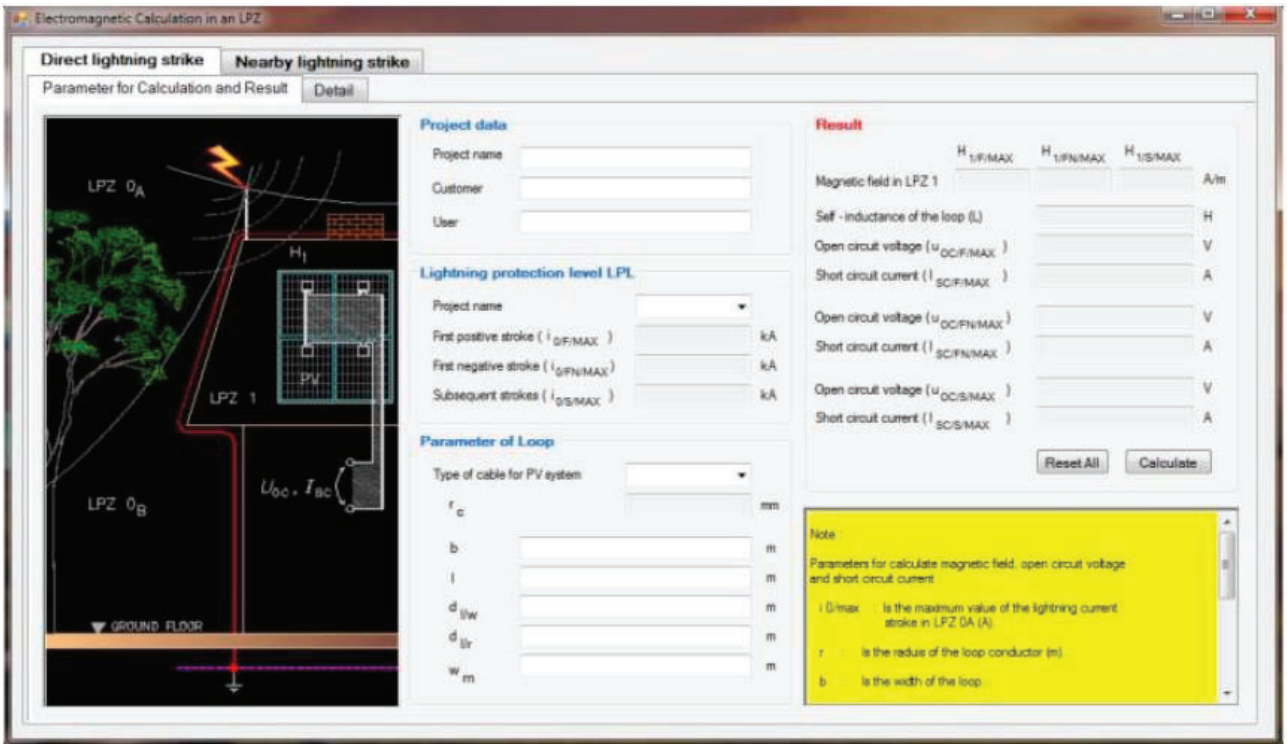


Figure 2.2: Example of Previously Developed GUI [27]

The graphical user interface contains an image of either the direct or nearby lightning strike scenario, depending on the tab selected. These images indicate where some parameters required as inputs are located physically. There is also a 'Detail' tab, which provides the user with the theory behind the calculations for each scenario. Various colours are used in the graphical user interface for heading text and as backgrounds for important information.

The results provided by the program are given in text format in mini text boxes. The parameter description, its value and unit of measurement are given. Some of the results calculated include the magnetic field induced, induced voltage and induced current. The same inputs used in the developed program were used to calculate the parameters using the conventional calculation method. The results obtained from the developed program were almost identical to those obtained via the conventional calculation method [27], confirming that the calculations were correctly translated into code.

The program appears to contain two major setbacks in addition to the cluttered main window. Firstly, the program does not contain a menu tab. Most users would appreciate the presence of a menu tab, since it is utilised in most major programs and would instil a sense of familiarity. Secondly, it appears that the

user cannot save the results obtained from a designed scenario. The ability to save scenarios or plots is vital to users, especially when the results are to be utilised in reports.

ii. Tool for Calculation of Voltage Drops and Selection of Cables for Building Installations

The development of a planning tool is presented in [28], which calculates voltage drops and provides optimal cable sizes for designs of building installations. The purpose of the developed program is to replace the tedious task planners face, which includes determining the required current to be supplied, selecting the appropriate cable with the correct current rating and determining the voltage drop along the feeder [28].

The software tool was implemented using Microsoft Visual Basic and can thus be run on any Microsoft Windows operating system. Microsoft Visual Basic was chosen since it allows for the creation of powerful applications which can utilise the main features of Microsoft Windows [28]. Furthermore, applications created using Microsoft Visual Basic are .exe files which can easily be distributed.

In the developed program, the user has the ability to have multiple design files open simultaneously. The home screen has a main menu similar to those seen in Microsoft applications such as Word and Excel, with each selectable drop-down menu having sub-menus. A help menu is also available, which describes the functions in the program [28].

The program contains a Data Input interface which allows the user to define their problem by a selection of variables. Some of the inputs required are power factor, load type, length and type of cable, number of cable cores, number of cables, installation method, voltage rating and temperature. Once all the data has been inserted, the 'Continue' button should be pressed.

The program assesses the system created based on the cable type and size inserted by the user. The voltage drop is then calculated. If the voltage drop calculated exceeds the predefined limit of 2.5%, the next larger cable of the same conductor type is assessed to check if the voltage drop adheres to the 2.5% limit. If not, the process is repeated until a suitably sized cable is recommended which provides a voltage drop lower than 2.5%.

The predefined voltage drop limit of 2.5% is one of the only drawbacks exhibited by this program. This could easily have been overcome by allowing the user to specify the voltage drop limit in the program's main window and performing the cable selection process using the user-selected voltage drop value.

Since the installation method is a required input, an interface exists which provides the user with a graphical representation of the twenty installation methods selectable [28]. This ensures that the user selects the correct installation method. The interface also contains a description of the installation method alongside the graphical representation.

The results provided by the program indicates a summary of the user inputs as well as the recommended cable size required for the installation of each feeder, based on the information received from the user. The cable size is determined from a different lookup table in the background, based on the current rating and the derating factors applicable. Although the results were not compared to any other method to assess the accuracy of the program, the program is currently being used by the Sewerage Department in Singapore for the determination of optimal cable sizes for building installations [28].

iii. Voltage Variation Calculation Tool Using Herman-Beta Method for Passive and Active Systems

The original Herman-Beta spreadsheet was developed in the mid-1990s and allowed for the analysis of passive radial feeders. In 2010, the spreadsheet was modified to facilitate the connection of distributed generators. This modified spreadsheet formed the basis of the voltage variation calculation tool developed in [3].

The program was implemented using the C# (C-Sharp) programming language and was developed as a freely available stand-alone program. The major contribution of this tool was that it allowed for the analysis of branched feeders and large networks, which was previously not catered for.

The developed program utilises a graphical user interface to allow customised designs from the user. The program contains libraries wherefrom equipment types are selected to build a radial LV feeder. Some of the inputs which the user needs to insert are source voltage, length of feeders, types of feeders, type of loads, temperature, design risk, customer voltage constraints and whether generators are present at nodes [3].

The program developer states that a graphical user interface was utilised instead of a command line interface since it is easier to use [3]. The approach of having users drag and drop elements to form radial feeders was decided against, mainly since the user would be limited to the space available in the network creation window [3]. Therefore, the program uses simple text boxes for the user's inputs. Although the use of text boxes is a better alternative to the drag and drop system, each text box needs information to be inserted individually. This becomes tedious for large networks, since network parameters are often similar across various nodes.

On the main screen of the program, the user is required to describe the radial feeder. Two radio buttons are used to choose between an active feeder and a passive feeder. The source voltage and temperature values can be incremented or decremented using spin buttons, which increase or decrease the value by 1 unit when the up or down arrow is pressed, respectively. The user names the node and enters the required information, described in the second paragraph in this section. Loads, generators and cables are selected from three built-in libraries, each with the option of adding to the existing library.

Immediately below the data input table, the main feeder topology is constructed as the user adds nodes to the network. This provides the user with a graphical representation of the information entered. The program uses parent and child relationships to draw the main feeder topology [3]. Once all information has been inserted, the user clicks on the "Proceed to Voltage Drop Calculation" button to obtain the results for the designed network.

The output window provides a graphical window of the feeder voltage for the blue, white and red phases. The minimum and maximum allowable feeder voltage is shown for the user to easily detect if the limits have been exceeded. The cable type, length and summary of the nodes in the feeder are also provided in summary form next to the graph. The user has the option to change the source voltage, temperature and design risk percentage directly from this window and can view the revised results by clicking the "Update" button. The edits done from this results window can be discarded or saved. If the edits are saved, the entries in the original data input sheet will be changed to the edited values [3].

The developer concluded by stating that the program could not be compared to other simulation packages, since most LV feeder voltage drop calculation programs rely on deterministic methods instead

of probabilistic methods. However, it was noted that the developed program is a user-friendly tool for the calculation of voltage variations on radial LV feeder systems and that the algorithms have been thoroughly tested [3].

2.6.2 Existing Network Performance Tools

When developing a network planning tool, it is worth looking at existing popular network planning tools for guidance related to program layout and functionality. In [29], four popular power system applications are assessed. It was discovered that the main features which makes these tools popular are highly intuitive graphical user interfaces, libraries containing typical equipment parameters/ratings, the capability of analysing large networks and the representation of outputs graphically, which allows non-tehnical audiences to understand the results obtained.

Further discussions of power system analysis programs are presented in [30], where 23 software tools were analysed. It is stated that power system analysis tools have the following features which make it user-friendly:

- A facility to design the user's system.
- A graphical user interface.
- Reporting of results graphically and/or in tabular format.
- Using appropriate colour combinations.
- The capability of being used on any machine regardless of the operating system being used.
- The ability to import/export data.
- A help function/user guide document.
- Third-party program compatibility.

There are also a few popular network planning tools that have the capability of performing probabilistic analysis, such as DIgSILENT PowerFactory and ReticMaster. It is worthwhile investigating these established software tools to obtain greater insight into what makes these tools useful and user-friendly.

i. DIgSILENT PowerFactory

DigSILENT PowerFactory is a powerful software tool used by utilities and consulting firms worldwide. One of the major features of the software is its ability to quickly and accurately perform load flow simulations. This feature was expanded on in 2018 with the creation of the Probabilistic Analysis module, which provides the capability of determining probabilistic load flows of networks.

In general, DIgSILENT PowerFactory has the following features that make it user-friendly and useful [31]:

- A GUI which allows the user to easily select components and build their network.
- A menubar catergorising tools and user options.
- A user manual and tutorials to assist the user, if required.
- The capability to import files from other popular simulation tools.
- The ability to save simulation files.
- The option to represent results graphically or in tables.

The Probabilistic Analysis module is presented in [32], where it is shown that the same features described above have been incorporated. Additionally, the module has the capability of receiving inputs via spreadsheets and represents the stochastic results graphically, factors which will strongly be considered in the development of the HBE-Python tool.

ii. ReticMaster

ReticMaster is a popular network performance analysis tool capable of performing load flow analysis, fault level studies and voltage analysis, among others. Additionally, ReticMaster contains all the features DIgSILENT PowerFactory utilises to ensure the software tool is user-friendly and useful [33].

The user interface is indicated in Figure 2.3. As with DIgSILENT PowerFactory, icons, a menubar, a help menu and a blank canvas to model networks are included.

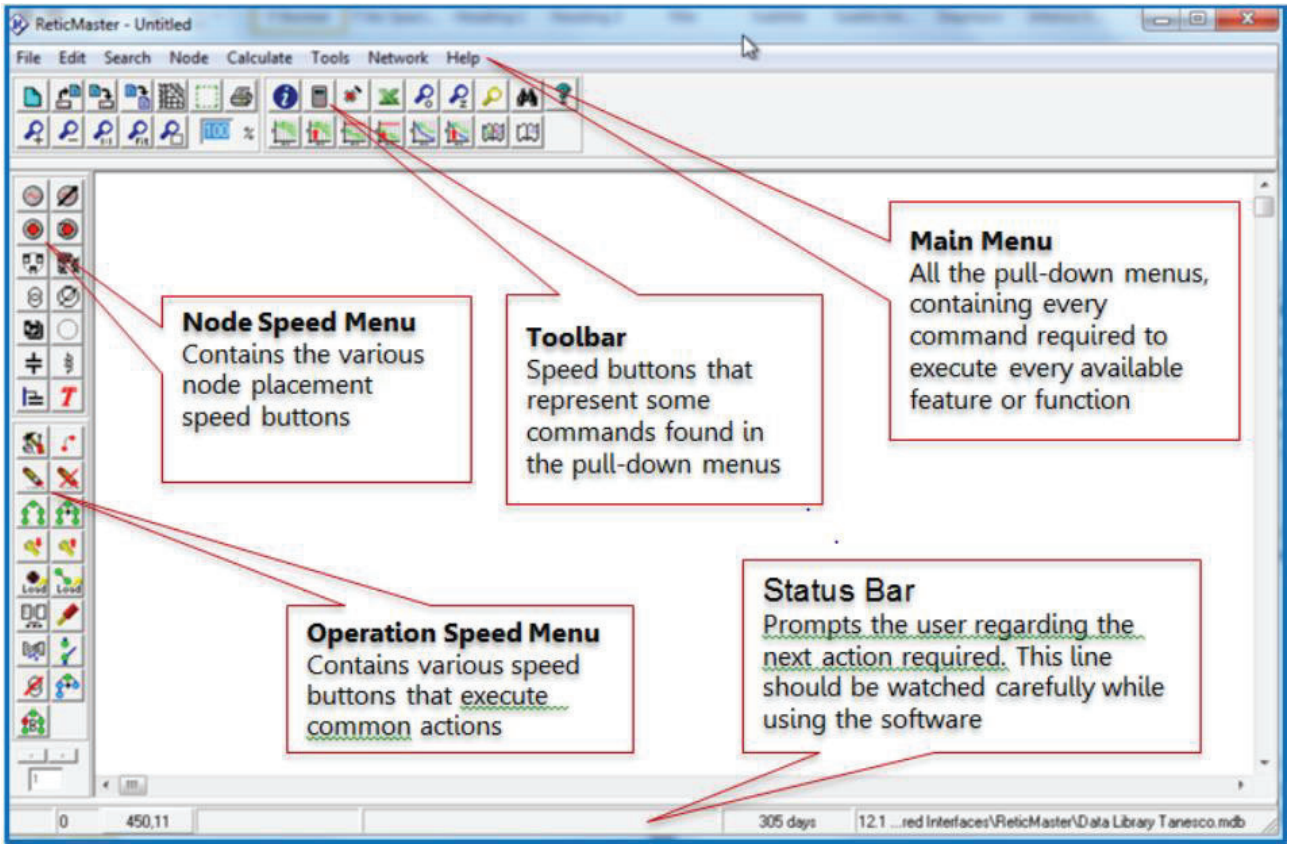


Figure 2.3: ReticMaster User Interface [33]

2.6.3 Guidelines and Considerations

In recent years, the majority of programs have adopted interfaces which are focused on graphical user interfaces and visuals. Generally, users are more likely to utilise their mouse to navigate and use a program as opposed to using keyboard commands. This is further iterated in [34], where it was noted that the greatest disadvantage of the OpenDSS tool is the lack of a user interface and the requirement for users to model network elements using scripts. It has also been noted that the purpose of utilising software is to simplify and accelerate tedious processes, not to replace human knowledge [28].

A telecommunications network planning tool guideline document was consulted to gain the knowledge required to develop the proposed network planning tool. This information can be adapted to power systems, since network planning for these two types of systems are similar. Only the guidelines applicable to the proposed network planning tool will be provided. To ensure the development of a successful network planning tool, the following guidelines should be followed [35]:

- The tool should have the capability of modelling systems containing multiple cable sizes and should have the relevant technical data linked to respective cables.
- The developed program must allow for information to be saved by the user, either as a simulation file or as an image for use in reports and presentations.

- The user should be able to alter the names of output files and should have the ability to simulate various scenarios.
- Documentation should be provided, either within the program via a help function or as additional files, which explains the program features, inputs required and the results that will be obtained once the program is run. A user guide should also be included to describe the processes that need to be followed when using the tool, as well as the events which trigger pre-defined error messages.
- The tool should be validated using a range of tests to ensure the tool operates as expected and provides the same results obtained via conventional methods.

In addition to the abovementioned guidelines, the following general methodology can be followed to develop the interface for a new tool [36]:

- Analyse the problem which needs to be addressed by the tool and determine the objectives of the program, as well as the user requirements.
- Determine which functions will have to be used to obtain the required outputs.
- Consider the general layout of the GUI and the associated buttons, text boxes and widgets which need to be included.
- Determine which features should be on the main window and which features, if any, should be on secondary windows (sub-functions).
- Assess factors such as brightness, contrast and colours used and ensure that optimal levels are selected to create a user-friendly interface.
- Allow for a group of users to provide feedback on the developed tool. Alterations should be made based on the feedback received.

2.7 Chapter Summary

Based on the literature review conducted in this chapter, the following key points can be deduced:

- As discussed in section 2.1, load flow techniques are broadly classified into two main categories, namely deterministic load flow and probabilistic load flow techniques. Probabilistic load flow techniques are further classified into analytical and numerical techniques. Probabilistic load flow methods account for uncertainties in power system such as varying loads and the intermittence of distributed generation, whereas deterministic methods do not.
- Monte Carlo simulations is the main numerical technique utilised to assess power systems containing uncertainty, as outlined in section 2.1.2. The main drawback with Monte Carlo simulations is that it only approaches an accurate answer after an extensive number of runs which increases the computational time significantly. Nevertheless, it is often used as a base case to assess other probabilistic load flow techniques, due to the high level of accuracy achievable.
- As determined in [7], the Herman-Beta method produces accurate, repeatable results at a relatively fast speed, compared to other existing methods. Since it has been adopted as a national standard in South Africa and is widely used, the adoption of the recently developed HBET will be much higher than a completely new method. The HBET is even more powerful than the previous versions of the HB method. However, it is not available to most network planners as it was developed using MATLAB. For this reason, the HBET needs to be deployed using a more accessible platform.
- As reviewed in section 2.2, the programming language which is the simplest to understand and use that provides the ability to develop powerful applications is Python. Python can interface with many existing network planning tools and contains built-in libraries and modules which could prove useful to the development of the tool. Additionally, Python wheels will allow for the manipulation of data and plotting of results using pre-defined pieces of code, instead of

programming these functions from scratch. Furthermore, a plethora of online forums exists, which can be used to obtain solutions to problems encountered while developing the tool, if the need arises. For these reasons, Python will be used to develop the HBE-Python tool.

- The information available to network planners at the design stage is discussed in section 2.3. This includes equipment specifications, site development plans, load data and load forecasts. This information is used to determine how much power needs to be supplied and which cables should be installed to feed the consumers.
- As discussed in section 2.4, network planners deal with uncertainty either by utilising standards which have been determined using the Herman-Beta method for LV feeders or by using some form of probabilistic load flow.
- Section 2.5 emphasises the importance of knowing the expected current flow and voltage drop along a proposed feeder. Network planners need to know what current is expected to flow through cables in their networks to determine the optimal sizes and types of cables that should be installed. Once this is determined, network planners are interested in the voltage variation experienced along their networks, since limits have to be adhered to at the consumer's terminals to ensure the expected quality of supply is achieved.
- Graphical user interfaces are often used to accept user inputs and produce outputs, since it is more engaging and intuitive than using command line functions.
- Previously developed software tools and a software development guideline were analysed to determine the key elements required in the HBE-Python tool. In addition to this, drawbacks of these programs were identified, which will not be utilised in the developed program. The following main features were identified:
  - The program should make use of a GUI instead of requiring the user to run the program from the command prompt [3]. The GUI should utilise space effectively and should not appear cluttered. To avoid cluttering the main window, results can be presented to the user via a pop-up window once the relevant button has been clicked.
  - In the event that the user is required to insert inputs on the program main window, text boxes, spin boxes and drop-down menus should be utilised,
  - Since most programs utilise a menubar, it would be useful to incorporate this in the HBE-Python tool. The menubar can be used to easily separate groups of tools/functions.
  - The user should be able to save plots/case files.
  - The program should contain equipment type libraries that can be customised by the user.
  - Results should be presented graphically or in tabular format, to allow for easy analysis and interpretation.
  - A spreadsheet input method should be utilised for node parameters, since having the user insert information in individual text boxes in the program main window becomes tedious, especially for larger systems.
  - The user should have the ability to revise their calculations simply by altering inputs on the main window.
  - A user guide should be included to detail the process of utilising the program.

3. Evolution of the Herman-Beta Method

This chapter elaborates on the Herman-Beta method by stating the assumptions used to develop the method and providing the equations used in its execution. The development of the method since its inception will be provided in chronological order, starting with the initial Herman-Beta algorithm for LV radial networks. Thereafter, the Herman-Beta algorithm for radial LV feeders with loads and DG is explored. Finally, the recently developed Herman-Beta Extended Transform, which extends the method to radial networks of any voltage with optional DG, is presented.

3.1 Introduction

Before the various development phases of the Herman-Beta method are discussed, it is necessary to introduce a few principles that were essential to the development of the method. These principles are presented in this section.

It is very important to note that work of three previous theory developments [2][4][17] for various versions of the Herman-Beta method was heavily drawn from in this section. This was done since the HBET equations needed to be translated to Python code, which required that the theory and modifications of each algorithm be fully understood and checked.

3.1.1 South African Load Research Project

In 1988, a load research project was initiated in South Africa [17]. The purpose of the project was to collect load data for multiple customer sites nationally. Load data was collected from over 20 sites, with at least 50 customers occupying each site [37].

The collection of load data was done via data loggers having the following main features [17]:

- Sampling interval of 5 minutes
- Resistance to insects and harsh weather conditions
- Compatibility with Microsoft Excel
- Recording of the time of data collection and the data logger it was collected from
- Embedded clock which allowed for the synchronous collection of data between loggers
- Inexpensive to implement

Analysis of the load data collected over many years indicated that modelling loads as currents is optimal for design purposes. It was also discovered that the power factor was almost unity when the peak demand occurred [37]. These findings have been crucial to the development of the Herman-Beta method.

3.1.2 Choice of Probability density function

Using the load data collected, histograms showing the distribution of load currents at the instant of peak demand were drawn. The histograms were drawn for customers from the low-income and middle-income groups [17]. To conduct probabilistic load flows, a probability density function that provides the best representation of the data in the histograms needs to be determined.

From the envelope of the histograms drawn, it was clear that the probability density function of the load currents should be asymmetrical, indicating the inadequacy of using the normal probability density function. Therefore, investigations were done to determine the optimal probability density function to

represent the distribution of load currents [17]. For various reasons, it was found that the Beta probability density function was the most suitable.

The reasons for using the Beta probability density function are as follows [17]:

- The α and β parameters of the Beta probability density function can be determined from the load data with ease.
- It has the ability to be skewed negatively or positively.
- It has a finite base ranging between 0 and 1, which can be scaled by the customer's circuit breaker rating to allow for the base to be extended to range between 0 and the circuit breaker current rating.
- The results obtained from 'Goodness of Fit' tests indicate that the Beta probability density function performs well by obtaining values close to the expected value.
- Easily incorporated into the calculation for voltage drops.

Due to the reasons stated above, the Herman-Beta method utilises the Beta probability density function to represent the information contained in the load data. The Beta probability density function has been used throughout the various modifications of the algorithm. The effectiveness of the Beta PDF has been verified by the accuracy of the results obtained when using the Herman-Beta method.

3.1.3 Summary of Herman-Beta Method Developments

The Herman-Beta method has undergone extensive development over the past few decades. The flow diagram in Figure 3.1 below depicts the development of the Herman-Beta method. A timeline is provided, along with the additional features incorporated at each stage of development.

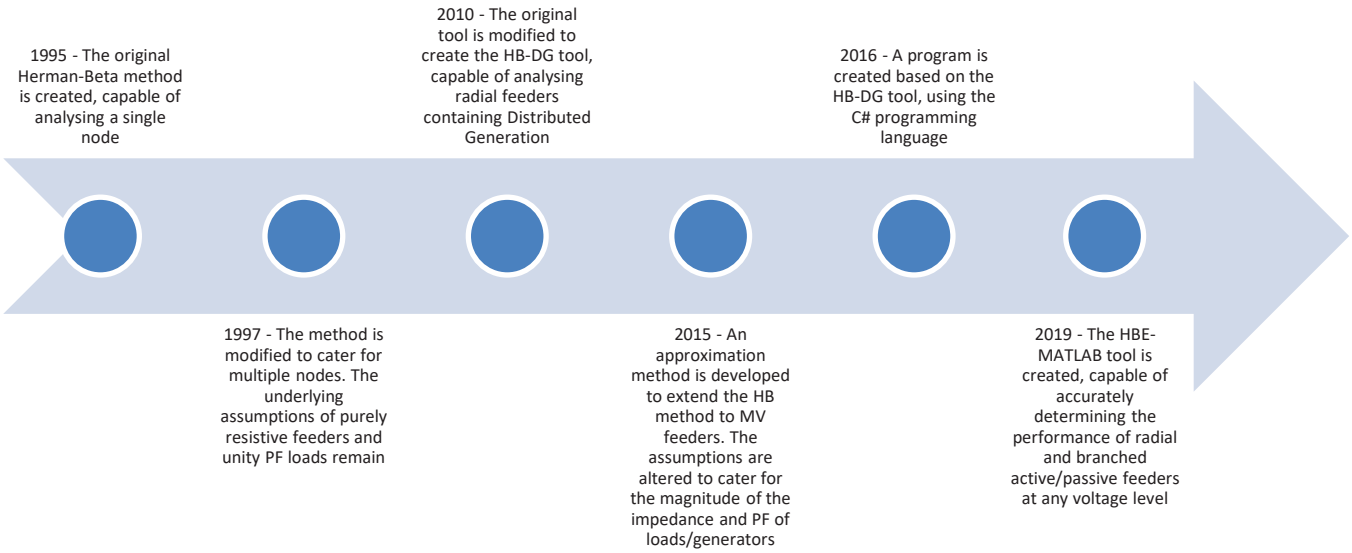


Figure 3.1: Timeline of Herman-Beta Method Developments

3.2 The Herman-Beta Method for LV Feeders

The Herman-Beta method was developed by Herman and Gaunt and is presented in [17]. The method presented allows for the design of LV radial networks using probabilistic load flow, specifically in

developing countries. Furthermore, [17] also presents a method which allows countries without an extensive collection of past load data to apply the Herman-Beta method. As stated previously, the Herman-Beta method has also been adopted as a South African National Standard [38] and is the preferred probabilistic voltage drop calculation method in South Africa.

3.2.1 Representation of Loads

In power system modelling, loads may be modelled as constant resistance, constant power or constant current. The load model chosen affects the collection of data as well as the analysis of the power system [17]. For the Herman-Beta method, loads are represented as constant current loads for the following reasons [17]:

- Residential consumer appliances are represented well using constant current loads.
- Constant current loads are not dependent on the voltage drop experienced across feeders.
- Measuring loads as currents is accurate and inexpensive.
- The representation of loads as constant currents is preferred over constant power loads, since constant power loads require voltage drop calculation methods which need to be iterated.
- The constant current load representation allows for similar behaviour to that observed from loads in reality, whereby a reduction in voltage results in a reduced load.
- Traditional voltage drop calculation methods for LV feeders that represented loads as constant power loads eventually converted the loads to currents at rated voltage. This step will therefore not be required if the loads are represented as constant current loads.

For the reasons provided above, all modifications of the Herman-Beta method represents loads as constant currents.

3.2.2 Assumptions

During the development of the Herman-Beta method, a number of assumptions were made. The assumptions made were as follows [17]:

- The voltage drop is at its maximum when the customer's load demand is at its maximum.
- The customer loads are assumed to operate at unity power factor. Readings taken by utilities have confirmed that the power factor of customer loads is unity at the time of maximum load demand, thereby justifying the assumption. Therefore, loads will be modelled as constant currents with a power factor of 1.
- The majority of customer loads requiring a significant amount of power are resistive.
- Customer loads can be represented by the Beta probability density function during any specific period.
- The impedance of LV feeders is regarded as purely resistive at specified temperatures, due to the negligible reactance component of these feeders.

These assumptions were used to produce the algorithm for the Herman-Beta method for LV feeders. The algorithm was developed for two systems, namely 3-phase, 4-wire systems and bi-phase systems. Bi-phase systems are not part of the scope of this dissertation and is therefore not be presented.

3.2.3 Algorithm for 3-Phase, 4-Wire Systems

The Herman-Beta method for LV feeders for 3-phase, 4-wire systems is presented in this section. Before the method can be presented, a model of this type of system needs to be drawn. This model was used to determine the equations used in the algorithm. The 3-phase, 4-wire system is modelled as in Figure 3.2. The algorithm and theory development was adopted from [17].

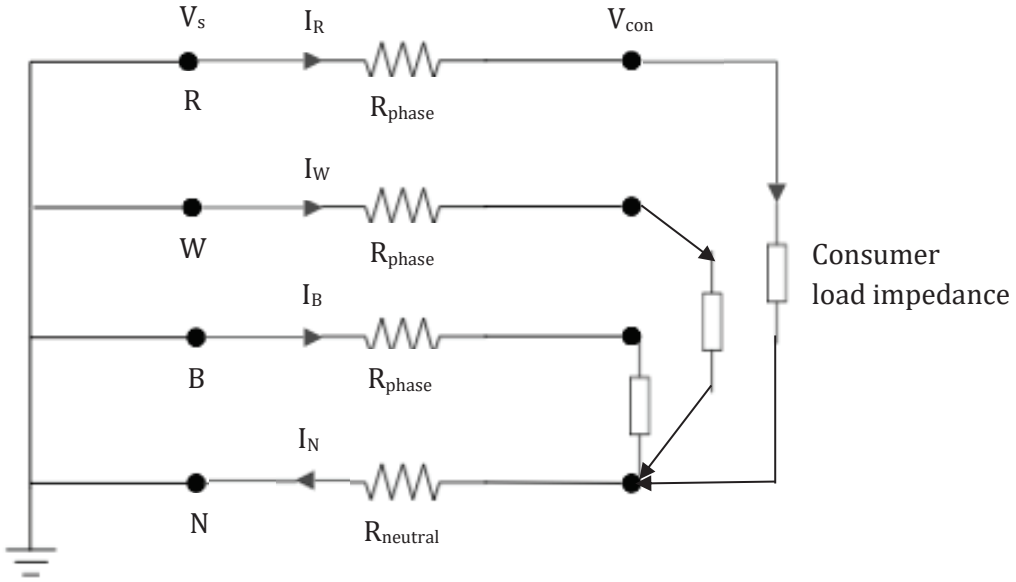


Figure 3.2: Model for 3-Phase, 4-Wire LV System

In the model, the nodes are labelled according to the phases in 3-phase systems, namely red (R), white (W) and blue (B) phase. All three phases are connected to a common neutral (N). The source voltage is denoted by V_s and the consumer voltage is denoted by V_{con}. The phase resistance (R_{phase}) is due to the resistance of the feeders used to connect the consumer loads to the source. As mentioned in the assumptions in Section 3.2.2, the feeder reactance is negligible and was thus ignored in the development of this algorithm.

With reference to Figure 3.2, assume there are n consumers at each node with Beta distributed loads. The number of consumers for the red, white and blue phase will thus be denoted by n_R, n_W and n_B respectively. The probabilistic load current per phase can then be expressed by the equations below.

$$I_R = C\left[X_1 + \ \ldots + X_{n_R}\right] \tag{3.1}$$

$$I_W = C\left[X_1 + \ \ldots + X_{n_W}\right] \tag{3.2}$$

$$I_B = C\left[X_1 + \ \ldots + X_{n_B}\right] \tag{3.3}$$

In the equations above, I_R, I_W and I_B are phase currents, C is a scaling factor (usually chosen as the customer's circuit breaker rating) used to extend the Beta plot range from 0 to the circuit breaker current rating and X is a random variable (ranging from 0 to 1) selected from the Beta PDF representing the customers' loads. The phasor of the neutral current is equivalent the sum of the phasors of the phase currents. Algebraically, this is expressed by the equation below.

$$\overrightarrow{I_N} = \overrightarrow{I_R} + \overrightarrow{I_W} + \overrightarrow{I_B} \tag{3.4}$$

Since the load currents are represented by a Beta PDF, the voltage drop across each phase resistance and furthermore, the consumer voltage, can also be represented by a Beta PDF. The objective is thus to determine the α and β parameters of the Beta PDF of the consumer voltage. In the equations that follow, the Beta PDF parameters for the load current will be represented by α and β whereas the Beta PDF parameters for the consumer voltage will be represented by α^* and β^*.

The first moment, denoted by $E(Y)$, and the second moment, denoted by $E(Y^2)$, of the Beta PDF can be determined from α, β and C using the equations that follow.

$$E(Y) = C\,\frac{\alpha}{\alpha+\beta} \tag{3.5}$$

$$E(Y^2) = C^2\,\frac{\alpha(\alpha+1)}{(\alpha+\beta)(\alpha+\beta+1)} \tag{3.6}$$

Since power systems consist of more than one node and the random variables at different nodes cannot simply be added before utilising the algorithm, the principle of superposition should be used. Thus, the voltage at a node can be calculated as the sum of voltages from current/voltage sources affecting that node. In the equations that follow, the subscript i will be used to indicate when the superposition principle is used. The effect of the load at each node i will be determined and added to provide a single PDF for voltage drop along the feeders analysed.

The minimum and maximum voltage for all consumers in the feeder section analysed can be expressed using real (Re) and imaginary (j) components. The total consumer voltage, denoted by V_{con_t}, has its maximum and minimum value determined by equations (3.7) and (3.8) below. V_s refers to the magnitude of the sending-end voltage. It should be noted that there is no imaginary component in the minimum voltage calculation for this type of system.

$$V_{max} = \sqrt{\left(V_s - \sum_{i=1}^{N} Re\left(\Delta V_{max_i}\right)\right)^2 + \left(\sum_{i=1}^{N} j\left(\Delta V_{max_i}\right)\right)^2} \tag{3.7}$$

$$V_{min} = \sqrt{\left(V_s - \sum_{i=1}^{N} Re\left(\Delta V_{min_i}\right)\right)^2 + \left(\sum_{i=1}^{N} j(\Delta V_{min_i})\right)^2} = V_s - \sum_{i=1}^{N} Re\left(\Delta V_{min_i}\right) \tag{3.8}$$

The derivation of the first and second moments of the total consumer voltage is rather tedious, but is shown for completeness in this section. Consider R_p as the resistance of the phase conductor between node i and node $(i-1)$ and R_n as the resistance of the neutral conductor for the same two nodes. Then, the real and imaginary components for the minimum and maximum voltage experienced due to a load being connected at node i only is determined as follows.

$$Re\left(\Delta V_{max_i}\right) = -\frac{1}{2}.k_i.R_{pi}.C_i.(n_{Wi} + n_{Bi}) \tag{3.9}$$

$$j\left(\Delta V_{max_i}\right) = \frac{\sqrt{3}}{2}.k_i.R_{pi}.C_i.(n_{Wi} - n_{Bi}) \tag{3.10}$$

$$\Delta V_{min_i} = (1+k_i).R_{pi}.C_i.n_{Ri} \tag{3.11}$$

Where,

$$R_{p_i} = \textstyle\sum_{s=1}^{i} R_{p_s} \tag{3.12}$$

$$k_i = \frac{\sum_{s=1}^{i} R_{n_s}}{\sum_{s=1}^{i} R_{p_s}} \tag{3.13}$$

The first moment of the real component of the total voltage drop, is calculated using the equation below.

$$E(Re(\Delta V_t)) = \sum_{i=1}^{N} E\big(Re(\Delta V_i)\big) = \sum_{i=1}^{N} C_{1i}.R_{pi}.C_i.G \tag{3.14}$$

Where,

$$C_{1i} = (1 + k_i)n_{Ri} - \frac{k_i(n_{Wi} + n_{Bi})}{2} \tag{3.15}$$

$$G = \frac{\alpha}{\alpha + \beta} \tag{3.16}$$

The second moment of the real component of the total voltage drop is calculated the equations below.

$$E(Re(\Delta V_i))^2 = {R_{pi}}^2.{C_i}^2(\,C_{2i}.H + C_{3i}.G^2) \tag{3.17}$$

$$E(Re(\Delta V_t))^2 = \sum_{i=1}^{N} E(Re(\Delta V_i))^2 + \sum_{n=1}^{N} \sum_{p=1,p\neq n}^{N} E(Re(\Delta V_n)).E(Re(\Delta V_p)) \tag{3.18}$$

Where,

$$C_{2i} = {k_i}^2\left(n_{Ri} + \frac{1}{4}n_{Wi} + \frac{1}{4}n_{Bi}\right) + n_{Ri}(2k_i + 1) \tag{3.19}$$

$$C_{3i} = F_{1i}.{k_i}^2 + F_{2i}.k_i + F_{3i} \tag{3.20}$$

$$H = \frac{\alpha(\alpha + 1)}{\alpha + \beta(\alpha + \beta + 1)} \tag{3.21}$$

Furthermore, constants F_{1i}, F_{2i} and F_{3i} are defined by equations (3.22) to (3.24).

$$F_{1i} = n_{Ri}(n_{Ri} - 1) - n_{Ri}(n_{Wi} + n_{Bi}) + \frac{1}{4}(n_{Wi} + n_{Bi} - 1)(n_{Wi} + n_{Bi}) \tag{3.22}$$

$$F_{2i} = n_{Ri}(2n_{Ri} - n_{Wi} - n_{Bi} - 2) \tag{3.23}$$

$$F_{3i} = n_{Ri}(n_{Ri} - 1) \tag{3.24}$$

The first moment of the imaginary component of the total voltage drop, is calculated in a similar manner that the real component was calculated, as indicated in Equation (3.25) below.

$$E(j(\Delta V_t)) = \sum_{i=1}^{N} E\big(j(\Delta V_i)\big) = \sum_{i=1}^{N} \frac{\sqrt{3}}{2}k_i(n_{Wi} - n_{Bi}).R_{pi}.C_i.G \tag{3.25}$$

The second moment of the imaginary component of the total voltage drop, is calculated using the following equations.

$$E(j(\Delta V_i))^2 = {R_{pi}}^2.{C_i}^2(\,C_{4i}.H + C_{5i}.G^2) \tag{3.26}$$

$$E(j(\Delta V_t))^2 = \sum_{i=1}^{N} {R_{pi}}^2 . {C_i}^2 (C_{4i} . H + C_{5i} . G^2) + \sum_{n=1}^{N} \sum_{p=1, p \neq n}^{N} E(j(\Delta V_n)) . E(j(\Delta V_p)) \quad (3.27)$$

Where,

$$C_{4i} = \frac{3}{4} {k_i}^2 (n_{Wi} + n_{Bi}) \quad (3.28)$$

$$C_{5i} = \frac{3}{4} {k_i}^2 [(n_{Wi} - n_{Bi})^2 - (n_{Wi} + n_{Bi})] \quad (3.29)$$

The Taylor Series Expansion can be used to find the first and second moments of the total consumer voltage. This can be done since there is usually only a small difference between the supply voltage (V_s) and the total consumer voltage (V_{con_t}). The resulting equations are as follows.

$$E(V_{con_t}) = V_s - E(Re(\Delta V_t) + \frac{E(j(\Delta V_t))^2}{2V_s} \quad (3.30)$$

$$E\left({V_{cont}}^2\right) = {V_s}^2 - 2V_s . E(Re(\Delta V_t) + E(Re(\Delta V_t))^2 + E(j(\Delta V_t))^2 \quad (3.31)$$

Finally, the first and second moments of the normalised total consumer voltage can be determined using equations (3.32) and (3.33) below. In the equations, the "$*$" denotes a normalised parameter.

$$E\left({V_{con_t}}^*\right) = \frac{E(V_{con_t}) - V_{min}}{V_{max} - V_{min}} = \frac{\alpha^*}{\alpha^* + \beta^*} \quad (3.32)$$

$$E\left({V_{con_t}}^{*2}\right) = \frac{E({V_{con_t}}^2) - 2V_{min} . E(V_{con_t}) + {V_{min}}^2}{(V_{max} - V_{min})^2} = \frac{\alpha^*(\alpha^* + 1)}{(\alpha^* + \beta^*)(\alpha^* + \beta^* + 1)} \quad (3.33)$$

The Beta PDF parameters for the normalised total consumer voltage is easily obtained from equations (3.32) and (3.33). The final step is to determine the consumer voltage corresponding to the chosen confidence level (ρ) , denoted by $V\%$, from the normalised voltage PDF. This is done as follows.

$$V_{con} = V\%(V_{max} - V_{min}) + V_{min} \quad (3.34)$$

Where $V\%$ is obtained using the built-in Beta-inverse function found in many software packages, such as Microsoft Excel. The inputs shown below are used in the Microsoft Excel program.

$$V\% = betainv\left[\left(\frac{100 - \rho}{100}\right), |\alpha^*|, |\beta^*|\right] \quad (3.35)$$

The voltage drop per phase is then determined by calculating the difference between the sending end voltage and the consumer voltage per phase.

3.3 The Herman-Beta Method for LV Feeders with Distributed Generation

The use of distributed generation in power systems is increasing rapidly. Distributed generation, specifically from renewable energy sources, produces intermittent power. It is thus inadequate to determine the load flow solution of a power system with distributed generation using deterministic methods. Since the Herman-Beta method had been accepted as the optimal solution for finding voltage drops on LV feeders using probabilistic load flow, it was decided to modify the method to include the

effects of distributed generation. The modified method (indicated by HB-DG) was presented in [4] and has also been implemented in a software program using the C# programming language (hereafter referred to as HB-C#) [3].

As with the Herman-Beta method for LV feeders presented in Section 3.1.3, loads are once again represented as constant currents. The same assumptions apply as before, with the additional assumption that distributed generators are placed a distance of 0.5m or less away from load nodes. It should be noted that the distributed generators cannot be placed directly at the load node, since this would invalidate the voltage drop equations. Furthermore, the distributed generators are represented as negative loads to obtain negative currents.

3.3.1 Algorithm for 3-Phase, 4-Wire Systems

The Herman-Beta method for LV feeders for 3-phase, 4-wire systems with distributed generation is presented in this section. Before the method is presented, a modified model which includes distributed generators needs to be drawn. This model will be used to determine the equations used in the algorithm. The 3-phase, 4-wire system with distributed generators is modelled as in Figure 3.3. The algorithm was adopted from [4].

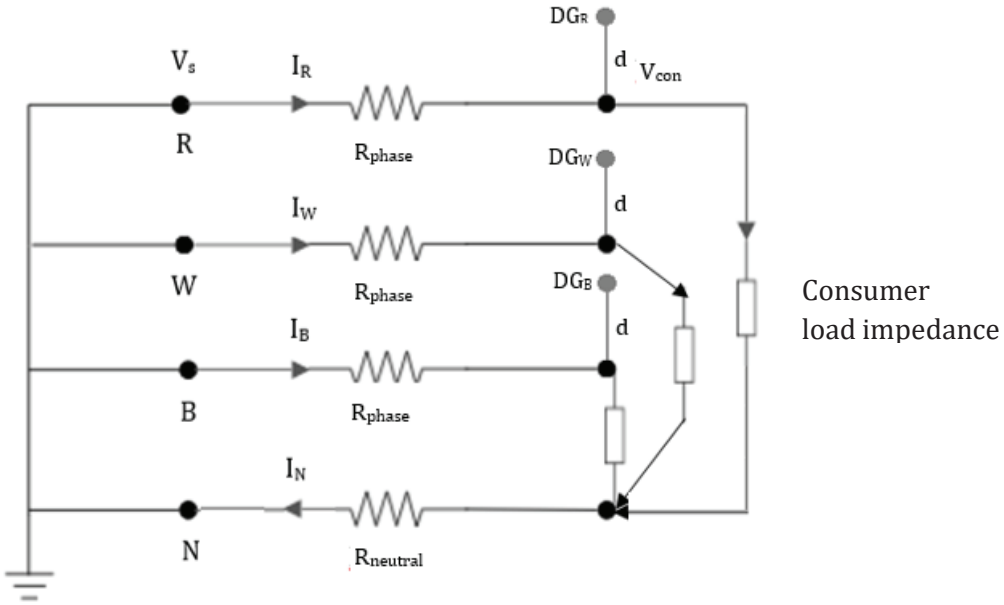


Figure 3.3: Model for 3-Phase, 4-Wire LV System with Distributed Generation

As seen from the figure above, the model for 3-phase, 4-wire LV systems has been modified to include distributed generation. The distributed generation per phase has been denoted by DG_R, DG_W and DG_B. The distance between the distributed generation and the load node is denoted by d and is chosen to have a value between 0.1m and 0.5m.

During generation, current would flow from the generator towards the source. Since the loads are represented as constant currents, simply modelling the distributed generators as negative loads will ensure current flows from the generator towards the source. This approach has been supported by numerous academics [4].

In summary, the following is used as a basis for the algorithm for 3-phase, 4-wire LV systems with distributed generation [4]:

- Distributed generators can be modelled as negative loads. This gives rise to the negative currents which is obtained during generation, since the loads are modelled as constant currents.
- Since the distributed generators are modelled as negative loads, the distribution of the negative currents is best represented using the Beta PDF, similar to consumer loads.
- Generators and customer loads cannot exist at the same node for algebraic integrity. Thus, a generator node is placed a short distance away from the customer node, typically within 0.5m.
- The DG per phase is assigned as a negative amount of customers.
- The majority of the original algorithm remains unchanged. Only a few expressions are modified to include the presence of distributed generation on LV feeders.

In [4], a system containing two nodes was used to determine the terms that would have to be modified to include the effect of distributed generation of LV feeders. An equal number of customers and distributed generators were assigned to each phase. From the investigation conducted, it was determined that the following terms should be modified to allow for the correct calculation of voltage variations in 3-phase, 4-wire systems containing distributed generation:

- The second moment of the total consumer voltage, $E\left({V_{con_t}}^2\right)$.
- The parameters of the Beta PDF, α and β.
- The extremities of the total consumer voltage, V_{min} and V_{max}.

i. Modification of the Second Moment of the Total Consumer Voltage

The modified terms used to calculate $E\left({V_{con_t}}^2\right)$ for 3-phase, 4-wire systems with DG are provided below [4].

$$F_{1i} = |n_{Ri}|(|n_{Ri}| - 1) - |n_{Ri}|(|n_{Wi}| + |n_{Bi}|) + \frac{1}{4}(|n_{Wi}| + |n_{Bi}| - 1)(|n_{Wi}| + |n_{Bi}|) \tag{3.36}$$

$$F_{2i} = |n_{Ri}|(2|n_{Ri}| - |n_{Wi}| - |n_{Bi}| - 2) \tag{3.37}$$

$$F_{3i} = |n_{Ri}|(|n_{Ri}| - 1) \tag{3.38}$$

$$C_{nk2i} = {k_i}^2\left(|n_{Ri}| + \frac{1}{4}|n_{Wi}| + \frac{1}{4}|n_{Bi}|\right) + |n_{Ri}|(2k_i + 1) \tag{3.39}$$

$$C_{nk4i} = \frac{3}{4}{k_i}^2(|n_{Wi}| + |n_{Bi}|) \tag{3.40}$$

$$C_{nk5i} = \frac{3}{4}{k_i}^2[(|n_{Wi}| - |n_{Bi}|)^2 - (|n_{Wi}| + |n_{Bi}|)] \tag{3.41}$$

Where the subscripts are defined as in Section 3.2.3 and $|x|$ denotes the absolute value of x.

ii. Modification of the Beta Probability density function Parameters

To calculate the percentile voltage for a specific percentage confidence (ρ) for these systems, the absolute values of the consumer voltage Beta PDF parameters are used, since the Beta PDF is not defined for values where α and/or β is less than or equal to 0 [4]. Hence, the equation to calculate the percentile voltage at the chosen confidence level is as follows.

$$V\% = betainv\left[\left(\frac{100 - \rho}{100}\right), |\alpha^*|, |\beta^*|\right] \tag{3.42}$$

iii. Modification of the Minimum and Maximum Consumer Voltages

For feeders without distributed generation, also known as passive feeders, the minimum phase voltage is obtained when the phase in question is loaded whilst the other phases are unloaded. Conversely, the maximum voltage is obtained when the phase in question is unloaded and the other phases are loaded.

For feeders with distributed generation, also known as active feeders, the minimum and maximum voltage conditions are the inverse to the passive feeder conditions. The minimum phase voltage is obtained when the phase in question has no distributed generation and the other phases do. Conversely, the maximum voltage is obtained when the phase in question has distributed generation and the other phases do not.

Since both load and generator nodes exist in networks containing distributed generation, the voltage extremities for both types of nodes are calculated independently. Hence, the equations to calculate the minimum and maximum voltages at load and generator nodes are different. To calculate the minimum and maximum voltages for load nodes, equations (3.7) to (3.13) should be used. The equations for DG nodes for the red phase are provided below.

$$V_{DG_{min}} = \sqrt{\left(V_s - \sum_{i=1}^{N} -\frac{1}{2}(n_{W\,i} + n_{B\,i})C_i.R_i.k_i\right)^2 + \left(\sum_{i=1}^{N} j\left(\frac{\sqrt{3}}{2}(n_{W\,i} - n_{B\,i})C_i.R_i.k_i\right)\right)^2} \tag{3.43}$$

$$V_{DG_{max}} = \sqrt{\left(V_s - \sum_{i=1}^{N} n_{R\,i}.C_i.R_i(1+k_i)\right)^2 + \left(\sum_{i=1}^{N} j(0)\right)^2} = \sqrt{\left(V_s - \sum_{i=1}^{N} n_{R\,i}.C_i.R_i(1+k_i)\right)^2} \tag{3.44}$$

3.4 The Herman-Beta Extended Transform

The Herman-Beta method was modified again in [39], where an approximation method for the Herman-Beta Extended Transform was developed. The approximation method extended the algorithm to include the calculation of voltage variations on MV radial feeders. The modifications made still allows for the assessment of voltage variations on LV radial networks. However, the modifications allow for the removal of some assumptions used in the previous LV algorithms, thus increasing the accuracy of the method.

The limitations of the original method are outlined in [7]. In the original Herman-Beta algorithm, feeders are assumed to be purely resistive and loads are assumed to have a power factor of 1. However, it was determined that these assumptions only apply to LV feeders at the instant of maximum demand. Therefore, the unaltered algorithm could not be used on MV radial feeders. Furthermore, error-free results would not be obtained in the assessment of voltage variations at periods other than the instant of maximum demand.

In an attempt to utilise the Herman-Beta method on MV radial networks, the unaltered, original method was applied to a MV network in [5]. Despite the Herman-Beta method for LV feeders being widely accepted and noted as the preferred method for calculating voltage variations in South Africa, poor results were obtained in the assessment of the MV network in [5] using the unaltered method. These poor results prompted the investigation of modifications required to extend the method for the analysis of voltage drops on MV feeders.

The approximation method for the Herman-Beta Extended Transform is presented in [7] and [39]. This variant of the method does not assume unity power factor loads and purely resistive feeders. Furthermore, the approximation method allows for the modelling of loads as either constant impedance, constant current or constant power loads.

The approximation method allowed for the removal of some assumptions by replacing the resistance in the relevant equations with the magnitude of the impedance. The angle of the impedance was not taken into account at this stage. Furthermore, the power factor of loads was taken into account by dividing the circuit breaker current by the power factor of loads/generators at each node. This formed the basis of the approximation method.

In [39], the results obtained using the approximation method were compared to those obtained through Monte Carlo Simulations. Although the Monte Carlo Simulations utilised the exact impedance (magnitude and angle), the results obtained using the approximation method with the approximated impedance (magnitude only) were reasonably accurate. It should be noted that these results were considerably more accurate than the results obtained using the unaltered method on MV feeders.

The results obtained from the approximation method indicated that a degree of error still existed. This prompted further investigation to develop a more accurate method. The Herman-Beta Extended Transform was thus developed for 3-phase, 4-wire and 3-phase, 3-wire systems [2]. Since only 3-phase, 4-wire systems will be catered for in the HBE-Python program, the 3-phase, 3-wire algorithm will not be presented in this dissertation. The development of the algorithm for 3-phase, 3-wire systems can be found in [2].

It is important to note that this study does not take the effect of any form of correlation into account, despite the developed algorithms allowing for it. Therefore, the 3-phase, 4-wire algorithm provided in the following subsection omits all equations that account for correlation.

3.4.1 Algorithm for 3-phase, 4-wire Systems

Before the modified algorithm and revised equations can be presented, it is necessary to illustrate the revised model for 3-phase, 4-wire systems. The revised model is presented in Figure 3.4 below.

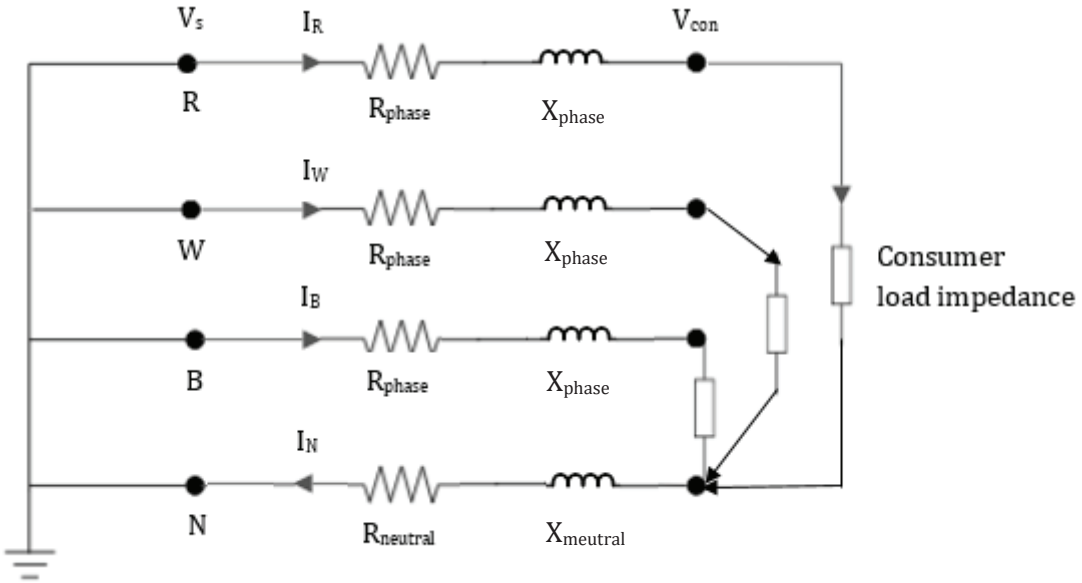


Figure 3.4: Modified Model for 3-Phase, 4-Wire MV System

As seen in the figure, the reactance of the phases and neutral have been included. This has been done since the reactance of MV feeders is not negligible [2], as in the LV feeder scenario. Before the reformulated equations based on the revised model is presented, the use of the per-unit system as a normalisation technique needs to be discussed.

One of the major changes introduced in the Herman-Beta Extended Transform is the use of a per-unit system. The per-unit system is a normalisation technique often used in power systems to simplify the analysis of networks. In addition to this, the per-unit system is used since it allows the load flow variables to be represented by values ranging from 0 to 1. This range aligns with the Beta PDF range, making it easier to calculate the voltage variation along feeders.

To convert all input values to the per-unit system, the base values of each input needs to be determined. The base values for apparent power (S_B) and voltage (V_B) will be specified by the user. The remaining base values are determined using the equations below.

$$I_B = \frac{S_B}{V_B} \tag{3.45}$$

$$Z_B = \frac{{V_B}^2}{S_B} \tag{3.46}$$

$$R_{p_B} = \frac{R_{p_i}}{Z_B} \tag{3.47}$$

$$X_{p_B} = \frac{X_{p_i}}{Z_B} \tag{3.48}$$

$$R_{n_B} = \frac{R_{n_i}}{Z_B} \tag{3.49}$$

$$X_{n_B} = \frac{X_{n_i}}{Z_B} \tag{3.50}$$

$$C_B = \frac{C_i}{I_B} \tag{3.51}$$

Using equations (3.47) to (3.50), the superposition elements of the resistance and reactance can be calculated. The equations are presented below.

$$R_i = \sum_{j=1}^{i} R_{p_{B_{(j)}}} \tag{3.52}$$

$$X_i = \sum_{j=1}^{i} X_{p_{B_{(j)}}} \tag{3.53}$$

$$R_n = \sum_{j=1}^{i} R_{n_{B_{(j)}}} \tag{3.54}$$

$$X_n = \sum_{j=1}^{i} X_{n_{B_{(j)}}} \tag{3.55}$$

$$k_r = \frac{R_n}{R_i} \tag{3.56}$$

$$k_x = \frac{X_n}{X_i} \tag{3.57}$$

The revised model provided in Figure 3.4 requires the system equations to be reformulated. The voltage variation and phase current for the red phase is provided below.

$$\Delta V = I_R\left(R_p + jX_p\right) + I_n(R_n + jX_n) \tag{3.58}$$

$$I_R = I_R \cos \emptyset_p \pm j\, I_R \sin \emptyset_p \tag{3.59}$$

Allowing for neutral currents and phase shifts between currents, the quadrature components of the voltage variation can be determined. The equations are provided below.

$$\Delta V_x = k_3 I_R - \frac{1}{2}k_1(I_W + I_B) - \frac{\sqrt{3}}{2}k_2(I_W - I_B) \tag{3.60}$$

$$\Delta V_y = k_4 I_R - \frac{1}{2}k_2(I_W + I_B) + \frac{\sqrt{3}}{2}k_1(I_W - I_B) \tag{3.61}$$

Where,

$$k_1 = k_r R_i \pm k_x X_i \tan \emptyset_i \tag{3.62}$$

$$k_2 = k_x X_i \mp k_r R_i \tan \emptyset_i \tag{3.63}$$

$$k_3 = (1 + k_r)R_i \pm (1 + k_x)X_i \tan \emptyset_i \tag{3.64}$$

$$k_4 = (1 + k_x)X_i \mp (1 + k_r)R_i \tan \emptyset_i \tag{3.65}$$

Where,

$$\emptyset_i = \text{acos}(PF_i) \tag{3.66}$$

It should be noted that the polarity in equations (3.62) to (3.65) are for inductive and capacitive loads respectively and the PF in equation (3.66) refers to the power factor at the respective node.

To determine the first and second moments of the consumer voltage, the first and second moments of the quadrature components of the voltage variation are required. The first moments for the voltage variation quadrature components are provided in equations (3.67) and (3.68), whereas the second moments are provided in (3.69) and (3.70). G, H and C_B are determined at each node using equations (3.16), (3.21) and (3.51) respectively.

$$E(\Delta V_{x_i}) = (k_3 n_R - k_1 C_1 - k_2 C_2).G.C_B \tag{3.67}$$

$$E(\Delta V_{y_i}) = (k_4 n_R - k_2 C_1 + k_1 C_2).G.C_B \tag{3.68}$$

$$E\left(\Delta {V_{x_i}}^2\right) = {C_B}^2(C_3 H + C_4 G^2) \tag{3.69}$$

$$E\left(\Delta {V_{y_i}}^2\right) = {C_B}^2(C_5 H + C_6 G^2) \tag{3.70}$$

Where,

$$\begin{bmatrix} C_1 \\ C_2 \\ C_3 \\ C_4 \\ C_5 \\ C_6 \end{bmatrix} = \begin{bmatrix} 0.5(n_W + n_B) \\ 0.5\sqrt{3}(n_W - n_B) \\ {k_3}^2|n_R| + c_1|n_W| + c_2|n_B| \\ c_3 + c_4 \\ {k_4}^2|n_R| + d_1|n_W| + d_2|n_B| \\ d_3 + d_4 \end{bmatrix} \tag{3.71}$$

The constants $c_1 - c_4$ and $d_1 - d_4$ are determined using equations (3.72) and (3.73) respectively.

$$\begin{bmatrix} c_1 \\ c_2 \\ c_3 \\ c_4 \end{bmatrix} = \begin{bmatrix} 0.5k_1k_2\sqrt{3} + 0.75{k_2}^2 + 0.25{k_1}^2 \\ -0.5k_1k_2\sqrt{3} + 0.75{k_2}^2 + 0.25{k_1}^2 \\ {k_3}^2|n_R|(|n_R| - 1) + c_1|n_W|(|n_W| - 1) + c_2|n_B|(|n_B| - 1) \\ |n_R||n_W|\left(-k_3k_2\sqrt{3} - k_1k_3\right) + |n_R||n_B|\left(k_3k_2\sqrt{3} - k_1k_3\right) + |n_W||n_B|\left(-1.5{k_2}^2 + 0.5{k_1}^2\right) \end{bmatrix} \tag{3.72}$$

$$\begin{bmatrix} d_1 \\ d_2 \\ d_3 \\ d_4 \end{bmatrix} = \begin{bmatrix} -0.5k_1k_2\sqrt{3} + 0.75{k_1}^2 + 0.25{k_2}^2 \\ 0.5k_1k_2\sqrt{3} + 0.75{k_1}^2 + 0.25{k_2}^2 \\ {k_4}^2|n_R|(|n_R| - 1) + d_1|n_W|(|n_W| - 1) + d_2|n_B|(|n_B| - 1) \\ |n_R||n_W|\left(k_4k_1\sqrt{3} - k_2k_4\right) + |n_R||n_B|\left(-k_4k_1\sqrt{3} - k_2k_4\right) + |n_W||n_B|\left(-1.5{k_1}^2 + 0.5{k_2}^2\right) \end{bmatrix} \tag{3.73}$$

To extend the calculation of the first and second moments of the quadrature components of the voltage variation to successive nodes, equations (3.67) to (3.70) are inserted into the equations below.

$$E(\Delta V_{x_t}) = \sum_{i=1}^{N} E(\Delta V_{x_i}) \tag{3.74}$$

$$E(\Delta V_{y_t}) = \sum_{i=1}^{N} E(\Delta V_{y_i}) \tag{3.75}$$

$$E({\Delta V_{x_t}}^2) = \sum_{i=1}^{N} E({\Delta V_{x_i}}^2) + \sum_{j=1}^{N} \sum_{k=1,k\neq j}^{N} E(\Delta V_{x_j})E(\Delta V_{x_k}) \tag{3.76}$$

$$E({\Delta V_{y_t}}^2) = \sum_{i=1}^{N} E({\Delta V_{y_i}}^2) + \sum_{j=1}^{N} \sum_{k=1,k\neq j}^{N} E(\Delta V_{y_j})E(\Delta V_{y_k}) \tag{3.77}$$

From equations (3.74) to (3.77), the first and second moments of the total consumer voltage can be determined. The equations are provided below.

$$E(V_c) = V_s\left(1 - \frac{E\left(\Delta V_{x_t}\right)}{V_s} + 0.5\frac{E({\Delta V_{y_t}}^2)}{{V_s}^2}\right) \tag{3.78}$$

$$E\left({V_c}^2\right) = {V_s}^2 - 2V_sE\left(\Delta V_{x_t}\right) + E\left({\Delta V_{x_t}}^2\right) + E({\Delta V_{y_t}}^2) \tag{3.79}$$

To determine the PDF of the consumer voltage, the α and β values need to be determined. The equations used to calculate the consumer voltage PDF parameters are provided in equations (3.80) and (3.81) below.

$$\alpha_{V_{con}} = \frac{E\left(V_c{}^2\right) - E(V_c)}{E(V_c) - \frac{E\left(V_c{}^2\right)}{E(V_c)}} \tag{3.80}$$

$$\beta_{V_{con}} = \frac{\alpha_{V_{con}}}{E(V_c)} - \alpha_{V_{con}} \tag{3.81}$$

The per-unit percentile consumer voltage is determined using the Beta parameters, wherefrom the actual percentile consumer voltage is determined by multiplying the per-unit value by the base voltage, V_B. The respective calculations are provided in equations (3.82) and (3.83) below.

$$V^*_{con\%} = betainv\left[\left(\frac{100-\rho}{100}\right), |\alpha_{V_{con}}|, |\beta_{V_{con}}|\right] \tag{3.82}$$

$$V_{con\%} = V^*_{con\%}.V_B \tag{3.83}$$

This concludes the voltage assessment portion of the algorithm for 3-phase, 4-wire systems. A step-by-step algorithm is provided in Appendix B

3.5 Chapter Summary

This chapter presented the evolution of the Herman-Beta method. The chapter can be summarised as follows:

- The Herman-Beta method requires a significant amount of inputs from the user. It is crucial that the user understands the meaning of all inputs before using the method.
- The equations used by the method are not straightforward and easy to understand. The program helps to eliminate the need for the user to manually calculate all results.
- The original algorithm only catered for passive LV radial networks.
- Various assumptions were utilised in the original Herman-Beta method, such as loads operating at unity PF and feeders being purely resistive.
- The original method was altered by allowing for the addition of generators through modified equations. To maintain algebraic integrity, the load nodes and generator nodes are modelled as being conceptually separate. Therefore, the generator nodes are assumed to be located 0.1m away from load nodes.
- For active feeders, the generators are inserted as negative values to produce a voltage rise. Some equations were changed to utilise absolute values to account for the negative values obtained when generators are present. The minimum and maximum consumer voltage equations were also modified.
- The modified Herman-Beta method allowing for the analysis of passive and radial LV feeders was reformulated to facilitate the analysis of MV feeders as well as LV feeders. The equations were modified to remove the assumptions which apply to LV feeders. The modified method, known as the Herman-Beta Extended Transform, takes into account the reactance of MV and HV feeders and allows for non-unity PF loads.
- The Herman-Beta Extended Transform uses a per-unit systems to easily relate the results to a Beta PDF output.

4. Methodology

This chapter outlines the methodology used in this study. The subsequent chapter is to be read in conjunction with this chapter and is an extension of the methodology followed. The general process is explained in this chapter, whilst detailed designs of the developed programs are provided in the subsequent chapter. Firstly, a brief introduction is given, outlining the general process followed to develop the network planning tool. Thereafter, the instruments that were used to develop the tools are discussed briefly. A flow chart showing the process of transforming inputs to outputs in the network planning tool is then provided. Thereafter, methods used to evaluate the accuracy and correct translation of the Herman-Beta Extended Transform to the programming language of choice are discussed. Finally, the research methodology is analysed and limitations are stated.

4.1 Introduction

The process followed to achieve the objectives of this study is as follows:

- Analyse existing tools developed for the Herman-Beta method and Herman-Beta Extended Transform to determine whether it was implemented accurately. If errors are found in the existing tools, these should be rectified in the developed programs, where possible.
- Since the previous versions of the Herman-Beta tool were developed using MS Excel, the Herman-Beta Extended Transform will first be developed using this program, to a certain extent. All aspects of the algorithm can be implemented using MS Excel, besides the application of the superposition principle that needs to be applied when determining moments. This is ideally done using matrices, which is a tedious process in MS Excel. The HBET tool developed using MS Excel will be referred to as HBE-Excel.
- After the HBE-Excel tool has been developed, a program will be created using the Python programming language. The equations will be replicated, and the results obtained from each tool will be compared during development to ensure the correct translation to code. The Python program (HBE-Python) will have the capability of readily applying the superposition principle, unlike the HBE-Excel tool.
- Once both tools have been developed, they will be evaluated against results obtained using the only previously developed program for the HBET [2], developed using MATLAB, to determine the accuracy with which they were implemented. The MATLAB program is referred to as the HBE-MATLAB tool in the remainder of this dissertation.

A detailed description of the process used to develop the network planning tools is provided in the subsequent sections.

4.2 Instruments Used to Develop the Network Planning Tool

Since its inception, the Herman-Beta method has evolved tremendously. Thus, a number of tools were developed previously for the use of network planners. The majority of these tools were developed using MS Excel, although recently tools have been developed using C# and MATLAB. The only tool previously developed for the Herman-Beta Extended Transform was developed using MATLAB, an expensive package which a limited number of network planners have access to. Since the Herman-Beta Extended Transform was never developed in MS Excel, the starting point in the development of the network planning tool was to firstly implement it using MS Excel. This tool, as well as other instruments used in the development of the network planning tool, are presented in this section.

4.2.1 *Previously Developed Programs*

The previously developed programs utilising the Herman-Beta method were analysed to determine the inputs which network planners had at the design stage, as well as the outputs they would be interested in. The founder of the method, Ron Herman, developed a very useful tool. The tool accepts a range of user inputs and provides the consumer-end voltage, voltage drop and current flowing through the radial feeder. Furthermore, it also provides the user with a feeder voltage profile, indicating the voltage at each node along the feeder. This tool was later modified to include the algorithm for LV feeders with distributed generators. The latest available version of this HB-DG tool will be analysed for accuracy and then modified to facilitate the Herman-Beta Extended Transform.

In Section 2.6, various developed tools were analysed in terms of their practicality and aesthetics. In addition to this, guidelines for the successful development of a network planning tool were obtained from [35] and are listed in Section 2.6.3. These previously developed tools and the network planning tool guidelines will be utilised in the development of the HBE-Python program.

4.2.2 *Herman-Beta Extended Transform Implemented Using MS Excel*

The Herman-Beta Extended Transform was first to be implemented using MS Excel. To do this, the HB-DG tool was scrutinised for accuracy. The portion of the tool which was of particular interest to this study was the portion which allowed for the analysis of 3-phase LV radial feeders with DG. The inputs and outputs of the spreadsheet tool would be used as a guideline for the development of the HBE-Excel tool.

The latest version of the Herman-Beta spreadsheet tool (version A3.ver4a) [40] was distributed with Creative Commons licence in 2010, and was thus acceptable for use in this work. No modifications were made to the latest version of the spreadsheet in this subsection.

The inputs of this spreadsheet tool are shown in blue in Figure 4.1 below. Cells which have not been highlighted are calculated or default values.

THREE-PHASE H-B VOLTDROPS WITH DG

Input Blue Cells Only - Results in Red

CABLES	°C
t1	20
t2	30
Nom Voltage	230.00
% Volt Limit	10

Code	R/km@t2	R/km@t1	T	k	Description
ABC25	1.248	1.20	228	0.5	ABC 25 mm² Al French std
ABC35	0.903	0.87	228	0.7	ABC 35 mm² Al French std
ABC50	0.667	0.64	228	1	ABC 50 mm² Al French std
ABC70	0.461	0.44	228	1.4	ABC 70 mm² Al French std
ABC95	0.333	0.32	228	1.9	ABC 95 mm² Al French std
A10	1.962	1.89	241	1	AIRDAC 10mm² Cu
35mmCu	0.544	0.52	241	1	35 mm Copper
50mmCu	0.402	0.39	241	1	50 mm Copper
70mmCu	0.278	0.27	241	1	70 mm Copper
95mmCu	0.200	0.19	241	1	95 mm Copper

Vs	230.00
D Risk %	10

Generators=neg
Enter in 'G' line

				Load Parameters			Conductor Details	
Node	Red	White	Blue	Alpha	Beta	Cb	Length	Cable
	ma	mb	mc			[A]	[m]	Code
1L	1	2	3	1.238	2.990	60	20	35mmCu
1G				1.000	1.000			35mmCu
2L	2	4	6	1.238	2.990	60	18	35mmCu
2G	- 1	- 1	- 1	1.000	1.000	60	0.1	35mmCu
3L	7	8	9	1.238	2.990	60	25	70mmCu
3G				1.000	1.000			70mmCu

Figure 4.1: Inputs Required by the HB-DG Tool [40]

As seen in the figure above, there are a wide range of inputs required from the user to obtain the voltage variation along the radial feeder. The list of inputs required from the user is as follows:

- Temperature, $t2$ at which the cable is expected to operate at.
- Nominal/source voltage of the system.
- Maximum allowable voltage drop limit.
- The % risk (% $risk = 100 - \%\ confidence\ level$) at which the percentile consumer voltage will be calculated.
- Cable resistance at temperature $t1$, constants T (dependent on cable material) and k ($\frac{R_n}{R_p}$), as well as the cable description and codename.
- Number of consumers connected to phases A, B and C per node and connection type (load/generator).
- Load/Generator PDF parameters, namely Alpha (α), Beta (β) and scaling factor/circuit breaker size.
- Cable parameters, namely cable length and type.

In addition to the inputs provided above, the Herman-Beta Extended Transform requires the user to enter a few additional inputs. These user inputs will be incorporated in the HBE-Excel tool. The additional user inputs required are as follows:

- Per-unit base apparent power, S_{base}
- Per-unit base voltage, V_{base}
- The parent node connected to each node. This will be used to configure feeders with laterals.

Once all inputs have been inserted in the correct format in the HB-DG tool, a large number of equations, arising from the superposition, are calculated in the background. The voltage and voltage deviation at the consumer end is provided in tabular format and a voltage profile is drawn. The voltage profile indicates the feeder voltage versus the distance from the source voltage. An example of the outputs of the program is shown in Table 1 and Figure 4.2.

Table 1: Example of Result Table Output from HB-DG Tool [40]

Results	Red	White	Blue	Unit
%-tile Vcon	225.58	221.77	218.02	V
%Volt drop	1.92	3.58	5.21	%V
%-tile Isum	200.19	278.59	356.04	A
Mean Isum	145.69	215.96	286.23	A
Stdev Isum	41.54	47.92	53.54	A

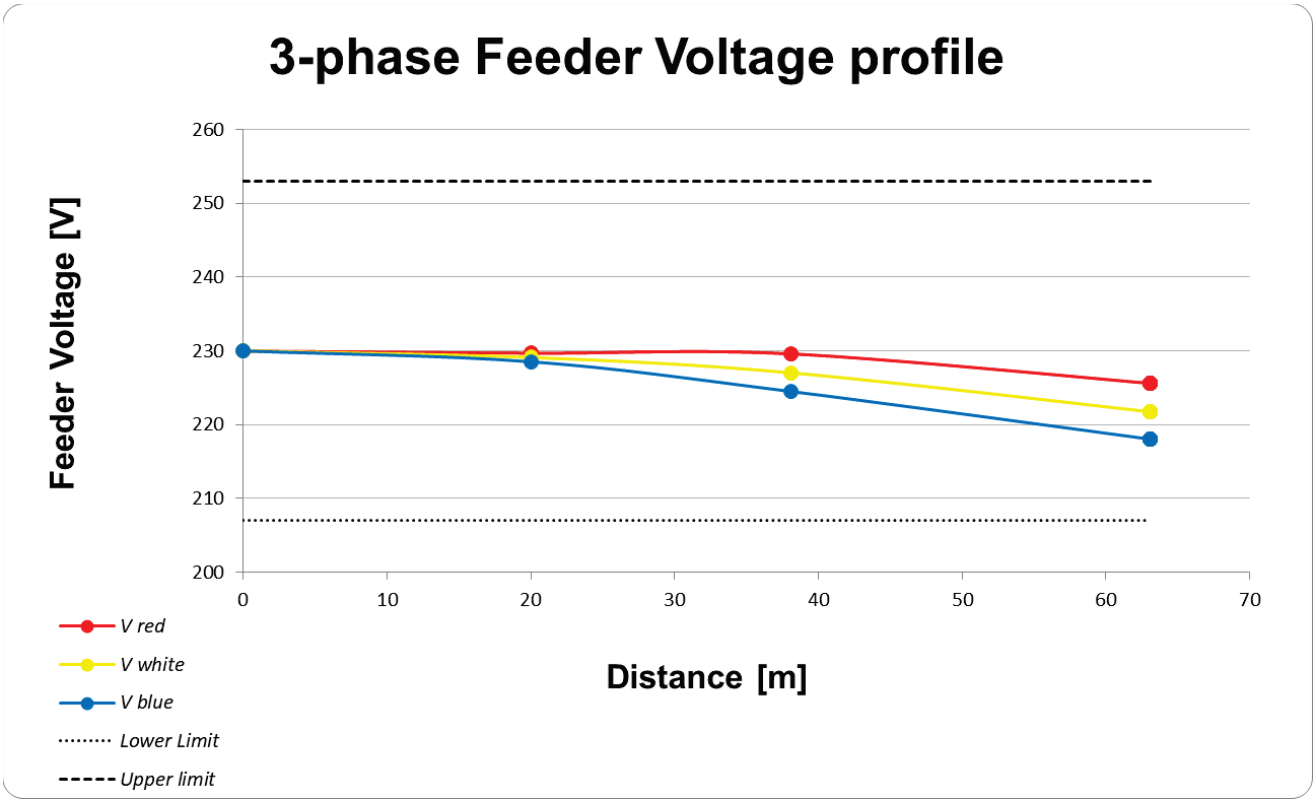


Figure 4.2: Example of Voltage Profile Output from HB-DG Tool [40]

As indicated by the results above, the consumer-end voltage and the voltage deviation per phase is provided. Thus, the user is immediately able to determine whether the system created adheres to the voltage deviation limits specified. This is also depicted in graphical format, as in Figure 4.2, where the voltage limits are indicated using dotted lines. Furthermore, the user can examine the change in voltage at each node, where each dot denotes a node on the respective trend.

The calculations performed in the latest spreadsheet tool will be revised to reflect the equations used in the HBET, as presented in Chapter 3.4. A detailed description of the development of the HBE–Excel tool is provided in Section 5.1.

4.2.3 Doctoral Thesis on Modifying the Herman-Beta Method for Feeders of Any Voltage

A Master's dissertation [39] provided an approximation method for the HBET by removing the assumptions of purely resistive feeders and loads operating at unity power factor. To facilitate this, the resistive components in the original equations were replaced with the magnitude of the impedance and the circuit breaker value was divided by the power factor to account for non-unity power factor loads.

Although the method utilised in the dissertation produced improved results, a degree of error still existed, due to the impedance angle not being taken into account, among other factors. The error was evident by comparison with Monte Carlo simulations. Thereafter, Chihota developed an accurate analytical method in his doctoral thesis [2]. The method and equations derived in the doctoral thesis are the most important instruments that were used to develop the network planning tool.

4.3 Program Flow Chart

A flow chart of the general sequence to be followed whilst using the HBE-Python tool will be shown in this section. The flow chart will indicate the process of utilising the inputs of the Herman-Beta Extended Transform to calculate the outputs required by the network planner. The flow chart will contain process boxes, represented by rectangles, manual inputs required from the user, represented by trapeziums and decision boxes, represented as diamonds. The flow chart is provided in Figure 4.3.

The flow chart indicates that the user will first be required to configure their network by utilising the 'Inputs' spreadsheet provided with the program. Provision will be made for the user to utilise cable or overhead line types which have not been included in the default library. This is done by adding the desired cable type, along with its parameters, to one of the conductor libraries. Once the user has created the system to be evaluated, the 'Inputs' spreadsheet should be saved and the HBE-Python program should be started.

The HBE-Python program will prompt the user to enter the initial input parameters, namely the per-unit base parameters, supply voltage, cable operating temperature, voltage limit and probabilistic confidence level. Clicking on the button below these initial inputs would provide the user with results in graphical and tabular format, provided all inputs have been inserted in the correct format. If the inputs were entered in the incorrect format, an error message will be displayed to the user, asking the user to revise the input values provided. Furthermore, if the user is not satisfied with the results obtained using their initial inputs, they will have the option of revising the initial inputs and recalculating the results.

It should be noted that the flow chart presented only indicates the process flow from initial inputs to the desired user outputs. All other built-in functions will be discussed in Chapter 5.

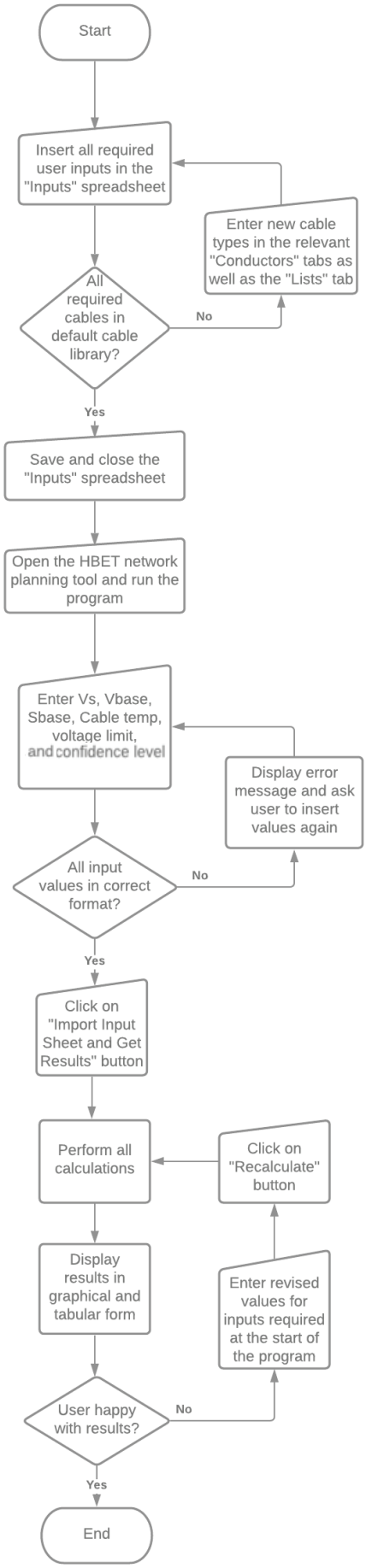


Figure 4.3: Program Flow Chart

4.4 Evaluation of Developed Tools

Once the network planning tools have been developed, they will need to be tested for accuracy. During development, the HBE-Excel tool will be tested by setting the PF of loads to unity and the reactive component of feeders to zero, whilst utilising the modified equations of the Herman-Beta Extended Transform. If the HBE-Excel tool has been implemented accurately, the results obtained should be the same as those produced by the HB-DG tool. Once this has been implemented successfully, the same procedure will be applied to generator nodes.

After the correct results for a system with unity PF loads/generators and purely resistive feeders are obtained, the results for a system with non-unity PF loads/generators containing feeders which are not purely resistive will be compared to the results obtained from an identical system using the HBE-MATLAB program. Due to the limited functionality of the HBE-Excel tool, only the node results at the end of the lateral are expected to match the results obtained using the HBE-MATLAB tool, due to the incapability of the HBE-Excel tool to apply the superposition principle to previous nodes.

Once the HBE-Excel tool has been developed, the HBE-Python network planning tool will be developed. The HBE-Python tool will have the functionality of easily applying the principle of superposition for moments and the results will thus be readily comparable to the HBE-MATLAB results. To thoroughly evaluate the developed program, a number of scenarios will be tested. The scenarios that will be tested are as follows:

- A 12-bus passive feeder with no lateral branches.
- A 12-bus active feeder with no lateral branches.
- A 33-bus passive feeder with lateral branches.
- A 33-bus active feeder with lateral branches.

The scenarios mentioned above were tested by Chihota and presented in [2]. These scenarios will be replicated in the HBE-Python program and the results will be compared with those obtained in [2]. If the HBET equations have been translated correctly into the code used for the HBE-Python tool, the results obtained will be identical.

4.5 Analysis and Limitations of Methodology

The general methodology was provided in this chapter and will be expanded on in Chapter 5, where detailed designs of the HBE-Excel and HBE-Python tools are given. The methodology proposed in this chapter is expected to allow for accurate implementation of the tools, since the results obtained by the HBE-Excel tool will be compared to the HBE-MATLAB results after each slight modification to the existing spreadsheet.

To ensure accurate results are obtained, the HB-DG tool will be checked for accuracy before modifying the equations for the development of the HBE-Excel tool. The accuracy of the HB-DG tool will be determined by checking whether the equations provided in the relevant studies, which allowed for the expansion of the method to active feeders, had been implemented correctly in the HB-DG tool. Any errors found will first be rectified before modifying the equations for the HBE-Excel tool.

Once the HBE-Excel tool has been developed, the HBE-Python tool will be developed. At this point, the most challenging part would be to convert the program to code. Another advantage of developing the HBE-Python tool after the HBE-Excel tool is that the calculations can be checked for accuracy while the program is being developed, by using the HBE-Excel tool as a reference.

The major limitation of this methodology is that the program will not be able to assess voltage variations on bi-phase radial networks, despite the development of the Herman-Beta method for active feeders on these systems. This is outside the scope of this research because the Herman-Beta Extended Transform had not been developed for bi-phase networks at the time of this study.

The HBE-Excel tool also has the limitation of not being able to easily apply the superposition principle when determining moments. This process requires the loading per phase, as well as the associated Beta parameters and PF at each node to be brought back to the node being assessed to accurately determine the voltage and voltage variation at that node. This limitation will not apply to the HBE-Python tool.

4.6 Chapter Summary

Based on the methodology presented in this chapter, the following summarising points can be made:

- Two network planning tools will be created. One will be developed using MS Excel (HBE-Excel) and the other using Python programming language (HBE-Python).
- Previously developed tools and the doctoral thesis wherein the HBET was developed were used as instruments in the development of the network planning tools.
- The required inputs and the produced outputs of the HBET are outlined in the provided algorithms in Appendix A
- The flow chart presented in this chapter does not indicate all possible functions of the Python network planning tool. It merely shows the main process that will be utilised by the user.
- The network planning tools will be evaluated using previously developed tools and results obtained in the doctoral thesis wherein the HBET was developed [2].
- The methodology described in this chapter is expected to allow for the development of accurate network planning tools.

5. Detailed Design

This chapter is to be read in conjunction with the previous chapter, which merely provided a brief description of the processes followed to conduct this study. In this chapter, the process of developing both network planning tools is discussed in detail. Firstly, the detailed design of the HBE-Excel tool is provided. Before developing the HBE-Excel tool, the HB-DG tool's formulas are checked for accuracy using simple tests. The verified HB-DG spreadsheet tool will be used as a reference for the development of the HBE-Excel tool. This will be done by adding the new inputs required to the spreadsheet and modifying the equations according to the Herman-Beta Extended Transform's equations. Finally, the new layout of the tool is presented.

The second detailed design that will be discussed is for the HBE-Python tool. Essentially, this was only implemented after the HBE-Excel tool was developed and tested, but is shown in this order for fluency. The detailed design of the HBE-Python program mainly focuses on the features, flow and aesthetics of the tool. The entire code used to develop the program is provided at the end of this dissertation as an appendix. The evaluation and testing of the developed tools are provided in the subsequent chapter.

5.1 Development of the HBE-Excel Tool

The HB-DG tool, which allowed for the probabilistic voltage variation analysis of passive and active radial LV feeders, was used as a reference to develop the HBE-Excel tool. However, it was decided that the HB-DG tool should be checked for accuracy before using it as a reference, to avoid errors from being carried through. Therefore, the first step in developing the HBE-Excel tool was to inspect and verify that all equations were implemented correctly. If any errors are found, they will first be corrected before modifying the equations for the development of the HBE-Excel tool.

5.1.1 Verification of the HB-DG Tool

The study conducted to create the HB-DG tool [4] presented modified equations to be utilised when using the method. These equations were compared to the formulas contained in the HB-DG tool. After careful inspection of all formulas, it was determined that the equations had been translated to MS Excel without error.

As an additional test, a few simple scenarios were modelled in the HB-DG tool as well as the original Herman-Beta spreadsheet, and the results compared. The generator nodes in the previously developed tool were ignored during this comparison, since the original method only catered for load nodes. The parameters used to compare the two tools are provided in Table 2 to Table. 4 below.

Table 2: System Parameters Used to Check Accuracy of HB-DG Tool

Parameter	Value
t_2	30
V_S	230
$\%\ Volt\ Drop\ Limit$	10
$\%\ Risk$	10
$Number\ of\ nodes\ in\ System$	3

Table 3: Node Parameters Used to Check Accuracy of HB-DG Tool

	No Consumers on phases			Load Parameters			Conductor Details	
Node	Red	White	Blue	Alpha	Beta	Cb	Length	Cable
	ma	mb	mc			[A]	[m]	Code
1	1	2	3	1.238	2.990	60	20	35mmCu
2	2	4	6	1.238	2.990	60	18	35mmCu
3	7	8	9	1.238	2.990	60	20	35mmCu

Table 3 shows the information inserted for load nodes in the original Herman-Beta spreadsheet tool. The same information was inserted in the HB-DG tool, with the number of consumers and cable lengths for generator nodes left blank. The node input parameters for the HB-DG tool is shown in Table. 4 below.

Table. 4: Load Node Parameter Inputs for HB-DG Tool

				Load Parameters			Conductor Details	
Node	Red	White	Blue	Alpha	Beta	Cb	Length	Cable
	ma	mb	mc			[A]	[m]	Code
1L	1	2	3	1.238	2.990	60	20	35mmCu
1G				1.238	2.990	60		35mmCu
2L	2	4	6	1.238	2.990	60	18	35mmCu
2G				1.238	2.990	60		35mmCu
3L	7	8	9	1.238	2.990	60	20	35mmCu
3G				1.238	2.990	60		35mmCu

Since identical information was inserted into the two spreadsheet tools, all results obtained are expected to be the same. If this is the case, it would prove that the formulas in the HB-DG tool are correct, for load nodes at least. The results obtained from the HB-DG tool are provided in Table 5 below.

Table 5: Results Obtained for Load Nodes during Accuracy Check

Results	Red	White	Blue	Unit
%-tile Vcon	224.44	220.44	216.51	V
%Volt drop	2.42	4.16	5.87	%V
%-tile Isum	225.34	304.44	382.35	A
Mean Isum	175.69	245.96	316.23	A
Stdev Isum	37.76	44.68	50.66	A

The results obtained from the original Herman-Beta spreadsheet tool were identical to those obtained in the table above, verifying that the HB-DG tool had been accurately implemented for load nodes. To test the accuracy of the generator nodes, the same node parameters were transferred from the load nodes to the generator nodes. However, as required for generator nodes, the loading per phase was changed to negative values. The input parameters for the testing of the generator nodes is shown in Table 6 below.

Table 6: Generator Node Parameter Inputs for HB-DG Tool

				Load Parameters			Conductor Details	
Node	Red	White	Blue	Alpha	Beta	Cb	Length	Cable
	ma	mb	mc			[A]	[m]	Code
1L				1.238	2.990	60	0.001	35mmCu
1G	-1	-2	-3	1.238	2.990	60	20	35mmCu
2L				1.238	2.990	60	0.001	35mmCu
2G	-2	-4	-6	1.238	2.990	60	18	35mmCu
3L				1.238	2.990	60	0.001	35mmCu
3G	-7	-8	-9	1.238	2.990	60	20	35mmCu

For this scenario, the results are expected to be equal but opposite to the results obtained in Table 5. This is true since a system consisting of generators only will cause current to flow from the end of the

feeder towards the source instead of from the source to the end of the feeder. The results are provided in Table 7.

Table 7: Results Obtained for Generator Nodes During Accuracy Check

Results	Red	White	Blue	Unit
%-tile Vcon	235.55	239.48	243.44	V
%Volt drop	-2.41	-4.12	-5.84	%V
%-tile Isum	-225.34	-304.44	-382.35	A
Mean Isum	-175.69	-245.96	-316.23	A
Stdev Isum	37.76	44.68	50.66	A

From the results above, it is evident that the voltage variation experienced at the end of the feeder is close, but not identical, to the results obtained for the system containing only loads. This may be due to the fact that there are slight variations between the equations for load nodes and generator nodes. As expected, the results obtained for the phase currents are exactly opposite but equal to those obtained in Table 5. From this check it is evident that the equations for the generator nodes have been implemented accurately.

Thus far, all checks done were to determine the accuracy with which the HB-DG tool was implemented. The results obtained indicates that the spreadsheet tool was implemented without error. The checks conducted provided a high level of confidence in using the HB-DG tool as a reference when developing the HBE-Excel tool.

5.1.2 Creating the HBE-Excel Tool

To develop the HBE-Excel tool, the HB-DG tool needs to be amended. Many formulas will needed to be changed completely, while others will have to be added, to align with the equations provided in Chapter 3.4. Furthermore, the additional inputs required from the user will have to be built into the HBE-Excel tool.

The additional inputs added to the HBE-Excel tool are as follows:

- Per-unit base apparent power, S_{base}
- Per-unit base voltage, V_{base}
- The reactance per kilometre at 20°C for each conductor in the predefined library.
- The number of nodes in the user's system.
- The parent node connected to each node.
- The power factor at each load/generator node.

The resultant layout of the HBE-Excel tool is shown in Figure 5.1 below.

HBET: 3-Phase, 4-Wire System

Input Blue Cells Only - Results in Red

CABLES	°C
t1	20
t2	40
Nom Voltage (kV)	11.00
% Volt Limit	10
Per-unit S base (MVA)	10
Per-unit V base (kV)	19.80
Number of nodes	3
Vs (kV)	0.56
D Risk %	5

Code	R/km@t2	X/km@t2	R/km@t1	X/km@t1	T	k	Current Rating	Description
ABC25	1.297	0.104	1.20	0.096	228	0.5	105	ABC 25 mm² Al French std
ABC35	0.938	0.104	0.87	0.096	228	0.7	144	ABC 35 mm² Al French std
ABC50	0.693	0.097	0.64	0.090	228	1	183	ABC 50 mm² Al French std
ABC70	0.479	0.096	0.44	0.089	228	1.4	228	ABC 70 mm² Al French std
ABC95	0.346	0.093	0.32	0.086	228	1.9	277	ABC 95 mm² Al French std
A10	2.035	1.471	1.89	1.366	241	1	50	AIRDAC 10mm² Cu
35mmCu	0.564	0.062	0.52	0.058	241	1	170	35 mm Copper
50mmCu	0.417	0.062	0.39	0.058	241	1	200	50 mm Copper
70mmCu	0.289	0.058	0.27	0.054	241	1	245	70 mm Copper
95mmCu	0.208	0.056	0.19	0.052	241	1	300	95 mm Copper

Generators=neg

Enter in 'G' line

Node	Parent Node	Child Node	Load Parameters: Red ma	White mb	Blue mc	Alpha	Beta	Cb [A]	Conductor Details: Power Factor	Length [m]	Cable Code
1L	Source	1	12	11	11	1.000	2.000	80	0.8	100	ABC25
1G	Source	1				3.000	4.000	60	1		ABC25
2L	1	2	4	4	4	1.000	2.000	60	0.9	100	ABC35
2G	1	2				5.000	6.000	60	1		ABC35
3L	2	3	7	8	9	1.000	2.000	70	0.85	100	ABC35
3G	2	3				5.000	6.000	80	1		ABC35

Figure 5.1: Layout of Input Sheet for HBE-Excel Tool

As seen in the figure above, the general aesthetic of the program remains unchanged. As before, input cells are highlighted blue, calculated values are indicated by white cells, whereas the final results obtained are indicated in red below the load parameter input table. Another major change to the spreadsheet is the requirement for the user to specify the parent node connected to each child node. After the user inserts a value for the number of nodes in their system, the 'Node' and 'Child Node' columns are automatically populated. The user is then required to specify the parent node of each child node, thus allowing for the analysis of feeders containing laterals.

After all the input values have been inserted in the required format, the equations of the HBET will be utilised in the background to determine the probabilistic current flow and voltage variation at the consumer end. In addition to the voltage profile drawn in the previously developed Herman-Beta spreadsheet tools, separate plots indicating the mean and standard deviations of the voltage at each node will be illustrated. Examples of the mean and standard deviation plots for the simple system indicated in Figure 5.1 is shown in Figure 5.2 and Figure 5.3 respectively.

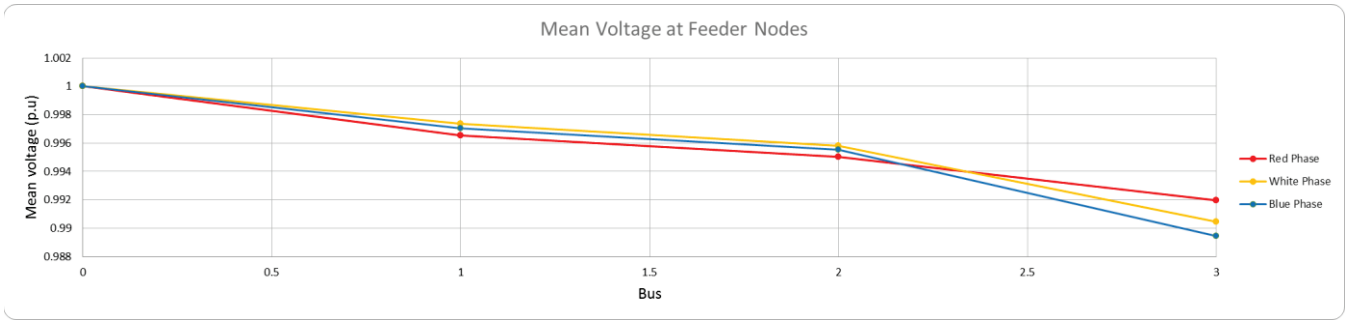


Figure 5.2: Example of Mean Voltage Plot for Simple System

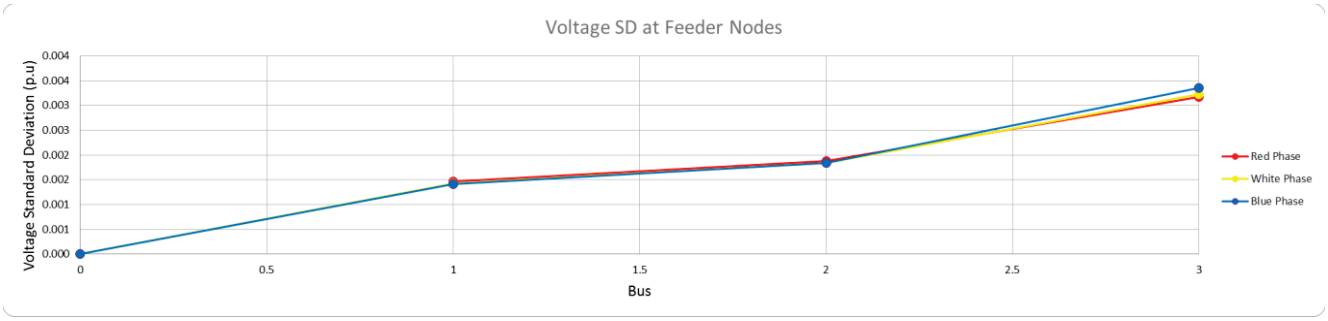


Figure 5.3: Example of Voltage Standard Deviation Plot for Simple System

The HBE-Excel tool's results are only correct for the final node, since superposition cannot easily be applied to upstream nodes. Accurate results can be obtained by developing a program in Microsoft using Microsoft Visual Basic. However, it was decided that the HBE-Python tool would be developed instead, due to its vast functionality and availability of online resources. Therefore, the development of the HBE-Excel tool will not be discussed any further.

5.2 Development of the HBE-Python Tool

After it was determined that HBE-Excel tool could not accurately determine the voltage variation at all nodes, the development of the HBE-Python tool commenced. The limitation of the HBE-Excel tool will be overcome using arrays in Python. The primary objective of the HBE-Python tool is to offer users the capability of utilising the HBET using an easy-to-use, freely available program. Python was selected as the programming language that would be used to develop the program, since it is freely available, can be used to easily create GUIs and has many powerful built-in functions.

5.2.1 Development Environment

The HBE-Python tool was developed and tested using the Anaconda distribution package. Using this distribution package ensures that all open-source packages, libraries and wheels are readily usable, without having to install packages manually via the command prompt. The Anaconda distribution package includes a special programming interface known as Spyder, which utilises the Python programming language but allows for greater analysis of results and easier debugging by instantaneously raising potential program errors. As of November 2022, the latest version of Spyder utilised Python v.3.9.

The development of the HBE-Python tool was conducted on a personal laptop. The technical specifications of the device are as follows:

- Model – HP EliteBook 850 G7 Notebook PC
- Processor - Intel (R) Core (TM) i5-10310U CPU @1.7GHz
- Installed memory -24 GB

5.2.2 Key Considerations

Before developing the program, a number of key features which should be utilised by the HBE-Python tool were identified. These were as follows:

- A menu bar, such as those contained in many existing programs, should be situated on the main window. The menu bar often consists of 'File' and 'Help' dropdown menus, among others.
- A user guide to explain to users how to utilise the program.
- An input spreadsheet to accept user inputs for the designed system.
- The user should have the option of adding cable/conductor types to the default library.

- Error messages should be produced when inputs are inserted in the incorrect format.
- The results produced by the program should be shown graphically. The results at the consumer end should also be tabulated.

Program developers often break down their programs into simpler functions, instead of developing the entire program as a single portion of code. By creating functions, the developer is able to focus on specific tasks to be completed by a function and can easily debug errors contained within that function. Once a function is completed, it is tested to determine if it performs the desired function.

Before developing a program, it is useful to determine which functions will be used to simplify and give structure to the program. This approach was followed with the development of the HBE-Python tool, wherefrom it was deduced that the following functions would be developed:

- An "About" function, that provides the user with background information to the HBET and the developed program.
- An "Exit Program" function, that terminates the program when the relevant option in the dropdown menubar is selected.
- Two "Beta Calculator Window" functions, which are used to configure and display the selected Beta calculator when the respective Beta parameter calculator option is selected.
- Two "Beta Parameter Calculation" functions, which performs the relevant calculations in the background and displays the Alpha and Beta parameter values to the user.
- A "User Guide" function, presenting the user with a document outlining the process of utilising the HBE-Python program.
- A "Proceed" function, which obtains all inputs from the main input window.
- A "Results" function, which obtains the inputs from the 'Inputs' spreadsheet and, in conjunction with the main input window parameters, performs all calculations associated with the HBET.
- A main function, where all functions are called from. The plotting of results was also included within the main function.

Instead of presenting each developed function, various components or main features of the program will be provided. Where possible, the functions are presented in code format. However, some functions contain numerous lines of code and thus, cannot be presented entirely. In these situations, pseudo-code is used to present the developed function/feature.

5.2.3 Graphical User Interface (GUI)

To ensure the program is as user-friendly as possible, a GUI was developed, since it is more intuitive to users than running programs from a command prompt. The GUI is presented on program start-up and allows the user to insert inputs related to the general system conditions and overarching factors which directly affect the results produced. These inputs were strategically chosen to allow the user the opportunity to recalculate their results directly from the main program window. The main window presented upon program start-up is shown in Figure 5.4 below.

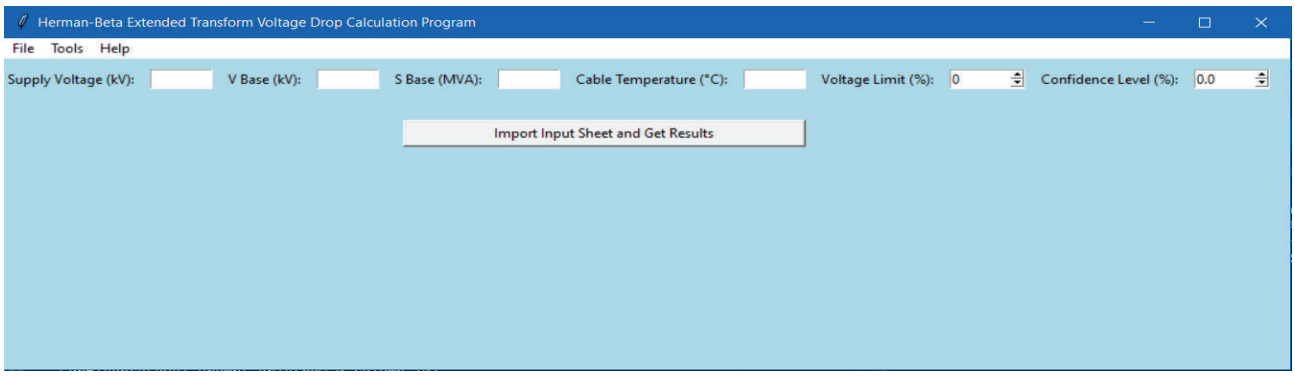


Figure 5.4: Program Main Window on Start-up

As seen in the figure above, the program contains a title, menu bar, input fields and a button which should be pressed once all input fields have been inserted. The program also contains the default window options that other Windows programs have in the top right corner, such as minimise, maximise and close window. The program window size has been defined in the code to ensure all widgets are visible upon start-up. The pseudo-code below outlines the steps taken to create the main program window.

Pseudo-code for Main Program Window Initialisation

```
1. Create empty start-up window
2. Define the window dimensions
3. Define window title
4. Create menu bar and configure background colours
5. For each sub-menu in menu bar, define seperate command functions
6. For 6 inputs:
        6.1 Create text label and place in required position
        6.2 Create input field (text box/spinbox)
7. Create button for the user to calculate results once all inputs have been inserted
```

The input fields have various methods of accepting user inputs. The supply voltage, base voltage, base apparent power and cable operating temperature can receive any user value. By utilising spin boxes for the voltage limit and confidence level inputs, the user is restricted to a pre-defined range and can increase or decrease the values inserted by using the associated arrows.. The list of available input values for the spin boxes are provided in Table 8 below.

Table 8: Selectable Ranges for Initial Inputs

Parameter	Range of Selectable Values
$\%\ Volt\ Drop\ Limit$	0 to 100
$\%\ Confidence\ Level$	0 to 100

The input parameters not listed in Table 8 were specifically designed to accept any values from the user to cater for the input of figures containing decimals. However, if a value entered in one or more of these fields is non-numerical, the program should not proceed. For this reason, error messages have been built into the program in the event that this occurs, and will be shown to the user when the “Import Input Sheet and Get Results” button has been pressed.

The following portion of code is part of the “Proceed” function and is used to present error messages to the user when one or more non-numerical inputs are entered on the initial input screen.

```
156     try:
157         while(ContinueLoop==0):
158             ContinueLoop=1
159             #Shows an error message if the values entered are in the incorrect format or non-integer values are received.
160             if((Vs_Entry.get()=='' or not(isinstance(float(Vs_Entry.get()),float))) or (Vbase_Entry.get()=='' or not(isin
161                 messagebox.showerror('Invalid Input Values','Error! Please enter integer values for all inputs')
162             else:
163                 ContinueLoop==1
164                 #Store the values obtained from the user inputs as simplified variables.
165                 Vs_Entered=Vs_Entry.get()
166                 Vbase_Entered=Vbase_Entry.get()
167                 Sbase_Entered=Sbase_Entry.get()
168                 Temp_Entered=Temp_Entry.get()
169                 VoltageLimit_Entered=VoltageLimit_Spin.get()
170                 Confidence_Level_Entered=ConfidenceLevel_Spin.get()
171                 SystemConfig_Entered=SystemConfig_Combo.get()
172                 Results()
173     except ValueError:
174         #Display error message if a non-numerical character has been inserted
175         messagebox.showerror('Incorrect values entered!','Error! Please enter numerical values only')
```

Figure 5.5: Code Used to Produce Error Messages When Incorrect Initial Inputs are Provided

As seen in the figure above, there are two instances of error messages being produced. The first, in Line 161, produces an error message when one or more input fields have been left blank or have received values that are not floats (Line 160 checks for all combinations of incorrect inputs but cannot be shown entirely due to the length of the line of code).

The second error message is shown to the user if a non-numerical input is received for one or more of the intial input parameters (Line 175). This error message is handled using 'try' and 'except' statements, since it allows the program to provide a predefined output instead of causing the program to crash, necessitating a program restart. As such, the user can read the error message while the program continues to run in the background. The user can then rectify the errors after closing the error message and proceed to the next step of the program.

The error message produced when non-numerical parameters are inserted as initial inputs is shown in Figure 5.6 below.

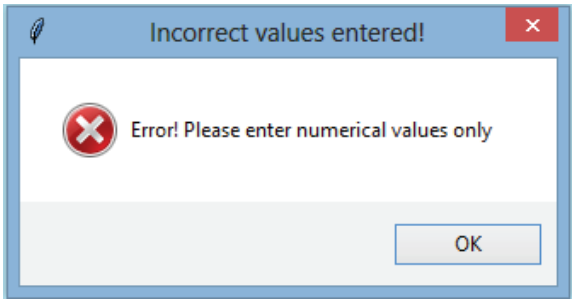


Figure 5.6: Error Message Shown When Non-numerical Values are Received for Initial Inputs

Once all values have been entered in the correct format, the "Import Input Sheet and Get Results" button should be pressed again. This will provide the user with the calculated results for the designed system. It is important to note that the initial inputs inserted in the main window works in conjunction with the 'Inputs' spreadsheet to perform the HBET calculations. Therefore, the user should ensure the spreadsheet is filled in correctly before running the HBE-Python program. The spreadsheet is discussed in the next sub-section.

While the user has the ability to enter any range of values for the voltage variation limit and the confidence level, it is assumed that the user will have the necessary knowledge required to select appropriate values for these parameters. For passive feeders, the typical values for the voltage limit and

confidence level is 10% and 95% respectively. For active feeders, typical values are 10% and 97.5% respectively.

5.2.4 The Inputs Spreadsheet

The majority of the inputs required from the user are inserted in an 'Inputs' spreadsheet. The spreadsheet approach was adopted to allow users to easily copy data from existing datasets and to duplicate parameters across nodes, as the parameters are often similar or identical. The spreadsheet requires the user to enter the following information:

- The conductor type connecting the nodes.
- The conductor length in metres.
- The system voltage level (LV or MV).
- The desired results plot (either voltage profile, mean voltage or standard deviation).
- The system layout, which is used to specify laterals in the system.
- The number of connections per phase for each node.
- The Beta PDF parameters, namely Alpha, Beta, and the circuit breaker rating (scaling factor) in Amps.
- The power factor.

The layout of the 'Inputs' spreadsheet is shown in Figure 5.7 below.

Conductor Properties									Load Properties							System Voltage	Plot Select (0 – Voltage Profile , 1 – Mean Voltage Plot, 2 – Standard Deviation Plot)	Feeder Layout
From node	To node	Conductor Type	k	T	Length [m]	Resistance at T1 [/km]	Reactance at T1 [/km]	Current Rating [A]	Alpha	Beta	C [A]	PF	Phase Loading			MV	1	[2 3 18;2 19 22;3 23 25;6 26 33]
													Red	White	Blue			
0	1	300mmAl	1	241	1000	0.0922	0.0470	335.00	49.5	49.5	18.42	0.857	1	1	1			
1	2	300mmAl	1	241	1000	0.4930	0.2512	335.00	49.5	49.5	15.56	0.914	1	1	1			
2	3	300mmAl	1	241	1000	0.3661	0.1864	335.00	49.5	49.5	22.78	0.832	1	1	1			
3	4	300mmAl	1	241	1000	0.3811	0.1941	335.00	49.5	49.5	10.6	0.894	1	1	1			

Figure 5.7: Layout of Inputs Spreadsheet

Although not shown in the figure above, the spreadsheet allows the user to enter the required Beta PDF parameters for generators as well. When using the spreadsheet, the user should first select the system voltage level, since the conductor types available for LV and MV systems differ. After the system voltage level is chosen, the associated conductor types are selectable from the dropdown menu available when clicking on cells in the 'Conductor Type' column.

Each conductor type has electrical parameters associated with it in a different tab in the spreadsheet, which gets pulled through to the main tab to automatically complete most of the columns in the Conductor Properties section. Further to this, the conductor section length is required. Thereafter, the user is required to specify the load and generator parameters. Finally, the user needs to specify the plot type desired and indicate the feeder layout.

The feeder layout is completed as follows:

- All laterals are inserted within square brackets and separated using semicolons i.e. [Lateral 1; Lateral 2; Lateral 3].
- Each lateral has three numbers associated with it, where the numbers are separated by a space. A lateral starting at node 15 and ending at node 20, which is connected to node 3 will be specified as [3 15 20].

- The starting node of any lateral cannot be less than 2. The spreadsheet assumes the first node is the source and hence, the information filled in the first row of the spreadsheet pertains to the second node.

Arrays are used throughout the program, since it is easy to manipulate data at specific positions within the array. It was decided that this capability would be useful for this application, since the array positions could be used to reference node numbers. The array functionality is used to determine the laterals in the user's system, based on the feeder layout input provided. The following portion of code is used to store the feeder layout input in an array.

```
#Obtain laterals from the Inputs spreadsheet
GetLaterals=sheet.cell_value(1,32)
#Create array from user's laterals input
arr = GetLaterals.split(';')
Lats=[]
for i in range(len(arr)):
    for j in range(len(arr[i])):
        try:
            if(j!=0):
                #Used to find node number containing more than one digit and replace the incorrect single digit
                if(arr[i][j]!=' ' and arr[i][j]!=']' and arr[i][j-1]!=' ' and arr[i][j-1]!='['):
                    print(arr[i][j-1])
                    print(arr[i][j])
                    del(Lats[-1])
                    Lats.append(int(str(arr[i][j-1])+str(arr[i][j])))
                else:
                    Lats.append(int(arr[i][j]))
            else:
                Lats.append(int(arr[i][j]))
        except ValueError:
            pass
```

Figure 5.8: Code Used to Obtain Laterals from Input Spreadsheet and Store it in an Array

Once the 'Inputs' spreadsheet has been completed, it should be saved by the user, since the developed program utilises the latest saved version of the spreadsheet when calculating results.

The input values required are readily available to network planners at the design stage, with the exception being the Beta parameters (Alpha and Beta). For this reason, it was decided that Beta parameter calculators should be included in the program.

5.2.5 Beta Parameter Calculators

The Herman-Beta method uses load currents modelled as Beta PDFs to perform probabilistic load flow analysis. The Beta PDF has the ability to be skewed negatively or positively and has a finite base ranging between 0 and 1 [17]. The Beta PDF being limited to values between 0 and 1 is especially useful for the HBET, where the per-unit (p.u) system is utilised. As such, the Beta PDF and p.u system ranges align and therefore, greatly simplifies calculations.

The Beta PDF represents the probability distribution of probabilities [41]. Therefore, the Beta PDF can be used to determine the probability of failure or success of events. As an example, to determine the probability that an event would have a success rate of at least 70%, the area under the Beta PDF to the right of the chosen success rate (70% in this case, indicated as 0.7 on the x-axis of the Beta PDF graph) would need to be integrated. The Beta PDF is represented using the following equation [41].

$$Beta\ PDF = \frac{x^{\alpha-1}(1-x)^{\beta-1}}{\beta(\alpha,\beta)} \tag{5.1}$$

Where,

$$\beta(\alpha,\beta) = \frac{\Gamma(\alpha)\Gamma(\beta)}{\Gamma(\alpha+\beta)} \tag{5.2}$$

In the equations above, x is the confidence level and Γ is the Gamma function. Examples of Beta PDF functions with varying Alpha and Beta values are indicated in Figure 5.9 below.

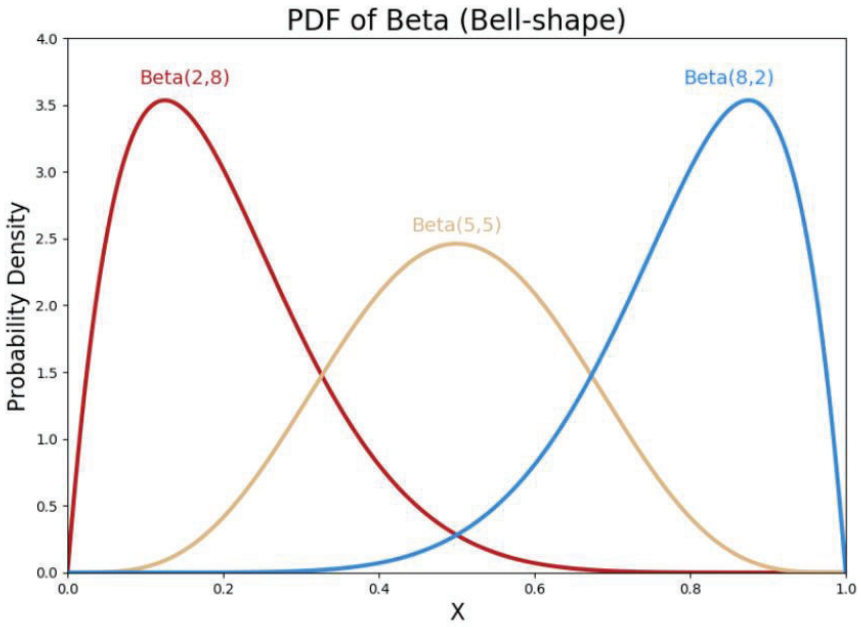


Figure 5.9: Effect of Varying Alpha and Beta Values on Beta PDF Graphs [41]

The Beta PDF parameters, Alpha (α) and Beta (β), can be found from the load data available to network planners. The parameters can be calculated from two sets of inputs, depending on the information available to the network planner. The parameters can be calculated using the following:

- ADMD, voltage and circuit breaker rating – used when fixed loads are available to the network planner.
- Standard deviation, mean and circuit breaker rating – Used when the network planner has a load distribution available.

The following equations are used in the program to determine the Beta PDF parameters using ADMD, voltage and the circuit breaker rating [17].

$$\propto = \frac{C_b V_s - 1000D\left({C_v}^2 + 1\right)}{C_b V_s {C_v}^2} \tag{5.3}$$

$$\beta = \frac{\propto (C_b V_s - 1000D)}{1000D} \tag{5.4}$$

Where,

$$C_v = 1.0738D^{-0.3274} \tag{5.5}$$

In the equations above, C_b is the circuit breaker rating, V_s is the supply voltage, C_v is the coefficient of variation and D is the ADMD value in kVA. Equations 5.6 and 5.7 below are used to determine the Beta PDF parameters using the standard deviation and mean of the load distribution as well as the circuit breaker rating [17].

$$\propto = \frac{\mu(C_b\mu - \mu^2 - \sigma^2)}{C_b\sigma^2} \tag{5.6}$$

$$\beta = \frac{(C_b - \mu)(C_b\mu - \mu^2 - \sigma^2)}{C_b\sigma^2} \tag{5.7}$$

In the equations above, μ is the mean of the load distribution, σ is the standard deviation of the load distribution and C_b is the circuit breaker rating.

Since the Beta PDF parameters are required as inputs at each node, it was decided that these two calculators will be incorporated into the HBE-Python tool. Both Beta parameter calculators can be found under the “Tools” dropdown in the menubar. The location of the calculators is shown in Figure 5.10 below.

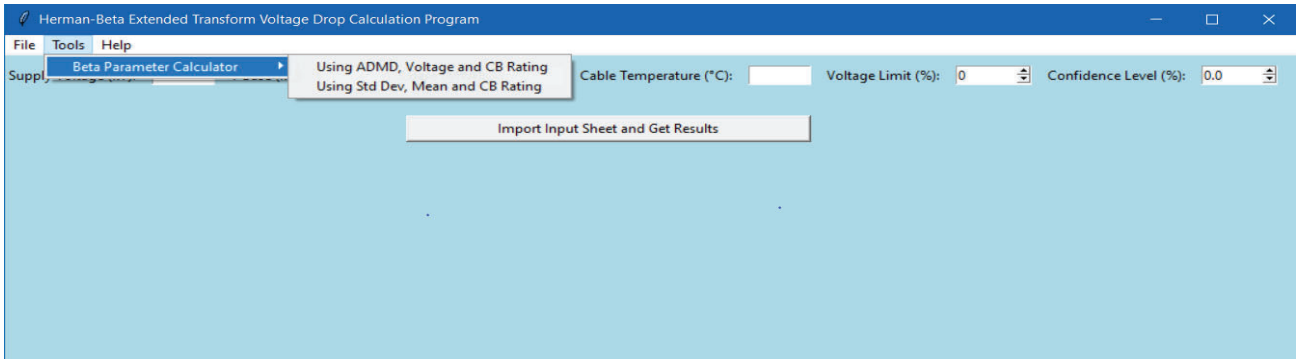


Figure 5.10: Locating the Beta Parameter Calculators in the HBE-Python Program

Clicking on either of the Beta parameter calculators will produce a pop-up window, requiring the user to enter the relevant information to determine the values of Alpha and Beta. One of the combinations of data the user requires to determine the Beta parameters is the ADMD, voltage and scaling factor. The second combination that can be used is the scaling factor, standard deviation and mean of the load profile.

Two functions were created for each Beta parameter calculator, one to create the pop-up window to accept user inputs and the other to perform the associated calculations. The code for the functions associated with each Beta parameter calculator is shown in Figure 5.11 and Figure 5.12 below.

```
#Create a new pop-up window to allow users to insert values to determine beta parameters based on ADMD, supply voltage and circuit breaker rating
def BetaCalculator_Option1():
    #Define global variables to be used in the Calculate_Option1 function
    global ADMD_Entry,CB_Entry,SupplyVoltage_Entry,BetaCalculator1_Window
    #Create a top-level window above the main program window
    BetaCalculator1_Window=Toplevel(window)
    BetaCalculator1_Window.title("Beta Parameter Calculator")
    BetaCalculator1_Window.geometry('450x350')
    #Asks the user to insert the supply voltage, ADMD and CB rating
    SupplyVoltage_Prompt=Label(BetaCalculator1_Window,text="Supply Voltage (V):",width=20).grid(row=1,column=0,padx=10,pady=10)
    SupplyVoltage_Entry=Entry(BetaCalculator1_Window,width=40)
    SupplyVoltage_Entry.grid(row=1,column=1,columnspan=2,padx=10,pady=10)
    ADMD_Prompt=Label(BetaCalculator1_Window,text="ADMD (kVA):",width=20).grid(row=2,column=0,padx=10,pady=10)
    ADMD_Entry=Entry(BetaCalculator1_Window,width=40)
    ADMD_Entry.grid(row=2,column=1,columnspan=2,padx=10,pady=10)
    CB_Prompt=Label(BetaCalculator1_Window,text="Circuit Breaker Rating (A):",width=20).grid(row=3,column=0,padx=10,pady=10)
    CB_Entry=Entry(BetaCalculator1_Window,width=40)
    CB_Entry.grid(row=3,column=1,columnspan=2,padx=10,pady=10)
    Calculate_Paramaters=Button(BetaCalculator1_Window,text='Calculate Beta Parameters', command=Calculate_Option1).grid(row=5,column=1,columnspan=2,padx=10,pady=10)

#Function used to calculate Beta parameters using ADMD, CB rating and supply voltage
def Calculate_Option1():
    global ADMD_Entry,CB_Entry,SupplyVoltage_Entry,BetaCalculator1_Window
    try:
        #If all values entered are in the correct format, calculate the Beta PDF parameters
        if((float(ADMD_Entry.get()) and float(CB_Entry.get())) and float(SupplyVoltage_Entry.get())):
            C_V=1.0738*(float(ADMD_Entry.get())**(-0.3274))
            CB=float(CB_Entry.get())
            Vs=float(SupplyVoltage_Entry.get())
            ADMD=float(ADMD_Entry.get())
            Alpha=((CB*Vs-(1000*ADMD*((C_V**2)+1)))/(CB*Vs*(C_V**2))
            Beta=Alpha*(CB*Vs-(1000*ADMD))/(ADMD*1000)
            Font=tkFont.Font(family="Lucida Grande", size=11)
            Calculated_Alpha=Label(BetaCalculator1_Window,text="The calculated value for Alpha is :  "+str("{:.3f}".format(Alpha)),font=Font,width=40,fg='blue').grid(row=6,column=0,columnspan=4,pady=20)
            Calculated_Beta=Label(BetaCalculator1_Window,text="The calculated value for Beta is :  "+str("{:.3f}".format(Beta)),font=Font,width=40,fg='blue').grid(row=7,column=0,columnspan=4,padx=10,pady=10)
    #Show error message if values have been entered in the incorrect format
    except ValueError:
        messagebox.showerror('Incorrect values entered!','Error! Please enter numerical values for ADMD, supply voltage and circuit breaker rating')
        #Keeps the Beta parameter calculator window displayed after the error message window is closed.
        BetaCalculator1_Window.deiconify()
```

Figure 5.11: Code Used to Create Beta Parameter Calculator 1

```
#Create a new pop-up window to allow users to insert values to determine beta parameters based on the load's mean, standard deviation and the circuit breaker rating
def BetaCalculator_Option2():
    global Mean_Entry,CB_Entry,StdDev_Entry,BetaCalculator2_Window
    BetaCalculator2_Window=Toplevel(window)
    BetaCalculator2_Window.title("Beta Parameter Calculator")
    BetaCalculator2_Window.geometry('450x350')
    #Allows the user to enter the mean, standard deviation and CB rating to calculate Beta PDF parameters
    Mean_Prompt=Label(BetaCalculator2_Window,text="Mean:",width=20).grid(row=1,column=0,padx=10,pady=10)
    Mean_Entry=Entry(BetaCalculator2_Window,width=40)
    Mean_Entry.grid(row=1,column=1,columnspan=2,padx=10,pady=10)
    StdDev_Prompt=Label(BetaCalculator2_Window,text="Standard Deviation:",width=20).grid(row=2,column=0,padx=10,pady=10)
    StdDev_Entry=Entry(BetaCalculator2_Window,width=40)
    StdDev_Entry.grid(row=2,column=1,columnspan=2,padx=10,pady=10)
    CB_Prompt=Label(BetaCalculator2_Window,text="Circuit Breaker Rating (A):",width=20).grid(row=3,column=0,padx=10,pady=10)
    CB_Entry=Entry(BetaCalculator2_Window,width=40)
    CB_Entry.grid(row=3,column=1,columnspan=2,padx=10,pady=10)
    Calculate_Paramaters=Button(BetaCalculator2_Window,text='Calculate Beta Parameters', command=Calculate_Option2).grid(row=5,column=1,columnspan=2,padx=10,pady=10)

#Function used to calculate Beta parameters using load's mean, standard deviation and the CB rating
def Calculate_Option2():
    global Mean_Entry,CB_Entry,StdDev_Entry,BetaCalculator2_Window
    try:
        #If all values entered are in the correct format, perform the calculation
        if((float(Mean_Entry.get()) and float(StdDev_Entry.get())) and float(CB_Entry.get())):
            CB=float(CB_Entry.get())
            Mean=float(Mean_Entry.get())
            StdDev=float(StdDev_Entry.get())
            Alpha=Mean*((CB*Mean-Mean**2-StdDev**2)/CB/StdDev**2)
            Beta=(CB-Mean)*((CB*Mean-Mean**2-StdDev**2)/CB/StdDev**2)
            Font=tkFont.Font(family="Lucida Grande", size=11)
            Calculated_Alpha=Label(BetaCalculator2_Window,text="The calculated value for Alpha is :  "+str("{:.3f}".format(Alpha)),font=Font,width=40,fg='blue').grid(row=6,column=0,columnspan=4,pady=20)
            Calculated_Beta=Label(BetaCalculator2_Window,text="The calculated value for Beta is :  "+str("{:.3f}".format(Beta)),font=Font,width=40,fg='blue').grid(row=7,column=0,columnspan=4,padx=10,pady=10)
    #Show error message if values have been entered in the incorrect format
    except ValueError:
        messagebox.showerror('Incorrect values entered!','Error! Please enter numerical values for the mean, standard deviation and circuit breaker rating')
        #Keeps the window displayed after the error message window is closed.
        BetaCalculator2_Window.deiconify()
```

Figure 5.12: Code Used to Create Beta Parameter Calculator 2

The layouts of the two Beta parameter calculators are shown in Figure 5.13 and Figure 5.14 below.

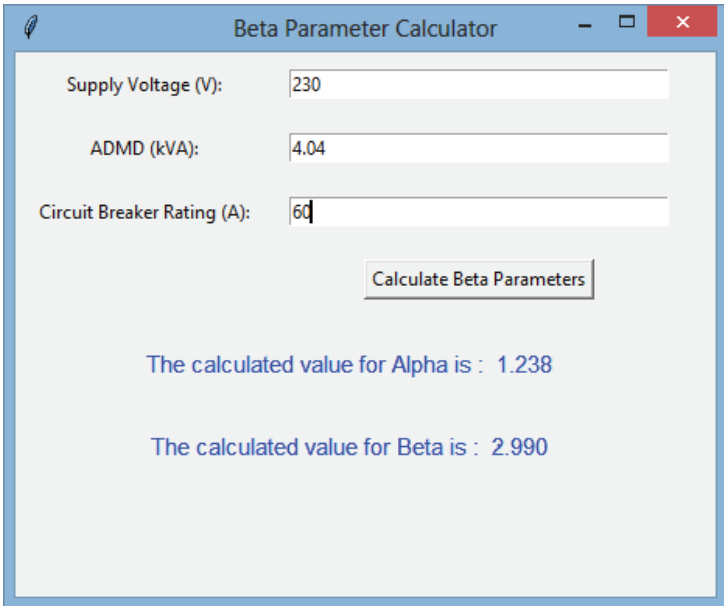


Figure 5.13: Beta Parameter Tool Using Supply Voltage, ADMD Scaling Factor

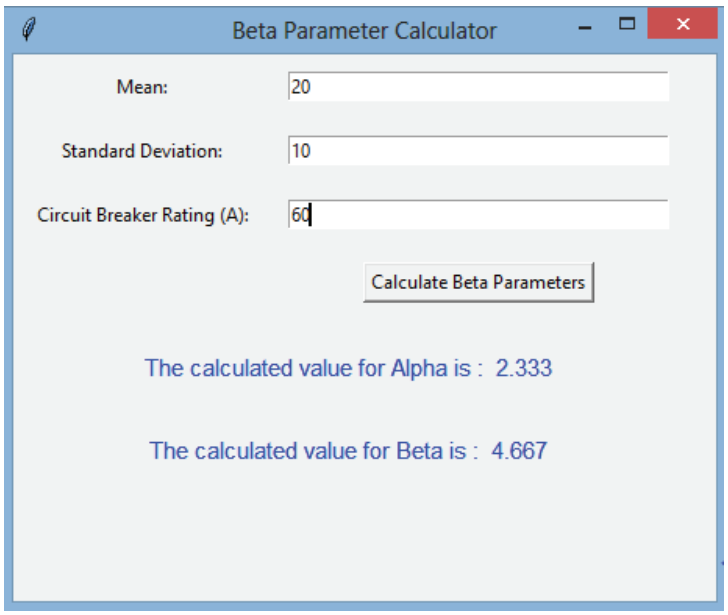


Figure 5.14: Beta Parameter Tool Using Mean, Standard Deviation and Scaling Factor

As seen in the layout figures above, the user has the option of inserting any value in the input fields. However, since the calculators will not produce any output if non-numerical inputs are inserted, an error message will once again be utilised to prompt the user to only insert numerical values. The error message resembles the pop-up window depicted in Figure 5.6. Once all values entered are in the required format and the 'Calculate Beta Parameters' button is pressed, the Beta parameters will be provided to the user in blue text, as depicted in the figures above. Once the user has determined all the required Beta parameters for his system, the calculator pop-up window can be closed.

5.2.6 Adding Cable Types to the Default Library

Within the 'Inputs' spreadsheet, the 'LV Conductors' and 'MV Conductors' tabs contain the default conductor types and their associated electrical parameters. The default cable types for LV and MV systems are shown in Table 9.

Table 9: List of Default Conductor Types

System Type	Cable Type
LV	ABC 25 mm^2 Al French std XLPE
	ABC 35 mm^2 Al French std XLPE
	ABC 50 mm^2 Al French std XLPE
	ABC 70 mm^2 Al French std XLPE
	ABC 95 mm^2 Al French std XLPE
	AIRDAC 10mm^2 4C Cu XLPE
	35 mm^2 4C Cu XLPE
	50 mm^2 4C Cu XLPE
	70 mm^2 4C Cu XLPE
	95 mm^2 4C Cu XLPE
MV	25 mm^2 3C Cu XLPE
	35 mm^2 3C Al XLPE
	50 mm^2 3C Al XLPE
	70 mm^2 3C Al XLPE
	95 mm^2 3C Cu XLPE
	120 mm^2 3C Al XLPE

	185 mm^2 3C Cu XLPE
	300 mm^2 3C Al XLPE

The 'LV Conductors' tab contains the same information as shown in the conductor section in Figure 5.1 and will therefore not be shown again. The 'MV Conductors' tab was populated using information from a popular MV cable manufacturer's datasheets. The information contained in the 'MV Conductors' tab is shown in Table 10.

Table 10: Electrical Information Contained in the MV Conductors Tab in the Inputs Spreadsheet

Code	R/km@t1	X/km@t1	T	Current Rating (A)	Description
25mmCu	0.72	0.10	228	105	25mm² 3C Cu XLPE
95mmCu	0.19	0.05	228	144	95mm² 3C Cu XLPE
185mmCu	0.10	0.07	228	183	185mm² 3C Cu XLPE
35mmAl	0.87	0.09	228	228	35mm² 3C Al XLPE
50mmAl	0.64	0.09	228	277	50mm² 3C Al XLPE
70mmAl	0.45	0.08	241	50	70mm² 3C Al XLPE
120mmAl	0.26	0.08	241	170	120mm² 3C Al XLPE
300mmAl	0.10	0.07	241	200	300mm² 3C Al XLPE

The final tab in the 'Inputs' spreadsheet is a 'Lists' tab. This is where the dropdown menus for the selection of the conductor types on the main tab are created. To add a conductor to the library, the user should add the codename for the new conductor in either the LV or MV column in the 'Lists' tab. This will add the new conductor to the dropdown menu selectable from the main tab for the specified voltage level.

The final step is to add the full set of electrical parameters (as in Table 10) associated with that codename to the tab associated with that voltage level. For instance, adding a new conductor type under the MV column in the 'Lists' tab will require the user to enter the electrical parameters for that conductor in the 'MV Conductors' tab. Once all these steps have been followed, the new conductor type can be utilised in the user's system.

5.2.7 Results W.indow

Once the initial inputs have been inserted in the program's main window and the 'Inputs' spreadsheet has been completed, the user can click on the 'Import Input Sheet and Get Results' button to view the results for the modelled system. The following pseudo-code (describing the "Results" function) provides a very high-level indication of the processes running in the background to determine the HBET results.

Pseudo-code for "Results" function

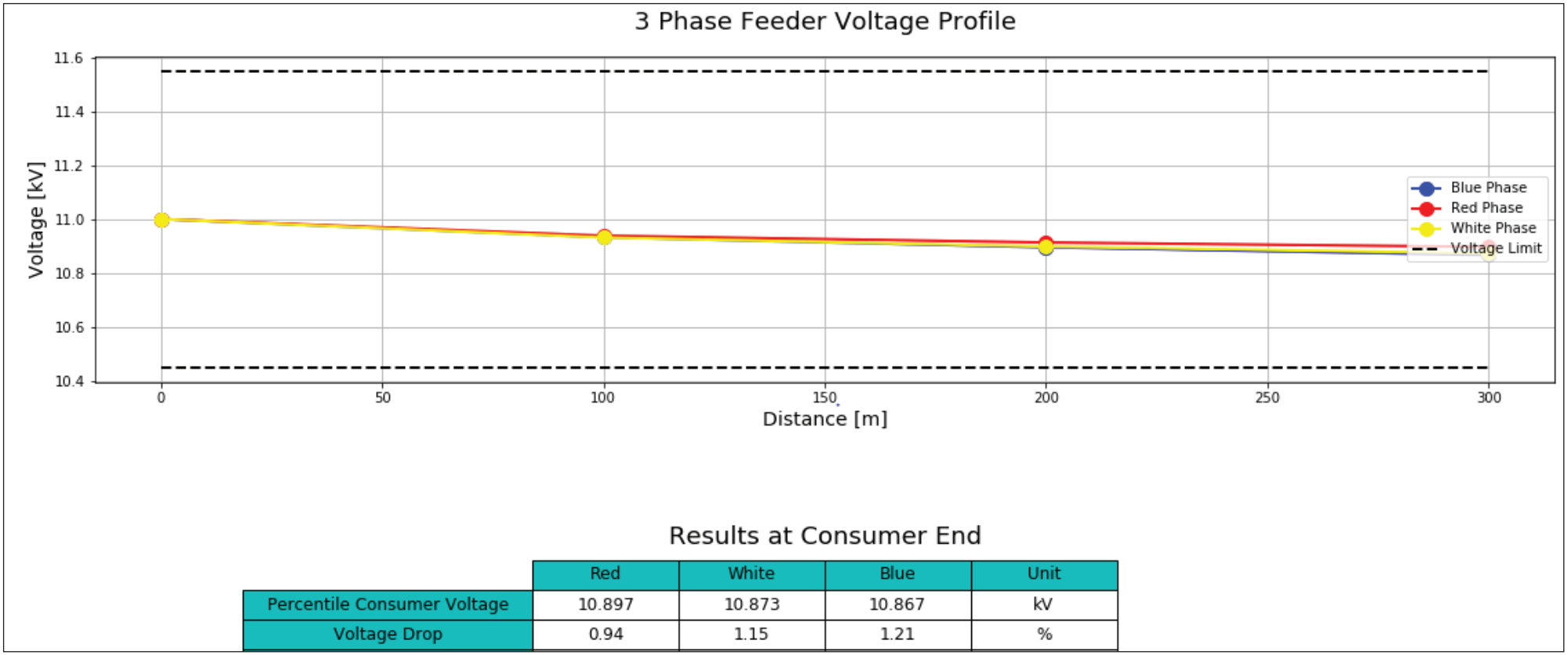
```
1. Initialise all arrays and variables
2. Get laterals from 'Inputs' spreadsheet and store in an array
3. for number of laterals in lateral array:
        3.1 Reset all variables and arrays
        3.2 Lump inputs back to previous nodes for superposition (for lateral only)
        3.3 Perform initial HBET calculations
        3.4 Find nodes not within current lateral and replace those array values with zeroes
        3.5 Determine the sum of moments at each node
        3.6 Perform remaining HBET voltage calculations per phase at each node
4. Sort node output values sequentially, according to original feeder node numbering
```

Provided all inputs have been inserted in the correct format, a new pop-up window will be shown on the user's PC, indicating the feeder voltage results graphically as well as the consumer-end results in tabular form. The pseudo-code below outlines the steps taken to create the results window.

Pseudo-code for Creation of Results Window

```
1. Create and configure two subplots within the plot window; one graph and one table
2. Create row and column headings for table and specify the units associated with each value
3. Set colour, justification, font and location of table
4. Identify user's desired plot from Inputs spreadsheet
        4.1 if selected plot = Percentile Voltage plot
                4.1.1 Determine upper and lower voltage limits based on user inputs
                4.1.2 Set plot title to "3 Phase Feeder Voltage Profile"
                4.1.3 Label axes accordingly
                4.1.4 Create traces for all phases based on calculated results
                4.1.5 Create traces for voltage limit levels
        4.2 else if selected plot = Mean Voltage plot
                4.2.1 Set plot title to "Mean Voltage Plot"
                4.2.2 Label axes accordingly
                4.2.3 Create traces for all phases based on calculated results
        4.3 else
                4.3.1 Set plot title to "Voltage Standard Deviation Plot"
                4.3.2 Label axes accordingly
                4.3.3 Create traces for all phases based on calculated results
5. Show plot
```

An example of the results obtained from the program is indicated in Figure 5.15.



Results at Consumer End

	Red	White	Blue	Unit
Percentile Consumer Voltage	10.897	10.873	10.867	kV
Voltage Drop	0.94	1.15	1.21	%

Figure 5.15: Results Obtained from the HBE-Python Program

As seen in the figure above, results are presented graphically and in tabular format. The graph indicates the consumer voltage for the red, white and blue phases at each node along the cumulative feeder distance. For instance, if a load is present at 10m and another load is present 25m away from the first, the graph will indicate the consumer voltage per phase at 10m and at 35m. Node positions are indicated using circular markers on all traces.

In addition to the consumer voltage per phase at each node, dotted lines are used to represent voltage limits, wherefrom the user can easily determine whether the designed system will exceed the predefined voltage limit. A legend has also been added to the graph. The axes have been labelled, along with a unit of measure, to provide clarity to the user regarding the information presented in the graph. The graph has also been given a title, since the user may wish to use the figure produced in reports or presentations. Gridlines were added to the graph, to enable the user to easily estimate values by inspection.

Most of the results pop-up window was produced using an installable python wheel (pre-written piece of code used to perform a specific set of functions) called Matplotlib. Matplotlib is used to easily manipulate data to produce graphs, tables and charts. By using Matplotlib, the pop-up window showing the results automatically contains some useful features. These include, but are not limited to:

- A save feature, allowing the user to save the results as an image of the desired format. Results cannot be exported to Excel or text files for further manipulation and needs to be analysed directly in the results window.
- A zoom feature, allowing the user to zoom into a user-defined rectangle, by clicking and dragging the mouse across the desired area.
- A reset view feature, allowing the user to return to the default view.
- A pan feature, allowing the user to move along the trace area.
- A configure subplots feature, allowing the user to change the layout of the graph and table in relation to one another.
- A co-ordinate identifier feature, showing the user the values of the x and y co-ordinates when hovering on a certain point on the graph.

Below the graphical representation of the results, a table of results can be found. It should be noted that the table is merely a summary of the results at the consumer end and does not provide the level of detail that the graph does. The result table can be used to quickly determine whether the system has been designed to adhere to voltage variation limits.

5.2.8 Additional Menu Bar Functions

To provide a step-by-step method of utilising the HBE-Python program, a user guide was written. The user guide can be found within the program under the 'Help' menu. The code used to incorporate the user guide in the program is shown in Figure 5.16 below.

```
140    #Opens the user guide
141    def User_Guide():
142        os.system('Guide.pdf')
143
144    Help_DropdownMenu = Menu(menubar, tearoff=0)
145    Help_DropdownMenu.add_command(label="User Guide", command=User_Guide)
146    menubar.add_cascade(label="Help", menu=Help_DropdownMenu)
```

Figure 5.16: Code Used to Incorporate User Guide

Furthermore, an “About” function can be found under the “File” dropdown menu, which has been created to provide the user with some additional background information about the program. The “About” window and code used to incorporate the window are shown in Figure 5.17 and Figure 5.18 respectively.

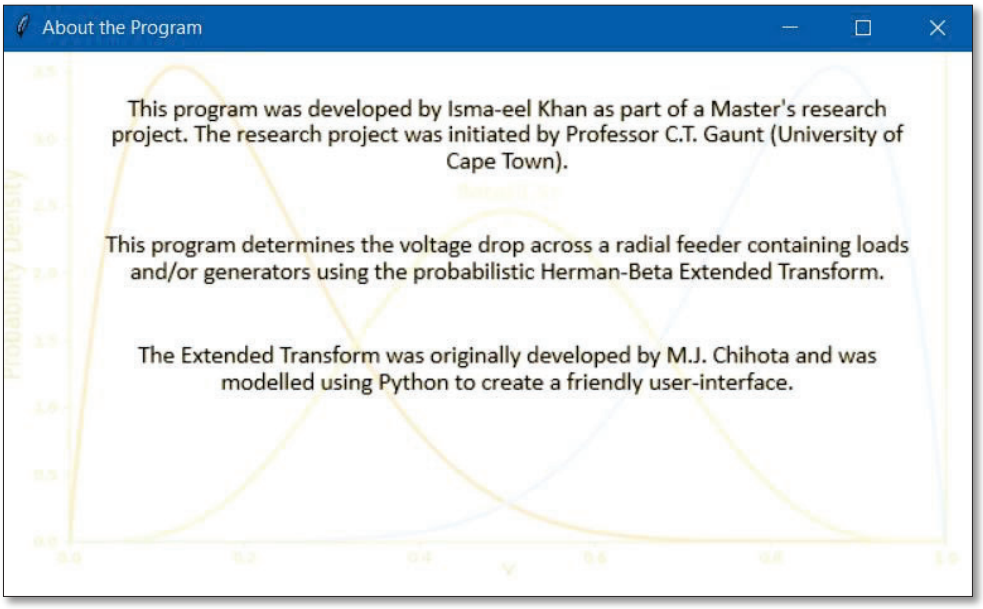


Figure 5.17: About Window Providing Background Information of the HBE-Python Tool

```
42  #Gives a brief desciption of the program and its development when the 'About' option is selected from the menubar
43  def About():
44      About_Window=Toplevel(window,bg='white')
45      About_Window.title("About the Program")
46      About_Window.geometry('620x350')
47      #Displays the background image with text in the About pop-up window.
48      About_image=PhotoImage(file='AboutBackgroundwithText.png')
49      About_label=Label(About_Window, image=About_image)
50      About_label.place(x=0, y=0, relwidth=1, relheight=1)
51      About_label.image = About_image
```

Figure 5.18: Code Used to Incorporate the About Window

Under the same dropdown menu, there is also an “Exit” function, which terminates the program if the user confirms that they want to close the program. The code used to create this function is shown in Figure 5.19. The pop-up window displayed when the user selects the “Quit” option is shown in Figure 5.20.

```
54  #Function used to exit the program
55  def Exit_Program():
56      #Confirms that the user is sure about exiting the program before doing so
57      QuitProgram=messagebox.askquestion("Quit Program","Are you sure you want to quit the program?")
58      if(QuitProgram=='yes'):
59          window.destroy()
```

Figure 5.19: Code Used to Create Exit Function

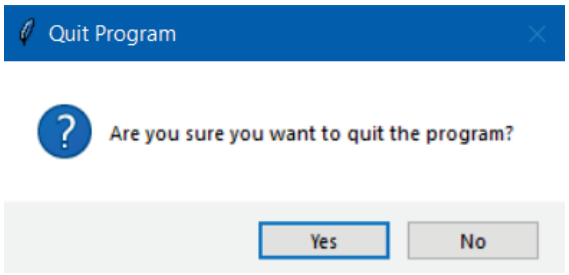


Figure 5.20: Exit Pop-up Window

5.3 Chapter Summary

This chapter discussed the detailed designs of both network planning tools, and can be summarised as follows:

- The accuracy of the HB-DG tool was determined in Section 5.1.1 by comparing the results obtained with those obtained using the original Herman-Beta spreadsheet tool. The results produced by both spreadsheets were identical, indicating the high degree of accuracy with which the HB-DG tool was implemented.
- The HBE-Excel tool was developed from the HB-DG tool. The aesthetics and operation of the tool remained intact. However, additional inputs were allowed for and a number of equations needed to be added, as stipulated in Section 5.1.2.
- The cable types for the two system voltage types (LV and MV) are not identical in the developed program.
- In addition to the percentile voltage plot, a mean voltage plot and a voltage standard deviation plot was added as outputs to the developed program. This was done since the HBE-MATLAB tool provided results using these two plots. The additional plots will facilitate the comparison of results produced by the HBE-Python and HBE-MATLAB programs.
- A set of initial inputs is required in the HBE-Python tool's main window, as outlined in Section 5.2.3
- In Section 5.2.4, an 'Inputs' spreadsheet is introduced, which allows for the easy setup of the user's system. The spreadsheet will be utilised in conjunction with the initial inputs to provide the calculated results.
- Error handling has been implemented in the HBE-Python tool to prevent the use of incorrect inputs to determine results.
- The user is required to enter a number of inputs, such as the number of customers connected to each phase at each node, cable types, ADMD, circuit breaker ratings, power factors and cable lengths. These parameters are usually readily available to network planners at the design stage.
- A user guide has been written to outline the process of utilising the program. It is also intended to provide answers to questions users may have, since no online support will be available for the program.
- A few key features were developed to simplify the development process and to facilitate easier debugging.

6. Results and Discussion

This chapter presents the results obtained from the HBE-Excel and HBE-Python programs and discusses the accuracy of these results compared to those obtained from the HBE-MATLAB tool. Four scenarios will be tested. The scenarios consist of a 12-bus passive feeder without laterals, a 12-bus active feeder without laterals, a 33-bus passive feeder with laterals and a 33-bus active feeder with laterals. All systems were originally simulated by Chihota in [2]. The system data used in the simulations was obtained in [2] and is shown in Appendix B for ease of reference.

6.1 12-bus Passive Feeder without Laterals

The first system that will be tested is a 12-bus system with no laterals. For a passive feeder, the mean voltage or node voltage is expected to decrease with each successive node. However, slight increases are possible, depending on the loading of the phases. The feeder layout for this system is shown in Figure 6.1 below.

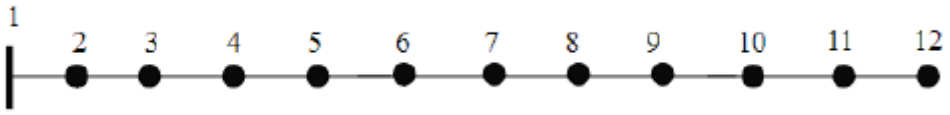


Figure 6.1: 12-bus Feeder Layout (Adapted from [2])

Although not explicitly shown in the figure, all nodes have loads attached to them. The system voltage is set to 11kV and a voltage scaling factor of 1.8 is used. The Alpha and Beta values for all nodes are set to 1,5 and 5 respectively, whereas the scaling factor and power factor varies per node. It should also be noted that k_r and k_x is equal to 1 for all scenarios. For a full set of data inputs used to run these simulations, refer to Appendix B .

Since the HBET is an analytic-probabilistic approach, it is necessary to provide the user with the percentile voltage plot, which is dependant on the design risk/confidence level that the user inserts. Therefore, the results obtained from the three programs for the percentile voltages at each node will be compared, to ensure accurate results are obtained.

Since a passive feeder is being simulated in this scenario, a confidence level of 95% (or a risk of 5%) was used in the determination of the percentile voltages. The results obtained from the HBE-Python tool are indicated in Figure 6.2 below.

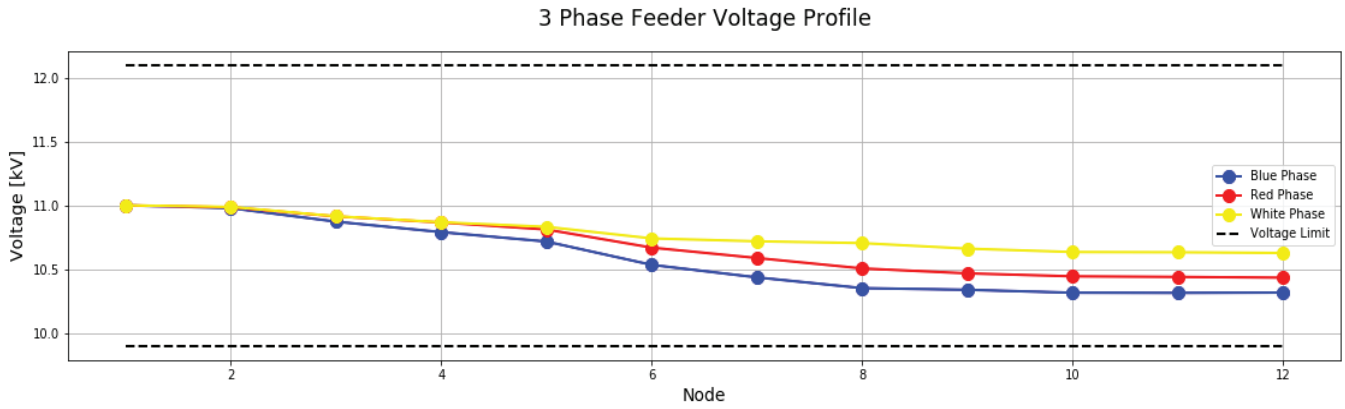


Figure 6.2: Percentile Voltage Plot for 12-bus Passive Feeder

As seen in the graph above, all three phases adhere to the voltage drop limit of 10% for the confidence level utilised. The comparison of results between the HBE-Python and HBE-MATLAB programs is provided later in this subsection, in Table 11.

In addition to the percentile voltage plots, mean voltage and voltage standard deviation plots will be utilised to compare the results obtained. The mean voltage and standard deviation plots obtained from the HBE-Python program are shown in Figure 6.3 and Figure 6.4 respectively.

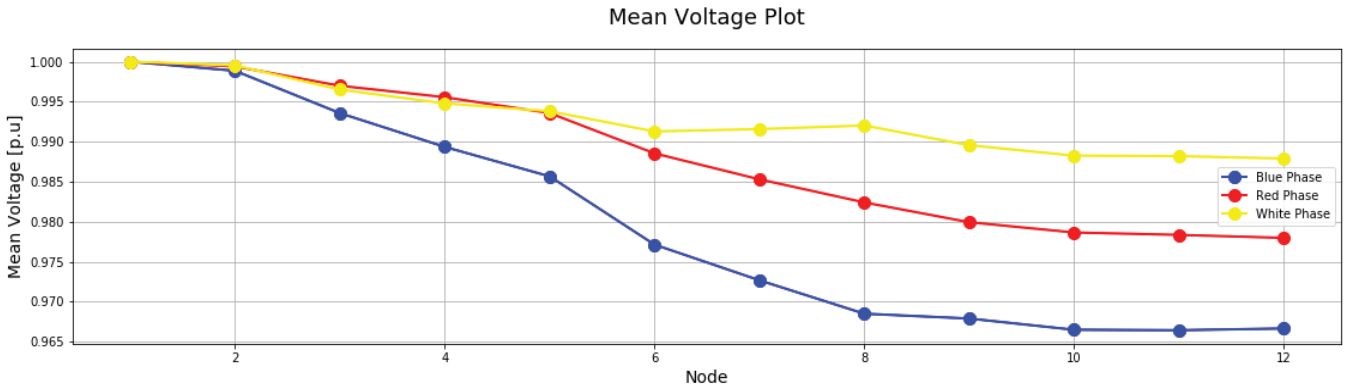


Figure 6.3: Mean Voltage Plot for 12-bus Passive Feeder

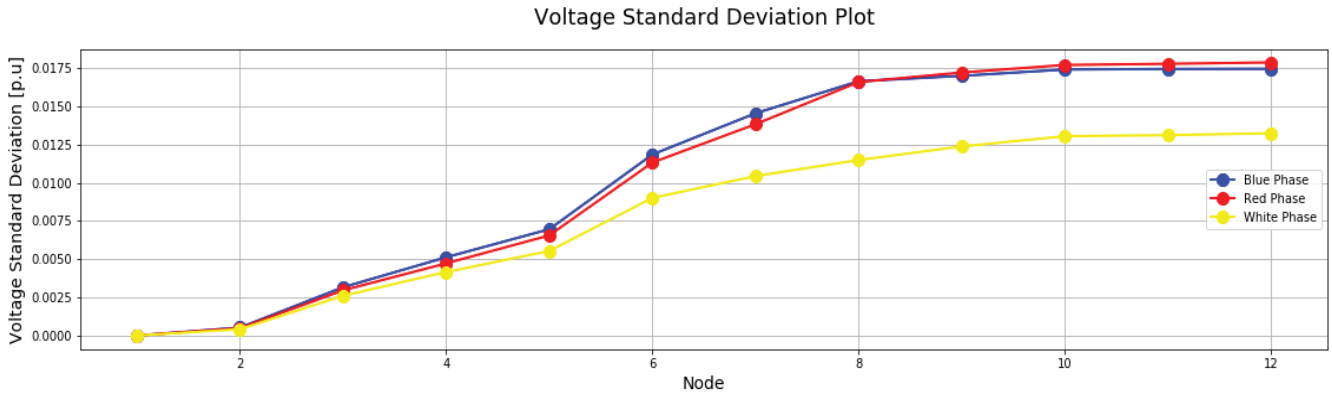


Figure 6.4: Voltage Standard Deviation Plot for 12-bus Passive Feeder

As seen in the figures above, the mean voltages for all phases generally decrease with an increase in distance from the source, whereas the standard deviations increase along the length of the feeder. To determine the accuracy of the results obtained, the results of the three programs need to be compared. The results are provided in Table 11 below.

Table 11: Node Results for 12-bus Passive Feeder

Node and phase	Percentile voltage (p.u)			Mean voltage (p.u)			Standard Deviation (p.u)		
	HBE-Python	HBE-MATLAB	HBE-Excel	HBE-Python	HBE-MATLAB	HBE-Excel	HBE-Python	HBE-MATLAB	HBE-Excel
2A	0.5548	0.5548	0.5554	0.9994	0.9994	0.9999	0.0004	0.0004	0.0001
2B	0.5549	0.5549	0.5556	0.9995	0.9995	1.0001	0.0004	0.0004	0.00005
2C	0.5544	0.5544	0.5553	0.9989	0.9989	0.9998	0.0005	0.0005	0.0001
3A	0.5512	0.5512	0.5545	0.9970	0.9970	0.9993	0.0030	0.0030	0.0007
3B	0.5512	0.5512	0.5546	0.9965	0.9965	0.9993	0.0026	0.0026	0.0006
3C	0.5491	0.5491	0.5555	0.9936	0.9936	1.0003	0.0032	0.0032	0.0003
4A	0.5488	0.5488	0.5550	0.9956	0.9956	1.0005	0.0047	0.0047	0.0009
4B	0.5489	0.5489	0.5523	0.9948	0.9948	0.9972	0.0042	0.0042	0.0018

4C	0.5450	0.5450	0.5531	0.9893	0.9893	0.9986	0.0051	0.0051	0.0018
5A	0.5460	0.5460	0.5538	0.9936	0.9936	0.9993	0.0065	0.0065	0.0015
5B	0.5471	0.5471	0.5528	0.9938	0.9938	0.9981	0.0055	0.0055	0.0019
5C	0.5412	0.5412	0.5521	0.9857	0.9857	0.9973	0.0070	0.0070	0.0021
6A	0.5388	0.5388	0.5513	0.9885	0.9885	0.9967	0.0113	0.0113	0.0027
6B	0.5425	0.5425	0.5505	0.9913	0.9913	0.9958	0.0090	0.0090	0.0030
6C	0.5320	0.5320	0.5529	0.9771	0.9771	0.9990	0.0118	0.0118	0.0023
7A	0.5347	0.5347	0.5532	0.9853	0.9853	1.0022	0.0138	0.0138	0.0039
7B	0.5413	0.5413	0.5416	0.9916	0.9916	0.9887	0.0104	0.0104	0.0084
7C	0.5271	0.5271	0.5426	0.9727	0.9727	0.9895	0.0145	0.0145	0.0078
8A	0.5306	0.5306	0.5355	0.9824	0.9824	0.9898	0.0166	0.0166	0.0157
8B	0.5406	0.5406	0.5458	0.9920	0.9920	0.9994	0.0115	0.0115	0.0102
8C	0.5228	0.5228	0.5243	0.9685	0.9685	0.9703	0.0166	0.0166	0.0162
9A	0.5287	0.5287	0.5323	0.9799	0.9799	0.9850	0.0172	0.0172	0.0163
9B	0.5384	0.5384	0.5425	0.9896	0.9896	0.9947	0.0124	0.0124	0.0111
9C	0.5222	0.5222	0.5259	0.9679	0.9679	0.9734	0.0170	0.0170	0.0162
10A	0.5275	0.5275	0.5340	0.9786	0.9786	0.9882	0.0177	0.0177	0.0164
10B	0.5371	0.5371	0.5395	0.9882	0.9882	0.9909	0.0130	0.0130	0.0120
10C	0.5210	0.5210	0.5223	0.9665	0.9665	0.9678	0.0174	0.0174	0.0168
11A	0.5273	0.5273	0.5309	0.9783	0.9783	0.9837	0.0178	0.0178	0.0170
11B	0.5370	0.5370	0.5412	0.9882	0.9882	0.9941	0.0131	0.0131	0.0121
11C	0.5209	0.5209	0.5190	0.9664	0.9664	0.9627	0.0174	0.0174	0.0174
12A	0.5270	0.5270	0.5270	0.9780	0.9780	0.9780	0.0179	0.0179	0.0179
12B	0.5367	0.5367	0.5367	0.9879	0.9879	0.9879	0.0132	0.0132	0.0132
12C	0.5211	0.5211	0.5211	0.9666	0.9666	0.9666	0.0175	0.0175	0.0175

As indicated in the table above, the results obtained from the HBE-Python and HBE-MATLAB programs are identical. This indicates that the HBE-Python tool was implemented correctly for passive feeders without laterals.

The results obtained using the HBE-Excel tool differs for all nodes, besides the node at the end of the lateral. This is because superposition needs to be applied to obtain accurate results for previous nodes, which would only be possible in Microsoft Excel using Microsoft Visual Basic. Since Python had already been selected as the programming language to develop the HBET tool, the Microsoft Visual Basic option was not explored further.

6.2 12-bus Active Feeder without Laterals

For this scenario, the feeder layout remains the same as shown in Figure 6.1. However, in addition to the loads present at each node, distributed generators have also been added at selected nodes. The DG information is provided in Table 12 below.

Table 12: DG Parameters for 12-bus Active Feeder [2]

Node	DG Parameters			
	α	β	Power (kVA)	PF
2	12.62	2.21	500	0.9

6	12.62	2.21	300	1.0
8	12.62	2.21	600	0.95
12	12.62	2.21	700	0.975

Since both loads and DG are now connected to the feeder, it is expected that voltage rise will occur at some nodes, whereas voltage drops may be experienced at others. The HBE-Python voltage plots for the 12-bus active feeder are provided in Figure 6.5 to Figure 6.7 below.

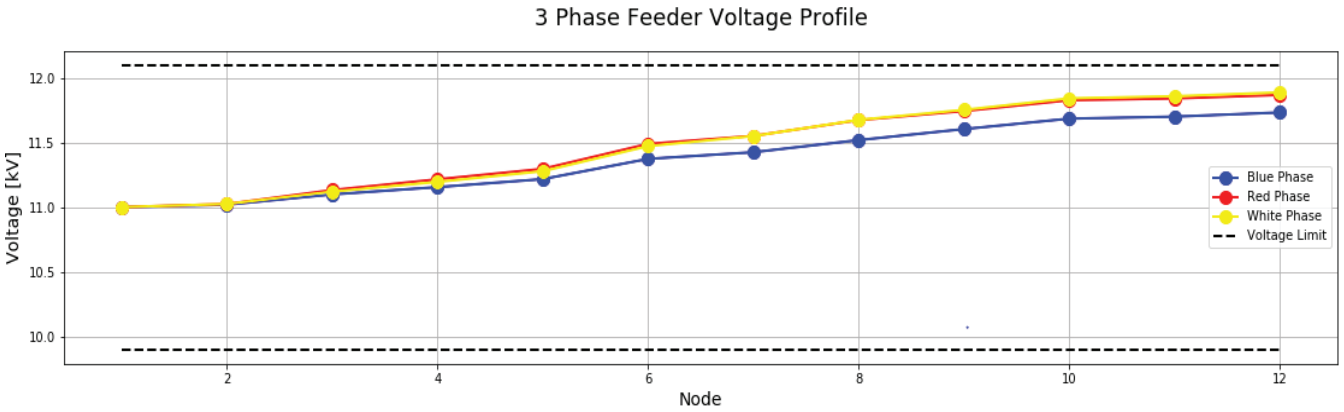


Figure 6.5: Percentile Voltage Plot for 12-bus Active Feeder

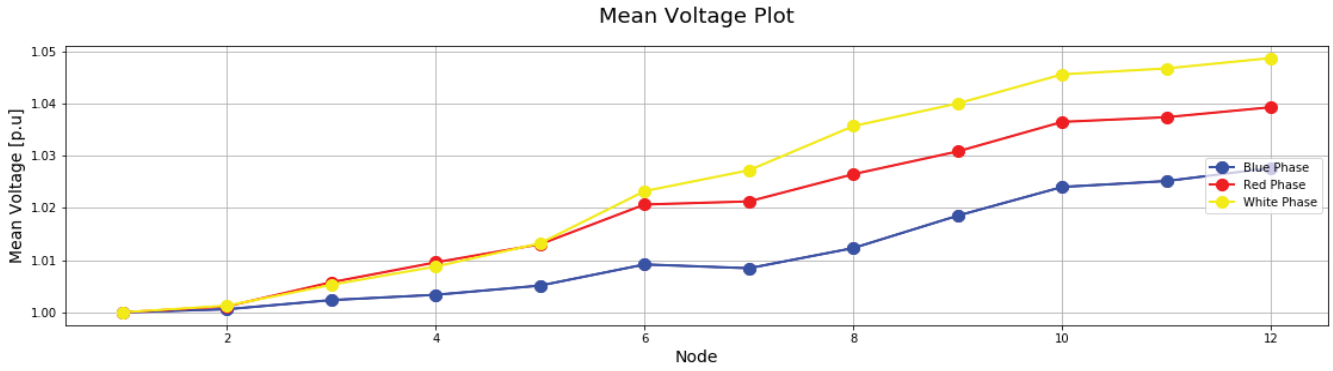


Figure 6.6: Mean Voltage Plot for 12-bus Active Feeder

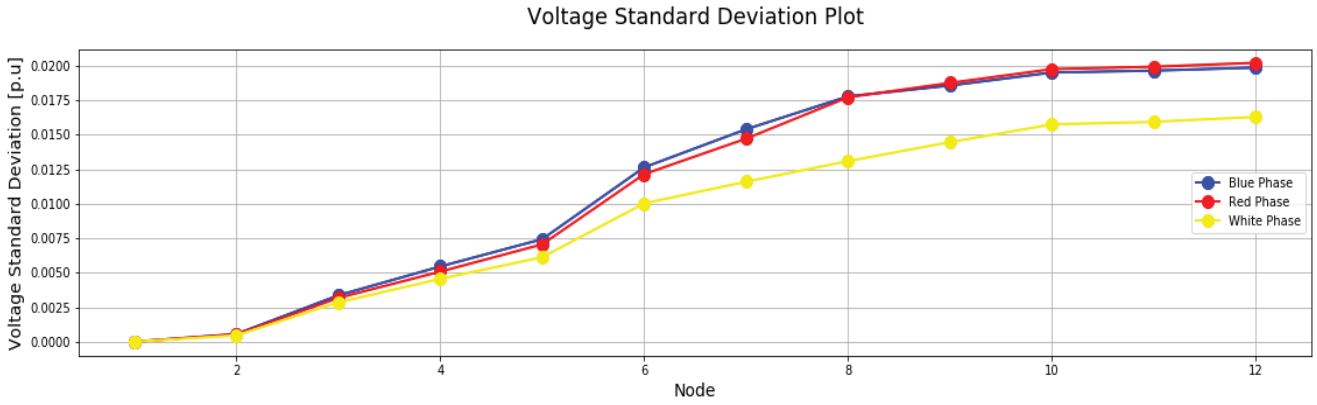


Figure 6.7: Voltage Standard Deviation Plot for 12-bus Active Feeder

The results obtained indicate that the introduction of DG contributes to a voltage rise, as expected [3]. The results at each node are provided in Table 13 below.

Table 13: Node Results for 12-bus Active Feeder

Node and phase	Percentile voltage (p.u)			Mean voltage (p.u)			Standard Deviation (p.u)		
	HBE-Python	HBE-MATLAB	HBE-Excel	HBE-Python	HBE-MATLAB	HBE-Excel	HBE-Python	HBE-MATLAB	HBE-Excel
2A	0.5568	0.5568	0.5559	1.0012	1.0012	1.0003	0.0005	0.0005	0.0002
2B	0.5568	0.5568	0.5560	1.0013	1.0013	1.0005	0.0005	0.0005	0.0001
2C	0.5565	0.5565	0.5559	1.0006	1.0006	1.0003	0.0006	0.0006	0.0002
3A	0.5622	0.5622	0.5561	1.0058	1.0058	0.9997	0.0032	0.0032	0.0007
3B	0.5616	0.5616	0.5561	1.0053	1.0053	0.9997	0.0028	0.0028	0.0006
3C	0.5605	0.5605	0.5563	1.0024	1.0024	1.0007	0.0034	0.0034	0.0003
4A	0.5664	0.5664	0.5571	1.0096	1.0096	1.0010	0.0051	0.0051	0.0009
4B	0.5654	0.5654	0.5563	1.0088	1.0088	0.9977	0.0046	0.0046	0.0018
4C	0.5634	0.5634	0.5569	1.0034	1.0034	0.9990	0.0055	0.0055	0.0018
5A	0.5705	0.5705	0.5570	1.0131	1.0131	0.9997	0.0071	0.0071	0.0015
5B	0.5696	0.5696	0.5568	1.0132	1.0132	0.9985	0.0061	0.0061	0.0019
5C	0.5665	0.5665	0.5566	1.0051	1.0051	0.9977	0.0074	0.0074	0.0021
6A	0.5802	0.5802	0.5601	1.0207	1.0207	1.0024	0.0121	0.0121	0.0030
6B	0.5794	0.5794	0.5599	1.0232	1.0232	1.0016	0.0100	0.0100	0.0032
6C	0.5744	0.5744	0.5610	1.0092	1.0092	1.0047	0.0126	0.0126	0.0026
7A	0.5834	0.5834	0.5644	1.0213	1.0213	1.0080	0.0147	0.0147	0.0041
7B	0.5833	0.5833	0.5617	1.0272	1.0272	0.9945	0.0116	0.0116	0.0085
7C	0.5770	0.5770	0.5615	1.0085	1.0085	0.9953	0.0154	0.0154	0.0079
8A	0.5895	0.5895	0.5808	1.0265	1.0265	1.0136	0.0177	0.0177	0.0163
8B	0.5896	0.5896	0.5803	1.0357	1.0357	1.0229	0.0131	0.0131	0.0111
8C	0.5817	0.5817	0.5704	1.0123	1.0123	0.9939	0.0178	0.0178	0.0167
9A	0.5931	0.5931	0.5787	1.0309	1.0309	1.0088	0.0188	0.0188	0.0169
9B	0.5935	0.5935	0.5786	1.0400	1.0400	1.0183	0.0145	0.0145	0.0119
9C	0.5860	0.5860	0.5722	1.0185	1.0185	0.9971	0.0186	0.0186	0.0168
10A	0.5973	0.5973	0.5806	1.0365	1.0365	1.0120	0.0198	0.0198	0.0169
10B	0.5980	0.5980	0.5774	1.0456	1.0456	1.0144	0.0157	0.0157	0.0127
10C	0.5901	0.5901	0.5696	1.0240	1.0240	0.9915	0.0195	0.0195	0.0173
11A	0.5980	0.5980	0.5788	1.0374	1.0374	1.0075	0.0199	0.0199	0.0175
11B	0.5988	0.5988	0.5793	1.0468	1.0468	1.0176	0.0159	0.0159	0.0128
11C	0.5909	0.5909	0.5674	1.0252	1.0252	0.9863	0.0196	0.0196	0.0179
12A	0.5993	0.5993	0.5993	1.0393	1.0393	1.0393	0.0202	0.0202	0.0202
12B	0.6003	0.6003	0.6003	1.0487	1.0487	1.0487	0.0163	0.0163	0.0163
12C	0.5925	0.5925	0.5925	1.0277	1.0277	1.0277	0.0199	0.0199	0.0199

As seen in the table, the results obtained from the HBE-Python and HBE-MATLAB programs are identical for all nodes and phases. Therefore, it is evident that the HBE-Python tool has been implemented correctly for active feeders not containing laterals. As before, the HBE-Excel tool only provided accurate results for the consumer-end node.

6.3 33-bus Passive Feeder with Laterals

The next system that was tested was the IEEE 33-bus test system containing three laterals. The system is operated radially at a voltage of 12.66kV. Once again, a scaling factor of 1.8 was used, implying a base voltage of 22.788kV. It should be noted that the scaling factor does not have much impact on the results. This was proven in [2], where a range of scaling factors were shown to provide almost identical results.

The feeder consists of a main radial feeder ranging from nodes 2 to 18. Laterals extending from nodes 19 to 22, 23 to 25 and 26 to 33 are connected to nodes 2, 3 and 6 respectively. The feeder layout was obtained from [2] and is shown in Figure 6.8 below.

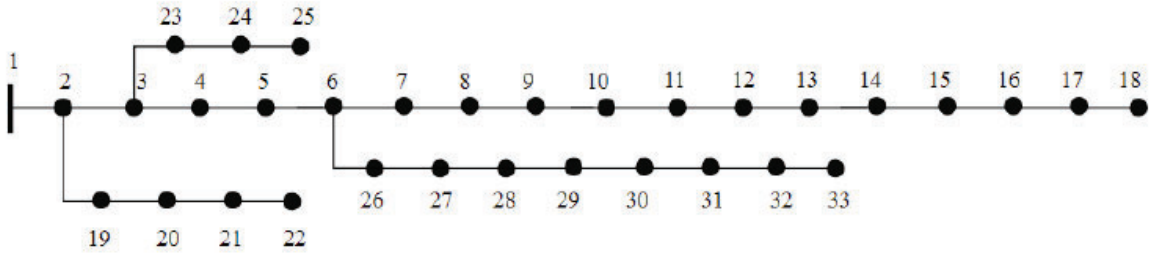


Figure 6.8: 33-Bus Feeder Layout [2]

Balanced loading was applied to all nodes of the system. Therefore, the percentile voltages, mean voltages and voltage standard deviations are expected to be identical across all phases at each node. For the full list of parameters used to simulate this network, refer to Appendix B . The HBE-Python voltage plots for the passive feeder are provided in Figure 6.9 to Figure 6.11 below.

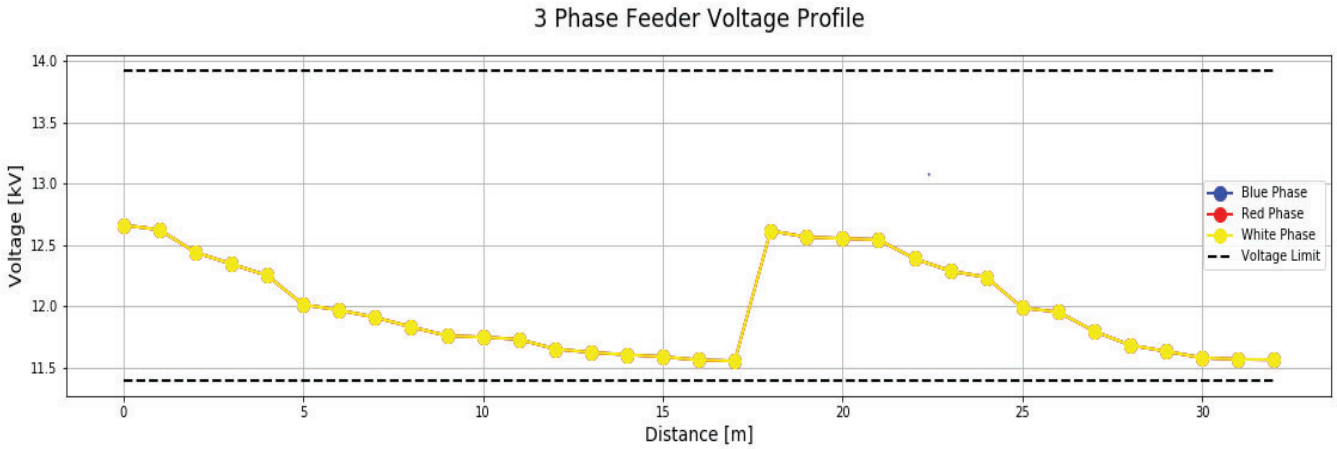


Figure 6.9: Percentile Voltage Plot for 33-bus Passive Feeder

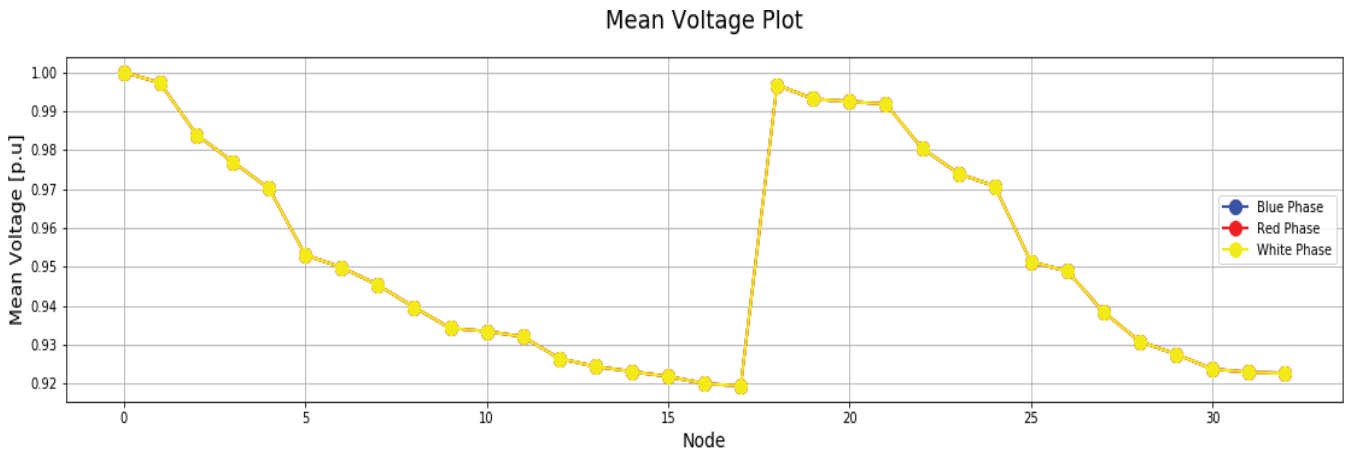


Figure 6.10: Mean Voltage Plot for 33-bus Passive Feeder

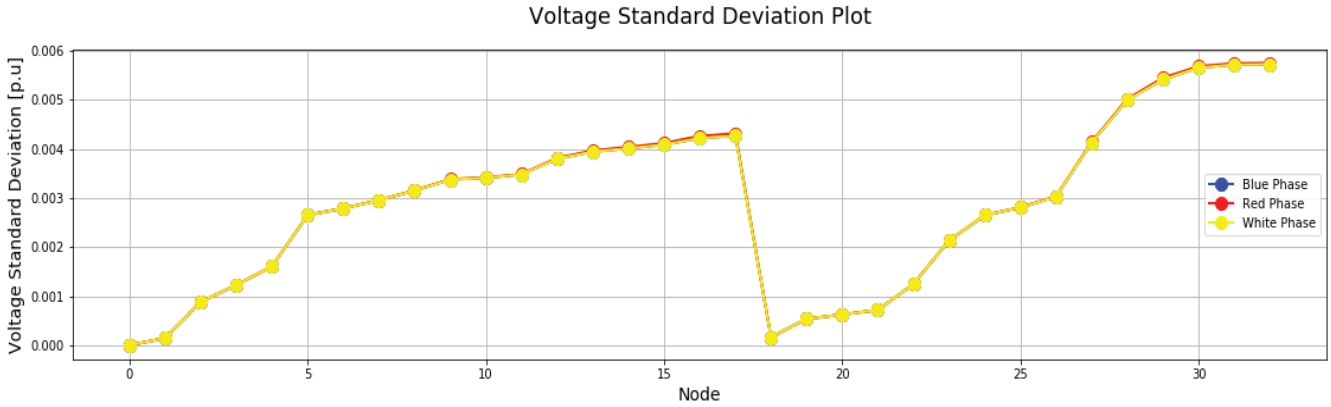


Figure 6.11: Voltage Standard Deviation Plot for 33-bus Passive Feeder

The MATLAB results were obtained by running the HBE-MATLAB program using the same conditions stated above.The results for all programs are provided in Table 14 below.

Table 14: Node Results for 33-bus Passive Feeder

Node and phase	Percentile voltage (p.u)			Mean voltage (p.u)			Standard Deviation (p.u)		
	HBE-Python	HBE-MATLAB	HBE-Excel	HBE-Python	HBE-MATLAB	HBE-Excel	HBE-Python	HBE-MATLAB	HBE-Excel
2	0.5539	0.5539	0.5555	0.9972	0.9972	0.9999	0.0001	0.0001	0.00002
3	0.5458	0.5458	0.5554	0.9839	0.9839	0.9995	0.0009	0.0009	0.00009
4	0.5416	0.5416	0.5550	0.9770	0.9770	0.9986	0.0012	0.0012	0.0002
5	0.5375	0.5375	0.5546	0.9701	0.9701	0.9979	0.0016	0.0016	0.0003
6	0.5271	0.5271	0.5542	0.9531	0.9531	0.9970	0.0027	0.0027	0.0003
7	0.5251	0.5251	0.5524	0.9499	0.9499	0.9928	0.0028	0.0028	0.0010
8	0.5225	0.5225	0.5500	0.9454	0.9454	0.9876	0.0030	0.0030	0.0015
9	0.5191	0.5191	0.5490	0.9396	0.9396	0.9857	0.0032	0.0032	0.0015
10	0.5159	0.5159	0.5478	0.9343	0.9343	0.9833	0.0034	0.0034	0.0016
11	0.5155	0.5155	0.5466	0.9335	0.9335	0.9811	0.0034	0.0034	0.0017
12	0.5146	0.5146	0.5450	0.9321	0.9321	0.9781	0.0035	0.0035	0.0018
13	0.5112	0.5112	0.5431	0.9265	0.9265	0.9743	0.0038	0.0038	0.0020
14	0.5100	0.5100	0.5389	0.9244	0.9244	0.9657	0.0039	0.0039	0.0027
15	0.5092	0.5092	0.5371	0.9231	0.9231	0.9622	0.0040	0.0040	0.0028
16	0.5084	0.5084	0.5349	0.9219	0.9219	0.9580	0.0041	0.0041	0.0029

17	0.5073	0.5073	0.5323	0.9200	0.9200	0.9531	0.0042	0.0042	0.0031
18	0.5069	0.5069	0.5281	0.9195	0.9195	0.9446	0.0043	0.0043	0.0036
19	0.5536	0.5536	0.5554	0.9967	0.9967	0.9997	0.0002	0.0002	0.00004
20	0.5512	0.5512	0.5549	0.9931	0.9931	0.9984	0.0005	0.0005	0.0003
21	0.5508	0.5508	0.5541	0.9924	0.9924	0.9966	0.0006	0.0006	0.0005
22	0.5503	0.5503	0.5530	0.9918	0.9918	0.9943	0.0007	0.0007	0.0007
23	0.5435	0.5435	0.5550	0.9804	0.9804	0.9988	0.0012	0.0012	0.0002
24	0.5391	0.5391	0.5525	0.9740	0.9740	0.9920	0.0021	0.0021	0.0014
25	0.5369	0.5369	0.5480	0.9707	0.9707	0.9821	0.0026	0.0026	0.0026
26	0.5259	0.5259	0.5536	0.9513	0.9513	0.9958	0.0028	0.0028	0.0004
27	0.5244	0.5244	0.5530	0.9489	0.9489	0.9946	0.0030	0.0030	0.0005
28	0.5176	0.5176	0.5522	0.9384	0.9384	0.9929	0.0041	0.0041	0.0006
29	0.5126	0.5126	0.5500	0.9309	0.9309	0.9881	0.0050	0.0050	0.0012
30	0.5104	0.5104	0.5424	0.9276	0.9276	0.9687	0.0054	0.0054	0.0046
31	0.5080	0.5080	0.5385	0.9238	0.9238	0.9612	0.0057	0.0057	0.0049
32	0.5075	0.5075	0.5327	0.9230	0.9230	0.9499	0.0057	0.0057	0.0055
33	0.5074	0.5074	0.5306	0.9227	0.9227	0.9460	0.0057	0.0057	0.0055

As seen from the table, the HBE-Python and HBE-MATLAB results obtained for all three parameters are identical for all nodes. It should also be noted that the results obtained are identical across all three phases for a specific node, since the system is a balanced one. The results obtained using the HBE-Excel tool are erroneuous for all nodes.

Based on the identical results obtained using the HBE-Python and HBE-MATLAB programs, as well as the extensive testing in [2] which indicated that the HBE-MATLAB program provided accurate results, it is clear that the HBE-Python program provides accurate results for passive networks containing laterals.

6.4 33-bus Active Feeder with Laterals

The final system that will be tested is a 33-bus active feeder. For this scenario, the same system parameters used in the previous scenario will be utilised for load nodes. In addition to this, distributed generators are placed at nodes 14, 18, 28 and 33. The parameters for the distributed generators are provided in Table 15 below.

Table 15: DG Parameters for 33-bus Active Feeder [2]

Node	DG Parameters			
	α	β	Power (kVA)	PF
14	12.62	2.21	500	0.90
18	12.62	2.21	750	0.95
28	12.62	2.21	1000	0.91
33	12.62	2.21	750	0.94

As before, the presence of DG on this feeder is expected to cause a voltage rise on certain nodes. The HBE-Python voltage plots for the active feeder with laterals are provided in Figure 6.12 to Figure 6.14 below.

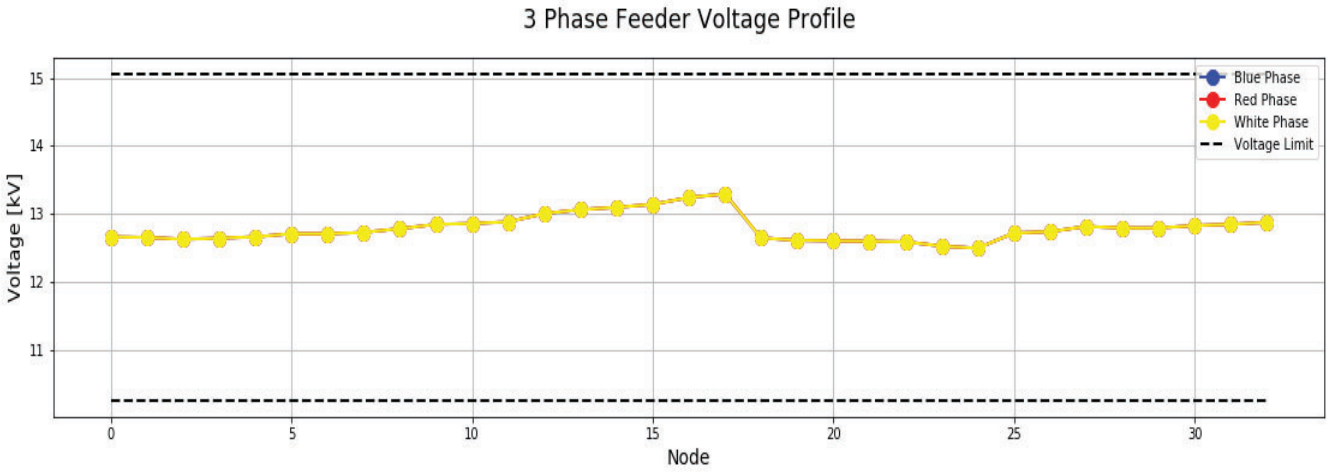


Figure 6.12: Percentile Voltage Plot for 33-bus Active Feeder

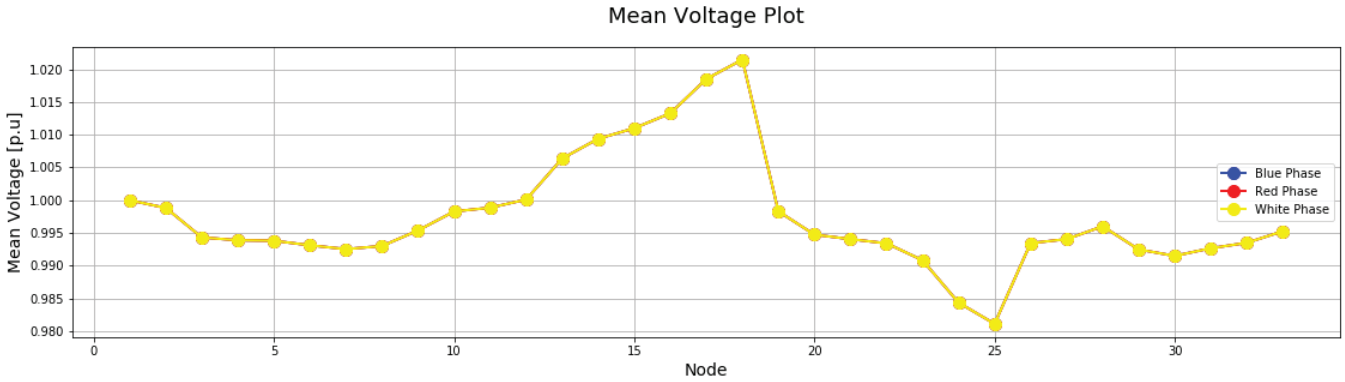


Figure 6.13: Mean Voltage Plot for 33-bus Active Feeder

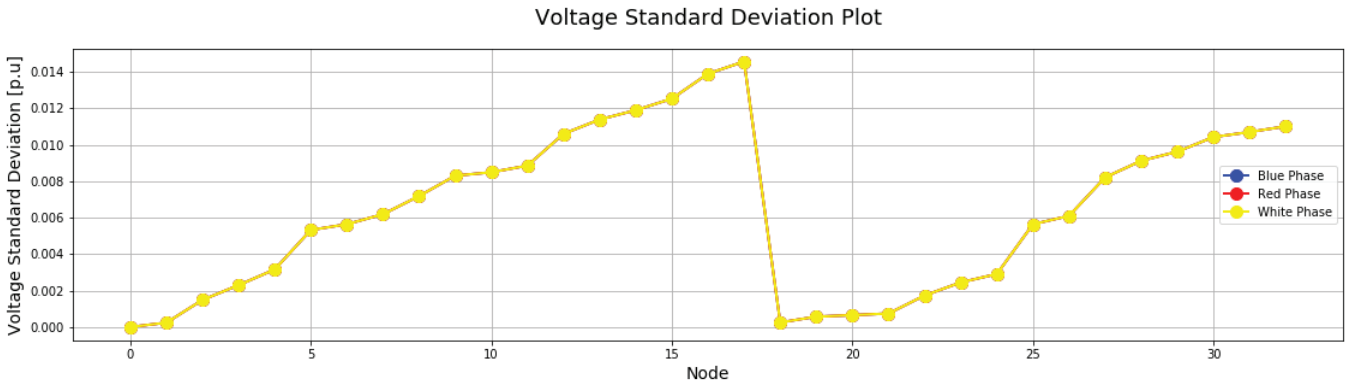


Figure 6.14: Voltage Standard Deviation Plot for 33-bus Active Feeder

The results obtained indicate that the mean voltage generally increases, as expected due to the presence of distributed generators. It can also be seen that the voltage decreases along the laterals connected to nodes 2 and 3. This was also expected, as there are no distributed generators present on these laterals. The results obtained are indicated in Table 16 below.

Table 16: Node Results for 33-bus Active Feeder

Node and phase	Percentile voltage (p.u)			Mean voltage (p.u)			Standard Deviation (p.u)		
	HBE-Python	HBE-MATLAB	HBE-Excel	HBE-Python	HBE-MATLAB	HBE-Excel	HBE-Python	HBE-MATLAB	HBE-Excel
2	0.5552	0.5552	0.5555	0.9988	0.9988	0.9999	0.0002	0.0002	0.00002
3	0.5540	0.5540	0.5554	0.9943	0.9943	0.9995	0.0015	0.0015	0.00009

4	0.5547	0.5547	0.5550	0.9939	0.9939	0.9986	0.0023	0.0023	0.0002
5	0.5556	0.5556	0.5546	0.9938	0.9938	0.9979	0.0032	0.0032	0.0003
6	0.5575	0.5575	0.5542	0.9931	0.9931	0.9970	0.0053	0.0053	0.0003
7	0.5575	0.5575	0.5524	0.9925	0.9925	0.9928	0.0056	0.0056	0.0010
8	0.5584	0.5584	0.5500	0.9930	0.9930	0.9876	0.0062	0.0062	0.0015
9	0.5608	0.5608	0.5490	0.9954	0.9954	0.9857	0.0072	0.0072	0.0015
10	0.5636	0.5636	0.5478	0.9983	0.9983	0.9833	0.0083	0.0083	0.0016
11	0.5642	0.5642	0.5466	0.9989	0.9989	0.9811	0.0085	0.0085	0.0017
12	0.5652	0.5652	0.5450	1.0001	1.0001	0.9781	0.0089	0.0089	0.0018
13	0.5706	0.5706	0.5431	1.0064	1.0064	0.9743	0.0106	0.0106	0.0020
14	0.5731	0.5731	0.5561	1.0094	1.0094	0.9908	0.0114	0.0114	0.0062
15	0.5746	0.5746	0.5542	1.0110	1.0110	0.9873	0.0119	0.0119	0.0063
16	0.5765	0.5765	0.5519	1.0133	1.0133	0.9831	0.0125	0.0125	0.0064
17	0.5809	0.5809	0.5493	1.0185	1.0185	0.9781	0.0139	0.0139	0.0064
18	0.5833	0.5833	0.5810	1.0215	1.0215	1.0230	0.0146	0.0146	0.0139
19	0.5549	0.5549	0.5554	0.9983	0.9983	0.9997	0.0002	0.0002	0.00004
20	0.5533	0.5533	0.5549	0.9948	0.9948	0.9984	0.0006	0.0006	0.0003
21	0.5530	0.5530	0.5541	0.9941	0.9941	0.9966	0.0007	0.0007	0.0005
22	0.5527	0.5527	0.5530	0.9934	0.9934	0.9943	0.0007	0.0007	0.0007
23	0.5524	0.5524	0.5550	0.9908	0.9908	0.9988	0.0017	0.0017	0.0002
24	0.5495	0.5495	0.5525	0.9844	0.9844	0.9920	0.0024	0.0024	0.0014
25	0.5482	0.5482	0.5480	0.9811	0.9811	0.9821	0.0029	0.0029	0.0026
26	0.5581	0.5581	0.5536	0.9935	0.9935	0.9958	0.0056	0.0056	0.0004
27	0.5589	0.5589	0.5530	0.9941	0.9941	0.9946	0.0061	0.0061	0.0005
28	0.5623	0.5623	0.5695	0.9960	0.9960	1.0164	0.0082	0.0082	0.0053
29	0.5613	0.5613	0.5670	0.9925	0.9925	1.0116	0.0091	0.0091	0.0054
30	0.5613	0.5613	0.5577	0.9915	0.9915	0.9923	0.0096	0.0096	0.0070
31	0.5628	0.5628	0.5537	0.9927	0.9927	0.9847	0.0104	0.0104	0.0072
32	0.5636	0.5636	0.5477	0.9935	0.9935	0.9734	0.0107	0.0107	0.0076
33	0.5649	0.5649	0.5663	0.9953	0.9953	1.0020	0.0110	0.0110	0.0106

Once again, the resuts obtained using the HBE-Python and HBE-MATLAB programs are identical. Therefore, the HBE-Python tool has been developed to accurately determine the performance of active feeders containing laterals. As before, the HBE-Excel tool provides erroneous results for all nodes.

6.5 Limitations of the HBE-Python Tool

Although the HBE-Python tool replicated the accurate results produced by the HBE-MATLAB tool, it does have some drawbacks. These limitations include the following:

- The user cannot analyse 3-phase, 3-wire systems since this portion of the HBET was not transcribed to Python code.
- Branch current analysis is not available. Therefore, the tool does not allow the user to determine the adequacy of conductors in the simulated feeder. Once again, this portion of the HBET was not transcribed to Python code and could be added.
- Graphical results cannot be exported to MS Excel or text files for further analysis.
- Some new features introduced during the development of the HBET, such as correlation, was not part of the scope of this work and was therefore not included in the HBE-Python tool.

6.6 Chapter Summary

Based on the results presented in this section, the following summarising points can be deduced:

- The HBE-MATLAB program was extensively tested and was proven to provide accurate results [2]. Since the results obtained using the developed HBE-Python program were identical to those obtained using the HBE-MATLAB program for all scenarios simulated, it is deduced that the algorithm was correctly translated to code and that the HBE-Python program can accurately determine the voltage performance of feeders with or without laterals.
- The HBE-Excel tool can only be used to determine the percentile voltage, mean voltage and voltage standard deviation for the consumer-end node in systems without laterals. The HBE-Excel tool is incapable of determining the performance of systems containing laterals.
- Despite the accurate transcription of the HBET to Python code, there are a few limtations to the HBE-Python tool. This includes, but is not limited to, the tool's inability to simulate 3-phase, 3-wire systems and analyse branch currents. As such, the HBE-Python tool is only suitable for voltage analysis on lateral and branched radial feeders.

7. Conclusions

The Herman-Beta Extended Transform is a powerful network development tool, with many more applications than have been covered in this dissertation. In addition to finding answers to the research questions outlined in Section 1.2.2, the main deliverable of this dissertation was to produce an easily accessible network planning tool using the HBET algorithm. This chapter specifies the answers to the research questions posed and evaluates the validity of the hypothesis.

7.1 Answers to the Research Questions

Most of the research questions posed have been answered throughout this dissertation, but will be presented in this subsection for ease of reference.

7.1.1 What other probabilistic load flow techniques have been developed?

Many probabilistic load flow techniques have been developed since the investigation of probabilistic load flow by Borkowska [6] in 1974. They have all been developed with the same goal, to provide accurate results quickly in systems containing uncertainty. In the literature review conducted, in Section 2.1.2, some of the developed probabilistic load flow techniques are discussed. These include the multilinearisation technique, quadratic probabilistic load flow, point-estimate method, least square method, convolution methods as well as the Cumulants and Gram-Charlier expansion method.

Although each of these methods have their advantages, the HBET was found to be the optimal technique in terms of accuracy and computational speed. Additonally, the HBET has the ability to handle probability density functions which are negatively or positively skewed, is able to account for neutral resistance and unbalanced feeder loading and has the capability of analysing both 3-phase, 3-wire as well as 3-phase, 4-wire system configurations.

7.1.2 How do network designers deal with uncertainty in power systems?

The answer to this research question was explored in Section 2.4. The most common approach utilised to determine future load demands is to use past load data if it is available, wherefrom trends can be drawn to determine the predicted load value and hence, the corresponding system performance. . In the absence of load data, the network planner often opts to analyse an area/development that is similar to the area under consideration and manipulates the existing data to extrapolate the proposed area's load demand.

In recent years, popular network planning tools such as DIgSILENT and ReticMaster have recognised the importance of accounting for uncertainty in power systems by incorporating PLF analysis in their toolboxes. This allows network planners to obtain results that are more accurate than those obtained using deterministic methods. The adoption of PLF tools is expected to increase in the near future due to the expected increase in renewable energy and the uptake of electric vehicles.

The analysis of systems containing uncertainty is expected to become increasingly important due to the uptake of renewable energy, Independent Power Producers and electric vehicles. The developed HBE-Python tool is intented to assist network planners in conducting voltage assessments of their radial systems containing uncertainty.

7.1.3 What approaches were used to develop other network planning tools and what features were implemented in these programs?

Previously developed tools and application development guidelines were consulted to answer this question. All previously developed tools utilise a GUI to easily accept information from the user. In addition to this, the programs are always developed to be as simple and user-friendly as possible. The use of buttons and widgets were also widely used. Most programs utilse a menubar to provide additional tools under various dropdown categories as well as guideline documents or videos to assist the user with utilising the program, if required. It was also noted that the ability to export results and/or graphs are extremely important, for the user to further analyse results or to use graphical results in reports.

Two popular network planning tools capable of performing PLF analysis were also investigated, namely DIgSILENT PowerFactory and ReticMaster. Apart from the features specified above, both tools allow for spreadsheets to be imported as inputs, which helped prompt the use of the spreadsheet input for the HBE-Python tool. All considerations mentioned in this subsection were taken into account when developing the HBE-Python tool.

7.1.4 What is the ideal programming software to implement the Herman-Beta Extended Transform based on ease of learning, functionality and online support?

Python was chosen since it can easily be interfaced with other commonly used simulation tools, such as DIgSILENT and OpenDSS. It is one of the simpler programming languages to learn and understand and has the capability of developing powerful applications. Furthermore, there is an abundance of resources available online to assist with the development of the tool, should this be necessary.

One of the available online resources are Python wheels. Wheels are pre-written pieces of code used to perform powerful functions. Additionally, Python has a built-in library specifically for the creation of easily customisable GUIs.

These features, in conjunction with the fact that Python is widely used, familiar to most programmers and is freely available, makes it the ideal programming software to implement the new HBE-Python tool.

7.1.5 What form of input is the most user-friendly and efficient for network planners?

A spreadsheet was chosen as the most efficient form of user input. This is especially useful for this program, since load data and parameters barely differ between nodes, and can easily be duplicated using a spreadsheet. Another efficient, user-friendly method of accepting inputs is via a GUI. Both methods were utilised in the development of the HBE-Python tool.

7.1.6 What output information will be useful to network planners and what results would they require to make informed decisions?

For radial network design in electrical systems, one of the most important outputs required by network planners is voltage variation (threshold values, not mean values) between the source and consumer ends, since there are voltage limits which utilities need to adhere to in order to ensure adequate quality of supply. Therefore, voltage variation is of utmost importance when determining if the proposed system has been adequately designed.

7.1.7 How can the developed software program be broken down into simpler functions?

The HBE-Python tool was broken down into functions based on various sub-processes needing to be followed or buttons being clicked. That being said, all processes which did not form part of the main program, such as the "About" function, Beta calculators and "User Guide" option, were developed as separate functions. In addition to these, the HBET calculations were developed as a separate function, due to the mere length of this portion of code. All functions utilised in the development of the HBE-Python tool can be found in Section 5.2.

7.1.8 What features should be implemented in the program to promote ease of use?

The features that were implemented in the HBE-Python program were obtained from the observation and investigation of other similar programs. From these programs, it was determined that a GUI should be used, simply because it is much more intuitive to use than any other form of navigating programs. Additionally, a menu bar would also be used, as many existing programs utilise it and it is an easy way allow users to find other useful built-in functions.

The following additional key features were implemented to promote ease of use:

- Error messages are shown when incorrect inputs are received from the user, instead of requiring the user to restart the program.
- An input spreadsheet to accept user inputs for the designed system. The spreadsheet approach was adopted to easily allow users to replicate parameters across nodes.
- The user should have the option of adding cable/conductor types to the default library, thereby offering a fully customisable user system.
- The results should be shown graphically and a summary table should be provided.
- A user guide to explain to users how to utilise the program.

7.2 Research Hypothesis Validity

At the start of this dissertation, the following hypothesis was made:

"An open-source platform can be developed for efficient handling of network data and implementation of the voltage assessment capabilities of the HBET."

The comparison of the percentile voltages, mean voltages and voltage standard deviations for the four systems simulated in Chapter 6 indicated that identical results are obtained using the HBE-Python and HBE-MATLAB tools. Furthermore, the newly developed tool was created using Python, which is readily and freely available to anyone who wishes to use it. Based on these two statements, the hypothesis has been validated.

This work has been based largely on the work completed by Chihota [2], which in turn was based off the work developed over decades by a handful of individuals. Although the voltage capability of the HBET has been successfully implemented in the HBE-Python tool, there are many more aspects of the HBET that could be utilised in future.

8. References

[1] K. Olikara, “Power quality issues, impacts, and mitigation for industrial customers”. Rockwell Automation, Inc.

[2] M. J. Chihota, "Extending the Herman-Beta transform for probabilistic load flow analysis of radial feeders", Ph.D, dissertation, Dept. Elect. Eng., Univ. of Cape Town, Cape Town, South Africa, 2019.

[3] E. Namanya, A. Waligo, A. Ipurale, C. T. Gaunt and R. Herman, "Voltage calculations program for LV feeders with distributed generation", 2016 IEEE PES PowerAfrica, 2016. Available: 10.1109/powerafrica.2016.7556595.

[4] E. Namanya, "Voltage calculation on low voltage feeders with distributed generation", M.S. thesis, Dept. Elect. Eng., Univ. of Cape Town, Cape Town, South Africa, 2014.

[5] L. C. Siebert, L. R. Ferreira, A. R. Aoki, A. F. Bonelli, A. R. R. de Souza and F. de O. Toledo, "Deterministic versus probabilistic approaches to self-healing in Smart Grid," *22nd International Conference and Exhibition on Electricity Distribution (CIRED 2013)*, 2013, pp. 1-4, doi: 10.1049/cp.2013.1236.

[6] B. Borkowska, "Probabilistic load flow," in *IEEE Transactions on Power Apparatus and Systems*, vol. PAS-93, no. 3, pp. 752-759, May 1974, doi: 10.1109/TPAS.1974.293973.

[7] M. J. Chihota and C. T. Gaunt, "Transform for probabilistic voltage computation on distribution Feeders with Distributed Generation," *2018 Power Systems Computation Conference (PSCC)*, 2018, pp. 1-7, doi: 10.23919/PSCC.2018.8442503.

[8] J. Reinders, J. Morren and H. Slootweg, "Comparing probabilistic load flow methods in dealing with uncertainties at TSO/DSO interface," *2017 52nd International Universities Power Engineering Conference (UPEC)*, 2017, pp. 1-5, doi: 10.1109/UPEC.2017.8232030.

[9] P. Chen, Z. Chen and B. Bak-Jensen, "Probabilistic load flow: A review," *2008 Third International Conference on Electric Utility Deregulation and Restructuring and Power Technologies*, 2008, pp. 1586-1591, doi: 10.1109/DRPT.2008.4523658.

[10] A. M. L. Da Silva and V. Arienti, "Probabilistic load flow by a multilinear simulation algorithm", *IEE Proceedings C Generation, Transmission and Distribution*, vol. 137, no. 4, p. 276, 1990. Available: 10.1049/ip-c.1990.0037.

[11] Li Xiaoming, Chen Xiaohui, Yin Xianggen, Xiang Tieyuan and Liu Huagang, "The algorithm of probabilistic load flow retaining nonlinearity," *Proceedings. International Conference on Power System Technology*, 2002, pp. 2111-2115 vol.4, doi: 10.1109/ICPST.2002.1047154.

[12] Chun-Lien Su, "Probabilistic load-flow computation using point estimate method," in *IEEE Transactions on Power Systems*, vol. 20, no. 4, pp. 1843-1851, Nov. 2005, doi: 10.1109/TPWRS.2005.857921.

[13] J. F. Dopazo, O. A. Klitin and A. M. Sasson, "Stochastic load flows," in *IEEE Transactions on Power Apparatus and Systems*, vol. 94, no. 2, pp. 299-309, March 1975, doi: 10.1109/T-PAS.1975.31855.

[14] L. S. Van Slyck and J. F. Dopazo, "Conventional load flow not suited for real time power system monitoring," *Proceedings of the 8th IEEE PICA Conference*, June, 1973.

[15] R. N. Allan, A. M. L. Da Silva and R. C. Burchett, "Evaluation methods and accuracy in probabilistic load flow solutions," in *IEEE Transactions on Power Apparatus and Systems*, vol. PAS-100, no. 5, pp. 2539-2546, May 1981, doi: 10.1109/TPAS.1981.316721.

[16] Pei Zhang and S. T. Lee, "Probabilistic load flow computation using the method of combined cumulants and Gram-Charlier expansion," in *IEEE Transactions on Power Systems*, vol. 19, no. 1, pp. 676-682, Feb. 2004, doi: 10.1109/TPWRS.2003.818743.

[17] R. Herman and C. T. Gaunt, "A practical probabilistic design procedure for LV residential distribution systems," in *IEEE Transactions on Power Delivery*, vol. 23, no. 4, pp. 2247-2254, Oct. 2008, doi: 10.1109/TPWRD.2008.919041.

[18] S. W. Heunis and R. Herman, "A probabilistic model for residential consumer loads," in *IEEE Transactions on Power Systems*, vol. 17, no. 3, pp. 621-625, Aug. 2002, doi: 10.1109/TPWRS.2002.800901.

[19] C. T. Gaunt, R. Herman, M. Dekenah, R. Sellick and S. Heunis, "Data collection, load modelling and probabilistic analysis for LV domestic electrification", 1999.

[20] *MATLAB - MathWorks*. [Online]. Available: https://www.mathworks.com/products/matlab.html?s_tid=hp_ff_p_matlab.

[21] Osako, "A highly opinionated review of programming languages for the novice", *Josako.freeservers.com*, 2004. [Online]. Available: http://josako.freeservers.com/tech/langreview/.

[22] "Top 10 programming languages of the world – 2019 to begin with", *GeeksforGeeks*, 2019. [Online]. Available: https://www.geeksforgeeks.org/top-10-programming-languages-of-the-world-2019-to-begin-with/.

[23] D. Swersky, "Top 43 programming languages: When and how to use them | Raygun Blog", *Raygun Blog*, 2018. [Online]. Available: https://raygun.com/blog/programming-languages/.

[24] M. T. Tsholoba and A. K. Raji, "Impact assessment of high penetration of rooftop PV in urban residential networks," *2019 International Conference on the Domestic Use of Energy (DUE)*, 2019, pp. 183-189.

[25] City of Cape Town, *Site development plans*. Cape Town: City of Cape Town, 2020, pp. 4-6. [Online]. Available: https://www.capetown.gov.za/Document-centre#k=site%20development%20plans

[26] N. Ip Cho and K. Awodele, "Comparison of four load models for reliability evaluation considering reconfiguration using Monte Carlo Simulation," *2012 IEEE International Conference on Power System Technology (POWERCON)*, 2012, pp. 1-6, doi: 10.1109/PowerCon.2012.6401384.

[27] Y. Fuangfung, S. Sinthusonthishat and P. Yutthagowith, "A software tool for induced voltages and currents calculation caused by lightning electromagnetic field in PV systems," *2015 12th International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology (ECTI-CON)*, 2015, pp. 1-4, doi: 10.1109/ECTICon.2015.7207015.

[28] S. P. Zhu, J. Liu, K. C. Tan and C. S. Tan, "A PC-based tool to calculate voltage drops and select cables for final subcircuits," *Proceedings 1995 International Conference on Energy Management and Power Delivery EMPD '95*, 1995, pp. 554-559 vol.2, doi: 10.1109/EMPD.1995.500787.

[29] P. Selvan and R. Anita, “Revelation for new user to select power system simulation software,” Journal of Asian Scientific Research, vol. 1, no. 7, pp. 366–375, Nov. 2011.

[30] L. Bam and W. Jewell, “Review: Power system analysis software tools,” *IEEE Power Engineering Society General Meeting, 2005*.

[31] (2022) *DIgSILENT PowerFactory* (SP1) [Computer Program].

[32] *Probablistic Analysis - DIgSILENT*. Available at: https://www.digsilent.de/en/probabilistic-analysis.html (Accessed: October 5, 2022).

[33] *Inspired Interfaces | ReticMaster*. Available at: https://inspiredinterfaces.co.za/ReticMaster.php (Accessed: October 6, 2022).

[34] I. Dzafic, I. Huseinagic, M. Music, and E. Halilovic, “Software package for power system analysis,” in *2014 IEEE International Energy Conference (ENERGYCON 2014): Dubrovnik, Croatia, 13-16 May 2014*.

[35] International Telecommunication Union, "Guidelines for network planning tools for developing countries and countries with economies in transition", Geneva, 2005.

[36] S. Ghosal, “Cognitive issues in interface design and product development,” *Proceedings of the First Regional Conference, IEEE Engineering in Medicine and Biology Society and 14th Conference of the Biomedical Engineering Society of India. An International Meet*.

[37] C. T. Gaunt, R. Herman and B. Bekker, "Probabilistic methods for renewable energy sources and associated electrical loads for Southern African distribution systems," *2009 CIGRE/IEEE PES Joint Symposium Integration of Wide-Scale Renewable Resources into the Power Delivery System*, 2009, pp. 1-7.

[38] Eskom, "NRS 034-1:2007 Electricity distribution - Guidelines for the provision of electricity distribution networks in residential areas Part 1: Planning and design of distribution networks", 2007.

[39] M. Chihota, "Applying the Herman-Beta probabilistic method to MV feeders", M.S. thesis, Dept. Elect. Eng., Univ. of Cape Town, Cape Town, South Africa, 2015.

[40] R. Herman and C. T. Gaunt, *HB13-A3.ver4a*. 2013.

[41] A. Kim, “Beta distribution - Intuition, examples and derivation”, 2020. [Online]. Available: https://towardsdatascience.com/beta-distribution-intuition-examples-and-derivation-cf00f4db57af (Accessed: October 18, 2022).

Appendix A : HBET Symbols and Algorithms

This appendix provides the steps taken to calculate the voltage drop across a feeder using the Herman-Beta Extended Transform. Firstly, a description of the symbols used in the equations are provided. Thereafter, the procedure for calculating the probabilistic voltage drop for 3-phase, 4-wire systems and 3-phase, 3-wire systems are presented.

A1: DESCRIPTION OF SYMBOLS

SYMBOL	DESCRIPTION
$\alpha_{V_{con}}$	Beta PDF parameter, Alpha, for normalised consumer voltage
$\beta_{V_{con}}$	Beta PDF parameter, Beta, for normalised consumer voltage
ρ	Confidence level used to determine outputs
$\rho_{x,y}$	Correlation coefficient between phase x and y
$\emptyset_i$	Power angle at node i
$\sigma_{V_{x_i}}$	Standard deviation of the x-component of the voltage at node i
$betainv$	Beta inverse function
C_i	Current limit for the respective node
C_B	Normalised current limit for the respective node
$E(\Delta V_{x_i})$	First moment of the x-component of the voltage drop at node i
$E(\Delta V_{y_i})$	First moment of the y-component of the voltage drop at node i
$E(\Delta V_{x_i}{}^2)$	Second moment of the x-component of the voltage drop at node i
$E(\Delta V_{y_i}{}^2)$	Second moment of the y-component of the voltage drop at node i
$E(\Delta V_{x_t})$	Sum of first moments of the x-component of the voltage drop up to node i
$E(\Delta V_{y_t})$	Sum of first moments of the y-component of the voltage drop up to node i
$E(\Delta V_{x_t}{}^2)$	Sum of second moments of the x-component of the voltage drop up to node i
$E(\Delta V_{y_t}{}^2)$	Sum of second moments of the y-component of the voltage drop up to node i
$E(V_c)$	First moment of the total normalised consumer voltage

$E(V_c^{\,2})$	Second moment of the total normalised consumer voltage
G	First moment of a Beta PDF
H	Second moment of a Beta PDF
I_B	System base current used to normalise values to a per-unit system
k_r	Neutral to phase conductor resistance ratio
k_x	Neutral to phase conductor reactance ratio
n_R	Number of customers connected to red phase (phase A) at node i
n_W	Number of customers connected to white phase (phase B) at node i
n_B	Number of customers connected to blue phase (phase C) at node i
N	Number of nodes in the feeder
PF_i	Power factor at node i
R_{n_B}	Normalised, temperature-adjusted neutral resistance of the respective branch
R_{p_B}	Normalised, temperature-adjusted phase resistance of the respective branch
R_{n_i}	Temperature-adjusted neutral resistance of the respective branch
R_{p_i}	Temperature-adjusted phase resistance of the respective branch
R_i	Total normalised temperature-adjusted phase resistance at node i
R_n	Total normalised temperature-adjusted neutral resistance at node i
S_B	System base power used to normalise values to a per-unit system
$V^*_{con\%}$	Normalised percentile consumer voltage
$V_{con\%}$	Percentile consumer voltage
V_B	System base voltage used to normalise values to a per-unit system
V_s	System supply/nominal voltage
X_{n_B}	Normalised, temperature-adjusted neutral reactance of the respective branch
X_{p_B}	Normalised, temperature-adjusted phase reactance of the respective branch
X_{n_i}	Temperature-adjusted neutral reactance of the respective branch
X_{p_i}	Temperature-adjusted phase reactance of the respective branch

X_i	Total normalised temperature-adjusted phase reactance at node i
X_n	Total normalised temperature-adjusted neutral reactance at node i
Z_B	System base impedance used to normalise values to a per-unit system

A2: PROCEDURE FOR USING THE HBET: 3-PHASE, 4-WIRE SYSTEMS

Step 1 – Select Network Parameters

- Specify the source/nominal voltage, denoted by V_s.
- Specify the per-unit base values for apparent power (S_B) and voltage (V_B).
- Provide the parameters describing the loads/generators at each node, namely α_i, β_i, C_i and PF_i.
- Specify the number of customers and generators connected at each node. The customers for the red, white and blue phase at node i are denoted by n_{R_i}, n_{W_i} and n_{B_i} respectively. Positive values are assigned to represent loads and negative values are used to represent generators.
- Specify the number of nodes in the radial network, denoted by N.
- Provide the resistance and reactance of the phase and neutral conductors between each node $(i-1)$ and node i, denoted by $R_{p_i}, R_{n_i}, X_{p_i}$ and X_{n_i}.
- Specify the $N \times N \times phases$ correlation matrix Rho for interphase, intraphase and internode correlation.
- Specify the confidence level as a percentage, denoted by ρ.

Step 2 – Determine Per-unit Equivalents of System Inputs

$$I_B = \frac{S_B}{V_B}\ ;\ Z_B = \frac{{V_B}^2}{S_B}$$

$$R_{p_B} = \frac{R_{p_i}}{Z_B};\ X_{p_B} = \frac{X_{p_i}}{Z_B};\ R_{n_B} = \frac{R_{ni}}{Z_B};\ X_{n_B} = \frac{X_{ni}}{Z_B};\ C_B = \frac{C_i}{I_B}$$

Step 3 – Determine Superposition Elements

$$R_i = \sum_{j=1}^{i} R_{p_{B_{(j)}}}\ ;\ X_i = \sum_{j=1}^{i} X_{p_{B_{(j)}}}\ ;\ R_n = \sum_{j=1}^{i} R_{n_{B_{(j)}}}\ ;\ X_n = \sum_{j=1}^{i} X_{n_{B_{(j)}}}$$

$$k_r = \frac{R_n}{R_i};\ k_x = \frac{X_n}{X_i}$$

Step 4 – Calculate First and Second Moments for Input Currents

$$G_i = \frac{\alpha_i}{\alpha_i+\beta_i};\ H_i = \frac{\alpha_i(\alpha_i+1)}{\alpha_i+\beta_i(\alpha_i+\beta_i+1)};\ \emptyset_i = \text{acos}(PF_i)$$

Step 5 – Calculate Constants $k_1 - k_4$; $c_1 - c_6$; $d_1 - d_6$; $C_1 - C_6$

$$\begin{bmatrix} k_1 \\ k_2 \\ k_3 \\ k_4 \end{bmatrix} = \begin{bmatrix} k_r R_i \pm k_x X_i \tan \emptyset_i \\ k_x X_i \mp k_r R_i \tan \emptyset_i \\ (1+k_r)R_i \pm (1+k_x)X_i \tan \emptyset_i \\ (1+k_x)X_i \mp (1+k_r)R_i \tan \emptyset_i \end{bmatrix}$$

$$\begin{bmatrix} c_1 \\ c_2 \\ c_3 \\ c_4 \\ c_5 \\ c_6 \end{bmatrix} = \begin{bmatrix} 0.5k_1k_2\sqrt{3} + 0.75{k_2}^2 + 0.25{k_1}^2 \\ -0.5k_1k_2\sqrt{3} + 0.75{k_2}^2 + 0.25{k_1}^2 \\ {k_3}^2|n_R|(|n_R|-1) + c_1|n_W|(|n_W|-1) + c_2|n_B|(|n_B|-1) \\ |n_R||n_W|\left(-k_3k_2\sqrt{3} - k_1k_3\right) + |n_R||n_B|\left(k_3k_2\sqrt{3} - k_1k_3\right) + |n_W||n_B|\left(-1.5{k_2}^2 + 0.5{k_1}^2\right) \\ \rho_{R,R}{k_3}^2|n_R|(|n_R|-1) + \rho_{W,W}c_1|n_W|(|n_W|-1) + \rho_{B,B}c_2|n_B|(|n_B|-1) \\ \rho_{R,W}|n_R||n_W|\left(-k_3k_2\sqrt{3} - k_1k_3\right) + \rho_{R,B}|n_R||n_B|\left(k_3k_2\sqrt{3} - k_1k_3\right) + \rho_{W,B}|n_W||n_B|\left(-1.5{k_2}^2 + 0.5{k_1}^2\right) \end{bmatrix}$$

$$\begin{bmatrix} d_1 \\ d_2 \\ d_3 \\ d_4 \\ d_5 \\ d_6 \end{bmatrix} = \begin{bmatrix} -0.5k_1k_2\sqrt{3} + 0.75{k_1}^2 + 0.25{k_2}^2 \\ 0.5k_1k_2\sqrt{3} + 0.75{k_1}^2 + 0.25{k_2}^2 \\ {k_4}^2|n_R|(|n_R|-1) + d_1|n_W|(|n_W|-1) + d_2|n_B|(|n_B|-1) \\ |n_R||n_W|\left(k_4k_1\sqrt{3} - k_2k_4\right) + |n_R||n_B|\left(-k_4k_1\sqrt{3} - k_2k_4\right) + |n_W||n_B|\left(-1.5{k_1}^2 + 0.5{k_2}^2\right) \\ \rho_{R,R}{k_4}^2|n_R|(|n_R|-1) + \rho_{W,W}d_1|n_W|(|n_W|-1) + \rho_{B,B}d_2|n_B|(|n_B|-1) \\ \rho_{R,W}|n_R||n_W|\left(k_4k_1\sqrt{3} - k_2k_4\right) + \rho_{R,B}|n_R||n_B|\left(-k_4k_1\sqrt{3} - k_2k_4\right) + \rho_{W,B}|n_W||n_B|\left(-1.5{k_1}^2 + 0.5{k_2}^2\right) \end{bmatrix}$$

$$\begin{bmatrix} C_1 \\ C_2 \\ C_3 \\ C_4 \\ C_5 \\ C_6 \end{bmatrix} = \begin{bmatrix} 0.5(n_W + n_B) \\ 0.5\sqrt{3}(n_W - n_B) \\ {k_3}^2|n_R| + c_1|n_W| + c_2|n_B| + c_5 + c_6 \\ c_3 + c_4 - c_5 - c_6 \\ {k_4}^2|n_R| + d_1|n_W| + d_2|n_B| + d_5 + d_6 \\ d_3 + d_4 - d_5 - d_6 \end{bmatrix}$$

Step 6 – Calculate Voltage Drop Moments at Each Node

$$E(\Delta V_{x_i}) = (k_3 n_R - k_1 C_1 - k_2 C_2).G.C_B$$

$$E\left(\Delta V_{y_i}\right) = (k_4 n_R - k_2 C_1 + k_1 C_2).G.C_B$$

$$E\left(\Delta {V_{x_i}}^2\right) = {C_B}^2(C_3 H + C_4 G^2)$$

$$E\left(\Delta {V_{y_i}}^2\right) = {C_B}^2(C_5 H + C_6 G^2)$$

Step 7 – Calculate Summated Voltage Drop Moments at the End of the Feeder

$$E(\Delta V_{x_t}) = \sum_{i=1}^{N} E(\Delta V_{x_i})$$

$$E(\Delta V_{y_t}) = \sum_{i=1}^{N} E(\Delta V_{y_i})$$

$$E(\Delta {V_{x_t}}^2) = \sum_{i=1}^{N} E(\Delta {V_{x_i}}^2) + \sum_{j=1}^{N} \sum_{k=1,k\neq j}^{N} \left\{E(\Delta V_{x_j})E(\Delta V_{x_k}) + \left(\sigma_{V_{x_j}}.\sigma_{V_{x_k}}.\rho_{j,k}\right)\right\}$$

$$E\left(\Delta {V_{y_t}}^2\right) = \sum_{i=1}^{N} E\left(\Delta {V_{y_i}}^2\right) + \sum_{j=1}^{N} \sum_{k=1,k\neq j}^{N} \left\{E(\Delta V_{y_j})E(\Delta V_{y_k}) + \left(\sigma_{V_{y_j}}.\sigma_{V_{y_k}}.\rho_{j,k}\right)\right\}$$

Where: $$\sigma_{V_{x_j}} = \sqrt{E\left({V_{x_j}}^2\right) - E(V_{x_j})^2} \;\; ; \;\; \sigma_{V_{x_k}} = \sqrt{E\left({V_{x_k}}^2\right) - E(V_{x_k})^2}$$

$$\sigma_{V_{y_j}} = \sqrt{E\left({V_{y_j}}^2\right) - E(V_{y_j})^2} \;\; ; \;\; \sigma_{V_{y_k}} = \sqrt{E\left({V_{y_k}}^2\right) - E(V_{y_k})^2}$$

Step 8 – Calculate Consumer Voltage Moments

$$E(V_c) \approx V_s\left(1 - \frac{E\left(\Delta V_{x_t}\right)}{V_s} + 0.5\frac{E\left(\Delta {V_{y_t}}^2\right)}{{V_s}^2}\right)$$

$$E\left({V_c}^2\right) \approx {V_s}^2 - 2V_s E\left(\Delta V_{x_t}\right) + E\left(\Delta {V_{x_t}}^2\right) + E\left(\Delta {V_{y_t}}^2\right)$$

Step 9 – Calculate Consumer Voltage Beta PDF Parameters

$$\alpha_{V_{con}} = \frac{E\left({V_c}^2\right) - E(V_c)}{E(V_c) - \frac{E\left({V_c}^2\right)}{E(V_c)}}$$

$$\beta_{V_{con}} = \frac{\alpha_{V_{con}}}{E(V_c)} - \alpha_{V_{con}}$$

Step 10 – Calculate Consumer Voltage Corresponding to Chosen Confidence Level

$$V^*_{con\%} = betainv\left[\left(\frac{100 - \rho}{100}\right), \left|\alpha_{V_{con}}\right|, \left|\beta_{V_{con}}\right|\right]$$

Step 11 – Rescale Consumer Voltage

$$V_{con\%} = V^*_{con\%}.V_B$$

A3: PROCEDURE FOR USING THE HBET: 3-PHASE, 3-WIRE SYSTEMS

Step 1 – Select Network Parameters

- Specify the source/nominal voltage, denoted by V_s.
- Specify the per-unit base values for apparent power (S_B) and voltage (V_B).
- Provide the parameters describing the loads/generators at each node, namely α_i, β_i, C_i and PF_i.
- Specify the number of customers and generators connected at each node. The customers for the red, white and blue phase at node i are denoted by n_{R_i}, n_{W_i} and n_{B_i} respectively. Positive values are assigned to represent loads and negative values are used to represent generators.
- Specify the number of nodes in the radial network, denoted by N.
- Provide the resistance and reactance of the phase and neutral conductors between each node $(i-1)$ and node i, denoted by $R_{p_i}, R_{n_i}, X_{p_i}$ and X_{n_i}.
- Specify the $N \times N \times phases$ correlation matrix Rho for interphase, intraphase and internode correlation.
- Specify the confidence level as a percentage, denoted by ρ.

Step 2 – Determine Per-unit Equivalents of System Inputs

$$I_B = \frac{S_B}{V_B}\ ;\ Z_B = \frac{{V_B}^2}{S_B}$$

$$R_{p_B} = \frac{R_{p_i}}{Z_B};\ X_{p_B} = \frac{X_{p_i}}{Z_B};\ C_B = \frac{C_i}{I_B}$$

Step 3 – Determine Superposition Elements

$$R_i = \sum_{j=1}^{i} R_{p_{B_{(j)}}}\ ;\ X_i = \sum_{j=1}^{i} X_{p_{B_{(j)}}}$$

Step 4 – Calculate First and Second Moments for Input Currents

$$G_i = \frac{\alpha_i}{\alpha_i+\beta_i};\ H_i = \frac{\alpha_i(\alpha_i+1)}{\alpha_i+\beta_i(\alpha_i+\beta_i+1)};\ \emptyset_i = \text{acos}(PF_i)$$

Step 5 – Calculate Constants $w_1 - w_2;\ e_1 - e_6;\ f_1 - f_6;\ F_1 - F_6$

$$\begin{bmatrix} w_1 \\ w_2 \end{bmatrix} = \begin{bmatrix} R_i \pm X_i \tan \emptyset_i \\ X_i \mp R_i \tan \emptyset_i \end{bmatrix}$$

$$\begin{bmatrix} e_1 \\ e_2 \\ e_3 \\ e_4 \\ e_5 \\ e_6 \end{bmatrix} = \begin{bmatrix} 0.5w_1w_2\sqrt{3} + 0.75{w_2}^2 + 0.25{w_1}^2 \\ -0.5w_1w_2\sqrt{3} + 0.75{w_2}^2 + 0.25{w_1}^2 \\ 4{w_1}^2|n_R|(|n_R|-1) + e_1|n_W|(|n_W|-1) + e_2|n_B|(|n_B|-1) \\ |n_R||n_W|\left(2w_1w_2\sqrt{3} - 2{w_1}^2\right) + |n_R||n_B|\left(-2w_1w_2\sqrt{3} + 2{w_1}^2\right) + |n_W||n_B|(-1.5{w_2}^2 + 0.5{w_1}^2) \\ \rho_{R,R}4{w_1}^2|n_R|(|n_R|-1) + \rho_{W,W}e_1|n_W|(|n_W|-1) + \rho_{B,B}e_2|n_B|(|n_B|-1) \\ \rho_{R,W}|n_R||n_W|\left(2w_1w_2\sqrt{3} - 2{w_1}^2\right) + \rho_{R,B}|n_R||n_B|\left(-2w_1w_2\sqrt{3} + 2{w_1}^2\right) + \rho_{W,B}|n_W||n_B|(-1.5{w_2}^2 + 0.5{w_1}^2) \end{bmatrix}$$

$$\begin{bmatrix} f_1 \\ f_2 \\ f_3 \\ f_4 \\ f_5 \\ f_6 \end{bmatrix} = \begin{bmatrix} -0.5w_1w_2\sqrt{3} + 0.75{w_1}^2 + 0.25{w_2}^2 \\ 0.5w_1w_2\sqrt{3} + 0.75{w_1}^2 + 0.25{w_2}^2 \\ 4{w_2}^2|n_R|(|n_R| - 1) + f_1|n_W|(|n_W| - 1) + f_2|n_B|(|n_B| - 1) \\ |n_R||n_W|\left(-2w_1w_2\sqrt{3} - 2{w_2}^2\right) + |n_R||n_B|\left(2w_1w_2\sqrt{3} + 2{w_2}^2\right) + |n_W||n_B|(-1.5{w_1}^2 + 0.5{w_2}^2) \\ \rho_{R,R}4{w_2}^2|n_R|(|n_R| - 1) + \rho_{W,W}f_1|n_W|(|n_W| - 1) + \rho_{B,B}f_2|n_B|(|n_B| - 1) \\ \rho_{R,W}|n_R||n_W|\left(-2w_1w_2\sqrt{3} - 2{w_2}^2\right) + \rho_{R,B}|n_R||n_B|\left(2w_1w_2\sqrt{3} + 2{w_2}^2\right) + \rho_{W,B}|n_W||n_B|(-1.5{w_1}^2 + 0.5{w_2}^2) \end{bmatrix}$$

$$\begin{bmatrix} F_1 \\ F_2 \\ F_3 \\ F_4 \\ F_5 \\ F_6 \end{bmatrix} = \begin{bmatrix} 2n_R + 0.5(n_W + n_B) \\ 0.5\sqrt{3}(n_W - n_B) \\ 4{k_1}^2|n_R| + e_1|n_W| + e_2|n_B| + e_5 + e_6 \\ e_3 + e_4 - e_5 - e_6 \\ 4{k_2}^2|n_R| + f_1|n_W| + f_2|n_B| + f_5 + f_6 \\ f_3 + f_4 - f_5 - f_6 \end{bmatrix}$$

Step 6 – Calculate Voltage Drop Moments at Each Node

$$E(\Delta V_{x_i}) = (F_1w_1 + F_2w_2).G.C_B$$

$$E\left((\Delta V_{y_i}\right) = (F_2w_1 - F_1w_2).G.C_B$$

$$E\left(\Delta {V_{x_i}}^2\right) = {C_B}^2(F_3H + F_4G^2)$$

$$E\left(\Delta {V_{y_i}}^2\right) = {C_B}^2(F_5H + F_6G^2)$$

Step 7 – Calculate Summated Voltage Drop Moments at the End of the Feeder

$$E(\Delta V_{x_t}) = \sum_{i=1}^{N} E(\Delta V_{x_i})$$

$$E(\Delta V_{y_t}) = \sum_{i=1}^{N} E(\Delta V_{y_i})$$

$$E(\Delta {V_{x_t}}^2) = \sum_{i=1}^{N} E(\Delta {V_{x_i}}^2) + \sum_{j=1}^{N} \sum_{k=1,k\neq j}^{N} \left\{E(\Delta V_{x_j})E(\Delta V_{x_k}) + \left(\sigma_{V_{x_j}}.\sigma_{V_{x_k}}.\rho_{j,k}\right)\right\}$$

$$E\left(\Delta {V_{y_t}}^2\right) = \sum_{i=1}^{N} E\left(\Delta {V_{y_i}}^2\right) + \sum_{j=1}^{N} \sum_{k=1,k\neq j}^{N} \left\{E(\Delta V_{y_j})E(\Delta V_{y_k}) + \left(\sigma_{V_{y_j}}.\sigma_{V_{y_k}}.\rho_{j,k}\right)\right\}$$

Where:

$$\sigma_{V_{x_j}} = \sqrt{E\left({V_{x_j}}^2\right) - E(V_{x_j})^2} \;\; ; \;\; \sigma_{V_{x_k}} = \sqrt{E\left({V_{x_k}}^2\right) - E(V_{x_k})^2}$$

$$\sigma_{V_{y_j}} = \sqrt{E\left({V_{y_j}}^2\right) - E(V_{y_j})^2} \;\; ; \;\; \sigma_{V_{y_k}} = \sqrt{E\left({V_{y_k}}^2\right) - E(V_{y_k})^2}$$

Step 8 – Calculate Consumer Voltage Moments

$$E(V_c) \approx V_s\left(1 - \frac{E\left(\Delta V_{x_t}\right)}{V_s} + 0.5\frac{E\left(\Delta {V_{y_t}}^2\right)}{{V_s}^2}\right)$$

$$E\left(V_c{}^2\right) \approx V_s{}^2 - 2V_s E\left(\Delta V_{x_t}\right) + E\left(\Delta V_{x_t}{}^2\right) + E\left(\Delta V_{y_t}{}^2\right)$$

Step 9 – Calculate Consumer Voltage Beta PDF Parameters

$$\alpha_{V_{con}} = \frac{E\left(V_c{}^2\right) - E(V_c)}{E(V_c) - \frac{E\left(V_c{}^2\right)}{E(V_c)}}$$

$$\beta_{V_{con}} = \frac{\alpha_{V_{con}}}{E(V_c)} - \alpha_{V_{con}}$$

Step 10 – Calculate Consumer Voltage Corresponding to Chosen Confidence Level

$$V^*_{con\%} = betainv\left[\left(\frac{100-\rho}{100}\right), |\alpha_{V_{con}}|, |\beta_{V_{con}}|\right]$$

Step 11 – Rescale Consumer Voltage

$$V_{con\%} = V^*_{con\%}.V_B$$

Appendix B : Simulation Input Data

B1: Network Parameters for 12-bus System

From node	To node	Line Resistance [Ω]	Line Reactance [Ω]	Alpha	Beta	Cb [A]	PF	Phase Loading		
								Red	White	Blue
1	2	0.0922	0.0470	1.5	5	45.9408	0.857	1	0	1
2	3	0.4930	0.2512	1.5	5	35.9436	0.986	1	1	0
3	4	0.3661	0.1864	1.5	5	56.8147	0.832	0	1	1
4	5	0.3811	0.1941	1.5	5	26.4263	0.894	1	0	1
5	6	0.8190	0.7070	1.5	5	30.7677	0.768	1	1	0
6	7	0.1872	0.6188	1.5	5	88.0875	0.894	0	1	1
7	8	0.7115	0.2351	1.5	5	142.037	0.555	1	0	1
8	9	1.0299	0.7400	1.5	5	29.5455	0.8	1	1	0
9	10	1.0440	0.7400	1.5	5	24.9149	0.949	0	1	1
10	11	0.1967	0.0651	1.5	5	23.7183	0.747	1	0	1
11	12	0.3744	0.1298	1.5	5	27.3639	0.864	1	1	0

B2: Network Parameters for 33-bus System

From node	To node	Line Resistance [Ω]	Line Reactance [Ω]	Alpha	Beta	Cb [A]	PF	Phase Loading		
								Red	White	Blue
1	2	0.0922	0.0470	49.5	49.5	18.4232	0.85749	1	1	1
2	3	0.4930	0.2512	49.5	49.5	15.559	0.91381	1	1	1
3	4	0.3661	0.1864	49.5	49.5	22.7839	0.83205	1	1	1
4	5	0.3811	0.1941	49.5	49.5	10.5975	0.89443	1	1	1
5	6	0.8190	0.7070	49.5	49.5	9.9914	0.94868	1	1	1
6	7	0.1872	0.6188	49.5	49.5	35.3249	0.89443	1	1	1
7	8	0.7115	0.2351	49.5	49.5	35.3249	0.89443	1	1	1
8	9	1.0299	0.7400	49.5	49.5	9.9914	0.94868	1	1	1
9	10	1.0440	0.7400	49.5	49.5	9.9914	0.94868	1	1	1
10	11	0.1967	0.0651	49.5	49.5	8.54396	0.83205	1	1	1
11	12	0.3744	0.1298	49.5	49.5	10.9735	0.86378	1	1	1
12	13	1.4680	1.1549	49.5	49.5	10.9735	0.86378	1	1	1
13	14	0.5416	0.7129	49.5	49.5	22.7839	0.83205	1	1	1
14	15	0.5909	0.5260	49.5	49.5	9.60942	0.98639	1	1	1
15	16	0.7462	0.5449	49.5	49.5	9.9914	0.94868	1	1	1
16	17	1.2889	1.7210	49.5	49.5	9.9914	0.94868	1	1	1
17	18	0.7320	0.5739	49.5	49.5	15.559	0.91381	1	1	1
18	19	0.1640	0.1565	49.5	49.5	15.559	0.91381	1	1	1
19	20	1.5042	1.3555	49.5	49.5	15.559	0.91381	1	1	1
20	21	0.4095	0.4784	49.5	49.5	15.559	0.91381	1	1	1

21	22	0.7089	0.9373	49.5	49.5	15.559	0.91381	1	1	1
22	23	0.4512	0.3084	49.5	49.5	16.2648	0.87416	1	1	1
23	24	0.8980	0.7091	49.5	49.5	73.4894	0.90286	1	1	1
24	25	0.8959	0.7071	49.5	49.5	73.4894	0.90286	1	1	1
25	26	0.2031	0.1034	49.5	49.5	10.2686	0.92308	1	1	1
26	27	0.2842	0.1447	49.5	49.5	10.2686	0.92308	1	1	1
27	28	1.0589	0.9338	49.5	49.5	9.9914	0.94868	1	1	1
28	29	0.8043	0.7006	49.5	49.5	21.947	0.86378	1	1	1
29	30	0.5074	0.2585	49.5	49.5	99.914	0.31623	1	1	1
30	31	0.9745	0.9629	49.5	49.5	26.15	0.90618	1	1	1
31	32	0.3105	0.3619	49.5	49.5	36.7447	0.90286	1	1	1
32	33	0.3411	0.5302	49.5	49.5	11.3919	0.83205	1	1	1

Appendix C : Network Planning Tool Source Code

The network planning tool developed was created in Python and will be included with this thesis submission. For completeness, the code used to develop the program is provided below. Comments have been included to provide clarity on certain portions of code.

```
#Import modules and packages required in the program
from tkinter import messagebox
from tkinter.ttk import *
from math import *
from scipy.stats import beta
import matplotlib
matplotlib.use("TkAgg")
import matplotlib.pyplot as plt
import tkinter.font as tkFont
import os
from tkinter import *
import tkinter.filedialog
import xlrd
import copy

#Specifies the path and name of the input file
loc = ("Inputs12Bus_PassiveFinal.xlsx")

#Opens the spreadsheet
wb = xlrd.open_workbook(loc)
#Used to differentiate between different tabs in the spreadsheet
sheet = wb.sheet_by_index(0)
sheet2 = wb.sheet_by_index(1)

#Creates the main program window
window=Tk()
#Give the window a title (The program name in this case)
window.title("Herman-Beta Extended Transform Voltage Drop Calculation Program")

#Create the menu bar
menubar = Menu(window)

#Define global variables
global LV_ResistanceT1
global LV_ReactanceT1
global Proceed_Button
global Vbase_Entry
global Temp_Entry
global VoltageLimit_Spin
global ConfidenceLevel_Spin
```

```
#Gives a brief desciption of the program and its development when the 'About' option is selected from the
menubar
def About():
   About_Window=Toplevel(window,bg='white')
   About_Window.title("About the Program")
   About_Window.geometry('620x350')
   #Displays the background image with text in the About pop-up window. This was done using another program
since the transparency of background images cannot easily be set using Tkinter
   About_image=PhotoImage(file='AboutBackgroundwithText.png')
   About_label=Label(About_Window, image=About_image)
   About_label.place(x=0, y=0, relwidth=1, relheight=1)
   About_label.image = About_image


#Function used to exit the program
def Exit_Program():
   #Confirms that the user is sure about exiting the program before doing so
   QuitProgram=messagebox.askquestion("Quit Program","Are you sure you want to quit the program?")
   if(QuitProgram=='yes'):
      window.destroy()

#Create a new pop-up window to allow users to insert values to determine beta parameters based on ADMD,
supply voltage and circuit breaker rating
def BetaCalculator_Option1():
   #Define global variables to be used in the Calculate_Option1 function
   global ADMD_Entry,CB_Entry,SupplyVoltage_Entry,BetaCalculator1_Window
   #Create a top-level window above the main program window
   BetaCalculator1_Window=Toplevel(window)
   BetaCalculator1_Window.title("Beta Parameter Calculator")
   BetaCalculator1_Window.geometry('450x350')
   #Asks the user to insert the supply voltage, ADMD and CB rating
   SupplyVoltage_Prompt=Label(BetaCalculator1_Window,text="Supply Voltage
(V):",width=20).grid(row=1,column=0,padx=10,pady=10)
   SupplyVoltage_Entry=Entry(BetaCalculator1_Window,width=40)
   SupplyVoltage_Entry.grid(row=1,column=1,columnspan=2,padx=10,pady=10)
   ADMD_Prompt=Label(BetaCalculator1_Window,text="ADMD
(kVA):",width=20).grid(row=2,column=0,padx=10,pady=10)
   ADMD_Entry=Entry(BetaCalculator1_Window,width=40)
   ADMD_Entry.grid(row=2,column=1,columnspan=2,padx=10,pady=10)
   CB_Prompt=Label(BetaCalculator1_Window,text="Circuit Breaker Rating
(A):",width=20).grid(row=3,column=0,padx=10,pady=10)
   CB_Entry=Entry(BetaCalculator1_Window,width=40)
   CB_Entry.grid(row=3,column=1,columnspan=2,padx=10,pady=10)
   Calculate_Paramaters=Button(BetaCalculator1_Window,text='Calculate Beta Parameters',
command=Calculate_Option1).grid(row=5,column=1,columnspan=2,padx=10,pady=10)

#Function used to calculate Beta parameters using ADMD, CB rating and supply voltage
def Calculate_Option1():
   global ADMD_Entry,CB_Entry,SupplyVoltage_Entry,BetaCalculator1_Window
   try:
      #If all values entered are in the correct format, calculate the Beta PDF parameters
      if((float(ADMD_Entry.get()) and float(CB_Entry.get())) and float(SupplyVoltage_Entry.get())):
         C_V=1.0738*(float(ADMD_Entry.get())**(-0.3274))
         CB=float(CB_Entry.get())
```

```
        Vs=float(SupplyVoltage_Entry.get())
        ADMD=float(ADMD_Entry.get())
        Alpha=((CB*Vs-(1000*ADMD*((C_V**2)+1))))/(CB*Vs*(C_V**2))
        Beta=Alpha*(CB*Vs-(1000*ADMD))/(ADMD*1000)
        Font=tkFont.Font(family="Lucida Grande", size=11)
        Calculated_Alpha=Label(BetaCalculator1_Window,text="The calculated value for Alpha is :
"+str("{:.3f}".format(Alpha)),font=Font,width=40,fg='blue').grid(row=6,column=0,columnspan=4,pady=20)
        Calculated_Beta=Label(BetaCalculator1_Window,text="The calculated value for Beta is :
"+str("{:.3f}".format(Beta)),font=Font,width=40,fg='blue').grid(row=7,column=0,columnspan=4,padx=10,pady=
10)
    #Show error message if values have been entered in the incorrect format
    except ValueError:
        messagebox.showerror('Incorrect values entered!','Error! Please enter numerical values for ADMD, supply
voltage and circuit breaker rating')
        #Keeps the Beta parameter calculator window displayed after the error message window is closed.
        BetaCalculator1_Window.deiconify()

#Create a new pop-up window to allow users to insert values to determine beta parameters based on the load's
mean, standard deviation and the circuit breaker rating
def BetaCalculator_Option2():
    global Mean_Entry,CB_Entry,StdDev_Entry,BetaCalculator2_Window
    BetaCalculator2_Window=Toplevel(window)
    BetaCalculator2_Window.title("Beta Parameter Calculator")
    BetaCalculator2_Window.geometry('450x350')
    #Allows the user to enter the mean, standard deviation and CB rating to calculate Beta PDF parameters

Mean_Prompt=Label(BetaCalculator2_Window,text="Mean:",width=20).grid(row=1,column=0,padx=10,pady=1
0)
    Mean_Entry=Entry(BetaCalculator2_Window,width=40)
    Mean_Entry.grid(row=1,column=1,columnspan=2,padx=10,pady=10)
    StdDev_Prompt=Label(BetaCalculator2_Window,text="Standard
Deviation:",width=20).grid(row=2,column=0,padx=10,pady=10)
    StdDev_Entry=Entry(BetaCalculator2_Window,width=40)
    StdDev_Entry.grid(row=2,column=1,columnspan=2,padx=10,pady=10)
    CB_Prompt=Label(BetaCalculator2_Window,text="Circuit Breaker Rating
(A):",width=20).grid(row=3,column=0,padx=10,pady=10)
    CB_Entry=Entry(BetaCalculator2_Window,width=40)
    CB_Entry.grid(row=3,column=1,columnspan=2,padx=10,pady=10)
    Calculate_Paramaters=Button(BetaCalculator2_Window,text='Calculate Beta Parameters',
command=Calculate_Option2).grid(row=5,column=1,columnspan=2,padx=10,pady=10)

#Function used to calculate Beta parameters using load's mean, standard deviation and the CB rating
def Calculate_Option2():
    global Mean_Entry,CB_Entry,StdDev_Entry,BetaCalculator2_Window
    try:
        #If all values entered are in the correct format, perform the calculation
        if((float(Mean_Entry.get()) and float(StdDev_Entry.get())) and float(CB_Entry.get())):
            CB=float(CB_Entry.get())
            Mean=float(Mean_Entry.get())
            StdDev=float(StdDev_Entry.get())
            Alpha=Mean*((CB*Mean-Mean**2-StdDev**2)/CB/StdDev**2)
            Beta=(CB-Mean)*((CB*Mean-Mean**2-StdDev**2)/CB/StdDev**2)
            Font=tkFont.Font(family="Lucida Grande", size=11)
```

```
        Calculated_Alpha=Label(BetaCalculator2_Window,text="The calculated value for Alpha is :
"+str("{:.3f}".format(Alpha)),font=Font,width=40,fg='blue').grid(row=6,column=0,columnspan=4,pady=20)
        Calculated_Beta=Label(BetaCalculator2_Window,text="The calculated value for Beta is :
"+str("{:.3f}".format(Beta)),font=Font,width=40,fg='blue').grid(row=7,column=0,columnspan=4,padx=10,pady=
10)
    #Show error message if values have been entered in the incorrect format
    except ValueError:
      messagebox.showerror('Incorrect values entered!','Error! Please enter numerical values for the mean,
standard deviation and circuit breaker rating')
      #Keeps the window displayed after the error message window is closed.
      BetaCalculator2_Window.deiconify()

#Opens the user guide
def User_Guide():
    os.system('Guide.pdf')

#Function used to accept initial user inputs
def Proceed():
    #Define global variables to be used throughout the program
    global counter
    global Temp_Entered
    global Vbase_Entered, Sbase_Entered
    global Confidence_Level_Entered
    global VoltageLimit_Entered
    global SystemConfig_Entered
    global Vs_Entered
    ContinueLoop=0
    #Repeats the loop and keeps the program running if an error message is displayed to the user
    try:
      while(ContinueLoop==0):
        ContinueLoop=1
        #Shows an error message if the values entered are in the incorrect format or non-integer values are
received.
            if((Vs_Entry.get()=='' or not(isinstance(float(Vs_Entry.get()),float))) or (Vbase_Entry.get()=='' or
            not(isinstance(float(Vbase_Entry.get()),float)))  or ((Sbase_Entry.get()=='' or
            not(isinstance(float(Sbase_Entry.get()),float))) or (Temp_Entry.get()=='' or
            not(isinstance(float(Temp_Entry.get()),float))) or (not(ConfidenceLevel_Spin.get().isdigit()) or
            not(float(VoltageLimit_Spin.get().isdigit())))):
                    messagebox.showerror('Invalid Input Values','Error! Please enter integer values for all inputs')
        else:
            ContinueLoop==1
            #Store the values obtained from the user inputs as simplified variables.
            Vs_Entered=Vs_Entry.get()
            Vbase_Entered=Vbase_Entry.get()
            Sbase_Entered=Sbase_Entry.g et()
            Temp_Entered=Temp_Entry.get()
            VoltageLimit_Entered=VoltageLimit_Spin.get()
            Confidence_Level_Entered=ConfidenceLevel_Spin.get()
            SystemConfig_Entered=SystemConfig_Combo.get()
            Results()
    except ValueError:
      #Display error message if a non-numerical character has been inserted
      messagebox.showerror('Incorrect values entered!','Error! Please enter numerical values only')
```

```
#This function is used to read data out of the user input spreaadsheet and perform calculations based on the initial and spreadsheet inputs received.
def Results():
  #Define global variables
  global Confidence_Level_Entered,VoltageLimit_Entered,TotalResistance,TotalReactance
  global Vs_Entered,Vbase_Entered,Temp_Entered, Sbase_Entered, SystemConfig_Entered, Proceed_Button
  #Obtain laterals from the Inputs spreadsheet
  GetLaterals=sheet.cell_value(1,32)
  #Create array from user's laterals input
  arr = GetLaterals.split(';')
  Lats=[]
  for i in range(len(arr)):
    for j in range(len(arr[i])):
      try:
        if(j!=0):
          #Finds node numbers containing more than one digit and replaces the incorrectly stored single digit with the double digit node number
          if(arr[i][j]!=' ' and arr[i][j]!=']' and arr[i][j-1]!=' ' and arr[i][j-1]!='['):
            print(arr[i][j-1])
            print(arr[i][j])
            del(Lats[-1])
            Lats.append(int(str(arr[i][j-1])+str(arr[i][j])))
          else:
            Lats.append(int(arr[i][j]))
        else:
          Lats.append(int(arr[i][j]))
      except ValueError:
        pass
  #Store the laterals as a different variable which can be manipulated to differentiate between laterals
  Data=copy.deepcopy(Lats)
  #Determines the number of laterals in the user's system
  Laterals=int(len(Data)/3)
  #Create arrays to store calculation results
  #These arrays are used for moments of laterals only
  N_matrix_r_RedLaterals=[]
  N_matrix_r_WhiteLaterals=[]
  N_matrix_r_BlueLaterals=[]
  N_matrix_i_RedLaterals=[]
  N_matrix_i_WhiteLaterals=[]
  N_matrix_i_BlueLaterals=[]
  N_matrix_r_RedGenLaterals=[]
  N_matrix_i_RedGenLaterals=[]
  N_matrix_r_WhiteGenLaterals=[]
  N_matrix_i_WhiteGenLaterals=[]
  N_matrix_r_BlueGenLaterals=[]
  N_matrix_i_BlueGenLaterals=[]
  N2_matrix_r_RedLaterals=[]
  N2_matrix_r_WhiteLaterals=[]
  N2_matrix_r_BlueLaterals=[]
  N2_matrix_i_RedLaterals=[]
  N2_matrix_i_WhiteLaterals=[]
  N2_matrix_i_BlueLaterals=[]
  N2_matrix_r_RedGenLaterals=[]
```

```
N2_matrix_i_RedGenLaterals=[]
N2_matrix_r_WhiteGenLaterals=[]
N2_matrix_i_WhiteGenLaterals=[]
N2_matrix_r_BlueGenLaterals=[]
N2_matrix_i_BlueGenLaterals=[]
#Stores values of moments for previous nodes (before node 'i')
N_matrix_r_Past_Red=[]
N_matrix_r_Past_RedGen=[]
N_matrix_r_Past_White=[]
N_matrix_r_Past_WhiteGen=[]
N_matrix_r_Past_Blue=[]
N_matrix_r_Past_BlueGen=[]
N_matrix_i_Past_Red=[]
N_matrix_i_Past_RedGen=[]
N_matrix_i_Past_White=[]
N_matrix_i_Past_WhiteGen=[]
N_matrix_i_Past_Blue=[]
N_matrix_i_Past_BlueGen=[]
N2_matrix_r_Past_Red=[]
N2_matrix_r_Past_RedGen=[]
N2_matrix_i_Past_Red=[]
N2_matrix_i_Past_RedGen=[]
N2_matrix_r_Past_White=[]
N2_matrix_r_Past_WhiteGen=[]
N2_matrix_i_Past_White=[]
N2_matrix_i_Past_WhiteGen=[]
N2_matrix_r_Past_Blue=[]
N2_matrix_r_Past_BlueGen=[]
N2_matrix_i_Past_Blue=[]
N2_matrix_i_Past_BlueGen=[]
#Past moments of the entire feeder
N_matrix_r_Past_RedFinal=[]
N_matrix_r_Past_RedGenFinal=[]
N_matrix_r_Past_WhiteFinal=[]
N_matrix_r_Past_WhiteGenFinal=[]
N_matrix_r_Past_BlueFinal=[]
N_matrix_r_Past_BlueGenFinal=[]
N_matrix_i_Past_RedFinal=[]
N_matrix_i_Past_RedGenFinal=[]
N_matrix_i_Past_WhiteFinal=[]
N_matrix_i_Past_WhiteGenFinal=[]
N_matrix_i_Past_BlueFinal=[]
N_matrix_i_Past_BlueGenFinal=[]
N2_matrix_r_Past_RedFinal=[]
N2_matrix_r_Past_RedGenFinal=[]
N2_matrix_i_Past_RedFinal=[]
N2_matrix_i_Past_RedGenFinal=[]
N2_matrix_r_Past_WhiteFinal=[]
N2_matrix_r_Past_WhiteGenFinal=[]
N2_matrix_i_Past_WhiteFinal=[]
N2_matrix_i_Past_WhiteGenFinal=[]
N2_matrix_r_Past_BlueFinal=[]
N2_matrix_r_Past_BlueGenFinal=[]
N2_matrix_i_Past_BlueFinal=[]
```

```
N2_matrix_i_Past_BlueGenFinal=[]
#Past moments for laterals
N_matrix_r_Past_RedLaterals=[]
N_matrix_r_Past_RedGenLaterals=[]
N_matrix_r_Past_WhiteLaterals=[]
N_matrix_r_Past_WhiteGenLaterals=[]
N_matrix_r_Past_BlueLaterals=[]
N_matrix_r_Past_BlueGenLaterals=[]
N_matrix_i_Past_RedLaterals=[]
N_matrix_i_Past_RedGenLaterals=[]
N_matrix_i_Past_WhiteLaterals=[]
N_matrix_i_Past_WhiteGenLaterals=[]
N_matrix_i_Past_BlueLaterals=[]
N_matrix_i_Past_BlueGenLaterals=[]
N2_matrix_r_Past_RedLaterals=[]
N2_matrix_r_Past_RedGenLaterals=[]
N2_matrix_i_Past_RedLaterals=[]
N2_matrix_i_Past_RedGenLaterals=[]
N2_matrix_r_Past_WhiteLaterals=[]
N2_matrix_r_Past_WhiteGenLaterals=[]
N2_matrix_i_Past_WhiteLaterals=[]
N2_matrix_i_Past_WhiteGenLaterals=[]
N2_matrix_r_Past_BlueLaterals=[]
N2_matrix_r_Past_BlueGenLaterals=[]
N2_matrix_i_Past_BlueLaterals=[]
N2_matrix_i_Past_BlueGenLaterals=[]
N_matrix_r_Past_RedAll=[]
N_matrix_r_Past_RedGenAll=[]
N_matrix_r_Past_WhiteAll=[]
N_matrix_r_Past_WhiteGenAll=[]
N_matrix_r_Past_BlueAll=[]
N_matrix_r_Past_BlueGenAll=[]
N_matrix_i_Past_RedAll=[]
N_matrix_i_Past_RedGenAll=[]
N_matrix_i_Past_WhiteAll=[]
N_matrix_i_Past_WhiteGenAll=[]
N_matrix_i_Past_BlueAll=[]
N_matrix_i_Past_BlueGenAll=[]
N2_matrix_r_Past_RedAll=[]
N2_matrix_r_Past_RedGenAll=[]
N2_matrix_i_Past_RedAll=[]
N2_matrix_i_Past_RedGenAll=[]
N2_matrix_r_Past_WhiteAll=[]
N2_matrix_r_Past_WhiteGenAll=[]
N2_matrix_i_Past_WhiteAll=[]
N2_matrix_i_Past_WhiteGenAll=[]
N2_matrix_r_Past_BlueAll=[]
N2_matrix_r_Past_BlueGenAll=[]
N2_matrix_i_Past_BlueAll=[]
N2_matrix_i_Past_BlueGenAll=[]
#Stores the moments of nodes located before the lateral connection point in seperate arrays
N_matrix_r_Pre_RedLaterals=[]
N_matrix_r_Pre_RedGenLaterals=[]
N_matrix_r_Pre_WhiteLaterals=[]
```

```
    N_matrix_r_Pre_WhiteGenLaterals=[]
    N_matrix_r_Pre_BlueLaterals=[]
    N_matrix_r_Pre_BlueGenLaterals=[]
    N_matrix_i_Pre_RedLaterals=[]
    N_matrix_i_Pre_RedGenLaterals=[]
    N_matrix_i_Pre_WhiteLaterals=[]
    N_matrix_i_Pre_WhiteGenLaterals=[]
    N_matrix_i_Pre_BlueLaterals=[]
    N_matrix_i_Pre_BlueGenLaterals=[]
    N2_matrix_r_Pre_RedLaterals=[]
    N2_matrix_r_Pre_RedGenLaterals=[]
    N2_matrix_i_Pre_RedLaterals=[]
    N2_matrix_i_Pre_RedGenLaterals=[]
    N2_matrix_r_Pre_WhiteLaterals=[]
    N2_matrix_r_Pre_WhiteGenLaterals=[]
    N2_matrix_i_Pre_WhiteLaterals=[]
    N2_matrix_i_Pre_WhiteGenLaterals=[]
    N2_matrix_r_Pre_BlueLaterals=[]
    N2_matrix_r_Pre_BlueGenLaterals=[]
    N2_matrix_i_Pre_BlueLaterals=[]
    N2_matrix_i_Pre_BlueGenLaterals=[]
    #To be used to calculate the sum of moments up to the node of concern
    N_matrix_r_Past_RedSum=0
    N_matrix_r_Past_RedGenSum=0
    N_matrix_r_Past_WhiteSum=0
    N_matrix_r_Past_WhiteGenSum=0
    N_matrix_r_Past_BlueSum=0
    N_matrix_r_Past_BlueGenSum=0
    N_matrix_i_Past_RedSum=0
    N_matrix_i_Past_RedGenSum=0
    N_matrix_i_Past_WhiteSum=0
    N_matrix_i_Past_WhiteGenSum=0
    N_matrix_i_Past_BlueSum=0
    N_matrix_i_Past_BlueGenSum=0
    N2_matrix_r_Past_RedSum=0
    N2_matrix_r_Past_RedGenSum=0
    N2_matrix_i_Past_RedSum=0
    N2_matrix_i_Past_RedGenSum=0
    N2_matrix_r_Past_WhiteSum=0
    N2_matrix_r_Past_WhiteGenSum=0
    N2_matrix_i_Past_WhiteSum=0
    N2_matrix_i_Past_WhiteGenSum=0
    N2_matrix_r_Past_BlueSum=0
    N2_matrix_r_Past_BlueGenSum=0
    N2_matrix_i_Past_BlueSum=0
    N2_matrix_i_Past_BlueGenSum=0
    count=0
    #These arrays will be used to retrieve the results from seperate laterals and combine it into one 'feeder' for
plotting
    FirstMoment_Vr_Red_SumFeeder=[]
    FirstMoment_Vi_Red_SumFeeder=[]
    SecondMoment_Vr_Red_SumFeeder=[]
    SecondMoment_Vi_Red_SumFeeder=[]
    FirstMoment_Vr_White_SumFeeder=[]
```

```
FirstMoment_Vi_White_SumFeeder=[]
SecondMoment_Vr_White_SumFeeder=[]
SecondMoment_Vi_White_SumFeeder=[]
FirstMoment_Vr_Blue_SumFeeder=[]
FirstMoment_Vi_Blue_SumFeeder=[]
SecondMoment_Vr_Blue_SumFeeder=[]
SecondMoment_Vi_Blue_SumFeeder=[]
B1TotalRedFeeder=[]
B2TotalRedFeeder=[]
B1TotalWhiteFeeder=[]
B2TotalWhiteFeeder=[]
B1TotalBlueFeeder=[]
B2TotalBlueFeeder=[]
for h in range(Laterals):
  #Set initial values for variables
  if(h>0):
    Data=copy.deepcopy(Lats[3*h:])
  Iteration=[]
  #Will be used to help determine where zeroes should be placed in moment arrays (due to laterals)
  Iteration.append(Data[0]-2+3)
  if(h==0):
    for g in range(Data[5],sheet.nrows-1):
      Iteration.append(g-2+3)
  if(h>0):
    for q in range(2,Data[0]-2+3):
      if(q==Data[0]):
        pass
      else:
        Iteration.append(q-2+3)
    for g in range(Data[5],Data[6]+1):
      Iteration.append(g-2+3)
  Iteration.sort()
  #Reset these values for each lateral
  SBase=0
  IBase=0
  ZBase=0
  RpBase=0
  XpBase=0
  RnBase=0
  XnBase=0
  RpBaseGen=0
  XpBaseGen=0
  RnBaseGen=0
  XnBaseGen=0
  Vs_pu=0
  TotalResistance=0
  TotalResistanceGen=0
  TotalReactance=0
  TotalReactanceGen=0
  Vi_Min_Gen_Red=0
  Vi_Min_Gen_White=0
  Vi_Min_Gen_Blue=0
  #Create arrays to store calculation results
  UpperVoltageLimit=[]
```

LowerVoltageLimit=[]
CbBase=[]
CbBaseGen=[]
CbBaseLumped=[]
CbBaseGenLumped=[]
Theta=[]
ThetaLumped=[]
ThetaGen=[]
ThetaGenLumped=[]
K=[]
GenTable_Entries=[]
ResistanceT1=[]
ReactanceT1=[]
ResistanceT2=[]
ReactanceT2=[]
Kr_Effective=[]
Kx_Effective=[]
Kr_EffectiveGen=[]
Kx_EffectiveGen=[]
ScalingFactorNew=[]
NodeResistance=[]
NodeReactance=[]
NodeReactanceGen=[]
H=[]
GSquared=[]
HLumped=[]
GSquaredLumped=[]
k1=[]
k2=[]
k3=[]
k4=[]
k1Gen=[]
k2Gen=[]
k3Gen=[]
k4Gen=[]
c1=[]
c2=[]
c3_Red=[]
c4_Red=[]
c1Gen=[]
c2Gen=[]
c3_RedGen=[]
c4_RedGen=[]
d1=[]
d2=[]
d3_Red=[]
d4_Red=[]
d1Gen=[]
d2Gen=[]
d3_RedGen=[]
d4_RedGen=[]
C1_Red=[]
C2_Red=[]
C3_Red=[]

C4_Red=[]
C5_Red=[]
C6_Red=[]
C1_RedGen=[]
C2_RedGen=[]
C3_RedGen=[]
C4_RedGen=[]
C5_RedGen=[]
C6_RedGen=[]
c3_White=[]
c4_White=[]
c3_WhiteGen=[]
c4_WhiteGen=[]
d3_White=[]
d4_White=[]
d3_WhiteGen=[]
d4_WhiteGen=[]
C1_White=[]
C2_White=[]
C3_White=[]
C4_White=[]
C5_White=[]
C6_White=[]
C1_WhiteGen=[]
C2_WhiteGen=[]
C3_WhiteGen=[]
C4_WhiteGen=[]
C5_WhiteGen=[]
C6_WhiteGen=[]
c3_Blue=[]
c4_Blue=[]
c3_BlueGen=[]
c4_BlueGen=[]
d3_Blue=[]
d4_Blue=[]
d3_BlueGen=[]
d4_BlueGen=[]
C1_Blue=[]
C2_Blue=[]
C3_Blue=[]
C4_Blue=[]
C5_Blue=[]
C6_Blue=[]
C1_BlueGen=[]
C2_BlueGen=[]
C3_BlueGen=[]
C4_BlueGen=[]
C5_BlueGen=[]
C6_BlueGen=[]
FirstMoment_Vr_Red=[]
FirstMoment_Vr_RedLumped=[]
SecondMoment_Vr_Red=[]
SecondMoment_Vr_RedLumped=[]
FirstMoment_Vi_Red=[]

FirstMoment_Vi_RedLumped=[]
SecondMoment_Vi_Red=[]
SecondMoment_Vi_RedLumped=[]
FirstMoment_Vr_Red_Sum=[]
FirstMoment_Vi_Red_Sum=[]
SecondMoment_Vr_Red_Sum=[]
SecondMoment_Vi_Red_Sum=[]
NVr_Red=0
NVrSquared_Red=0
NVi_Red=0
NViSquared_Red=0
FirstMoment_Vc_Red=[]
SecondMoment_Vc_Red=[]
NormalisedAlpha_Red=[]
NormalisedBeta_Red=[]
Percentile_Vc_Red=[]
Percentile_Vc_Gen_Red=[]
FirstMoment_Vr_White=[]
FirstMoment_Vr_WhiteLumped=[]
SecondMoment_Vr_White=[]
SecondMoment_Vr_WhiteLumped=[]
FirstMoment_Vi_White=[]
FirstMoment_Vi_WhiteLumped=[]
SecondMoment_Vi_White=[]
SecondMoment_Vi_WhiteLumped=[]
FirstMoment_Vr_White_Sum=[]
FirstMoment_Vi_White_Sum=[]
SecondMoment_Vr_White_Sum=[]
SecondMoment_Vi_White_Sum=[]
NVr_White=0
NVrSquared_White=0
NVi_White=0
NViSquared_White=0
FirstMoment_Vc_White=[]
SecondMoment_Vc_White=[]
NormalisedAlpha_White=[]
NormalisedBeta_White=[]
Percentile_Vc_White=[]
FirstMoment_Vr_Blue=[]
FirstMoment_Vr_BlueLumped=[]
SecondMoment_Vr_Blue=[]
SecondMoment_Vr_BlueLumped=[]
FirstMoment_Vi_Blue=[]
FirstMoment_Vi_BlueLumped=[]
SecondMoment_Vi_Blue=[]
SecondMoment_Vi_BlueLumped=[]
FirstMoment_Vr_Blue_Sum=[]
FirstMoment_Vi_Blue_Sum=[]
SecondMoment_Vr_Blue_Sum=[]
SecondMoment_Vi_Blue_Sum=[]
NVr_Blue=0
NVrSquared_Blue=0
NVi_Blue=0
NViSquared_Blue=0

FirstMoment_Vc_Blue=[]
SecondMoment_Vc_Blue=[]
NormalisedAlpha_Blue=[]
NormalisedBeta_Blue=[]
Percentile_Vc_Blue=[]
Percentile_Vc_Gen_Blue=[]
Loads_PhaseA=[]
Loads_PhaseB=[]
Loads_PhaseC=[]
Loads_PhaseALumped=[]
Loads_PhaseBLumped=[]
Loads_PhaseCLumped=[]
Alpha=[]
Beta=[]
ScalingFactor=[]
PowerFactor=[]
AlphaLumped=[]
BetaLumped=[]
ScalingFactorLumped=[]
PowerFactorLumped=[]
CableLength=[]
Gens_PhaseA=[]
Gens_PhaseB=[]
Gens_PhaseC=[]
Gens_PhaseALumped=[]
Gens_PhaseBLumped=[]
Gens_PhaseCLumped=[]
GenAlpha=[]
GenBeta=[]
GenAlphaLumped=[]
GenBetaLumped=[]
GenScalingFactor=[]
GenScalingFactorLumped=[]
GenScalingFactorNew=[]
GenPowerFactor=[]
GenPowerFactorLumped=[]
GenCableLength=[]
ResistanceT1Gen=[]
ReactanceT1Gen=[]
ResistanceT2Gen=[]
ReactanceT2Gen=[]
NodeResistanceGen=[]
HGen=[]
GSquaredGen=[]
HGenLumped=[]
GSquaredGenLumped=[]
FirstMoment_Vr_Gen_Red=[]
FirstMoment_Vr_Gen_RedLumped=[]
SecondMoment_Vr_Gen_Red=[]
SecondMoment_Vr_Gen_RedLumped=[]
FirstMoment_Vi_Gen_Red=[]
FirstMoment_Vi_Gen_RedLumped=[]
SecondMoment_Vi_Gen_Red=[]
SecondMoment_Vi_Gen_RedLumped=[]

FirstMoment_Vr_Gen_Red_Sum=0
FirstMoment_Vi_Gen_Red_Sum=0
NVr_Gen_Red=0
NVrSquared_Gen_Red=0
NVi_Gen_Red=0
NViSquared_Gen_Red=0
FirstMoment_Vc_Gen_Red=[]
SecondMoment_Vc_Gen_Red=[]
NormalisedAlpha_Gen_Red=[]
NormalisedBeta_Gen_Red=[]
D_Risk=0
Percentile_Vc_Gen_Red=[]
V_Con_Red=[float(Vs_Entered)]
V_Con_Gen_Red=[float(Vs_Entered)]
FirstMoment_Vr_Gen_White=[]
FirstMoment_Vr_Gen_WhiteLumped=[]
SecondMoment_Vr_Gen_White=[]
SecondMoment_Vr_Gen_WhiteLumped=[]
FirstMoment_Vi_Gen_White=[]
FirstMoment_Vi_Gen_WhiteLumped=[]
SecondMoment_Vi_Gen_White=[]
SecondMoment_Vi_Gen_WhiteLumped=[]
FirstMoment_Vr_Gen_White_Sum=0
FirstMoment_Vi_Gen_White_Sum=0
NVr_Gen_White=0
NVrSquared_Gen_White=0
NVi_Gen_White=0
NViSquared_Gen_White=0
FirstMoment_Vc_Gen_White=[]
SecondMoment_Vc_Gen_White=[]
NormalisedAlpha_Gen_White=[]
NormalisedBeta_Gen_White=[]
Percentile_Vc_Gen_White=[]
V_Con_White=[float(Vs_Entered)]
V_Con_Gen_White=[float(Vs_Entered)]
FirstMoment_Vr_Gen_Blue=[]
FirstMoment_Vr_Gen_BlueLumped=[]
SecondMoment_Vr_Gen_Blue=[]
SecondMoment_Vr_Gen_BlueLumped=[]
FirstMoment_Vi_Gen_Blue=[]
FirstMoment_Vi_Gen_BlueLumped=[]
SecondMoment_Vi_Gen_Blue=[]
SecondMoment_Vi_Gen_BlueLumped=[]
FirstMoment_Vr_Gen_Blue_Sum=0
FirstMoment_Vi_Gen_Blue_Sum=0
NVr_Gen_Blue=0
NVrSquared_Gen_Blue=0
NVi_Gen_Blue=0
NViSquared_Gen_Blue=0
FirstMoment_Vc_Gen_Blue=[]
SecondMoment_Vc_Gen_Blue=[]
NormalisedAlpha_Gen_Blue=[]
NormalisedBeta_Gen_Blue=[]
Percentile_Vc_Blue=[]

```
Percentile_Vc_Gen_Blue=[]
V_Con_Blue=[float(Vs_Entered)]
V_Con_Gen_Blue=[float(Vs_Entered)]
KGen=[]
TotalDistance=[0]
Distance=0
IMean_Red=[]
IMeanSquared_Red=[]
Sum_IMean_Red=0
FirstMoment_IRed=0
SecondMoment_IRed=0
NormalisedFirstMoment_IRed=[]
NormalisedSecondMoment_IRed=[]
NormalisedAlpha_IRed=[]
NormalisedBeta_IRed=[]
NormalisedPercentile_IRed=[]
Percentile_IRed=[]
IMax_Red=0
IMin_Red=0
IMean_Red_Gen=[]
IMeanSquared_Red_Gen=[]
Sum_IMean_Red_Gen=0
FirstMoment_IRed_Gen=0
SecondMoment_IRed_Gen=0
NormalisedFirstMoment_IRed_Gen=[]
NormalisedSecondMoment_IRed_Gen=[]
NormalisedAlpha_IRed_Gen=[]
NormalisedBeta_IRed_Gen=[]
NormalisedPercentile_IRed_Gen=[]
Percentile_IRed_Gen=[]
IMax_Red_Gen=0
IMin_Red_Gen=0
IMean_White=[]
IMeanSquared_White=[]
Sum_IMean_White=0
FirstMoment_IWhite=0
SecondMoment_IWhite=0
NormalisedFirstMoment_IWhite=[]
NormalisedSecondMoment_IWhite=[]
NormalisedAlpha_IWhite=[]
NormalisedBeta_IWhite=[]
NormalisedPercentile_IWhite=[]
Percentile_IWhite=[]
IMax_White=0
IMean_White_Gen=[]
IMeanSquared_White_Gen=[]
Sum_IMean_White_Gen=0
FirstMoment_IWhite_Gen=0
SecondMoment_IWhite_Gen=0
NormalisedFirstMoment_IWhite_Gen=[]
NormalisedSecondMoment_IWhite_Gen=[]
NormalisedAlpha_IWhite_Gen=[]
NormalisedBeta_IWhite_Gen=[]
NormalisedPercentile_IWhite_Gen=[]
```

Percentile_IWhite_Gen=[]
IMax_White_Gen=0
IMean_Blue=[]
IMeanSquared_Blue=[]
Sum_IMean_Blue=0
FirstMoment_IBlue=0
SecondMoment_IBlue=0
NormalisedFirstMoment_IBlue=[]
NormalisedSecondMoment_IBlue=[]
NormalisedAlpha_IBlue=[]
NormalisedBeta_IBlue=[]
NormalisedPercentile_IBlue=[]
Percentile_IBlue=[]
IMax_Blue=0
IMean_Blue_Gen=[]
IMeanSquared_Blue_Gen=[]
Sum_IMean_Blue_Gen=0
FirstMoment_IBlue_Gen=0
SecondMoment_IBlue_Gen=0
NormalisedFirstMoment_IBlue_Gen=[]
NormalisedSecondMoment_IBlue_Gen=[]
NormalisedAlpha_IBlue_Gen=[]
NormalisedBeta_IBlue_Gen=[]
NormalisedPercentile_IBlue_Gen=[]
Percentile_IBlue_Gen=[]
IMax_Blue_Gen=0
a=0
N_matrix_r_Red=[]
N_matrix_r_White=[]
N_matrix_r_Blue=[]
N_matrix_i_Red=[]
N_matrix_i_White=[]
N_matrix_i_Blue=[]
N_matrix_r_RedGen=[]
N_matrix_r_WhiteGen=[]
N_matrix_r_BlueGen=[]
N_matrix_i_RedGen=[]
N_matrix_i_WhiteGen=[]
N_matrix_i_BlueGen=[]
N2_matrix_r_Red=[]
N2_matrix_r_RedGen=[]
N2_matrix_i_Red=[]
N2_matrix_i_RedGen=[]
B1TempRed=[]
B1Red=[]
B1TempWhite=[]
B1White=[]
B1TempBlue=[]
B1Blue=[]
B2TempRed=[]
B2Red=[]
B2TempWhite=[]
B2White=[]
B2TempBlue=[]

B2Blue=[]
B1TempRedGen=[]
B1RedGen=[]
B1TempWhiteGen=[]
B1WhiteGen=[]
B1TempBlueGen=[]
B1BlueGen=[]
B2TempRedGen=[]
B2RedGen=[]
B2TempWhiteGen=[]
B2WhiteGen=[]
B2TempBlueGen=[]
B2BlueGen=[]
B1TotalTempRed=[]
B1TotalRed=[]
B1TotalTempWhite=[]
B1TotalWhite=[]
B1TotalTempBlue=[]
B1TotalBlue=[]
B2TotalTempRed=[]
B2TotalRed=[]
B2TotalTempWhite=[]
B2TotalWhite=[]
B2TotalTempBlue=[]
B2TotalBlue=[]
B1TotalTempRedGen=[]
B1TotalRedGen=[]
B1TotalTempWhiteGen=[]
B1TotalWhiteGen=[]
B1TotalTempBlueGen=[]
B1TotalBlueGen=[]
B2TotalTempRedGen=[]
B2TotalRedGen=[]
B2TotalTempWhiteGen=[]
B2TotalWhiteGen=[]
B2TotalTempBlueGen=[]
B2TotalBlueGen=[]
B1RedPast=[]
B1TempRedPast=[]
B1TempWhitePast=[]
B1TempBluePast=[]
B2TempRedPast=[]
B2TempWhitePast=[]
B2TempBluePast=[]
B1TempRedGenPast=[]
B2TempRedGenPast=[]
B1TempWhiteGenPast=[]
B2TempWhiteGenPast=[]
B1TempBlueGenPast=[]
B2TempBlueGenPast=[]
MeanRed=[5]
MeanWhite=[5]
MeanBlue=[5]
StdDevRed=[0]

StdDevWhite=[0]
StdDevBlue=[0]
n=sheet.nrows-3
Nodes=[]
N_matrix_r_Past_RedFinal=[]
N_matrix_r_Past_RedGenFinal=[]
N_matrix_r_Past_WhiteFinal=[]
N_matrix_r_Past_WhiteGenFinal=[]
N_matrix_r_Past_BlueFinal=[]
N_matrix_r_Past_BlueGenFinal=[]
N_matrix_i_Past_RedFinal=[]
N_matrix_i_Past_RedGenFinal=[]
N_matrix_i_Past_WhiteFinal=[]
N_matrix_i_Past_WhiteGenFinal=[]
N_matrix_i_Past_BlueFinal=[]
N_matrix_i_Past_BlueGenFinal=[]
N2_matrix_r_Past_RedFinal=[]
N2_matrix_r_Past_RedGenFinal=[]
N2_matrix_i_Past_RedFinal=[]
N2_matrix_i_Past_RedGenFinal=[]
N2_matrix_r_Past_WhiteFinal=[]
N2_matrix_r_Past_WhiteGenFinal=[]
N2_matrix_i_Past_WhiteFinal=[]
N2_matrix_i_Past_WhiteGenFinal=[]
N2_matrix_r_Past_BlueFinal=[]
N2_matrix_r_Past_BlueGenFinal=[]
N2_matrix_i_Past_BlueFinal=[]
N2_matrix_i_Past_BlueGenFinal=[]
N_matrix_r_Past_RedSum=0
N_matrix_r_Past_RedGenSum=0
N_matrix_r_Past_WhiteSum=0
N_matrix_r_Past_WhiteGenSum=0
N_matrix_r_Past_BlueSum=0
N_matrix_r_Past_BlueGenSum=0
N_matrix_i_Past_RedSum=0
N_matrix_i_Past_RedGenSum=0
N_matrix_i_Past_WhiteSum=0
N_matrix_i_Past_WhiteGenSum=0
N_matrix_i_Past_BlueSum=0
N_matrix_i_Past_BlueGenSum=0
N2_matrix_r_Past_RedSum=0
N2_matrix_r_Past_RedGenSum=0
N2_matrix_i_Past_RedSum=0
N2_matrix_i_Past_RedGenSum=0
N2_matrix_r_Past_WhiteSum=0
N2_matrix_r_Past_WhiteGenSum=0
N2_matrix_i_Past_WhiteSum=0
N2_matrix_i_Past_WhiteGenSum=0
N2_matrix_r_Past_BlueSum=0
N2_matrix_r_Past_BlueGenSum=0
N2_matrix_i_Past_BlueSum=0
N2_matrix_i_Past_BlueGenSum=0
N_matrix_r_Past_RedAll=[]
N_matrix_r_Past_RedGenAll=[]

```
N_matrix_r_Past_WhiteAll=[]
N_matrix_r_Past_WhiteGenAll=[]
N_matrix_r_Past_BlueAll=[]
N_matrix_r_Past_BlueGenAll=[]
N_matrix_i_Past_RedAll=[]
N_matrix_i_Past_RedGenAll=[]
N_matrix_i_Past_WhiteAll=[]
N_matrix_i_Past_WhiteGenAll=[]
N_matrix_i_Past_BlueAll=[]
N_matrix_i_Past_BlueGenAll=[]
N2_matrix_r_Past_RedAll=[]
N2_matrix_r_Past_RedGenAll=[]
N2_matrix_i_Past_RedAll=[]
N2_matrix_i_Past_RedGenAll=[]
N2_matrix_r_Past_WhiteAll=[]
N2_matrix_r_Past_WhiteGenAll=[]
N2_matrix_i_Past_WhiteAll=[]
N2_matrix_i_Past_WhiteGenAll=[]
N2_matrix_r_Past_BlueAll=[]
N2_matrix_r_Past_BlueGenAll=[]
N2_matrix_i_Past_BlueAll=[]
N2_matrix_i_Past_BlueGenAll=[]
#Lumps inputs back to previous nodes for superposition
for i in (Iteration):
    Loads_PhaseA.append(sheet.cell_value(i, 13))
    Loads_PhaseB.append(sheet.cell_value(i, 14))
    Loads_PhaseC.append(sheet.cell_value(i, 15))
    Gens_PhaseA.append(sheet.cell_value(i, 20)*-1)
    Gens_PhaseB.append(sheet.cell_value(i, 21)*-1)
    Gens_PhaseC.append(sheet.cell_value(i, 22)*-1)
    AlphaLumped.append(sheet.cell_value(i, 9))
    BetaLumped.append(sheet.cell_value(i, 10))
    GenAlphaLumped.append(sheet.cell_value(i, 16))
    GenBetaLumped.append(sheet.cell_value(i, 17))
    ScalingFactorLumped.append(sheet.cell_value(i, 11))
    GenScalingFactorLumped.append(sheet.cell_value(i, 18))
    PowerFactorLumped.append(sheet.cell_value(i, 12))
    GenPowerFactorLumped.append(sheet.cell_value(i, 19))
    ThetaLumped.append(acos(sheet.cell_value(i, 12)))
    ThetaGenLumped.append(acos(sheet.cell_value(i, 19)))
for i in (Iteration):
    Alpha.append(sheet.cell_value(i, 9))
    Beta.append(sheet.cell_value(i, 10))
    ScalingFactor.append(sheet.cell_value(i, 11))
    PowerFactor.append(sheet.cell_value(i, 12))
    CableLength.append(sheet.cell_value(i, 5))
    GenAlpha.append(sheet.cell_value(i, 16))
    GenBeta.append(sheet.cell_value(i, 17))
    GenScalingFactor.append(sheet.cell_value(i, 18))
    GenPowerFactor.append(sheet.cell_value(i, 19))
    #Automatically places generators a length of 0.1m away from load nodes to maintain algebraic integrity
    if((Gens_PhaseA[a]+Gens_PhaseB[a]+Gens_PhaseC[a])>=1):
        GenCableLength.append(0.1)
    else:
```

```
    GenCableLength.append(0)
Distance=Distance+CableLength[a]+GenCableLength[a]
 #Determines the distance of each node away from the source. This is used to specify node locations on
    the output graph
TotalDistance.append(Distance)
#Get cable characteristics of cables used from input spreadsheet
K.append(sheet.cell_value(i, 3))
ResistanceT1.append(sheet.cell_value(i, 6))
ReactanceT1.append(sheet.cell_value(i, 7))
KGen.append(sheet.cell_value(i, 3))
ResistanceT1Gen.append(sheet.cell_value(i, 6))
ReactanceT1Gen.append(sheet.cell_value(i, 7))
#Determine the effect of temperature on the resistance and reactance values
ResistanceT2.append(ResistanceT1[a]*((sheet.cell_value(i, 4)+
float(Temp_Entered))/(sheet.cell_value(i, 4)+20)))
ReactanceT2.append(ReactanceT1[a]*((sheet.cell_value(i, 4)+
float(Temp_Entered))/(sheet.cell_value(i, 4)+20)))
#Determines the resistance and reactance by multiplying the cable length by the respective value per km
NodeResistance.append(float(ResistanceT2[a])*float(CableLength[a])/1000)
NodeReactance.append(float(ReactanceT2[a])*float(CableLength[a])/1000)
if(a==0):
    TotalResistance=TotalResistance+NodeResistance[a]
    TotalNeutralResistance=NodeResistance[a]*K[a]
    TotalReactance=TotalReactance+NodeReactance[a]
    TotalNeutralReactance=NodeReactance[a]*K[a]
else:
    TotalResistance=TotalResistance+NodeResistance[a]+NodeResistanceGen[a-1]
    TotalReactance=TotalReactance+NodeReactance[a]+NodeReactanceGen[a-1]
    TotalNeutralResistance=NodeResistance[a]*K[a]+TotalNeutralResistanceGen
    TotalNeutralReactance=NodeReactance[a]*K[a]+TotalNeutralReactanceGen
#Take the effect of temperature into account for generators
ResistanceT2Gen.append(ResistanceT1Gen[a]*((sheet.cell_value(i, 4)+
float(Temp_Entered))/(sheet.cell_value(i, 4)+20)))
ReactanceT2Gen.append(ReactanceT1Gen[a]*((sheet.cell_value(i, 4)+
float(Temp_Entered))/(sheet.cell_value(i, 4)+20)))
#Determines the resistance and reactance for cables to generators by multiplying the cable length by the
respective value per km
NodeResistanceGen.append(float(ResistanceT2Gen[a])*float(GenCableLength[a])/1000)
NodeReactanceGen.append(float(ReactanceT2Gen[a])*float(GenCableLength[a])/1000)
TotalResistanceGen=TotalResistance+NodeResistanceGen[a]
TotalReactanceGen=TotalReactance+NodeReactanceGen[a]
TotalNeutralReactanceGen=TotalNeutralReactance+NodeReactanceGen[a]*KGen[a]
TotalNeutralResistanceGen=TotalNeutralResistance+NodeResistanceGen[a]*KGen[a]
#Determine the value of k, H, G, Theta and scaling factor at each load and generator node
Kr_Effective.append(TotalNeutralResistance/TotalResistance)
Kx_Effective.append(TotalNeutralReactance/TotalReactance)
H.append(Alpha[a]*(Alpha[a]+1)/((Alpha[a]+Beta[a])*(Alpha[a]+Beta[a]+1)))
GSquared.append(Alpha[a]**2/(Alpha[a]+Beta[a])**2)
Theta.append(acos(PowerFactor[a]))
Kr_EffectiveGen.append(TotalNeutralResistanceGen/TotalResistanceGen)
Kx_EffectiveGen.append(TotalNeutralReactanceGen/TotalReactanceGen)
HGen.append(GenAlpha[a]*(GenAlpha[a]+1)/((GenAlpha[a]+GenBeta[a])*(GenAlpha[a]+GenBeta[a]+1)))
GSquaredGen.append(GenAlpha[a]**2/(GenAlpha[a]+GenBeta[a])**2)
ThetaGen.append(acos(GenPowerFactor[a]))
```

```
ScalingFactorNew.append(ScalingFactor[a])
GenScalingFactorNew.append(GenScalingFactor[a])
#Determines the percentage risk from the confidence level percentage entered
D_Risk=100-int(Confidence_Level_Entered)
#Determine the remaining base values
IBase=(float(Sbase_Entered)*10**6)/(float(Vbase_Entered)*10**3)
ZBase=((float(Vbase_Entered)*10**3)**2)/(float(Sbase_Entered)*10**6)
RpBase=TotalResistance/ZBase
XpBase=TotalReactance/ZBase
RnBase=TotalNeutralResistance/ZBase
XnBase=TotalNeutralReactance/ZBase
CbBase.append(ScalingFactorNew[a]/IBase)
RpBaseGen=TotalResistanceGen/ZBase
XpBaseGen=TotalReactanceGen/ZBase
RnBaseGen=TotalNeutralResistanceGen/ZBase
XnBaseGen=TotalNeutralReactanceGen/ZBase
CbBaseGen.append(GenScalingFactorNew[a]/IBase)
Vs_pu=float(Vs_Entered)/float(Vbase_Entered)
#Current calculations for red phase
IMean_Red.append((Loads_PhaseA[a]*sqrt(GSquared[a])*CbBase[a])/PowerFactor[a])
IMean_Red_Gen.append((Gens_PhaseA[a]*sqrt(GSquaredGen[a])*CbBaseGen[a])/GenPowerFactor[a])
IMeanSquared_Red.append(((abs(Loads_PhaseA[a])*H[a]*CbBase[a]**2+(abs(Loads_PhaseA[a])*
(abs(Loads_PhaseA[a])-1)*GSquared[a]*CbBase[a]**2))/PowerFactor[a]**2))
IMeanSquared_Red_Gen.append(((abs(Gens_PhaseA[a])*HGen[a]*CbBaseGen[a]**2+(abs(Gens_PhaseA[a]
)*(abs(Gens_PhaseA[a])-1)*GSquaredGen[a]*CbBaseGen[a]**2))/GenPowerFactor[a]**2))
if(a==0):
   Sum_IMean_Red=0
else:
   Sum_IMean_Red=Sum_IMean_Red_Gen+IMean_Red_Gen[a-1]
Sum_IMean_Red_Gen=Sum_IMean_Red+IMean_Red[a]
#Current calculations for white phase
IMean_White.append((Loads_PhaseB[a]*sqrt(GSquared[a])*CbBase[a])/PowerFactor[a])
IMean_White_Gen.append((Gens_PhaseB[a]*sqrt(GSquaredGen[a])*CbBaseGen[a])/GenPowerFactor[a])
IMeanSquared_White.append(((abs(Loads_PhaseB[a])*H[a]*CbBase[a]**2+(abs(Loads_PhaseB[a])*(abs(L
oads_PhaseB[a])-1)*GSquared[a]*CbBase[a]**2))/PowerFactor[a]**2))
IMeanSquared_White_Gen.append(((abs(Gens_PhaseB[a])*HGen[a]*CbBaseGen[a]**2+(abs(Gens_PhaseB[
a])*(abs(Gens_PhaseB[a])-1)*GSquaredGen[a]*CbBaseGen[a]**2))/GenPowerFactor[a]**2))
if(a==0):
   Sum_IMean_White=0
else:
   Sum_IMean_White=Sum_IMean_White_Gen+IMean_White_Gen[a-1]
Sum_IMean_White_Gen=Sum_IMean_White+IMean_White[a]
#Current calculations for blue phase
IMean_Blue.append((Loads_PhaseC[a]*sqrt(GSquared[a])*CbBase[a])/PowerFactor[a])
IMean_Blue_Gen.append((Gens_PhaseC[a]*sqrt(GSquaredGen[a])*CbBaseGen[a])/GenPowerFactor[a])
IMeanSquared_Blue.append(((abs(Loads_PhaseC[a])*H[a]*CbBase[a]**2+(abs(Loads_PhaseC[a])*(abs(Lo
ads_PhaseC[a])-1)*GSquared[a]*CbBase[a]**2))/PowerFactor[a]**2))
IMeanSquared_Blue_Gen.append(((abs(Gens_PhaseC[a])*HGen[a]*CbBaseGen[a]**2+(abs(Gens_PhaseC[a
])*(abs(Gens_PhaseC[a])-1)*GSquaredGen[a]*CbBaseGen[a]**2))/GenPowerFactor[a]**2))
if(a==0):
   Sum_IMean_Blue=0
else:
   Sum_IMean_Blue=Sum_IMean_Blue_Gen+IMean_Blue_Gen[a-1]
Sum_IMean_Blue_Gen=Sum_IMean_Blue+IMean_Blue[a]
```

```
        #If 3-phase, 4 wire system is selected, perform the associated calculations
        if(SystemConfig_Entered=='3-phase, 4 wire'):
            #Calculate current moments for red phase
            FirstMoment_IRed=FirstMoment_IRed_Gen+IMean_Red[a]
            FirstMoment_IRed_Gen=FirstMoment_IRed+IMean_Red_Gen[a]
            SecondMoment_IRed=SecondMoment_IRed_Gen+IMeanSquared_Red[a]+Sum_IMean_Red*IMean_Red[a]
            SecondMoment_IRed_Gen=SecondMoment_IRed+IMeanSquared_Red_Gen[a]+Sum_IMean_Red_Gen*
            IMean_Red_Gen[a]
            #Calculate current moments for white phase
            FirstMoment_IWhite=FirstMoment_IWhite_Gen+IMean_White[a]
            FirstMoment_IWhite_Gen=FirstMoment_IWhite+IMean_White_Gen[a]
            SecondMoment_IWhite=SecondMoment_IWhite_Gen+IMeanSquared_White[a]+Sum_IMean_White
            *IMean_White[a]

SecondMoment_IWhite_Gen=SecondMoment_IWhite+IMeanSquared_White_Gen[a]+Sum_IMean_White_Gen*IMe
an_White_Gen[a]
            # print('FirstMoment_IWhite= ',FirstMoment_IWhite)
            # print('FirstMoment_IWhite_Gen= ',FirstMoment_IWhite_Gen)
            # print('SecondMoment_IWhite= ',SecondMoment_IWhite)
            # print('SecondMoment_IWhite_Gen= ',SecondMoment_IWhite_Gen)
            #Calculate current moments for blue phase
            FirstMoment_IBlue=FirstMoment_IBlue_Gen+IMean_Blue[a]
            FirstMoment_IBlue_Gen=FirstMoment_IBlue+IMean_Blue_Gen[a]

SecondMoment_IBlue=SecondMoment_IBlue_Gen+IMeanSquared_Blue[a]+Sum_IMean_Blue*IMean_Blue[a]

SecondMoment_IBlue_Gen=SecondMoment_IBlue+IMeanSquared_Blue_Gen[a]+Sum_IMean_Blue_Gen*IMean_Bl
ue_Gen[a]
            # print('FirstMoment_IBlue= ',FirstMoment_IBlue)
            # print('FirstMoment_IBlue_Gen= ',FirstMoment_IBlue_Gen)
            # print('SecondMoment_IBlue= ',SecondMoment_IBlue)
            # print('SecondMoment_IBlue_Gen= ',SecondMoment_IBlue_Gen)
            #Calculates constants for red phase
            print('a =',a)
            for x in range(a,len(AlphaLumped)):
                Loads_PhaseALumped.append(Loads_PhaseA[x])
                Loads_PhaseBLumped.append(Loads_PhaseB[x])
                Loads_PhaseCLumped.append(Loads_PhaseC[x])
                Gens_PhaseALumped.append(Gens_PhaseA[x])
                Gens_PhaseBLumped.append(Gens_PhaseB[x])
                Gens_PhaseCLumped.append(Gens_PhaseC[x])
                CbBaseLumped.append(ScalingFactorLumped[x]/IBase)
                CbBaseGenLumped.append(GenScalingFactorLumped[x]/IBase)
                HLumped.append(AlphaLumped[x]*(AlphaLumped[x]+1)/((AlphaLumped[x]+BetaLumped[x])*
                (AlphaLumped[x]+BetaLumped[x]+1)))
                GSquaredLumped.append(AlphaLumped[x]**2/(AlphaLumped[x]+BetaLumped[x])**2)
                HGenLumped.append(GenAlphaLumped[x]*(GenAlphaLumped[x]+1)/((GenAlphaLumped[x]+GenBet
                aLumped[x])*(GenAlphaLumped[x]+GenBetaLumped[x]+1)))
                GSquaredGenLumped.append(GenAlphaLumped[x]**2/(GenAlphaLumped[x]+
                GenBetaLumped[x])**2)
                k1.append(Kr_Effective[a]*RpBase*cos(ThetaLumped[x])+Kx_Effective[a]*XpBase*
                sin(ThetaLumped[x]))
                k2.append(Kx_Effective[a]*XpBase*cos(ThetaLumped[x])-Kr_Effective[a]*
                RpBase*sin(ThetaLumped[x]))
```

```
k3.append((1+Kr_Effective[a])*RpBase*cos(ThetaLumped[x])+(1+Kx_Effective[a])*XpBase*
sin(ThetaLumped[x]))
 k4.append((1+Kx_Effective[a])*XpBase*cos(ThetaLumped[x])-(1+Kr_Effective[a])*
RpBase*sin(ThetaLumped[x]))
k1Gen.append(Kr_EffectiveGen[a]*RpBaseGen*cos(ThetaGenLumped[x])+Kx_EffectiveGen[a]*
XpBaseGen*sin(ThetaGenLumped[x]))
k2Gen.append(Kx_EffectiveGen[a]*XpBaseGen*cos(ThetaGenLumped[x])-Kr_EffectiveGen[a]*
RpBaseGen*sin(ThetaGenLumped[x]))
k3Gen.append((1+Kr_EffectiveGen[a])*RpBaseGen*cos(ThetaGenLumped[x])+(1+Kx_EffectiveGen[a])*
XpBaseGen*sin(ThetaGenLumped[x]))
k4Gen.append((1+Kx_EffectiveGen[a])*XpBaseGen*cos(ThetaGenLumped[x])-(1+Kr_EffectiveGen[a])*
RpBaseGen*sin(ThetaGenLumped[x]))
  if (a==len(AlphaLumped)-1):
    for x in range(0,len(k1)):
      c1.append(0.5*k1[x]*k2[x]*sqrt(3)+0.75*k2[x]**2+0.25*k1[x]**2)
      c2.append(-0.5*k1[x]*k2[x]*sqrt(3)+0.75*k2[x]**2+0.25*k1[x]**2)
      c3_Red.append(k3[x]**2*abs(Loads_PhaseALumped[x])*(abs(Loads_PhaseALumped[x])-
      1)+c1[x]*abs(Loads_PhaseBLumped[x])* (abs(Loads_PhaseBLumped[x])-1)+c2[x]*
      abs(Loads_PhaseCLumped[x])*(abs(Loads_PhaseCLumped[x])-1))
      c4_Red.append(abs(Loads_PhaseALumped[x])*abs(Loads_PhaseBLumped[x])*(-
      1*k3[x]*k2[x]*sqrt(3)-
      k1[x]*k3[x])+abs(Loads_PhaseALumped[x])*abs(Loads_PhaseCLumped[x])*(k3[x]*k2[x]*sqrt(3)-
      k1[x]*k3[x])+abs(Loads_PhaseBLumped[x])*abs(Loads_PhaseCLumped[x])*(-
      1.5*k2[x]**2+0.5*k1[x]**2))
      c1Gen.append(0.5*k1Gen[x]*k2Gen[x]*sqrt(3)+0.75*k2Gen[x]**2+0.25*k1Gen[x]**2)
      c2Gen.append(-0.5*k1Gen[x]*k2Gen[x]*sqrt(3)+0.75*k2Gen[x]**2+0.25*k1Gen[x]**2)
      c3_RedGen.append(k3Gen[x]**2*abs(Gens_PhaseALumped[x])*
       (abs(Gens_PhaseALumped[x])-1)+c1Gen[x]*abs(Gens_PhaseBLumped[x])*
       (abs(Gens_PhaseBLumped[x])-1)+c2Gen[x]*
       abs(Gens_PhaseCLumped[x])*(abs(Gens_PhaseCLumped[x])-1))
      c4_RedGen.append(abs(Gens_PhaseALumped[x])*abs(Gens_PhaseBLumped[x])*(-
       1*k3Gen[x]*k2Gen[x]*sqrt(3)-
       k1Gen[x]*k3Gen[x])+abs(Gens_PhaseALumped[x])*abs(Gens_PhaseCLumped[x])*(k3Gen[x]*k2G
       en[x]*sqrt(3)-
       k1Gen[x]*k3Gen[x])+abs(Gens_PhaseBLumped[x])*abs(Gens_PhaseCLumped[x])*(-
       1.5*k2Gen[x]**2+0.5*k1Gen[x]**2))
      d1.append(-0.5*k1[x]*k2[x]*sqrt(3)+0.75*k1[x]**2+0.25*k2[x]**2)
      d2.append(0.5*k1[x]*k2[x]*sqrt(3)+0.75*k1[x]**2+0.25*k2[x]**2)
      d3_Red.append(k4[x]**2*abs(Loads_PhaseALumped[x])*(abs(Loads_PhaseALumped[x])-
       1)+d1[x]*abs(Loads_PhaseBLumped[x])*(abs(Loads_PhaseBLumped[x])-1)+
       d2[x]*abs(Loads_PhaseCLumped[x])*(abs(Loads_PhaseCLumped[x])-1))
       d4_Red.append(abs(Loads_PhaseALumped[x])*abs(Loads_PhaseBLumped[x])*(k4[x]*k1[x]*
       sqrt(3)-k2[x]*k4[x])+abs(Loads_PhaseALumped[x])*abs(Loads_PhaseCLumped[x])*(-
       k4[x]*k1[x]*sqrt(3)-k2[x]*k4[x])+abs(Loads_PhaseBLumped[x])*
       abs(Loads_PhaseCLumped[x])*(-1.5*k1[x]**2+0.5*k2[x]**2))
      d1Gen.append(-0.5*k1Gen[x]*k2Gen[x]*sqrt(3)+0.75*k1Gen[x]**2+0.25*k2Gen[x]**2)
      d2Gen.append(0.5*k1Gen[x]*k2Gen[x]*sqrt(3)+0.75*k1Gen[x]**2+0.25*k2Gen[x]**2)
      d3_RedGen.append(k4Gen[x]**2*abs(Gens_PhaseALumped[x])*
       (abs(Gens_PhaseALumped[x])-1)+d1Gen[x]*abs(Gens_PhaseBLumped[x])*
       (abs(Gens_PhaseBLumped[x])-1)+d2Gen[x]*
       abs(Gens_PhaseCLumped[x])*(abs(Gens_PhaseCLumped[x])-1))
       d4_RedGen.append(abs(Gens_PhaseALumped[x])*abs(Gens_PhaseBLumped[x])*(k4Gen[x]*k1Ge
       n[x]*sqrt(3)-k2Gen[x]*k4Gen[x])+abs(Gens_PhaseALumped[x])*abs(Gens_PhaseCLumped[x])*(-
```

```
k4Gen[x]*k1Gen[x]*sqrt(3)-k2Gen[x]*k4Gen[x])+abs(Gens_PhaseBLumped[x])*
abs(Gens_PhaseCLumped[x])*(-1.5*k1Gen[x]**2+0.5*k2Gen[x]**2))
C1_Red.append(0.5*(Loads_PhaseBLumped[x]+Loads_PhaseCLumped[x]))
C2_Red.append(0.5*sqrt(3)*(Loads_PhaseBLumped[x]-Loads_PhaseCLumped[x]))
C3_Red.append(k3[x]**2*abs(Loads_PhaseALumped[x])+c1[x]*
abs(Loads_PhaseBLumped[x])+c2[x]*abs(Loads_PhaseCLumped[x]))
C4_Red.append(c3_Red[x]+c4_Red[x])
C5_Red.append(k4[x]**2*abs(Loads_PhaseALumped[x])+d1[x]*abs(Loads_PhaseBLumped[x])+
d2[x]*abs(Loads_PhaseCLumped[x]))
C6_Red.append(d3_Red[x]+d4_Red[x])
C1_RedGen.append(0.5*(Gens_PhaseBLumped[x]+Gens_PhaseCLumped[x]))
C2_RedGen.append(0.5*sqrt(3)*(Gens_PhaseBLumped[x]-Gens_PhaseCLumped[x]))
C3_RedGen.append(k3Gen[x]**2*abs(Gens_PhaseALumped[x])+c1Gen[x]*
abs(Gens_PhaseBLumped[x])+c2Gen[x]*abs(Gens_PhaseCLumped[x]))
C4_RedGen.append(c3_RedGen[x]+c4_RedGen[x])
C5_RedGen.append(k4Gen[x]**2*abs(Gens_PhaseALumped[x])+d1Gen[x]*abs(Gens_PhaseBLum
ped[x])+d2Gen[x]*abs(Gens_PhaseCLumped[x]))
C6_RedGen.append(d3_RedGen[x]+d4_RedGen[x])
#Calculates constants for white phase
c3_White.append(k3[x]**2*abs(Loads_PhaseBLumped[x])*(abs(Loads_PhaseBLumped[x])-
1)+c1[x]*abs(Loads_PhaseCLumped[x])*(abs(Loads_PhaseCLumped[x])-
1)+c2[x]*abs(Loads_PhaseALumped[x])*(abs(Loads_PhaseALumped[x])-1))
c4_White.append(abs(Loads_PhaseBLumped[x])*abs(Loads_PhaseCLumped[x])*(-
1*k3[x]*k2[x]*sqrt(3)-
k1[x]*k3[x])+abs(Loads_PhaseBLumped[x])*abs(Loads_PhaseALumped[x])*(k3[x]*k2[x]*sqrt(3)
-k1[x]*k3[x])+abs(Loads_PhaseCLumped[x])*abs(Loads_PhaseALumped[x])*(-
1.5*k2[x]**2+0.5*k1[x]**2))
c3_WhiteGen.append(k3Gen[x]**2*abs(Gens_PhaseBLumped[x])*(abs(Gens_PhaseBLumped[x])-
1)+c1Gen[x]*abs(Gens_PhaseCLumped[x])*(abs(Gens_PhaseCLumped[x])-
1)+c2Gen[x]*abs(Gens_PhaseALumped[x])*(abs(Gens_PhaseALumped[x])-1))
c4_WhiteGen.append(abs(Gens_PhaseBLumped[x])*abs(Gens_PhaseCLumped[x])*(-
1*k3Gen[x]*k2Gen[x]*sqrt(3)-
k1Gen[x]*k3Gen[x])+abs(Gens_PhaseBLumped[x])*abs(Gens_PhaseALumped[x])*(k3Gen[x]*k2G
en[x]*sqrt(3)-k1Gen[x]*k3Gen[x])+abs(Gens_PhaseCLumped[x])*
abs(Gens_PhaseALumped[x])*(-1.5*k2Gen[x]**2+0.5*k1Gen[x]**2))
d3_White.append(k4[x]**2*abs(Loads_PhaseBLumped[x])*(abs(Loads_PhaseBLumped[x])-1)+
d1[x]*abs(Loads_PhaseCLumped[x])*(abs(Loads_PhaseCLumped[x])-1)+
d2[x]*abs(Loads_PhaseALumped[x])*(abs(Loads_PhaseALumped[x])-1))
d4_White.append(abs(Loads_PhaseBLumped[x])*abs(Loads_PhaseCLumped[x])*(k4[x]*k1[x]*
sqrt(3)-k2[x]*k4[x])+abs(Loads_PhaseBLumped[x])*abs(Loads_PhaseALumped[x])*(-
k4[x]*k1[x]*sqrt(3)-k2[x]*k4[x])+
abs(Loads_PhaseCLumped[x])*abs(Loads_PhaseALumped[x])*(-1.5*k1[x]**2+0.5*k2[x]**2))
d3_WhiteGen.append(k4Gen[x]**2*abs(Gens_PhaseBLumped[x])*(abs(Gens_PhaseBLumped[x])-
1)+d1Gen[x]*abs(Gens_PhaseCLumped[x])*(abs(Gens_PhaseCLumped[x])-
1)+d2Gen[x]*abs(Gens_PhaseALumped[x])*(abs(Gens_PhaseALumped[x])-1))
d4_WhiteGen.append(abs(Gens_PhaseBLumped[x])*abs(Gens_PhaseCLumped[x])*(k4Gen[x]*k1
Gen[x]*sqrt(3)-k2Gen[x]*k4Gen[x])+abs(Gens_PhaseBLumped[x])*
abs(Gens_PhaseALumped[x])*(-k4Gen[x]*k1Gen[x]*sqrt(3)-k2Gen[x]*k4Gen[x])+
abs(Gens_PhaseCLumped[x])*abs(Gens_PhaseALumped[x])*(-1.5*k1Gen[x]**2+0.5*
k2Gen[x]**2))
C1_White.append(0.5*(Loads_PhaseCLumped[x]+Loads_PhaseALumped[x]))
C2_White.append(0.5*sqrt(3)*(Loads_PhaseCLumped[x]-Loads_PhaseALumped[x]))
C3_White.append(k3[x]**2*abs(Loads_PhaseBLumped[x])+c1[x]*abs(Loads_PhaseCLumped[x])+
c2[x]*abs(Loads_PhaseALumped[x]))
```

```
C4_White.append(c3_White[x]+c4_White[x])
C5_White.append(k4[x]**2*abs(Loads_PhaseBLumped[x])+d1[x]*abs(Loads_PhaseCLumped[x])+
d2[x]*abs(Loads_PhaseALumped[x]))
C6_White.append(d3_White[x]+d4_White[x])
C1_WhiteGen.append(0.5*(Gens_PhaseCLumped[x]+Gens_PhaseALumped[x]))
C2_WhiteGen.append(0.5*sqrt(3)*(Gens_PhaseCLumped[x]-Gens_PhaseALumped[x]))
C3_WhiteGen.append(k3Gen[x]**2*abs(Gens_PhaseBLumped[x])+c1Gen[x]*
abs(Gens_PhaseCLumped[x])+c2Gen[x]*abs(Gens_PhaseALumped[x]))
C4_WhiteGen.append(c3_WhiteGen[x]+c4_WhiteGen[x])
C5_WhiteGen.append(k4Gen[x]**2*abs(Gens_PhaseBLumped[x])+d1Gen[x]*abs(Gens_PhaseCLum
ped[x])+d2Gen[x]*abs(Gens_PhaseALumped[x]))
C6_WhiteGen.append(d3_WhiteGen[x]+d4_WhiteGen[x])
#Calculates constants for blue phase
c3_Blue.append(k3[x]**2*abs(Loads_PhaseCLumped[x])*(abs(Loads_PhaseCLumped[x])-1)+
c1[x]*abs(Loads_PhaseALumped[x])*(abs(Loads_PhaseALumped[x])-1)+c2[x]*
abs(Loads_PhaseBLumped[x])* (abs(Loads_PhaseBLumped[x])-1))
c4_Blue.append(abs(Loads_PhaseCLumped[x])*abs(Loads_PhaseALumped[x])*(-
1*k3[x]*k2[x]*sqrt(3)-k1[x]*k3[x])+abs(Loads_PhaseCLumped[x])*
abs(Loads_PhaseBLumped[x])*(k3[x]*k2[x]*sqrt(3)-
k1[x]*k3[x])+abs(Loads_PhaseALumped[x])*abs(Loads_PhaseBLumped[x])*(-
1.5*k2[x]**2+0.5*k1[x]**2))
c3_BlueGen.append(k3Gen[x]**2*abs(Gens_PhaseCLumped[x])*
(abs(Gens_PhaseCLumped[x])-1)+c1Gen[x]*abs(Gens_PhaseALumped[x])*
(abs(Gens_PhaseALumped[x])-1)+c2Gen[x]*abs(Gens_PhaseBLumped[x])*
(abs(Gens_PhaseBLumped[x])-1))
c4_BlueGen.append(abs(Gens_PhaseCLumped[x])*abs(Gens_PhaseALumped[x])*(-
1*k3Gen[x]*k2Gen[x]*sqrt(3)-
k1Gen[x]*k3Gen[x])+abs(Gens_PhaseCLumped[x])*abs(Gens_PhaseBLumped[x])*(k3Gen[x]*
k2Gen[x]*sqrt(3)-
k1Gen[x]*k3Gen[x])+abs(Gens_PhaseALumped[x])*abs(Gens_PhaseBLumped[x])*(-
1.5*k2Gen[x]**2+0.5*k1Gen[x]**2))
d3_Blue.append(k4[x]**2*abs(Loads_PhaseCLumped[x])*(abs(Loads_PhaseCLumped[x])-
1)+d1[x]*abs(Loads_PhaseALumped[x])*(abs(Loads_PhaseALumped[x])-
1)+d2[x]*abs(Loads_PhaseBLumped[x])*(abs(Loads_PhaseBLumped[x])-1))
d4_Blue.append(abs(Loads_PhaseCLumped[x])*abs(Loads_PhaseALumped[x])*
(k4[x]*k1[x]*sqrt(3)-k2[x]*k4[x])+abs(Loads_PhaseCLumped[x])*
abs(Loads_PhaseBLumped[x])*(-k4[x]*k1[x]*sqrt(3)-k2[x]*k4[x])+
abs(Loads_PhaseALumped[x])*abs(Loads_PhaseBLumped[x])*(-1.5*k1[x]**2+0.5*k2[x]**2))
d3_BlueGen.append(k4Gen[x]**2*abs(Gens_PhaseCLumped[x])*
(abs(Gens_PhaseCLumped[x])-1)+d1Gen[x]*abs(Gens_PhaseALumped[x])*
(abs(Gens_PhaseALumped[x])-1)+d2Gen[x]*abs(Gens_PhaseBLumped[x])*
(abs(Gens_PhaseBLumped[x])-1))
d4_BlueGen.append(abs(Gens_PhaseCLumped[x])*abs(Gens_PhaseALumped[x])*
(k4Gen[x]*k1Gen[x]*sqrt(3)-k2Gen[x]*k4Gen[x])+abs(Gens_PhaseCLumped[x])*
abs(Gens_PhaseBLumped[x])*(-k4Gen[x]*k1Gen[x]*sqrt(3)-k2Gen[x]*k4Gen[x])+
abs(Gens_PhaseALumped[x])*abs(Gens_PhaseBLumped[x])*(-1.5*k1Gen[x]**2+
0.5*k2Gen[x]**2))
C1_Blue.append(0.5*(Loads_PhaseALumped[x]+Loads_PhaseBLumped[x]))
C2_Blue.append(0.5*sqrt(3)*(Loads_PhaseALumped[x]-Loads_PhaseBLumped[x]))
C3_Blue.append(k3[x]**2*abs(Loads_PhaseCLumped[x])+c1[x]*abs(Loads_PhaseALumped[x])+
c2[x]*abs(Loads_PhaseBLumped[x]))
C4_Blue.append(c3_Blue[x]+c4_Blue[x])
C5_Blue.append(k4[x]**2*abs(Loads_PhaseCLumped[x])+d1[x]*abs(Loads_PhaseALumped[x])+
d2[x]*abs(Loads_PhaseBLumped[x]))
```

```
C6_Blue.append(d3_Blue[x]+d4_Blue[x])
C1_BlueGen.append(0.5*(Gens_PhaseALumped[x]+Gens_PhaseBLumped[x]))
C2_BlueGen.append(0.5*sqrt(3)*(Gens_PhaseALumped[x]-Gens_PhaseBLumped[x]))
 C3_BlueGen.append(k3Gen[x]**2*abs(Gens_PhaseCLumped[x])+c1Gen[x]*
 abs(Gens_PhaseALumped[x])+c2Gen[x]*abs(Gens_PhaseBLumped[x]))
C4_BlueGen.append(c3_BlueGen[x]+c4_BlueGen[x])
 C5_BlueGen.append(k4Gen[x]**2*abs(Gens_PhaseCLumped[x])+d1Gen[x]*
 abs(Gens_PhaseALumped[x])+d2Gen[x]*abs(Gens_PhaseBLumped[x]))
C6_BlueGen.append(d3_BlueGen[x]+d4_BlueGen[x])
#Calculate voltage-drop moments for red phase
FirstMoment_Vr_Red.append((k3[x]*Loads_PhaseALumped[x]-k1[x]*C1_Red[x]-k2[x]*
 C2_Red[x])*CbBaseLumped[x]*sqrt(GSquaredLumped[x]))
FirstMoment_Vr_Gen_Red.append((k3Gen[x]*Gens_PhaseALumped[x]-k1Gen[x]*C1_RedGen[x]-
 k2Gen[x]*C2_RedGen[x])*CbBaseGenLumped[x]*sqrt(GSquaredGenLumped[x]))
 SecondMoment_Vr_Red.append(CbBaseLumped[x]**2*(C3_Red[x]*HLumped[x]+C4_Red[x]*
 GSquaredLumped[x]))
 SecondMoment_Vr_Gen_Red.append(CbBaseGenLumped[x]**2*(C3_RedGen[x]*HGenLumped[x]+
 C4_RedGen[x]*GSquaredGenLumped[x]))
FirstMoment_Vi_Red.append((k4[x]*Loads_PhaseALumped[x]-k2[x]*C1_Red[x]+k1[x]*
 C2_Red[x])*CbBaseLumped[x]*sqrt(GSquaredLumped[x]))
FirstMoment_Vi_Gen_Red.append((k4Gen[x]*Gens_PhaseALumped[x]-k2Gen[x]*C1_RedGen[x]+
 k1Gen[x]*C2_RedGen[x])*CbBaseGenLumped[x]*sqrt(GSquaredGenLumped[x]))
 SecondMoment_Vi_Red.append(CbBaseLumped[x]**2*(C5_Red[x]*HLumped[x]+C6_Red[x]*
 GSquaredLumped[x]))
 SecondMoment_Vi_Gen_Red.append(CbBaseGenLumped[x]**2*(C5_RedGen[x]*HGenLumped[x]+
 C6_RedGen[x]*GSquaredGenLumped[x]))
#Calculate voltage-drop moments for white phase
FirstMoment_Vr_White.append((k3[x]*Loads_PhaseBLumped[x]-k1[x]*C1_White[x]-k2[x]*
 C2_White[x])*CbBaseLumped[x]*sqrt(GSquaredLumped[x]))
FirstMoment_Vr_Gen_White.append((k3Gen[x]*Gens_PhaseALumped[x]-k1Gen[x]*
 C1_WhiteGen[x]-k2Gen[x]*C2_WhiteGen[x])*CbBaseGenLumped[x]*
 sqrt(GSquaredGenLumped[x]))
 SecondMoment_Vr_White.append(CbBaseLumped[x]**2*(C3_White[x]*HLumped[x]+C4_White[x]
*GSquaredLumped[x]))
 SecondMoment_Vr_Gen_White.append(CbBaseGenLumped[x]**2*(C3_WhiteGen[x]*
 HGenLumped[x]+C4_WhiteGen[x]*GSquaredGenLumped[x]))
FirstMoment_Vi_White.append((k4[x]*Loads_PhaseBLumped[x]-k2[x]*C1_White[x]+
 k1[x]*C2_White[x])*CbBaseLumped[x]*sqrt(GSquaredLumped[x]))
FirstMoment_Vi_Gen_White.append((k4Gen[x]*Gens_PhaseALumped[x]-
 k2Gen[x]*C1_WhiteGen[x]+k1Gen[x]*C2_WhiteGen[x])*CbBaseGenLumped[x]*
 sqrt(GSquaredGenLumped[x]))
 SecondMoment_Vi_White.append(CbBaseLumped[x]**2*(C5_White[x]*HLumped[x]+
 C6_White[x]*GSquaredLumped[x]))
 SecondMoment_Vi_Gen_White.append(CbBaseGenLumped[x]**2*(C5_WhiteGen[x]*
 HGenLumped[x]+C6_WhiteGen[x]*GSquaredGenLumped[x]))
#Calculate voltage-drop moments for white phase
FirstMoment_Vr_Blue.append((k3[x]*Loads_PhaseCLumped[x]-k1[x]*C1_Blue[x]-
 k2[x]*C2_Blue[x])*CbBaseLumped[x]*sqrt(GSquaredLumped[x]))
FirstMoment_Vr_Gen_Blue.append((k3Gen[x]*Gens_PhaseALumped[x]-k1Gen[x]*C1_BlueGen[x]-
 k2Gen[x]*C2_BlueGen[x])*CbBaseGenLumped[x]*sqrt(GSquaredGenLumped[x]))
 SecondMoment_Vr_Blue.append(CbBaseLumped[x]**2*(C3_Blue[x]*HLumped[x]+C4_Blue[x]*
 GSquaredLumped[x]))
 SecondMoment_Vr_Gen_Blue.append(CbBaseGenLumped[x]**2*(C3_BlueGen[x]*HGenLumped[x]
 +C4_BlueGen[x]*GSquaredGenLumped[x]))
```

```
    FirstMoment_Vi_Blue.append((k4[x]*Loads_PhaseCLumped[x]-k2[x]*C1_Blue[x]+
    k1[x]*C2_Blue[x])*CbBaseLumped[x]*sqrt(GSquaredLumped[x]))
    FirstMoment_Vi_Gen_Blue.append((k4Gen[x]*Gens_PhaseALumped[x]-
    k2Gen[x]*C1_BlueGen[x]+k1Gen[x]*C2_BlueGen[x])*CbBaseGenLumped[x]*
    sqrt(GSquaredGenLumped[x]))
    SecondMoment_Vi_Blue.append(CbBaseLumped[x]**2*(C5_Blue[x]*HLumped[x]+C6_Blue[x]*
    GSquaredLumped[x]))
    SecondMoment_Vi_Gen_Blue.append(CbBaseGenLumped[x]**2*(C5_BlueGen[x]*HGenLumped[x]
    +C6_BlueGen[x]*GSquaredGenLumped[x]))
#Copies value for manipulation
FirstMoment_Vr_RedCopy=copy.deepcopy(FirstMoment_Vr_Red)
FirstMoment_Vr_WhiteCopy=copy.deepcopy(FirstMoment_Vr_White)
FirstMoment_Vr_BlueCopy=copy.deepcopy(FirstMoment_Vr_Blue)
FirstMoment_Vi_RedCopy=copy.deepcopy(FirstMoment_Vi_Red)
FirstMoment_Vi_WhiteCopy=copy.deepcopy(FirstMoment_Vi_White)
FirstMoment_Vi_BlueCopy=copy.deepcopy(FirstMoment_Vi_Blue)
FirstMoment_Vr_Gen_RedCopy=copy.deepcopy(FirstMoment_Vr_Gen_Red)
FirstMoment_Vi_Gen_RedCopy=copy.deepcopy(FirstMoment_Vi_Gen_Red)
FirstMoment_Vr_Gen_WhiteCopy=copy.deepcopy(FirstMoment_Vr_Gen_White)
FirstMoment_Vi_Gen_WhiteCopy=copy.deepcopy(FirstMoment_Vi_Gen_White)
FirstMoment_Vr_Gen_BlueCopy=copy.deepcopy(FirstMoment_Vr_Gen_Blue)
FirstMoment_Vi_Gen_BlueCopy=copy.deepcopy(FirstMoment_Vi_Gen_Blue)
SecondMoment_Vr_RedCopy=copy.deepcopy(SecondMoment_Vr_Red)
SecondMoment_Vr_Gen_RedCopy=copy.deepcopy(SecondMoment_Vr_Gen_Red)
SecondMoment_Vi_RedCopy=copy.deepcopy(SecondMoment_Vi_Red)
SecondMoment_Vi_Gen_RedCopy=copy.deepcopy(SecondMoment_Vi_Gen_Red)
SecondMoment_Vr_WhiteCopy=copy.deepcopy(SecondMoment_Vr_White)
SecondMoment_Vr_Gen_WhiteCopy=copy.deepcopy(SecondMoment_Vr_Gen_White)
SecondMoment_Vi_WhiteCopy=copy.deepcopy(SecondMoment_Vi_White)
SecondMoment_Vi_Gen_WhiteCopy=copy.deepcopy(SecondMoment_Vi_Gen_White)
SecondMoment_Vr_BlueCopy=copy.deepcopy(SecondMoment_Vr_Blue)
SecondMoment_Vr_Gen_BlueCopy=copy.deepcopy(SecondMoment_Vr_Gen_Blue)
SecondMoment_Vi_BlueCopy=copy.deepcopy(SecondMoment_Vi_Blue)
SecondMoment_Vi_Gen_BlueCopy=copy.deepcopy(SecondMoment_Vi_Gen_Blue)
Laterals=int(len(Data)/3)
for i in range(len(Iteration)):
    #This array will pinpoint zero value positions in moment arrays
    p=[]
    if(i==0):
        StoreValue=copy.deepcopy(Data)
        for h in range(Laterals):
            N_matrix_r_Red=FirstMoment_Vr_RedCopy[:len(Iteration)]
            N_matrix_r_White=FirstMoment_Vr_WhiteCopy[:len(Iteration)]
            N_matrix_r_Blue=FirstMoment_Vr_BlueCopy[:len(Iteration)]
            N_matrix_i_Red=FirstMoment_Vi_RedCopy[:len(Iteration)]
            N_matrix_i_White=FirstMoment_Vi_WhiteCopy[:len(Iteration)]
            N_matrix_i_Blue=FirstMoment_Vi_BlueCopy[:len(Iteration)]
            N_matrix_r_RedGen=FirstMoment_Vr_Gen_RedCopy[:len(Iteration)]
            N_matrix_i_RedGen=FirstMoment_Vi_Gen_RedCopy[:len(Iteration)]
            N_matrix_r_WhiteGen=FirstMoment_Vr_Gen_WhiteCopy[:len(Iteration)]
            N_matrix_i_WhiteGen=FirstMoment_Vi_Gen_WhiteCopy[:len(Iteration)]
            N_matrix_r_BlueGen=FirstMoment_Vr_Gen_BlueCopy[:len(Iteration)]
            N_matrix_i_BlueGen=FirstMoment_Vi_Gen_BlueCopy[:len(Iteration)]
            N2_matrix_r_Red=SecondMoment_Vr_RedCopy[:len(Iteration)]
```

```
    N2_matrix_r_RedGen=SecondMoment_Vr_Gen_RedCopy[:len(Iteration)]
    N2_matrix_i_Red=SecondMoment_Vi_RedCopy[:len(Iteration)]
    N2_matrix_i_RedGen=SecondMoment_Vi_Gen_RedCopy[:len(Iteration)]
    N2_matrix_r_White=SecondMoment_Vr_WhiteCopy[:len(Iteration)]
    N2_matrix_r_WhiteGen=SecondMoment_Vr_Gen_WhiteCopy[:len(Iteration)]
    N2_matrix_i_White=SecondMoment_Vi_WhiteCopy[:len(Iteration)]
    N2_matrix_i_WhiteGen=SecondMoment_Vi_Gen_WhiteCopy[:len(Iteration)]
    N2_matrix_r_Blue=SecondMoment_Vr_BlueCopy[:len(Iteration)]
    N2_matrix_r_BlueGen=SecondMoment_Vr_Gen_BlueCopy[:len(Iteration)]
    N2_matrix_i_Blue=SecondMoment_Vi_BlueCopy[:len(Iteration)]
    N2_matrix_i_BlueGen=SecondMoment_Vi_Gen_BlueCopy[:len(Iteration)]
    for m in range(len(N_matrix_r_Red)):
        if((StoreValue[5]-2)<=m<=(StoreValue[6]-2)):
            p.append(m)
        elif((m==StoreValue[0]-2) and h==0):
            p.append(m)
        else:
            pass
    del(StoreValue[:3])
#Sets required node values to 0 for each iteration
for q in range(1,len(p)):
    if((p[q]-p[q-1])!=1):
        for d in range(p[q-1]-1,p[q]-2):
            N_matrix_r_Red[d]=0
            N_matrix_r_White[d]=0
            N_matrix_r_Blue[d]=0
            N_matrix_i_Red[d]=0
            N_matrix_i_White[d]=0
            N_matrix_i_Blue[d]=0
            N_matrix_r_RedGen[d]=0
            N_matrix_i_RedGen[d]=0
            N_matrix_r_WhiteGen[d]=0
            N_matrix_i_WhiteGen[d]=0
            N_matrix_r_BlueGen[d]=0
            N_matrix_i_BlueGen[d]=0
            N2_matrix_r_Red[d]=0
            N2_matrix_r_White[d]=0
            N2_matrix_r_Blue[d]=0
            N2_matrix_i_Red[d]=0
            N2_matrix_i_White[d]=0
            N2_matrix_i_Blue[d]=0
            N2_matrix_r_RedGen[d]=0
            N2_matrix_i_RedGen[d]=0
            N2_matrix_r_WhiteGen[d]=0
            N2_matrix_i_WhiteGen[d]=0
            N2_matrix_r_BlueGen[d]=0
            N2_matrix_i_BlueGen[d]=0
    elif(p[len(p)-1]!=(len(N_matrix_r_Red)-1)):
        for d in range(p[len(p)-1]+1,len(N_matrix_r_Red)):
            N_matrix_r_Red[d]=0
            N_matrix_r_White[d]=0
            N_matrix_r_Blue[d]=0
            N_matrix_i_Red[d]=0
            N_matrix_i_White[d]=0
```

```
            N_matrix_i_Blue[d]=0
            N_matrix_r_RedGen[d]=0
            N_matrix_i_RedGen[d]=0
            N_matrix_r_WhiteGen[d]=0
            N_matrix_i_WhiteGen[d]=0
            N_matrix_r_BlueGen[d]=0
            N_matrix_i_BlueGen[d]=0
            N2_matrix_r_Red[d]=0
            N2_matrix_r_White[d]=0
            N2_matrix_r_Blue[d]=0
            N2_matrix_i_Red[d]=0
            N2_matrix_i_White[d]=0
            N2_matrix_i_Blue[d]=0
            N2_matrix_r_RedGen[d]=0
            N2_matrix_i_RedGen[d]=0
            N2_matrix_r_WhiteGen[d]=0
            N2_matrix_i_WhiteGen[d]=0
            N2_matrix_r_BlueGen[d]=0
            N2_matrix_i_BlueGen[d]=0
#Store values in lateral arrays if the node is one of the lateral connection points
if(len(Data)==len(Lats) and i==(Data[0]-2)):
    N_matrix_r_RedLaterals.append(N_matrix_r_Red)
    N_matrix_r_WhiteLaterals.append(N_matrix_r_White)
    N_matrix_r_BlueLaterals.append(N_matrix_r_Blue)
    N_matrix_i_RedLaterals.append(N_matrix_i_Red)
    N_matrix_i_WhiteLaterals.append(N_matrix_i_White)
    N_matrix_i_BlueLaterals.append(N_matrix_i_Blue)
    N_matrix_r_RedGenLaterals.append(N_matrix_r_RedGen)
    N_matrix_i_RedGenLaterals.append(N_matrix_i_RedGen)
    N_matrix_r_WhiteGenLaterals.append(N_matrix_r_WhiteGen)
    N_matrix_i_WhiteGenLaterals.append(N_matrix_i_WhiteGen)
    N_matrix_r_BlueGenLaterals.append(N_matrix_r_BlueGen)
    N_matrix_i_BlueGenLaterals.append(N_matrix_i_BlueGen)
    N2_matrix_r_RedLaterals.append(N2_matrix_r_Red)
    N2_matrix_r_WhiteLaterals.append(N2_matrix_r_White)
    N2_matrix_r_BlueLaterals.append(N2_matrix_r_Blue)
    N2_matrix_i_RedLaterals.append(N2_matrix_i_Red)
    N2_matrix_i_WhiteLaterals.append(N2_matrix_i_White)
    N2_matrix_i_BlueLaterals.append(N2_matrix_i_Blue)
    N2_matrix_r_RedGenLaterals.append(N2_matrix_r_RedGen)
    N2_matrix_i_RedGenLaterals.append(N2_matrix_i_RedGen)
    N2_matrix_r_WhiteGenLaterals.append(N2_matrix_r_WhiteGen)
    N2_matrix_i_WhiteGenLaterals.append(N2_matrix_i_WhiteGen)
    N2_matrix_r_BlueGenLaterals.append(N2_matrix_r_BlueGen)
    N2_matrix_i_BlueGenLaterals.append(N2_matrix_i_BlueGen)
#Ensures laterals receive the correct values
if(len(Data)!=len(Lats) and i==(Data[0]-2)):
    N_matrix_r_Red.clear()
    N_matrix_r_White.clear()
    N_matrix_r_Blue.clear()
    N_matrix_i_Red.clear()
    N_matrix_i_White.clear()
    N_matrix_i_Blue.clear()
    N_matrix_r_RedGen.clear()
```

```
    N_matrix_i_RedGen.clear()
    N_matrix_r_WhiteGen.clear()
    N_matrix_i_WhiteGen.clear()
    N_matrix_r_BlueGen.clear()
    N_matrix_i_BlueGen.clear()
    N2_matrix_r_Red.clear()
    N2_matrix_r_White.clear()
    N2_matrix_r_Blue.clear()
    N2_matrix_i_Red.clear()
    N2_matrix_i_White.clear()
    N2_matrix_i_Blue.clear()
    N2_matrix_r_RedGen.clear()
    N2_matrix_i_RedGen.clear()
    N2_matrix_r_WhiteGen.clear()
    N2_matrix_i_WhiteGen.clear()
    N2_matrix_r_BlueGen.clear()
    N2_matrix_i_BlueGen.clear()
    N_matrix_r_Red=copy.deepcopy(N_matrix_r_RedLaterals[0])
    N_matrix_r_White=copy.deepcopy(N_matrix_r_WhiteLaterals[0])
    N_matrix_r_Blue=copy.deepcopy(N_matrix_r_BlueLaterals[0])
    N_matrix_i_Red=copy.deepcopy(N_matrix_i_RedLaterals[0])
    N_matrix_i_White=copy.deepcopy(N_matrix_i_WhiteLaterals[0])
    N_matrix_i_Blue=copy.deepcopy(N_matrix_i_BlueLaterals[0])
    N_matrix_r_RedGen=copy.deepcopy(N_matrix_r_RedGenLaterals[0])
    N_matrix_i_RedGen=copy.deepcopy(N_matrix_i_RedGenLaterals[0])
    N_matrix_r_WhiteGen=copy.deepcopy(N_matrix_r_WhiteGenLaterals[0])
    N_matrix_i_WhiteGen=copy.deepcopy(N_matrix_i_WhiteGenLaterals[0])
    N_matrix_r_BlueGen=copy.deepcopy(N_matrix_r_BlueGenLaterals[0])
    N_matrix_i_BlueGen=copy.deepcopy(N_matrix_i_BlueGenLaterals[0])
    N2_matrix_r_Red=copy.deepcopy(N2_matrix_r_RedLaterals[0])
    N2_matrix_r_White=copy.deepcopy(N2_matrix_r_WhiteLaterals[0])
    N2_matrix_r_Blue=copy.deepcopy(N2_matrix_r_BlueLaterals[0])
    N2_matrix_i_Red=copy.deepcopy(N2_matrix_i_RedLaterals[0])
    N2_matrix_i_White=copy.deepcopy(N2_matrix_i_WhiteLaterals[0])
    N2_matrix_i_Blue=copy.deepcopy(N2_matrix_i_BlueLaterals[0])
    N2_matrix_r_RedGen=copy.deepcopy(N2_matrix_r_RedGenLaterals[0])
    N2_matrix_i_RedGen=copy.deepcopy(N2_matrix_i_RedGenLaterals[0])
    N2_matrix_r_WhiteGen=copy.deepcopy(N2_matrix_r_WhiteGenLaterals[0])
    N2_matrix_i_WhiteGen=copy.deepcopy(N2_matrix_i_WhiteGenLaterals[0])
    N2_matrix_r_BlueGen=copy.deepcopy(N2_matrix_r_BlueGenLaterals[0])
    N2_matrix_i_BlueGen=copy.deepcopy(N2_matrix_i_BlueGenLaterals[0])
  if(len(Data)!=len(Lats) and i<=(Data[0]-2)):
    N_matrix_r_Red.clear()
    N_matrix_r_White.clear()
    N_matrix_r_Blue.clear()
    N_matrix_i_Red.clear()
    N_matrix_i_White.clear()
    N_matrix_i_Blue.clear()
    N_matrix_r_RedGen.clear()
    N_matrix_i_RedGen.clear()
    N_matrix_r_WhiteGen.clear()
    N_matrix_i_WhiteGen.clear()
    N_matrix_r_BlueGen.clear()
    N_matrix_i_BlueGen.clear()
```

```
    N2_matrix_r_Red.clear()
    N2_matrix_r_White.clear()
    N2_matrix_r_Blue.clear()
    N2_matrix_i_Red.clear()
    N2_matrix_i_White.clear()
    N2_matrix_i_Blue.clear()
    N2_matrix_r_RedGen.clear()
    N2_matrix_i_RedGen.clear()
    N2_matrix_r_WhiteGen.clear()
    N2_matrix_i_WhiteGen.clear()
    N2_matrix_r_BlueGen.clear()
    N2_matrix_i_BlueGen.clear()
    N_matrix_r_Red=copy.deepcopy(N_matrix_r_RedLaterals[0])
    N_matrix_r_White=copy.deepcopy(N_matrix_r_WhiteLaterals[0])
    N_matrix_r_Blue=copy.deepcopy(N_matrix_r_BlueLaterals[0])
    N_matrix_i_Red=copy.deepcopy(N_matrix_i_RedLaterals[0])
    N_matrix_i_White=copy.deepcopy(N_matrix_i_WhiteLaterals[0])
    N_matrix_i_Blue=copy.deepcopy(N_matrix_i_BlueLaterals[0])
    N_matrix_r_RedGen=copy.deepcopy(N_matrix_r_RedGenLaterals[0])
    N_matrix_i_RedGen=copy.deepcopy(N_matrix_i_RedGenLaterals[0])
    N_matrix_r_WhiteGen=copy.deepcopy(N_matrix_r_WhiteGenLaterals[0])
    N_matrix_i_WhiteGen=copy.deepcopy(N_matrix_i_WhiteGenLaterals[0])
    N_matrix_r_BlueGen=copy.deepcopy(N_matrix_r_BlueGenLaterals[0])
    N_matrix_i_BlueGen=copy.deepcopy(N_matrix_i_BlueGenLaterals[0])
    N2_matrix_r_Red=copy.deepcopy(N2_matrix_r_RedLaterals[0])
    N2_matrix_r_White=copy.deepcopy(N2_matrix_r_WhiteLaterals[0])
    N2_matrix_r_Blue=copy.deepcopy(N2_matrix_r_BlueLaterals[0])
    N2_matrix_i_Red=copy.deepcopy(N2_matrix_i_RedLaterals[0])
    N2_matrix_i_White=copy.deepcopy(N2_matrix_i_WhiteLaterals[0])
    N2_matrix_i_Blue=copy.deepcopy(N2_matrix_i_BlueLaterals[0])
    N2_matrix_r_RedGen=copy.deepcopy(N2_matrix_r_RedGenLaterals[0])
    N2_matrix_i_RedGen=copy.deepcopy(N2_matrix_i_RedGenLaterals[0])
    N2_matrix_r_WhiteGen=copy.deepcopy(N2_matrix_r_WhiteGenLaterals[0])
    N2_matrix_i_WhiteGen=copy.deepcopy(N2_matrix_i_WhiteGenLaterals[0])
    N2_matrix_r_BlueGen=copy.deepcopy(N2_matrix_r_BlueGenLaterals[0])
    N2_matrix_i_BlueGen=copy.deepcopy(N2_matrix_i_BlueGenLaterals[0])
  #Stores moments in arrays for nodes appearing before lateral connection points
  if(len(Data)==len(Lats) and i==(Data[0]-2)):
    N_matrix_r_Pre_RedLaterals.append(N_matrix_r_Red)
    N_matrix_r_Pre_WhiteLaterals.append(N_matrix_r_White)
    N_matrix_r_Pre_BlueLaterals.append(N_matrix_r_Blue)
    N_matrix_i_Pre_RedLaterals.append(N_matrix_i_Red)
    N_matrix_i_Pre_WhiteLaterals.append(N_matrix_i_White)
    N_matrix_i_Pre_BlueLaterals.append(N_matrix_i_Blue)
    N_matrix_r_Pre_RedGenLaterals.append(N_matrix_r_RedGen)
    N_matrix_i_Pre_RedGenLaterals.append(N_matrix_i_RedGen)
    N_matrix_r_Pre_WhiteGenLaterals.append(N_matrix_r_WhiteGen)
    N_matrix_i_Pre_WhiteGenLaterals.append(N_matrix_i_WhiteGen)
    N_matrix_r_Pre_BlueGenLaterals.append(N_matrix_r_BlueGen)
    N_matrix_i_Pre_BlueGenLaterals.append(N_matrix_i_BlueGen)
    N2_matrix_r_Pre_RedLaterals.append(N2_matrix_r_Red)
    N2_matrix_r_Pre_WhiteLaterals.append(N2_matrix_r_White)
    N2_matrix_r_Pre_BlueLaterals.append(N2_matrix_r_Blue)
    N2_matrix_i_Pre_RedLaterals.append(N2_matrix_i_Red)
```

```
            N2_matrix_i_Pre_WhiteLaterals.append(N2_matrix_i_White)
            N2_matrix_i_Pre_BlueLaterals.append(N2_matrix_i_Blue)
            N2_matrix_r_Pre_RedGenLaterals.append(N2_matrix_r_RedGen)
            N2_matrix_i_Pre_RedGenLaterals.append(N2_matrix_i_RedGen)
            N2_matrix_r_Pre_WhiteGenLaterals.append(N2_matrix_r_WhiteGen)
            N2_matrix_i_Pre_WhiteGenLaterals.append(N2_matrix_i_WhiteGen)
            N2_matrix_r_Pre_BlueGenLaterals.append(N2_matrix_r_BlueGen)
            N2_matrix_i_Pre_BlueGenLaterals.append(N2_matrix_i_BlueGen)
        #Used to move to the successive node after each iteration
        del(FirstMoment_Vr_RedCopy[:len(Iteration)])
        del(FirstMoment_Vr_WhiteCopy[:len(Iteration)])
        del(FirstMoment_Vr_BlueCopy[:len(Iteration)])
        del(FirstMoment_Vi_RedCopy[:len(Iteration)])
        del(FirstMoment_Vi_WhiteCopy[:len(Iteration)])
        del(FirstMoment_Vi_BlueCopy[:len(Iteration)])
        del(FirstMoment_Vr_Gen_RedCopy[:len(Iteration)])
        del(FirstMoment_Vi_Gen_RedCopy[:len(Iteration)])
        del(FirstMoment_Vr_Gen_WhiteCopy[:len(Iteration)])
        del(FirstMoment_Vi_Gen_WhiteCopy[:len(Iteration)])
        del(FirstMoment_Vr_Gen_BlueCopy[:len(Iteration)])
        del(FirstMoment_Vi_Gen_BlueCopy[:len(Iteration)])
        del(SecondMoment_Vr_RedCopy[:len(Iteration)])
        del(SecondMoment_Vr_Gen_RedCopy[:len(Iteration)])
        del(SecondMoment_Vi_RedCopy[:len(Iteration)])
        del(SecondMoment_Vi_Gen_RedCopy[:len(Iteration)])
        del(SecondMoment_Vr_WhiteCopy[:len(Iteration)])
        del(SecondMoment_Vr_Gen_WhiteCopy[:len(Iteration)])
        del(SecondMoment_Vi_WhiteCopy[:len(Iteration)])
        del(SecondMoment_Vi_Gen_WhiteCopy[:len(Iteration)])
        del(SecondMoment_Vr_BlueCopy[:len(Iteration)])
        del(SecondMoment_Vr_Gen_BlueCopy[:len(Iteration)])
        del(SecondMoment_Vi_BlueCopy[:len(Iteration)])
        del(SecondMoment_Vi_Gen_BlueCopy[:len(Iteration)])
    else:
        StoreValue=copy.deepcopy(Data)
        N_matrix_r_Red=FirstMoment_Vr_RedCopy[:(len(Iteration)-i)]
        N_matrix_r_White=FirstMoment_Vr_WhiteCopy[:(len(Iteration)-i)]
        N_matrix_r_Blue=FirstMoment_Vr_BlueCopy[:(len(Iteration)-i)]
        N_matrix_i_Red=FirstMoment_Vi_RedCopy[:(len(Iteration)-i)]
        N_matrix_i_White=FirstMoment_Vi_WhiteCopy[:(len(Iteration)-i)]
        N_matrix_i_Blue=FirstMoment_Vi_BlueCopy[:(len(Iteration)-i)]
        N_matrix_r_RedGen=FirstMoment_Vr_Gen_RedCopy[:(len(Iteration)-i)]
        N_matrix_i_RedGen=FirstMoment_Vi_Gen_RedCopy[:(len(Iteration)-i)]
        N_matrix_r_WhiteGen=FirstMoment_Vr_Gen_WhiteCopy[:(len(Iteration)-i)]
        N_matrix_i_WhiteGen=FirstMoment_Vi_Gen_WhiteCopy[:(len(Iteration)-i)]
        N_matrix_r_BlueGen=FirstMoment_Vr_Gen_BlueCopy[:(len(Iteration)-i)]
        N_matrix_i_BlueGen=FirstMoment_Vi_Gen_BlueCopy[:(len(Iteration)-i)]
        N2_matrix_r_Red=SecondMoment_Vr_RedCopy[:(len(Iteration)-i)]
        N2_matrix_r_RedGen=SecondMoment_Vr_Gen_RedCopy[:(len(Iteration)-i)]
        N2_matrix_i_Red=SecondMoment_Vi_RedCopy[:(len(Iteration)-i)]
        N2_matrix_i_RedGen=SecondMoment_Vi_Gen_RedCopy[:(len(Iteration)-i)]
        N2_matrix_r_White=SecondMoment_Vr_WhiteCopy[:(len(Iteration)-i)]
        N2_matrix_r_WhiteGen=SecondMoment_Vr_Gen_WhiteCopy[:(len(Iteration)-i)]
        N2_matrix_i_White=SecondMoment_Vi_WhiteCopy[:(len(Iteration)-i)]
```

```
N2_matrix_i_WhiteGen=SecondMoment_Vi_Gen_WhiteCopy[:(len(Iteration)-i)]
N2_matrix_r_Blue=SecondMoment_Vr_BlueCopy[:(len(Iteration)-i)]
N2_matrix_r_BlueGen=SecondMoment_Vr_Gen_BlueCopy[:(len(Iteration)-i)]
N2_matrix_i_Blue=SecondMoment_Vi_BlueCopy[:(len(Iteration)-i)]
N2_matrix_i_BlueGen=SecondMoment_Vi_Gen_BlueCopy[:(len(Iteration)-i)]
#Replace moment values with 0 where required (due to laterals)
for h in range(Laterals):
  if(h==0 and (i<(StoreValue[6]-i))):
    for f in range(i,StoreValue[6]-i):
      p.append(f)
  elif((i>=(StoreValue[6]-i)) and h==0):
    for e in range(0,StoreValue[6]-i):
      p.append(e)
  elif(StoreValue[5]<=(i+2)<=StoreValue[6]):
    for d in range((i+2),StoreValue[6]+1):
      p.append(d-i-2)
  elif((i+2)>(StoreValue[0])):
    pass
  else:
    for m in range(len(N_matrix_r_Red)):
      if((StoreValue[5]-2)<=m<=(StoreValue[6]-2)):
        p.append(m)
      elif((m==StoreValue[0]-2) and h==0):
        p.append(m)
      else:
        pass
  del(StoreValue[:3])

for q in range(0,len(p)):
  if(len(p)==1):
    for d in range(1,len(N_matrix_r_Red)):
      N_matrix_r_Red[d]=0
      N_matrix_r_White[d]=0
      N_matrix_r_Blue[d]=0
      N_matrix_i_Red[d]=0
      N_matrix_i_White[d]=0
      N_matrix_i_Blue[d]=0
      N_matrix_r_RedGen[d]=0
      N_matrix_i_RedGen[d]=0
      N_matrix_r_WhiteGen[d]=0
      N_matrix_i_WhiteGen[d]=0
      N_matrix_r_BlueGen[d]=0
      N_matrix_i_BlueGen[d]=0
      N2_matrix_r_Red[d]=0
      N2_matrix_r_White[d]=0
      N2_matrix_r_Blue[d]=0
      N2_matrix_i_Red[d]=0
      N2_matrix_i_White[d]=0
      N2_matrix_i_Blue[d]=0
      N2_matrix_r_RedGen[d]=0
      N2_matrix_i_RedGen[d]=0
      N2_matrix_r_WhiteGen[d]=0
      N2_matrix_i_WhiteGen[d]=0
      N2_matrix_r_BlueGen[d]=0
```

```
        N2_matrix_i_BlueGen[d]=0
    else:
      if((p[q]-p[q-1])!=1):
        for d in range(p[q-1],p[q]-i):
          N_matrix_r_Red[d]=0
          N_matrix_r_White[d]=0
          N_matrix_r_Blue[d]=0
          N_matrix_i_Red[d]=0
          N_matrix_i_White[d]=0
          N_matrix_i_Blue[d]=0
          N_matrix_r_RedGen[d]=0
          N_matrix_i_RedGen[d]=0
          N_matrix_r_WhiteGen[d]=0
          N_matrix_i_WhiteGen[d]=0
          N_matrix_r_BlueGen[d]=0
          N_matrix_i_BlueGen[d]=0
          N2_matrix_r_Red[d]=0
          N2_matrix_r_White[d]=0
          N2_matrix_r_Blue[d]=0
          N2_matrix_i_Red[d]=0
          N2_matrix_i_White[d]=0
          N2_matrix_i_Blue[d]=0
          N2_matrix_r_RedGen[d]=0
          N2_matrix_i_RedGen[d]=0
          N2_matrix_r_WhiteGen[d]=0
          N2_matrix_i_WhiteGen[d]=0
          N2_matrix_r_BlueGen[d]=0
          N2_matrix_i_BlueGen[d]=0
      elif(p[len(p)-1]!=(len(N_matrix_r_Red)-1)):
        for d in range(p[len(p)-1],len(N_matrix_r_Red)):
          N_matrix_r_Red[d]=0
          N_matrix_r_White[d]=0
          N_matrix_r_Blue[d]=0
          N_matrix_i_Red[d]=0
          N_matrix_i_White[d]=0
          N_matrix_i_Blue[d]=0
          N_matrix_r_RedGen[d]=0
          N_matrix_i_RedGen[d]=0
          N_matrix_r_WhiteGen[d]=0
          N_matrix_i_WhiteGen[d]=0
          N_matrix_r_BlueGen[d]=0
          N_matrix_i_BlueGen[d]=0
          N2_matrix_r_Red[d]=0
          N2_matrix_r_White[d]=0
          N2_matrix_r_Blue[d]=0
          N2_matrix_i_Red[d]=0
          N2_matrix_i_White[d]=0
          N2_matrix_i_Blue[d]=0
          N2_matrix_r_RedGen[d]=0
          N2_matrix_i_RedGen[d]=0
          N2_matrix_r_WhiteGen[d]=0
          N2_matrix_i_WhiteGen[d]=0
          N2_matrix_r_BlueGen[d]=0
          N2_matrix_i_BlueGen[d]=0
```

```
for k in range(int(len(Lats)/3)):
   if(len(Data)==len(Lats) and i==(Data[3*k]-2)):
      N_matrix_r_RedLaterals.append(N_matrix_r_Red)
      N_matrix_r_WhiteLaterals.append(N_matrix_r_White)
      N_matrix_r_BlueLaterals.append(N_matrix_r_Blue)
      N_matrix_i_RedLaterals.append(N_matrix_i_Red)
      N_matrix_i_WhiteLaterals.append(N_matrix_i_White)
      N_matrix_i_BlueLaterals.append(N_matrix_i_Blue)
      N_matrix_r_RedGenLaterals.append(N_matrix_r_RedGen)
      N_matrix_i_RedGenLaterals.append(N_matrix_i_RedGen)
      N_matrix_r_WhiteGenLaterals.append(N_matrix_r_WhiteGen)
      N_matrix_i_WhiteGenLaterals.append(N_matrix_i_WhiteGen)
      N_matrix_r_BlueGenLaterals.append(N_matrix_r_BlueGen)
      N_matrix_i_BlueGenLaterals.append(N_matrix_i_BlueGen)
      N2_matrix_r_RedLaterals.append(N2_matrix_r_Red)
      N2_matrix_r_WhiteLaterals.append(N2_matrix_r_White)
      N2_matrix_r_BlueLaterals.append(N2_matrix_r_Blue)
      N2_matrix_i_RedLaterals.append(N2_matrix_i_Red)
      N2_matrix_i_WhiteLaterals.append(N2_matrix_i_White)
      N2_matrix_i_BlueLaterals.append(N2_matrix_i_Blue)
      N2_matrix_r_RedGenLaterals.append(N2_matrix_r_RedGen)
      N2_matrix_i_RedGenLaterals.append(N2_matrix_i_RedGen)
      N2_matrix_r_WhiteGenLaterals.append(N2_matrix_r_WhiteGen)
      N2_matrix_i_WhiteGenLaterals.append(N2_matrix_i_WhiteGen)
      N2_matrix_r_BlueGenLaterals.append(N2_matrix_r_BlueGen)
      N2_matrix_i_BlueGenLaterals.append(N2_matrix_i_BlueGen)
for k in range(int(len(Lats)/3)):
   if(len(Data)==len(Lats) and i<(Data[3*k]-2)):
      N_matrix_r_Pre_RedLaterals.append(N_matrix_r_Red)
      N_matrix_r_Pre_WhiteLaterals.append(N_matrix_r_White)
      N_matrix_r_Pre_BlueLaterals.append(N_matrix_r_Blue)
      N_matrix_i_Pre_RedLaterals.append(N_matrix_i_Red)
      N_matrix_i_Pre_WhiteLaterals.append(N_matrix_i_White)
      N_matrix_i_Pre_BlueLaterals.append(N_matrix_i_Blue)
      N_matrix_r_Pre_RedGenLaterals.append(N_matrix_r_RedGen)
      N_matrix_i_Pre_RedGenLaterals.append(N_matrix_i_RedGen)
      N_matrix_r_Pre_WhiteGenLaterals.append(N_matrix_r_WhiteGen)
      N_matrix_i_Pre_WhiteGenLaterals.append(N_matrix_i_WhiteGen)
      N_matrix_r_Pre_BlueGenLaterals.append(N_matrix_r_BlueGen)
      N_matrix_i_Pre_BlueGenLaterals.append(N_matrix_i_BlueGen)
      N2_matrix_r_Pre_RedLaterals.append(N2_matrix_r_Red)
      N2_matrix_r_Pre_WhiteLaterals.append(N2_matrix_r_White)
      N2_matrix_r_Pre_BlueLaterals.append(N2_matrix_r_Blue)
      N2_matrix_i_Pre_RedLaterals.append(N2_matrix_i_Red)
      N2_matrix_i_Pre_WhiteLaterals.append(N2_matrix_i_White)
      N2_matrix_i_Pre_BlueLaterals.append(N2_matrix_i_Blue)
      N2_matrix_r_Pre_RedGenLaterals.append(N2_matrix_r_RedGen)
      N2_matrix_i_Pre_RedGenLaterals.append(N2_matrix_i_RedGen)
      N2_matrix_r_Pre_WhiteGenLaterals.append(N2_matrix_r_WhiteGen)
      N2_matrix_i_Pre_WhiteGenLaterals.append(N2_matrix_i_WhiteGen)
      N2_matrix_r_Pre_BlueGenLaterals.append(N2_matrix_r_BlueGen)
      N2_matrix_i_Pre_BlueGenLaterals.append(N2_matrix_i_BlueGen)
if(len(Data)!=len(Lats) and i<=(Data[0]-2)):
   for k in range(1,int(len(Lats)/3)):
```

```
if(i==(Lats[3*k]-2)):
    o=k-1
    N_matrix_r_Red.clear()
    N_matrix_r_White.clear()
    N_matrix_r_Blue.clear()
    N_matrix_i_Red.clear()
    N_matrix_i_White.clear()
    N_matrix_i_Blue.clear()
    N_matrix_r_RedGen.clear()
    N_matrix_i_RedGen.clear()
    N_matrix_r_WhiteGen.clear()
    N_matrix_i_WhiteGen.clear()
    N_matrix_r_BlueGen.clear()
    N_matrix_i_BlueGen.clear()
    N2_matrix_r_Red.clear()
    N2_matrix_r_White.clear()
    N2_matrix_r_Blue.clear()
    N2_matrix_i_Red.clear()
    N2_matrix_i_White.clear()
    N2_matrix_i_Blue.clear()
    N2_matrix_r_RedGen.clear()
    N2_matrix_i_RedGen.clear()
    N2_matrix_r_WhiteGen.clear()
    N2_matrix_i_WhiteGen.clear()
    N2_matrix_r_BlueGen.clear()
    N2_matrix_i_BlueGen.clear()

    N_matrix_r_Red=copy.deepcopy(N_matrix_r_RedLaterals[o])
    N_matrix_r_White=copy.deepcopy(N_matrix_r_WhiteLaterals[o])
    N_matrix_r_Blue=copy.deepcopy(N_matrix_r_BlueLaterals[o])
    N_matrix_i_Red=copy.deepcopy(N_matrix_i_RedLaterals[o])
    N_matrix_i_White=copy.deepcopy(N_matrix_i_WhiteLaterals[o])
    N_matrix_i_Blue=copy.deepcopy(N_matrix_i_BlueLaterals[o])
    N_matrix_r_RedGen=copy.deepcopy(N_matrix_r_RedGenLaterals[o])
    N_matrix_i_RedGen=copy.deepcopy(N_matrix_i_RedGenLaterals[o])
    N_matrix_r_WhiteGen=copy.deepcopy(N_matrix_r_WhiteGenLaterals[o])
    N_matrix_i_WhiteGen=copy.deepcopy(N_matrix_i_WhiteGenLaterals[o])
    N_matrix_r_BlueGen=copy.deepcopy(N_matrix_r_BlueGenLaterals[o])
    N_matrix_i_BlueGen=copy.deepcopy(N_matrix_i_BlueGenLaterals[o])
    N2_matrix_r_Red=copy.deepcopy(N2_matrix_r_RedLaterals[o])
    N2_matrix_r_White=copy.deepcopy(N2_matrix_r_WhiteLaterals[o])
    N2_matrix_r_Blue=copy.deepcopy(N2_matrix_r_BlueLaterals[o])
    N2_matrix_i_Red=copy.deepcopy(N2_matrix_i_RedLaterals[o])
    N2_matrix_i_White=copy.deepcopy(N2_matrix_i_WhiteLaterals[o])
    N2_matrix_i_Blue=copy.deepcopy(N2_matrix_i_BlueLaterals[o])
    N2_matrix_r_RedGen=copy.deepcopy(N2_matrix_r_RedGenLaterals[o])
    N2_matrix_i_RedGen=copy.deepcopy(N2_matrix_i_RedGenLaterals[o])
    N2_matrix_r_WhiteGen=copy.deepcopy(N2_matrix_r_WhiteGenLaterals[o])
    N2_matrix_i_WhiteGen=copy.deepcopy(N2_matrix_i_WhiteGenLaterals[o])
    N2_matrix_r_BlueGen=copy.deepcopy(N2_matrix_r_BlueGenLaterals[o])
    N2_matrix_i_BlueGen=copy.deepcopy(N2_matrix_i_BlueGenLaterals[o])
else:
    if(k==(int(len(Lats)/3)-1)):
        N_matrix_r_Red.clear()
```

```
            N_matrix_r_White.clear()
            N_matrix_r_Blue.clear()
            N_matrix_i_Red.clear()
            N_matrix_i_White.clear()
            N_matrix_i_Blue.clear()
            N_matrix_r_RedGen.clear()
            N_matrix_i_RedGen.clear()
            N_matrix_r_WhiteGen.clear()
            N_matrix_i_WhiteGen.clear()
            N_matrix_r_BlueGen.clear()
            N_matrix_i_BlueGen.clear()
            N2_matrix_r_Red.clear()
            N2_matrix_r_White.clear()
            N2_matrix_r_Blue.clear()
            N2_matrix_i_Red.clear()
            N2_matrix_i_White.clear()
            N2_matrix_i_Blue.clear()
            N2_matrix_r_RedGen.clear()
            N2_matrix_i_RedGen.clear()
            N2_matrix_r_WhiteGen.clear()
            N2_matrix_i_WhiteGen.clear()
            N2_matrix_r_BlueGen.clear()
            N2_matrix_i_BlueGen.clear()
            N_matrix_r_Red=copy.deepcopy(N_matrix_r_Pre_RedLaterals[i])
            N_matrix_r_White=copy.deepcopy(N_matrix_r_Pre_WhiteLaterals[i])
            N_matrix_r_Blue=copy.deepcopy(N_matrix_r_Pre_BlueLaterals[i])
            N_matrix_i_Red=copy.deepcopy(N_matrix_i_Pre_RedLaterals[i])
            N_matrix_i_White=copy.deepcopy(N_matrix_i_Pre_WhiteLaterals[i])
            N_matrix_i_Blue=copy.deepcopy(N_matrix_i_Pre_BlueLaterals[i])
            N_matrix_r_RedGen=copy.deepcopy(N_matrix_r_Pre_RedGenLaterals[i])
            N_matrix_i_RedGen=copy.deepcopy(N_matrix_i_Pre_RedGenLaterals[i])
            N_matrix_r_WhiteGen=copy.deepcopy(N_matrix_r_Pre_WhiteGenLaterals[i])
            N_matrix_i_WhiteGen=copy.deepcopy(N_matrix_i_Pre_WhiteGenLaterals[i])
            N_matrix_r_BlueGen=copy.deepcopy(N_matrix_r_Pre_BlueGenLaterals[i])
            N_matrix_i_BlueGen=copy.deepcopy(N_matrix_i_Pre_BlueGenLaterals[i])
            N2_matrix_r_Red=copy.deepcopy(N2_matrix_r_Pre_RedLaterals[i])
            N2_matrix_r_White=copy.deepcopy(N2_matrix_r_Pre_WhiteLaterals[i])
            N2_matrix_r_Blue=copy.deepcopy(N2_matrix_r_Pre_BlueLaterals[i])
            N2_matrix_i_Red=copy.deepcopy(N2_matrix_i_Pre_RedLaterals[i])
            N2_matrix_i_White=copy.deepcopy(N2_matrix_i_Pre_WhiteLaterals[i])
            N2_matrix_i_Blue=copy.deepcopy(N2_matrix_i_Pre_BlueLaterals[i])
            N2_matrix_r_RedGen=copy.deepcopy(N2_matrix_r_Pre_RedGenLaterals[i])
            N2_matrix_i_RedGen=copy.deepcopy(N2_matrix_i_Pre_RedGenLaterals[i])
            N2_matrix_r_WhiteGen=copy.deepcopy(N2_matrix_r_Pre_WhiteGenLaterals[i])
            N2_matrix_i_WhiteGen=copy.deepcopy(N2_matrix_i_Pre_WhiteGenLaterals[i])
            N2_matrix_r_BlueGen=copy.deepcopy(N2_matrix_r_Pre_BlueGenLaterals[i])
            N2_matrix_i_BlueGen=copy.deepcopy(N2_matrix_i_Pre_BlueGenLaterals[i])

    del(FirstMoment_Vr_RedCopy[:(len(Iteration)-i)])
    del(FirstMoment_Vr_WhiteCopy[:(len(Iteration)-i)])
    del(FirstMoment_Vr_BlueCopy[:(len(Iteration)-i)])
    del(FirstMoment_Vi_RedCopy[:(len(Iteration)-i)])
    del(FirstMoment_Vi_WhiteCopy[:(len(Iteration)-i)])
    del(FirstMoment_Vi_BlueCopy[:(len(Iteration)-i)])
```

```
    del(FirstMoment_Vr_Gen_RedCopy[:(len(Iteration)-i)])
    del(FirstMoment_Vi_Gen_RedCopy[:(len(Iteration)-i)])
    del(FirstMoment_Vr_Gen_WhiteCopy[:(len(Iteration)-i)])
    del(FirstMoment_Vi_Gen_WhiteCopy[:(len(Iteration)-i)])
    del(FirstMoment_Vr_Gen_BlueCopy[:(len(Iteration)-i)])
    del(FirstMoment_Vi_Gen_BlueCopy[:(len(Iteration)-i)])
    del(SecondMoment_Vr_RedCopy[:(len(Iteration)-i)])
    del(SecondMoment_Vr_Gen_RedCopy[:(len(Iteration)-i)])
    del(SecondMoment_Vi_RedCopy[:(len(Iteration)-i)])
    del(SecondMoment_Vi_Gen_RedCopy[:(len(Iteration)-i)])
    del(SecondMoment_Vr_WhiteCopy[:(len(Iteration)-i)])
    del(SecondMoment_Vr_Gen_WhiteCopy[:(len(Iteration)-i)])
    del(SecondMoment_Vi_WhiteCopy[:(len(Iteration)-i)])
    del(SecondMoment_Vi_Gen_WhiteCopy[:(len(Iteration)-i)])
    del(SecondMoment_Vr_BlueCopy[:(len(Iteration)-i)])
    del(SecondMoment_Vr_Gen_BlueCopy[:(len(Iteration)-i)])
    del(SecondMoment_Vi_BlueCopy[:(len(Iteration)-i)])
    del(SecondMoment_Vi_Gen_BlueCopy[:(len(Iteration)-i)])
#This section determines the B1 and B2 for each node
if(len(N_matrix_r_Red)>2):
    for k in range(len(N_matrix_r_Red)):
        for j in range(len(N_matrix_r_Red)):
            if(j!=k):
                B1TempRed.append(N_matrix_r_Red[k]*N_matrix_r_Red[j])
                B1TempWhite.append(N_matrix_r_White[k]*N_matrix_r_White[j])
                B1TempBlue.append(N_matrix_r_Blue[k]*N_matrix_r_Blue[j])
                B2TempRed.append(N_matrix_i_Red[k]*N_matrix_i_Red[j])
                B2TempWhite.append(N_matrix_i_White[k]*N_matrix_i_White[j])
                B2TempBlue.append(N_matrix_i_Blue[k]*N_matrix_i_Blue[j])
                B1TempRedGen.append(N_matrix_r_RedGen[k]*N_matrix_r_RedGen[j])
                B2TempRedGen.append(N_matrix_i_RedGen[k]*N_matrix_i_RedGen[j])
                B1TempWhiteGen.append(N_matrix_r_WhiteGen[k]*N_matrix_r_WhiteGen[j])
                B2TempWhiteGen.append(N_matrix_i_WhiteGen[k]*N_matrix_i_WhiteGen[j])
                B1TempBlueGen.append(N_matrix_r_BlueGen[k]*N_matrix_r_BlueGen[j])
                B2TempBlueGen.append(N_matrix_i_BlueGen[k]*N_matrix_i_BlueGen[j])
elif(len(N_matrix_r_Red)==2):
    for k in range(len(N_matrix_r_Red)):
        for j in range(len(N_matrix_r_Red)):
            if(j!=k):
                B1TempRed.append(N_matrix_r_Red[k]*N_matrix_r_Red[j])
                B1TempWhite.append(N_matrix_r_White[k]*N_matrix_r_White[j])
                B1TempBlue.append(N_matrix_r_Blue[k]*N_matrix_r_Blue[j])
                B2TempRed.append(N_matrix_i_Red[k]*N_matrix_i_Red[j])
                B2TempWhite.append(N_matrix_i_White[k]*N_matrix_i_White[j])
                B2TempBlue.append(N_matrix_i_Blue[k]*N_matrix_i_Blue[j])
                B1TempRedGen.append(N_matrix_r_RedGen[k]*N_matrix_r_RedGen[j])
                B2TempRedGen.append(N_matrix_i_RedGen[k]*N_matrix_i_RedGen[j])
                B1TempWhiteGen.append(N_matrix_r_WhiteGen[k]*N_matrix_r_WhiteGen[j])
                B2TempWhiteGen.append(N_matrix_i_WhiteGen[k]*N_matrix_i_WhiteGen[j])
                B1TempBlueGen.append(N_matrix_r_BlueGen[k]*N_matrix_r_BlueGen[j])
                B2TempBlueGen.append(N_matrix_i_BlueGen[k]*N_matrix_i_BlueGen[j])
elif(len(N_matrix_r_Red)==1):
    B1TempRed.append(0)
    B1TempWhite.append(0)
```

```
    B1TempBlue.append(0)
    B2TempRed.append(0)
    B2TempWhite.append(0)
    B2TempBlue.append(0)
    B1TempRedGen.append(0)
    B2TempRedGen.append(0)
    B1TempWhiteGen.append(0)
    B2TempWhiteGen.append(0)
    B1TempBlueGen.append(0)
    B2TempBlueGen.append(0)

if(len(N_matrix_r_Past_Red)>2):
    for k in range(len(N_matrix_r_Past_Red)):
        for j in range(len(N_matrix_r_Past_Red)):
            if(j!=k):
                B1TempRedPast.append(N_matrix_r_Past_Red[k]*N_matrix_r_Past_Red[j])
                B1TempWhitePast.append(N_matrix_r_Past_White[k]*N_matrix_r_Past_White[j])
                B1TempBluePast.append(N_matrix_r_Past_Blue[k]*N_matrix_r_Past_Blue[j])
                B2TempRedPast.append(N_matrix_i_Past_Red[k]*N_matrix_i_Past_Red[j])
                B2TempWhitePast.append(N_matrix_i_Past_White[k]*N_matrix_i_Past_White[j])
                B2TempBluePast.append(N_matrix_i_Past_Blue[k]*N_matrix_i_Past_Blue[j])
                B1TempRedGenPast.append(N_matrix_r_Past_RedGen[k]*N_matrix_r_Past_RedGen[j])
                B2TempRedGenPast.append(N_matrix_i_Past_RedGen[k]*N_matrix_i_Past_RedGen[j])
                B1TempWhiteGenPast.append(N_matrix_r_Past_WhiteGen[k]*
                 N_matrix_r_Past_WhiteGen[j])
                B2TempWhiteGenPast.append(N_matrix_i_Past_WhiteGen[k]*
                 N_matrix_i_Past_WhiteGen[j])
                B1TempBlueGenPast.append(N_matrix_r_Past_BlueGen[k]*N_matrix_r_Past_BlueGen[j])
                B2TempBlueGenPast.append(N_matrix_i_Past_BlueGen[k]*N_matrix_i_Past_BlueGen[j])
elif(len(N_matrix_r_Past_Red)==2):
    for k in range(len(N_matrix_r_Past_Red)):
        for j in range(len(N_matrix_r_Past_Red)):
            if(j!=k):
                B1TempRedPast.append(N_matrix_r_Past_Red[k]*N_matrix_r_Past_Red[j])
                B1TempWhitePast.append(N_matrix_r_Past_White[k]*N_matrix_r_Past_White[j])
                B1TempBluePast.append(N_matrix_r_Past_Blue[k]*N_matrix_r_Past_Blue[j])
                B2TempRedPast.append(N_matrix_i_Past_Red[k]*N_matrix_i_Past_Red[j])
                B2TempWhitePast.append(N_matrix_i_Past_White[k]*N_matrix_i_Past_White[j])
                B2TempBluePast.append(N_matrix_i_Past_Blue[k]*N_matrix_i_Past_Blue[j])
                B1TempRedGenPast.append(N_matrix_r_Past_RedGen[k]*N_matrix_r_Past_RedGen[j])
                B2TempRedGenPast.append(N_matrix_i_Past_RedGen[k]*N_matrix_i_Past_RedGen[j])
               B1TempWhiteGenPast.append(N_matrix_r_Past_WhiteGen[k]*
               N_matrix_r_Past_WhiteGen[j])
               B2TempWhiteGenPast.append(N_matrix_i_Past_WhiteGen[k]*
               N_matrix_i_Past_WhiteGen[j])
               B1TempBlueGenPast.append(N_matrix_r_Past_BlueGen[k]*N_matrix_r_Past_BlueGen[j])
               B2TempBlueGenPast.append(N_matrix_i_Past_BlueGen[k]*N_matrix_i_Past_BlueGen[j])
elif(len(N_matrix_r_Past_Red)==1):
    B1TempRedPast.append(0)
    B1TempWhitePast.append(0)
    B1TempBluePast.append(0)
    B2TempRedPast.append(0)
    B2TempWhitePast.append(0)
    B2TempBluePast.append(0)
```

```
    B1TempRedGenPast.append(0)
    B2TempRedGenPast.append(0)
    B1TempWhiteGenPast.append(0)
    B2TempWhiteGenPast.append(0)
    B1TempBlueGenPast.append(0)
    B2TempBlueGenPast.append(0)
B1Red.append(sum(B1TempRed))
B1RedPast.append(sum(B1TempRedPast))
B1White.append(sum(B1TempWhite))
B1Blue.append(sum(B1TempBlue))
B2Red.append(sum(B2TempRed))
B2White.append(sum(B2TempWhite))
B2Blue.append(sum(B2TempBlue))
B1RedGen.append(sum(B1TempRedGen))
B2RedGen.append(sum(B2TempRedGen))
B1WhiteGen.append(sum(B1TempWhiteGen))
B2WhiteGen.append(sum(B2TempWhiteGen))
B1BlueGen.append(sum(B1TempBlueGen))
B2BlueGen.append(sum(B2TempBlueGen))
#Reset values after each iteration
B1TempRed=[]
B1TempWhite=[]
B1TempBlue=[]
B2TempRed=[]
B2TempWhite=[]
B2TempBlue=[]
B1TempRedGen=[]
B2TempRedGen=[]
B1TempWhiteGen=[]
B2TempWhiteGen=[]
B1TempBlueGen=[]
B2TempBlueGen=[]
B1TempRedPast=[]
B1TempWhitePast=[]
B1TempBluePast=[]
B2TempRedPast=[]
B2TempWhitePast=[]
B2TempBluePast=[]
B1TempRedGenPast=[]
B2TempRedGenPast=[]
B1TempWhiteGenPast=[]
B2TempWhiteGenPast=[]
B1TempBlueGenPast=[]
B2TempBlueGenPast=[]
#Determines the B1 and B2 values for the entire feeder
if(i==0):
    for y in range(len(N_matrix_r_Red)):
        if(len(N_matrix_r_Red)==n):
            for z in range(len(N_matrix_r_Red)):
                B1TotalTempRed.append(N_matrix_r_Red[y]*N_matrix_r_RedGen[z])
                B1TotalTempRed.append(N_matrix_r_Red[z]*N_matrix_r_RedGen[y])
                B2TotalTempRed.append(N_matrix_i_Red[y]*N_matrix_i_RedGen[z])
                B2TotalTempRed.append(N_matrix_i_Red[z]*N_matrix_i_RedGen[y])
                B1TotalTempWhite.append(N_matrix_r_White[y]*N_matrix_r_WhiteGen[z])
```

```
                        B1TotalTempWhite.append(N_matrix_r_White[z]*N_matrix_r_WhiteGen[y])
                        B2TotalTempWhite.append(N_matrix_i_White[y]*N_matrix_i_WhiteGen[z])
                        B2TotalTempWhite.append(N_matrix_i_White[z]*N_matrix_i_WhiteGen[y])
                        B1TotalTempBlue.append(N_matrix_r_Blue[y]*N_matrix_r_BlueGen[z])
                        B1TotalTempBlue.append(N_matrix_r_Blue[z]*N_matrix_r_BlueGen[y])
                        B2TotalTempBlue.append(N_matrix_i_Blue[y]*N_matrix_i_BlueGen[z])
                        B2TotalTempBlue.append(N_matrix_i_Blue[z]*N_matrix_i_BlueGen[y])
                else:
                    for count in range(len(N_matrix_r_Past_RedFinal)):
                        for y in range(len(N_matrix_r_Past_RedFinal[count])):
                            for counter in range(len(N_matrix_r_Past_RedFinal)):
                                for z in range(len(N_matrix_r_Past_RedFinal[counter])):
                                    if(count!=counter):
B1TotalTempRed.append(N_matrix_r_Past_RedFinal[count][y]*N_matrix_r_Past_RedFinal[counter][z])
B1TotalTempRed.append(N_matrix_r_Past_RedGenFinal[count][y]*N_matrix_r_Past_RedGenFinal[counter][z])
B2TotalTempRed.append(N_matrix_i_Past_RedFinal[count][y]*N_matrix_i_Past_RedFinal[counter][z])
B2TotalTempRed.append(N_matrix_i_Past_RedGenFinal[count][y]*N_matrix_i_Past_RedGenFinal[counter][z])
B1TotalTempWhite.append(N_matrix_r_Past_WhiteFinal[count][y]*N_matrix_r_Past_WhiteFinal[counter][z])
B1TotalTempWhite.append(N_matrix_r_Past_WhiteGenFinal[count][y]*N_matrix_r_Past_WhiteGenFinal[counter
][z])
B2TotalTempWhite.append(N_matrix_i_Past_WhiteFinal[count][y]*N_matrix_i_Past_WhiteFinal[counter][z])
B2TotalTempWhite.append(N_matrix_i_Past_WhiteGenFinal[count][y]*N_matrix_i_Past_WhiteGenFinal[counter]
[z])
B1TotalTempBlue.append(N_matrix_r_Past_BlueFinal[count][y]*N_matrix_r_Past_BlueFinal[counter][z])
B1TotalTempBlue.append(N_matrix_r_Past_BlueGenFinal[count][y]*N_matrix_r_Past_BlueGenFinal[counter][z])
B2TotalTempBlue.append(N_matrix_i_Past_BlueFinal[count][y]*N_matrix_i_Past_BlueFinal[counter][z])
B2TotalTempBlue.append(N_matrix_i_Past_BlueGenFinal[count][y]*N_matrix_i_Past_BlueGenFinal[counter][z])
                                    else:
                                        pass
                                for z in range(len(N_matrix_r_Past_RedFinal[counter])):
B1TotalTempRed.append(N_matrix_r_Past_RedFinal[count][y]*N_matrix_r_Past_RedGenFinal[counter][z])
B1TotalTempRed.append(N_matrix_r_Past_RedFinal[count][y]*N_matrix_r_Past_RedGenFinal[counter][z])
B2TotalTempRed.append(N_matrix_i_Past_RedFinal[count][y]*N_matrix_i_Past_RedGenFinal[counter][z])
B2TotalTempRed.append(N_matrix_i_Past_RedFinal[count][y]*N_matrix_i_Past_RedGenFinal[counter][z])
B1TotalTempWhite.append(N_matrix_r_Past_WhiteFinal[count][y]*
N_matrix_r_Past_WhiteGenFinal[counter][z])
B1TotalTempWhite.append(N_matrix_r_Past_WhiteFinal[count][y]*
N_matrix_r_Past_WhiteGenFinal[counter][z])
B2TotalTempWhite.append(N_matrix_i_Past_WhiteFinal[count][y]*N_matrix_i_Past_WhiteGenFinal[counter][z])
B2TotalTempWhite.append(N_matrix_i_Past_WhiteFinal[count][y]*N_matrix_i_Past_WhiteGenFinal[counter][z])
B1TotalTempBlue.append(N_matrix_r_Past_BlueFinal[count][y]*N_matrix_r_Past_BlueGenFinal[counter][z])
B1TotalTempBlue.append(N_matrix_r_Past_BlueFinal[count][y]*N_matrix_r_Past_BlueGenFinal[counter][z])
B2TotalTempBlue.append(N_matrix_i_Past_BlueFinal[count][y]*N_matrix_i_Past_BlueGenFinal[counter][z])
B2TotalTempBlue.append(N_matrix_i_Past_BlueFinal[count][y]*N_matrix_i_Past_BlueGenFinal[counter][z])
                        for z in range(len(N_matrix_r_Red)):
                            B1TotalTempRed.append(N_matrix_r_Past_RedFinal[count][y]*N_matrix_r_Red[z])
                            B1TotalTempRed.append(N_matrix_r_Past_RedFinal[count][y]*N_matrix_r_Red[z])
                            B1TotalTempRed.append(N_matrix_r_Past_RedFinal[count][y]*N_matrix_r_RedGen[z])
                            B1TotalTempRed.append(N_matrix_r_Past_RedFinal[count][y]*N_matrix_r_RedGen[z])
                            B1TotalTempRed.append(N_matrix_r_Past_RedGenFinal[count][y]*N_matrix_r_Red[z])
                            B1TotalTempRed.append(N_matrix_r_Past_RedGenFinal[count][y]*N_matrix_r_Red[z])
B1TotalTempRed.append(N_matrix_r_Past_RedGenFinal[count][y]*N_matrix_r_RedGen[z])
B1TotalTempRed.append(N_matrix_r_Past_RedGenFinal[count][y]*N_matrix_r_RedGen[z])
                            B2TotalTempRed.append(N_matrix_i_Past_RedFinal[count][y]*N_matrix_i_Red[z])
```

```
                        B2TotalTempRed.append(N_matrix_i_Past_RedFinal[count][y]*N_matrix_i_Red[z])
                        B2TotalTempRed.append(N_matrix_i_Past_RedFinal[count][y]*N_matrix_i_RedGen[z])
                        B2TotalTempRed.append(N_matrix_i_Past_RedFinal[count][y]*N_matrix_i_RedGen[z])
                        B2TotalTempRed.append(N_matrix_i_Past_RedGenFinal[count][y]*N_matrix_i_Red[z])
                        B2TotalTempRed.append(N_matrix_i_Past_RedGenFinal[count][y]*N_matrix_i_Red[z])
B2TotalTempRed.append(N_matrix_i_Past_RedGenFinal[count][y]*N_matrix_i_RedGen[z])
B2TotalTempRed.append(N_matrix_i_Past_RedGenFinal[count][y]*N_matrix_i_RedGen[z])
                        B1TotalTempWhite.append(N_matrix_r_Past_WhiteFinal[count][y]*N_matrix_r_White[z])
                        B1TotalTempWhite.append(N_matrix_r_Past_WhiteFinal[count][y]*N_matrix_r_White[z])
B1TotalTempWhite.append(N_matrix_r_Past_WhiteFinal[count][y]*N_matrix_r_WhiteGen[z])
B1TotalTempWhite.append(N_matrix_r_Past_WhiteFinal[count][y]*N_matrix_r_WhiteGen[z])
B1TotalTempWhite.append(N_matrix_r_Past_WhiteGenFinal[count][y]*N_matrix_r_White[z])
B1TotalTempWhite.append(N_matrix_r_Past_WhiteGenFinal[count][y]*N_matrix_r_White[z])
B1TotalTempWhite.append(N_matrix_r_Past_WhiteGenFinal[count][y]*N_matrix_r_WhiteGen[z])
B1TotalTempWhite.append(N_matrix_r_Past_WhiteGenFinal[count][y]*N_matrix_r_WhiteGen[z])
                        B2TotalTempWhite.append(N_matrix_i_Past_WhiteFinal[count][y]*N_matrix_i_White[z])
                        B2TotalTempWhite.append(N_matrix_i_Past_WhiteFinal[count][y]*N_matrix_i_White[z])
B2TotalTempWhite.append(N_matrix_i_Past_WhiteFinal[count][y]*N_matrix_i_WhiteGen[z])
B2TotalTempWhite.append(N_matrix_i_Past_WhiteFinal[count][y]*N_matrix_i_WhiteGen[z])
B2TotalTempWhite.append(N_matrix_i_Past_WhiteGenFinal[count][y]*N_matrix_i_White[z])
B2TotalTempWhite.append(N_matrix_i_Past_WhiteGenFinal[count][y]*N_matrix_i_White[z])
B2TotalTempWhite.append(N_matrix_i_Past_WhiteGenFinal[count][y]*N_matrix_i_WhiteGen[z])
B2TotalTempWhite.append(N_matrix_i_Past_WhiteGenFinal[count][y]*N_matrix_i_WhiteGen[z])
                        B1TotalTempBlue.append(N_matrix_r_Past_BlueFinal[count][y]*N_matrix_r_Blue[z])
                        B1TotalTempBlue.append(N_matrix_r_Past_BlueFinal[count][y]*N_matrix_r_Blue[z])
                        B1TotalTempBlue.append(N_matrix_r_Past_BlueFinal[count][y]*N_matrix_r_BlueGen[z])
                        B1TotalTempBlue.append(N_matrix_r_Past_BlueFinal[count][y]*N_matrix_r_BlueGen[z])
                        B1TotalTempBlue.append(N_matrix_r_Past_BlueGenFinal[count][y]*N_matrix_r_Blue[z])
                        B1TotalTempBlue.append(N_matrix_r_Past_BlueGenFinal[count][y]*N_matrix_r_Blue[z])
B1TotalTempBlue.append(N_matrix_r_Past_BlueGenFinal[count][y]*N_matrix_r_BlueGen[z])
B1TotalTempBlue.append(N_matrix_r_Past_BlueGenFinal[count][y]*N_matrix_r_BlueGen[z])
                        B2TotalTempBlue.append(N_matrix_i_Past_BlueFinal[count][y]*N_matrix_i_Blue[z])
                        B2TotalTempBlue.append(N_matrix_i_Past_BlueFinal[count][y]*N_matrix_i_Blue[z])
                        B2TotalTempBlue.append(N_matrix_i_Past_BlueFinal[count][y]*N_matrix_i_BlueGen[z])
                        B2TotalTempBlue.append(N_matrix_i_Past_BlueFinal[count][y]*N_matrix_i_BlueGen[z])
                        B2TotalTempBlue.append(N_matrix_i_Past_BlueGenFinal[count][y]*N_matrix_i_Blue[z])
                        B2TotalTempBlue.append(N_matrix_i_Past_BlueGenFinal[count][y]*N_matrix_i_Blue[z])
B2TotalTempBlue.append(N_matrix_i_Past_BlueGenFinal[count][y]*N_matrix_i_BlueGen[z])
B2TotalTempBlue.append(N_matrix_i_Past_BlueGenFinal[count][y]*N_matrix_i_BlueGen[z])
                    # print('N_matrix_r_Red =',N_matrix_r_Red)
                    # print('N_matrix_r_Past_RedFinal =',N_matrix_r_Past_RedFinal)
                  for y in range(len(N_matrix_r_Red)):
                    if(len(N_matrix_r_Red)!=n):
                      for z in range(len(N_matrix_r_Red)):
                        print
                        B1TotalTempRed.append(N_matrix_r_Red[y]*N_matrix_r_RedGen[z])
                        B1TotalTempRed.append(N_matrix_r_RedGen[y]*N_matrix_r_Red[z])
                        B2TotalTempRed.append(N_matrix_i_Red[y]*N_matrix_i_RedGen[z])
                        B2TotalTempRed.append(N_matrix_i_RedGen[y]*N_matrix_i_Red[z])
                        B1TotalTempWhite.append(N_matrix_r_White[y]*N_matrix_r_WhiteGen[z])
                        B1TotalTempWhite.append(N_matrix_r_WhiteGen[y]*N_matrix_r_White[z])
                        B2TotalTempWhite.append(N_matrix_i_White[y]*N_matrix_i_WhiteGen[z])
                        B2TotalTempWhite.append(N_matrix_i_WhiteGen[y]*N_matrix_i_White[z])
                        B1TotalTempBlue.append(N_matrix_r_Blue[y]*N_matrix_r_BlueGen[z])
```

```
            B1TotalTempBlue.append(N_matrix_r_BlueGen[y]*N_matrix_r_Blue[z])
            B2TotalTempBlue.append(N_matrix_i_Blue[y]*N_matrix_i_BlueGen[z])
            B2TotalTempBlue.append(N_matrix_i_BlueGen[y]*N_matrix_i_Blue[z])
    B1TotalRed.append(sum(B1TotalTempRed))
    B2TotalRed.append(sum(B2TotalTempRed))
    B1TotalWhite.append(sum(B1TotalTempWhite))
    B2TotalWhite.append(sum(B2TotalTempWhite))
    B1TotalBlue.append(sum(B1TotalTempBlue))
    B2TotalBlue.append(sum(B2TotalTempBlue))
    N_matrix_r_Past_Red=[]
    N_matrix_r_Past_RedGen=[]
    N_matrix_r_Past_White=[]
    N_matrix_r_Past_WhiteGen=[]
    N_matrix_r_Past_Blue=[]
    N_matrix_r_Past_BlueGen=[]
    N_matrix_i_Past_Red=[]
    N_matrix_i_Past_RedGen=[]
    N_matrix_i_Past_White=[]
    N_matrix_i_Past_WhiteGen=[]
    N_matrix_i_Past_Blue=[]
    N_matrix_i_Past_BlueGen=[]
    N2_matrix_r_Past_Red=[]
    N2_matrix_r_Past_RedGen=[]
    N2_matrix_i_Past_Red=[]
    N2_matrix_i_Past_RedGen=[]
    N2_matrix_r_Past_White=[]
    N2_matrix_r_Past_WhiteGen=[]
    N2_matrix_i_Past_White=[]
    N2_matrix_i_Past_WhiteGen=[]
    N2_matrix_r_Past_Blue=[]
    N2_matrix_r_Past_BlueGen=[]
    N2_matrix_i_Past_Blue=[]
    N2_matrix_i_Past_BlueGen=[]
    #Places zero values at the required nodes for moments of previous nodes
    if(len(Lats)==len(Data)):
      for k in range(int(len(Lats)/3)):
        if(i==(Data[3*k]-2)):
            N_matrix_r_Past_Red=copy.deepcopy(N_matrix_r_Red)
            N_matrix_r_Past_RedGen=copy.deepcopy(N_matrix_r_RedGen)
            N_matrix_r_Past_White=copy.deepcopy(N_matrix_r_White)
            N_matrix_r_Past_WhiteGen=copy.deepcopy(N_matrix_r_WhiteGen)
            N_matrix_r_Past_Blue=copy.deepcopy(N_matrix_r_Blue)
            N_matrix_r_Past_BlueGen=copy.deepcopy(N_matrix_r_BlueGen)
            N_matrix_i_Past_Red=copy.deepcopy(N_matrix_i_Red)
            N_matrix_i_Past_RedGen=copy.deepcopy(N_matrix_i_RedGen)
            N_matrix_i_Past_White=copy.deepcopy(N_matrix_i_White)
            N_matrix_i_Past_WhiteGen=copy.deepcopy(N_matrix_i_WhiteGen)
            N_matrix_i_Past_Blue=copy.deepcopy(N_matrix_i_Blue)
            N_matrix_i_Past_BlueGen=copy.deepcopy(N_matrix_i_BlueGen)
            N2_matrix_r_Past_Red=copy.deepcopy(N2_matrix_r_Red)
            N2_matrix_r_Past_RedGen=copy.deepcopy(N2_matrix_r_RedGen)
            N2_matrix_i_Past_Red=copy.deepcopy(N2_matrix_i_Red)
            N2_matrix_i_Past_RedGen=copy.deepcopy(N2_matrix_i_RedGen)
            N2_matrix_r_Past_White=copy.deepcopy(N2_matrix_r_White)
```

```
N2_matrix_r_Past_WhiteGen=copy.deepcopy(N2_matrix_r_WhiteGen)
N2_matrix_i_Past_White=copy.deepcopy(N2_matrix_i_White)
N2_matrix_i_Past_WhiteGen=copy.deepcopy(N2_matrix_i_WhiteGen)
N2_matrix_r_Past_Blue=copy.deepcopy(N2_matrix_r_Blue)
N2_matrix_r_Past_BlueGen=copy.deepcopy(N2_matrix_r_BlueGen)
N2_matrix_i_Past_Blue=copy.deepcopy(N2_matrix_i_Blue)
N2_matrix_i_Past_BlueGen=copy.deepcopy(N2_matrix_i_BlueGen)
for b in range(len(N_matrix_r_Past_Red)):
  if(b==(Data[0]-2)):
    pass
  elif((Data[3*k+1]-2-i)<=b<=(Data[3*k+2]-2-i)):
    pass
  else:
    N_matrix_r_Past_Red[b]=0
    N_matrix_r_Past_RedGen[b]=0
    N_matrix_r_Past_White[b]=0
    N_matrix_r_Past_WhiteGen[b]=0
    N_matrix_r_Past_Blue[b]=0
    N_matrix_r_Past_BlueGen[b]=0
    N_matrix_i_Past_Red[b]=0
    N_matrix_i_Past_RedGen[b]=0
    N_matrix_i_Past_White[b]=0
    N_matrix_i_Past_WhiteGen[b]=0
    N_matrix_i_Past_Blue[b]=0
    N_matrix_i_Past_BlueGen[b]=0
    N2_matrix_r_Past_Red[b]=0
    N2_matrix_r_Past_RedGen[b]=0
    N2_matrix_i_Past_Red[b]=0
    N2_matrix_i_Past_RedGen[b]=0
    N2_matrix_r_Past_White[b]=0
    N2_matrix_r_Past_WhiteGen[b]=0
    N2_matrix_i_Past_White[b]=0
    N2_matrix_i_Past_WhiteGen[b]=0
    N2_matrix_r_Past_Blue[b]=0
    N2_matrix_r_Past_BlueGen[b]=0
    N2_matrix_i_Past_Blue[b]=0
    N2_matrix_i_Past_BlueGen[b]=0
N_matrix_r_Past_RedLaterals.append(N_matrix_r_Past_Red)
N_matrix_r_Past_RedGenLaterals.append(N_matrix_r_Past_RedGen)
N_matrix_r_Past_WhiteLaterals.append(N_matrix_r_Past_White)
N_matrix_r_Past_WhiteGenLaterals.append(N_matrix_r_Past_WhiteGen)
N_matrix_r_Past_BlueLaterals.append(N_matrix_r_Past_Blue)
N_matrix_r_Past_BlueGenLaterals.append(N_matrix_r_Past_BlueGen)
N_matrix_i_Past_RedLaterals.append(N_matrix_i_Past_Red)
N_matrix_i_Past_RedGenLaterals.append(N_matrix_i_Past_RedGen)
N_matrix_i_Past_WhiteLaterals.append(N_matrix_i_Past_White)
N_matrix_i_Past_WhiteGenLaterals.append(N_matrix_i_Past_WhiteGen)
N_matrix_i_Past_BlueLaterals.append(N_matrix_i_Past_Blue)
N_matrix_i_Past_BlueGenLaterals.append(N_matrix_i_Past_BlueGen)
N2_matrix_r_Past_RedLaterals.append(N2_matrix_r_Past_Red)
N2_matrix_r_Past_RedGenLaterals.append(N2_matrix_r_Past_RedGen)
N2_matrix_i_Past_RedLaterals.append(N2_matrix_i_Past_Red)
N2_matrix_i_Past_RedGenLaterals.append(N2_matrix_i_Past_RedGen)
N2_matrix_r_Past_WhiteLaterals.append(N2_matrix_r_Past_White)
```

```
            N2_matrix_r_Past_WhiteGenLaterals.append(N2_matrix_r_Past_WhiteGen)
            N2_matrix_i_Past_WhiteLaterals.append(N2_matrix_i_Past_White)
            N2_matrix_i_Past_WhiteGenLaterals.append(N2_matrix_i_Past_WhiteGen)
            N2_matrix_r_Past_BlueLaterals.append(N2_matrix_r_Past_Blue)
            N2_matrix_r_Past_BlueGenLaterals.append(N2_matrix_r_Past_BlueGen)
            N2_matrix_i_Past_BlueLaterals.append(N2_matrix_i_Past_Blue)
            N2_matrix_i_Past_BlueGenLaterals.append(N2_matrix_i_Past_BlueGen)
        else:
            if(len(N_matrix_r_Red)==1):
                pass
            elif(k==0):
                N_matrix_r_Past_Red.append(N_matrix_r_Red[0])
                N_matrix_r_Past_RedGen.append(N_matrix_r_RedGen[0])
                N_matrix_r_Past_White.append(N_matrix_r_White[0])
                N_matrix_r_Past_WhiteGen.append(N_matrix_r_WhiteGen[0])
                N_matrix_r_Past_Blue.append(N_matrix_r_Blue[0])
                N_matrix_r_Past_BlueGen.append(N_matrix_r_BlueGen[0])
                N_matrix_i_Past_Red.append(N_matrix_i_Red[0])
                N_matrix_i_Past_RedGen.append(N_matrix_i_RedGen[0])
                N_matrix_i_Past_White.append(N_matrix_i_White[0])
                N_matrix_i_Past_WhiteGen.append(N_matrix_i_WhiteGen[0])
                N_matrix_i_Past_Blue.append(N_matrix_i_Blue[0])
                N_matrix_i_Past_BlueGen.append(N_matrix_i_BlueGen[0])
                N2_matrix_r_Past_Red.append(N2_matrix_r_Red[0])
                N2_matrix_r_Past_RedGen.append(N2_matrix_r_RedGen[0])
                N2_matrix_i_Past_Red.append(N2_matrix_i_Red[0])
                N2_matrix_i_Past_RedGen.append(N2_matrix_i_RedGen[0])
                N2_matrix_r_Past_White.append(N2_matrix_r_White[0])
                N2_matrix_r_Past_WhiteGen.append(N2_matrix_r_WhiteGen[0])
                N2_matrix_i_Past_White.append(N2_matrix_i_White[0])
                N2_matrix_i_Past_WhiteGen.append(N2_matrix_i_WhiteGen[0])
                N2_matrix_r_Past_Blue.append(N2_matrix_r_Blue[0])
                N2_matrix_r_Past_BlueGen.append(N2_matrix_r_BlueGen[0])
                N2_matrix_i_Past_Blue.append(N2_matrix_i_Blue[0])
                N2_matrix_i_Past_BlueGen.append(N2_matrix_i_BlueGen[0])
    else:
        for k in range(1,int(len(Lats)/3)):
            if(i==(Data[0]-2)):
                N_matrix_r_Past_Red=copy.deepcopy(N_matrix_r_Red)
                N_matrix_r_Past_RedGen=copy.deepcopy(N_matrix_r_RedGen)
                N_matrix_r_Past_White=copy.deepcopy(N_matrix_r_White)
                N_matrix_r_Past_WhiteGen=copy.deepcopy(N_matrix_r_WhiteGen)
                N_matrix_r_Past_Blue=copy.deepcopy(N_matrix_r_Blue)
                N_matrix_r_Past_BlueGen=copy.deepcopy(N_matrix_r_BlueGen)
                N_matrix_i_Past_Red=copy.deepcopy(N_matrix_i_Red)
                N_matrix_i_Past_RedGen=copy.deepcopy(N_matrix_i_RedGen)
                N_matrix_i_Past_White=copy.deepcopy(N_matrix_i_White)
                N_matrix_i_Past_WhiteGen=copy.deepcopy(N_matrix_i_WhiteGen)
                N_matrix_i_Past_Blue=copy.deepcopy(N_matrix_i_Blue)
                N_matrix_i_Past_BlueGen=copy.deepcopy(N_matrix_i_BlueGen)
                N2_matrix_r_Past_Red=copy.deepcopy(N2_matrix_r_Red)
                N2_matrix_r_Past_RedGen=copy.deepcopy(N2_matrix_r_RedGen)
                N2_matrix_i_Past_Red=copy.deepcopy(N2_matrix_i_Red)
                N2_matrix_i_Past_RedGen=copy.deepcopy(N2_matrix_i_RedGen)
```

```
    N2_matrix_r_Past_White=copy.deepcopy(N2_matrix_r_White)
    N2_matrix_r_Past_WhiteGen=copy.deepcopy(N2_matrix_r_WhiteGen)
    N2_matrix_i_Past_White=copy.deepcopy(N2_matrix_i_White)
    N2_matrix_i_Past_WhiteGen=copy.deepcopy(N2_matrix_i_WhiteGen)
    N2_matrix_r_Past_Blue=copy.deepcopy(N2_matrix_r_Blue)
    N2_matrix_r_Past_BlueGen=copy.deepcopy(N2_matrix_r_BlueGen)
    N2_matrix_i_Past_Blue=copy.deepcopy(N2_matrix_i_Blue)
    N2_matrix_i_Past_BlueGen=copy.deepcopy(N2_matrix_i_BlueGen)
    for b in range(len(N_matrix_r_Past_Red)):
      if(b==(Data[0]-2)):
        pass
      elif((Data[5]-2-i)<=b<=(Data[6]-2-i)):
        N_matrix_r_Past_Red[b]=0
        N_matrix_r_Past_RedGen[b]=0
        N_matrix_r_Past_White[b]=0
        N_matrix_r_Past_WhiteGen[b]=0
        N_matrix_r_Past_Blue[b]=0
        N_matrix_r_Past_BlueGen[b]=0
        N_matrix_i_Past_Red[b]=0
        N_matrix_i_Past_RedGen[b]=0
        N_matrix_i_Past_White[b]=0
        N_matrix_i_Past_WhiteGen[b]=0
        N_matrix_i_Past_Blue[b]=0
        N_matrix_i_Past_BlueGen[b]=0
        N2_matrix_r_Past_Red[b]=0
        N2_matrix_r_Past_RedGen[b]=0
        N2_matrix_i_Past_Red[b]=0
        N2_matrix_i_Past_RedGen[b]=0
        N2_matrix_r_Past_White[b]=0
        N2_matrix_r_Past_WhiteGen[b]=0
        N2_matrix_i_Past_White[b]=0
        N2_matrix_i_Past_WhiteGen[b]=0
        N2_matrix_r_Past_Blue[b]=0
        N2_matrix_r_Past_BlueGen[b]=0
        N2_matrix_i_Past_Blue[b]=0
        N2_matrix_i_Past_BlueGen[b]=0
      else:
        pass
  elif(i<(Data[0]-2)):
    if(i==(Lats[3*k]-2)):
      N_matrix_r_Past_Red=copy.deepcopy(N_matrix_r_Past_RedLaterals[k])
      N_matrix_r_Past_RedGen=copy.deepcopy(N_matrix_r_Past_RedGenLaterals[k])
      N_matrix_r_Past_White=copy.deepcopy(N_matrix_r_Past_WhiteLaterals[k])
      N_matrix_r_Past_WhiteGen=copy.deepcopy(N_matrix_r_Past_WhiteGenLaterals[k])
      N_matrix_r_Past_Blue=copy.deepcopy(N_matrix_r_Past_BlueLaterals[k])
      N_matrix_r_Past_BlueGen=copy.deepcopy(N_matrix_r_Past_BlueGenLaterals[k])
      N_matrix_i_Past_Red=copy.deepcopy(N_matrix_i_Past_RedLaterals[k])
      N_matrix_i_Past_RedGen=copy.deepcopy(N_matrix_i_Past_RedGenLaterals[k])
      N_matrix_i_Past_White=copy.deepcopy(N_matrix_i_Past_WhiteLaterals[k])
      N_matrix_i_Past_WhiteGen=copy.deepcopy(N_matrix_i_Past_WhiteGenLaterals[k])
      N_matrix_i_Past_Blue=copy.deepcopy(N_matrix_i_Past_BlueLaterals[k])
      N_matrix_i_Past_BlueGen=copy.deepcopy(N_matrix_i_Past_BlueGenLaterals[k])
      N2_matrix_r_Past_Red=copy.deepcopy(N2_matrix_r_Past_RedLaterals[k])
      N2_matrix_r_Past_RedGen=copy.deepcopy(N2_matrix_r_Past_RedGenLaterals[k])
```

```
        N2_matrix_i_Past_Red=copy.deepcopy(N2_matrix_i_Past_RedLaterals[k])
        N2_matrix_i_Past_RedGen=copy.deepcopy(N2_matrix_i_Past_RedGenLaterals[k])
        N2_matrix_r_Past_White=copy.deepcopy(N2_matrix_r_Past_WhiteLaterals[k])
        N2_matrix_r_Past_WhiteGen=copy.deepcopy(N2_matrix_r_Past_WhiteGenLaterals[k])
        N2_matrix_i_Past_White=copy.deepcopy(N2_matrix_i_Past_WhiteLaterals[k])
        N2_matrix_i_Past_WhiteGen=copy.deepcopy(N2_matrix_i_Past_WhiteGenLaterals[k])
        N2_matrix_r_Past_Blue=copy.deepcopy(N2_matrix_r_Past_BlueLaterals[k])
        N2_matrix_r_Past_BlueGen=copy.deepcopy(N2_matrix_r_Past_BlueGenLaterals[k])
        N2_matrix_i_Past_Blue=copy.deepcopy(N2_matrix_i_Past_BlueLaterals[k])
        N2_matrix_i_Past_BlueGen=copy.deepcopy(N2_matrix_i_Past_BlueGenLaterals[k])
    else:
        if(k==1):
            N_matrix_r_Past_Red.append(N_matrix_r_Red[0])
            N_matrix_r_Past_RedGen.append(N_matrix_r_RedGen[0])
            N_matrix_r_Past_White.append(N_matrix_r_White[0])
            N_matrix_r_Past_WhiteGen.append(N_matrix_r_WhiteGen[0])
            N_matrix_r_Past_Blue.append(N_matrix_r_Blue[0])
            N_matrix_r_Past_BlueGen.append(N_matrix_r_BlueGen[0])
            N_matrix_i_Past_Red.append(N_matrix_i_Red[0])
            N_matrix_i_Past_RedGen.append(N_matrix_i_RedGen[0])
            N_matrix_i_Past_White.append(N_matrix_i_White[0])
            N_matrix_i_Past_WhiteGen.append(N_matrix_i_WhiteGen[0])
            N_matrix_i_Past_Blue.append(N_matrix_i_Blue[0])
            N_matrix_i_Past_BlueGen.append(N_matrix_i_BlueGen[0])
            N2_matrix_r_Past_Red.append(N2_matrix_r_Red[0])
            N2_matrix_r_Past_RedGen.append(N2_matrix_r_RedGen[0])
            N2_matrix_i_Past_Red.append(N2_matrix_i_Red[0])
            N2_matrix_i_Past_RedGen.append(N2_matrix_i_RedGen[0])
            N2_matrix_r_Past_White.append(N2_matrix_r_White[0])
            N2_matrix_r_Past_WhiteGen.append(N2_matrix_r_WhiteGen[0])
            N2_matrix_i_Past_White.append(N2_matrix_i_White[0])
            N2_matrix_i_Past_WhiteGen.append(N2_matrix_i_WhiteGen[0])
            N2_matrix_r_Past_Blue.append(N2_matrix_r_Blue[0])
            N2_matrix_r_Past_BlueGen.append(N2_matrix_r_BlueGen[0])
            N2_matrix_i_Past_Blue.append(N2_matrix_i_Blue[0])
            N2_matrix_i_Past_BlueGen.append(N2_matrix_i_BlueGen[0])
else:
    if(len(N_matrix_r_Red)==1):
        pass
    elif(k==1):
        N_matrix_r_Past_Red.append(N_matrix_r_Red[0])
        N_matrix_r_Past_RedGen.append(N_matrix_r_RedGen[0])
        N_matrix_r_Past_White.append(N_matrix_r_White[0])
        N_matrix_r_Past_WhiteGen.append(N_matrix_r_WhiteGen[0])
        N_matrix_r_Past_Blue.append(N_matrix_r_Blue[0])
        N_matrix_r_Past_BlueGen.append(N_matrix_r_BlueGen[0])
        N_matrix_i_Past_Red.append(N_matrix_i_Red[0])
        N_matrix_i_Past_RedGen.append(N_matrix_i_RedGen[0])
        N_matrix_i_Past_White.append(N_matrix_i_White[0])
        N_matrix_i_Past_WhiteGen.append(N_matrix_i_WhiteGen[0])
        N_matrix_i_Past_Blue.append(N_matrix_i_Blue[0])
        N_matrix_i_Past_BlueGen.append(N_matrix_i_BlueGen[0])
        N2_matrix_r_Past_Red.append(N2_matrix_r_Red[0])
        N2_matrix_r_Past_RedGen.append(N2_matrix_r_RedGen[0])
```

```
                    N2_matrix_i_Past_Red.append(N2_matrix_i_Red[0])
                    N2_matrix_i_Past_RedGen.append(N2_matrix_i_RedGen[0])
                    N2_matrix_r_Past_White.append(N2_matrix_r_White[0])
                    N2_matrix_r_Past_WhiteGen.append(N2_matrix_r_WhiteGen[0])
                    N2_matrix_i_Past_White.append(N2_matrix_i_White[0])
                    N2_matrix_i_Past_WhiteGen.append(N2_matrix_i_WhiteGen[0])
                    N2_matrix_r_Past_Blue.append(N2_matrix_r_Blue[0])
                    N2_matrix_r_Past_BlueGen.append(N2_matrix_r_BlueGen[0])
                    N2_matrix_i_Past_Blue.append(N2_matrix_i_Blue[0])
                    N2_matrix_i_Past_BlueGen.append(N2_matrix_i_BlueGen[0])

                N_matrix_r_Past_RedFinal.append(N_matrix_r_Past_Red)
                N_matrix_r_Past_RedGenFinal.append(N_matrix_r_Past_RedGen)
                N_matrix_r_Past_WhiteFinal.append(N_matrix_r_Past_White)
                N_matrix_r_Past_WhiteGenFinal.append(N_matrix_r_Past_WhiteGen)
                N_matrix_r_Past_BlueFinal.append(N_matrix_r_Past_Blue)
                N_matrix_r_Past_BlueGenFinal.append(N_matrix_r_Past_BlueGen)
                N_matrix_i_Past_RedFinal.append(N_matrix_i_Past_Red)
                N_matrix_i_Past_RedGenFinal.append(N_matrix_i_Past_RedGen)
                N_matrix_i_Past_WhiteFinal.append(N_matrix_i_Past_White)
                N_matrix_i_Past_WhiteGenFinal.append(N_matrix_i_Past_WhiteGen)
                N_matrix_i_Past_BlueFinal.append(N_matrix_i_Past_Blue)
                N_matrix_i_Past_BlueGenFinal.append(N_matrix_i_Past_BlueGen)
                N2_matrix_r_Past_RedFinal.append(N2_matrix_r_Past_Red)
                N2_matrix_r_Past_RedGenFinal.append(N2_matrix_r_Past_RedGen)
                N2_matrix_i_Past_RedFinal.append(N2_matrix_i_Past_Red)
                N2_matrix_i_Past_RedGenFinal.append(N2_matrix_i_Past_RedGen)
                N2_matrix_r_Past_WhiteFinal.append(N2_matrix_r_Past_White)
                N2_matrix_r_Past_WhiteGenFinal.append(N2_matrix_r_Past_WhiteGen)
                N2_matrix_i_Past_WhiteFinal.append(N2_matrix_i_Past_White)
                N2_matrix_i_Past_WhiteGenFinal.append(N2_matrix_i_Past_WhiteGen)
                N2_matrix_r_Past_BlueFinal.append(N2_matrix_r_Past_Blue)
                N2_matrix_r_Past_BlueGenFinal.append(N2_matrix_r_Past_BlueGen)
                N2_matrix_i_Past_BlueFinal.append(N2_matrix_i_Past_Blue)
                N2_matrix_i_Past_BlueGenFinal.append(N2_matrix_i_Past_BlueGen)
                for u in range(len(N_matrix_r_Past_RedFinal[len(N_matrix_r_Past_RedFinal)-1])):
                N_matrix_r_Past_RedAll.append(N_matrix_r_Past_RedFinal[len
               (N_matrix_r_Past_RedFinal)-1][u])
               N_matrix_r_Past_RedGenAll.append(N_matrix_r_Past_RedGenFinal[len
               (N_matrix_r_Past_RedFinal)-1][u])
                    N_matrix_r_Past_WhiteAll.append(N_matrix_r_Past_WhiteFinal[len(N_matrix_r_Past_RedFinal)-
1][u])
N_matrix_r_Past_WhiteGenAll.append(N_matrix_r_Past_WhiteGenFinal[len(N_matrix_r_Past_RedFinal)-1][u])
                    N_matrix_r_Past_BlueAll.append(N_matrix_r_Past_BlueFinal[len(N_matrix_r_Past_RedFinal)-
1][u])
N_matrix_r_Past_BlueGenAll.append(N_matrix_r_Past_BlueGenFinal[len(N_matrix_r_Past_RedFinal)-1][u])
                    N_matrix_i_Past_RedAll.append(N_matrix_i_Past_RedFinal[len(N_matrix_r_Past_RedFinal)-1][u])
N_matrix_i_Past_RedGenAll.append(N_matrix_i_Past_RedGenFinal[len(N_matrix_r_Past_RedFinal)-1][u])
                    N_matrix_i_Past_WhiteAll.append(N_matrix_i_Past_WhiteFinal[len(N_matrix_r_Past_RedFinal)-
1][u])
N_matrix_i_Past_WhiteGenAll.append(N_matrix_i_Past_WhiteGenFinal[len(N_matrix_r_Past_RedFinal)-1][u])
                    N_matrix_i_Past_BlueAll.append(N_matrix_i_Past_BlueFinal[len(N_matrix_r_Past_RedFinal)-
1][u])
```

```
N_matrix_i_Past_BlueGenAll.append(N_matrix_i_Past_BlueGenFinal[len(N_matrix_r_Past_RedFinal)-1][u])
                    N2_matrix_r_Past_RedAll.append(N2_matrix_r_Past_RedFinal[len(N_matrix_r_Past_RedFinal)-
1][u])
N2_matrix_r_Past_RedGenAll.append(N2_matrix_r_Past_RedGenFinal[len(N_matrix_r_Past_RedFinal)-1][u])
                    N2_matrix_i_Past_RedAll.append(N2_matrix_i_Past_RedFinal[len(N_matrix_r_Past_RedFinal)-
1][u])
N2_matrix_i_Past_RedGenAll.append(N2_matrix_i_Past_RedGenFinal[len(N_matrix_r_Past_RedFinal)-1][u])
N2_matrix_r_Past_WhiteAll.append(N2_matrix_r_Past_WhiteFinal[len(N_matrix_r_Past_RedFinal)-1][u])
N2_matrix_r_Past_WhiteGenAll.append(N2_matrix_r_Past_WhiteGenFinal[len(N_matrix_r_Past_RedFinal)-1][u])
N2_matrix_i_Past_WhiteAll.append(N2_matrix_i_Past_WhiteFinal[len(N_matrix_r_Past_RedFinal)-1][u])
N2_matrix_i_Past_WhiteGenAll.append(N2_matrix_i_Past_WhiteGenFinal[len(N_matrix_r_Past_RedFinal)-1][u])
                    N2_matrix_r_Past_BlueAll.append(N2_matrix_r_Past_BlueFinal[len(N_matrix_r_Past_RedFinal)-
1][u])
N2_matrix_r_Past_BlueGenAll.append(N2_matrix_r_Past_BlueGenFinal[len(N_matrix_r_Past_RedFinal)-1][u])
                    N2_matrix_i_Past_BlueAll.append(N2_matrix_i_Past_BlueFinal[len(N_matrix_r_Past_RedFinal)-
1][u])
N2_matrix_i_Past_BlueGenAll.append(N2_matrix_i_Past_BlueGenFinal[len(N_matrix_r_Past_RedFinal)-1][u])
                # print('N2_matrix_r_Past_Red =',N2_matrix_r_Past_Red)
                # print('N2_matrix_r_Past_RedAll =',N2_matrix_r_Past_RedAll)
                # print('N_matrix_r_Past_RedSum =',N_matrix_r_Past_RedSum)
                # print('N2_matrix_r_Past_RedSum =',N2_matrix_r_Past_RedSum)
                if(len(Data)!=len(Lats) and i<=(Data[0]-2)):
                    for k in range(1,int(len(Lats)/3)):
                        if(i==(Lats[3*k]-2)):
                            o=k-1
                            print('o =',o)
FirstMoment_Vr_RedLumped.append(sum(N_matrix_r_RedLaterals[o][:len(N_matrix_r_RedLaterals[o])]))
FirstMoment_Vr_Gen_RedLumped.append(sum(N_matrix_r_RedGenLaterals[o][:(len(N_matrix_r_RedLaterals[o]
))]))
FirstMoment_Vi_RedLumped.append(sum(N_matrix_i_RedLaterals[o][:(len(N_matrix_r_RedLaterals[o]))]))
FirstMoment_Vi_Gen_RedLumped.append(sum(N_matrix_i_RedGenLaterals[o][:(len(N_matrix_r_RedLaterals[o])
)]))
SecondMoment_Vr_RedLumped.append(sum(N2_matrix_r_RedLaterals[o][:len(N2_matrix_r_RedLaterals[o])])+
B1Red[i])
SecondMoment_Vr_Gen_RedLumped.append(sum(N2_matrix_r_RedGenLaterals[o][:len(N2_matrix_r_RedGenLat
erals[o])])+B1RedGen[i])
SecondMoment_Vi_RedLumped.append(sum(N2_matrix_i_RedLaterals[o][:len(N2_matrix_i_RedLaterals[o])])+B
2Red[i])
SecondMoment_Vi_Gen_RedLumped.append(sum(N2_matrix_i_RedGenLaterals[o][:len(N2_matrix_i_RedGenLate
rals[o])])+B2RedGen[i])
FirstMoment_Vr_WhiteLumped.append(sum(N_matrix_r_WhiteLaterals[o][:len(N_matrix_r_WhiteLaterals[o])]))
FirstMoment_Vr_Gen_WhiteLumped.append(sum(N_matrix_r_WhiteGenLaterals[o][:(len(N_matrix_r_WhiteLater
als[o]))]))
FirstMoment_Vi_WhiteLumped.append(sum(N_matrix_i_WhiteLaterals[o][:(len(N_matrix_r_WhiteLaterals[o]))])
)
FirstMoment_Vi_Gen_WhiteLumped.append(sum(N_matrix_i_WhiteGenLaterals[o][:(len(N_matrix_r_WhiteLater
als[o]))]))
SecondMoment_Vr_WhiteLumped.append(sum(N2_matrix_r_WhiteLaterals[o][:len(N2_matrix_r_WhiteLaterals[
o])])+B1White[i])
SecondMoment_Vr_Gen_WhiteLumped.append(sum(N2_matrix_r_WhiteGenLaterals[o][:len(N2_matrix_r_White
GenLaterals[o])])+B1WhiteGen[i])
SecondMoment_Vi_WhiteLumped.append(sum(N2_matrix_i_WhiteLaterals[o][:len(N2_matrix_i_WhiteLaterals[o
])])+B2White[i])
```

```
SecondMoment_Vi_Gen_WhiteLumped.append(sum(N2_matrix_i_WhiteGenLaterals[o][:len(N2_matrix_i_WhiteGenLaterals[o])])+B2WhiteGen[i])
FirstMoment_Vr_BlueLumped.append(sum(N_matrix_r_BlueLaterals[o][:len(N_matrix_r_BlueLaterals[o])]))
FirstMoment_Vr_Gen_BlueLumped.append(sum(N_matrix_r_BlueGenLaterals[o][:(len(N_matrix_r_BlueLaterals[o]))]))
FirstMoment_Vi_BlueLumped.append(sum(N_matrix_i_BlueLaterals[o][:(len(N_matrix_r_BlueLaterals[o]))]))
FirstMoment_Vi_Gen_BlueLumped.append(sum(N_matrix_i_BlueGenLaterals[o][:(len(N_matrix_r_BlueLaterals[o]))]))
SecondMoment_Vr_BlueLumped.append(sum(N2_matrix_r_BlueLaterals[o][:len(N2_matrix_r_BlueLaterals[o])])+B1Blue[i])
SecondMoment_Vr_Gen_BlueLumped.append(sum(N2_matrix_r_BlueGenLaterals[o][:len(N2_matrix_r_BlueGenLaterals[o])])+B1BlueGen[i])
SecondMoment_Vi_BlueLumped.append(sum(N2_matrix_i_BlueLaterals[o][:len(N2_matrix_i_BlueLaterals[o])])+B2Blue[i])
SecondMoment_Vi_Gen_BlueLumped.append(sum(N2_matrix_i_BlueGenLaterals[o][:len(N2_matrix_i_BlueGenLaterals[o])])+B2BlueGen[i])
                        # print('N_matrix_r_RedLaterals =',N_matrix_r_RedLaterals)
                        # print('N_matrix_r_WhiteLaterals =',N_matrix_r_WhiteLaterals)
                        # print('N_matrix_r_BlueLaterals =',N_matrix_r_BlueLaterals)
                        break
                    else:
                        if(k==(int(len(Lats)/3)-1)):
FirstMoment_Vr_RedLumped.append(sum(N_matrix_r_Pre_RedLaterals[i][:(len(N_matrix_r_Pre_RedLaterals[i]))]))
FirstMoment_Vr_Gen_RedLumped.append(sum(N_matrix_r_Pre_RedGenLaterals[i][:(len(N_matrix_r_Pre_RedLaterals[i]))]))
FirstMoment_Vi_RedLumped.append(sum(N_matrix_i_Pre_RedLaterals[i][:(len(N_matrix_r_Pre_RedLaterals[i]))]))
FirstMoment_Vi_Gen_RedLumped.append(sum(N_matrix_i_Pre_RedGenLaterals[i][:(len(N_matrix_r_Pre_RedLaterals[i]))]))
SecondMoment_Vr_RedLumped.append(sum(N2_matrix_r_Pre_RedLaterals[i][:len(N2_matrix_r_Pre_RedLaterals[i])])+B1Red[i])
SecondMoment_Vr_Gen_RedLumped.append(sum(N2_matrix_r_Pre_RedGenLaterals[i][:len(N2_matrix_r_Pre_RedGenLaterals[i])])+B1RedGen[i])
SecondMoment_Vi_RedLumped.append(sum(N2_matrix_i_Pre_RedLaterals[i][:len(N2_matrix_i_Pre_RedLaterals[i])])+B2Red[i])
SecondMoment_Vi_Gen_RedLumped.append(sum(N2_matrix_i_Pre_RedGenLaterals[i][:len(N2_matrix_i_Pre_RedGenLaterals[i])])+B2RedGen[i])
FirstMoment_Vr_WhiteLumped.append(sum(N_matrix_r_Pre_WhiteLaterals[i][:(len(N_matrix_r_Pre_WhiteLaterals[i]))]))
FirstMoment_Vr_Gen_WhiteLumped.append(sum(N_matrix_r_Pre_WhiteGenLaterals[i][:(len(N_matrix_r_Pre_WhiteLaterals[i]))]))
FirstMoment_Vi_WhiteLumped.append(sum(N_matrix_i_Pre_WhiteLaterals[i][:(len(N_matrix_r_Pre_WhiteLaterals[i]))]))
FirstMoment_Vi_Gen_WhiteLumped.append(sum(N_matrix_i_Pre_WhiteGenLaterals[i][:(len(N_matrix_r_Pre_WhiteLaterals[i]))]))
SecondMoment_Vr_WhiteLumped.append(sum(N2_matrix_r_Pre_WhiteLaterals[i][:len(N2_matrix_r_Pre_WhiteLaterals[i])])+B1White[i])
SecondMoment_Vr_Gen_WhiteLumped.append(sum(N2_matrix_r_Pre_WhiteGenLaterals[i][:len(N2_matrix_r_Pre_WhiteGenLaterals[i])])+B1WhiteGen[i])
SecondMoment_Vi_WhiteLumped.append(sum(N2_matrix_i_Pre_WhiteLaterals[i][:len(N2_matrix_i_Pre_WhiteLaterals[i])])+B2White[i])
SecondMoment_Vi_Gen_WhiteLumped.append(sum(N2_matrix_i_Pre_WhiteGenLaterals[i][:len(N2_matrix_i_Pre_WhiteGenLaterals[i])])+B2WhiteGen[i])
```

```
FirstMoment_Vr_BlueLumped.append(sum(N_matrix_r_Pre_BlueLaterals[i][:(len(N_matrix_r_Pre_BlueLaterals[i]
))]))
FirstMoment_Vr_Gen_BlueLumped.append(sum(N_matrix_r_Pre_BlueGenLaterals[i][:(len(N_matrix_r_Pre_BlueL
aterals[i]))]))
FirstMoment_Vi_BlueLumped.append(sum(N_matrix_i_Pre_BlueLaterals[i][:(len(N_matrix_r_Pre_BlueLaterals[i]
))]))
FirstMoment_Vi_Gen_BlueLumped.append(sum(N_matrix_i_Pre_BlueGenLaterals[i][:(len(N_matrix_r_Pre_BlueLa
terals[i]))]))
SecondMoment_Vr_BlueLumped.append(sum(N2_matrix_r_Pre_BlueLaterals[i][:len(N2_matrix_r_Pre_BlueLater
als[i])])+B1Blue[i])
SecondMoment_Vr_Gen_BlueLumped.append(sum(N2_matrix_r_Pre_BlueGenLaterals[i][:len(N2_matrix_r_Pre_Bl
ueGenLaterals[i])])+B1BlueGen[i])
SecondMoment_Vi_BlueLumped.append(sum(N2_matrix_i_Pre_BlueLaterals[i][:len(N2_matrix_i_Pre_BlueLateral
s[i])])+B2Blue[i])
SecondMoment_Vi_Gen_BlueLumped.append(sum(N2_matrix_i_Pre_BlueGenLaterals[i][:len(N2_matrix_i_Pre_Bl
ueGenLaterals[i])])+B2BlueGen[i])
                        # print('N_matrix_r_Pre_RedLaterals=',N_matrix_r_Pre_RedLaterals)
                        # print('N_matrix_r_Pre_WhiteLaterals=',N_matrix_r_Pre_WhiteLaterals)
                        # print('N_matrix_r_Pre_BlueLaterals=',N_matrix_r_Pre_BlueLaterals)
                else:
                    FirstMoment_Vr_RedLumped.append(sum(N_matrix_r_Red[:(len(Iteration)-i)]))
                    FirstMoment_Vr_Gen_RedLumped.append(sum(N_matrix_r_RedGen[:(len(Iteration)-i)]))
                    FirstMoment_Vi_RedLumped.append(sum(N_matrix_i_Red[:(len(Iteration)-i)]))
                    FirstMoment_Vi_Gen_RedLumped.append(sum(N_matrix_i_RedGen[:(len(Iteration)-i)]))
                    SecondMoment_Vr_RedLumped.append(sum(N2_matrix_r_Red[:(len(Iteration)-i)])+B1Red[i])
                    SecondMoment_Vr_Gen_RedLumped.append(sum(N2_matrix_r_RedGen[:(len(Iteration)-
i)])+B1RedGen[i])
                    SecondMoment_Vi_RedLumped.append(sum(N2_matrix_i_Red[:(len(Iteration)-i)])+B2Red[i])
                    SecondMoment_Vi_Gen_RedLumped.append(sum(N2_matrix_i_RedGen[:(len(Iteration)-
i)])+B2RedGen[i])
                    FirstMoment_Vr_WhiteLumped.append(sum(N_matrix_r_White[:(len(Iteration)-i)]))
                    FirstMoment_Vr_Gen_WhiteLumped.append(sum(N_matrix_r_WhiteGen[:(len(Iteration)-i)]))
                    FirstMoment_Vi_WhiteLumped.append(sum(N_matrix_i_White[:(len(Iteration)-i)]))
                    FirstMoment_Vi_Gen_WhiteLumped.append(sum(N_matrix_i_WhiteGen[:(len(Iteration)-i)]))
                    SecondMoment_Vr_WhiteLumped.append(sum(N2_matrix_r_White[:(len(Iteration)-
i)])+B1White[i])
                    SecondMoment_Vr_Gen_WhiteLumped.append(sum(N2_matrix_r_WhiteGen[:(len(Iteration)-
i)])+B1WhiteGen[i])
                    SecondMoment_Vi_WhiteLumped.append(sum(N2_matrix_i_White[:(len(Iteration)-
i)])+B2White[i])
                    SecondMoment_Vi_Gen_WhiteLumped.append(sum(N2_matrix_i_WhiteGen[:(len(Iteration)-
i)])+B2WhiteGen[i])
                    FirstMoment_Vr_BlueLumped.append(sum(N_matrix_r_Blue[:(len(Iteration)-i)]))
                    FirstMoment_Vr_Gen_BlueLumped.append(sum(N_matrix_r_BlueGen[:(len(Iteration)-i)]))
                    FirstMoment_Vi_BlueLumped.append(sum(N_matrix_i_Blue[:(len(Iteration)-i)]))
                    FirstMoment_Vi_Gen_BlueLumped.append(sum(N_matrix_i_BlueGen[:(len(Iteration)-i)]))
                    SecondMoment_Vr_BlueLumped.append(sum(N2_matrix_r_Blue[:(len(Iteration)-i)])+B1Blue[i])
                    SecondMoment_Vr_Gen_BlueLumped.append(sum(N2_matrix_r_BlueGen[:(len(Iteration)-
i)])+B1BlueGen[i])
                    SecondMoment_Vi_BlueLumped.append(sum(N2_matrix_i_Blue[:(len(Iteration)-i)])+B2Blue[i])
                    SecondMoment_Vi_Gen_BlueLumped.append(sum(N2_matrix_i_BlueGen[:(len(Iteration)-
i)])+B2BlueGen[i])
                del(FirstMoment_Vr_Red[:(n-i)])
                del(FirstMoment_Vr_Gen_Red[:(n-i)])
```

```
del(SecondMoment_Vr_Red[:(n-i)])
del(SecondMoment_Vr_Gen_Red[:(n-i)])
del(FirstMoment_Vi_Red[:(n-i)])
del(FirstMoment_Vi_Gen_Red[:(n-i)])
del(SecondMoment_Vi_Red[:(n-i)])
del(SecondMoment_Vi_Gen_Red[:(n-i)])
del(FirstMoment_Vr_White[:(n-i)])
del(FirstMoment_Vr_Gen_White[:(n-i)])
del(SecondMoment_Vr_White[:(n-i)])
del(SecondMoment_Vr_Gen_White[:(n-i)])
del(FirstMoment_Vi_White[:(n-i)])
del(FirstMoment_Vi_Gen_White[:(n-i)])
del(SecondMoment_Vi_White[:(n-i)])
del(SecondMoment_Vi_Gen_White[:(n-i)])
del(FirstMoment_Vr_Blue[:(n-i)])
del(FirstMoment_Vr_Gen_Blue[:(n-i)])
del(SecondMoment_Vr_Blue[:(n-i)])
del(SecondMoment_Vr_Gen_Blue[:(n-i)])
del(FirstMoment_Vi_Blue[:(n-i)])
del(FirstMoment_Vi_Gen_Blue[:(n-i)])
del(SecondMoment_Vi_Blue[:(n-i)])
del(SecondMoment_Vi_Gen_Blue[:(n-i)])
B1TotalTempRed=[]
B2TotalTempRed=[]
B1TotalTempWhite=[]
B2TotalTempWhite=[]
B1TotalTempBlue=[]
B2TotalTempBlue=[]
N_matrix_r_Red=[]
N_matrix_r_White=[]
N_matrix_r_Blue=[]
N_matrix_i_Red=[]
N_matrix_i_White=[]
N_matrix_i_Blue=[]
N_matrix_r_RedGen=[]
N_matrix_i_RedGen=[]
N_matrix_r_WhiteGen=[]
N_matrix_i_WhiteGen=[]
N_matrix_r_BlueGen=[]
N_matrix_i_BlueGen=[]
N2_matrix_r_Red=[]
N2_matrix_r_RedGen=[]
N2_matrix_i_Red=[]
N2_matrix_i_RedGen=[]
N2_matrix_r_White=[]
N2_matrix_r_WhiteGen=[]
N2_matrix_i_White=[]
N2_matrix_i_WhiteGen=[]
N2_matrix_r_Blue=[]
N2_matrix_r_BlueGen=[]
N2_matrix_i_Blue=[]
N2_matrix_i_BlueGen=[]
#Determine the sum of moments at each node
if(i==0):
```

```
FirstMoment_Vr_Red_Sum.append(sum(FirstMoment_Vr_RedLumped)+
sum(FirstMoment_Vr_Gen_RedLumped))
FirstMoment_Vi_Red_Sum.append(sum(FirstMoment_Vi_RedLumped)+sum(FirstMoment_Vi_Gen_RedLumped))
SecondMoment_Vr_Red_Sum.append(sum(SecondMoment_Vr_RedLumped)+
sum(SecondMoment_Vr_Gen_RedLumped))
                SecondMoment_Vi_Red_Sum.append(sum(SecondMoment_Vi_RedLumped)+
sum(SecondMoment_Vi_Gen_RedLumped))
                FirstMoment_Vr_White_Sum.append(sum(FirstMoment_Vr_WhiteLumped)+
sum(FirstMoment_Vr_Gen_WhiteLumped))
                FirstMoment_Vi_White_Sum.append(sum(FirstMoment_Vi_WhiteLumped)+
sum(FirstMoment_Vi_Gen_WhiteLumped))
                SecondMoment_Vr_White_Sum.append(sum(SecondMoment_Vr_WhiteLumped)+
sum(SecondMoment_Vr_Gen_WhiteLumped))
                SecondMoment_Vi_White_Sum.append(sum(SecondMoment_Vi_WhiteLumped)+
sum(SecondMoment_Vi_Gen_WhiteLumped))
                FirstMoment_Vr_Blue_Sum.append(sum(FirstMoment_Vr_BlueLumped)+
sum(FirstMoment_Vr_Gen_BlueLumped))
FirstMoment_Vi_Blue_Sum.append(sum(FirstMoment_Vi_BlueLumped)+
sum(FirstMoment_Vi_Gen_BlueLumped))
SecondMoment_Vr_Blue_Sum.append(sum(SecondMoment_Vr_BlueLumped)+
sum(SecondMoment_Vr_Gen_BlueLumped))
SecondMoment_Vi_Blue_Sum.append(sum(SecondMoment_Vi_BlueLumped)+
sum(SecondMoment_Vi_Gen_BlueLumped))
            else:
FirstMoment_Vr_Red_Sum.append(FirstMoment_Vr_RedLumped[i]+FirstMoment_Vr_Gen_RedLumped[i]+N_mat
rix_r_Past_RedSum+N_matrix_r_Past_RedGenSum)
FirstMoment_Vi_Red_Sum.append(FirstMoment_Vi_RedLumped[i]+FirstMoment_Vi_Gen_RedLumped[i]+N_matr
ix_i_Past_RedSum+N_matrix_i_Past_RedGenSum)
SecondMoment_Vr_Red_Sum.append(SecondMoment_Vr_RedLumped[i]+SecondMoment_Vr_Gen_RedLumped[i]
+N2_matrix_r_Past_RedSum+N2_matrix_r_Past_RedGenSum+sum(B1RedPast))
SecondMoment_Vi_Red_Sum.append(SecondMoment_Vi_RedLumped[i]+SecondMoment_Vi_Gen_RedLumped[i]+
N2_matrix_i_Past_RedSum+N2_matrix_i_Past_RedGenSum)
FirstMoment_Vr_White_Sum.append(FirstMoment_Vr_WhiteLumped[i]+FirstMoment_Vr_Gen_WhiteLumped[i]+
N_matrix_r_Past_WhiteSum+N_matrix_r_Past_WhiteGenSum)
FirstMoment_Vi_White_Sum.append(FirstMoment_Vi_WhiteLumped[i]+FirstMoment_Vi_Gen_WhiteLumped[i]+
N_matrix_i_Past_WhiteSum+N_matrix_i_Past_WhiteGenSum)
SecondMoment_Vr_White_Sum.append(SecondMoment_Vr_WhiteLumped[i]+SecondMoment_Vr_Gen_WhiteLum
ped[i]+N2_matrix_r_Past_WhiteSum+N2_matrix_r_Past_WhiteGenSum)
SecondMoment_Vi_White_Sum.append(SecondMoment_Vi_WhiteLumped[i]+SecondMoment_Vi_Gen_WhiteLum
ped[i]+N2_matrix_i_Past_WhiteSum+N2_matrix_i_Past_WhiteGenSum)
FirstMoment_Vr_Blue_Sum.append(FirstMoment_Vr_BlueLumped[i]+FirstMoment_Vr_Gen_BlueLumped[i]+N_m
atrix_r_Past_BlueSum+N_matrix_r_Past_BlueGenSum)
FirstMoment_Vi_Blue_Sum.append(FirstMoment_Vi_BlueLumped[i]+FirstMoment_Vi_Gen_BlueLumped[i]+N_ma
trix_i_Past_BlueSum+N_matrix_i_Past_BlueGenSum)
SecondMoment_Vr_Blue_Sum.append(SecondMoment_Vr_BlueLumped[i]+SecondMoment_Vr_Gen_BlueLumped[
i]+N2_matrix_r_Past_BlueSum+N2_matrix_r_Past_BlueGenSum)
SecondMoment_Vi_Blue_Sum.append(SecondMoment_Vi_BlueLumped[i]+SecondMoment_Vi_Gen_BlueLumped[i
]+N2_matrix_i_Past_BlueSum+N2_matrix_i_Past_BlueGenSum)
                N_matrix_r_Past_RedSum=sum(N_matrix_r_Past_RedAll)
                N_matrix_r_Past_RedGenSum=sum(N_matrix_r_Past_RedGenAll)
                N_matrix_r_Past_WhiteSum=sum(N_matrix_r_Past_WhiteAll)
                N_matrix_r_Past_WhiteGenSum=sum(N_matrix_r_Past_WhiteGenAll)
                N_matrix_r_Past_BlueSum=sum(N_matrix_r_Past_BlueAll)
                N_matrix_r_Past_BlueGenSum=sum(N_matrix_r_Past_BlueGenAll)
```

```
        N_matrix_i_Past_RedSum=sum(N_matrix_i_Past_RedAll)
        N_matrix_i_Past_RedGenSum=sum(N_matrix_i_Past_RedGenAll)
        N_matrix_i_Past_WhiteSum=sum(N_matrix_i_Past_WhiteAll)
        N_matrix_i_Past_WhiteGenSum=sum(N_matrix_i_Past_WhiteGenAll)
        N_matrix_i_Past_BlueSum=sum(N_matrix_i_Past_BlueAll)
        N_matrix_i_Past_BlueGenSum=sum(N_matrix_i_Past_BlueGenAll)
        N2_matrix_r_Past_RedSum=sum(N2_matrix_r_Past_RedAll)
        N2_matrix_r_Past_RedGenSum=sum(N2_matrix_r_Past_RedGenAll)
        N2_matrix_i_Past_RedSum=sum(N2_matrix_i_Past_RedAll)
        N2_matrix_i_Past_RedGenSum=sum(N2_matrix_i_Past_RedGenAll)
        N2_matrix_r_Past_WhiteSum=sum(N2_matrix_r_Past_WhiteAll)
        N2_matrix_r_Past_WhiteGenSum=sum(N2_matrix_r_Past_WhiteGenAll)
        N2_matrix_i_Past_WhiteSum=sum(N2_matrix_i_Past_WhiteAll)
        N2_matrix_i_Past_WhiteGenSum=sum(N2_matrix_i_Past_WhiteGenAll)
        N2_matrix_r_Past_BlueSum=sum(N2_matrix_r_Past_BlueAll)
        N2_matrix_r_Past_BlueGenSum=sum(N2_matrix_r_Past_BlueGenAll)
        N2_matrix_i_Past_BlueSum=sum(N2_matrix_i_Past_BlueAll)
        N2_matrix_i_Past_BlueGenSum=sum(N2_matrix_i_Past_BlueGenAll)
    #Sorts the sum of moment values in numerical order, taking into account laterals
    if(len(FirstMoment_Vr_Red_Sum)==len(Iteration)):
      if(len(Data)==len(Lats)):
        for e in range(Data[0]-2,(Data[0]-2)+Data[6]-Data[5]+2):
          FirstMoment_Vr_Red_SumFeeder.append(FirstMoment_Vr_Red_Sum[e])
          FirstMoment_Vr_White_SumFeeder.append(FirstMoment_Vr_White_Sum[e])
          FirstMoment_Vr_Blue_SumFeeder.append(FirstMoment_Vr_Blue_Sum[e])
          FirstMoment_Vi_Red_SumFeeder.append(FirstMoment_Vi_Red_Sum[e])
          FirstMoment_Vi_White_SumFeeder.append(FirstMoment_Vi_White_Sum[e])
          FirstMoment_Vi_Blue_SumFeeder.append(FirstMoment_Vi_Blue_Sum[e])
          SecondMoment_Vr_Red_SumFeeder.append(SecondMoment_Vr_Red_Sum[e])
          SecondMoment_Vr_White_SumFeeder.append(SecondMoment_Vr_White_Sum[e])
          SecondMoment_Vr_Blue_SumFeeder.append(SecondMoment_Vr_Blue_Sum[e])
          SecondMoment_Vi_Red_SumFeeder.append(SecondMoment_Vi_Red_Sum[e])
          SecondMoment_Vi_White_SumFeeder.append(SecondMoment_Vi_White_Sum[e])
          SecondMoment_Vi_Blue_SumFeeder.append(SecondMoment_Vi_Blue_Sum[e])
          B1TotalRedFeeder.append(B1TotalRed[e])
          B2TotalRedFeeder.append(B2TotalRed[e])
          B1TotalWhiteFeeder.append(B1TotalWhite[e])
          B2TotalWhiteFeeder.append(B2TotalWhite[e])
          B1TotalBlueFeeder.append(B1TotalBlue[e])
          B2TotalBlueFeeder.append(B2TotalBlue[e])
      else:
        for e in range(Data[0]-2+1,len(FirstMoment_Vr_Red_Sum)):
          FirstMoment_Vr_Red_SumFeeder.append(FirstMoment_Vr_Red_Sum[e])
          FirstMoment_Vr_White_SumFeeder.append(FirstMoment_Vr_White_Sum[e])
          FirstMoment_Vr_Blue_SumFeeder.append(FirstMoment_Vr_Blue_Sum[e])
          FirstMoment_Vi_Red_SumFeeder.append(FirstMoment_Vi_Red_Sum[e])
          FirstMoment_Vi_White_SumFeeder.append(FirstMoment_Vi_White_Sum[e])
          FirstMoment_Vi_Blue_SumFeeder.append(FirstMoment_Vi_Blue_Sum[e])
          SecondMoment_Vr_Red_SumFeeder.append(SecondMoment_Vr_Red_Sum[e])
          SecondMoment_Vr_White_SumFeeder.append(SecondMoment_Vr_White_Sum[e])
          SecondMoment_Vr_Blue_SumFeeder.append(SecondMoment_Vr_Blue_Sum[e])
          SecondMoment_Vi_Red_SumFeeder.append(SecondMoment_Vi_Red_Sum[e])
          SecondMoment_Vi_White_SumFeeder.append(SecondMoment_Vi_White_Sum[e])
          SecondMoment_Vi_Blue_SumFeeder.append(SecondMoment_Vi_Blue_Sum[e])
```

```
                B1TotalRedFeeder.append(B1TotalRed[e])
                B2TotalRedFeeder.append(B2TotalRed[e])
                B1TotalWhiteFeeder.append(B1TotalWhite[e])
                B2TotalWhiteFeeder.append(B2TotalWhite[e])
                B1TotalBlueFeeder.append(B1TotalBlue[e])
                B2TotalBlueFeeder.append(B2TotalBlue[e])
        #Perform remaining current calculations for red phase
        IMax_Red=sheet.cell_value(a+3, 8)/IBase
        IMin_Red=-1*sheet.cell_value(a+3, 8)/IBase
        IMax_Red_Gen=IMax_Red
        IMin_Red_Gen=IMin_Red
        # print('IMin_Red =',IMin_Red)
        # print('IMax_Red =',IMax_Red)
        NormalisedFirstMoment_IRed.append((FirstMoment_IRed-IMin_Red)/abs(IMax_Red-IMin_Red))
        NormalisedFirstMoment_IRed_Gen.append((FirstMoment_IRed_Gen-IMin_Red_Gen)/abs(IMax_Red_Gen-
IMin_Red_Gen))
        # print('NormalisedFirstMoment_IRed =',NormalisedFirstMoment_IRed)
        # print('NormalisedFirstMoment_IRed_Gen =',NormalisedFirstMoment_IRed_Gen)
        NormalisedSecondMoment_IRed.append((SecondMoment_IRed-
2*IMin_Red*FirstMoment_IRed+IMin_Red**2)/(IMax_Red-IMin_Red)**2)
        NormalisedSecondMoment_IRed_Gen.append((SecondMoment_IRed_Gen-
2*IMin_Red_Gen*FirstMoment_IRed_Gen+IMin_Red_Gen**2)/(IMax_Red_Gen-IMin_Red_Gen)**2)
        # print('NormalisedSecondMoment_IRed =',NormalisedSecondMoment_IRed)
        # print('NormalisedSecondMoment_IRed_Gen =',NormalisedSecondMoment_IRed_Gen)
        if(Loads_PhaseA[a]==0):
          NormalisedAlpha_IRed.append(0)
          NormalisedBeta_IRed.append(0)
        else:
          NormalisedAlpha_IRed.append((NormalisedSecondMoment_IRed[a]-
NormalisedFirstMoment_IRed[a])/(NormalisedFirstMoment_IRed[a]-
NormalisedSecondMoment_IRed[a]/NormalisedFirstMoment_IRed[a]))
          NormalisedBeta_IRed.append((NormalisedAlpha_IRed[a]/NormalisedFirstMoment_IRed[a]-
NormalisedAlpha_IRed[a]))
        NormalisedPercentile_IRed.append(beta.ppf(1-
int(D_Risk)/100,abs(NormalisedAlpha_IRed[a]),abs(NormalisedBeta_IRed[a])))
        Percentile_IRed.append(((NormalisedPercentile_IRed[a]*(IMax_Red-IMin_Red))+IMin_Red)*IBase)
        if(Gens_PhaseA[a]==0):
          NormalisedAlpha_IRed_Gen.append(0)
          NormalisedBeta_IRed_Gen.append(0)
        else:
          NormalisedAlpha_IRed_Gen.append((NormalisedSecondMoment_IRed_Gen[a]-
NormalisedFirstMoment_IRed_Gen[a])/(NormalisedFirstMoment_IRed_Gen[a]-
NormalisedSecondMoment_IRed_Gen[a]/NormalisedFirstMoment_IRed_Gen[a]))
NormalisedBeta_IRed_Gen.append((NormalisedAlpha_IRed_Gen[a]/NormalisedFirstMoment_IRed_Gen[a]-
NormalisedAlpha_IRed_Gen[a]))
        NormalisedPercentile_IRed_Gen.append(beta.ppf(1-
int(D_Risk)/100,abs(NormalisedAlpha_IRed_Gen[a]),abs(NormalisedBeta_IRed_Gen[a])))
        Percentile_IRed_Gen.append(((NormalisedPercentile_IRed_Gen[a]*(IMax_Red_Gen-
IMin_Red_Gen))+IMin_Red_Gen)*IBase)
        #Perform remaining current calculations for white phase
        IMax_White=sheet.cell_value(a+3, 8)/IBase
        IMin_White=-1*sheet.cell_value(a+3, 8)/IBase
        IMax_White_Gen=IMax_Red
        IMin_White_Gen=IMin_Red
```

```
        NormalisedFirstMoment_IWhite.append((FirstMoment_IWhite-IMin_White)/abs(IMax_White-
IMin_White))
        NormalisedFirstMoment_IWhite_Gen.append((FirstMoment_IWhite_Gen-
IMin_White_Gen)/abs(IMax_White_Gen-IMin_White_Gen))
        NormalisedSecondMoment_IWhite.append((SecondMoment_IWhite-
2*IMin_White*FirstMoment_IWhite+IMin_White**2)/(IMax_White-IMin_White)**2)
        NormalisedSecondMoment_IWhite_Gen.append((SecondMoment_IWhite_Gen-
2*IMin_White_Gen*FirstMoment_IWhite_Gen+IMin_White_Gen**2)/(IMax_White_Gen-IMin_White_Gen)**2)
        if(Loads_PhaseB[a]==0):
          NormalisedAlpha_IWhite.append(0)
          NormalisedBeta_IWhite.append(0)
        else:
          NormalisedAlpha_IWhite.append((NormalisedSecondMoment_IWhite[a]-
NormalisedFirstMoment_IWhite[a])/(NormalisedFirstMoment_IWhite[a]-
NormalisedSecondMoment_IWhite[a]/NormalisedFirstMoment_IWhite[a]))
          NormalisedBeta_IWhite.append((NormalisedAlpha_IWhite[a]/NormalisedFirstMoment_IWhite[a]-
NormalisedAlpha_IWhite[a]))
        NormalisedPercentile_IWhite.append(beta.ppf(1-
int(D_Risk)/100,abs(NormalisedAlpha_IWhite[a]),abs(NormalisedBeta_IWhite[a])))
        Percentile_IWhite.append(((NormalisedPercentile_IWhite[a]*(IMax_White-
IMin_White))+IMin_White)*IBase)
        if(Gens_PhaseB[a]==0):
          NormalisedAlpha_IWhite_Gen.append(0)
          NormalisedBeta_IWhite_Gen.append(0)
        else:
          NormalisedAlpha_IWhite_Gen.append((NormalisedSecondMoment_IWhite_Gen[a]-
NormalisedFirstMoment_IWhite_Gen[a])/(NormalisedFirstMoment_IWhite_Gen[a]-
NormalisedSecondMoment_IWhite_Gen[a]/NormalisedFirstMoment_IWhite_Gen[a]))

NormalisedBeta_IWhite_Gen.append((NormalisedAlpha_IWhite_Gen[a]/NormalisedFirstMoment_IWhite_Gen[a]
-NormalisedAlpha_IWhite_Gen[a]))
        NormalisedPercentile_IWhite_Gen.append(beta.ppf(1-
int(D_Risk)/100,abs(NormalisedAlpha_IWhite_Gen[a]),abs(NormalisedBeta_IWhite_Gen[a])))
        Percentile_IWhite_Gen.append(((NormalisedPercentile_IWhite_Gen[a]*(IMax_White_Gen-
IMin_White_Gen))+IMin_White_Gen)*IBase)
        #Perform remaining current calculations for blue phase
        IMax_Blue=sheet.cell_value(a+3, 8)/IBase
        IMin_Blue=-1*sheet.cell_value(a+3, 8)/IBase
        IMax_Blue_Gen=IMax_Blue
        IMin_Blue_Gen=IMin_Blue
        NormalisedFirstMoment_IBlue.append((FirstMoment_IBlue-IMin_Blue)/abs(IMax_Blue-IMin_Blue))
        NormalisedFirstMoment_IBlue_Gen.append((FirstMoment_IBlue_Gen-
IMin_Blue_Gen)/abs(IMax_Blue_Gen-IMin_Blue_Gen))
        NormalisedSecondMoment_IBlue.append((SecondMoment_IBlue-
2*IMin_Blue*FirstMoment_IBlue+IMin_Blue**2)/(IMax_Blue-IMin_Blue)**2)
        NormalisedSecondMoment_IBlue_Gen.append((SecondMoment_IBlue_Gen-
2*IMin_Blue_Gen*FirstMoment_IBlue_Gen+IMin_Blue_Gen**2)/(IMax_Blue_Gen-IMin_Blue_Gen)**2)
        if(Loads_PhaseC[a]==0):
          NormalisedAlpha_IBlue.append(0)
          NormalisedBeta_IBlue.append(0)
        else:
          NormalisedAlpha_IBlue.append((NormalisedSecondMoment_IBlue[a]-
NormalisedFirstMoment_IBlue[a])/(NormalisedFirstMoment_IBlue[a]-
NormalisedSecondMoment_IBlue[a]/NormalisedFirstMoment_IBlue[a]))
```

```
        NormalisedBeta_IBlue.append((NormalisedAlpha_IBlue[a]/NormalisedFirstMoment_IBlue[a]-
NormalisedAlpha_IBlue[a]))
      NormalisedPercentile_IBlue.append(beta.ppf(1-
int(D_Risk)/100,abs(NormalisedAlpha_IBlue[a]),abs(NormalisedBeta_IBlue[a])))
      Percentile_IBlue.append(((NormalisedPercentile_IBlue[a]*(IMax_Blue-IMin_Blue))+IMin_Blue)*IBase)
      if(Gens_PhaseC[a]==0):
        NormalisedAlpha_IBlue_Gen.append(0)
        NormalisedBeta_IBlue_Gen.append(0)
      else:
        NormalisedAlpha_IBlue_Gen.append((NormalisedSecondMoment_IBlue_Gen[a]-
NormalisedFirstMoment_IBlue_Gen[a])/(NormalisedFirstMoment_IBlue_Gen[a]-
NormalisedSecondMoment_IBlue_Gen[a]/NormalisedFirstMoment_IBlue_Gen[a]))
NormalisedBeta_IBlue_Gen.append((NormalisedAlpha_IBlue_Gen[a]/NormalisedFirstMoment_IBlue_Gen[a]-
NormalisedAlpha_IBlue_Gen[a]))
      NormalisedPercentile_IBlue_Gen.append(beta.ppf(1-
int(D_Risk)/100,abs(NormalisedAlpha_IBlue_Gen[a]),abs(NormalisedBeta_IBlue_Gen[a])))
      Percentile_IBlue_Gen.append(((NormalisedPercentile_IBlue_Gen[a]*(IMax_Blue_Gen-
IMin_Blue_Gen))+IMin_Blue_Gen)*IBase)
      # print('NormalisedAlpha_IBlue =',NormalisedAlpha_IBlue)
      # print('NormalisedBeta_IBlue =',NormalisedBeta_IBlue)
      # print('NormalisedPercentile_IBlue =',NormalisedPercentile_IBlue)
      # print('Percentile_IBlue =',Percentile_IBlue)
      # print('NormalisedAlpha_IBlue_Gen =',NormalisedAlpha_IBlue_Gen)
      # print('NormalisedBeta_IBlue_Gen =',NormalisedBeta_IBlue_Gen)
      # print('NormalisedPercentile_IBlue_Gen =',NormalisedPercentile_IBlue_Gen)
      # print('Percentile_IBlue_Gen =',Percentile_IBlue_Gen)
      #Perform remaining voltage calculations for all phases
      if(len(FirstMoment_Vr_Red_SumFeeder)==n):
        for a in range(len(FirstMoment_Vr_Red_SumFeeder)):
          NVr_Red=FirstMoment_Vr_Red_SumFeeder[a]
          NVrSquared_Red=SecondMoment_Vr_Red_SumFeeder[a]+B1TotalRedFeeder[a]
          NVi_Red=FirstMoment_Vi_Red_SumFeeder[a]
          NViSquared_Red=SecondMoment_Vi_Red_SumFeeder[a]+B2TotalRedFeeder[a]
          FirstMoment_Vc_Red.append(Vs_pu-
          NVr_Red+0.5*NViSquared_Red/Vs_pu+0.5*NVr_Red*NViSquared_Red/Vs_pu**2+
          0.5*NVrSquared_Red*NViSquared_Red/Vs_pu**3)
          SecondMoment_Vc_Red.append(Vs_pu**2-(2*Vs_pu*NVr_Red)+NVrSquared_Red+NViSquared_Red)
          NormalisedAlpha_Red.append(abs((SecondMoment_Vc_Red[a]-
          FirstMoment_Vc_Red[a])/(FirstMoment_Vc_Red[a]-
          SecondMoment_Vc_Red[a]/FirstMoment_Vc_Red[a])))
          NormalisedBeta_Red.append(abs(NormalisedAlpha_Red[a]/FirstMoment_Vc_Red[a]-
          NormalisedAlpha_Red[a]))
          Percentile_Vc_Red.append(beta.ppf(int(D_Risk)/100,abs(NormalisedAlpha_Red[a]),abs(NormalisedB
          eta_Red[a])))
          V_Con_Red.append(Percentile_Vc_Red[len(Percentile_Vc_Red)-1]*float(Vbase_Entered))
          NVr_White=FirstMoment_Vr_White_SumFeeder[a]
          NVrSquared_White=SecondMoment_Vr_White_SumFeeder[a]+B1TotalWhiteFeeder[a]
          NVi_White=FirstMoment_Vi_White_SumFeeder[a]
          NViSquared_White=SecondMoment_Vi_White_SumFeeder[a]+B2TotalWhiteFeeder[a]
          FirstMoment_Vc_White.append(Vs_pu-
          NVr_White+0.5*NViSquared_White/Vs_pu+0.5*NVr_White*NViSquared_White/Vs_pu**2+0.5*NVrSqu
          ared_White*NViSquared_White/Vs_pu**3)
          SecondMoment_Vc_White.append(Vs_pu**2-(2*Vs_pu*NVr_White)+
          NVrSquared_White+NViSquared_White)
```

```
            NormalisedAlpha_White.append(abs((SecondMoment_Vc_White[a]-FirstMoment_Vc_White[a])/
            (FirstMoment_Vc_White[a]-SecondMoment_Vc_White[a]/FirstMoment_Vc_White[a])))
            NormalisedBeta_White.append(abs(NormalisedAlpha_White[a]/FirstMoment_Vc_White[a]-
            NormalisedAlpha_White[a]))
            Percentile_Vc_White.append(beta.ppf(int(D_Risk)/100,abs(NormalisedAlpha_White[a]),abs(Normalis
            edBeta_White[a])))
            V_Con_White.append(Percentile_Vc_White[len(Percentile_Vc_White)-1]*float(Vbase_Entered))
            NVr_Blue=FirstMoment_Vr_Blue_SumFeeder[a]
            NVrSquared_Blue=SecondMoment_Vr_Blue_SumFeeder[a]+B1TotalBlueFeeder[a]
            NVi_Blue=FirstMoment_Vi_Blue_SumFeeder[a]
            NViSquared_Blue=SecondMoment_Vi_Blue_SumFeeder[a]+B2TotalBlueFeeder[a]
            FirstMoment_Vc_Blue.append(Vs_pu-
            NVr_Blue+0.5*NViSquared_Blue/Vs_pu+0.5*NVr_Blue*NViSquared_Blue/Vs_pu**2+0.5*NVrSquared_
            Blue*NViSquared_Blue/Vs_pu**3)
            SecondMoment_Vc_Blue.append(Vs_pu**2-(2*Vs_pu*NVr_Blue)+NVrSquared_Blue+NViSquared_Blue)
            NormalisedAlpha_Blue.append(abs((SecondMoment_Vc_Blue[a]-
            FirstMoment_Vc_Blue[a])/(FirstMoment_Vc_Blue[a]-
            SecondMoment_Vc_Blue[a]/FirstMoment_Vc_Blue[a])))
            NormalisedBeta_Blue.append(abs(NormalisedAlpha_Blue[a]/FirstMoment_Vc_Blue[a]-
            NormalisedAlpha_Blue[a]))
            Percentile_Vc_Blue.append(beta.ppf(int(D_Risk)/100,abs(NormalisedAlpha_Blue[a]),abs(Normalised
            Beta_Blue[a])))
            V_Con_Blue.append(Percentile_Vc_Blue[len(Percentile_Vc_Blue)-1]*float(Vbase_Entered))
        a+=1

#Get user's desired plot type from Inputs spreadsheet
Plot_Select=sheet.cell_value(1,31)
for a in range(len(V_Con_Blue)):
    print('len(V_Con_Blue) =',len(V_Con_Blue))
    Nodes.append(a)
    if(Plot_Select==0):
        #Sets the upper and lower voltage limit based on the user input received
        UpperVoltageLimit.append(float(Vs_Entered)*(1+float(VoltageLimit_Entered)/100))
        LowerVoltageLimit.append(float(Vs_Entered)*(1-float(VoltageLimit_Entered)/100))
        toplimit=float(Vs_Entered)*(1+2*float(VoltageLimit_Entered)/100)
        bottomlimit=float(Vs_Entered)*(1-2*float(VoltageLimit_Entered)/100)
        #Creates a subplot for the results graph
        ax = plt.subplot2grid((4, 1), (0, 0), rowspan=2)
        #Creates a subplot for the results table
        axTable1 = plt.subplot2grid((4, 1), (2, 0),rowspan=1)
        #Removes the axis from the table below the plot
        axTable1.axis("off")
        axTable1.axes.get_xaxis().set_visible(False)
        axTable1.axes.get_yaxis().set_visible(False)
        #Specifies table headings, results and units for display in the results table
        formatted_V_Con_Red = "{:.3f}".format(V_Con_Red[-1])
        formatted_V_Con_White = "{:.3f}".format(V_Con_White[-1])
        formatted_V_Con_Blue = "{:.3f}".format(V_Con_Blue[-1])
        VoltageDropRed=(float(Vs_Entered)-float(formatted_V_Con_Red))/float(Vs_Entered)*100
        VoltageDropWhite=(float(Vs_Entered)-float(formatted_V_Con_White))/float(Vs_Entered)*100
        VoltageDropBlue=(float(Vs_Entered)-float(formatted_V_Con_Blue))/float(Vs_Entered)*100
        formatted_VoltageDropRed="{:.2f}".format(VoltageDropRed)
        formatted_VoltageDropWhite="{:.2f}".format(VoltageDropWhite)
        formatted_VoltageDropBlue="{:.2f}".format(VoltageDropBlue)
```

```
        formatted_Isum_Red = "{:.2f}".format(Percentile_IRed_Gen[-1])
        formatted_Isum_White = "{:.2f}".format(Percentile_IWhite_Gen[-1])
        formatted_Isum_Blue = "{:.2f}".format(Percentile_IBlue_Gen[-1])
        formatted_MeanIsum_Red = "{:.2f}".format(((NormalisedAlpha_IRed_Gen[-
1]/(NormalisedAlpha_IRed_Gen[-1]+NormalisedBeta_IRed_Gen[-1]))*(IMax_Red_Gen-
IMin_Red_Gen)+IMin_Red_Gen)*IBase)
        formatted_MeanIsum_White = "{:.2f}".format(((NormalisedAlpha_IWhite_Gen[-
1]/(NormalisedAlpha_IWhite_Gen[-1]+NormalisedBeta_IWhite_Gen[-1]))*(IMax_White_Gen-
IMin_White_Gen)+IMin_White_Gen)*IBase)
        formatted_MeanIsum_Blue = "{:.2f}".format(((NormalisedAlpha_IBlue_Gen[-
1]/(NormalisedAlpha_IBlue_Gen[-1]+NormalisedBeta_IBlue_Gen[-1]))*(IMax_Blue_Gen-
IMin_Blue_Gen)+IMin_Blue_Gen)*IBase)
        formatted_StdDevIsum_Red = "{:.2f}".format(sqrt(abs(SecondMoment_IRed_Gen-
FirstMoment_IRed_Gen**2)))
        formatted_StdDevIsum_White = "{:.2f}".format(sqrt(abs(SecondMoment_IWhite_Gen-
FirstMoment_IWhite_Gen**2)))
        formatted_StdDevIsum_Blue = "{:.2f}".format(sqrt(abs(SecondMoment_IBlue_Gen-
FirstMoment_IBlue_Gen**2)))
        #Create table row and column headings
        val1 = ('Red', 'White', 'Blue', 'Unit')
        val2 = ['Percentile Consumer Voltage','Voltage Drop','Percentile Isum','Mean Isum','StdDev Isum']
        val3 = [[formatted_V_Con_Red, formatted_V_Con_White, formatted_V_Con_Blue,'kV'],
[formatted_VoltageDropRed, formatted_VoltageDropWhite,
formatted_VoltageDropBlue,'%'],[formatted_Isum_Red,formatted_Isum_White,formatted_Isum_Blue,'A'],[formatt
ed_MeanIsum_Red,formatted_MeanIsum_White,formatted_MeanIsum_Blue,'A'],[formatted_StdDevIsum_Red,for
matted_StdDevIsum_White,formatted_StdDevIsum_Blue,'A']]
        #Configures the results table
        Table = axTable1.table(cellText = val3,
                               cellLoc='center',
                               rowLoc='center',
                               rowColours='c'*7,
                               rowLabels = val2,
                               colColours='c'*4,
                               colLabels = val1,
                               loc ='bottom')
        Table.auto_set_font_size(False)
        Table.set_fontsize(12)
        cell_dict = Table.get_celld()
        for i in range(6):
          for j in range(4):
            cell_dict[(i,j)].set_width(0.1)
            cell_dict[(i,j)].set_height(0.2)
        for i in range(1,6):
          cell_dict[(i,-1)].set_width(0.1)
          cell_dict[(i,-1)].set_height(0.2)

      elif(Plot_Select==1):
        #Creates a subplot for the results graph
        ax = plt.subplot2grid((4, 1), (0, 0), rowspan=2)
        #Creates a subplot for the results table
        axTable1 = plt.subplot2grid((4, 1), (2, 0),rowspan=1)
        #Removes the axis from the table below the plot
        axTable1.axis("off")
        axTable1.axes.get_xaxis().set_visible(False)
```

```
        axTable1.axes.get_yaxis().set_visible(False)
        #Specifies table headings, results and units for display in the results table
        if(a<n):
          MeanRed.append(NormalisedAlpha_Red[a]/(NormalisedAlpha_Red[a]+NormalisedBeta_Red[a])/Vs_pu)
MeanWhite.append(NormalisedAlpha_White[a]/(NormalisedAlpha_White[a]+NormalisedBeta_White[a])/Vs_pu
)
MeanBlue.append(NormalisedAlpha_Blue[a]/(NormalisedAlpha_Blue[a]+NormalisedBeta_Blue[a])/Vs_pu)
        print('MeanRed =',MeanRed)
        # print('MeanWhite =',MeanWhite)
        # print('MeanBlue =',MeanBlue)
        formatted_MeanRed="{:.4f}".format(MeanRed[-1])
        formatted_MeanWhite="{:.4f}".format(MeanWhite[-1])
        formatted_MeanBlue="{:.4f}".format(MeanBlue[-1])
        #Create table row and column headings
        val1 = ('Red', 'White', 'Blue', 'Unit')
        val2 = ['Mean Voltage']
        val3 = [[formatted_MeanRed, formatted_MeanWhite, formatted_MeanBlue,'p.u']]
        #Configures the results table
        Table = axTable1.table(cellText = val3,
                        cellLoc='center',
                        rowLoc='center',
                        rowColours='c'*7,
                        rowLabels = val2,
                        colColours='c'*4,
                        colLabels = val1,
                        loc ='bottom')
        Table.auto_set_font_size(False)
        Table.set_fontsize(12)
        cell_dict = Table.get_celld()
        for i in range(2):
          for j in range(4):
            cell_dict[(i,j)].set_width(0.1)
            cell_dict[(i,j)].set_height(0.2)
        for i in range(1,2):
          cell_dict[(i,-1)].set_width(0.1)
          cell_dict[(i,-1)].set_height(0.2)
      else:
        #Creates a subplot for the results graph
        ax = plt.subplot2grid((4, 1), (0, 0), rowspan=2)
        #Creates a subplot for the results table
        axTable1 = plt.subplot2grid((4, 1), (2, 0),rowspan=1)
        #Removes the axis from the table below the plot
        axTable1.axis("off")
        axTable1.axes.get_xaxis().set_visible(False)
        axTable1.axes.get_yaxis().set_visible(False)
        #Specifies table headings, results and units for display in the results table
        if(a<n):

StdDevRed.append(sqrt((NormalisedAlpha_Red[a]*NormalisedBeta_Red[a])/((NormalisedAlpha_Red[a]+Norm
alisedBeta_Red[a]+1)*((NormalisedAlpha_Red[a]+NormalisedBeta_Red[a])**2)))/Vs_pu)

StdDevWhite.append(sqrt((NormalisedAlpha_White[a]*NormalisedBeta_White[a])/((NormalisedAlpha_White[a
]+NormalisedBeta_White[a]+1)*((NormalisedAlpha_White[a]+NormalisedBeta_White[a])**2)))/Vs_pu)
```

```
StdDevBlue.append(sqrt((NormalisedAlpha_Blue[a]*NormalisedBeta_Blue[a])/((NormalisedAlpha_Blue[a]+Nor
malisedBeta_Blue[a]+1)*((NormalisedAlpha_Blue[a]+NormalisedBeta_Blue[a])**2)))/Vs_pu)
        print('StdDevRed =',StdDevRed)
        print('StdDevWhite =',StdDevWhite)
        print('StdDevBlue =',StdDevBlue)
        formatted_StdDevRed="{:.4f}".format(StdDevRed[-1])
        formatted_StdDevWhite="{:.4f}".format(StdDevWhite[-1])
        formatted_StdDevBlue="{:.4f}".format(StdDevBlue[-1])
        #Create table row and column headings
        val1 = ('Red', 'White', 'Blue', 'Unit')
        val2 = ['Mean Voltage']
        val3 = [[formatted_StdDevRed, formatted_StdDevWhite, formatted_StdDevBlue,'p.u']]
        #Configures the results table
        Table = axTable1.table(cellText = val3,
                        cellLoc='center',
                        rowLoc='center',
                        rowColours='c'*7,
                        rowLabels = val2,
                        colColours='c'*4,
                        colLabels = val1,
                        loc ='bottom')
        Table.auto_set_font_size(False)
        Table.set_fontsize(12)
        cell_dict = Table.get_celld()
        for i in range(2):
          for j in range(4):
            cell_dict[(i,j)].set_width(0.1)
            cell_dict[(i,j)].set_height(0.2)
        for i in range(1,2):
          cell_dict[(i,-1)].set_width(0.1)
          cell_dict[(i,-1)].set_height(0.2)
  print('Plot_Select =',Plot_Select)
  mng = plt.get_current_fig_manager()
  #Opens the results window in maximised mode
  mng.window.state("zoomed")

  #Used to create the actual plots on the graph. Markers are used to specify the results at each node
  # print('TotalDistance =',TotalDistance)
  # print('V_Con_Blue =',V_Con_Blue)
  if(Plot_Select==0):
    ax.set_title('3 Phase Feeder Voltage Profile',fontsize=18,pad=20)
    ax.plot(TotalDistance, V_Con_Blue,marker='o',markersize=10,color='blue',linewidth=2,label='Blue Phase')
    ax.plot(TotalDistance, V_Con_Red,marker='o',markersize=10,color='red',linewidth=2,label='Red Phase')
    ax.plot(TotalDistance, V_Con_White,marker='o',markersize=10,color='yellow',linewidth=2,label='White
Phase')
    ax.plot(TotalDistance,UpperVoltageLimit,linestyle='dashed',color='black',linewidth=2)
    ax.plot(TotalDistance, LowerVoltageLimit,linestyle='dashed',color='black',linewidth=2,label='Voltage Limit')
    #Create axis labels
    ax.set_xlabel('Distance [m]',fontsize=14)
    ax.set_ylabel('Voltage [kV]',fontsize=14)
  elif(Plot_Select==1):
    ax.set_title('Mean Voltage Plot',fontsize=18,pad=20)
    ax.plot(Nodes, MeanBlue,marker='o',markersize=10,color='blue',linewidth=2,label='Blue Phase')
```

```
        ax.plot(Nodes, MeanRed,marker='o',markersize=10,color='red',linewidth=2,label='Red Phase')
        ax.plot(Nodes, MeanWhite,marker='o',markersize=10,color='yellow',linewidth=2,label='White Phase')
        #Create axis labels
        ax.set_xlabel('Node',fontsize=14)
        ax.set_ylabel('Mean Voltage [p.u]',fontsize=14)
    else:
        ax.set_title('Voltage Standard Deviation Plot',fontsize=18,pad=20)
        ax.plot(Nodes, StdDevBlue,marker='o',markersize=10,color='blue',linewidth=2,label='Blue Phase')
        ax.plot(Nodes, StdDevRed,marker='o',markersize=10,color='red',linewidth=2,label='Red Phase')
        ax.plot(Nodes, StdDevWhite,marker='o',markersize=10,color='yellow',linewidth=2,label='White Phase')
        #Create axis labels
        ax.set_xlabel('Node',fontsize=14)
        ax.set_ylabel('Voltage Standard Deviation [p.u]',fontsize=14)
    #Create a legend and specify its location in the results window
    ax.legend(loc="right")
    #Create gridlines for the results graph
    ax.grid(b=True,which='both',axis='both')
    #Creates a heading for the results table
    plt.title('Results at Consumer End',fontsize=18,y=0.05)
    #Show the plot
    plt.show()
    if(a==(len(V_Con_Blue)-1)):
        #Removes the 'Import Input Sheet and Get Results' button
        Proceed_Button.destroy()
        #Creates a 'Restart' button to allow the user the recalculate results by changing initial input values
        Restart_Button=Button(window,width=10,text="Recalculate",command=Proceed)
        Restart_Button.grid(row=1,column=5,columnspan=3,padx=10,pady=20,sticky=NSEW)
        #Creates a 'Quit' button which will exit the program when clicked
        Quit_Button=Button(window,width=10,text="Exit Program",command=Exit_Program)
        Quit_Button.grid(row=2,column=5,columnspan=3,padx=10,pady=10,sticky=NSEW)

#Creates a 'File' dropdown menu
File_DropdownMenu = Menu(menubar, tearoff=0)
#Add options to the File dropdown menu. Also specify which function should run when am option is selected.
File_DropdownMenu.add_command(label="About", command=About)
File_DropdownMenu.add_command(label="Exit", command=Exit_Program)
menubar.add_cascade(label="File", menu=File_DropdownMenu)

#Create a 'Tools' dropdown menu
Tools_DropdownMenu = Menu(menubar, tearoff=0)
menubar.add_cascade(label="Tools", menu=Tools_DropdownMenu)
submenu = Menu(Tools_DropdownMenu,tearoff=0)
submenu.add_command(label="Using ADMD, Voltage and CB Rating",command=BetaCalculator_Option1)
submenu.add_command(label="Using Std Dev, Mean and CB Rating",command=BetaCalculator_Option2)
Tools_DropdownMenu.add_cascade(label='Beta Parameter Calculator   ', menu=submenu, underline=0)

#Create a 'Help' dropdown menu which contains the link to the program user guide
Help_DropdownMenu = Menu(menubar, tearoff=0)
Help_DropdownMenu.add_command(label="User Guide", command=User_Guide)
menubar.add_cascade(label="Help", menu=Help_DropdownMenu)

#Insert text to request base voltage from user
Vs=Label(window,text="Supply Voltage (kV):",bg='light blue')
```

```
#Insert the above text in a specified position in the grid.
Vs.grid(row=0,column=0)

#Create entry box to accept user input
Vs_Entry=Entry(window,width=8)
Vs_Entry.grid(row=0,column=1,padx=10,pady=10)

#Creates labels and entry grids for the remaining inputs required from the user
Vs_Prompt=Label(window,text="V Base (kV):",bg='light blue')
Vs_Prompt.grid(row=0,column=2)
Vbase_Entry=Entry(window,width=8)
Vbase_Entry.grid(row=0,column=3,padx=10,pady=10)

Sbase_Prompt=Label(window,text="S Base (MVA):",bg='light blue')
Sbase_Prompt.grid(row=0,column=4)
Sbase_Entry=Entry(window,width=8)
Sbase_Entry.grid(row=0,column=5,padx=10,pady=10)

Temp_Prompt=Label(window,text="Cable Temperature (°C):",bg='light blue')
Temp_Prompt.grid(row=0,column=6)
Temp_Entry=Entry(window,width=8)
Temp_Entry.grid(row=0,column=7,padx=10,pady=10)

VoltageLimit_Prompt=Label(window,text="Voltage Limit (%):",bg='light blue')
VoltageLimit_Prompt.grid(row=0,column=8)
VoltageLimit_Spin=Spinbox(window, from_=0, to=100, width=8)
VoltageLimit_Spin.grid(row=0,column=9,padx=10,pady=10)

ConfidenceLevel_Prompt=Label(window,text="Confidence Level (%):",bg='light blue')
ConfidenceLevel_Prompt.grid(row=0,column=10)
ConfidenceLevel_Spin=Spinbox(window, from_=0, to=100, width=8)
ConfidenceLevel_Spin.grid(row=0,column=11,padx=10,pady=10)

#Creates a dropdown menu for the user to select the system configuration
SystemConfig_Prompt=Label(window,text="System Configuration:",bg='light blue')
SystemConfig_Prompt.grid(row=0,column=12)
SystemConfig_Combo=Combobox(window,width=15)
SystemConfig_Combo['values']=("3-phase, 3 wire","3-phase, 4 wire")
SystemConfig_Combo.current(0)
SystemConfig_Combo.grid(row=0,column=13,padx=10,pady=10)

#Creates a button which imports the user input spreadsheet when clicked
Proceed_Button=Button(window,width=10,text="Import Input Sheet and Get Results",command=Proceed)
Proceed_Button.grid(row=1,column=6,columnspan=4,padx=10,pady=20,sticky=NSEW)

#Used to add the menu to the display
window.config(menu=menubar,bg='light blue')

#Sets the size of the GUI window
window.geometry('1350x300')

#Function used to keep the window displayed to user
window.mainloop()
```