Load Modeling using Synchrophasor Data for Improved Contingency Analysis

Hema A. Retty

Dissertation submitted to the faculty of the Virginia Polytechnic Institute and State University in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

In

Electrical Engineering

November 30, 2015

Blacksburg, Virginia

Keywords: Power Systems, Load Modeling, Machine Learning, Data Mining, Regression, Neural Networks, Bounded-Variable Least Squares (BVLS), Phasor Measurement Units (PMUs)

Load Modeling using Synchrophasor Data for Improved Contingency Analysis

Hema A. Retty

Abstract

For decades, researchers have sought to make the North American power system as reliable as possible with many security measures in place to include redundancy. Yet the increasing number of blackouts and failures have highlighted the areas that require improvement. Meeting the increasing demand for energy and the growing complexity of the loads are two of the main challenges faced by the power grid. In order to prepare for contingencies and maintain a secure state, power engineers must perform simulations using steady state and dynamic models of the system. The results from the contingency studies are only as accurate as the models of the grid components. The load components are generally the most difficult to model since they are controlled by the consumer. This study focuses on developing static and dynamic load models using advanced mathematical approximation algorithms and wide area measurement devices, which will improve the accuracy of the system analysis and hopefully decrease the frequency of blackouts.

The increasing integration of phasor measurement units (PMUs) into the power system allows us to take advantage of synchronized measurements at a high data rate. These devices are capable of changing the way we manage online security within the Energy Management System (EMS) and can enhance our offline tools. This type of data helps us redevelop the measurement-based approach to load modeling.

The static ZIP load model composition is estimated using a variation of the method of least squares, called bounded-variable least squares. The bound on the ZIP load parameters allows the measurement matrix to be slightly correlated. The ZIP model can be determined within a small range of error that won't affect the contingency studies. Machine learning is used to design the dynamic load model. Neural network training is applied to fault data obtained near the load bus and the derived network model can estimate the load parameters. The neural network is trained using simulated data and then applied to real PMU measurements. A PMU algorithm was developed to transform the simulated measurements into a realistic representation of phasor data. These new algorithms will allow us to estimate the load models that are used in contingency studies.

To My Loving Parents,
Athirame and Chandrika Retty

Acknowledgements

As the proverb goes, “it takes a village to raise a child”; similarly, without the immeasurable support of my professors, mentors, friends and family throughout my education, I would not have been able to pursue this PhD and get to where I am today.

I would like to express my sincere gratitude to my advisor, Dr. Virgilio Centeno. It is somewhat cliché to say that I could not have completed my PhD without Dr. Centeno but truthfully his support during my Masters Degree was the only reason I even considered starting a PhD. I could not have imagined having a better advisor for my graduate studies. His patience, time, and motivation were a constant source of inspiration in my life inside and outside the power lab. There is one more incredible person without whom this PhD would not have been possible. I am extremely grateful to Dr. James Thorp for his invaluable mentorship over the past 3 years. His passion for research is commendable and has inspired me to push my goals even further.

Besides my advisors, I would like to thank the rest of my PhD committee, Dr. Jaime De La Ree, Dr. Dhruv Batra and Dr. Anil Vullikanti, for their insightful comments and encouragement, but also for their hard questions which incented me to widen my research from various perspectives. I thank my fellow labmates for the stimulating discussions, the supportive pep talks, and for all the fun we have had in the lab over the years.

I would like to thank my parents, Athirame and Chandrika Retty, for all their support and sacrifices throughout my life in order to make my education their top priority. They have celebrated my accomplishments, encouraged me during my failures and always believed in me even when I didn’t believe in myself. I thank you from the bottom of my heart.

Finally, I would like to thank my fiancé, Jake Webb, for supporting me throughout this PhD and my life in general. From proof-reading my writing and watching me practice my presentations to being by my side through the stressful times, you have been my rock for the past three years and I look forward to spending the rest of my life with you.

Table of Contents

List of Figures

List of Tables

Chapter 1. Introduction

1.1 Motivation

In recent years, the Pacific Southwest power system has been prone to be heavily burdened during the summer months, mainly due to air conditioning load. On September 8, 2011 [1], the region faced a severe blackout which lasted 11 minutes and led to cascading outages, leaving 2.7 million customers without electricity. San Diego (as shown in figure 1-1 below), accounting for almost 1.5 million of those customers, was the largest city without power. The disturbance affected rush hour traffic, schools, businesses, flights, public transportation, water and sewage utilities and households. The initial event that led to the outages was the loss of a single 500 kV transmission line. However, the system is designed to withstand N-1 contingencies, such as the loss of a single line; therefore, there must have been other factors at play which prevented the power grid from operating securely. After extensive investigation by the Federal Energy Regulatory Commission (FERC) and North American Electric Reliability Corporation (NERC) with the cooperation of local utilities, the report [1] showed that "the system was not being operated in a secure N-1 state" and "many entities' real-time tools, such as State Estimator and Real-Time Contingency Analysis (RTCA), are restricted by models that do not accurately or fully reflect facilities".

In order for power utilities to conduct accurate contingency studies and prevention, the models used in their simulation tools must represent the actual power system. As most components in the grid, such as the transmission lines and generating plants, are designed and controlled by the power entities, the load is generally controlled by the customer and therefore the utility must make its own intelligent assumptions and predictions about it. The load can vary depending on the geographic location, the season, the time of day or even the occurrence of special events such as the Super Bowl. The load is therefore one of the most difficult components to model, since it is ever-changing in size and dynamics. Incorrect models of the load may lead to poor contingency planning, leaving system operators ill-prepared [2].

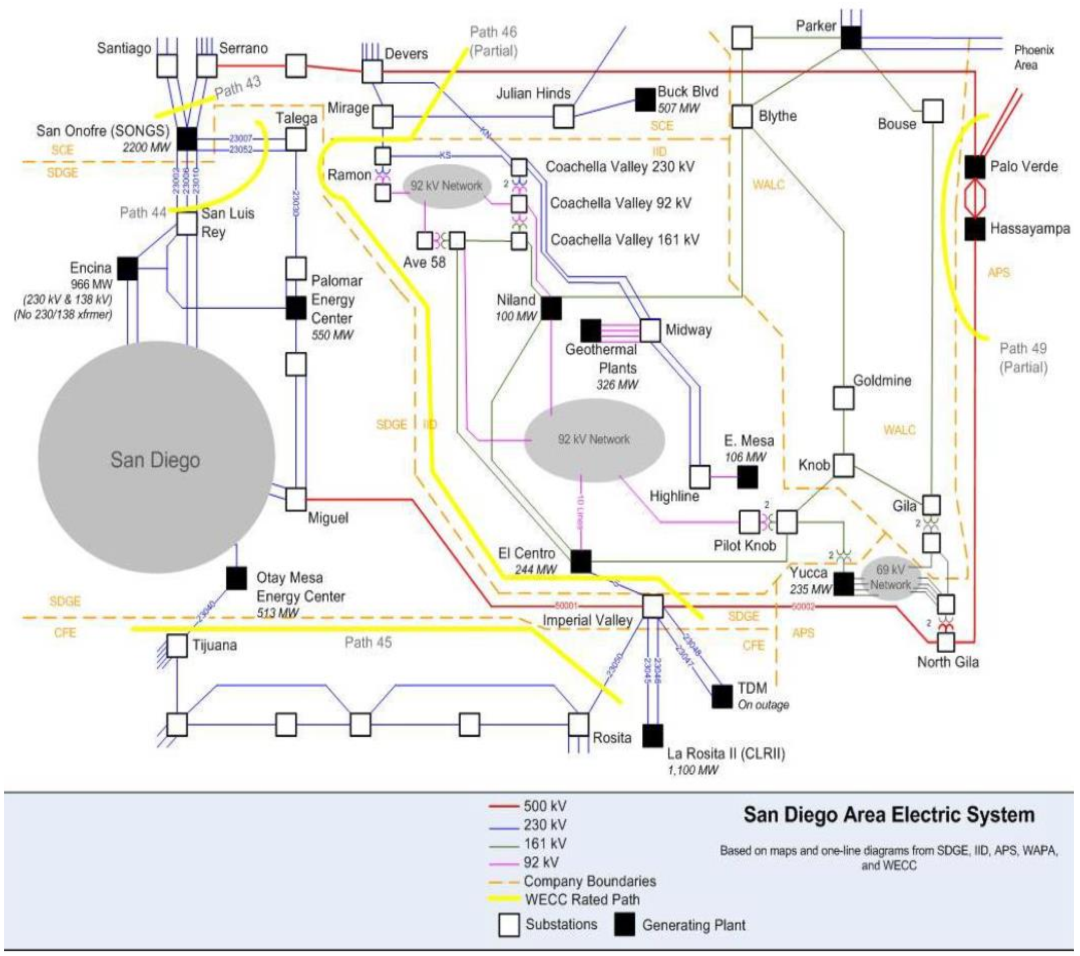


Figure 1-1. Affected Regions in San Diego Area Electric System – September 8, 2011

Unlike generator modeling, load representation has remained the least accurate. Load modeling becomes increasingly important, as loads play a greater role in the power system stability. Resistive loads, such as space heating, incandescent lighting, resistive water heating and cooking, are decreasing. These loads are voltage sensitive; when the voltages decline, the resistive loads reduce their demand. They do not have much of an effect on stability since they are able to provide relief for the transmission grid. On the other hand, fluorescent lights, electronic drives, electronic chargers, light-inertia compressor motors in air-conditioners and heat-pumps are increasing their infiltration into the system. These loads are more energy efficient but are capable of making the grid more susceptible. Electronic loads behave mostly as constant power loads with respect to voltage and frequency deviations. As grid voltages decline, the electronic loads increase their current draw from the power system, thereby further degrading voltage stability and damping of power oscillations.

Generally, loads are modelled using historical and statistical data based on the region, season and time of day. SCADA measurements of voltage and current can be used to validate the models. These methods are based on numerous assumptions, allowing room for significant error. With the increasing integration of phasor measurement units (PMUs) in the power system, that allow synchronized measurements to be taken up to 30 times per second, a new channel to system component modeling has opened up. The objective of this dissertation is to better prepare the power system for contingencies using synchrophasor data.

1.2 Synchrophasors and Wide Area Measurement Systems

The power grid is under observation 24 hours a day and 365 days a year in order to function securely at all times. In the foreground, system operators work around the clock to maintain this stability. Alongside them, power engineers working in areas such as operations, control or planning, conduct studies to resolve issues, prevent disturbances and make improvements to the grid. In order to perform these day-to-day operations reliably, the power system must always be completely observable. Under these conditions, the system state can be estimated and the measurements can be used for applications [3]. The frequency of the state estimation highly depends on the rate of the incoming measurements. The North American power grid relies on the Energy Management System (EMS) [4], which performs monitoring, and control functions, also known as the Supervisory Control and Data Acquisition (SCADA) system. Remote terminal units (RTUs) connected to analog sensors convert the signals into digital data and transmits the measurements to the control system. The SCADA system is capable of presenting system wide data every 2 to 10 seconds. With delays in transmission and inconsistency in time tagging among devices, it can be difficult to synchronize the sequence of measurements from multiple devices when parsing through the data.

There have been some breakthroughs in power system measuring devices. Dr. Arun Phadke and Dr. James Thorp invented the Phasor Measurements Unit (PMU) in 1988 at Virginia Tech [5]. This electronic device takes a raw A/C signal and applies an analog to digital conversion to compute the phasor measurement [6]. A phasor is a simplified complex number representation of a sinusoidal waveform. It is made up of the magnitude, phase angle and frequency of the original analog signal. Figure 1-2 describes the relationship between the two forms.

$$A\cos(\omega t + \theta) = A\angle\theta$$

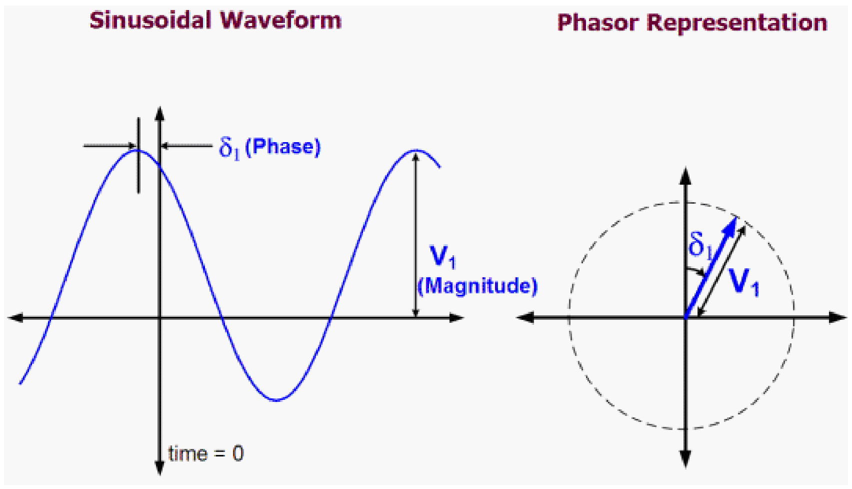


Figure 1-2. Sinusoid to Phasor Representation [7]

The invention of synchrophasor technology provides a wide breadth of opportunities not only for state estimation, but also in many other areas of power systems. The phasor representation of the measurement allows for easier use with SCADA applications. The term “synchrophasor” corresponds to time-synchronized phasor measurement. Each PMU is connected to a GPS clock allowing all PMU measurements to be time synchronized no matter the location. This overcomes the issues of time discrepancies and delays, prevalent with the earlier RTUs. The network of PMUs and other related devices that give the system observability [8] is known as the Wide Area Measurement System (WAMS), shown in figure 1-3. The PMU is connected to a GPS clock using an IRIG-B cable, allowing all measurements to be time-synchronized and removing any discrepancies caused by data transmission delays. Each individual PMU sends its data to a Phasor Data Concentrator (PDC) to aggregate the measurements before sending them to the SCADA applications. Sometimes, a super-PDC collects data from the smaller substation PDCs before transmitting to the applications. The PMU can take measurements at 30 or 60 frames per second and the PDC is capable of handling the same data rate. Currently, the implementation of PMUs has been mainly at the high voltage level. As manufacturing and installation costs decrease, the PMUs will be deployed at the lower voltage levels.

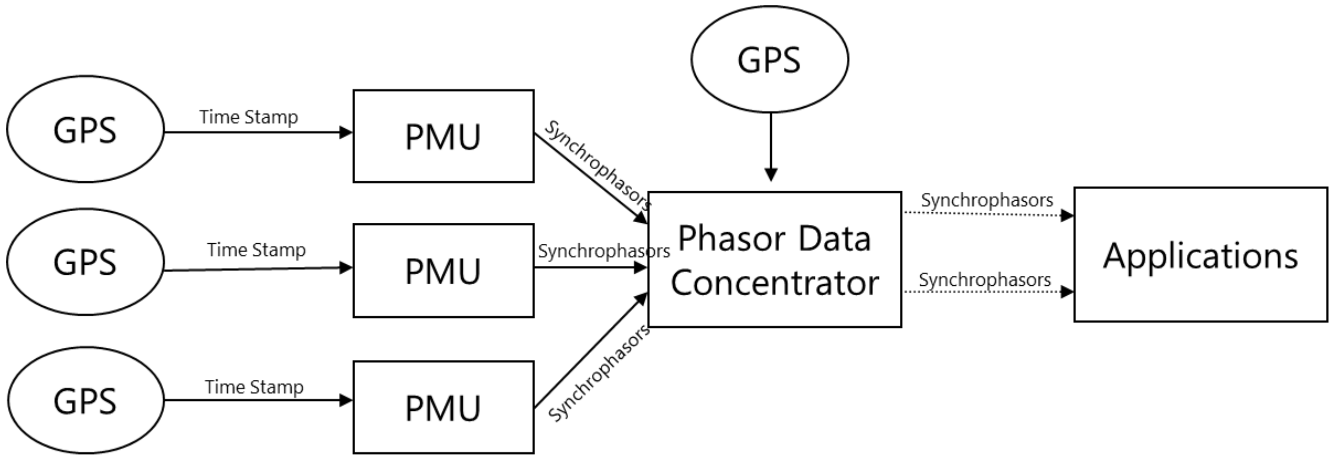


Figure 1-3. Wide Area Measurement System

Even though the PMU has been around for quite some time now, it is only in recent years that utilities and other power entities have begun to adopt this technology. One of the main reasons for this is the cost associated with manufacturing and installation of the PMUs. There would also be time associated with the development of new standards and training for these measurement devices, along with the integration with other devices. The reason for the recent changes are due to the increasing need for frequent and reliable measurements. In

2003, the entire Northeast of the United States was engulfed in darkness during the largest blackout in North American history. The combination of system failure and human error led to the cascading failures that turned off the lights from Maryland to Ontario. Following the blackout, the US-Canada Power System Outage Task Force were charged with the assignment of parsing through data to piece together the sequence of events and finding the cause for the initial disturbance. Since there were no active PMUs on the system, it took them months to align the data and discover that the initial disturbance stemmed from a generating plant in Eastlake, Ohio. Subsequently, the government dedicated new funds to aide in the development of the power system. This has allowed utilities to install PMUs, establishing wide area measurement systems. Since then, PMUs have made phenomenal improvements in the power industry. For example, in the aftermath of the 2011 Southwest blackout surrounding San Diego, it only took days to figure out the cause for the disturbances. PMU data can be used for many types of applications; there is new research being developed in this area each day. For the purpose of this dissertation, we will discuss how PMUs can be used to derive load models and perform sensitivity analysis.

1.3 Power System Simulations

Predicting the behaviour of the system during an outage or a heavy load and understanding the effects of the electro-mechanical components such as generators and motors during these events is critical to maintaining a secure state [9]. Planning and operations studies rely heavily on executing both, off-line and real-time, simulations to better prepare for these contingencies [10]. The off-line simulations are purely software based, whereas the real-time simulations interface the software with electronic hardware devices to include real-time data such as measurements.

The network and dynamic models are integrated into the simulation software package to perform steady state and dynamic studies. The network model includes bus information, line/shunt impedances, voltage levels, power generated and power consumed. The dynamic models comprise of generators, exciters, governors and loads. These models include the dynamics properties of the components to simulate their transient behaviour.

In order to run a dynamic or time-dependent simulation, the system model must initially be stable or in steady state. This can be accomplished by running a power flow on the network model and making sure it converges successfully. The static model can be used to predict the system state after changes such as an increase or decrease in load or even a disturbance such as a loss of transmission line, generator or load. The power flow solution provides us with the system state after it has reached an equilibrium. In order to capture the transient or sub-transient behaviour of the system due to the electro-mechanical components, a dynamic simulation must be performed. The simulation provides us with information about the initial response of the system after a small or large change.

Ideally, the dynamic response of the system would exactly match the real system response when the same change occurs. However, this is generally not the case. Many a times, the simulation software is used after a disturbance has occurred; to piece together the sequence of events and to prevent it from occurring again. The measurement data during the disturbance is collected and compared to the system simulation. In most cases, the simulation will not match the real event data and modifications will be made to the system model until the simulation produces the same results. There are two possible causes for this discrepancy; one or more parts of the system model were not updated regularly. The other reason is related to the accuracy of which the system can be modeled. It is very difficult to model every single component in the grid along with all its properties. Most models are an aggregated version of multiple components of similar type, using simplified parameters. These parameters highlight the most significant properties of the component. The short, medium and long transmission line models are a good example of this simplification in steady state analysis. In dynamic simulations, there are many more models for each type of component. Every software package may also include models that are unique to it. Each dynamic model can be applied to a single bus or to a whole area or zone in the system. This study will only focus on the modeling of the static and dynamic load components.

1.4 Sensitivity Methods

N-1 contingency analysis is a measure of the post-outage voltages and power flows to determine which contingencies push the system limits the furthest. The static load used in these power flows is generally assumed to be a 100% constant power load, where the voltage is regulated to maintain a constant power when the current fluctuates. Our research proves that the N-1 contingency analysis results are dependent on the actual load composition, and since loads types have changed significantly in recent years, the previous assumption is invalid. The end goal is to conduct accurate contingency analysis, but this does not necessarily imply that the load model must be known. Sensitivity techniques combined with the aid of new synchrophasor technology may be used to determine the post-contingency line flows. Sauer [11], [12] re-introduced the method of line outage distribution factors (LODFs) which uses admittance values to estimate the power flow along transmission lines. He later developed a revised technique based on phasor measurements in [12]. This method uses bus injection and power flow measurements instead of elements from the system model such as the Y_{bus} matrix. In [13], Ilic and Phadke used the decoupled method to calculate distribution factors in order to determine reactive power line flows. They studied the relationship between reactive power and voltage magnitudes; however, their errors were greater than 10%. In [14], Singh and Srivastava built on the P-δ and Q-V relationship by calculating distribution factors based on the load flow Jacobian matrix but they still required Y_{bus} matrix. Two new methods founded on these existing methodologies are introduced to estimate post-outage system measurements. These two methods are distribution factors (DFs) and voltage sensitivity analysis. Both of these techniques allow us to compute a ranked N-1 contingency list without knowing the load model parameters.

1.5 Parameter Estimation

In addition to sensitivity techniques, many planning and contingency studies will require performing simulations offline or in a hybrid real-time/off-line environment [15]. An accurate system model will be needed to perform these simulations. The loads will be the most difficult component to model. Each load bus has one or more load models with a specific set of parameters. Static and dynamic loads are modeled differently. The static load is mainly used in power flow studies and the dynamic load, which can be combined with a static load, is used in transient studies. Synchrophasor data will completely change the way we approach load modeling. The high data rate from the PMUs allow the modeling techniques to be applied in almost real-time. When measurements are received at 30 frames per second, the load model parameters can be estimated and the system model can be updated almost instantaneously. The computational time for the parameter estimation process will be the only significant part of the time delay. Many approximation methods were studied for both static and dynamic load modeling. The one method that fared well in all our tests for the static model prediction is a least squares derivative called Bounded Variable Least Squares. The dynamic load is much more complex compared to the static load and after extensive research on non-linear approximation techniques, a machine learning method called Neural Networks is proposed. The development of these algorithms and their implementation on a test system is described in detail in later chapters. All data in this study has been extracted from simulation; however, a PMU algorithm has also been developed so that the simulated measurements realistically represent actual measurements. Load model parameter estimation is the main contribution of this research project.

1.6 Overview of Dissertation

The next chapter briefly reviews the different types of load models and their equations. Chapter 3 describes the computation of N-1 contingency ranking using load models through simulation and compares them with results found through the distribution factor approach and voltage sensitivity method. The detailed derivation of the PMU algorithm and its application to simulated measurements is provided in chapter 4. The Bounded Variable Least Squares method used for static load parameter estimation is illustrated in chapter 5. Followed by Neural Network Machine Learning for dynamic load parameter estimation in chapter 6. Finally, the conclusions and proposed future work are highlighted in chapter 7.

Chapter 2. Review of Load Models and Methods

Loads are composed of many different components such as lighting, heating, air conditioning, motors, and much more. These individual devices are aggregated and represented by highly simplified models of load components [16]. The components are divided into two categories, static and dynamic. The static models are functions of the voltage and/or frequency at any given time, whereas the dynamic models represent loads with time-dependent characteristics. There are two main types of static load models, the ZIP load and the frequency load. However, for dynamic loads, there are many models with different types and compositions [17]. A few of these will be discussed here.

2.1 Static Load Model

The static load model represents the load components that require a constant demand. This model is also used to represent all load types in steady state studies. A basic static load model, as shown in figure 2-1, is the constant impedance (Z), constant current (I), and constant power (P) model, also known as the ZIP model [18]. This model is made up of any composition of the 3 parameters as long as they sum up to 100% of the total load power. There is another load model that is a variation of the ZIP model' this model has a frequency component to it. For this study, we will only consider the ZIP model.

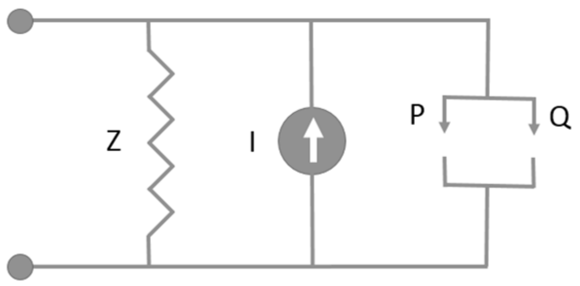


Figure 2-1. Static Load Model

The voltage dependent polynomial model for a static ZIP load is represented using the equations below [19]:

$$P = P_0\,[p_1|V|^2 + p_2|V| + p_3] \tag{2.1}$$

$$Q = Q_0\,[q_1|V|^2 + q_2|V| + q_3] \tag{2.2}$$

P and Q in equations (2.1) and (2.2) refer to the total active and reactive power consumed by the load, respectively. P_0 and Q_0 are the active and reactive power at the initial operating point and |V| is the voltage magnitude at the load bus. p_1 and q_1, p_2 and q_2 and p_3 and q_3 in (2.3) and (2.4) refer to the composition of constant impedance, constant current and constant power, respectively. The following relationship must always be true:

$$p_1 + p_2 + p_3 = 1 \tag{2.3}$$

$$q_1 + q_2 + q_3 = 1 \tag{2.4}$$

2.2 Dynamic Load Model

There are many types of dynamic loads, depending on the components being modeled. The Western Electricity Coordinating Council (WECC) developed an interim complex load model, CLODBL, (see figure 2-2) that is still in use today [17]. This model is an incorrect representation of an actual load bus. It had a fixed 20% induction motor load across the entire system and the load was directly connected to the high voltage transmission bus. Southern

California Edison created a better model to study Fault Induced Delayed Voltage Recovery (FIDVR) events; this model separated the residential air-conditioning load from the rest of the dynamic components (see figure 2-3). The WECC has recently created a new and improved composite load model, CMLDBL, to replace the interim model. This model (found in figure 2-4) differentiates between the various types of motors and has a separate component for electronic and static loads [20].

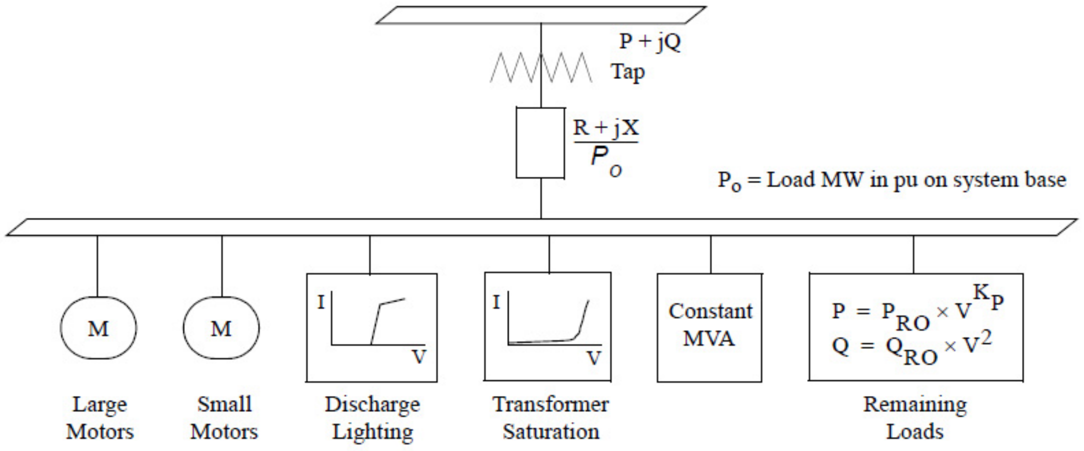


Figure 2-2. WECC Complex Load Model CLODBL (Interim) [21]

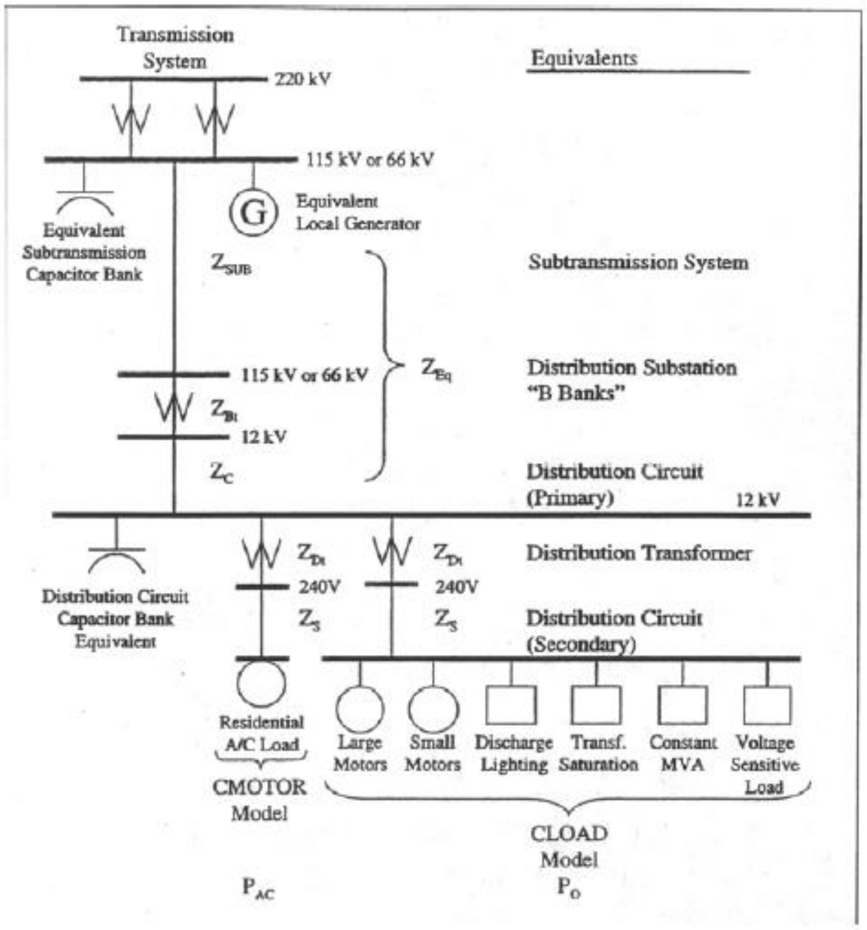


Figure 2-3. SCE Load Model to study FIDVR Events [17]

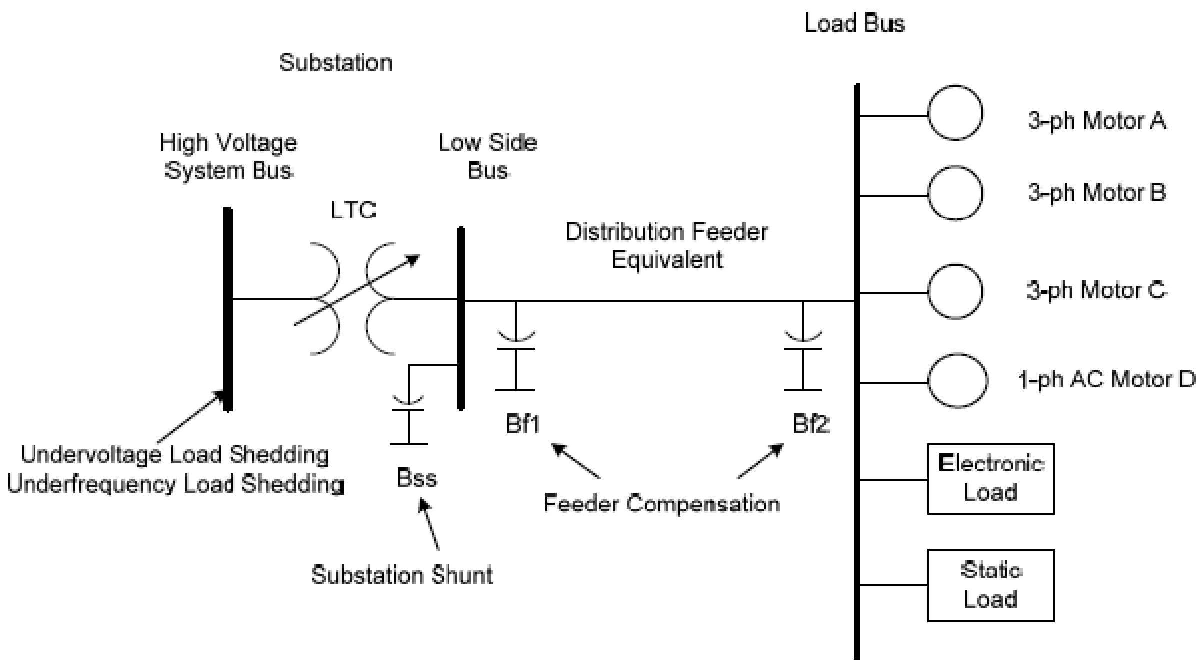


Figure 2-4. WECC Complex Load Model CMLDBL (New) [20]

The general load model that consists of many different components is referred to as the composite load model, as shown in figure 2-5. It parameters include the percent composition of each load component along with a few dynamic properties. There are many versions of this model used throughout the industry.

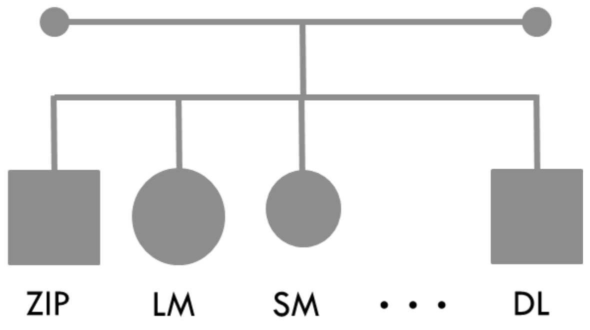


Figure 2-5. Composite Load Model

The component composition equations associated with this type of model are similar to the ZIP model:

$$P = P_0[p_{ZIP} + p_{LM} + p_{SM} + \ \dots + p_{DL}] \tag{2.5}$$

$$Q = Q_0[q_{ZIP} + q_{LM} + q_{SM} + \ \dots + q_{DL}] \tag{2.6}$$

$$p_{ZIP} + p_{LM} + p_{SM} + \cdots + p_{DL} = 1 \tag{2.7}$$

$$q_{ZIP} + q_{LM} + q_{SM} + \cdots + q_{DL} = 1 \tag{2.8}$$

2.3 Induction Motor Model

The induction motor is the most commonly modelled dynamic load component since motors consume over 60% of the power in a system. The motor parameters vary depending on the specific model used. CIM5BL, which is used in this study, consists of 20 parameters, of which a few are described here. The full list of parameters can be found in appendix A. The equivalent circuit of a dual-cage induction motor referred to on the stator side is illustrated in the figure 2-6. The main components of a single cage motor are the stator resistance and reactance (R_s, X_s), the rotor resistance and reactance (R_R, X_R) and the magnetizing reactance (X_m). The dual-cage has an additional rotor resistance and reactance as you can see below. These variables affect the dynamic behaviour of the motor and they have the most impact on the rest of the system during sudden disturbances.

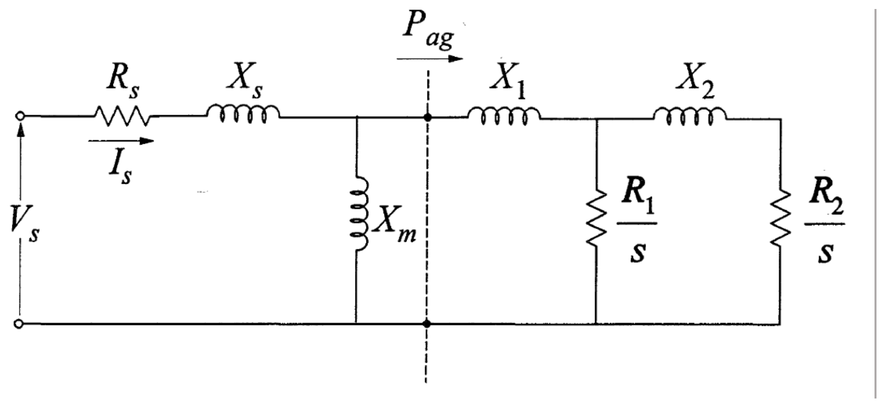


Figure 2-6. Equivalent Circuit of Dual-Cage Induction Motor [22]

Modeling the induction motor can be quite tricky due to its dynamic properties. The actual rotor and stator voltage is calculated using the machine equations in the d-q-o axes and then converted using parks transformation [23]. However, it can be difficult to extract the impedance values from them. An easier method to derive the parameters would be through the equations based on the equivalent circuit**,** at the appropriate voltage (V_B) and apparent power (S) level. These quantities along with inertia (H), and moment of inertia (J) are described in the formulae below. If the equations are not satisfied, then the system to initialize in the simulation environment or it will not converge during the run time. The parameter estimation problem described in chapter 6 will require having a large set of motor parameter combinations. A sample set of machine parameters for a small and large industrial motor are provided in tables 2-1 and 2-2, respectively. A large set of parameter combinations can be derived using these two sample sets and the equations below [24].

Equivalent Impedance:
$$Z_{eq} = \frac{V_S}{I_S} \tag{2-9}$$

$$Z_{eq} = (Rs + jX_S) + (R_r + jX_r) \tag{2-10}$$

Impedance Base:
$$Z_{Base} = \frac{{V_{Base}}^2}{S_{Base}} \tag{2-11}$$

Inertia:
$$H = \frac{{}^1\!/_2 J w^2}{S} \quad \text{where,} \quad w = \frac{4\pi f}{\#\, of\ poles} \qquad \text{(2-12) - (2-13)}$$

Table 2-1. Small Induction Motor Parameters [25]

R_s	X_s	X_m	R_r	X_r	H	m
0.078	0.065	2.67	0.044	0.049	0.5	2.0

Table 2-2. Large Induction Motor Parameters [25]

R_s	X_s	X_m	R_r	X_r	H	m
0.007	0.0409	3.62	0.0062	0.0267	1.6	2.0

In this study, we will refer to the combination of an induction motor and a static load as the complex load model (shown in figure 2-7) [26]. The complete set of parameters for this model include properties of CIM5BL [22] and the percent composition of static load and large induction motor load, respectively. This model will be used for the parameter estimation algorithm later on.

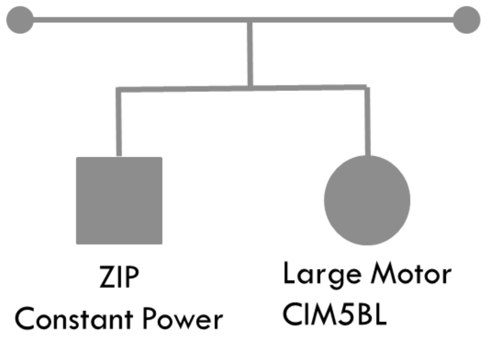


Figure 2-7. Complex Load Model

2.4 Load Modeling Methods

There are two types of methods used to model a load, the component-based and the measurement-based [25]. The component-based, commonly referred to as the bottom-up approach, involves aggregating qualitative data and combining it with statistical data for a given region to determine the types of load components and their respective properties. For example, on a highly simplified level, counting the number of homes on a particular street and estimating the number of air conditioning units and their sizes based on the type and size of the home. An example of this method is illustrated in figure 2-8.

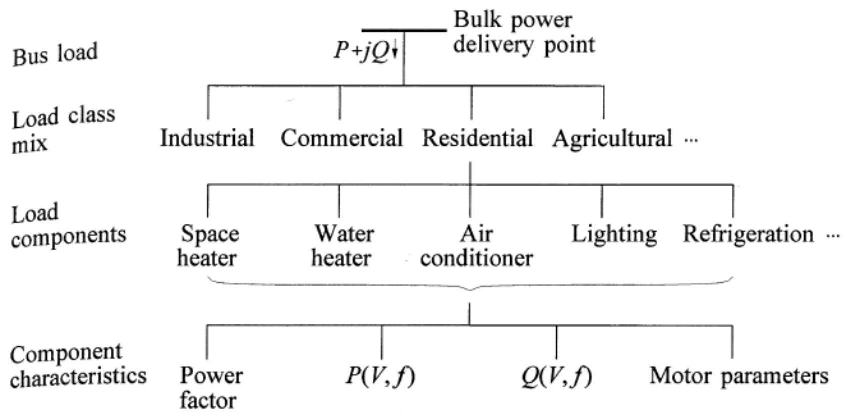


Figure 2-8. Component–Based Approach

The measurement-based method or otherwise known as the top-down approach uses measurements collected over a period of time to determine the types of load components based on their attributed dynamic properties [27]. The measurements are used to study the steady state load voltage characteristics, the steady state load frequency characteristics and the dynamic load voltage characteristics [23], separately, depending on the type of load being modelled. The measured responses can be fitted to the predicted expressions designed for each load [28].

The component-based method has been more popular due to its simplicity and reliability in the past. In recent years, there has been an increasing amount of studies based on the measurement-based approach. One reason for this change is the implementation of the PMU, which has brought in measurements at high data rates [29]. Another reason is the substantial improvements in mathematical based methodologies, such as optimization and data mining.

Chapter 3. N-1 Contingency Ranking

An N-1 contingency refers to the outage of a single transmission line or generator [30]. This type of contingency analysis measures the effect of these outages on the rest of the system in the form of line limit and bus voltage violations [31]. The contingencies are ranked based on their severity and only the highest ranked contingencies are monitored by online Energy Management System (EMS) [32]. This list of contingencies is derived using a combination of load flow simulations on the network model and significant cases extracted from historical data [33]. It is critical to the state of the power grid that all major contingencies are included in this list [34].

3.1 Computation of N-1 Contingency Ranking

3.1.1 Performance Index Formulation

In this study, the ranked contingency list is solely based on N-1 contingency analysis using load flow solutions since the test system has no historical data. This list is computed using a combination of line flow and voltage performance index (PI) functions shown below.

$$PI_{line} = \sum_{j \in S_L} W_j \left[\frac{P_j}{P_{j\,max}} \right]^n \tag{3.1}$$

$$PI_{volt} = \sum_{i \in S_V} W_i \left[\frac{\left| V_i - V_i^{lim} \right|}{V_i^{lim}} \right] \tag{3.2}$$

$$PI_{total} = PI_{line} + PI_{volt} \tag{3.3}$$

The line performance index, PI_{line}, in equation (3.1) is calculated using load flows that exceed their transmission line limits. P_j is the power flow of the line during the contingency and $P_{j\,max}$ is the line limit. W_j is the weight given to that particular line and n is a suitable index. The voltage performance index, PI_{volt}, in equation (3.2) is a function of the bus voltages that are outside their acceptable range. V_i is the voltage of the bus that is violating its bounds and $V_{i\ lim}$ is the upper or lower voltage limit that is being exceeded in per unit. W_i is the weight given to that particular bus [35]. The weights used for the buses and transmission lines are based on their severity. These PIs are recalculated when there are changes in the topology of the system. The total performance index, PI_{total}, in equation (3.3) is the sum of both line and voltage performance indices. Each line or generator outage is applied independently and the PI_{total} of each outage is ranked from highest to lowest. Only the outages corresponding to the highest ranking PI_{total} are saved in the contingency list used by EMS.

3.1.2 Contingency Ranking - ZIP Load Model Base Case

In the base case, the power flow is calculated after each N-1 outage and their corresponding bus voltages and line flows are used in calculating the performance indices. As mentioned in the previous chapter, all loads are configured to consume 100% constant power during contingency analysis, based on the assumption that the load model does not significantly affect the post-outage results.

In order to test this theory, the performance indices methodology is applied to the IEEE 118 Bus System [36] (figure 3-1) by simulating outages using the PSLF power flow package [16]. Each set of results were based on one of the three unique combination of load model parameters, 100% constant P, 100% constant I, and 100% constant Y, at bus 59. The post-outage line flows and bus voltages were compared to their respective limits. The total performance indices were computed and ranked according to equations (3.1), (3.2) and (3.3). A uniform random number generator was used to assign weights, W_i and W_j, for each bus and line, respectively. The PSLF code written to run the load flow cases is included in appendix B.1 and the MATLAB code developed to compute the ranking is provided in appendix B.2.

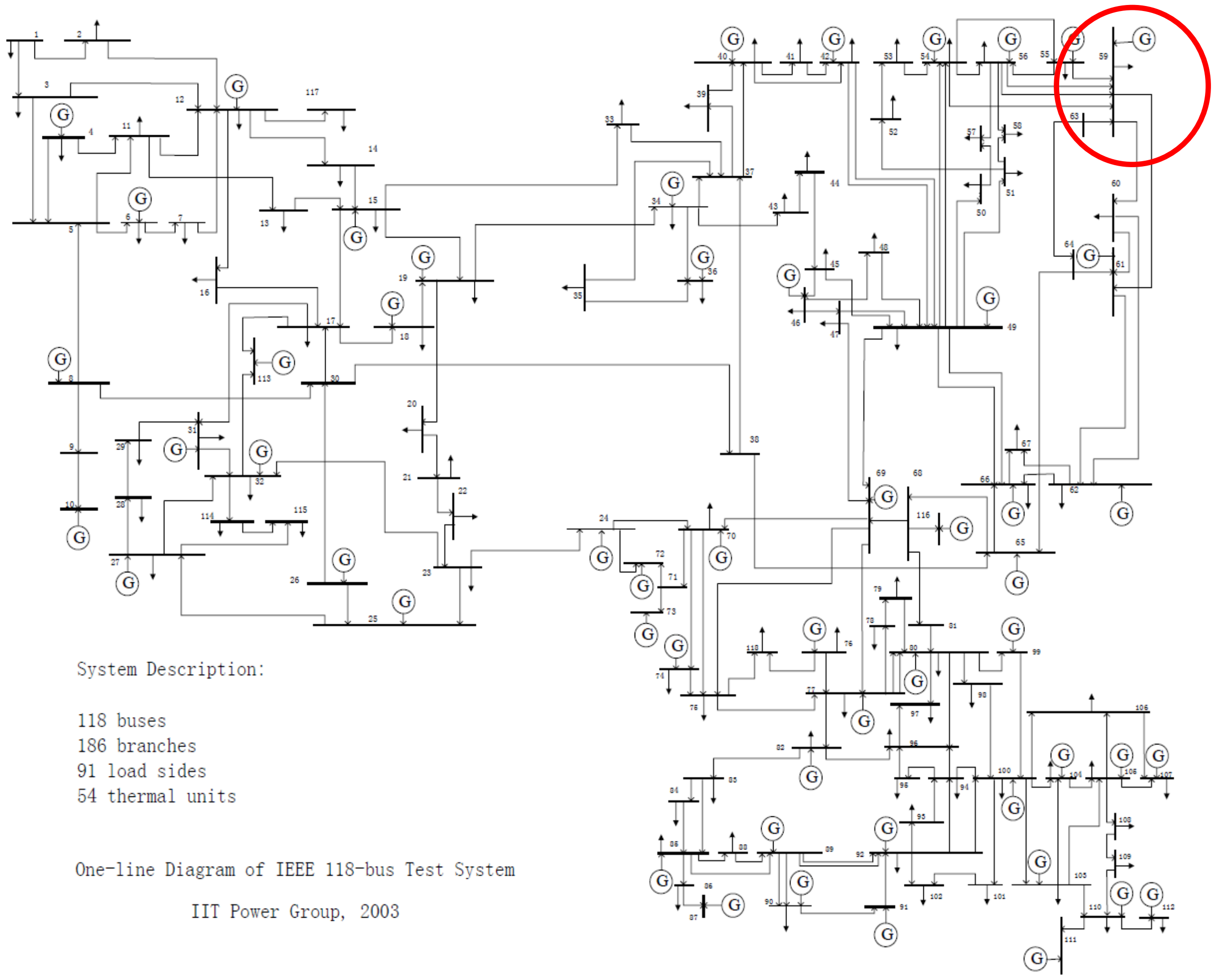


Figure 3-1. IEEE 118 Bus Test System [36]

The top ranking 35 out of 231 contingencies for constant P, constant I, and constant Z load models are displayed in table 3-1. The line outages are illustrated using its corresponding bus numbers; for example, the line outage between buses 75 and 118 is listed as “75 118”. Similarly, the generator outages are displayed using the generator bus number; for example, generator at bus 100 is “G 100”. The table shows that the contingencies are not ranked in the same order for each parameter set. Additionally, the 34th ranked contingency, line 22-23, for the constant I, and 32nd for constant Z load model does not appear in the constant P ranking whereas line 63 – 64 in the constant P model does not appear in the other two rankings. This shows that the load model parameters can have a significant effect on the post-outage lines flows and bus voltages. In this particular case, line 24 – 70 causes a bus voltage violation at bus 91 when the load is modelled as a constant I or constant Z load but not when it is a constant P load. This contingency may have been overlooked if the load was modelled incorrectly on an actual system. This test provides us with the results of only changing the load model at a single bus; these ranked lists will change even drastically when the load composition differs at every single bus.

Table 3-1. Load Flow Based Contingency List

Rank	Constant Power		Constant Current		Constant Impedance	
1	G	100	G	80	G	80
2	75	118	23	24	9	10
3	G	10	G	10	69	70
4	38	65	68	69	38	65
5	65	68	5	8	68	69
6	5	8	38	65	65	68
7	G	89	8	9	G	26
8	G	26	G	89	75	118
9	9	10	65	68	5	8
10	G	80	75	118	8	9
11	68	69	9	10	G	89
12	69	70	69	70	G	10
13	8	9	G	26	80	81
14	23	24	G	100	68	81
15	68	81	68	81	G	100
16	G	25	64	65	G	5
17	80	81	G	25	64	65
18	37	38	80	81	37	38
19	64	65	37	38	G	49
20	G	49	G	49	69	75
21	69	75	69	75	G	66
22	G	66	G	66	G	21
23	G	54	G	12	30	38
24	G	12	G	54	G	54
25	30	38	30	38	G	65
26	G	65	G	65	25	26
27	25	26	25	26	69	77
28	69	77	69	77	100	103
29	100	103	100	103	76	77
30	76	77	76	77	26	30
31	26	30	26	30	G	61
32	G	61	G	61	22	23
33	G	59	G	59	G	59
34	63	64	22	23	24	70
35	59	63	24	70	45	46

3.2 Distribution Factor Method

Computing power flows for every single contingency can be computationally intensive and time consuming, especially for a large system. Other methods to find the worst-case contingencies without executing N number of power flow calculations were explored. These methods were based on collecting high data rate measurements such as synchrophasor data. One such method is the distribution factor approach [37], which uses PMU data to compute power flow sensitivities [38] to estimate the system state after a line outage. Sensitivity calculations can provide the relative impact of a particular change in the system using injection shift factors (ISFs) and line outage distribution factors (LODFs) [39]. This distribution factor was tested to extrinsically compute a contingency ranking without knowing the load model composition of a system.

3.2.1 Computation of Injection Shift Factors

Injection shift factors (ISFs) in equation (3.4) represent the amount of change in power flow, relative to the change in injection at a particular bus. $\Delta P_{k-l}^{i}(t)$ is the change in flow on line *k – l* due to the an injection at bus *i* over a period of time t and $\Delta P_i(t)$ is the change in injection at bus *i* over time t.

$$\Psi_{k-l}^{i} = \frac{\Delta P_{k-l}^{i}(t)}{\Delta P_i(t)} \qquad (3.4)$$

The ISFs can be derived using measurements of the bus injections and line flows of a system as long as the number of measurements is greater than the rank of the system. Equation (3.5) is the matrix representation of the relationship between injection and load flow measurements and the ISFs for each line [12]. The inverse can be taken to solve for Ψ_{k-l}^{i} :

$$\Delta P_{k-l} = \Delta P_i \Psi_{k-l}^{i} \qquad (3.5)$$

3.2.2 Computation of Line Outage Distribution Factors

A line outage distribution factor (LODF) represents the percentage of power flow on a line before the outage that is redistributed onto another line. For example, if line m – n had an outage and this caused a change in power flow on line k – l, the new flow on line k – l would be based on the respective LODF (Ξ). The LODF is a function of the ISFs and can be calculated using equation (3.6) [12].

$$\Xi_{k-l}^{m-n} = \frac{\Psi_{k-l}^{m} - \Psi_{k-l}^{n}}{1 - (\Psi_{m-n}^{m} - \Psi_{m-n}^{n})} \qquad (3.6)$$

The post-outage power flow along the line k-l is a sum of the pre-outage flow (along k-l) and the LODF proportion of the pre-outage flow along the faulted line m-n:

$$P_{k-l}^{post-outage} = P_{k-l}^{pre-outage} + \Xi_{k-l}^{m-n} * P_{m-n}^{pre-outage} \qquad (3.7)$$

The LODF can be calculated successfully for most line outages. A few exceptions to the LODF equation are when the line outage causes islanding or if the line is radial. If the line outage causes islanding, it is quite difficult to predict what kind of effect the disconnected part of the system can have on the rest of the network. In this case, the LODF is assumed to be zero [40]. When the disconnected line is a radial line, the bus that connected it to the rest of the system is now treated as an injection. For these reasons, the contingency ranking in the following case study will only be conducted on non-radial and non-islanding lines.

3.2.3 Contingency Ranking – Distribution Factor Method (Case 2)

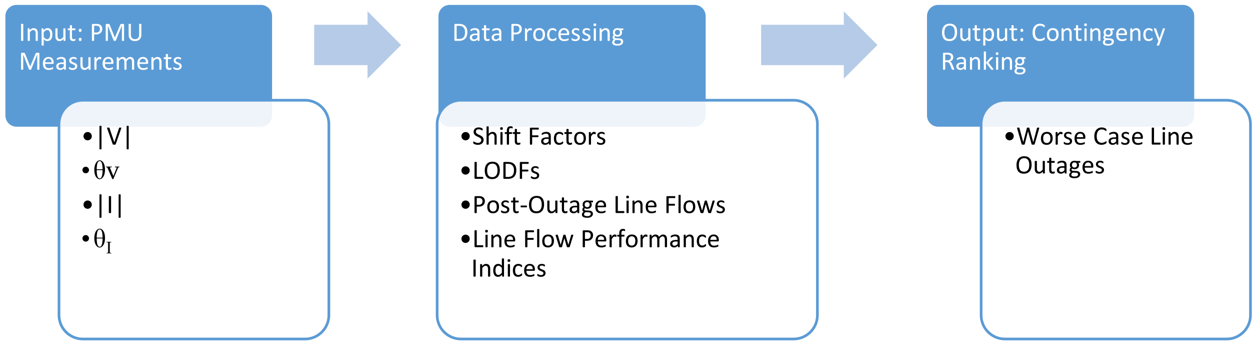


Figure 3-2. Contingency Ranking using DFs

In case 2, the contingency ranking of the IEEE 118 bus system is determined using the distribution factor method. This process is illustrated in figure 3-2. Similar to the base case, the load parameters at bus 59 are the variables. Simulated PMU data is collected over a short period where loads are scaled in order to create some variation in the power flows. The power flows and injections for the whole system are required to compute the ISFs and LODFs using equations (3.5) and (3.6), respectively [41] [42]. A unique matrix of shift factors are found for each of the 3 sets of load model parameters, 100% constant P, 100% constant I and 100% constant Z. The post-outage line flows for each transmission line are estimated using equation (3.7). The line flow and voltage performance indices for each line outage is calculated using equations (3.1) – (3.3). The index totals are used to rank the worst-case contingencies. The PSLF code written to simulate PMU data and the MATLAB code developed to calculate DFs and the ranking are included in appendix C1 and C2, respectively.

The post-outage line flows are estimated before the contingency ranking is calculated. These estimates are compared to the actual post-outage flows to see how close our predictions can be. The actual post-outage flow is found by computing the powerflow of the system after the line outage takes place. A few line flows for the outage of line for the constant power load model are compared in table 3-2 and another set of line flows are compared for the outage of line 9-10 in table 3-3. The error is calculated relative to the actual powerflow measurement. The predictions for the line 44-45 outage seem to be good, with errors less than 10%; however, the predictions for the outage of line 5-11 are completely unreasonable, with some errors higher than 70%. This shows that the line outage distribution factors may be accurate for a subset of outages, but not all.

Table 3-2. Estimated Post-Outage Powerflow – Line 44-45

Line	Actual Post-Outage Line Flow (MW)	Estimated Post-Outage Line Flow (MW)	Error (%) $\lvert MW_{Actual} - MW_{Estimate} \rvert$
35-36	23.483	24.514	4.392
59-60	-57.649	-61.891	7.359

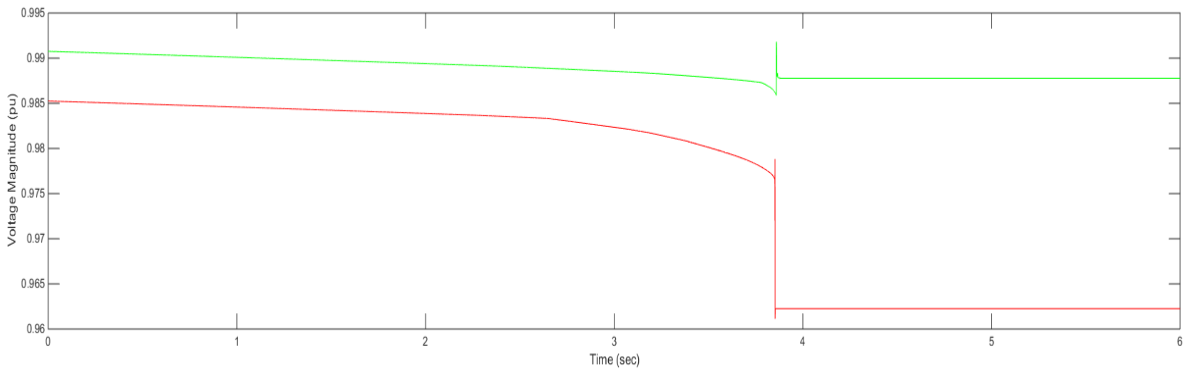
Table 3-3. Estimated Post-Outage Powerflow – Line 5-11

Bus Number	Actual Post-Outage Line Flow (MW)	Estimated Post-Outage Line Flow (MW)	Error (%) $\lvert MW_{Actual} - MW_{Estimate} \rvert$
13-15	90.621	26.384	70.886
8-9	-93.245	-59.109	36.608

The next step is to find the contingency ranking using the distribution factor estimates. The ranking for constant P, constant I, and constant Z loads using the DF method is compared to the constant P results from the base case in table 3-4. This ranking is performed on a smaller subset of the total transmission lines in the system since radial and islanding lines were eliminated. A slight change in ranking can be seen between the different load models; but no contingencies are missing in any of the DF lists when compared against each other. There are however significant differences between the base case list in the first column and the DF constant power list in the second column. There are four line outages that cause high-ranking violations that are missing from the DF version. The contingency ranking derived using distribution factors only considers line flow violations due to line outages; whereas, load model changes seem to have a smaller impact on line flows compared to bus voltages. This can be seen in figure 3-3 where the voltage magnitude and power flow captured at the load bus is plotted for two different load models. The system undergoes the same sequence of events with both load models.

Table 3-4. Distribution Factor Based Contingency List

Rank	Base Case Constant Power	DF Method Constant Power	DF Method Constant Current	DF Method Constant Impedance
1	75 - 118	75 - 118	75 - 118	69 - 70
2	38 -65	38 - 65	38 - 65	38 - 65
3	65 -68	65 - 68	65 - 68	65 - 68
4	8 - 9	69 - 70	69 - 70	75 - 118
5	9 - 10	23 - 24	23 - 24	69 - 77
6	69 - 70	69 - 77	69 - 77	69 - 75
7	23 - 24	69 - 75	69 - 75	47 - 69
8	64- 65	47 - 69	47 - 69	49 - 69
9	100 - 103	49 - 69	49 - 69	23 - 24
10	69 - 77	64 - 65	64 - 65	26 - 30
11	63 - 64	26 - 30	26 - 30	64 - 65
12	26 - 30	70 - 71	70 - 71	70 - 71
13	69 - 75	24 - 70	24 - 70	24 - 70
14	24 - 70	71 - 72	71 - 72	71 - 72
15	42 - 49	100 - 103	100 - 103	100 - 103



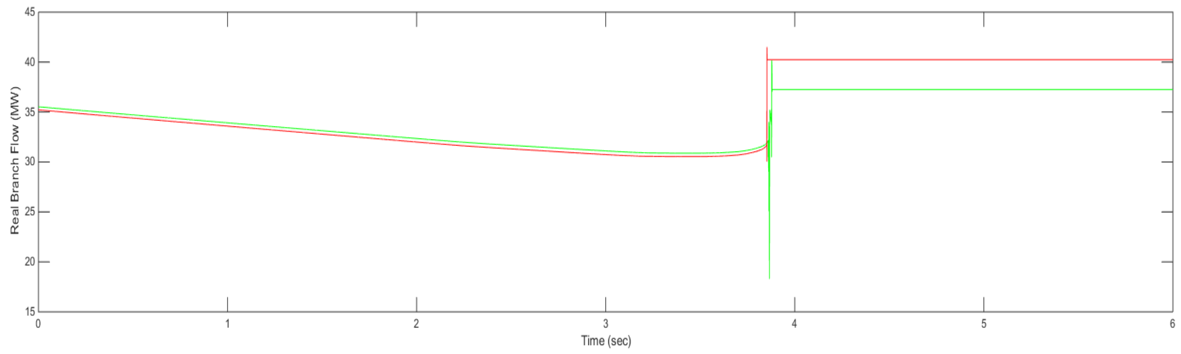


Figure 3-3. (a) Voltage Magnitude and (b) Real Powerflow – Line 11-13

A few significant limitations prevent the LODF method from becoming a reliable method for contingency ranking. It is not accurate enough to track changes between different load compositions, which is primarily due to the lack of voltage sensitivities [43]. The system state after a generator outage cannot be estimated and there is a subset of lines where the LODF cannot be computed. Another limitation to this method is the amount of PMU measurements required for the distribution factors computation Currently, this method cannot be implemented due to the lack of PMUs; however, this may not be an issue in the future with increasing PMU placement.

3.3 Voltage Sensitivity Method

Sensitivity and distribution factor (DF) methods have long been used to determine post-outage quantities in performing contingency analysis. Traditionally, sensitivity ratios and distribution factors are computed using the network admittances and the system measurements at the operating point. In order to be independent of system topology and models, the sensitivity matrix needs to be derived without knowledge of the admittance matrix (Y_{bus}). PMUs can fill in this gap by providing measurements that can be used to calculate voltage sensitivity ratios directly. Voltage magnitudes can be computed using voltage sensitivities similar to that in [14]. However, the voltage sensitivity matrix will be derived using phasor measurements instead of the admittance values. The sensitivities will reflect all static and dynamic components in the system in its most recent state.

Voltage sensitivities represent the change in voltage magnitude or angle [44], at bus i, with respect to a change in power injection, at bus j, while all other quantities are constant. The four types of quantities represented in a voltage sensitivity matrix are $\partial V/\partial P$, $\partial V/\partial Q$, $\partial\theta/\partial P$, and $\partial\theta/\partial Q$. In [13], the Q-V decoupled method was used to derive the sensitivity ratios. The decoupling can be effective for small systems but these quantities are more interdependent as the size of the system grows. This method also provides normalized errors that range from 7% to 24%, which is quite high. In [14], the measurements are not decoupled; in fact, the individual effect of real and reactive power on both voltage magnitude and angle are decomposed into two sets of equations that provide the four quantities listed above. This method can be applied to any size system as long as its Jacobian can be computed [45].

3.3.1 Jacobian-Based Sensitivity Computation

The Jacobian, J, is made up of four quadrants, where n is the number of buses and np is the number of P-Q buses. The size of J is (2n-np-1) x (2n-np-1) and bus 1 is assumed to be the slack bus.

$$[J] = \begin{bmatrix} J1 & J2 \\ J3 & J4 \end{bmatrix} = \begin{bmatrix} \begin{bmatrix} \partial P_2/\partial\theta_2 & \cdots & \partial P_2/\partial\theta_n \\ \vdots & \ddots & \vdots \\ \partial P_n/\partial\theta_2 & \cdots & \partial P_n/\partial\theta_n \end{bmatrix} & \begin{bmatrix} \partial P_2/\partial V_2 & \cdots & \partial P_2/\partial V_{1+np} \\ \vdots & \ddots & \vdots \\ \partial P_n/\partial V_2 & \cdots & \partial P_n/\partial V_{1+np} \end{bmatrix} \\ \begin{bmatrix} \partial Q_2/\partial\theta_2 & \cdots & \partial Q_2/\partial\theta_n \\ \vdots & \ddots & \vdots \\ \partial Q_{1+np}/\partial\theta_2 & \cdots & \partial Q_{1+np}/\partial\theta_n \end{bmatrix} & \begin{bmatrix} \partial Q_2/\partial V_2 & \cdots & \partial Q_2/\partial V_{1+np} \\ \vdots & \ddots & \vdots \\ \partial Q_{1+np}/\partial V_2 & \cdots & \partial Q_{1+np}/\partial V_{1+np} \end{bmatrix} \end{bmatrix}$$

(3.8)

The extended Jacobian, J*, used in sensitivity studies treats P-V buses like P-Q type buses so its size is (2n-2) x (2n-2). The slack bus is not included in the Jacobian, hence the -2.

The Newton-Raphson load flow equations [46] use the Jacobian to relate power mismatch to voltage corrections. The equation below illustrates this relationship [14]:

$$\begin{bmatrix}\Delta P\\ \Delta Q\end{bmatrix} = [J^*]\begin{bmatrix}\Delta\theta\\ \Delta V\end{bmatrix} \qquad (3.9)$$

To produce the entries in each Jacobian sub-matrix, the following simplified load flow equations are used, where ik and ii refer to off-diagonal and diagonal entries, respectively [47]:

$J1_{ik} = -|Y_{ik}V_iV_k|\sin(\theta_{ik} + \delta_k - \delta_i)$ $\qquad J2_{ik} = |Y_{ik}V_iV_k|\cos(\theta_{ik} - \delta_k - \delta_i)$

$J1_{ii} = -Q_i - V_i^2B_{ii}$ $\qquad J2_{ii} = P_i + V_i^2G_{ii}$

$J3_{ik} = -|Y_{ik}V_iV_k|\cos(\theta_{ik} - \delta_k - \delta_i)$ $\qquad J4_{ik} = -Q_i - V_i^2B_{ii}$

$J3_{ii} = P_i - V_i^2G_{ii}$ $\qquad J4_{ii} = Q_i - V_i^2B_{ii}$ $\qquad (3.10)$

The equations are in polar coordinates, where Y_{ik} refers to the bus admittance matrix elements, θ_{ik} and δ_i refer to the admittance angle and voltage angle, respectively. P_i, Q_i, V_i and δ_i are measurements at the base operating point.

The voltage sensitivity matrix is defined as $[S] = [J^*]^{-1}$ and by using the Moore–Penrose formula for pseudoinverse, the following equation can be formed to relate change in injection with change in voltage:

$$\begin{bmatrix}\Delta\theta\\ \Delta V\end{bmatrix} = [S]\begin{bmatrix}\Delta P\\ \Delta Q\end{bmatrix} \qquad (3.11)$$

3.3.2 Phasor Measurement-Based Sensitivity Computation

The Jacobian-based voltage sensitivity matrix requires knowledge of the network topology, specifically the Y_{bus}, to compute the load flow equations. Today, with the aid of wide area measurement systems (WAMS), voltage sensitivities can be determined using actual measurements instead of approximate system models. The voltage sensitivity matrix, [S], can be formed using the measurement relationships described in section 2.3.1, without having to find $[J^*]$. The pseudoinverse is applied to equation (3.11) to get (3.12).

$$[S]\begin{bmatrix}\Delta P\\ \Delta Q\end{bmatrix}\begin{bmatrix}\Delta P\\ \Delta Q\end{bmatrix}^T = \begin{bmatrix}\Delta\theta\\ \Delta V\end{bmatrix}\begin{bmatrix}\Delta P\\ \Delta Q\end{bmatrix}^T \quad \Rightarrow \quad [S] = \begin{bmatrix}\Delta\theta\\ \Delta V\end{bmatrix}\begin{bmatrix}\Delta P\\ \Delta Q\end{bmatrix}^T\left(\begin{bmatrix}\Delta P\\ \Delta Q\end{bmatrix}\begin{bmatrix}\Delta P\\ \Delta Q\end{bmatrix}^{T^{-1}}\right) \tag{3.12}$$

As the inverse of the Jacobian, the [S] matrix produced in (3.12) is a measure of the change in voltage over the change in injection at each bus shown in equation (3.13).

$$[S] = \begin{bmatrix} \begin{bmatrix} \partial\theta_2/\partial P_2 & \cdots & \partial\theta_2/\partial P_n \\ \vdots & \ddots & \vdots \\ \partial\theta_n/\partial P_2 & \cdots & \partial\theta_n/\partial P_n \end{bmatrix} & \begin{bmatrix} \partial\theta_2/\partial Q_2 & \cdots & \partial\theta_2/\partial Q_n \\ \vdots & \ddots & \vdots \\ \partial\theta_n/\partial Q_2 & \cdots & \partial\theta_n/\partial Q_n \end{bmatrix} \\ \begin{bmatrix} \partial V_2/\partial P_2 & \cdots & \partial V_2/\partial P_n \\ \vdots & \ddots & \vdots \\ \partial V_n/\partial P_2 & \cdots & \partial V_n/\partial P_n \end{bmatrix} & \begin{bmatrix} \partial V_2/\partial Q_2 & \cdots & \partial V_2/\partial Q_n \\ \vdots & \ddots & \vdots \\ \partial V_n/\partial Q_2 & \cdots & \partial V_n/\partial Q_n \end{bmatrix} \end{bmatrix} \tag{3.13}$$

This relationship can be used to predict the change in bus voltage due to a change in injection. In order to estimate the change in voltage at each bus after a transmission line or generator outage, the change in injections, ΔP, must be approximated.

$$\Delta P = P_{\text{post-outage}} - P_{\text{pre-outage}} \tag{3.14}$$

Since the pre-outage [S]matrix is used to make the post-contingency predictions, the line and generator outages must translate into changes in injections at the respective buses. The outage of line i-j implies that the pre-outage line flows, P_{ij}, P_{ji} Q_{ij}, and Q_{ji}, are lost and this loss is equivalent to the change in injections at both ends of the line [14]. The change in bus injections due to the outage of line i-j are:

$$\Delta P_i = P_{ij} \qquad \Delta P_j = P_{ji} \qquad \Delta Q_i = Q_{ij} \qquad \Delta Q_j = Q_{ji} \tag{3.15}$$

The generator outage is simulated by treating the loss of generation as a change in bus injection. The generator outage at bus i is represented by:

$$\Delta P_i = -\Delta P_{Gi} \qquad \Delta Q_i = -\Delta Q_{Gi} \qquad (3.16)$$

With such large power mismatches, the change in voltage is decomposed into its real and reactive components shown in equation (3.17) and the post-outage voltage magnitudes and phase angles are estimated using equation (3.18). The voltage performance indices for each contingency is calculated using these results.

$$\begin{bmatrix} \Delta\theta^P \\ \Delta V^P \end{bmatrix} = [S]\begin{bmatrix} \Delta P \\ 0 \end{bmatrix} \qquad \begin{bmatrix} \Delta\theta^Q \\ \Delta V^Q \end{bmatrix} = [S]\begin{bmatrix} 0 \\ \Delta Q \end{bmatrix} \qquad (3.17)$$

$$\theta_{post-outage} = \theta_{pre-outage} + \Delta\theta^P + \Delta\theta^Q$$

$$V_{post-outage} = V_{pre-outage} + \Delta V^P + \Delta V^Q \qquad (3.18)$$

3.3.3 Contingency Ranking – Voltage Sensitivity Method (Case 3)

The contingency ranking in case 3 is based on the calculated voltage sensitivities. Once [S] is computed, the matrix can be multiplied with a unique vector that corresponds to a particular contingency (equation 3.17) and the post-outage estimates can be determined. The performance indices are calculated and used to extract the list of worst-case contingencies. This process is illustrated in figure 3-4 below.

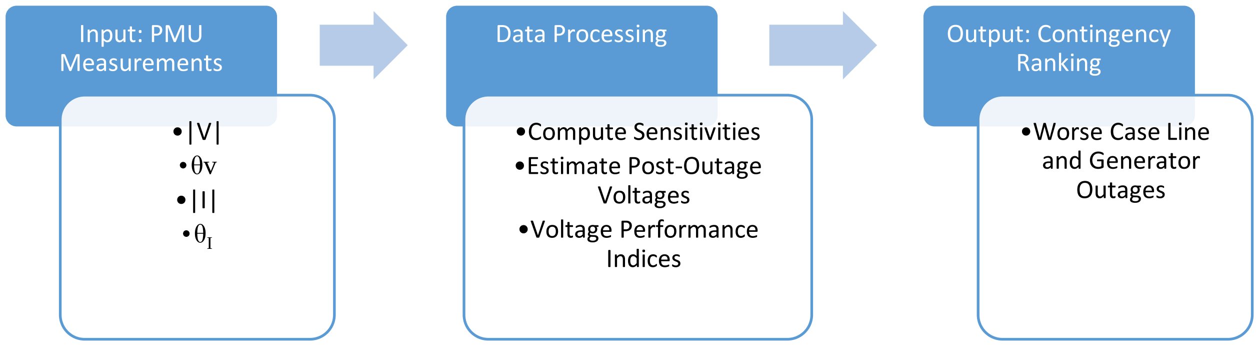

Input: PMU Measurements
•|V|
•θ_V
•|I|
•θ_I
Data Processing
•Compute Sensitivities
•Estimate Post-Outage Voltages
•Voltage Performance Indices
Output: Contingency Ranking
•Worse Case Line and Generator Outages

Figure 3-4. Contingency Ranking using Voltage Sensitivities

The voltage sensitivity method is applied to the IEEE 118 Bus System (figure 3-1) to compute the worst-case contingency list. Two types of contingencies were tested in this study, transmission line outage and generator outage. Many tests were run to assess this estimation method. The base operating point of each test was changed by scaling the load by 100%-130% before the contingency. Bus voltages and injections were collected for approximately 300 seconds in which every 5 seconds the total load would be scaled up by a random increment. The cumulative total load scale is no more than 160% of the original load to imitate the daily load curve. Data was collected at a rate of 4 samples per second so that in total each measurement had 1200 samples. The voltage and injection matrices were oversampled to reduce error during least squares computation and to prevent a singularity due to the small variation in measurement data. The Method of Least Squares (LS) is applied to equation (3.12) to compute the sensitivity matrix. No changes in topology or load models are made during this time. Multiple sets of random load increments can be used to create variation in the measured quantities. Since the load is scaled and LS is applied, the derived sensitivity matrix is an average of multiple operating points. The PSLF code written to simulate PMU data and the MATLAB code developed to calculate the sensitivities and ranking are included in appendix D1 and D2, respectively.

The post-contingency voltages are estimated before the contingency ranking is calculated. These estimates are compared to the actual post-outage voltages to see how close our predictions can be. The actual post-outage voltage is found by computing the powerflow of the system after the line or generator outage takes place. A few bus voltages for the outage of a generator at bus 56 for the constant power load model are compared in table 3-5 and another set of voltages are compared for the outage of line 9-10 in table 3-6. The voltage predictions for the bus 56 generator outage are very good, with errors less than $5x10^{-3}$; however, the predictions for the outage of line 9-10 are completely unreasonable, with some erros as high as 63%. This shows that the voltage sensitivities may be accurate for a subset of contingencies, but not all. The next step is to find the contingency ranking using the voltage sensitivity estimates.

Table 3-5. Estimated Voltage after Generator Outage at Bus 56

Bus Number	Actual Post-Outage Voltage (p.u.)	Estimated Post-Outage Voltage (p.u.)	Error (%) $\lvert V_{Actual} - V_{Estimate} \rvert$
10	1.0499	1.05	$4.3\text{x}10^{-3}$
27	0.967	0.968	$2.0\text{x}10^{-3}$
72	0.9801	0.98	$3.3\text{x}10^{-3}$
90	0.9849	0.985	$3.3\text{x}10^{-3}$

Table 3-6. Estimated Voltage after Line Outage between Buses 9-10

Bus Number	Actual Post-Outage Voltage (p.u.)	Estimated Post-Outage Voltage (p.u.)	Error (%) $\lvert V_{Actual} - V_{Estimate} \rvert$
18	1.0232	0.3863	63.688
37	1.0500	0.5413	50.871
50	1.0400	0.4085	63.162
118	0.9692	1.5507	58.161

Comparing the contingency rankings from the voltage estimates with the load flow solutions is a good indicator of the overall accuracy of the sensitivity approach. The results from this method are shown in table 3-7. The first column is the ranking for the base case only using the voltage performance index and the second column is the voltage sensitivity based list. The 100% constant power load model is used in both cases. Similar to the base case, "G 100" refers to the loss of generator at bus 100 and "75 118" refers to the loss of transmission line between buses 75 and 118. The grey highlighted cells in the table are the contingencies that appear in both lists. This shows that approximately 50% of the contingencies that occur in the base case are not detected using the sensitivities even though both lists are only based on voltage violations.

Table 3-7. Voltage Sensitivity Based Contingency List

Rank	Base Case Constant Power		Voltage Sensitivity Constant Power	
1	G	100	G	89
2	75	118	G	26
3	G	10	G	10

4	38	65	G	80
5	65	68	G	66
6	G	89	G	25
7	G	26	9	10
8	9	10	25	27
9	G	80	G	69
10	69	70	26	30
11	8	9	23	25
12	23	24	G	65
13	68	81	8	9
14	G	25	G	100
15	69	75	49	51
16	G	54	23	32
17	G	49	22	23
18	64	65	69	70
19	76	77	38	65
20	30	38	45	49
21	G	66	37	39
22	22	23	12	117
23	45	46	3	5
24	G	73	G	59
25	G	46	103	110

The sensitivity method makes a bold assumption that line outages can be represented as a change in injection at two buses; however, this does not always hold true. Voltage sensitivities make poor estimates in cases where the outage is severe such as a large generator or a heavily loaded line. In these conditions, the power system finds other ways to balance itself to return to a secure state. Sensitivities and distribution factors are good for rough estimates of voltage and line flows but not accurate enough for contingency analysis. N-1 contingency analysis may not be possible without accurate load models and power flow computations.

Chapter 4. Phasor Measurement Unit Algorithm

The majority of power systems research is performed using simulated data as it is very difficult to obtain a large enough database of real data, especially system measurements. Once the algorithms have been developed and tested in a simulation environment, it is applied to an actual system and is expected to behave as predicted. In order to develop a reliable parameter estimation method for static and dynamic load models, the measurement data sets used in training must represent real PMU data.

4.1 Simulation vs. Real PMU Data

The data produced using a simulation software such as General Electric's PSLF or Siemen's PSS/E is recorded at a very high data rate with a short sampling window that does not represent a PMU. Figure 4-1 (a) is the plot of a response to a real three-phase fault on a high voltage transmission line. The plot shows the voltage magnitude at a bus approximately 60 miles from the fault. The curve starts with the pre-fault voltage, followed by a drop when the fault occurs. The voltage goes back up when the breaker is opened but then the voltage drops when the breaker re-closes back into the fault. The breaker then re-opens and remains open for the duration of the fault. Figure 4-1 (b) is the simulated fault response on the IEEE 118 bus system using PSS/E. As you can see, the simulated curve has a much sharper drop and jump in voltage; the flat trough lines tell us that the sampling window is quite narrow. For this reason, a PMU algorithm is created so that it can convert the simulated data into realistic PMU measurements.

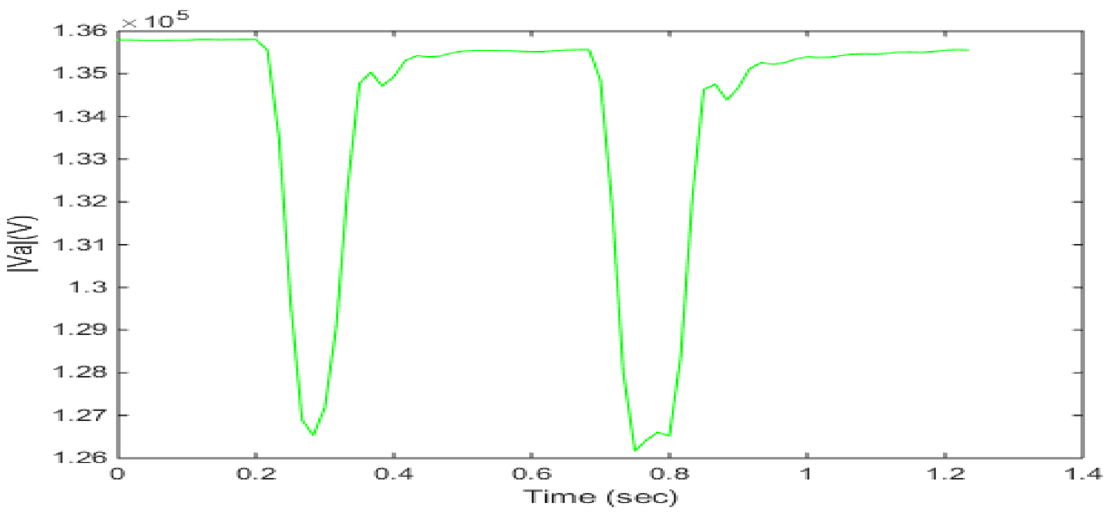


Figure 4-1 (a). Fault Response – Real PMU Data

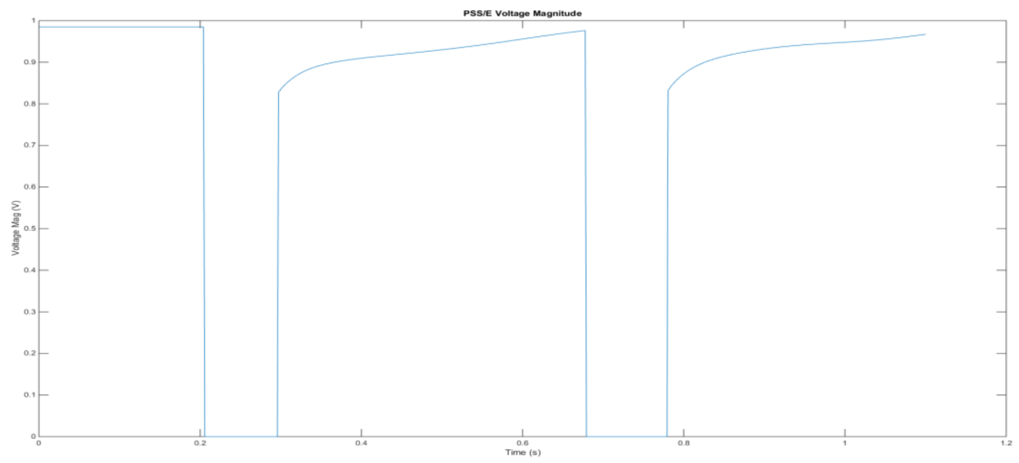


Figure 4-1 (a). Fault Response – PSS/E Simulated PMU Data

4.2 PMU Algorithm Development

In order to develop this algorithm, the behaviour of a real PMU was studied; specifically, the steps the PMU takes to convert its raw measurements into phasors. A significant event with enough variation in measurements was needed to truly test our method. We were able to obtain data from a PMU and a digital fault recorder (DFR) during a three-phase fault. The fault recorder was closest to the fault and the PMU was approximately 60 miles away on a direct line as shown in figure 4-2.

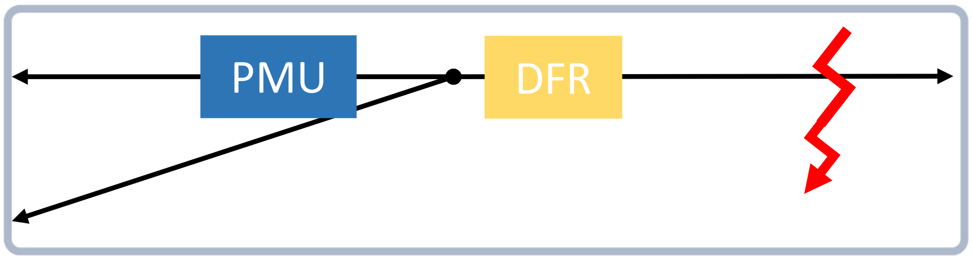


Figure 4-2. Fault Location and Measuring Devices

The PMU is assumed to receive raw measurements similar to the DFR and then apply a series of conversions to output the phasors. Figure 4-3 illustrates these steps in sequence. Step one includes receiving the analog measurements at a high data rate and then steps two to five represent the analog-to-digital converter [48] (ADC). The ADC filters the data at the Nyquist frequency and then down-samples it to the PMU frequency of 30 or 60 frames per second. The phasor is computed by applying a Discrete Fourier Transform (DFT) across a window and the output is quantized before being placed in the synchrophasor data frame [49]. The specifications of each PMU may vary depending on the manufacturer, but this is the fundamental method that most PMUs are based on. The fault data obtained was from a Schweitzer SEL PMU, so the algorithm developed here will be specific to this piece of hardware.

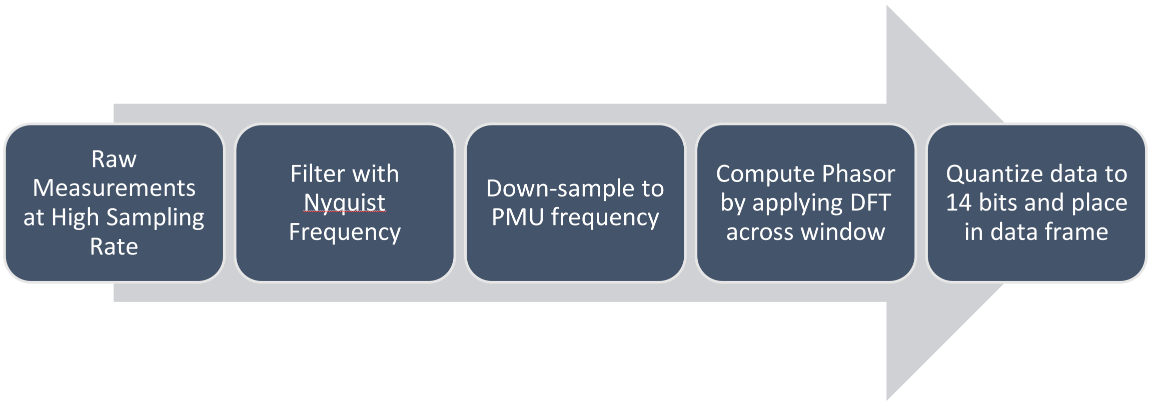

Raw Measurements at High Sampling Rate
Filter with Nyquist Frequency
Down-sample to PMU frequency
Compute Phasor by applying DFT across window
Quantize data to 14 bits and place in data frame

Figure 4-3. Phasor Measurement Unit Data Process

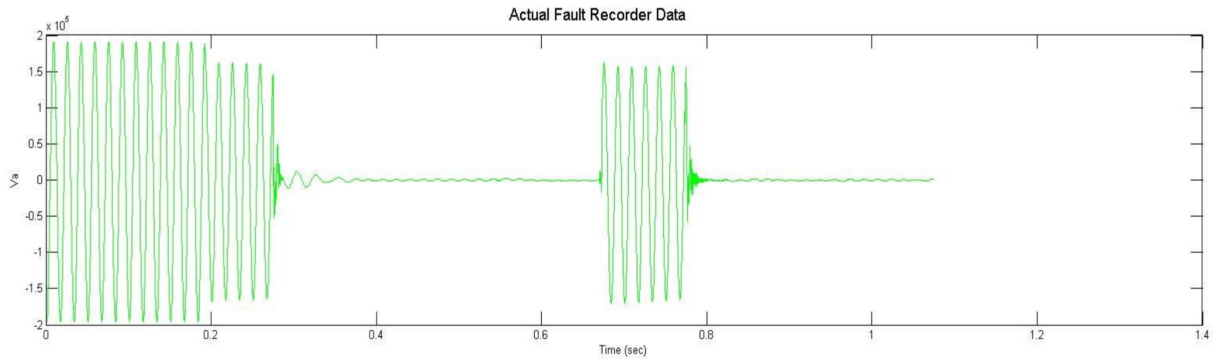


Figure 4-4. DFR Fault Response – Voltage

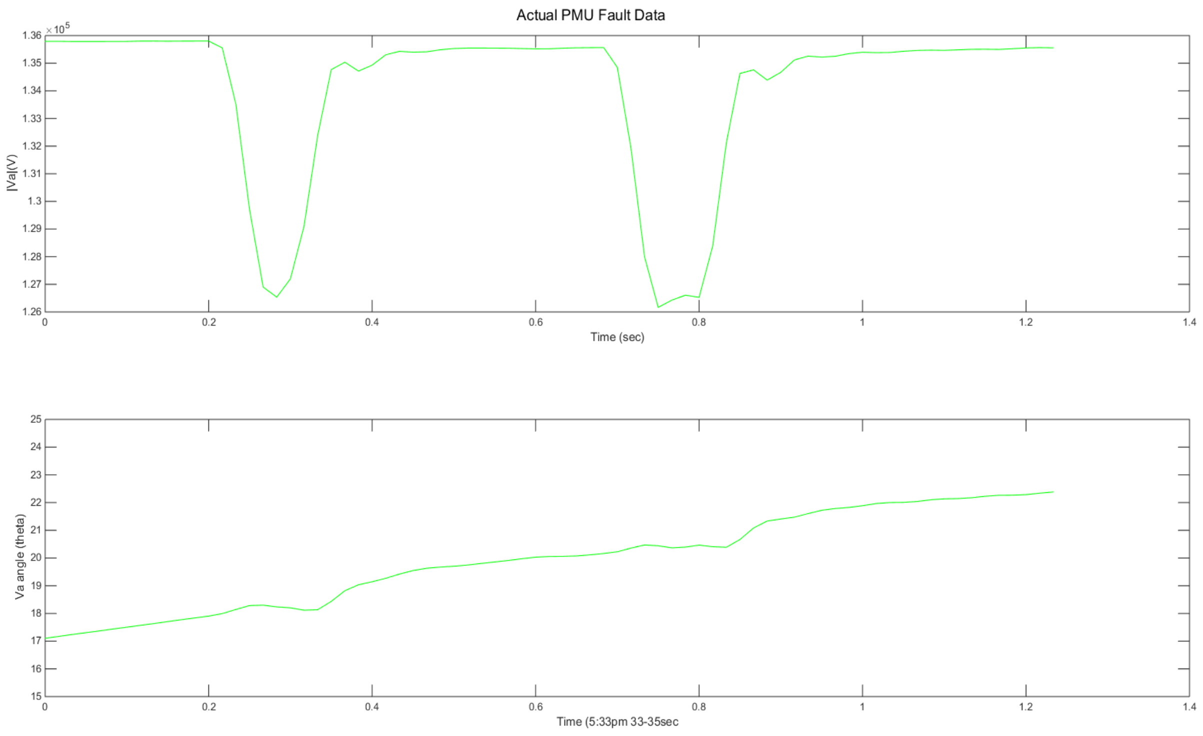


Figure 4-5 (a) and (b). PMU Fault Response – Voltage Magnitude and Phase Angle

The data captured by the DFR was compared to the measurements obtained from the PMU to ensure that each segment of the fault response can be seen by both devices. Figure 4-4 is the voltage signal captured by the DFR and figures 4-5 (a) and (b) is the voltage magnitude and phase angle output of the PMU, respectively. Both devices have a steady pre-fault voltage until approximately 0.2 seconds, which is followed by a drop in voltage during fault. When the breaker opens, the DFR voltage signal goes to zero and the PMU voltage magnitude goes back up, slightly lower than the pre-fault value. When the breaker recloses just before 0.7 seconds, both DFR and PMU voltages return to their initial fault measurements. After about 6 cycles, the system realizes that the fault is still on and the breaker re-opens. The DFR voltage goes to zero and the PMU voltage magnitude is slightly lower than pre-fault. The DFR output is at 5760 frames per second and the PMU was sampled at 60 frames per second.

The simulated data collected using PSS/E is comparable to the raw measurements taken by the PMU. The software-based PMU algorithm developed here will effectively replicate the real PMU's analog to digital conversion process, which can then be applied to the simulated data. Assuming the PMU's input is similar to the analog signal recorded by the DFR, the signal conversion scheme should be able to transform the DFR measurements into the PMU output. Since the DFR was much closer to the fault than the PMU, DFR-like data was generated to simulate a DFR near the PMU as shown in figure 4-6. This would be used as the input to the PMU algorithm.

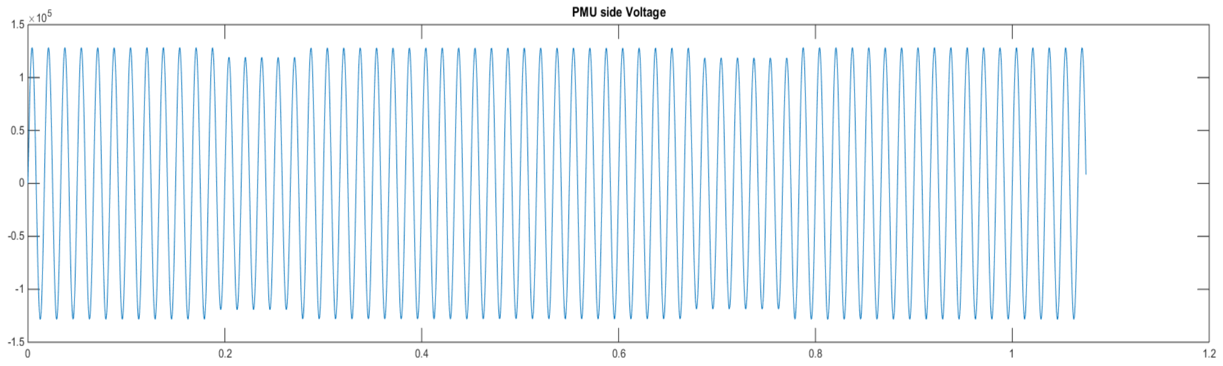


Figure 4-6. Simulated DFR Voltage – Input

Following the PMU data conversion process shown in figure 4-3 above, a generic PMU algorithm was developed using MATLAB. This program would read the DFR data at 5760 frames per second and apply a low-pass filter to it at Nyquist frequency. The data would be down-sampled to 60 frames per second and the phasor would be computed by applying a DFT across a one-cycle window. The data is then quantized into 14 bits according to specification. This generic algorithm is applied to the simulated DFR data and after multiple iterations, each time fine-tuning the sampling and filter parameters, the output is compared to the real PMU signal. The output from the PMU algorithm, referred to as the applied PMU, and the real PMU voltage curves are plotted in figure 4-7. The cut-off frequency of the LPF is set to 360 Hz and the DFT uses 6 samples in a cycle. The two curves, although similar, are not identical enough to validate the algorithm. The trough part of each curve is quite different; the applied PMU has sharper edges and a steeper slope, which leads us to believe that the size of the DFT window is incorrect. This window size must be increased to create a smoother curve. Windows with 1, 2, 3 and 4 cycles were tested and the results show that a 3 cycle window matches the real PMU output most closely. Figure 4-8 provides a demonstration of the final algorithm with 3 cycles and a quantization of 20 volts for a 500 kV line. The slope of the curves in the above plot are identical compared to the 1 cycle window plot. These results show that the PMU algorithm can provide us with a very good approximation to real PMU data.

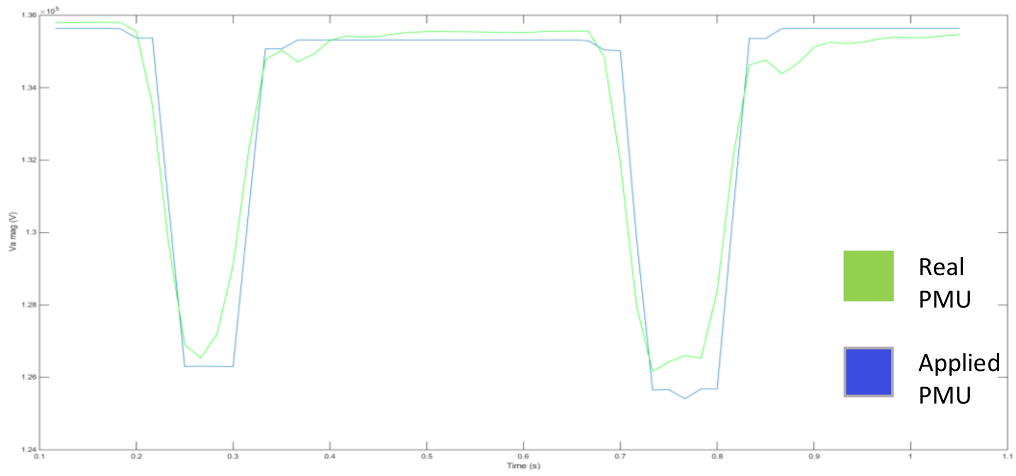


Figure 4-7. PMU Algorithm applied to DFR Data – 1 Cycle

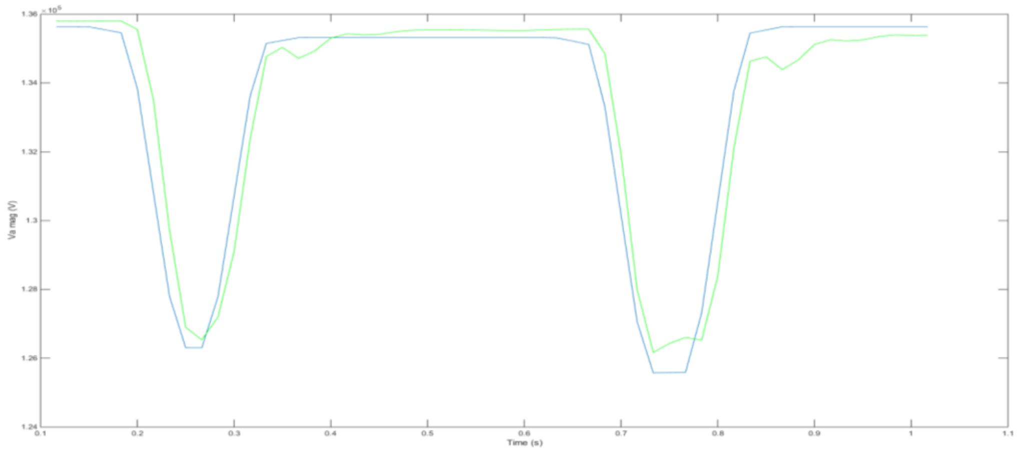


Figure 4-8. PMU Algorithm applied to DFR Data – 3 Cycles

The PMU algorithm was developed to convert simulated data into realistic PMU data. It was finally applied to simulated data captured using PSS/E. The same fault response used in the algorithm development was also simulated in PSS/E on the IEEE 118 bus system. Figure 4-9 and 4-10 show the PSS/E measurements before and after applying the conversion. Both figures include the voltage magnitude and phase angle plots. The first few points in the applied plot are the result of an incomplete DFT window and can be ignored. The PMU algorithm code developed in MATLAB can be found in appendix E.

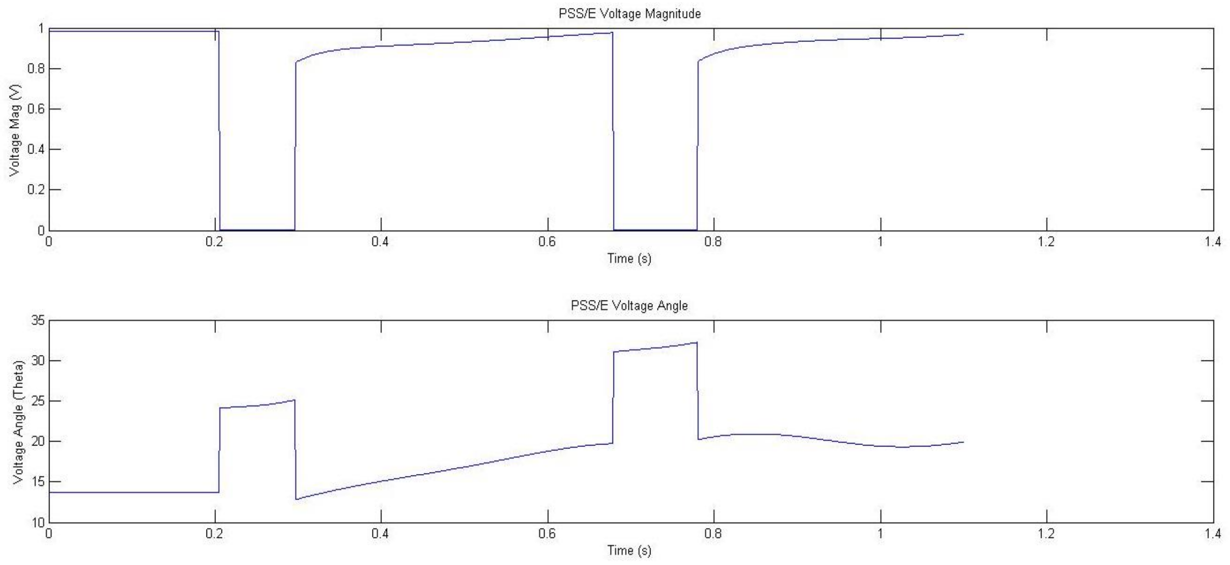


Figure 4-9. Original PSS/E Simulation Data

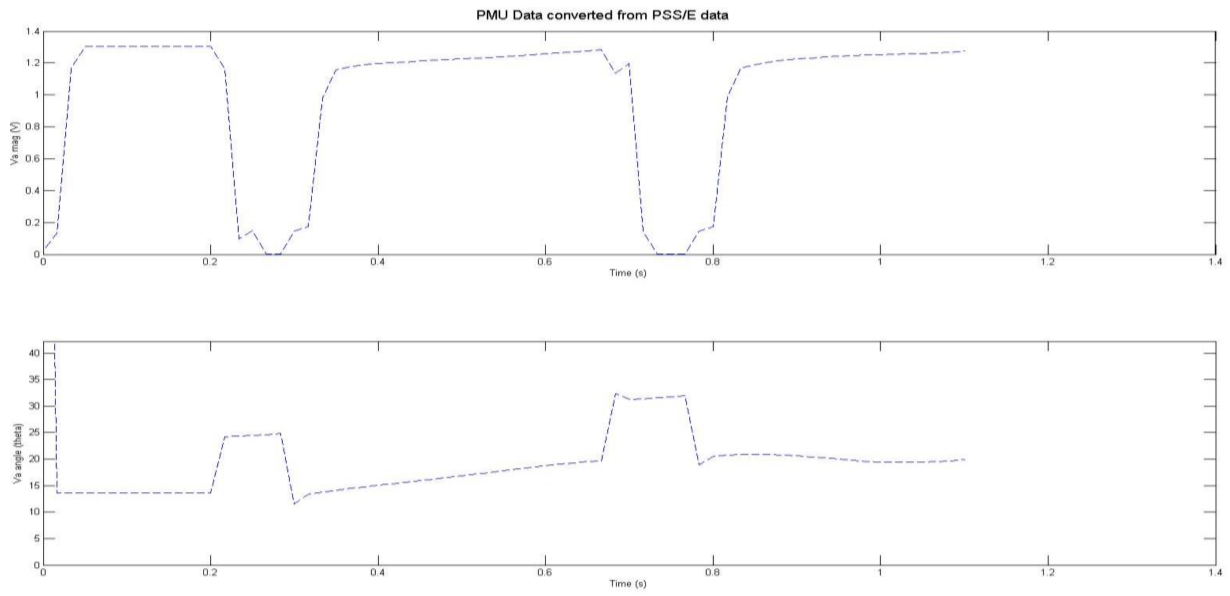


Figure 4-10. PMU Algorithm Applied to Simulated Data

Chapter 5. Static Load Model Parameter Estimation

After analysing the results from the distribution factor and voltage sensitivity methods, it seems like the most reliable way to find the N-1 contingency ranking would be to have an accurate model of the power system and run power flow simulations to estimate the post-outage system state. To keep the system model up-to-date and precise, the actual load model composition at every load bus must be determined. This can be accomplished with the measurement-based approach using bus voltage and injection measurements at each load bus. In order to create real-time updates to the system models, these measurements must be obtained at a high data rate. This can be accomplished using PMUs or cost effective devices such as power quality meters or protection relays with dual-functionality [50] [51].

5.1 ZIP Load Model Equation

The static ZIP load model equation from chapter 2 is revisited here, with parameters p_1, p_2 and p_3. In order to understand the parameters and their relationship with the load model, let us derive this equation using power equations.

$$P_L = [|V|^2\ |V|\ 1]\begin{bmatrix} p_1 \\ p_2 \\ p_3 \end{bmatrix} \tag{5-1}$$

The total active power consumed by the load is a summation of the active power consumed by each of the three possible static elements; constant power, constant current and constant impedance.

$$P_L = P_Z + P_I + P_P \tag{5-2}$$

The power consumed by a constant power element is equal to the percentage of power allocated for it. The power consumed by a constant current element is a function of the current required by the element and the voltage combined with the percentage of power allocated. Similarly, the power consumed by a constant impedance element is a function of the impedance of the element and the voltage combined with the percentage of power allocated. *P_{Total}* refers to the total power assigned to the load; this is different from the total power consumed by the load.

$$P_Z = \frac{|V|^2}{Z} \qquad P_I = |V|I^* \qquad P_P = p_3 P_{Total} \tag{5-3}$$

The impedance and current constants, Z and I^*, are based on the initial conditions of the system, similar to the constant power element.

$$P_{Z,i} = p_1 P_{Total} \qquad P_{I,i} = p_2 P_{Total} \tag{5-4}$$

$$Z = \frac{|V_i|^2}{p_1 P_{Total}} \qquad I^* = \frac{p_2 P_{Total}}{|V_i|} \tag{5-5}$$

The power consumed by each element:

$$P_Z = |V|^2 . \frac{p_1 P_{Total}}{|V_i|^2} \qquad P_I = |V| . \frac{p_2 P_{Total}}{|V_i|} \qquad P_P = p_3 P_{Total} \tag{5-6}$$

The total power consumed by the load:

$$P_{Load} = P_{Total}(p_1 \frac{|V|^2}{|V_i|^2} + p_2 \frac{|V|}{|V_i|} + p_3) \tag{5-7}$$

where $\frac{P_{Load}}{P_{Total}} = P_L$

Equation 5-1 is derivation of this equation, where the total power consumed and the voltage magnitudes are normalized by the initial conditions.

5.2 Method of Least Squares

5.2.1 Least Squares Algorithm

From the papers reviewed, it seems like the most common technique used to estimate these parameters is the method of least squares [52] [53]. The least squares approximation is used to solve for *Ax=b* when there is no simple solution [54]. Matrix, *A*, comprises of known quantities and has the size of m rows by n columns. *x* is the column vector of unknown variables and *b* is a column vector of real measurements. A simple inverse function cannot be applied since *A* is not a squares matrix. Least squares is generally applied when the number of equations exceeds the number of unknowns and there is some error in the measurements. This implies that *Ax = b + e* where *e* is the error vector.

In this case:

$$A = [|V|^2 \; |V| \; 1] \qquad x = \begin{bmatrix} p_1 \\ p_2 \\ p_3 \end{bmatrix} \qquad b + e = P_L$$

The least squares solution for *x* is found by minimizing the error vector, *e,* using the formulation:

$$||Ax - b||^2 = ||e||^2 = ||e|| \tag{5-8}$$

There are many ways to look at this optimization problem using methods from geometry, algebra and calculus and they all converge to the same solution. In order to transform *A* into a square matrix, the following relationship is applied:

$$_{min}||Ax - b||^2$$
$$\text{when} \quad A^T A x = A^T b \tag{5-9}$$

When the square matrix is sparse and symmetric, numerical solutions such as Cholesky Decomposition can be used to solve for the linear system. For simple, non-sparse matrices, the inverse of the square matrix can be used to solve for *x*, as in equation 5-10.

$$x = (A^T A)^{-1} \, A^T b \tag{5-10}$$

Substituting the variables for the ZIP load model, into the equation above, helps solve for the parameters:

$$\begin{bmatrix} p_1 \\ p_2 \\ p_3 \end{bmatrix} = ([|V|^2 \; |V| \; 1]^T [|V|^2 \; |V| \; 1])^{-1} \, [|V|^2 \; |V| \; 1]^T P_L \tag{5-11}$$

5.2.2 Least Squares Approximation on Test System

To assess the validity of this method, a simple experiment is conducted using the IEEE 118 bus system. In the initial three test cases, the load at bus 59 is either 100% constant power, 100% constant current or 100% constant impedance. The least squares approximation will be use to estimate these loads models as $[p_1\ p_2\ p_3] = [1\ 0\ 0]$, $[0\ 1\ 0]$, or $[0\ 0\ 1]$, respectively. The system is set up in PSLF and all load buses, except for bus 59, are modelled as 100% constant power since this is generally their default setting. The simulation is run by computing consecutive power flow solutions on the steady state model of the network. Each power flow represents one measurement set and the solution from the previous power flow provides the initial conditions for the next solution. The voltage magnitude and bus injection is recorded for the load bus for the duration of the simulation. The bus injection is measured using the summation of the generation and power flowing into the bus, minus the power flowing out of the bus. This measurement has a small error within 0.1 % of the total load power. For now, this test will only be applied to the real power component of the load.

The simulation starts with the system being in steady state and light loaded. Certain changes or events need to be introduced to produce some variation in our collected measurements. The events should be normal changes that occur on a regular day since the load models are to be estimated on a consistent basis. A valid range of voltage variation is approximately 5%; anything greater would require a severe disturbance and which is quite unrealistic. One type of system change is the daily load schedule; the load increases by 60% throughout the day. All load buses except for bus 59 are scaled. To prevent the power flow solution from diverging, the loads must be scaled in small increments. Since the simulation data is supposed to represent PMU data, each power flow corresponds to a PMU output at 30 frames per second. The timing of the loads scaling must be set up accordingly. For example, if the total load is scaled from 100MW to 160MW over a period of 6 hours then there will be 6hrs x 60mins x 60secs x 30fps = 648 000 sets of measurements where the total load is incremented by 60/648 000 = 93 Watts each time. However, scaling the load is not enough to cause a significant change at load bus 59. Additionally, a few realistic N-1 contingencies are applied to the system at various times. For example, the transmission line between bus 59 and 60 is removed temporarily while the loads are still being scaled and then the line is reconnected. Similarly, some loss in generation is introduced in the system and then it is returned back to its original state. The same process is repeated two more times, each with a different load model at bus 59. The voltage magnitude and the bus injection curves for the constant power load model can be seen in figures 5-1 (a) and (b), respectively.

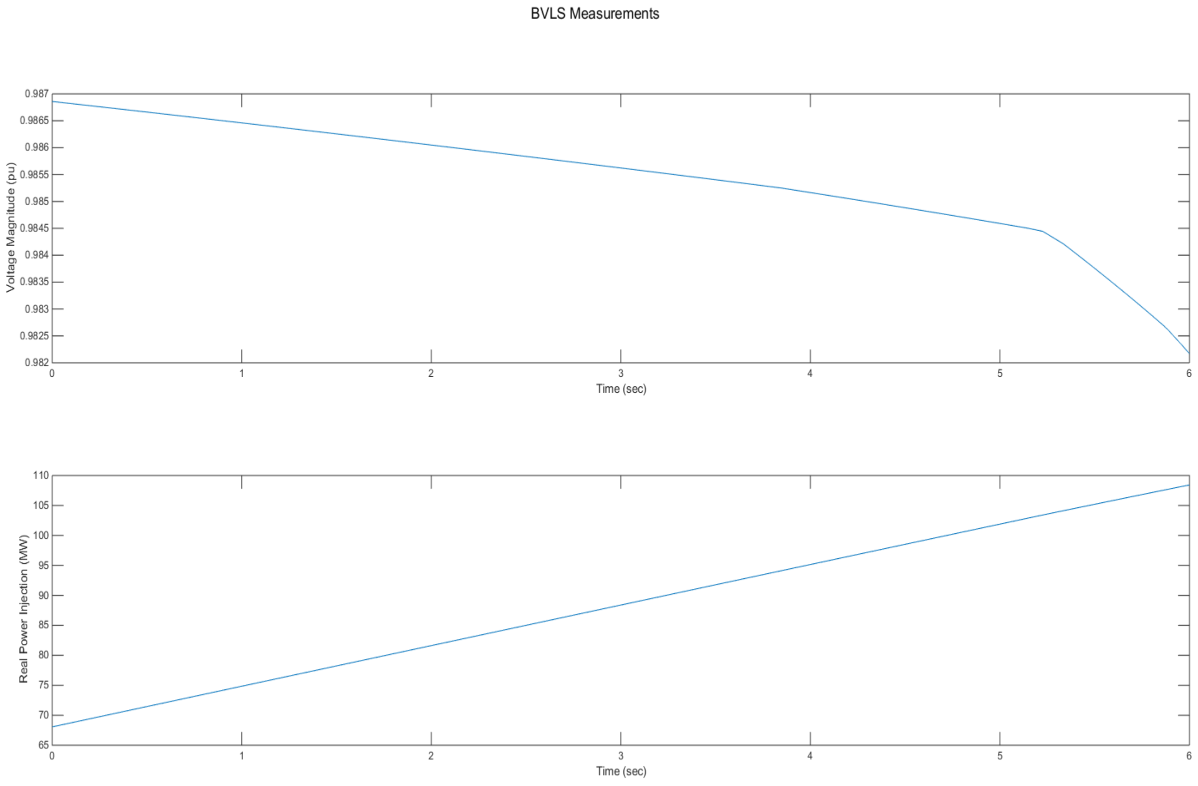


Figure 5-1 (a). Voltage Magnitude (b). Real Power Bus Injection

In test 1, the data is imported into MATLAB and the least squares approximation is applied to it, with voltage magnitudes in the *A* matrix and the corresponding load bus injections in the *b* vector. The bus voltages and injections are all normalized using the initial voltage and injection. The parameters, p_1, p_2, and p_3, estimated for each load model are shown in table 5-1.

As you can see, the estimates are completely wrong. In fact, they are in the order of 10^3 instead of between 0 and 1. After further analysis, it is found that the *A* matrix is ill-conditioned with a condition number of 10^6. This method does not provide a unique solution.

Table 5-1. Least Squares Parameter Estimates – Test 1

	Actual Parameters			Estimated Parameters		
	P_1	P_2	P_3	$P_{1,e}$	$P_{2,e}$	$P_{3,e}$
Constant Z	1	0	0	-1.9813×10^3	3.9085×10^3	-1.9267×10^3
Constant I	0	1	0	-1.8961×10^3	3.7399×10^3	-1.8431×10^3
Constant P	0	0	1	-1.8101×10^3	3.5694×10^3	-1.7586×10^3

There seems to be many repetitive measurements in the data set; this may filter out good measurements and lead to ill-conditioning. A MATLAB function, unique(), is applied to the *A* matrix to remove consecutive repeated rows and their corresponding load injections are deleted from the *b* vector. This brings down the size of the matrix to just over 500 rows. Once again, the least squares approximation is applied in test 2 with a reduced measurement set and the results are displayed in table 5-2. The results have not improved after making these modifications. The load model parameters are still in the order of 10^3.

Table 5-2. Least Squares Parameter Estimates – Test 2

	Actual Parameters			Estimated Parameters		
	P_1	P_2	P_3	$P_{1,e}$	$P_{2,e}$	$P_{3,e}$
Constant Z	1	0	0	-2.0228×10^3	3.9904×10^3	-1.9670×10^3
Constant I	0	1	0	-1.9342×10^3	3.8148×10^3	-1.8800×10^3
Constant P	0	0	1	-1.8336×10^3	3.6157×10^3	-1.7814×10^3

After checking the data and the least squares algorithm multiple times, no resolution was found. This method cannot successfully estimate the correct parameters for the load because it requires matrix *A* to be well-conditioned. It is highly unlikely to create such a matrix based on measurements from a real system since voltages have a limited range in a stable system. The constant Z parameter depends on $|V|^2$ and the constant P parameter only depends on the bus injection, so in some cases may be enough variation to solve for them. However, the constant I parameter is linearly dependent on $|V|$, which has a small range, therefore there is not a large enough change in measurements for the least squares method to be applied. After going through the procedures described in the research papers, it seems like most cases were using measurements with very large changes in the voltage in order to successfully implement this method. Their voltage variations were much higher than 5% and the contingencies placed on the system were beyond N-1. A well-conditioned *A* matrix was found under these conditions.

5.2.3 Variations of Least Squares

To work around this problem, other approximation techniques were studied and a few variations of the Least Squares approximation were found. There were three methods that seemed promising and they were all tested to see if the load model parameters can be correctly estimated within a small range of error, ε [16]. The Non-Negative Least Squares (NNLS) approximation is a constrained version of the least squares problem where the solution cannot be negative. The first published algorithm was by Lawson and Hanson where the solution uses the active set method [55]. This ensures that the constraints are part of the "active" set if they are valid at the candidate solution. The solution slowly converges to a set of parameters that are non-negative and have minimal error. The NNLS method was tested first and its estimated parameters were within an error of +/-5% when the load models did not have a constant I component, and the voltage varied +/0.5 per unit. However, when the load model had any component of constant I, the solution was outside an acceptable range of error (>100%).

The Least Squares Curve Fit approximation is a MATLAB function that takes a set of points and tries to best-fit them to a curve using a user defined starting value [56]. This function also allows the user to set a lower and upper bound to the solution. It finds the parameter vector, *x*, by solving the non-linear curve-fitting problem using a Least Squares approach,

$$\min_{x} \|F(x, xdata) - ydata\|_2^2 = \min_{x} \sum_{i} (F(x, xdata_i) - ydata_i)^2$$

where xdata and ydata are equivalent to the A matrix and b vector, respectively. The candidate solution is updated in every iteration until it meets the function tolerance. This method finds the local minimum based on the starting values. The LS Curve Fit function was applied to the IEEE 118 bus system at load bus 59 using the same measurement set from the NNLS method. The starting values were always set to [0 0 0] and the upper and lower bounds were set as [0 1] for each parameter. Though not perfect, this was the first approximation method that seemed to produce some positive results. A few more combinations of load model parameters were tested in addition to the three parameter sets used in the earlier approximations. The results are shown in table 5-3. All parameters are within a respectable error of 5% except for the 100% constant current load model. The linear relationship between current and voltage incorrectly estimates this load as 100% constant impedance. This method can correctly estimate the [0 1 0] load model if the starting values were closer to the solution. Due to this result, no further testing was conducted using the LS Curve Fit method and another variation of the Least Squares approximation, Bounded Variable Least Squares, was explored.

Table 5-3. Least Squares Curve Fit Approximation

Actual Parameters			Estimated Parameters		
P_1	P_2	P_3	$P_{1,e}$	$P_{2,e}$	$P_{3,e}$
1	0	0	1.00	0.0278	0
0	1	0	1.00	0.0433	0
0	0	1	0.0001	0.0590	1.00
0.4	0.2	0.4	0.3887	0.2147	0.3925
0	0.5	0.5	0.1017	0.2974	0.5988

5.3 Bounded Variable Least Squares (BVLS)

5.3.1 BLVS Methodology

The Bounded Variable Least Squares (BVLS) approximation is an unconstrained Least Squares method that sets an upper and lower bound on the solution while minimizing the error. The MATLAB package developed by Mullen [57] uses the Stark-Parker algorithm to implement BVLS [58]. This algorithm is modelled on Non-Negative Least Squares and solves the problem,

$$\min_{l \le x \le u} \|Ax - b\|_2 \qquad (5\text{-}6)$$

where A is an m by n matrix, x is the parameter vector of size n and *l* and *u* are the lower and upper bounds, respectively. It follows the active set strategy, but instead of one, two active sets are maintained; one for the lower bound and the other for the upper bound. There are two options for initializing the algorithm; each referred to as the "warm-start" and the "cold-start", respectively. The "warm-start" sets the initial values for the parameter vector, x, so that the system reaches convergence sooner. The "cold-start" is the opposite, where the initial parameters are set as the lower bounds by default. The set of linear equations are solved using the gradient of the objective, which minimizes the error vector. The algorithm progresses in an iterative loop, each time finding a new parameter that fits between the bounds. The iterations stop once a solution is found or if the maximum number of iterations has been reached. This method works well if the system has multiple solutions when $x \in \mathrm{R}$, but only a unique solution when x is within specified bounds.

This is an outline of the Stark-Parker algorithm:

1) Initialize lower and upper bounds, *l* and *u*. Set F contains indices of "free" components of the working solution, *x*, that are between the bounds *l* and *u*. L and U contain the components of *x* at their corresponding bounds. Initialize F = 0 and U = 0. Set L = {1,…,n} and every element of *x* to its lower bound.
2) Computer the negative of the gradient of the squared objective function, $w = A^T(b - Ax)$
3) Apply the Kuhn-Tucker test for convergence. If:
   (i) F = {1,…,n} or
   (ii) $w_j \le 0$ for all $j \in L$ and $w_j \ge 0$ for all $j \in U$
   Solution is found, skip the following steps.

4) Find $t^* = \arg\max_{t \in L \cup U} s_t w_t$, where $s_t = 1$ if $t \in L$ and $s_t = -1$ if $t \in U$.

   If t^* is not unique, select t^* arbitrarily and move to the set F.

5) Let b' be the data vector minus the predictions of the bound variables.

   $b'_j = b_j - \sum_{k \in L \cup U} A_{jkx_j}$.

   Let A' be the matrix composed of those columns of *A* whose indices are in F.

   Let j' denote the index of the column of A' corresponding to the original index $j \in F$.

   Find: $z = \arg\min \|A'z - b'\|_2^2$.

6) If: $l_j < z_j' < u_j$ for all j', then z_j' is a possible solution. Set $x_j = z_j'$ and go to step 2.

7) B is the set of components of z that are outside the bounds *l* and *u*.

   Compute: $q' = \arg\min_{j' \in B} min \left\{ \left| \frac{l_j - x_j}{z_{j'} - x_j} \right|, \left| \frac{u_j - x_j}{z_{j'} - x_j} \right| \right\}$

8) Set: $\alpha = \min \left\{ \left| \frac{l_q - x_q}{z_{q'} - x_q} \right|, \left| \frac{u_q - x_q}{z_{q'} - x_q} \right| \right\}$

9) Set $x_j = x_j + \alpha (z_j' - x_j)$ for all $j \in F$

10) Move every component of *x* that is at or below its lower bound to L and every component that is at or above its upper bound to U. Go to step 5.

5.3.2 BVLS on Test System

The BVLS method was tested on the IEEE 118 bus system to estimate the parameters at load bus 59. Each data set included load bus voltage and injection measurements for every set of model parameters. Data was recorded for a simulated time of 15 minutes at a rate of 30 frames per second for each load composition. Any changes made to the system, were after the initial 5 seconds. It generally takes a few cycles for the system to restore its balance after a disturbance or shift in flows. Each contingency or load scaling was followed by at least a 2-second gap to allow the system dynamics to stabilize. No other changes were applied to the system within these 2 seconds. A certain amount of voltage variation is required for parameter estimation. Since it is unlikely to have many different contingencies over a short period in an actual system, loads were scaled at nearby buses to achieve this voltage fluctuation. The load at the bus 59 was not scaled because it would require linearization of the nominal voltage and bus injection for the measurement set [53]. Consecutive repetitive measurements were removed from the data set before applying BVLS.

The BVLS approximation was tested on a 100 sets of load compositions using combinations of constant P, I and Z. Table 5-4 displays the results for a few of the parameter sets. The first column has the actual load parameters used in the simulation and the second column is the estimated parameters. The last column gives the total absolute error, ε, of the parameter estimation. This is the sum total of the errors, not the error per parameter.

$$\varepsilon = (|p_1 - p_{1e}| + |p_2 - p_{2e}| + |p_3 - p_{3e}|) \times 100\% \qquad (5\text{-}7)$$

In this testing, BVLS has proven to be the most accurate method of estimation, using a voltage variation of only +/-0.02 per unit and providing results within an error of +/-6% per measurement. This amount of voltage variation is realistic at a load bus in an actual power system. Comparatively, the Bounded Variable Least Squares approximation has the most consistent results in parameter estimation with the smallest amount of error. These estimates can be further improved if there were a larger range in voltage magnitude.

Table 5-4. Bounded Variable Least Squares Approximation

Actual Parameters			Estimated Parameters			Absolute Error
P_1	P_2	P_3	$P_{1,e}$	$P_{2,e}$	$P_{3,e}$	ε (%)
1	0	0	1.00	0.0293	0	2.93
0	1	0	0	1	0	0
0	0	1	0.0292	0	1.00	2.92
0.4	0.2	0.4	0.3943	0.2037	0.3978	1.16
0	0.5	0.5	0.0295	0.4401	0.5284	11.78

Following these positive results, the load model parameters estimated using the BVLS approximation were used to perform an N-1 contingency ranking. The rankings shown in table 5-5 for constant P, constant I, and constant Z were exactly the same as the lists created using the actual load model parameters. This is because the error per parameter is so small that it has very little effect on the load flow solution after an N-1 contingency.

Table 5-3. N-1 Contingency Ranking with Estimated Parameters

	Actual Load Parameters						Estimated Load Parameters					
Rank	**Constant Power**		**Constant Current**		**Constant Impedance**		**Constant Power**		**Constant Current**		**Constant Impedance**	
1	G	100	G	80	G	80	G	100	G	80	G	80
2	75	118	23	24	9	10	75	118	23	24	9	10
3	G	10	G	10	69	70	G	10	G	10	69	70
4	38	65	68	69	38	65	38	65	68	69	38	65
5	65	68	5	8	68	69	65	68	5	8	68	69
6	5	8	38	65	65	68	5	8	38	65	65	68
7	G	89	8	9	G	26	G	89	8	9	G	26
8	G	26	G	89	75	118	G	26	G	89	75	118
9	9	10	65	68	5	8	9	10	65	68	5	8
10	G	80	75	118	8	9	G	80	75	118	8	9
11	68	69	9	10	G	89	68	69	9	10	G	89
12	69	70	69	70	G	10	69	70	69	70	G	10
13	8	9	G	26	80	81	8	9	G	26	80	81
14	23	24	G	100	68	81	23	24	G	100	68	81
15	68	81	68	81	G	100	68	81	68	81	G	100
16	G	25	64	65	G	5	G	25	64	65	G	5
17	80	81	G	25	64	65	80	81	G	25	64	65
18	37	38	80	81	37	38	37	38	80	81	37	38
19	64	65	37	38	G	49	64	65	37	38	G	49
20	G	49	G	49	69	75	G	49	G	49	69	75
21	69	75	69	75	G	66	69	75	69	75	G	66
22	G	66	G	66	G	21	G	66	G	66	G	21
23	G	54	G	12	30	38	G	54	G	12	30	38
24	G	12	G	54	G	54	G	12	G	54	G	54
25	30	38	30	38	G	65	30	38	30	38	G	65
26	G	65	G	65	25	26	G	65	G	65	25	26
27	25	26	25	26	69	77	25	26	25	26	69	77
28	69	77	69	77	100	103	69	77	69	77	100	103
29	100	103	100	103	76	77	100	103	100	103	76	77
30	76	77	76	77	26	30	76	77	76	77	26	30
31	26	30	26	30	G	61	26	30	26	30	G	61
32	G	61	G	61	22	23	G	61	G	61	22	23
33	G	59	G	59	G	59	G	59	G	59	G	59
34	63	64	22	23	24	70	63	64	22	23	24	70
35	59	63	24	70	45	46	59	63	24	70	45	46

5.3.3 Conclusions

The load model parameter estimation using Bounded Variable Least Squares is a viable option for updating the power system models. It provides reliable and accurate results compared to the other methods tested. It does not require unrealistic types of disturbances on the power system as seen in other studies. The computation of the algorithm is fast enough to be implemented in almost real-time. The only limitations to this method include the period of time needed to collect the data while the load composition remains static, it cannot undergo any changes. During this time, the range of voltage variation required at the load bus is approximately 2%. Future work before implementation on a real power system would include applying the algorithm to two or more load buses at one time.

After the loads are modeled, the N-1 contingency analysis is performed by computing the power flows. This can be a computationally intensive for a large power grid. This is why, currently, power system planners in contingency studies use a combination of power flow and sensitivity analysis. Proposed future work in this area would be to develop a method to compute faster load flows. By taking advantage of emerging technologies in parallel computing, this could lead to a break-through in the power industry.

Chapter 6. Induction Motor Load Parameter Estimation

The types of loads connected to the power grid have changed significantly in recent years with the high number of induction motors due to air-conditioning load and the advancement of electronic devices. The increase in complexity of the loads makes modeling an even more vital task. The Western Electricity Coordinating Council (WECC) has introduced a new model [17] that more closely represents the load in the present day power system. This new model (illustrated in chapter 2) has a greater number of components compared to existing models, which make parameter estimation challenging. Dynamic loads are more complex than the static model seen in the previous chapter. The addition of a single induction motor to the model will increase the difficulty in parameter estimation since not only does the composition of the load need to be computed, but also the parameters of the induction machine need to be found [59]. Similarly, in the new WECC load model, the parameters for each dynamic component must be estimated.

6.1 Problem Statement

Generally, the induction motor demand represents the largest portion of the dynamic components in the load, usually around 20%. For the purpose of simplicity, this study will only focus on the complex model, which includes the induction motor combined with the static load (in figure 2-7). The proposed methodology can later be expanded to accommodate all dynamic load components. The induction motor model, CIM5BL (appendix A), found in the Siemens PSS/E Model Library is used for parameter estimation. The main parameters of this complex load that have an effect on the system dynamics, excluding motor starting, are in table 6-1. The remaining parameters only affect the motor starting or the per unit calculations of the model.

Table 6-1. Composite Load Model Parameters

Parameter	Description
LM	% Induction Motor Composition
SL	% Static Load Composition (100-LM)
R_s	Stator Resistance
X_s	Stator Reactance
R_r	Rotor Resistance
X_r	Rotor Reactance
X_m	Magnetizing Reactance
R_2	Rotor Resistance 2 (Dual Cage)
X_2	Rotor Reactance 2 (Dual Cage)
H	Inertia
D	Damping Factor

The majority of dynamic load modeling techniques that are currently being researched are based on optimization methods; genetic algorithms seemed to be the most popular [60] [61] [62]. Many parameter estimation methods were analyzed for this problem [63] [64]. Our criteria required a non-linear regression technique that can be trained to estimate a parameter set in almost real-time. Although genetic algorithms can be a good solution for parameter estimation, they cannot be applied in real-time. In addition, these methods cannot deduce additional characteristics about the system under study. We needed to develop a general load model off-line using known values and then apply it to the system in real-time. Machine learning fits this criteria very nicely.

6.2 Machine Learning

There has been a huge boom in the field of machine learning recently, mainly due to the large amounts of data that is being collected (i.e. Google) and the increase in computational resources. Machine learning, an area of Artificial Intelligence (AI), is the study of advanced pattern recognition and data prediction. It does not follow static programmed instructions; rather, it uses example inputs to make data-driven decisions. It can parse through large sets of data and pick out the relevant relationships to construct a behavioural model. All machine-learning algorithms include this fundamental structure [65]:

Input: x

Output: y

Data: $(x_1,...,x_N)$ and $(y_1,...,y_N)$

Model Class: Neural Network, Decision Trees, etc

Target Function: f: $X \rightarrow Y$

6.2.1 Supervised and Unsupervised Learning

There are two main styles of learning algorithms, supervised and un-supervised [66] [67]. Both methods use known data to form their generalized model. The supervised learning focuses on making an accurate prediction whereas the unsupervised learning focuses on quantifying the accuracy of the description of the data. For example, let's say we have a data set of 1000 images represented by the input vector, x, and each image was paired with a corresponding output class, y, which classifies whether the image is of a human being or not a human being. The objective of the model would be to predict whether a new image x^* is human or not.

Input: $x \rightarrow$ Images

Output: $y \rightarrow$ {Human, Not Human}

Data: $D = \{(x_1,...,x_{1000}), (y_1,...,y_{1000})\}$

Predict: $y(x^*)$ using x^*

Since the predictor has been clearly defined, this is categorized as supervised learning. In another example, let us say Amazon.com is conducting a study to better understand its customers' purchasing patterns. It is collecting data using the statistics of 10 items for a group of 10000 customers.

Input: $\{x_i, i = 1,\ldots,10000\} \rightarrow$ customer purchase record for 10 items

Output: Undefined

Data: $D = \{x_i, i = 1,\ldots,10000\}$

Predict: $f(x) = a$, where a is a constant

The algorithm parses through all the data and finds common purchasing patterns such as whether the customer is likely to buy item 6 on the list if he/she buys item 2.

6.2.2 Classification and Regression

In this study, our primary focus will be supervised learning. There are two types of tasks which use supervised learning; classification and regression. Classification produces a model where the output belongs to 1 out of a possible n discrete classes. For example, let us say we have an input of 1000 images and an output of 10 possible objects. Each image can be composed of many objects that may or may not be a part of the 10 output classes. The classification model will be trained using these 1000 images and their corresponding classes. The learned model should be able to read a new image, x^*, and determine whether it fits into one of the 10 output classes. There can be no new output classes beyond what was used in training.

Input: $x^* \rightarrow$ Images

Output: $y \rightarrow \{y^1,\ldots,y^{10})$ $\rightarrow$ Object Type

Data: $D = \{(x_i, y_j),\ldots, (x_{1000}, y_{1000})\}$

Predict: $y(x^*)$, where $y \in \{y^1,\ldots,y^{10})$

For regression tasks, the model produced is able to predict an output from a continuous spectrum. The regression method is generally used to estimate numerical variables. For example, if we have a set of meteorological data, that includes *n* vectors of the temperature, the season, the time of day, the cloud coverage, etc. and we want to use it to develop a predictive model for temperature estimation. The input would be a new set of meteorological data without the temperature and the output would be the estimated temperature. The estimate does not have to be a numerical value that was already used in the training of the model. The prediction is an interpolation of the training data.

Input: $x^* \rightarrow$ {Season, time, cloud coverage, etc}

Output: y→Temperature

Data: D = {(x_i, y_j),…, (x_n, y_n)}

Predict: y(x^*), where y∈R

6.3 Neural Networks

After assessing a few of the machine learning algorithms, Neural Networks were chosen to solve this problem. Neural Nets are highly accurate and versatile in their learning methods. They can be configured to solve problems of all sizes and complexity, and there are many pre-built programs readily available. Unlike decision trees, the neural net is somewhat of a black box, where the algorithm creates its own adaptive control system. The user does not specify the relationship between any of the data points, except for differentiating between inputs and outputs. This is a huge benefit in terms of modeling aggregate loads; since each load component actually represents multiple individual loads, each having their own parameters. The combined effect of each of these loads is captured in one set of measurements; making it difficult to derive your own functions.

6.3.1 Neural Network Methodology

Artificial neural networks (ANN) are statistical learning models that are based on biological neural networks that are a part of the human central nervous system. The artificial nodes, also called neurons, are connected together to form a network much like a biological neural network [68]. They can perform functions collectively or in parallel, much like the brain. In 1943, McCulloch and Pitts created the first computational model for neural networks that was based on mathematics and which differentiated ANNs from its biological foundation. One of the most basic neural nets is a single-layer network developed by Rosenblatt in 1957 called the perceptron.

$$f(x) = \begin{cases} 1\ if\ w \times x + b > 0 \\ 0\ else \end{cases} \qquad (6\text{-}1)$$

It's an algorithm used to learn a binary classifier where a function maps the input, x, to a binary output, f(x) = 0 or 1 as described in equation 6-1. *w* is the weight vector and *b* is the bias. This algorithm has two layers, the input and the output, which transfer data via synapses, as seen in figure 6-1. Each layer has its own set of weights. The synapses store weight parameters, w, which are used to manipulate the data.

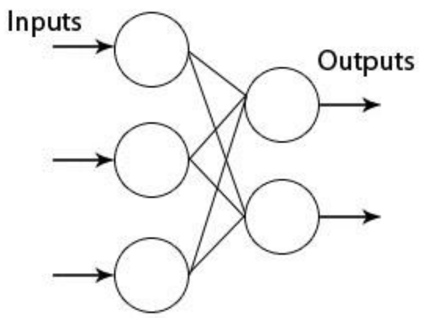


Figure 6-1. Single Layer Perceptron [69]

The activation function converts a neuron's weighted input to its output activation; it is basically the function that it applied at every node. The activation function for the single layer perceptron is the Heavyside step function, H(x).

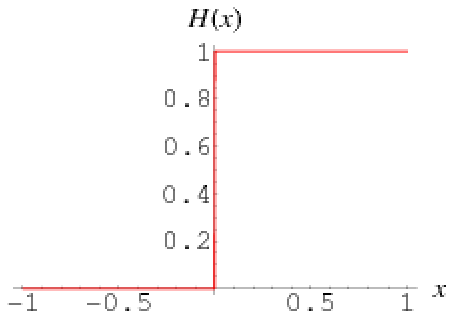


Figure 6-2. Heavyside Step Function [70]

The single layer perceptron is the simplest form of a feedforward network. The output from one layer is the input to the next layer. A feedforward network is where the connections only move in one direction, there are no loops or feedback. Back-propagation is used in sophisticated neural networks such as multilayer perceptrons, which include at least one hidden layer between the input and output layers. It trains the neurons using an optimization method to update the weights and minimize the error function in every iteration. The non-linear optimization is generally based on the gradient descent method where the derivative of the error function is taken with respect to the weight vector and the weights are updated to decrease the error in every cycle. For this reason, back-propagation requires the activation function to be differentiable. A simple model of a multilayer perceptron with one hidden layer is shown in figure 6-3. Here is a breakdown of the steps used in backpropagation:

1) Follow feedforward propagation through the neural network to generate the output activations
2) Back-propagate the output activations through the neural network to find the difference between the input and output values. Also known as the output deltas.
3) Multiply each output delta with the input activation at each weight synapse to get the new gradient of the weight
4) Find the learning rate by subtracting the weight by a ratio of the gradient. The learning rate is inversely proportional to the accuracy of the training, but directly proportional to speed of learning.

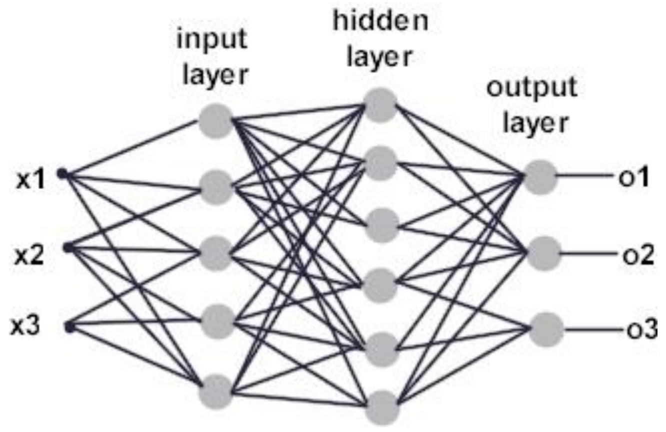


Figure 6-3. Multilayer Perceptron [71]

Back-propagation is only used during the training process. A network can be described as feedforward backpropagation if the network follows a feedforward direction but the learning process uses backpropagation.

6.3.2 Neural Network Design

There have been many variations and names for artificial neural networks but their underlying commonality is their paralleling processing capabilities. In order to design an ANN to model your data, these three parameters need to be specified:

1) The number of hidden layers and hidden neurons in each layer
2) The learning process for updating the weights of the interconnections, the loss function.
3) The activation function

6.3.2.1 Neural Network Architecture

A multilayer network can have one or more hidden layers; however, the Universal Approximation Theorem for neural networks states that a 3 layer network (1 hidden layer) with linear outputs can uniformly approximate any continuous function using the required amount of hidden neurons, with an acceptable amount of accuracy. Each hidden layer must have a defined number of hidden neurons. Increasing the number of hidden layers or number of hidden neurons can increase the accuracy of the approximation.

6.3.2.2 Learning Process and Cost Function

The learning process for updating the weights depends on the model chosen to minimize the cost function (equation 6-2); this model is based on optimization theory [72]. The cost function is made up of an error term and a regularization term. As mentioned earlier, this is usually some form of gradient descent applied using backpropagation. There are three types of backpropagation that can be chosen for the model; steepest descent, quasi-Newton and conjugate gradient. Levenberg-Marquardt (LM) [73] is a form of quasi-Newton backpropagation and is used in many neural network programs. The LM algorithm may not produce accurate results for problems with large data sets; in this case, other training functions such as Bayesian Regularization or Scaled Conjugate Gradient can be used [74].

$$L(x) = \frac{1}{N}\sum_{i=1}^{N} Error\big(y_i - y(x)\big) - Reg(x) \tag{6-2}$$

6.3.2.3 Activation Function

The Heavyside step function is one type of activation function. Due to its linear nature, it can be applied to complex models with the use of a large number of neurons. The sigmoid activation function can be applied using a much smaller number of neurons [68]. The normalized sigmoid function is commonly used in multiplayer perceptrons in the form of a hyperbolic tangent, described by equation 6-3 and figure 6-4.

$$g(w_0 + \sum_i w_i x_i) = \frac{1}{1 + e^{-(w_0 + \sum_i w_i x_i)}} \tag{6-3}$$

Where, g is the sigmoid function, $(w_0 + \sum_i w_i x_i)$ is the input at the current node, and *w* is the weight vector (*w₀* is the bias). This function is differentiable and therefore suitable for back-propagation.

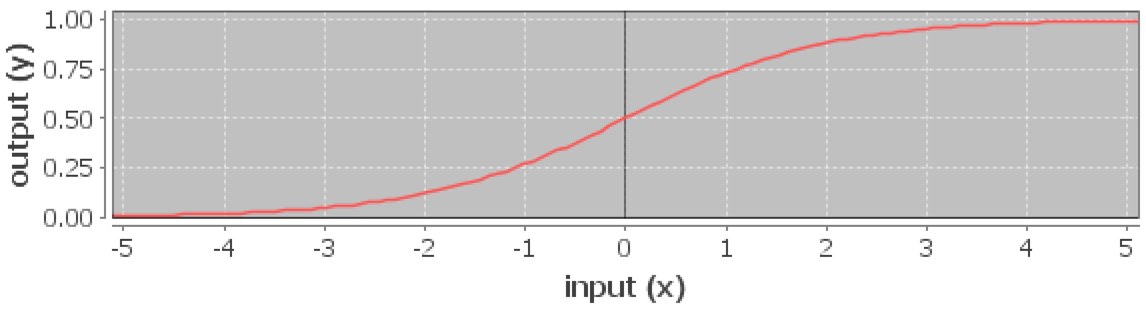


Figure 6-4. Sigmoid Activation Function [75]

6.3.2.4 Data Sets and Overfitting

During the training process, the data is usually divided into three subsets; the training, the validation and the testing sets. The error for each subset is calculated separately and can be measured using a different loss function. The training set is used for computing the gradient and updating the network weights and biases. This set has the majority of the data, generally greater than 50%. The validation set is used to check the training process by monitoring the validation error. The validation error will decrease along with the training error during the initial phase of training. The validation error will hit a minimum before it begins to rise again, all while the training error is still decreasing. The point at which the validation error increases is when the network is overfitting the data. The neural network model is usually saved when the validation error is at a minimum, to avoid overfitting; this process is called early stopping. Overfitting refers to when the network is very well trained for the specific data set presented to it such that the training error is quite small. It has memorized the training set; however, when new data is presented to the network, the error becomes quite large since the network has not learned to generalize to new data. Another way to improve generalization is called regularization. This is the second component in the cost function; based on the mean of squares of the weights and biases. The network response is more gradual since it causes it to have smaller weights and biases. There is no validation process when using a regularization method so it can be difficult to see whether there is still overfitting. The regularization is based on finding the optimal performance ratio parameter. Bayesian Regularization applies regularization by automatically updating this ratio and finding the optimal parameter. The network reaches convergence when the sum-squared error, the sum-squared weights and the number of parameters have reached a constant value.

The test set is the last subset of data. As its name implies, it is used to test and compare our model. The derived model is applied to the test set in every iteration but the test set error is not used during the training process. If the error on the test set reaches a minimum at a different iteration than the minimum of the validation error, then the data set may be divided poorly. If the test set error is unacceptable then the model must be re-trained from the start. The test set error cannot be used as the validation error to tweak the current training process.

6.3.3 Pre-Processing and Post-Processing Data

The neural network is the non-linear mapping of input data onto output variables. However, if raw data is used as the input to the neural network, then there is a high probability of obtaining poor results. For non-trivial cases, the input data is generally pre-processed and the output data is post-processed before training the model as shown in figure 6-5 (a). Any subsequent test data is also pre-processed before applying the trained model as shown in figure 6-5 (b).

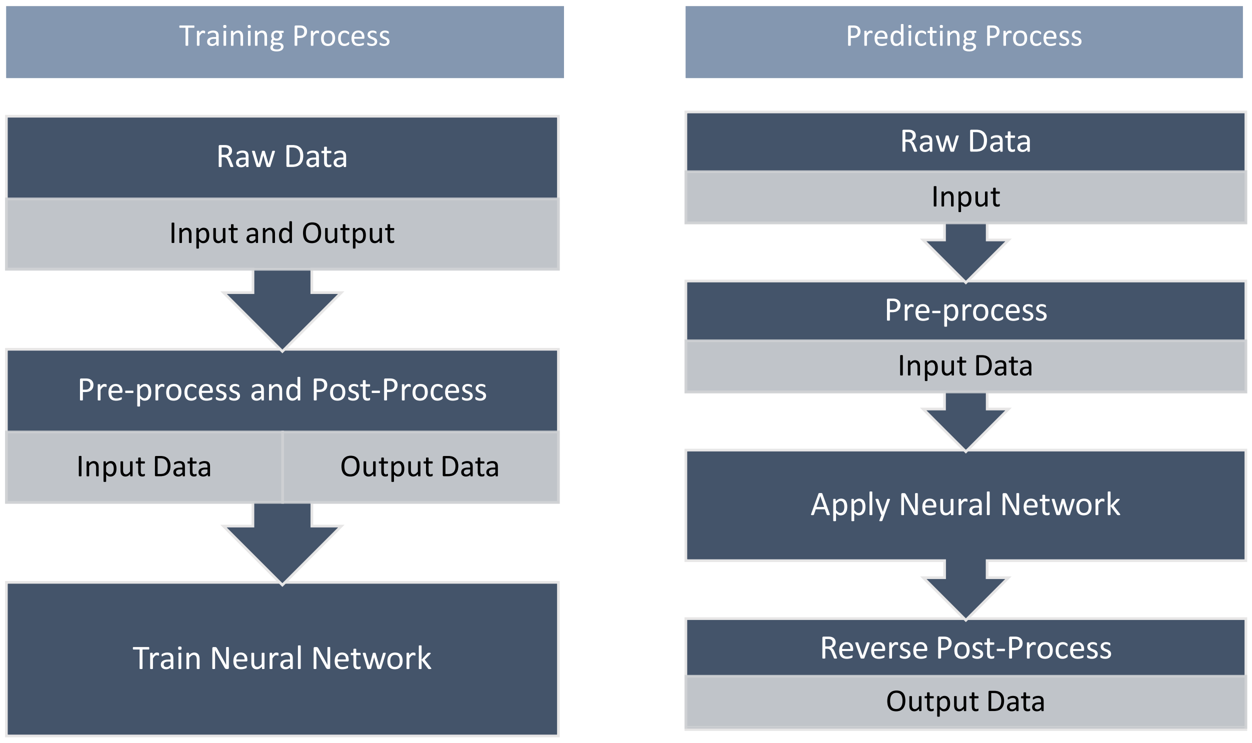


Figure 6-5. (a) Training Data Process and (b) New Data Process

There are many reasons for processing the data before network training. Many of them are based on prior knowledge of the data set and others are used to derive new information from the data. The basic neural network algorithm treats each input value as mutually exclusive to the rest of the data set. Any relationship between these data points is lost during training, therefore pre-processing and post-processing is applied to aide in the training. Two common forms of processing include normalization of data and feature extraction. Feature extraction is usually applied to reduce the dimensionality of the data. This can simply mean discarding a subset of the inputs or it can refer to making combinations of certain inputs. All of these methods are described in detail below.

6.3.3.1 Normalize Parameters

When there are input or output variables that have very different values, then they should be normalized. For example, if the input data set comprised of length measurements in different units, then the length values should be normalized to the same scale. If the measurements are not normalized then the values on the smaller scale can be orders of magnitude higher than the values on the larger scale, thereby giving greater importance to the larger values. Another reason to normalize between zero and unity is to ensure that the inputs and outputs are of the same order as the weights. If the input data is not normalized then the weights must be scaled according to each input parameter. In our problem, the input measurement set comprises of voltage, current and frequency already normalized to the per unit scale. The output variables, which are the machine parameters, are on various scales, therefore they need to be normalized using a linear transformation [76]. The normalizing of the output variables is also very important in calculating the mean squared error since the error for each parameter must be valued equally. The parameters are normalized using the following equations:

$$\text{Mean: } \overline{x_n} = \frac{1}{N}\sum_{i=1}^{N} x_i^n \qquad (6\text{-}4)$$

$$\text{Variance: } \sigma_n^2 = \frac{1}{N-1}\sum_{i=1}^{N}(x_i^n - \overline{x_n})^{\,2} \qquad (6\text{-}5)$$

$$\text{Normalized data: } \tilde{x}_i^n = \frac{x_i^n - \bar{x}_n}{\sigma_n} \qquad (6\text{-}6)$$

Equations 6-4 and 6-5 calculate the mean and variance, respectively, of each variable type in the parameter set. Equation 6-6 finds the normalized value of each parameter type using the computed mean and variance. The transformed parameter set will have a zero mean and unit variance.

6.3.3.2 Principle Component Analysis

The curse of dimensionality refers to the high number of dimensions within our data space that can reduce the performance of our neural network. If each measurement, x_i, is considered a feature and each feature adds one dimension to the data, then as the number of features increase, the dimensions increase, therefore increasing the distance between each possible point in the data space. In order to have an accurate model, the distance between each possible data point must be small. The only way this distance can be decreased while maintaining the number of dimensions would be to increase the size of the training observations. When there is a limited amount of training data available, reducing this distance with the same number of dimensions is impossible. The only way to reduce this distance would be to reduce the number of dimensions. One way of reducing the dimensionality is to intelligently discard some highly correlated variables in the input data.

In our problem, we have a measurement vector of voltage, current and frequency for n, number of observations. During data collection, there may be certain time intervals where the measurements for every observation are equal, meaning the voltage, current or frequency during a certain period may be equal no matter what the load model parameters are set to. These redundant measurements can be removed from our data space to reduce the dimensionality. The same set of discarded measurements must be removed from the test set and any future data sets to which the model is applied. The same method can be applied to highly correlated data.

Principle component analysis (PCA) is a procedure that can take a set of variables, which consists of some correlated data and transforms it into a matrix of linearly uncorrelated variables. These variables are called the principle components of the original matrix. The resultant number of principle components for each observation is equal and always smaller than the number of variables in the original observation vector, thereby reducing dimensionality. This orthogonal transformation organizes the principle components in the order of highest to lowest amount of variance. The principle component algorithm and equations can be found in section 8.6 of [76]. PCA is applied to voltage, current and frequency separately. The process of applying PCA to each measurement type is as follows:

1) Form m by n matrix of measurements (e.g. voltage) where m is the consecutive measurements within one observation and n is the number of observations.
2) Apply PCA to the rows of the measurement matrix, where the correlation between each measurement in the same row is found
3) PCA will result in p by n matrix where p is the new number of measurements or principle components per observation
4) Combine the principle component matrices for each measurement type to create the input matrix to the neural network training

6.3.3.3 Fourier Transform

Another way to reduce the dimensionality of a data set is by finding linear or non-linear combinations within the original data to form new inputs for the neural network. Each of these combinations are the extracted features of the data. Feature extraction can be performed as a pre-processing step if the criteria is clearly defined to differentiate between the subset of data that will be passed onto the network input and the subset that will be discarded. One popular method of feature extraction on time-series data is by applying the Fast Fourier Transform (FFT) to obtain the frequency spectrum of the data [77]. The FFT is a faster version of the Discrete Fourier Transform (DFT) and produces the same or better result. There are many versions of the FFT function depending on the software package used.

The DFT is defined by:

$$X_k = \sum_{n=0}^{N-1} x_n e^{-i2\pi k\frac{n}{N}} \tag{6-7}$$

where x_n are complex numbers in the time series $0 \le n \le N-1$. This method ensures that the time-series characteristic of the data is not lost during neural network training. The time-series information can be useful for some neural network training processes, but there are also cases where this information holds no value. It is up to the neural network designer to test whether each pre-processing or post-processing step improves the accuracy of the neural network or increase its computational speed.

6.3.4 Ensemble Methods: Bagging

The type of machine learning algorithm chosen to solve a problem can lead to a certain amount of bias or variance. Complex algorithms tend to have a low amount of bias but a high amount of variance, whereas simple learners have a low amount of variance and a high amount of bias. Bias refers to the error that stems from incorrect assumptions made by the learning algorithm. This error can prevent the algorithm from learning the relevant relationships between the input features and the target outputs. A high bias is also called under-fitting. Variance is the error produced during the learning process when the model becomes sensitive to small irrelevant values in the training set, such as random noise. A high variance model ends up being over-fitted to this specific data set and cannot be accurately applied to a new set of features. Both of these errors can prevent the model from generalizing beyond their training set and this problem is referred to as the bias-variance tradeoff [66].

Weak learners are described as classifiers that are only slightly correlated with the true classification and strong learners are well correlated. Neural networks are considered to be sophisticated learners. Weak learners would not be able to perform non-linear regression it order to accurately estimate the parameters in our load model. Complex algorithms such as neural networks are able to learn with large amounts of data, therefore they have low bias. However, they can also struggle to generalize and tend to over-fit to their training set, resulting in high variance. In order to overcome the bias-variance issue, ensemble methods can be applied depending on the learning method chosen. Ensemble methods involve the weighted averaging of many trained models based on the same feature and target sets. The two types of ensemble methods are boosting and bootstrap averaging, also known as bagging. Boosting is generally used with weak learning algorithms to reduce bias; they learn from the samples that are misclassified and re-weight them to correctly classify. Bagging is used with strong learners to reduce variance. Bagging involves bootstrap sampling of the training data set with replacement, which is used to learn multiple models and find their average.

Bagging is applied to the neural network model in this parameter estimation problem and is described in further detail here. Suppose we have a training dataset D = {(x_1, y_1),…,(x_n, y_n)} where x are the features and y are the target outputs. Bootstrapping produces *B* number of datasets, each randomly sampled from the training data D. Bootstrapping with replacement ensures that each randomly sampled set, S_i *(1≤i≤B)*, is the same size, n, as the original set D, since some random data points are repeated one or more times. *B* unique models are learned using their corresponding training sets. The models are then applied to the same test set to produce a committee of *B* number of estimates. The committee is averaged to find the final target output. The test set must be a completely unique set of data that has no overlap with the original set D. This ensures that there is no overfitting in the cross-validation. This process is illustrated in figure 6-6. The reason bagging provides a better solution than a single model can be shown by comparing their respective expected errors [66].

Expected Error based on Single Model:

$$Err_S = E\left[\left(y(x) - f^i(x)\right)^2\right]$$
$$= E[e_i(x)^2]$$

where y(x) is the target output and f^i(x) is the prediction.

Average Expected Error for Multiple Independent Predictions:

$$Err_{AVG} = \frac{1}{B}\sum_{i=1}^{B} E\left[\left(y(x) - f^i(x)\right)^2\right]$$

$$= \frac{1}{B}\sum_{i=1}^{B} E[e_i(x)^2]$$

where y(x) is the same target output and $f^i(x)$ is the prediction for each of the *B* models.

Expected Error based on Committee Model:

$$Err_{COM} = E[\left(\frac{1}{B}\sum_{i=1}^{B} y(x) - f^i(x)\right)^2]$$

$$= E[\left(\frac{1}{B}\sum_{i=1}^{B} e_i(x)\right)^2]$$

Assume the errors have zero mean and are uncorrelated:

$$(1) \quad E[e_i(x)] = 0$$

$$(2) \quad E[e_{i1}(x).e_{i2}(x)] = E[e_{i1}(x)].E[e_{i2}(x)] = 0$$

Then:

$$Err_{COM} = \frac{1}{B^2}\left[\sum_{i=1}^{B} E[e_i(x)^2] + \sum_{i=1}^{B} E[e_i(x)] \times \ldots \times E[e_B(x)]\right]$$

$$= \frac{1}{B^2}\sum_{i=1}^{B} E[e_i(x)^2]$$

$$= \frac{1}{B^2} E\left[\left(y(x) - f^i(x)\right)^2\right]$$

$$= \frac{1}{B} Err_{AVG}$$

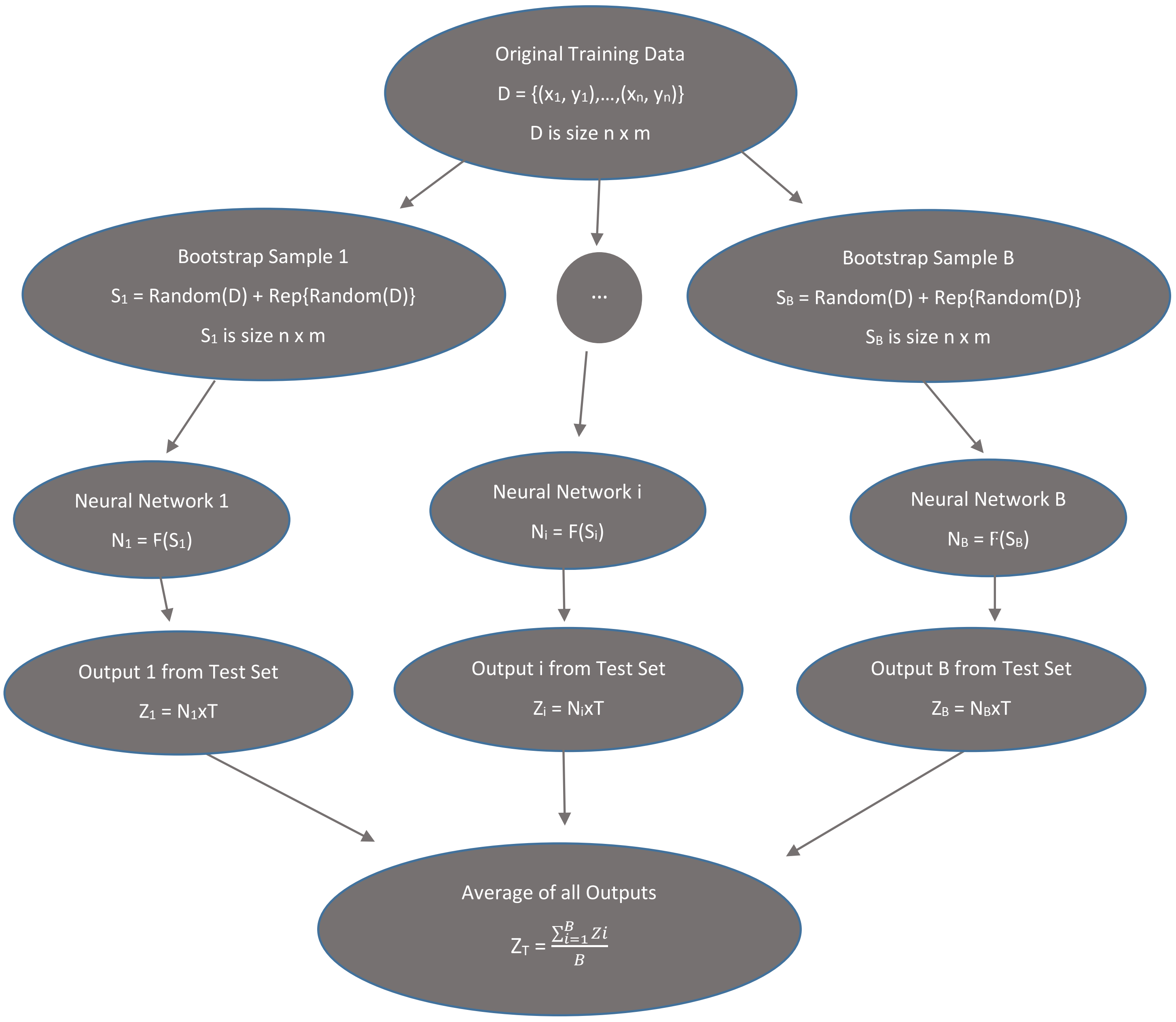


Figure 6-6. Bootstrap Averaging Process

6.4 Parameter Estimation on Test System

The neural network algorithm described in the previous sections will be applied to the IEEE 118 bus system to determine the complex load model parameters at load bus 11. The pre-processing, post-processing and ensemble methods studied earlier will be used to improve the parameter estimation. The PMU algorithm developed in chapter 3 will also be applied to the data set to represent realistic PMU measurements. The basic neural network training procedure is shown in figure 6-7. The measurements are collected using simulation software, PSS/E, and the PMU algorithm is applied to them. Each observation consists of the target load parameters and their corresponding measurements. The data is divided into the training, validation and test sets before passing through the algorithm. Successful completion of the learning process will produce a neural network for that specific load bus and event.

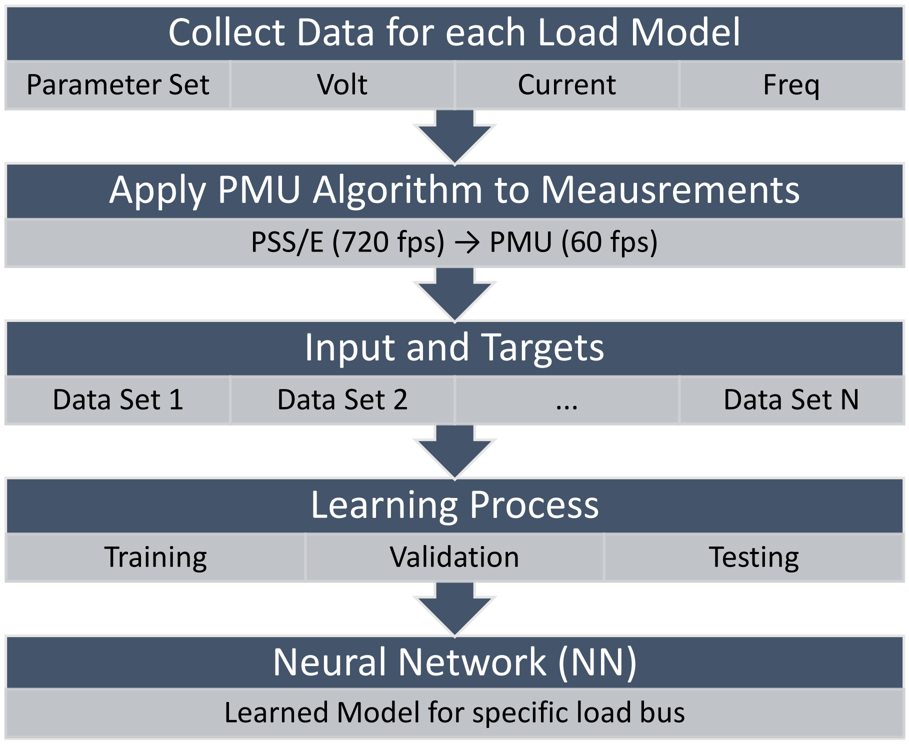


Figure 6-7. Neural Network Training

When the load model parameters need to be predicted in the actual power system, this model can be applied in almost real-time. The real PMU measurements from the load bus are recorded during the event, and then mapped to their target parameters using the designed neural network. This process is depicted in figure 6-8.

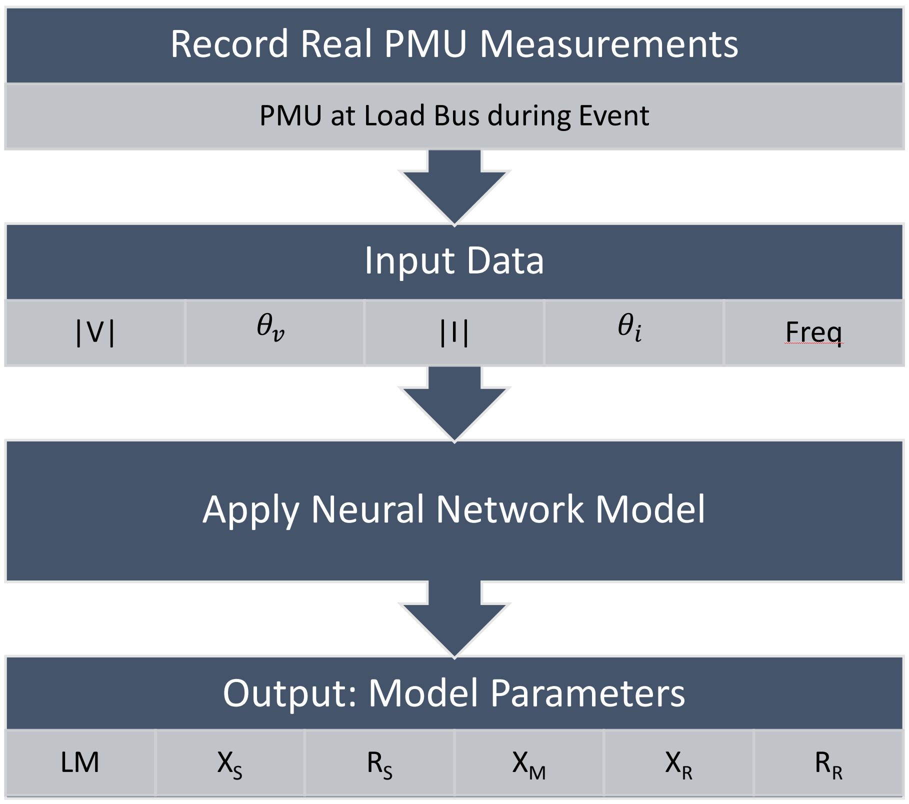


Figure 6-8. Neural Network Implementation

6.4.1 Test Setup and Data Collection

The complex load model that consists of the induction motor model, CIM5BL, and a constant power static load was added to the bus 11 on the IEEE 118 bus system. This replaced the 100% constant power static load that was previously there. Unlike the static load model, the induction motor model cannot be placed directly onto the high voltage load bus with a large power demand. This aggregate model must be divided into many individual loads with the appropriate amount of demand. Each individual load model is connected to a new low voltage bus, 200. Load bus 200 is connected to the high voltage bus, 11, through a two-winding transformer. The same parameters are used for each load model to ensure they behave in the same way and to replicate one large load bus. The number of individual loads needed, depends on the total amount of power demand on the induction motor. The higher the induction motor composition, LM, the greater number of load models are required. The minimum number of separate load models is set as 50 for a 10% induction motor load. An additional model is added for every 10% increase in induction motor load.

The initial parameter estimation algorithm was used to estimate only four parameters in the complex model. In order to set a baseline, no pre-processing, post-processing or ensemble methods were applied in the initial test. The four parameters that were estimated are:

1. LM - Percent Composition of Large Induction Motor
2. R_s – Stator Resistance
3. X_s – Stator Reactance
4. X_m – Magnetizing Reactance

The percent composition of the static load, SL, will not be counted as a parameter since it is easily computing by SL = 100 - LM. A MATLAB program was written to generate over 600 vectors of load model parameter sets. The range and number of unique values for each parameter was established and combinations of all four parameters were placed in each vector. The final matrix of parameters was saved to a text file. The code for creating the parameter matrix can be found in appendix F.1. The range of each of the estimated parameters is shown in the table below. The constant values of the parameters that are not estimated are in the second column of the table.

Table 6-2. Initial Test Parameter Values

Estimated Parameters	Constant Parameters
$0 \leq LM \leq 100$	$R_r = 0.013$
$0.012 \leq R_s \leq 0.024$	$X_r = 0.067$
$0.05 \leq X_s \leq 0.2$	$R_2 = 0.009$
$3.5 \leq X_m \leq 6.5$	$X_2 = 0.17$
	H = 1
	D = 0

The next part of the process is to collect the simulated data for each parameter set using PSS/E. The powerflow model for the IEEE 118 bus system is converged and the dynamic simulation is set up with a complex load model at bus 11. A set of pre-initialization processes are conducted before the dynamic simulation can be executed. These steps include:

1. NETG - Netting of the generation that does not have a dynamic model; this generation is converted into a negative load.
2. CONL - Conversion of all loads
3. CONG - Conversion of all generators
4. ORDR - Ordering Network Buses for Matrix Manipulation
5. FACT - Factorizing the Network Admittance Matrix
6. TYSL - Solving the Converted Case

Once these steps are completed and there are no resultant errors or warnings, the dynamic simulation can be initialized. The dynamic model initialization will flag any problems associated with the network model or dynamic parameters. The simulation is run for 1 minute and 5.4 seconds where data is captured at a rate of 720 frames per second. The recorded measurements are voltage magnitude and phase angle, current magnitude and phase angle and frequency at the load bus. The dynamic model parameter estimation requires much more variation in the measurements compared to the static load model. Instead of scaling the loads over a long period, a three-phase line to ground fault is placed near the load bus. The simulation is similar to the real sequence of events captured by the DFR [78] and PMU described in chapter 3. The simulated events are as follows:

1. Run from 0 to 12 seconds with no contingencies – steady state
2. Fault along transmission line between buses 11 and 4, at t = 12 seconds
3. Run with fault on line until t = 17.2 seconds
4. Trip breaker, transmission line disconnected at t = 17.2 seconds
5. Run with disconnected line until t = 40.2 seconds
6. Close breaker, transmission line re-connected at 40.2 seconds
7. Fault is still on transmission line. Run with fault until t = 46.2 seconds

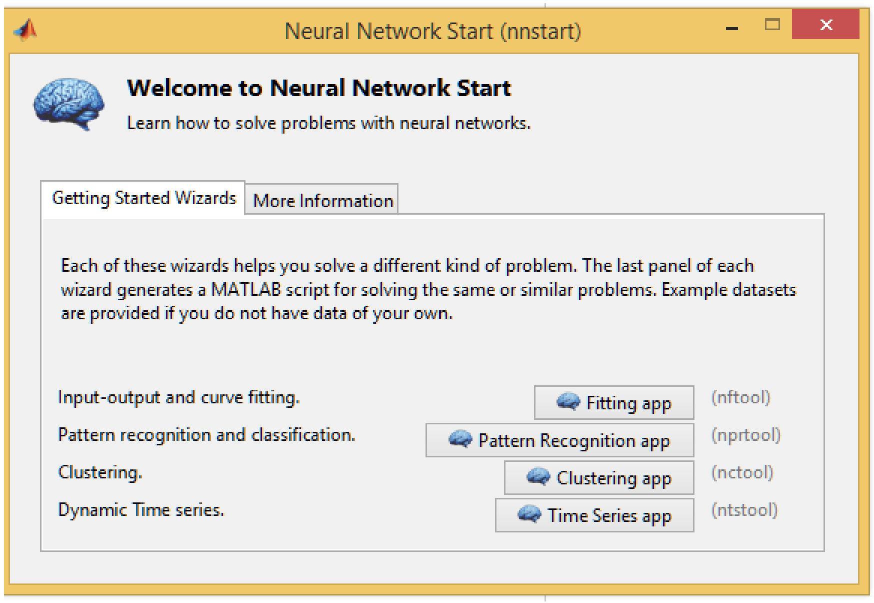
8. Trip breaker, disconnect faulted line at t = 46.2seconds
9. Run simulation with breaker tripped until t = 65.4 seconds

To obtain a large enough data set, the same process is repeated for each parameter vector in the saved text file, approximately 600 sets. The recorded measurements for each parameter vector are converted into individual text files. A python script was written to automate this process for all the parameter vectors; it can be found in appendix F.2.

The data collection is complete for the four parameter estimation. The PMU algorithm provided in appendix E was applied to every measurement set before the data was used for neural network training. The algorithm was slightly modified to accommodate the PSS/E recording rate of 720 frames per second. The data is imported from each text file into a .MAT file so that it can easily be accessed by the MATLAB program in the workspace. The neural network can now be designed using the MATLAB Neural Net Toolbox.

6.4.2 Neural Network Design

The MATLAB Neural Network Toolbox provides functions to model most common types of supervised and unsupervised neural networks. It provides a GUI interface to select the main characteristics of your problem and learning method. There are many more features within this toolbox that can be accessed using a MATLAB script. Figure 6-9 displays the start menu of the toolbox which allows you to select the type of problem you are trying to solve. The parameter estimation problem requires a neural network to map the numerical inputs to the numerical outputs; therefore the first option, the curve fitting app, is the most appropriate tool to use.


Neural Network Start (nnstart)

Welcome to Neural Network Start

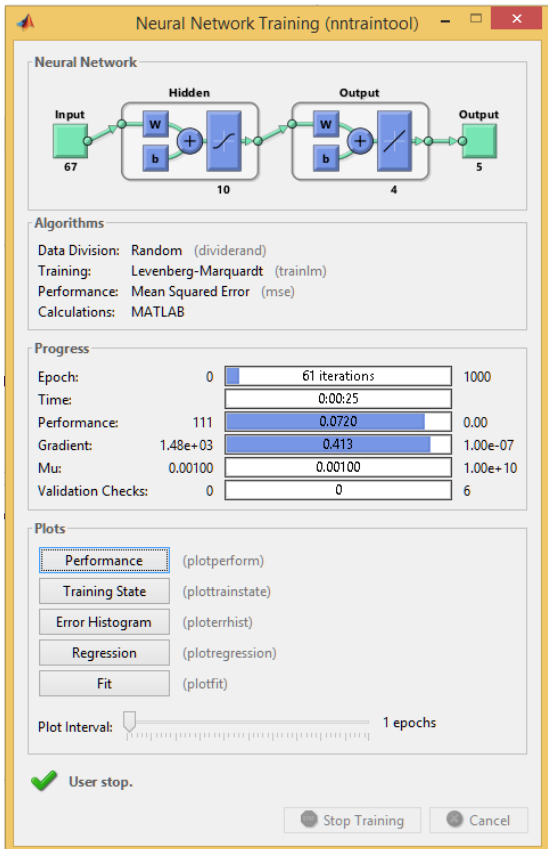
Learn how to solve problems with neural networks.

Each of these wizards helps you solve a different kind of problem. The last panel of each wizard generates a MATLAB script for solving the same or similar problems. Example datasets are provided if you do not have data of your own.

Input-output and curve fitting.	Fitting app	(nftool)
Pattern recognition and classification.	Pattern Recognition app	(nprtool)
Clustering.	Clustering app	(nctool)
Dynamic Time series.	Time Series app	(ntstool)

Figure 6-9. Neural Network Toolbox Start Menu

Following the steps in the GUI, the inputs and targets are selected from the workspace. The inputs are the measurement sets and the targets are the corresponding load model parameters. The toolbox will prompt you to divide your data into training, validation and test sets. The default training proportion is set to 70% and the validation and test sets make up 15% each. The GUI is set to use only one hidden layer in the network with a default of 10 hidden neurons. The next step is to train the network; the default training algorithm is Levenberg-Marquardt (L-M). The generalization is checked by measuring the mean squared error (MSE) of the validation set. This algorithm stops training when the generalization stops improving; which occurs when the MSE increases. Once the training process starts, it can be observed through the *nntraintool* interface shown in figure 6-10. The user has the option of stopping or canceling the training before completion.



Algorithms

Data Division: Random (dividerand)
Training: Levenberg-Marquardt (trainlm)
Performance: Mean Squared Error (mse)
Calculations: MATLAB

Progress

Epoch:	0	61 iterations	1000
Time:		0:00:25	
Performance:	111	0.0720	0.00
Gradient:	1.48e+03	0.413	1.00e-07
Mu:	0.00100	0.00100	1.00e+10
Validation Checks:	0	0	6

User stop.

Figure 6-10. Neural Network Training Tool

As you can see above, there are many parameters of the training process that have been preselected such as the maximum number of epochs, the number of iterations used for validation, and error threshold, mu. After the training process is complete, the GUI allows the user to generate a MATLAB script based on the neural network designed. This script will be used as a starting point to design our training function.

Using many of the default settings, the script was used to design a neural network to estimate the target parameters with a data set of 600 observations. Since we are dealing with a large data set with a high number of dimensions, the number of hidden neurons is initially increased to 30. The configuration of the first learning algorithm can be found in table 6-3. The epochs refers to the maximum number of iterations used in training; the number of time the loss function is calculated and the weights and bias are updated. The validation checks are the number of iterations performed after the validation MSE starts to increase. This is to ensure that the minimum MSE has been found and the MSE is in fact increasing during each validation check. The updated model at the end of the validation checks will be the final neural network.

Table 6-3. Initial Neural Network Configuration

Target Parameters	4
Number of Observations	600
Training Function	Levenberg-Marquardt (L-M)
Number of Hidden Layers	1
Humber of Hidden Neurons	30
Training Set	70%
Validation Set	15%
Test Set	15%
Epochs	1000
Validation Checks	6
Performance Function	Mean Squared Error

Once the network was configured, the weights and bias were initialized and the training process began. The training stopped once 1000 epochs were reached. The training and validation error never reached a minimum, therefore an accurate network could not be found. This is a key sign of under-fitting. The size of the data set needed to be increase to prevent the under-fitting. Considering the range of each of the target parameters, more points within each range were required. Following the data collection process described in the previous section, a set of over 2000 observations were acquired. To accommodate this larger data set, the number of hidden neurons was increased to 50 in the one hidden layer. The training processes ended at 416 epochs with a MSE of 10^{-2}. The results from this can be found in figure 6-11 which includes a plot of the training, validation and test errors. The MATLAB script can be found in appendix F.3.

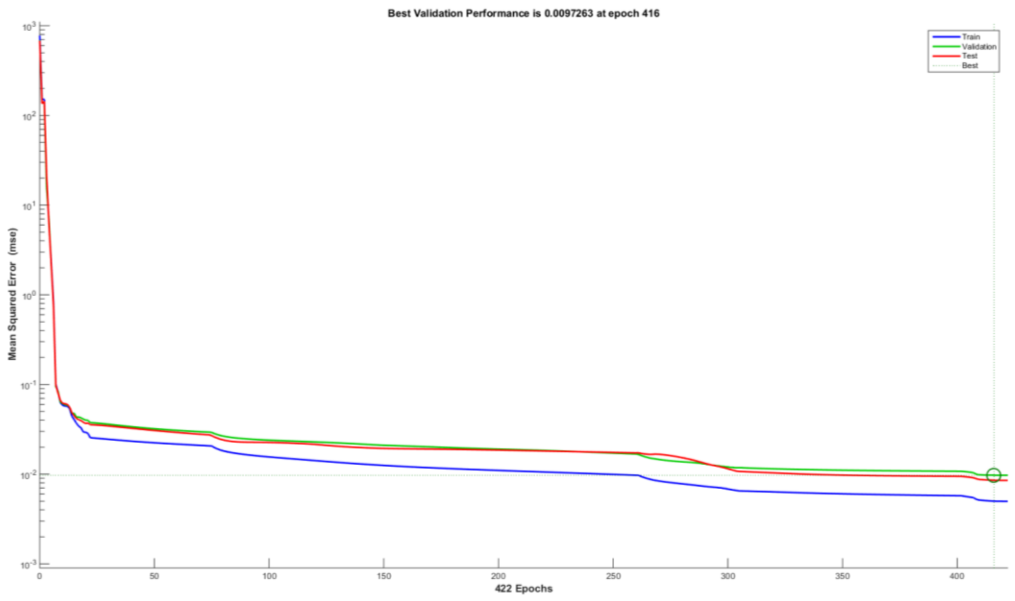


Figure 6-11. Training Performance using L-M method

A small set of new measurements were taken using unique load model parameters that are not part of the original data set. These new measurements are tested with the neural network model to check for generalization. The results from this new test set were quite good, with the maximum error of a parameter at 9%.

6.4.3 Improved Neural Network

Following these results, the number of parameters being estimated was increased. This led to a larger number of parameter combinations to fill the target data space. The input data grew exponentially, increasing the complexity of the estimation problem. The methods used to improve the neural network and its approximation are discussed in this section.

The number of parameters was increased to 5 by adding the inertia variable. The same configuration from the previous training was applied to the new data set of 2500 observations and another unique test set was recorded. The errors for this new parameter set were fairly high, with some parameter errors as large as 40%, leading us to believe that there has been over-fitting. The training process is repeated using the Bayesian Regularization (B-R) method to see if the model can be improved. The training stopped at 1000 epochs with a MSE of 10^{-3}. The performance can be seen in figure 6-12. This model was applied to the unique measurement set to find the target parameters. The results had slightly improved compared to the L-M method, but were still not within an acceptable range. Increasing the maximum number of epochs was not the solution to this problem.

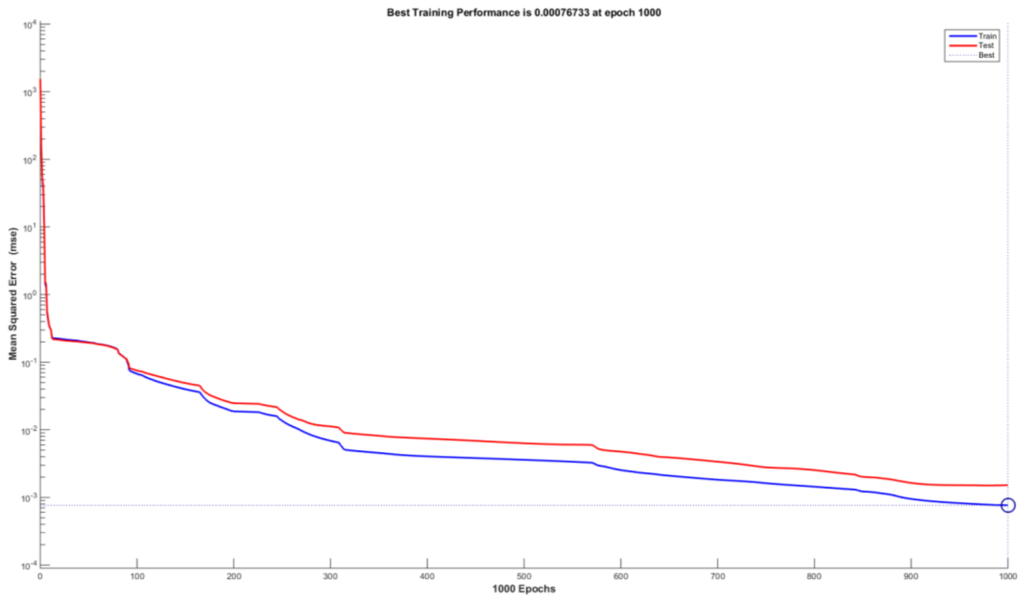


Figure 6-12. Training Performance using B-R Method

When the problem increases in complexity, the sensitivity of the neural network grows. The presentation of our data becomes much more important due to this sensitivity. The significant features from the data must be extracted and normalized. This is where we apply the pre-processing and post-processing steps described earlier. The two most important data transformations are normalizing the parameter vectors and removing the highly correlated measurements. The function normr() is used to normalize the rows in the parameter matrix to a length of 1 using the linear transformation in equation 6-6. The measurements with the least amount of correlation are extracted by applying principle component analysis (PCA) using the function processpca(). This function also standardizes the data between 0 and 1. The default fraction of variance contribution variable, maxfrac, is zero. Increasing this variable will allow for some correlation in the data. After the measurements were processed using PCA, the number of features in our data set had drastically reduced. This step eliminated many redundant measurements that may have produced a bias within the network.

Table 6-4. Parameter Values for NN 2

Estimated Parameters	Constant Parameters
$0 \leq LM \leq 100$	$R_r = 0.013$
$0.012 \leq R_s \leq 0.024$	$X_r = 0.067$
$0.05 \leq X_s \leq 0.2$	$R_2 = 0.009$
$3.5 \leq X_m \leq 6.5$	$X_2 = 0.17$
$0.4 \leq H \leq 1.2$	$D = 0$

The network training for 5 parameter estimation went through many iterations. The parameters and their range are shown in table 6-4. The characteristics of the neural network algorithm were modified and tuned with each training process. The normalization and PCA functions were revised to better suit the data. The number of epochs was reduced to a more reasonable value of 100 and the validation checks were reduced to 3. The best results obtained using the NN characteristics in table 6-5 provided a mean squared error of 1. The highest error found for a parameter estimate was 30%, which is quite large.

The neural network algorithm is further improved by modifying the topology of the network. Another hidden layer is introduced, creating a network of 4 layers as shown in figure 6-13. The second hidden layer consists of 20 hidden neurons. The new network model is trained using the same data set and a slightly revised version of the configuration in table 6-5. The results, in figure 6-14, show that the new network has a mean squared error of less than 10^{-3}, which is very good. However, there still exists a large error of 10% for a particular stator reactance.

Table 6-5. Neural Network Configuration 2

Target Parameters	5
Number of Observations	2500
Training Function	Bayesian Regularization (B-R)
Number of Hidden Layers	1
Humber of Hidden Neurons	50
Training Set	70%
Validation Set	15%
Test Set	15%
Epochs	100
Validation Checks	3
Performance Function	Mean Squared Error

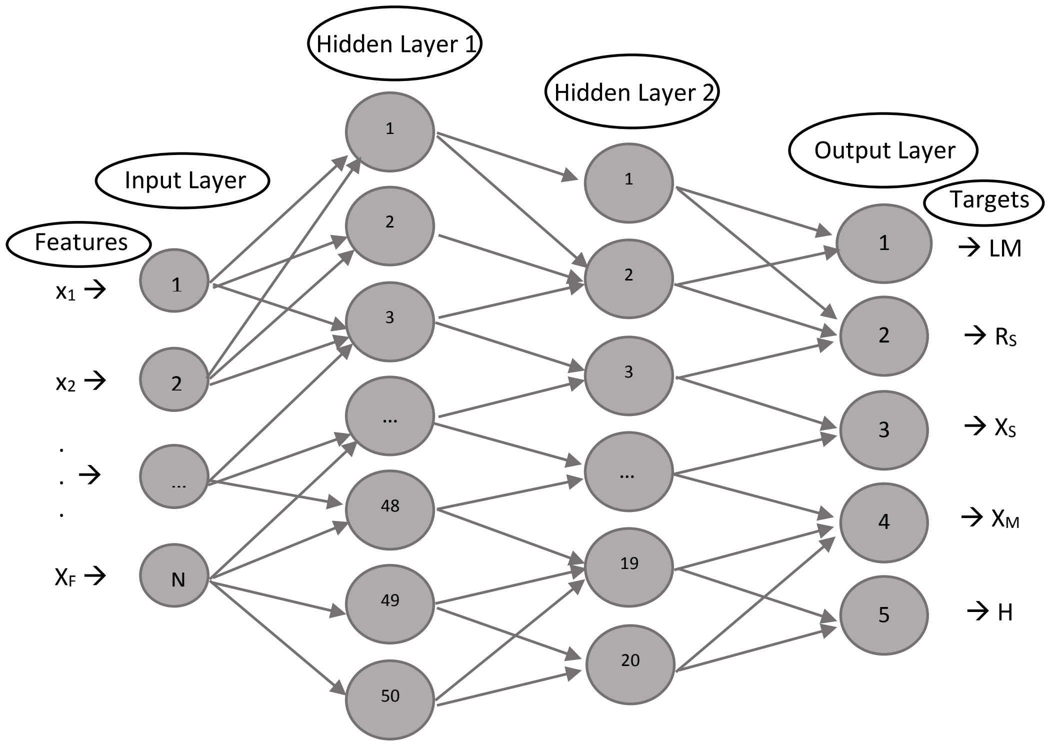


Figure 6-13. Neural Network Topology with Two Hidden Layers

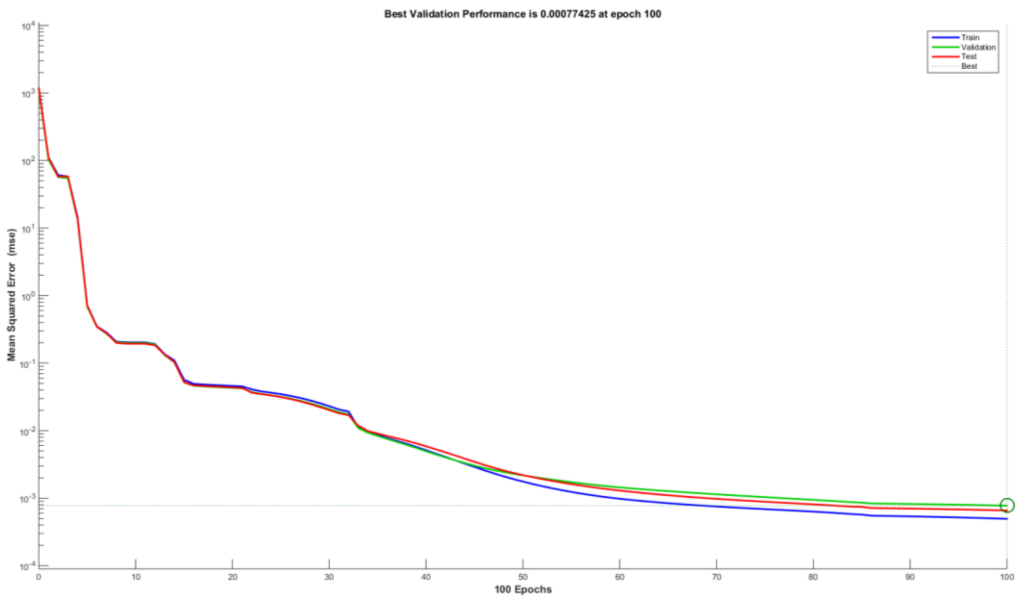


Figure 6-14. Performance of NN 3 with Two Hidden Layers

Another pre-processing step that was applied to the measurement set was the computation of the Fast Fourier Transform (FFT). The FFT converts the time signal into the frequency domain. This step was performed before the PCA since the Fourier requires having the actual time-series data. The same neural network configuration was applied to the new set of features. The new network map produced slightly worse results compared to the previous test. The mean squared error was between 10^{-1} and 10^{-2}. This implies that the FFT does not extract valuable information from this set of measurements, but rather it loses pertinent features from the time-domain data. The FFT will not be applied to any future cases.

Ensemble averaging can prevent the training data from overfitting the model. The next step is to apply the ensemble method of Bagging to the neural network learning process. This technique involves restructuring many components of the neural network program. The data is divided into two subsets, the learning and the test sets. The learning set is comprised of a random selection of 80% of the total data. The test set is not used until all the committee networks are trained. The learning set is bootstrapped sampled 10 times in order to implement a bagging of 10 neural networks. Each bootstrap sample is made up of 70% of the original learning set and 30% are repetitions. The random permutation function, randperm(), is used to create these sample sets. Each learning sample set is further divided into a training and validation set. Each training set is made up of a random sampling of 85% of its corresponding learning set. Once the data sets are initialized, ten neural networks are trained consecutively using their appropriate bootstrapped samples. Each network is then applied to the test set and ten sets of parameters are estimated. The target parameter vector is found by averaging the group of ten parameter vectors. The target parameter error, also known as the committee error, is computed using equations 6-8 and 6-9 below. The code developed for bagging with neural networks can be found in appendix G.

Parameter Vector Error for each Subset, *i*

$$\widehat{E_\iota} = \frac{|\hat{y} - \widehat{f_\iota(x)}|}{(\hat{y}_{max} - \hat{y}_{min})} \qquad (6\text{-}8)$$

where y_{max} and y_{min} refer to the vectors of maximum and minimum values of each parameter, respectively.

Target Parameter Vector Error

$$\widehat{E_t} = \frac{1}{10}\sum_{i=1}^{10} \widehat{E_\iota}$$

$$\widehat{E_t} = \frac{1}{10\,(\hat{y}_{max} - \hat{y}_{min})}\sum_{i=1}^{10} \left|\hat{y} - \widehat{f_\iota(x)}\right| \qquad (6\text{-}9)$$

At this point, the parameter vector has been modified to 6 variables; the inertia has been removed and the rotor resistance and reactance have been added so that the total number of targets are only increased by one. The rotor resistance range is between 0.008 and 0.02 Ohms and the rotor reactance is within 0.05 and 0.2 Ohms. A total of 3000 parameter vectors are created and their corresponding measurements are collected. The neural networks are trained using this new data set with an ensemble averaging of 10. The test set is made up of a unique random sampling of 20% of the total data set. The error for each target parameter based on the test set is shown in table 6-6. The results are very good, with the largest error being under 4%. The next step would be to gradually increase the number of parameters being estimated.

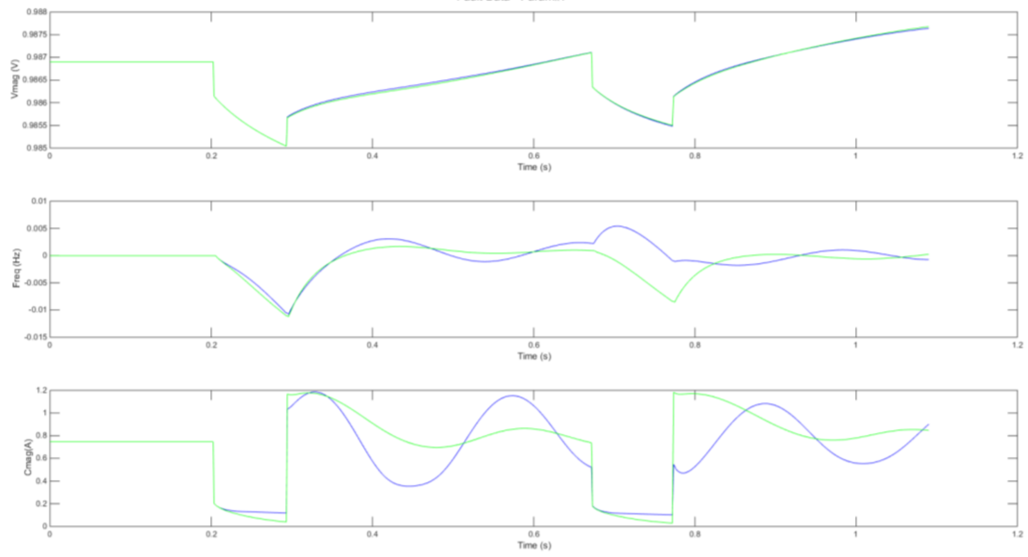
Table 6-6. Error for 6 Target Parameters using Bagging

Target Parameter	Committee Error (%)
LM	0.1613
R_S	0.8010
X_S	1.9591
X_M	3.9138
R_R	0.3240
X_R	1.9413

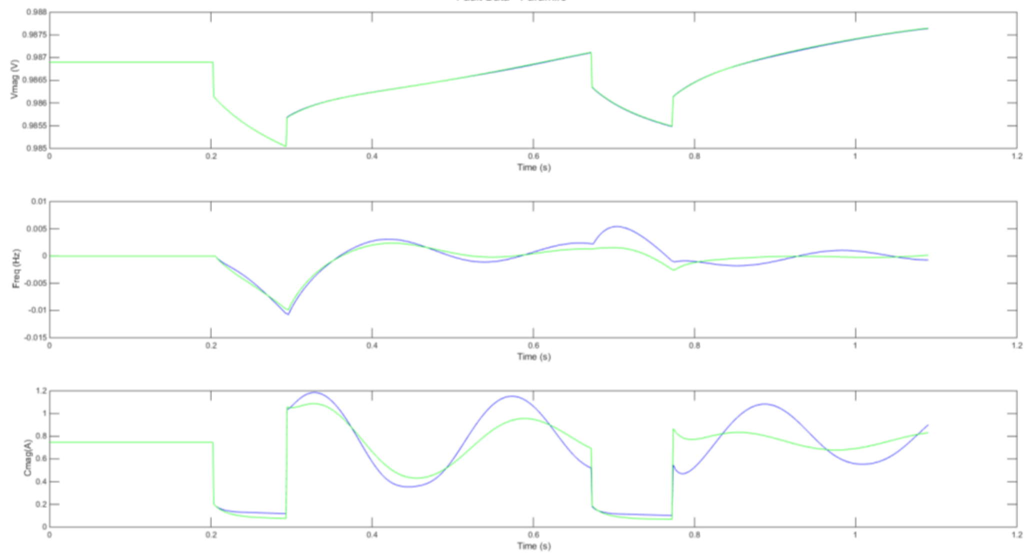
6.4.4 Optimizing Neural Network

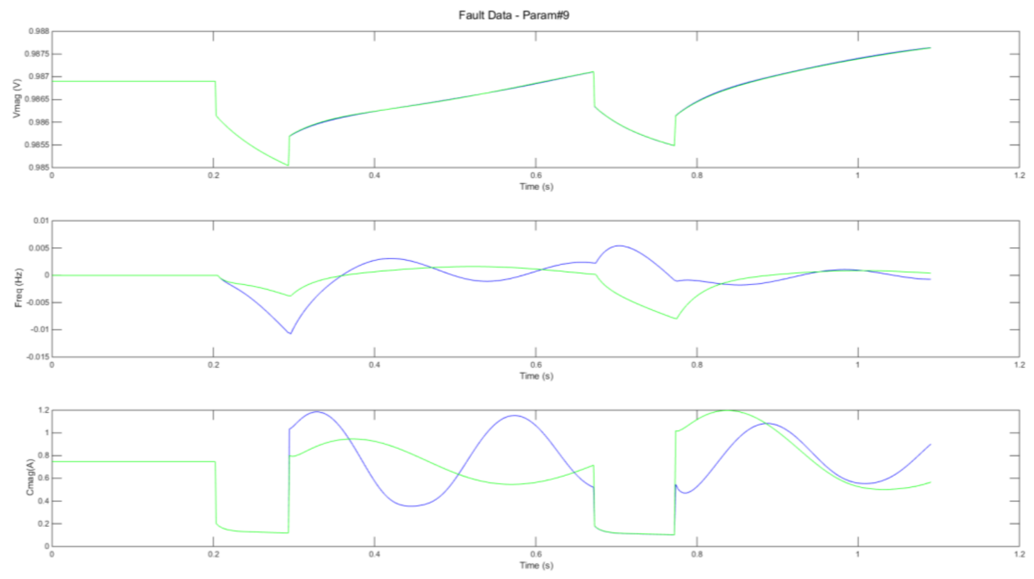
The final goal in this project is to successfully train a neural network so that it is capable of estimating all the load model parameters that have an effect on the dynamics of the power system. Ten significant parameters of the complex load were described earlier and thus far, we have been able to correctly estimate six of them. All parameters must create enough variation in the measurements in order to be estimated. In order to check this variation, a comparison was made between the minimum and maximum value of each parameter. The base case consisted of a load model where all the parameters were at their minimum value. The simulation was executed and the voltage magnitude, current magnitude and frequency was recorded. The same process was repeated 10 more times; in each case, a single parameter was set to its maximum value. The plots used to test the variation of the last four parameters are shown in figures 6-15 (a) – (d). The plots for the first six parameters are stored in appendix H. As you can see, the plots associated with R_2, X_2, and H, in 6-15 (a) to (c), show some visible variation. However, the last plot, 6-15 (d), comparing the measurements for damping factor parameter, D, shows very little variation, and therefore cannot be estimated. This reduces the parameter estimation problem to a possible nine variables.

Fault Data - Param#7



Fault Data - Param#8





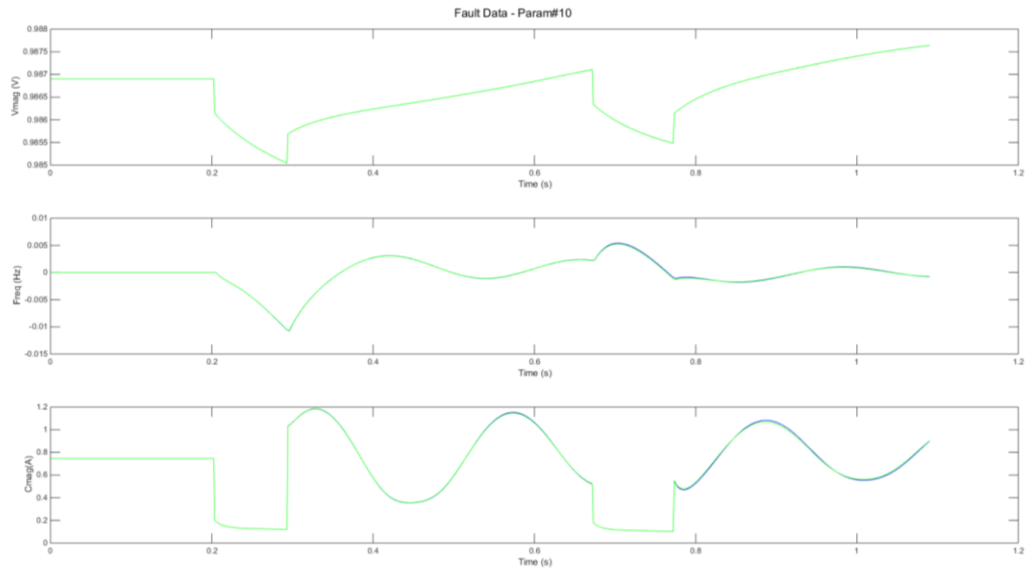


Figure 6-15 (a) – (d). Parameters R_2, X_2, H and D

The number of estimated parameters is increased to seven; the inertia variable is added to the set. Initially, the data set was collected measurements for 4000 parameter vectors; however, this provided poor results. The data set was then increased to over 9000 parameter vectors, filling in the target data space. Similar to the six parameter method, the neural network was trained using 10 bootstrap samples. The results are tabulated below.

Table 6-7. Error for 7 Target Parameters

Target Parameter	Committee Error (%)
LM	0.2126
R_S	0.9981
X_S	2.5276
X_M	5.9040
R_R	0.5562
X_R	2.4830
H	0.03851

The same process is repeated for all 9 parameters using the range of values listed in the table 6-8. Over 83,000 parameter combinations and their corresponding measurements were saved in the new data set. The targets were normalized and PCA was applied to the inputs for feature extraction. The neural network configuration and design was modified many times to obtain the best results. The configuration used for 9-parameter estimation is shown in table 6-9. The network shown in figure 6-16 is captured using the function-fitting tool in MATLAB. The results for the target parameters are tabulated in table 6-10.

Table 6-8. Values for 9 Parameters

Estimated Parameters	Constant Parameters
$0 \le LM \le 100$	D = 0
$0.012 \le R_s \le 0.024$	
$0.05 \le X_s \le 0.2$	
$3.5 \le X_m \le 6.5$	
$0.008 \le R_R \le 0.02$	
$0.05 \le X_R \le 0.2$	
$0.005 \le R_2 \le 0.015$	
$0.1 \le X_2 \le 0.2$	
$0.4 \le H \le 1.2$	

Table 6-9. Neural Network Configuration for 9 Parameters

Target Parameters	9
Number of Observations	83510
Training Function	Bayesian Regularization (B-R)
Number of Hidden Layers	2
Humber of Hidden Neurons	50 - 20
Bagging	10
Learning Set	80%
Test Set	20%
Bootstrap Sample Ratio	70/30 % Repetition
Training Set	85% of Bootstrap Sample
Validation Set	15% of Bootstrap Sample
Epochs	100
Validation Checks	5
Error Function	Bagging Percent Error

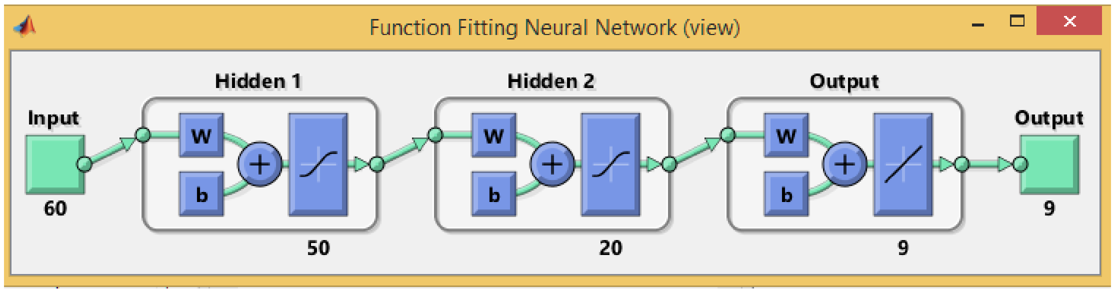


Figure 6-16. 9 Parameter Neural Network

Table 6-10. Error for 9 Target Parameters

Target Parameter	Committee Error (%)
LM	0.470782325987197
R_S	2.73958684910263
X_S	5.56063990373774
X_M	8.46258160911934
R_R	1.46231426002560
X_R	5.87447310379827
R_2	1.71262311919912
X_2	3.07359453764188
H	0.873924000749324

Now that we are able to successfully estimate 9 of the complex load model parameters; the final step is to improve these results by reducing their error. The error function can be optimized by modifying the topology of the neural network. The number of hidden layers and hidden neurons in each layer has remained constant since the introduction of the second hidden layer. A simple optimization program is developed to find the best network topology.

As the number of target parameters increased, so did the size of the data set and the complexity of the network. These characteristics exponentially increased the computational time for the learning algorithm. The 9-parameter estimation using a committee of 10 bags takes approximately 7 days to run on an Intel i7 quad core processor with 32GB of RAM. The optimization program could takes months, if not years to complete for the 9-parameter model. This is why the initial optimization was performed on the 4-parameter network and the resultant optimal topology was applied on the 9 parameter network.

First, the neural net for 4 target parameters is optimized on the 2 hidden layer model. The base case starts with 50 and 20 hidden neurons in the hidden layers, respectively. The number of hidden neurons in the first hidden layer is increased in increments of one until the error does not improve. If the error increase on the first iteration then the number of hidden neurons are decreased in increments of one until the error stops decreasing. If the error increases in either direction, then the number of hidden neurons remains unchanged. The same process is applied to the second hidden layer until the error has reached a minimum. The optimal network design for two hidden layers would have 55 and 20 hidden neurons, respectively. The target parameter error is stored to be compared later on.

Next, the number of hidden layers is increased to three with the initialization of 50, 20 and 15 neurons, respectively. The same optimization process is applied to this new model. Since the new model is slightly more complex, the optimization process is applied twice such that hidden layer one is repeated after the first attempt at hidden layer 3; this is followed by hidden layers, two and three, respectively. The optimization code can be found in appendix I.

The resultant network topology for three hidden layers converges to 56, 25 and 16 hidden neurons, respectively. The target parameter errors for the original network, the optimal 4-layer network and the optimal 5-layer network are compared in table 6-11. As you can see, the error of each target parameter has been reduced in each optimization step.

Table 6-11. 4 Target Parameter Error Comparison

Target Parameter	Original Network Error (%)	Optimal 4 Layer Network Error (%)	Optimal 5 Layer Network Error (%)
LM	0.03055	0.01066	0.01009
R_S	0.80790	0.32791	0.11265
X_R	0.19967	0.05834	0.04099
X_M	0.14912	0.03381	0.02041

Using this new neural network layout, the 9-parameter estimation algorithm is performed to see if the error is reduced. Once these results are compared, the number of hidden neurons in each hidden layer are incremented by one, individually, to see how the error changes. The same process is repeated but the number of hidden neurons is decremented by one this time. After comparing all the error vectors, it is found that [56 25 16] is indeed the optimal neural network topology, even for a 9-parameter estimation model. The target parameter error using the original network and the optimal network are compared in table 6-12. The results show that the error has decreased for every target parameter.

Table 6-12. 9 Target Parameter Error Comparison

Target Parameter	Original Network Error (%)	Optimal 5 Layer Network Error (%)
LM	0.4708	0.3328
R_S	2.7396	1.6216
X_s	5.5606	4.9363
X_M	8.4626	8.4110
R_R	1.4623	0.9818
X_R	5.8744	5.1784
R_2	1.7126	1.1556
X_2	3.0736	2.0022
H	0.8739	0.5882

The final neural network topology is illustrated in figure 6-17.

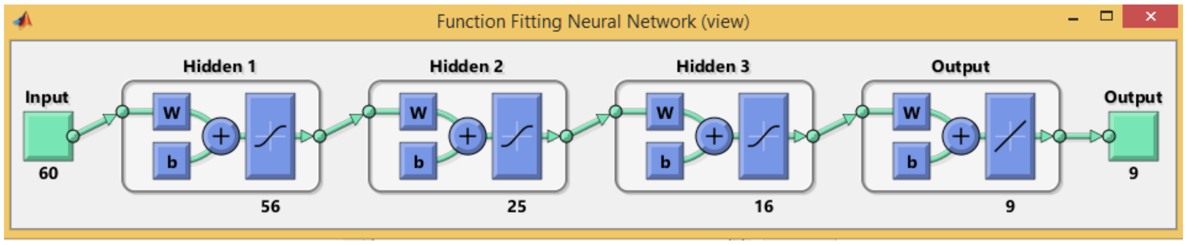


Figure 6-17. Final Neural Network Model

6.5 Conclusions

The final neural network designed here has an average error of 2.8% per parameter. The results show that the maximum errors are contributed to the reactance variables, X_S, X_M, and X_R, with committee errors up to 4.94%, 8.41% and 5.18%, respectively. Errors within this range will not generally produce a significant difference between the dynamic behaviour of the model compared to the behaviour of the actual system. This can be demonstrated by comparing the response at the load bus between the correct model parameters and the estimated model parameters using the errors shown in table 6-12. Plots of the voltage magnitude, current magnitude and frequency are shown in figure 6-18. The blue curves correspond to the correct dynamic model and the green curves to the estimated dynamic model.

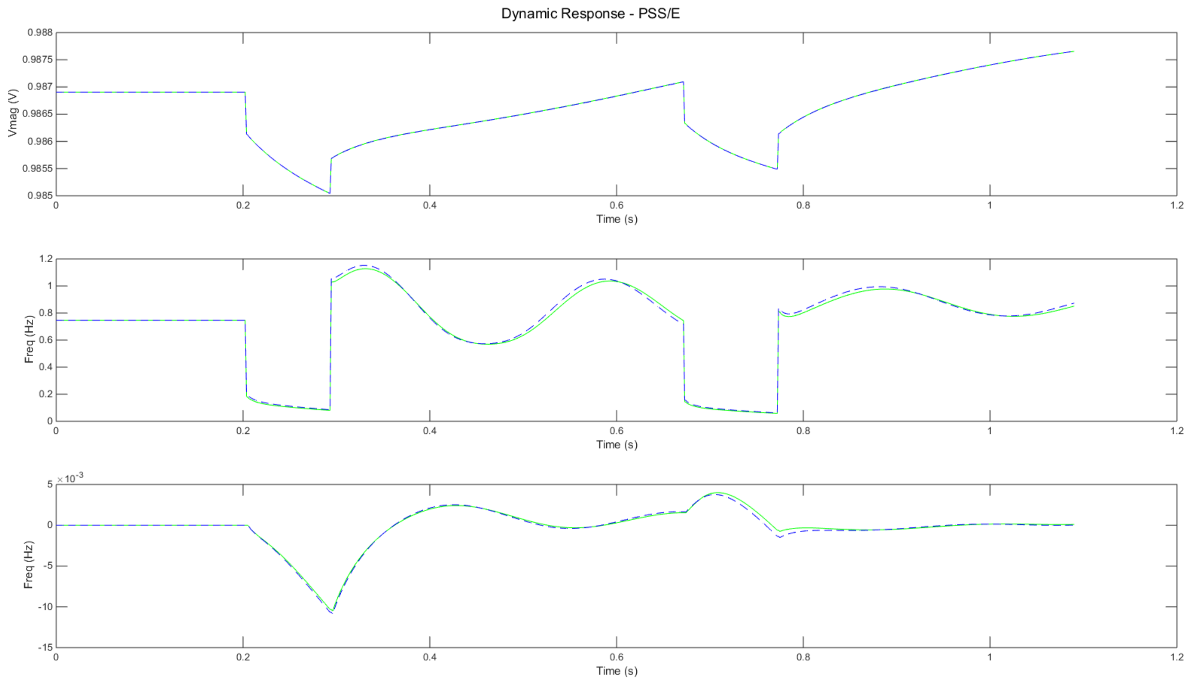


Figure 6-18. Dynamic Response of Actual and Estimated Load Models

Chapter 7. Conclusion

The research conducted in this dissertation had two main goals; one was to improve the accuracy of our load models to perform better contingency analysis and the other was to increase our understanding of synchrophasor technology and expand its applications. The advancements we made towards load modeling would not have been possible without the use of phasor measurements. Even though wide area phasor measurements were introduced many decades ago, power utilities had made very little progress towards integrating them with the existing EMS. Recently, there has been much more focus on creating a “smarter” and more reliable power grid. One of the key components in achieving this goal is by taking advantage of benefits provided by synchrophasors. This has encouraged power entities to finally invest resources into the implementation and integration of this technology, starting with the higher voltage regions. The synchronized, high frequency, time-stamped measurements will allow us to develop new innovative solutions for our online and off-line applications.

7.1 Major Contributions

The main contributions from this work are developing measurement-based methods to estimate load model parameters. These parameters are inserted into the defined load model for a particular or subset of load buses. The end goal is to conduct reliable contingency studies by running simulations on the power system model, which includes these updated load bus models. The two estimation methods designed are distinguished between static and dynamic models.

1. Static ZIP Load Model Parameter Estimation: This method finds the composition of constant impedance (Z), constant current (I), and constant power (P) in the ZIP load model. It applies the Bounded-Variable Least Squares (BVLS) algorithm to the phasor measurements collected at the load bus to approximate the three parameters. Each parameter is bounded between 0 and 1.

2. Dynamic Composite Load Model Parameter Estimation: This method finds the parameters required to model the composite load, which is made up of a static component and an induction motor load. Two of the parameters contribute to the percent composition of the static and induction loads, respectively. The rest of the parameters account for the dynamic properties of the induction machine. The Neural Network algorithm is used on the simulation-based measurement set to train a network model for the load. The derived network model is applied to the real phasor measurement set to approximate the parameters.

Both of these techniques are significant improvements over the methods that are currently used in the industry and they provide better results than many other load-modeling studies found in the literature. The BVLS method for static models provides an accurate parameter estimation compared to the regular Least Squares approximation. This method accounts for the reasonable amount of error in measurement sets, which may prevent regular Least Squares from providing a unique solution. This technique is also less likely to converge at an incorrect local minimum compared to other variations of the Least Squares approximation due to the bounded variables. Very few advanced algorithms are being utilized in the power industry to model dynamic loads. There have been a few recent studies in this area and they have mainly focused on using optimization techniques to converge to the load model parameters. Our machine learning method gives us the benefit of having greater control over the parameter estimation. We can evaluate and tune the amount of acceptable error in our load model based on our requirements and data set. We can also easily change our computation time by modifying the structure of the neural network and selection of the training algorithm. Unlike the optimization approach, the parameters of the load model can be estimated in almost real-time once the network model has been trained. This is a huge advantage towards online load modeling and EMS. Similarly, the BVLS method for static load modeling can also be applied in almost real-time due to its low computational time. Both, the static and dynamic load modeling methods developed here, are applied on data collected using realistic events in the power system. The type of events used in the research are important because they must have a high rate of occurrence in the real power grid. Some of the studies that were reviewed had conducted experiments that required unrealistic events to occur in order to obtain positive results. The severe events introduced during these experiments provided a wide range of variability in the measurements. This variability reduced the correlation in the measurement set, thus making the parameter estimation process much easier by delivering unique results. Since the events used in this study were realistic and non-severe, the measurement sets had some correlation. This made designing the approximation algorithms more difficult, yet robust.

One significant contribution from this work is the development of the phasor measurement algorithm (PMU). Until now, many studies that involve simulated phasor measurements have made the assumption that the simulated data represents the real PMU data. Once these experiments are applied to the real power system, the results will vary drastically, if not become completely invalid due to the discrepancies between the theoretical and actual data. Our development process has shown the differences between the real and simulated data and the final algorithm that was derived can be applied to almost any simulated measurement set to provide a more realistic representation of phasor data. After applying the PMU algorithm to the measurements used for dynamic load modeling, the measurement set becomes less ideal and the results and errors are more similar to those from a real system. The neural network model is trained using simulated data even when integrated into a real Energy Management System. The PMU algorithm will still need to be used on the simulated data so that the trained network model is accurate enough to be applied to the real measurement set. These are the reasons for why the PMU algorithm is so important in studies using simulated data. The algorithm derived here can be modified to fit other types and models of PMUs.

One of the initial contributions that led to the proposal of this entire project is the study of the effect of load models on contingency ranking. The general procedure to calculate the principal index (PI) after each outage was found and the PIs were ranked to derive the list of worst-case contingencies. The rankings computed for three unique ZIP load compositions proved that the load-type had a noticeable effect on the post-outage voltages and power flows.

This result negated the assumption that load type has no effect in steady state contingency studies. This test found that not all loads should continue to be modelled as 100% constant power type during contingency analysis.

N-1 contingency analysis can be quite time consuming for large power systems since the load flow must be calculated after every outage and compared to the pre-outage measurements. Sensitivity methods were studied in order to perform contingency analysis within a much smaller computation time. The Line Outage Distribution Factor (LODF) approach, developed by Sauer, was modified and tested in order to find the worst case contingencies based on three types of static load models. A new voltage sensitivity method was developed to obtain the same objective as the LODF approach. Even though both techniques were able to estimate post-outage measurements for some specific contingency cases, this was not sufficient for an accurate contingency ranking. These methods are still quite valuable in measuring sensitivities in the system using synchrophasors.

7.2 Conclusions

This study brings light to the difficulties faced in accurate contingency analysis of an evolving power system due to the demand at the load buses. The effects of contingencies can be predicted by simulating the events that occur. The events are created by modeling a realistic representation of the power system in a software simulation package. The system can be modeled using its static properties or its combined dynamic and static properties. The effects being studied will decide on what type of model is required. The accuracy of the contingency analysis is dependent on the validity of the generated system models. Load models have generally been the most complicated to predict; where invalid assumptions and lack of real data have led to incorrect models. Recent growth in phasor measurement technology and machine learning algorithms will allow us to develop precise static and dynamic models of the load. The research conducted here focused on developing new load modeling techniques using the synchrophasors. Power engineers can apply these new measures to improve the current Energy Management System and conduct better planning studies.

The Bounded-Variable Least Squares (BVLS) approach to estimating static load model parameters has proven to be the most accurate among the methods studied and tested. Each of the three parameters can be approximated within a 6% error margin based on the test cases. This amount of error is very reasonable and does not significantly affect our contingency analysis results. As shown in chapter 5, even if each load parameter carried an error of 6%, the contingency ranking would remain the same compared to the correct load parameter based ranking. Since this error is calculated based on our selective parameters sets, and not derived through a mathematical proof, there is a possibility of having a larger error with a new parameter set. These results are consistent only when there is a minimum of 2% variation in the bus voltage measurements. Anything less than 2% will result in a highly correlated measurement matrix. The events applied to the IEEE 118 bus system during our tests provided this voltage variation. The events required to produce the same variation on another system may differ. In a much larger power system, the effects of scaling the load or generation may not cascade far enough to cause significant changes.

Dynamic load modeling contributed to the largest portion of this study. After realizing the disadvantages of optimization techniques, machine learning concepts and algorithms were reviewed. The neural network method was chosen over other approaches due to its non-linear regression capabilities. The neural network configuration designed in chapter 6 was able to accurately approximate all nine of the composite load model parameters. Similar to the static load-modeling problem, the measurement set must have some variation in it. The fault used to create these variations for the IEEE 118 bus system may not have the same effect on a different power system. The error calculated for each parameter estimate is an average error. This does not provide us with a maximum possible error for that parameter. The error can be reduced by having a comprehensive data set; however, the time required to collect this amount of data will increase proportionally and the computation time for the neural network training will increase exponentially.

Other limitations of both approximation methods include the assumptions made during the collection of the measurement sets. The data is used to predict one unique parameter set; therefore, it is assumed that the load model is the same for the whole measurement set. In reality, this may not always hold true. The load composition and properties may change during the time the measurements are collected. A large measurement set is collected over a long period where the load can easily change. The errors found in the parameter estimation may increase in real world applications when the loads are not perfectly consistent throughout the measurement set.

The PMU algorithm developed here is capable of transforming the simulated data into much more realistic PMU looking data. This bridges the gap between research conducted in a controlled simulation environment and a fully developed procedure that is applied to a real power system. This algorithm is also a useful tool in offline applications. The program must be modified according to the simulation data settings and the real PMU that it is trying to emulate. Even after these modifications and fine-tuning, the simulated data may not look exactly like the real data. These differences are apparent in the plots shown in chapter 4 where the PMU algorithm output is compared to the real PMU output.

7.3 Future Work

One of the largest potential areas for furthering this research would be to extrapolate the static and dynamic load model estimation methods to include more than one load bus. Ideally, the algorithm is complete only when the parameters for every single load bus can be estimated accurately. As the number of load buses increase, the number of parameters increase, therefore requiring a much larger data set. The complexity of the neural network model will have to increase for the dynamic model estimation. There may even be a finite limit to the number of parameters that can successfully be estimated for both, the dynamic and the static load models. The computational time will absolutely increase for both methods.

The neural network algorithm developed for dynamic load modeling was tested on the induction motor model, IEEE CIM5BL, combined with a static load. The next step in this study would be to apply this process to the new WECC complex load model and estimate its parameters. This new model incorporates the many types of load components, including the induction motor. This will allow the dynamic load modeling algorithm to be applied to almost any load bus.

One important area for improvement is to expand the types of faults that can be used in the dynamic load model approximation. In this study, only the three-phase fault was used; this limits the capabilities of this algorithm since it can only be applied under a particular circumstance. When other types of faults, such as single-phase faults, are each used to train a unique neural network, then the probability of estimating the load model is increased. This is because the likelihood of any type of fault occurrence is much higher than the three-phase fault. The neural network structure may need to be modified according to the type of fault data collected. The power system software, PSS/E, was used to simulate the three-phase fault near the load bus. Single phase faults cannot be simulated in PSS/E, therefore another software package must be found, which includes all the necessary faults and can perform dynamic simulations on a large power system.

The load modeling methods developed in this dissertation have been executed on a regular CPU with quad core processing capabilities. The memory of the CPU has been increased to 32 GB to accommodate the machine learning algorithm for the 9 parameter approximation. The code for both static and dynamic load modeling has been mainly written using MATLAB. After implemented the other future work described in this section, the computational time required to approximate the static and the dynamic loads will increase, especially for a large power system. The static load modeling may still be feasible with the current hardware and software combination; however, the machine-learning algorithm will not be feasible anymore. The computational time of these algorithms can be slightly reduced by re-writing the code in a lower level programming language such as C. However, this may not be a sufficient improvement for the neural network algorithm. The proposed solution to this problem is to re-develop these algorithms using advanced parallel processing computing such as graphics processing unit (GPU). Recently, GPU programming combined with a CPU has made some major waves in the computing industry. Companies such as NVIDIA have introduced APIs in more user-friendly languages that are compatible with their GPUs. NVIDIA's CUDA program has become quite popular in non-graphics coding due to its ease of use with the C language. This method has a huge potential to improve computational time in neural network training as well as many other areas of power systems. The load modeling methods developed in this study can only be applied to a real power system after this future research has successfully been completed.

Bibliography

[1] Federal Energy Regulatory Commission and North American Electric Reliability Corporation, "Arizona-Southern California Outages on September 8, 2011," FERC, 2012.

[2] M. Pai, P. Sauer, B. Lesieutre and R. Adapa, "Structural Stability in Power Systems - Effect of Load Models," in *IEEE/NTUA Athens Power tech Conference: Planning, Operation and Control of Today's Electric Power Systems*, Athens, 1993.

[3] L. Xie, D. H. Choi, S. Kar and H. V. Poor, "Fully Distributed State Estimation for Wide-Areaa Monitoring Systems," *IEEE Transactions on Smart Grid,* vol. 3, no. 3, pp. 1154-1169, 2012.

[4] D. Atanackovic and G. Dwernychuk, "B.C. Hydro Approach to Load Modeling in State Estimation," in *Power and Energy Society General Meeting*, Vancouver, 2012.

[5] A. Phadke, "Synchronized phasor measurements-A historical overview," in *Transmission and Distribution Conference and Exhibition 2002: Asia Pacific*, Yokohama, 2002.

[6] J. Li, X. Xie, J. Xiao and J. Wu, "The Framework and Algorithm of a New Phasor Measurement Unit," in *IEEE International Conference on Electric Utility Deregulation, Restructuring and Power Technologies*, Hong Kong, 2004.

[7] "Phasor RTDMS," Consortium for Electric Reliability Technology Solutions, [Online]. Available: http://www.phasor-rtdms.com/phaserconcepts/phasor_adv_faq.html. [Accessed 2015].

[8] K. D. Jones, A. Pal and J. S. Thorp, "Methodology for Performing Synchrophasor Data Conditioning and Validation," *IEEE Transactions on Power Systems,* vol. 30, no. 3, pp. 1121-1130, 2015.

[9] S. Tsai, "Study of Global Power System Frequency Behaviour Based on Simulations and FNET Measurements," Virginia Tech, Blacksburg, 2005.

[10] H. T. Ma and B. H. Chowdhury, "Dynamic Simulations of Cascading Failures," in *38th North American Power Symposium, 2006. NAPS 2006*, Carbondale, 2006.

[11] P. Sauer, "On the Formulation of Power Distribution Factors for Linear Load Flow Methods," *IEEE Transactions on Power Apparatus and Systems,* Vols. PAS-100, no. 2, 1981.

[12] Y. C. Chen, P. W. Sauer and A. D. Domınguez-Garcıa, "Online Computation of Power System Linear Sensitivity Distribution Factors," in *Bulk Power System Dynamics and Control - IX Optimization, Security and Control of the Emerging Power Grid (IREP), 2013 IREP Symposium*, Rethymno, 2013.

[13] A. M. Ilic-Spong, "Redistribution of reach power flow in contingency studies," *IEEE Transactions on Power Systems,* vol. 1, no. 3, 1986.

[14] S. Singh, "Improved Voltage and Reactive Power Distribution for Outage Studies," *IEEE Transactions on Power Systems,* vol. 12, no. 3, 1997.

[15] D. Chatterjee, J. Webb, Q. Gao, M.YVaiman, M. Vaiman and M. Povolotskiy, "N-1-1 AC Contingency Analysis as a Part of NERC Compliance Studies at Midwest ISO," in *IEEE PES Transmission and Distribution Conference and Exposition*, New Orleans, 2010.

[16] H. Retty, "The Effect of Load Model Composition on N-1 Contingency Ranking," in *IEEE PES Innovative Smart Grid Technologies*, Washington, 2015.

[17] Western Electricity Coordinating Council, "Composite Load Model for Dynamic Simulations," WECC MVWG, 2012.

[18] P. Kundur, Power System Stability and Control, Toronto: McGraw Hill, 1994.

[19] J. Wei, J. Wang, L. Yang and Q. Wu, "Power System Aggregate Load Area Modeling," in *IEEE/PES Transmission and Distribution Conference & Exhibition: Asia and Pacific*, Dalian, 2005.

[20] "WECC Dynamic Composite Load Model Specifications," Western Electricity Coordinating Council, 2015.

[21] R. Nath, "PSS/E Models for Dynamic Simulation of Somposite Load," Siemens Energy, Inc, 2010.

[22] Seimens PTI PSS/E, "PSS/E 32.0 Model Library," Seimens Energy, Inc, Schenectady, 2009.

[23] J. D. Glover, M. S. Sarma and T. J. Overbye, Power System Analysis and Design, Stamford: Cengage Learning, 2010.

[24] B. Lesieutre, P. Sauer and M. Pai, "Development and Comparative Study of Induction Machine Based Dynamic P, Q Load Models," *IEEE Transactions of Power Systems,* vol. 10, no. 1, pp. 182-191, 1995.

[25] P. Kundur, Power System Stability and Control, Toronto: McGraw Hill, 1994.

[26] S. XiongHua, C. GenJun, J. Ping and Z. DaoNong, "A new generalized load model of power systems and its applications," in *The International Conference on Advanced Power System Automation and Protection*, Nanjing, 2011.

[27] B. Choi and H. Chiang, "On the Local Identifiability of Load Model Parameters in Measurement-based Approach," *Journal of Electrical Engineering & Technology ,* vol. 4, no. 2, pp. 149-158, 2009.

[28] S. He, "Modeling Power System Load using Intelligent Methods," University of Waterloo, Waterloo, 2006.

[29] J. Zhang, J. Wen, S. Cheng and Z. Dong, "Realization of the WAMS Based Power System Aggregate Load Area Model," in *Power and Energy Society General Meeting - Conversion and Delivery of Electrical Energy in the 21st Century*, Pittsburgh, 2008.

[30] Q. Chen and J.D.McCalley, "Identifying High Risk N-k Contingencies for Online Security Assessment," *IEEE Transactions on Power Systems,* pp. 823-834, 2005.

[31] Z. Hussain, Z. Chen and P. Thogersen, "Fast and Precise Method of Contingency Ranking in Modern Power System," in *IEEE Jordan Conference on Applied Electrical Engineering and Computing Technologies*, Jordan, 2011.

[32] A. O. Ekwue, "A review of automatic contingency selection algorithms for online security analysis," in *Third International Conference on Power System Monitoring and Control*, London, 1991.

[33] North American Electric Reliability Corporation, "Evaluation of Criteria, Methods and PRactices Used for System Design, Planning and Analysis Response to NERC Blackout Recommendation 13c," NERC, Washington, D.C, 2005.

[34] I. Musirin, T. Khawa and A. Rahman, "Simulation Technique for Voltage Collapse Prediction and Contingency Ranking in Power System," in *Student Conference on Research and development Proceedings*, Shah Alam, 2002.

[35] IIT Roorkee, "NPTEL E-Learning Courses from the IIts and IIsc," 2014. [Online]. Available: http://nptel.ac.in/courses/108107028/. [Accessed Aug 2014].

[36] University of Washington - Electrical Engineering, "Power Systems Test Case Archive," May 1993. [Online]. Available: http://www.ee.washington.edu/research/pstca/pf118/pg_tca118bus.htm. [Accessed May 2012].

[37] R. Baldick, "Variation of distribution factors with loading," *IEEE Transactions on Power Systems,* vol. 18, no. 4, pp. 1316-1323, 2003.

[38] J. Zhu, Optimization of Power System Operation, Hoboken: John Wiley & Sons, Inc, 2009.

[39] A. Wood, B. Woolenberg and G. Sheble, Power Generation, Operation and Control, Hoboken: John Wiley & Sons, Inc, 2014.

[40] R. Christie, "Power Systems Test Case Archive," May 1993. [Online]. Available: http://www.ee.washington.edu/research/pstca/pf118/pg_tca118bus.htm. [Accessed Apr 2014].

[41] P. Ruiz and P. Sauer, "Voltage and Reactive Power Estimation for Contingency Analysis Using Sensitivities," *IEEE Transactions on Power Systems,* vol. 22, no. 2, pp. 639-647, 2007.

[42] Y. Chen, A. Dominguez-Garcia and P. Sauer, "Measurement-Based Estimation of Linear Sensitivity Distribution Factors and Applications," *IEEE Transactions on Power Systems,* vol. 29, no. 3, pp. 1372-1382, 2014.

[43] K. Hatipoglu, I. Fidan and G. Radman, "Investigating Effect of Voltage Changes on Static ZIP Load Model in a Microgrid Environment," in *North American Power Symposium*, Champaign, 2012.

[44] T. Overbye, M. Pai and P. Sauer, "Some Aspects of the Energy Function Approach to Angle and Voltage Stability Analysis in Power Systems," in *Proceedings of the 31st IEEE Conference on Decision and Control, 1992*, Tucson, 1992.

[45] H. Retty, "Phasor Measurement Based Voltage Sensitivities for Contingency Analysis," in *IEEE Power Engineering Society General Meeting*, Denver, 2015.

[46] M. A. Pai, Computer Techniques in Power System Analysis, New Delhi: Tata McGraw-Hill Publishing Company Ltd., 1980.

[47] C. P. Salomon, G. Lambert-Torres, H. Martins, C. Ferreira and C. Costa, "Load Flow Computation via Particle Swarm Optimization," in *IEE/IAS International Conference on Industry Applications*, Sao Paolo, 2010.

[48] A. Phadke and J. Thorp, Synchronized Phasor Measurements and Their Applications, Springer Science & Business Media, 2008.

[49] IEEE PES WG - Synchrophasor Measurements for Power Systems, "C37.118.1-2011 - IEEE Standard for Synchrophasor Measurements for Power Systems," 2011. [Online]. Available: https://standards.ieee.org/findstds/standard/C37.118.1-2011.html. [Accessed 11 2011].

[50] Schweitzer Engineering Laboratories, "Legacy SEL-311C Data Sheet," 2013. [Online]. Available: https://www.selinc.com/LegacySEL-311C/. [Accessed Aug 2014].

[51] Schweitzer Engineering Laboratories, "SEL-735 Power Quality Meter Data Sheet," 2014. [Online]. Available: https://www.selinc.com/SEL-735_Power_Quality_Meter/. [Accessed Aug 2014].

[52] M. Tesfasilassie, M. Zarghami, M. Vaziri and A. Rahimi, An Estimative Approach for CVR Effectiveness Using Aggregated Load Modeling, Washington, DC: IEEE PES Innovative Smart Grid Technologies Conference (ISGT), 2014.

[53] S. Han, J. Kim, B. Lee, H. Song, H. Kim, J.-h. Shin and T.-k. Kim, "Measurement-based Static Load Modeling Using the PMU data Installed on the University Load," *Journal of Electrical Engineering & Technology,* vol. 7, no. 5, pp. 653-658, 2012.

[54] G. Strang and K. Borre, Linear Algebra, Wellesley, 1997.

[55] C. L. Lawson and R. J. Hanson, Solving Least Squares Problems, Englewood Cliffs: Society for Industrial and Applied Mathematics, 1995.

[56] The Mathworks, Inc, "lsqcurvefit," The Mathworks, Inc, 2013. [Online]. Available: http://www.mathworks.com/help/optim/ug/lsqcurvefit.html. [Accessed 2014].

[57] K. Mullen, "Package: The Stark-Parker algorithm for bounded-variable least squares," [Online]. Available: http://cran.r-project.org/web/packages/bvls/bvls.pdf. [Accessed Dec 2013].

[58] P. Stark and R. PArker, "Bounded-Variable Least-Squares: an Algorithm and Applications," Berkeley.

[59] L. Rodriguez-Garcia, S. Perez-Londono and J. Mora-Florez, "A Methodology for Composite Load Modeling in Power Systems Considering Distributed Generation," in *Transmission and Distribution: Latin America Conference and Exposition (T&D-LA), 2012 Sixth IEEE/PES*, Montevideo, 2012.

[60] L. Rodriguez-Garcia, S. Perez-Londono and J. Mora-Florez, "Particle Swarm Optimization applied in Power System Measurement-Based Load Modeling," in *IEEE Congress on Evolutionary Computation*, Cancun, 2013.

[61] J. Wen, S. Wang, S. Cheng, Q. Wu and D. Shimmin, "Measurement Based Power System Load Modeling Using A Population Diversity Genetic Algorithm," in *POWERCON International Conference on Power System Technology*, Beijing, 1998.

[62] P. Zhang and H. Bai, "Derivation of Load Model Parameters using Improved Genetic Algorithm," in *Third International Conference on Electric Utility Deregulation and Restructuring and Power Technologies, 2008.*, Nanjing, 2008.

[63] J. Zhu, Optimization of Power System Operation, Hoboken: John Wiley & Sons, Inc, 2009.

[64] X. Yanhui, S. Dajun and Q. YingChun, "Research on Feability of Composite Load Modeling Based on WAMS," in *Asia-Pacific Power and Energy Engineering Conference (APPEEC)*, Wuhan, 2011.

[65] D. Batra, "Introduction to Machine Learning Lecture," Blacksburg, 2015.

[66] T. Hastie, R. Tibshirani and J. Friedman, The Elements of Statistical Learning, Springer, 2008.

[67] D. Barber, Bayesian Reasoning and Machine Learning, Cambridge University Press, 2014.

[68] A. I. Galushkin, Neural Network Theory, Berlin: Springer, 2008.

[69] Neuroph Java Neural Network Framework, "Neuroph," [Online]. Available: http://neuroph.sourceforge.net/tutorials/Perceptron.html. [Accessed Sep 2015].

[70] E. Weizstein, "Heavyside Step Function," Wolfram MathWorld, Sep 2015. [Online]. Available: http://mathworld.wolfram.com/HeavisideStepFunction.html. [Accessed Sep 2015].

[71] Rudjer Boskovic Institute, "Neural Networks," Rudjer Boskovic Institute, Sep 2015. [Online]. Available: http://dms.irb.hr/tutorial/tut_nnets_short.php. [Accessed Sep 2015].

[72] M. Hall, I. Witten and E. Frank, Data Mining; Practical Machine Learning Tools and Techniques, Morgan Kaufmann Publishers, 2011.

[73] W. Press, S. Tekolsky, W. Vetterling and B. Flannery, Numerical Recipes in C, Cambridge: Cambridge Univeristy Press, 1992.

[74] Mathworks, "Fit Data with a Neural Network R2015b," Mathworks , 2015. [Online]. Available: http://www.mathworks.com/help/nnet/gs/fit-data-with-a-neural-network.html#f9-33554. [Accessed Mar 2015].

[75] J. Heaton, "Sigmoid Activation Function," heatonresearch, Apr 2011. [Online]. Available: http://www.heatonresearch.com/w/index.php?title=Sigmoid_Activation_Function&action=info. [Accessed May 2015].

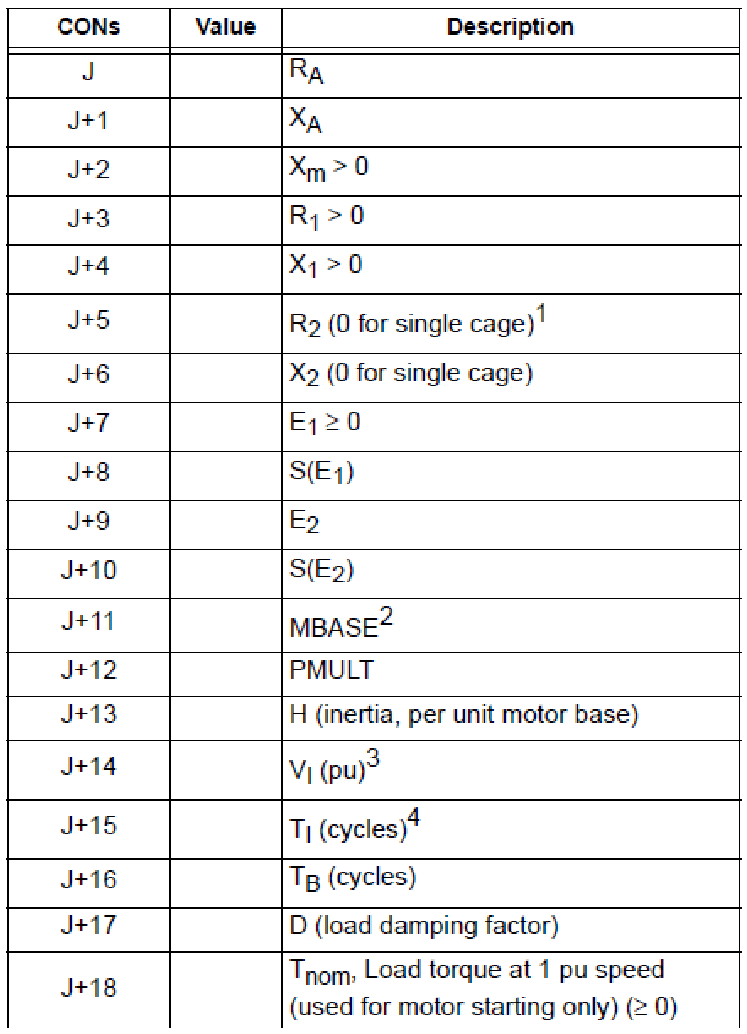
[76] C. M. Bishop, Neural Networks for Pattern Recognition, Oxford: Clarendon Press, 1995.

[77] C. F. Boos, G. R. Scolaro, M. C. V. Pereira and F. M. Azevedo, "Analysis of Pre-Processing Methods for Aritificail Neural Neywork Pattern Recognition of EEG Signals," in *World Congress on Medical Physics and Biomedical Engineering*, 2013.

[78] S. Guo and T. Overbye, "Parameter Estimation of a Complex Load Model Using Phasor Measurements," in *2012 IEEE Power and Energy Conference at Illinois*, Champaign, 2012.

Appendix

Appendix A. CIM5BL Induction Motor Parameters



CONs	Value	Description
J		R_A
J+1		X_A
J+2		$X_m > 0$
J+3		$R_1 > 0$
J+4		$X_1 > 0$
J+5		R_2 (0 for single cage)[1]
J+6		X_2 (0 for single cage)
J+7		$E_1 \geq 0$
J+8		$S(E_1)$
J+9		E_2
J+10		$S(E_2)$
J+11		MBASE[2]
J+12		PMULT
J+13		H (inertia, per unit motor base)
J+14		V_I (pu)[3]
J+15		T_I (cycles)[4]
J+16		T_B (cycles)
J+17		D (load damping factor)
J+18		T_{nom}, Load torque at 1 pu speed (used for motor starting only) (≥ 0)

Appendix B. Load Flow (Base Case) - N -1 Contingency Ranking

B.1 Contingency Analysis - EPCL Code

```
/* Steady State Contingency Analysis */
/*---------------------------------------------*/
/* Initializations and load base case    */
/*---------------------------------------------*/



$base = "ieee118.sav"                          /* Load-Flow Data under different seasonal
conditions */
@lbus = 59                                     /* Load Model Bus */
$scale = "1.0r"
$ivolt = "118bus_ivolt.txt"               /*Initial voltage after load conversion */

for @param = 1 to 3
        /* Retrieve Saved Case */
        @ret = getf($base)
        dispar[0].noprint = 0
        @cases = casepar[0].nbus /* 118 */
        @lines = casepar[0].nbrsec /* 186 - 9*/
        @trans = casepar[0].ntran  /* 9 */
        @ngens = casepar[0].ngen /* 54 */
        @lbusnum = next_index(4, @lbus, "1", -1)
        @bnum = bix(@lbus)
        /*Scale All Loads */
        @ret = scal(" ", "1", "0", "a", $scale, $scale, $scale, $scale)

        /* Convert Load Model */
        @ret = soln("1")
        @totldp = load[@lbusnum].p + volt[@bnum].vm*load[@lbusnum].ip +
((volt[@bnum].vm)*(volt[@bnum].vm))*load[@lbusnum].g
        @totldq = load[@lbusnum].q + volt[@bnum].vm*load[@lbusnum].iq +
((volt[@bnum].vm)*(volt[@bnum].vm))*load[@lbusnum].b

        if (@param=1)
                $output = "118buscrP.txt"
                load[@lbusnum].p = @totldp
                load[@lbusnum].ip = 0
                load[@lbusnum].g = 0
```

```
		load[@lbusnum].q = @totldq
		load[@lbusnum].iq = 0
		load[@lbusnum].b = 0
	elseif(@param=2)
		$output = "118buscrI.txt"
		load[@lbusnum].ip = @totldp/volt[@bnum].vm
		load[@lbusnum].p = 0
		load[@lbusnum].g = 0
		load[@lbusnum].iq = @totldq/volt[@bnum].vm
		load[@lbusnum].q = 0
		load[@lbusnum].b = 0
	elseif(@param=3)
		$output = "118buscrZ.txt"
		load[@lbusnum].g = @totldp/((volt[@bnum].vm)*(volt[@bnum].vm))
		load[@lbusnum].p = 0
		load[@lbusnum].ip = 0
		load[@lbusnum].b = @totldq/((volt[@bnum].vm)*(volt[@bnum].vm))
		load[@lbusnum].q = 0
		load[@lbusnum].iq = 0
	endif
	@ret = soln("1")

	/* Measure Initial Voltage at buses */
	@ret = setlog( $ivolt )
	for @bid = 0 to @cases-1
		@bvolt = volt[@bid].vm
		logprint($ivolt,  @bvolt, " ")
	next
	logprint($ivolt, "<")
	@ret = close ( $ivolt )
	@ret = setlog( $output )

	/* Line Contingencies */
	for @rec = 0 to @lines-1
		secdd[@rec].st = 0
		@ret = soln("1")

		/* Measure powerflow on lines */
		@ret = flowcalc(1)
		for @fid = 1 to @cases
			for @fid2 = 1 to @cases
				@from = @fid
```

```
                        @to = @fid2
                        $ck = "1"
                        for @i = 1 to 2
                                @k = flow_index(@from, @to, $ck)
                                if(@k < 0)
                                        quitfor
                                endif
                                @mw1 = flox[@k].p
                                @mw2 = flox[@k].q
                                logprint($output,  @mw1, " ")
                                $ck = "2"
                        next
                next
        next

        /* Measure Voltage at buses */
        for @bid = 0 to @cases-1
                @bvolt = volt[@bid].vm
                logprint($output,  @bvolt, " ")
        next

        logprint($output, "<")
        secdd[@rec].st = 1
next

/* T-line Contingencies */
for @rec = 0 to @trans-1
        tran[@rec].st = 0
        @ret = soln("1")

        /* Measure powerflow on lines */
        @ret = flowcalc(1)
        for @fid = 1 to @cases
                for @fid2 = 1 to @cases
                        @from = @fid
                        @to = @fid2
                        $ck = "1"
                        for @i = 1 to 2
                                @k = flow_index(@from, @to, $ck)
                                if(@k < 0)
                                        quitfor
                                endif
                        @mw1 = flox[@k].p
                        @mw2 = flox[@k].q
                        logprint($output,  @mw1, " ")
                        $ck = "2"
                    next
                next
            next

            /* Measure Voltage at buses */
            for @bid = 0 to @cases-1
                @bvolt = volt[@bid].vm
                logprint($output,  @bvolt, " ")
            next

            logprint($output, "<")
            tran[@rec].st = 1
        next

        /* Generator Contingencies */
        for @rec = 0 to @ngens-1
            gens[@rec].st = 0
            @ret = soln("1")

            /* Measure powerflow on lines */
            @ret = flowcalc(1)
            for @fid = 1 to @cases
                for @fid2 = 1 to @cases
                    @from = @fid
                    @to = @fid2
                    $ck = "1"
                    for @i = 1 to 2
                        @k = flow_index(@from, @to, $ck)
                        if(@k < 0)
                            quitfor
                        endif
                        @mw1 = flox[@k].p
                        @mw2 = flox[@k].q
                        logprint($output,  @mw1, " ")
                        $ck = "2"
                    next
                next
            next
```

```
                /* Measure Voltage at buses */
                for @bid = 0 to @cases-1
                        @bvolt = volt[@bid].vm
                        logprint($output,  @bvolt, " ")
                next

                logprint($output, "<")
                gens[@rec].st = 1
        next
        @ret = close ( $output )
next

end
```

B.2 Performance Indices and Ranking - MATLAB Code

```
%% Contingency Ranking using Static Load Model
%
%% Retrieve Data from PSLF File
clc
close all
clear all

contlist = 35; %Number of contingencies in ranked list
bnum = 118; %Number of buses
a1 = 1; %Range of wgts
b1 = 1;
a2 = 1;
b2 = 1;
idx1 = 1;
idx2 = 1;
filename1 = '118buscrP15.txt'; %uigetfile('*txt');     % gets the file name
of contingency case
filename1I = '118buscrI15.txt'; %uigetfile('*txt');
filename1Z = '118buscrZ15.txt'; %uigetfile('*txt');
filename2 = '118bus_linelimits.txt'; %uigetfile('*txt');      % gets the file
name of line coordinates and limits
filename3 = '118bus_linenotransgen.txt'; %uigetfile('*txt');      % gets the
file name of line and generator coordinates
filename4  = '118bus_ivolt (2).txt';  %Initial Voltages after Load Conversion
filename5 = '118bus_buslimits.txt'; %Bus PU Limits
fid = fopen(filename1, 'r');
data1 = dlmread(filename1);
data1(178:186,:) = [];
fclose(fid);
fidI = fopen(filename1I, 'r');
data1I = dlmread(filename1I);
data1I(178:186,:) = [];
```

```
fclose(fidI);
fidZ = fopen(filename1Z, 'r');
data1Z = dlmread(filename1Z);
data1Z(178:186,:) = [];
fclose(fidZ);
fid2 = fopen(filename2, 'r');
lind = dlmread(filename2);
fclose(fid2);
fid3 = fopen(filename3, 'r');
linetrans = dlmread(filename3);
fclose(fid3);
fid4 = fopen(filename4, 'r');
ivolt = dlmread(filename4);
fclose(fid4);
fid5 = fopen(filename5, 'r');
buslimits = dlmread(filename5);
fclose(fid5);
linelim = lind(:, 3);
lind = lind(:, 1:2);
numcont = size(data1, 1);
numlimits = size(data1, 2) - bnum;
r1 = a1 + (b1-a1).*rand(1,numlimits);
wgt = ones(numcont,1)*r1;
r2 = a2 + (b2-a2).*rand(1,bnum);
%alp = ones(numcont,1)*r2;
linemat = ones(numcont,1)*linelim';
delvlim = buslimits(:,1) - buslimits(:,2);
voltarray = [];

for param = 1:3
    if(param==1)
        ldata = data1(:, 1:numlimits);
        bdata = data1(:, numlimits+1:numlimits+bnum);
    elseif(param==2)
        ldata = data1I(:, 1:numlimits);
        bdata = data1I(:, numlimits+1:numlimits+bnum);
    else
        ldata = data1Z(:, 1:numlimits);
        bdata = data1Z(:, numlimits+1:numlimits+bnum);
    end

    %--Compute Amount of Bus Voltage Exceeding Limits--%
    delvup = [];
    delvlow = [];

    for i=1:numcont
        uplim = bdata(i,:)>(buslimits(:,1)');
        delvup = abs(ivolt(param, uplim) - bdata(i, uplim));
        lowlim = bdata(i,:)<(buslimits(:,2)');
        delvlow = abs(ivolt(param, lowlim) - bdata(i, lowlim));
        upfunc = (r2(uplim)./2.0).*((delvup./delvlim(uplim)').^2);
        lowfunc = (r2(lowlim)./2.0).*((delvlow./delvlim(lowlim)').^2);
        voltarray(i,1) = sum(upfunc) + sum(lowfunc);
    end
    %voltarray = zeros(numcont,1);  %Used to test without voltage violations

    %--Compute Amount of Redistribution on Each Line--%
    Arraycopy2 = abs(ldata);
    Arraycopy3 = Arraycopy2./linemat;
    Arraycopy4 = Arraycopy3;
    Arraycopy4(ind2sub(size(Arraycopy3), (find(Arraycopy3<1)))) = 0;
    Arraycopy4 = (Arraycopy4.^idx1).*wgt;

    %--Find Line Outages with Highest Sum of Flow/Limit Ratio--%
    Arraycopy4 = sum(Arraycopy4,2);
    Arraycopy5 = Arraycopy4 + voltarray;
    %Arraycopy5 = voltarray; %Only considering voltage violations %
    Arraycopy = Arraycopy5;
    Index = zeros(contlist,1);
    for j = 1:contlist
        [a, Index(j)] = max(Arraycopy);
        Arraycopy(Index(j)) = -inf;
    end
    maxValues(:, param) = Arraycopy5(Index);
    ContLines(:, 2*param-1:2*param) = linetrans(Index, :);
end
ContLines
maxValues
k=1;
for i = 1:contlist

if((ContLines(i,1)~=ContLines(i,3))||(ContLines(i,1)~=ContLines(i,5))||(ContL
ines(i,5)~=ContLines(i,3)))
        diffcont(k) = i;
        k = k + 1;

elseif((ContLines(i,2)~=ContLines(i,4))||(ContLines(i,2)~=ContLines(i,6))||(C
ontLines(i,6)~=ContLines(i,4)))
        diffcont(k) = i;
        k = k + 1;
    end
end
diffcont';
% sum(sum(data1-data1I))
% sum(sum(data1-data1Z))
```

Appendix C. Distribution Factors (Case 2) – N -1 Contingency Ranking

C.1 Simulated PMU Data – EPCL Code

```
/* Very simple program to run a dynamic simulation */
/* An inrun EPCL program, 118out.p, is used to output results to a text file */


/*---------------------------------------------*/
/* Initializations and load base case              */
/*---------------------------------------------*/

$base = "ieee118.sav"                                    /* Load-Flow Data under different seasonal
conditions */
$basedy = "ieee118_h59.dyd"                                        /* Dynamic Data
           */
$outchan = "118bus_p.chf"                      /* Output Channel                */
$initrep = "118busp_i.rep"                               /* Init Report                        */
$dynrep = "118busp_d.rep"                                /* Dynamic Report                 */
$inrunP = "inl59_specP.p"                      /* Inrun EPCL to output data    */
$inrunI = "inl59_specI.p"
$inrunZ = "inl59_specZ.p"
$lbus = "59"
@lbusnum = 59
$lbus2 = "56"
@lbusnum2 = 56
$lbus3 = "60"
$gbus = "59"
$lbus4 = "55"
$scale = "1.0r"

/*---------------------------------------------*/
/* Dynamic Simulation Variables              */
/*---------------------------------------------*/
@tinit = 1
@tstep1 = 5
@tstep2 = 6
@tend = 18                                                      /* Time to end simulation      */

for @i = 1 to 3
/* Retrieve Saved Case */
```

```
@ret = getf($base)
/*Scale All Loads */
@ret = scal(" ", "1", "0", "a", $scale, $scale)
dispar[0].noprint = 0

/* Retrieve Dynamic Data */
@ret = psds()                               /* MUST be included to setup modlib   */
@ret = rdyd($basedy,$dynrep,1,1,1)          /* All flags 1 */

if(@i=1)
        /* Convert Load Model */
        @ret = setmodpar(1, @lbusnum, 0, "1", 1, "blwscc","p1", 0.0)
        @ret = setmodpar(1, @lbusnum, 0, "1", 1, "blwscc","q1", 0.0)
        @ret = setmodpar(1, @lbusnum, 0, "1", 1, "blwscc","p2", 0.0)
        @ret = setmodpar(1, @lbusnum, 0, "1", 1, "blwscc","q2", 0.0)
        @ret = setmodpar(1, @lbusnum, 0, "1", 1, "blwscc","p3", 1.0)
        @ret = setmodpar(1, @lbusnum, 0, "1", 1, "blwscc","q3", 1.0)
        $inrun = $inrunP
elseif(@i=2)
        @ret = setmodpar(1, @lbusnum, 0, "1", 1, "blwscc","p1", 0.0)
        @ret = setmodpar(1, @lbusnum, 0, "1", 1, "blwscc","q1", 0.0)
        @ret = setmodpar(1, @lbusnum, 0, "1", 1, "blwscc","p2", 1.0)
        @ret = setmodpar(1, @lbusnum, 0, "1", 1, "blwscc","q2", 1.0)
        @ret = setmodpar(1, @lbusnum, 0, "1", 1, "blwscc","p3", 0.0)
        @ret = setmodpar(1, @lbusnum, 0, "1", 1, "blwscc","q3", 0.0)
        $inrun = $inrunI
elseif(@i=3)
        @ret = setmodpar(1, @lbusnum, 0, "1", 1, "blwscc","p1", 1.0)
        @ret = setmodpar(1, @lbusnum, 0, "1", 1, "blwscc","q1", 1.0)
        @ret = setmodpar(1, @lbusnum, 0, "1", 1, "blwscc","p2", 0.0)
        @ret = setmodpar(1, @lbusnum, 0, "1", 1, "blwscc","q2", 0.0)
        @ret = setmodpar(1, @lbusnum, 0, "1", 1, "blwscc","p3", 0.0)
        @ret = setmodpar(1, @lbusnum, 0, "1", 1, "blwscc","q3", 0.0)
        $inrun = $inrunZ
endif

/* Setting In-RUN -> MUST BE BEFORE INIT */
dypar[0].run_epcl = $inrun                  /* Name of the EPCL to In-Run*/

/* Initialize Dynamic Simulation */
@ret = init($outchan,$initrep,"1","0")              /* Output Channel: firstdyn.chf */
```

```
                                                /* "1" Fix Bad Data, "0" dont turn off unused
models */
/* Set Parameters for Run */                    /*Run to @tfault, remove caps, apply fault, run to
@clrfault, run to tend*/
dypar[0].tpause = @tinit
dypar[0].nplot = 4
dypar[0].nscreen = 999999                       /* Dont Print to Screen (makes simulation run
faster) */
@ret = run()

dypar[0].tpause = @tstep1
dypar[0].nplot = 4
dypar[0].nscreen = 999999                       /* Dont Print to Screen (makes simulation run
faster) */
@ret = run()

@return=scal("", $lbus3, $lbus3,"b","4.00r","1.00r","","")

dypar[0].tpause = @tstep2
dypar[0].nplot = 4
dypar[0].nscreen = 999999               /* Dont Print to Screen (makes simulation run faster */
@ret = run()

@return=scal("", $lbus3, $lbus3,"b","0.25r","1.00r","","")

dypar[0].tpause = @tend
dypar[0].nplot = 4
dypar[0].nscreen = 999999                       /* Dont Print to Screen (makes simulation run
faster */
@ret = run()

@ret = dsst()                                   /* Stop dynamic simulation */
next

end
/*--------------------------------------*/
/* In-RUN EPCL                                  */
/*--------------------------------------*/
@onetime = -0.008               /* Time before initialization */
$output = "l59_ConstP.txt"                              /* Output file*/
@buses = 118
@lbus = 59
```

```
@ld = 48
dim #current[6]
#current[0] = 54
#current[1] = 55
#current[2] = 56
#current[3] = 60
#current[4] = 61
#current[5] = 63
/*--------------------------------------------*/
/*Monitor the Supervisory Boundary (in-run EPCL)*/
/*--------------------------------------------*/
@ret = setlog($output)
logprint($output, "<",dypar[0].time, ">")                /* print the time step */

/* Measure all line currents and bus voltages */
@ret = flowcalc(1)
@lind = @lbus-1
@vmag = volt[@lind].vm
logprint($output,  @vmag, " ")
@vang = volt[@lind].va
logprint($output,  @vang, " ")
@vfreq = netw[@lind].f
logprint($output,  @vfreq, " ")
@mamps = 0
@mloss = 0
for @fid = 0 to 5
        @from = #current[@fid]
        @to = @lbus
        $ck = "1"
        for @i = 1 to 2
                @k = flow_index(@from, @to, $ck)
                if(@k < 0)
                        quitfor
                endif

                /* Use this section to read current to load */
                /* @mamps = @mamps + flox[@k].amps */

                /* Use this section to read real power to load */
                @mamps = @mamps + flox[@k].p
                @mloss = @mloss + flox[@k].mwloss
```

```
                        /* Use this section to record current along lines */
                        /* @mamps = flox[@k].amps */
                        /* logprint($output,  @mamps, " ")  */
                        $ck = "2"
            next
next
/* Add gen output at bus 59 */
@mamps = @mamps + @mloss + gens[24].pgen

/* Use this section to record current/real power to load */
logprint($output,  @mamps, " ")

/* This section records assigned power to load */
@lp = load[@ld].p
@lq = load[@ld].q
@lip = load[@ld].ip
@liq = load[@ld].iq
@lg = load[@ld].g
@lb = load[@ld].b
@lmp = load[@ld].pm
@lmq = load[@ld].qm
@ltotp = @lp + @lip + @lg + @lmp
@ltotq = @lq + @liq + @lb + @lmq
logprint($output, @ltotp, " ")

@ret = close ( $output )
```

C.2 Distribution Factors, Performance Indices and Ranking - MATLAB Code

```
%% Parameter Estimation of Static Load Model
%
%% Retrieve Data from PSLF File
clc
close all
clear all

buses = 118;
a1 = 1; %Range of wgts
b1 = 20;
idx1 = 3;
zinj = [5 9 30 37 38 63 64 68 71 81]'; % Insert zero inj shift factors for
buses with no load/no gen %
isl = [8 9; 9 10; 12 117; 71 73; 85 86; 86 87; 110 111; 110 112; 68 116;];
filename1 = uigetfile('*txt');      % gets the file name of contingency case P
filename4 = uigetfile('*txt');      % gets the file name of contingency case I
filename5 = uigetfile('*txt');      % gets the file name of contingency case Z
```

```
filename2 = '118bus_linelimits.txt'; %uigetfile('*txt');      % gets the file
name of line coordinates and limits
filename3 = '118bus_linenotrans.txt'; %gets file with only line coordinates
(no transformers)
fid = fopen(filename1, 'r');
data1 = dlmread(filename1);
fclose(fid);
fid2 = fopen(filename2, 'r');
lind = dlmread(filename2);
fclose(fid2);
fid3 = fopen(filename3, 'r');
notrans = dlmread(filename3);
fclose(fid3);
linelim = lind(:, 3);
lind = lind(:, 1:2);
varnum1 = size(data1, 2);
varnum = 4;
n = size(data1, 1);
cut = mod(n, varnum);
n = n - cut;
numlines = varnum1 - buses - 1;
linelim = linelim*ones(1,numlines);
r = a1 + (b1-a1).*rand(1,numlines);
wgt = ones(numlines,1)*r;

for param=1:3
    if param==2
        fid4 = fopen(filename4, 'r');
        data1 = dlmread(filename4);
        fclose(fid4);
    elseif param==3
        fid5 = fopen(filename5, 'r');
        data1 = dlmread(filename5);
        fclose(fid5);
    end
    j = 1;
    dpinj = zeros((n-varnum)/varnum,buses);
    dqinj = zeros((n-varnum)/varnum,buses);
    dplines = zeros((n-varnum)/varnum, varnum1-buses-1);
    time1 = zeros((n-varnum)/varnum,1);
    for i=1:varnum:(n-varnum)
        time1(j,1) = data1(i+varnum, 1);
        dpinj(j,1:buses) = data1((i+varnum), (varnum1-buses+1:varnum1)) -
data1(i, (varnum1-buses+1:varnum1));
        dplines(j,1:numlines) = data1((i+varnum), (2:varnum1-buses)) -
data1(i, (2:varnum1-buses));
        j = j + 1;
    end

    rnk = rank(dpinj)
    zerobuschg = find(sum(dpinj,1)==0);
    numnnz = nnz(zerobuschg)
    N1 = j - 1;       %size of data
    H = dpinj;
    H(:, zinj) = [];
    b = dplines(:,1);
    G = H'*H;
    x = G\(H'*b);

    %--Compute all Injection Shift Factors and Line Outage Dist. Factors --%
    xlines = zeros(size(x,1),numlines);
    lodf = zeros(numlines, numlines);
    for i = 1:numlines
        xlines(:, i) = G\(H'*dplines(:,i));
    end

    for var = zinj-1
        xlines = insertrows(xlines, zeros(1, numlines), var);
    end

    for i = 1:numlines
        for j = 1:numlines
            lodf(j,i) = (xlines(lind(j,1),i) - xlines(lind(j,2),i))/(1 -
(xlines(lind(j,1),j) - xlines(lind(j,2),j)));
        end
    end
    for i=1:size(isl, 1)
        j1 = strmatch(isl(i,:), lind);
        lodf(j1, :) = 0;
        j2 = strmatch([isl(i,2), isl(i,1)], lind);
        lodf(j2, :) = 0;
    end

    %--Compute Amount of Redistribution on Each Line--%
    ssflow = (data1(240*varnum, (buses+2):varnum1))';
    ssflow = ssflow*ones(1,size(lodf,2));
    Arraycopy1 = lodf.*ssflow;
    Arraycopy2 = Arraycopy1 + ssflow';
    Arraycopy2(1:numlines+1:numlines*numlines) = 0;
    Arraycopy3 = abs(Arraycopy2)./linelim';
    Arraycopy4 = Arraycopy3;
    Arraycopy4(ind2sub(size(Arraycopy3), (find(Arraycopy3<1)))) = 0;
    Arraycopy4 = (Arraycopy4.^idx1).*wgt;

    %--Find Lines that Exceed Limits--%
    Arraycopy4 = sum(Arraycopy4,2);
    Arraycopy = Arraycopy4;
    for j = 1:40
        [a, Index(j,1)] = max(Arraycopy);
        Arraycopy(Index(j,1)) = -inf;
    end
    maxValues(:, param) = Arraycopy4(Index);
    ContLines(:, param*2-1:param*2) = lind(Index, :);
    data1 = [];
end
ContLines
maxValues
```

Appendix D. Voltage Sensitivities (Case 3) – N -1 Contingency Ranking

D.1 Simulated PMU Data - EPCL Code

```
/* A program to run a dynamic simulation to calculate voltage sensitivities*/
/* An inrun EPCL program is used to output results to a text file */

/*---------------------------------------------*/
/* Initializations and load base case              */
/*---------------------------------------------*/
$base = "ieee118.sav"                             /* Load-Flow Data  */
@lbus = 59                                        /* Load Model Bus */
$scale = "1.00r"
$randtxt = "randscal-3.txt"
@totscl = 500-1
dim #randscal[500]
dim #pinjarray[118]
dim #qinjarray[118]

/* Read Load Scaling Numbers from Textfile */
@err = setinput($randtxt)
for @i = 0 to @totscl
          @ret = inline($randtxt, $rand)
          #randscal[@i] = atof($rand)
next
@ret = close($randtxt)

/*---------------------------------------------*/
/* Simulation Variables                      */
/*---------------------------------------------*/
for @param = 1 to 1              /* Multiple load models */
          /* Retrieve Saved Case */
          @ret = getf($base)
          dispar[0].noprint = 0
          @cases = casepar[0].nbus /* 118 */
          @lines = casepar[0].nbrsec /* 186 - 9*/
          @trans = casepar[0].ntran   /* 9 */
          @lbusnum = next_index(4, @lbus, "1", -1)
          @bnum = bix(@lbus)

          /*Scale All Loads and Gens */
```

```
        /*
        @ret = soln("1")
        @ret = scal(" ", "1", "0", "a", $scale, $scale, $scale, $scale)
        dispar[0].noprint = 0
        */

        /* Convert Load Model */
        @ret = soln("1")
        @totldp = load[@lbusnum].p + volt[@bnum].vm*load[@lbusnum].ip +
((volt[@bnum].vm)*(volt[@bnum].vm))*load[@lbusnum].g
        @totldq = load[@lbusnum].q + volt[@bnum].vm*load[@lbusnum].iq +
((volt[@bnum].vm)*(volt[@bnum].vm))*load[@lbusnum].b

        if (@param=1)
                $output = "118busP.txt"
                load[@lbusnum].p = @totldp
                load[@lbusnum].ip = 0
                load[@lbusnum].g = 0
                load[@lbusnum].q = @totldq
                load[@lbusnum].iq = 0
                load[@lbusnum].b = 0
        elseif(@param=2)
                $output = "118busI.txt"
                load[@lbusnum].ip = @totldp/volt[@bnum].vm
                load[@lbusnum].p = 0
                load[@lbusnum].g = 0
                load[@lbusnum].iq = @totldq/volt[@bnum].vm
                load[@lbusnum].q = 0
                load[@lbusnum].b = 0
        elseif(@param=3)
                $output = "118busZ.txt"
                load[@lbusnum].g = @totldp/((volt[@bnum].vm)*(volt[@bnum].vm))
                load[@lbusnum].p = 0
                load[@lbusnum].ip = 0
                load[@lbusnum].b = @totldq/((volt[@bnum].vm)*(volt[@bnum].vm))
                load[@lbusnum].q = 0
                load[@lbusnum].iq = 0
        endif
        @ret = soln("1")
        @ret = flowcalc(1)
        @ret = setlog($output)
        logprint($output, "<","-1", ">")                /* print the time step */
        /* Print Voltages to text file */
        for @fid2 = 0 to 117
                @vmag = volt[@fid2].vm
                @vang = volt[@fid2].va
                @vfreq = 0
                logprint($output, @vmag, " ", @vang, " ", @vfreq, " ")
        next

        /* Measure powerflow on lines */
        for @fid = 1 to 118
                #pinjarray[@fid-1] = 0
                #qinjarray[@fid-1] = 0
                for @fid2 = 1 to 118
                        @from = @fid
                        @to = @fid2
                        $ck = "1"
                        for @i = 1 to 2
                                @k = flow_index(@from, @to, $ck)
                                if(@k < 0)
                                        quitfor
                                endif
                                #pinjarray[@fid-1] = #pinjarray[@fid-1] + flox[@k].p /* +
    flox[@k].mwloss */
                                #qinjarray[@fid-1] = #qinjarray[@fid-1] + flox[@k].q /* +
    flox[@k].mvloss */
                                $ck = "2"
                        next
                next
                logprint($output, #pinjarray[@fid-1], " ", #qinjarray[@fid-1], " ")
        next

        /* Change injection at generator buses by scaling all loads*/
        @scaledn = 1.0
        @casest = 0
        @scalsum = 0
        for @ldscl = @casest to @totscl
                @scalsum = @scalsum + #randscal[@ldscl]
                @scaleup = @scaledn*(@scalsum + 1.00)
                @scaledn = 1/(@scalsum + 1.00)
                logbuf($scaleinc, @scaleup)
                $scaleinc = $scaleinc + "r"
                @ret = scal(" ", "1", "0", "a", $scaleinc, $scaleinc, $scaleinc, $scaleinc)
                dispar[0].noprint = 1  /* 0 to print log, 1 to skip print */
                @ret = soln("1")
                @ret = flowcalc(1)
                logprint($output, "<",@ldscl, ">")                /* print the time step */

                /* Print Voltages to text file */
                for @fid2 = 0 to 117
                        @vmag = volt[@fid2].vm
                        @vang = volt[@fid2].va
                        @vfreq = 0
                        logprint($output, @vmag, " ", @vang, " ", @vfreq, " ")
                next

                /* Measure powerflow on lines */
                for @fid = 1 to 118
                        #pinjarray[@fid-1] = 0
                        #qinjarray[@fid-1] = 0
                        for @fid2 = 1 to 118
                                @from = @fid
                                @to = @fid2
                                $ck = "1"
                                for @i = 1 to 2
                                    @k = flow_index(@from, @to, $ck)
                                    if(@k < 0)
                                        quitfor
                                    endif
                                    #pinjarray[@fid-1] = #pinjarray[@fid-1] + flox[@k].p /* +
    flox[@k].mwloss */
                                    #qinjarray[@fid-1] = #qinjarray[@fid-1] + flox[@k].q /* +
    flox[@k].mvloss */
                                    $ck = "2"
                                next
                        next
                        logprint($output, #pinjarray[@fid-1], " ", #qinjarray[@fid-1], " ")
                next
                /*logterm("<", "from", @from, ">", "totscl", @totscl)*/
        next
        @ret = close ( $output )
    next
    end
```

```
/*-------------------------------------*/
/* In-RUN EPCL                                    */
/*-------------------------------------*/
/* Constants */
@onetime = -0.008                /* Time before initialization */
$output = "118SensOutP.txt" /* Output file*/
dim #pinjarray[118]
dim #qinjarray[118]

/* Run at t=0 and every second */
/* @tprint = 0.250
if (mod(round(dypar[0].time, 3), @tprint) < 1e-3) */

/* Run at t=0 and 30x a second */
/* @tprint = 0.03333
if (mod(round(dypar[0].time, 5), @tprint) < 1e-3) */
@tprint = 1/30
@time0 = round(mod(dypar[0].time/@tprint, 1), 2)
if ((@time0 < 0.05) or (@time0 > 0.90))

/*---------------------------------------------*/
/*Monitor the Supervisory Boundary (in-run EPCL)*/
/*---------------------------------------------*/
@ret = setlog($output)
logprint($output, "<",dypar[0].time, ">")                /* print the time step */
@ret = flowcalc(1)

/* Print Voltages to text file */
for @fid2 = 0 to 117
        @vmag = volt[@fid2].vm
        @vang = volt[@fid2].va
        @vfreq = 0      /* netw[@fid2].f */
        logprint($output, @vmag, " ", @vang, " ", @vfreq, " ")
next

/* Measure powerflow on lines */
for @fid = 1 to 118
        #pinjarray[@fid-1] = 0
        #qinjarray[@fid-1] = 0
        for @fid2 = 1 to 118
                @from = @fid
                @to = @fid2
                $ck = "1"
                for @i = 1 to 2
                        @k = flow_index(@from, @to, $ck)
                        if(@k < 0)
                                quitfor
                        endif
                        #pinjarray[@fid-1] = #pinjarray[@fid-1] + flox[@k].p /* + flox[@k].mwloss
*/
                        #qinjarray[@fid-1] = #qinjarray[@fid-1] + flox[@k].q /* + flox[@k].mvloss
*/
                        $ck = "2"
                next
        next
        logprint($output, #pinjarray[@fid-1], " ", #qinjarray[@fid-1], " ")
next

@ret = close ( $output )
endif
```

D.2 Voltage Sensitivities, Performance Indices and Ranking - MATLAB Code

```
clc
close all
clear all

filename0 = '118precontP10.txt'; %uigetfile('*txt');   %gets file name of pre
contingency measurements (generation and line flow) %
filenamelim = '118bus_linegennotrans.txt'; %uigetfile('*txt');      % gets
the file name of line/gen coordinates and limits
filename1 = uigetfile('*txt');     % gets the file name of simulation data
'118SensOutP_scale500-2.txt' %
filename2 = uigetfile('*txt');   %Add two or more data sets or different load
models '118SensOutP_scale500-3.txt' %
%filename3 = uigetfile('*txt');

nbus = 118;
ngen = 54;
nlines = 177; %354 Measurements for both directions of each line%
swing = 69;
a2 = 1;
b2 = 1;
contlist = 35;

for f=1:1
    if(f==1)
        filename = filename1;
        col = 'b';
    elseif(f==2)
```

```
            filename = filename2;
            col = 'g';
        else
            filename = filename3;
            col = 'r';
        end
        % Read Data %
        fid = fopen(filename);
        data1 = dlmread(filename);
        fclose(fid);
        %data1(1:3, :) = [];

        % Add two or more data sets together %
        fid02 = fopen(filename2);
        data02 = dlmread(filename2);
        fclose(fid02);
%       fid03 = fopen(filename3);
%       data03 = dlmread(filename3);
%       fclose(fid03);
        data1 = [data1; data02]; %data03];

        nnew = size(data1, 1);
        time1 = data1(:, 1);
        vmag = data1(:, 2:3:(nbus*3+1))';
        vang = data1(:, 3:3:(nbus*3+1))';
        vang2 = wrapToPi(vang - ones(size(vang))*diag(vang(swing,:)));
        vfreq = data1(:, 4:3:(nbus*3+1))';
        pinj = data1(:, (nbus*3+2):2:(nbus*5))';
        qinj = data1(:, (nbus*3+3):2:(nbus*5+1))';
%       pinj(abs(pinj)<1e-4) = 0; %remove zero injection buses
%       qinj(abs(qinj)<1e-4) = 0;

        % Calculate Sensitivity Matrix %
        del_vmag = (vmag(:, 2:nnew) - vmag(:, 1:nnew-1)); %./vmag(:, 1:nnew-1);
%!!!!LOOK INTO THIS (del_vmag/vmag0)
        del_vang = wrapToPi(vang2(:, 2:nnew) - vang2(:, 1:nnew-1));
        del_pinj = pinj(:, 2:nnew) - pinj(:, 1:nnew-1);
        del_qinj = qinj(:, 2:nnew) - qinj(:, 1:nnew-1);
        x = [del_pinj; del_qinj];
        b = [del_vang; del_vmag];

        % Test One - vmag and vang using Pinj and Qinj %
        sens = b*(x')/(x*(x'));

        % Read Pre Contingency Measurements %
        fid2 = fopen(filename0);
        data2 = dlmread(filename0);
        fclose(fid2);
        ndata2 = size(data2, 1);
        gen0 = data2(1:ngen, :);
        line0 = data2(ngen+1:ndata2, :);
        conts = [gen0(:,1) zeros(ngen,1); line0(:,1:2)];
        newvolt = zeros(nbus*2, ngen+nlines);
```

```
newvoltmag = zeros(nbus,ngen+nlines);

% Compute Generator Post Contingency Voltages %
for g = 1:ngen
    test = zeros(236,1);
    test(gen0(g,1)) = -gen0(g,2);
    test(gen0(g,1)+nbus) = -gen0(g,3);
    bnew = sens*test;
    del_newvolt = bnew;
    newvolt(:,g) = del_newvolt + [vang2(:,61); vmag(:,61)];
    newvolt(1:nbus,g) = wrapToPi(newvolt(1:nbus,g));
    newvoltmag(:,g) = newvolt(nbus+1:nbus*2,g);
end
%newvoltmag(:,1:ngen) = 1;   % Remove Gen Outage Estimates to test CR %

% Compute Line Post Contingency Voltages %
for l = 1:nlines
    test = zeros(236,1);
    test(line0(l,1)) = line0(l,3);
    test(line0(l,2)) = line0(l,5);
    test(line0(l,1)+ nbus) = line0(l,4);
    test(line0(l,2)+ nbus) = line0(l,6);
    bnew = sens*test;
    del_newvolt = bnew;
    newvolt(:,ngen+l) = del_newvolt + [vang2(:,61); vmag(:,61)];
    newvolt(1:nbus,ngen+l) = wrapToPi(newvolt(1:nbus,ngen+l));
    newvoltmag(:,ngen+l) = newvolt(nbus+1:nbus*2,ngen+l);
end

% Create Ranking By Checking Limits of Estimated Voltages %
fid3 = fopen(filenamelim);
data3 = dlmread(filenamelim);
fclose(fid3);
buslimits = data3(1:nbus, 1:2);
coords = zeros(nbus+nlines, 2);
coords(1:nbus,1) = linspace(1,nbus,nbus);
coords(nbus+1:nbus+nlines, :) = data3(nbus+1:nbus+nlines,1:2);
r2 = a2 + (b2-a2).*rand(1,nbus);
%alp = ones(ngen+nlines,1)*r2;
delvlim = buslimits(:,1) - buslimits(:,2);
delvup = [];
delvlow = [];
voltarray = [];

for c=1:ngen+nlines
    uplim = newvoltmag(:,c)>(buslimits(:,1));
    delvup = abs(vmag(uplim, 3) - newvoltmag(uplim, c));
    lowlim = newvoltmag(:,c)<(buslimits(:,2));
    delvlow = abs(vmag(lowlim, 3) - newvoltmag(lowlim, c));
    upfunc = (r2(uplim)'./2.0).*((delvup./delvlim(uplim)).^2);
    lowfunc = (r2(lowlim)'./2.0).*((delvlow./delvlim(lowlim)).^2);
    voltarray(c,1) = sum(upfunc) + sum(lowfunc);
end
```

```
    Arraycopy = voltarray;
    Index = zeros(contlist,1);
    for j = 1:contlist
        [a, Index(j)] = max(Arraycopy);
        Arraycopy(Index(j)) = -inf;
    end
    maxValues(:, f) = voltarray(Index);
    ContLines(:, 2*f-1:2*f) = conts(Index,:); %coords(Index,:);
end
ContLines
maxValues
```

Appendix E. PMU Algorithm – MATLAB Code

```
%%% PMU Algorithm applied to PSS/E %%%
%%% 04/07/2014 %%%

clc
close all
clear all

ltime = 1.09;
pad = 6;

% Parameters %
% filenamep = 'machparam_rate_h.txt';
% fidp = fopen(filenamep);
% param = dlmread(filenamep);
% fclose(fidp);
% Test
fidp = 0;
param = [10.0, 1.5, 0.008, 0.05, 1.0]; %[98.2, 4.44, 0.01584, 0.197, 1.0];

% Size of Data Set %
if(fidp)
    n = size(param,1);
elseif(size(param,1)==1)
    n = 1;
else
    prompt = 'Enter number of data sets:';
    n = input(prompt);
end

%Read PSS/E Data
deletep = 0;
d = 1;
m = n;
for i=1:n
    %filename = uigetfile('*dat');     % gets the file name of contingency
case
    if round(param(i,1))==param(i,1)
        LMp = [num2str(param(i,1)) '.0'];
    else
        LMp = num2str(param(i,1));
    end
    if round(param(i,5))==param(i,5)
        Hp = [num2str(param(i,5)) '.0'];
    else
        Hp = num2str(param(i,5));
    end
    %filename = ['118bus_load11_lines11-4_' num2str(param(i,1)) '_'
num2str(param(i,2)) '_' num2str(param(i,3)) '_' num2str(param(i,4)) '_'
num2str(param(i,5)) '.dat'];
```

```
    filename = ['118bus_load11_lines11-4_' LMp '_' num2str(param(i,2)) '_'
num2str(param(i,3)) '_' num2str(param(i,4)) '_' Hp '.dat'];
    if exist(filename, 'file')
        fid = fopen(filename);
        while ~feof(fid)
            data = textscan(fid, '%f');
            volt(i, :) = data{1,1};
            fgets(fid);
            data = textscan(fid, '%f');
            theta(i, :) = data{1,1};
            fgets(fid);
            data = textscan(fid, '%f');
            voltref(i, :) = data{1,1};
            fgets(fid);
            data = textscan(fid, '%f');
            thetaref(i, :) = data{1,1};
        end
        fclose(fid);
    elseif(i==n)
        volt(i,:) = 0;
        theta(i,:) = 0;
        voltref(i,:) = 0;
        thetaref(i,:) = 0;
        m = m - 1;
        deletep(d) = i;
        d = d + 1;
    else
        m = m - 1;
        deletep(d) = i;
        d = d + 1;
    end

end
if (deletep)
    param(deletep, :) = [];
    volt(deletep, :) = [];
    theta(deletep, :) = [];
    voltref(deletep, :) = [];
    thetaref(deletep, :) = [];
end
ldata = size(volt, 2);
% %Padding PSS/E Data
% p = 1;
% for i=1:ldata
%     voltp(:,p) = volt(:,i);
%     thetap(:,p) = theta(:,i);
%     voltp(:,(p+1):(p+5)) = zeros(m,pad-1);
%     thetap(:,(p+1):(p+5)) = zeros(m,pad-1);
%     p = p + pad;
% end
%No Padding - 720Hz sample
voltp = volt;
thetap = theta;
```

```
%Plot PSS/E Meas.
t = 0:1/720:((size(voltp,2)/720)-(1/720));
figure
subplot(2,1,1)
plot(t,voltp);
xlabel('Time (s)');
ylabel('Voltage Mag (V)');
title('PSS/E Voltage Magnitude');
subplot(2,1,2)
plot(t,thetap);
xlabel('Time (s)');
ylabel('Voltage Angle (Theta)');
title('PSS/E Voltage Angle');

% Filter (LPF) and Downsample Data
fPSSE = 720;    %Sampling rate
n = 12; %Num. of samples in 1 cycle
fN = n/2*60;    %Nyquist Freq
N = 36; %Num. of samples in 3 cycles
Fpass = fN/fPSSE;  % Passband Frequency
Fstop = 0.51;    % Stopband Frequency
Apass = 1;       % Passband Ripple (dB)
Astop = 60;      % Stopband Attenuation (dB)
%h = fdesign.lowpass('fp,fst,ap,ast', Fpass, Fstop, Apass, Astop);
h = fdesign.lowpass('n,fp,fst',N, Fpass, Fstop);
% h = fdesign.decimator(16, 'Lowpass', 'fp,fst,ap,ast', Fpass,Fstop,...
% Apass,Astop); % Downsample LPF
%Hd = design(h, 'equiripple', 'MinOrder', 'any', 'StopbandShape', 'flat');
Hd = design(h, 'equiripple');
%Hd2 = design(h, 'butter', 'MatchExactly', 'passband');
[angf1, angf2] = butter(3, 0.50);
vin = voltp;
vm = filter(Hd, vin);
vm = downsample(vm, 6/Fpass);
vain = thetap;
va = filter(angf1, angf2, vain);
%va = filter(Hd, vain);
va = downsample(va, 6/Fpass);
[vr, vi] = pol2cart(va.*(pi/180), vm);

% Quantize and convert to polar coords
% Vr = quant(vr, 4.0e-5);   %Quantize voltage - 20 V
% Vi = quant(vi, 4.0e-5);
Vr = vr;   %PSS/E voltage is per unit - uncomment if not quantizing
Vi = vi;
V = Vr + 1i*Vi;
Vmag = abs(V);
Vang = angle(V);
Vang = wrapTo2Pi(Vang).*180/pi;

t = 0:(1/60):(1+(size(Vmag,2)-61)/60);
figure
suptitle('PMU Data converted from PSS/E data');
subplot(2,1,1)
```

```
plot(t,Vmag, '--');
xlabel('Time (s)')
ylabel('Va mag (V)')
%axis([0 1.4 1.20e5 1.4e5]);
subplot(2,1,2)
plot(t,Vang, '--');
xlabel('Time (s)')
ylabel('Va angle (theta)')
axis([0 1.4 -5.0 (max(thetap, [], 2)+10)]);
```

Appendix F. Initial Complex Load Parameter Estimation

F.1 Parameter Vector Generator – MATLAB

```
%Create table of random number vectors for P and LM mach
clc
clear all
close all

lm = 10:10:100;
rs = 0.024:-0.006:0.012;
xs = 0.2:-0.075:0.05;
xm = 3.5:1.5:6.5;
rr = 0.02:-0.004:0.008;
xr = 0.2:-0.075:0.05;
r2 = 0.005:0.005:0.015;
x2 = 0.1:0.05:0.2;
H = 0.4:0.4:1.2;
cparam1 = [1; 0.06; 1.2; 0.6; 2.3; 0]; %H=1.0;
cparam2 = [0; 0; 0; 1; 45];
combo = combvec(lm, rs, xs, xm, rr, xr, r2, x2, H);
% combo2 = combvec(lm2, rs2, xs2, xm2, rr2, xr2);
% combo = [combo1, combo2];
nomach = [0.0; 0.0; 0.0; 0.0; 0.00; 0.00; 0.0; 0.0; 0.0];
filename = 'machparam9.txt';
fid = fopen(filename, 'w');
fprintf(fid, '%g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g\n',
nomach(1:8,:), cparam1, nomach(9,:),cparam2);
for i=1:size(combo,2)
    fprintf(fid, '%g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g
%g\n', combo(1:8,i), cparam1, combo(9,i), cparam2);
end
fclose(fid);
```

F.2 Simulated Measurement Data – Python

```
# Collect Fault Data for Complex Load  - Ind Motor and ZIP
# Xfmr at load bus - modelled at LV
# Hema Retty
# Created on 02/24/2015
# Modified on 05/06/2015

from __future__ import with_statement
from contextlib import contextmanager
from xlsxwriter.workbook import Workbook
from copy import deepcopy
import os,sys
import numpy as np
import itertools
##import matplotlib.pyplot as plt

@contextmanager
def silence(file_object=None):
    """
    Discard stdout (i.e. write to null device) or
    optionally write to given file-like object.
    """
    if file_object is None:
        file_object = open(os.devnull, 'w')

    old_stdout = sys.stdout
    try:
        sys.stdout = file_object
        yield
    finally:
        sys.stdout = old_stdout

PSSE_LOCATION = r"C:\Program Files (x86)\PTI\PSSE32\PSSBIN"
sys.path.append(PSSE_LOCATION)
os.environ['PATH'] = os.environ['PATH'] + ';' +  PSSE_LOCATION

import psspy
psspy.throwPsseExceptions=True
import redirect
redirect.psse2py()
import dyntools

psspy.psseinit(2000)

#--------------------------------
#Read parameters from file - [%LM,%R1,%X1,%Xm, %R2, %X2]
f = open("machparam9.txt")
param3 = f.readlines()
param3 = np.loadtxt(param3)
#param3 = [[23, 42, 15, 20], [0, 0, 0, 100]]  #Test Case
param3 = np.array(param3)
f.close()
pend = len(param3)
```

```
#Start Simulation
#with silence():
nLM = 50

for pid in range(5, pend):
    if (param3[pid][0]>0)&(param3[pid][0]<100):
        nLDs = nLM + 1
    elif param3[pid][0]==0:
        nLDs = 1
    elif param3[pid][0]==100:
        nLDs = nLM
    LID = []
    with silence():
        # PSS/E Saved case
        CASE = r"C:\Users\Hema\OneDrive\Documents\Research\Neural
Network\ieee118bus_PSSEcorrected_rq_kv.sav"
        psspy.case(CASE)
        psspy.fnsl()

        #Variables
        loadbus = 11 #83 #59 #Change Load Bus
        tbus = 200  #Transformer bus at load
        refbus = 93 #Reference Bus
        bfault = [11, 4] #[84, 83] #[59, 60] #Fault Branch
        cont = 20 #Contingency-branch number
        param = [[100.0, 0.0, 0.0, 100.0], [0.0, 0.0, 0.0, 0.0], [100.0, 0.0, 100.0, 0.0], [0.0, 100.0, 0.0,
100.0]] #[real:%I %Y imag:%I %Y] Load Parameters:[I Y],[P P],[I I],[Y Y]

        #Add Transformer at Load Bus
        ierr = psspy.bus_data_2(i=tbus,name='tbus')
        ierr,psspy.realaro = psspy.two_winding_data_3(loadbus, tbus, '1', [], [0.0, 0.037])

        #Split Load into two IDs
        ierr, cmpval = psspy.loddt2(loadbus, 'BL', 'MVA', 'ACT')
        ierr = psspy.load_data_3(loadbus, 'BL',intgar1=0)
        Pinit = cmpval.real #Xfmr ratio
        Qinit = cmpval.imag
        Pind = Pinit*param3[pid][0]/100
        Qind = Qinit*param3[pid][0]/100
        Pzip = Pinit - Pind
        Qzip = Qinit - Qind
        tempZIP = np.array([param[0][1]/100, param[0][0]/100, (100-param[0][0]-
param[0][1])/100, param[0][3]/100, param[0][2]/100, (100-param[0][2]-param[0][3])/100])
#[real:Z I P, imag: Z I P]

        #tempmach = [param3[pid][1], param3[pid][2], param3[pid][3], param3[pid][4]]
        tempmach = list(param3[pid][1:20]) #1:19
        if param3[pid][0]>0:
            for iLM in range(nLM):
                LID.append('%d' % (iLM+1))
                ierrl1 = psspy.load_data_3(tbus, LID[iLM], [] , [Pind/nLM, Qind/nLM, 0, 0, 0, 0])
        if param3[pid][0]<100:
            ierrl2 = psspy.load_data_3(tbus, 'BL', [] , [Pzip, Qzip, 0, 0, 0, 0])
        psspy.fnsl()

        #Create lists of branches and 2 winding transformers and load buses
        ierr3, totbrn = psspy.abrncount(-1, 1, 1, 1)
        ierr4, brnlist = psspy.abrnint(-1, 1, 1, 1, 1, ['FROMNUMBER', 'TONUMBER'])
        #ierr, tot2wind = psspy.atrncount(-1, 1, 1, 1, 1) #Count 2 winding xmfrs
        #ierr1, twowindlist = psspy.atrnint(-1, 1, 1, 1, 1, ['FROMNUMBER', 'TONUMBER']) #Array of
2 winding xmfrs
        ierr, totgen = psspy.agenbuscount(-1, 1)
        ierr2, genlist = psspy.agenbusint(-1, 1, 'NUMBER')
        ierr5, loadlist = psspy.alodbusint(-1, 1, 'NUMBER')
        loadlist = loadlist[0]
        ierr6, otherloads = psspy.alodbusint(-1, 1, 'NUMBER')
        otherloads = otherloads[0]
        if tbus in otherloads:
            otherloads.remove(tbus)
        loadnum = len(loadlist)
        othernum = len(otherloads)
        #outfile = r'C:\Users\Hema\Documents\FaultAnalysis\118bus_' + 'load' + loadbus.__str__()
+ '_lines' + bfault[0].__str__() + '-' + bfault[1].__str__() + '_' + param3[pid][0].__str__() + '_' +
param3[pid][1].__str__() + '_' + param3[pid][2].__str__() + '_' + param3[pid][3].__str__() + '.out'
        #datfile = r'C:\Users\Hema\Documents\FaultAnalysis\118bus_' + 'load' + loadbus.__str__()
+ '_lines' + bfault[0].__str__() + '-' + bfault[1].__str__() + '_' + param3[pid][0].__str__() + '_' +
param3[pid][1].__str__() + '_' + param3[pid][2].__str__() + '_' + param3[pid][3].__str__() + '.dat'
        ### Change name according to number of parameters
        outfile = r'C:\Users\Hema\OneDrive\Documents\Research\Neural Network\P LM mach all-
9param\118bus_' + 'load' + loadbus.__str__() + '_lines' + bfault[0].__str__() + '-' +
bfault[1].__str__() + '_' + param3[pid][0].__str__() + '_' + param3[pid][1].__str__() + '_' +
param3[pid][2].__str__() + '_' + param3[pid][3].__str__() + '_' + param3[pid][4].__str__() + '_' +
param3[pid][5].__str__() + '_' + param3[pid][6].__str__() + '_' + param3[pid][7].__str__() + '_' +
param3[pid][14].__str__() +'.out'
    datfile = r'C:\Users\Hema\OneDrive\Documents\Research\Neural Network\P LM mach all-
9param\118bus_' + 'load' + loadbus.__str__() + '_lines' + bfault[0].__str__() + '-' +
bfault[1].__str__() + '_' + param3[pid][0].__str__() + '_' + param3[pid][1].__str__() + '_' +
param3[pid][2].__str__() + '_' + param3[pid][3].__str__() + '_' + param3[pid][4].__str__() + '_' +
param3[pid][5].__str__() + '_' + param3[pid][6].__str__() + '_' + param3[pid][7].__str__() + '_' +
param3[pid][14].__str__() +'.dat'

    #Create SIDs
    psspy.bsys(1, 0, [0, 0], 0, [], loadnum, loadlist)
    psspy.bsys(2, 0, [0, 0], 0, [], 1, tbus)
    psspy.bsys(3, 0, [0, 0], 0, [], 2, [tbus, refbus])
    psspy.bsys(4, 0, [0, 0], 0, [], othernum, otherloads)

    #PowerFlow
    psspy.fnsl()

    #Dynamic Simulation Setup
    CASE2 = r"C:\Users\Hema\OneDrive\Documents\Research\Neural
Network\ieee118PSSE_rq.dyr"
    ierrdyn = psspy.dyre_new([1, 1, 1, 1], CASE2)
    psspy.dynamics_solution_param_2(1000)
    psspy.dynamics_solution_param_2(realar3=0.0013888889)  #Change Time Step for PMU
Alg.
    psspy.chsb(3, 0, [1, -1, -1, 1, 14, 0])     #Bus Voltage Mag and Angle  #Choose output
variables and channel index
    psspy.chsb(2, 0, [5, -1, -1, 1, 12, 0])    #Bus frequency
    psspy.chsb(2, 0, [6, -1, -1, 1, 25, 0])    #Load Power - P
    psspy.chsb(2, 0, [6+nLDs, -1, -1, 1, 26, 0])    #Load Power - Q

    #Add Load Model(s)
    #psspy.add_load_model(loadbus, 'BL',0,1,'CLODBL',0,[],[],8,[param3[pid][0],
param3[pid][1], 0, 0, param3[pid][2], 0.0, 0.0, 0.0])  #Add Complex Load Model
    if param3[pid][0]<100:
        psspy.add_load_model(tbus, 'BL', 0, 1, 'IEELBL', 0, [], [], 14, [tempZIP[0], tempZIP[1],
tempZIP[2], tempZIP[3], tempZIP[4], tempZIP[5], 0, 0, 2, 1, 0, 2, 1, 0])
    if param3[pid][0]>0:
        #psspy.add_load_model(loadbus, 'ML', 0, 1, 'CIM5BL', 1, 1, [] , 19, [0, 0, 3.8, 0.013, 0.067,
0.009, 0.17,0,0,0,0, 0, 1, 1.5, 0.0,0,0,0,0]) #[0, 0, Xm, R1, X1, R2, X2, 0,..., H,..]
```

```
            #psspy.add_load_model(loadbus, 'ML', 0, 1, 'CIM5BL', 1, 1, [] , 19, [0.011, 0.106,
tempmach[0], tempmach[1], tempmach[2], 0.009, 0.17,0,0,0,0, 0, 1, tempmach[3], 0.0,0,0,0,0])
#[0, 0, Xm, R1, X1, R2, X2, 0,..., H,..]
            for iLM in range(nLM):
                psspy.add_load_model(tbus, LID[iLM], 0, 1, 'CIM5BL', 1, 1, [] , 19, tempmach[:])
        #Create subsystem of generator buses with dynamics and net the rest of the generators
        #psspy.bsys(5, 0, [0, 0], 0, [], macnum, maclist)
        #psspy.netg(5, 0)

        #Convert Load at all Load Buses
        psspy.conl(1, 1, 1, [0, 0])
        ierr, rlods = psspy.conl(1, 1, 2, [0, 0], param[0]) #change 1st param to 2 and 2nd param to 0
for load bus or both params to 1 for all buses
        psspy.conl(1, 1, 3)

        #PSS/E Dyanmic Simulation Pre-Initialization Processes
        psspy.cong()
        psspy.ordr(1)
        psspy.fact()
        psspy.tysl(0)

    with silence():
        # Initialize and run Dynamic Simulation
        ierr1 = psspy.strt(0, outfile) #Check if load model parameters are correct
        try:
            ierr2 = psspy.okstrt()
        except psspy.OkstrtError:
            ierr2 = 1
    if (ierr1+ierr2)==0:
        try:
            with silence():
                psspy.run(0, 0) #3rd parameter 2=60/sec and 4=30/sec
                psspy.run(0, 0.20)
                psspy.dist_branch_fault(bfault[0], bfault[1], '1')
                psspy.run(0, 0.29)
                psspy.dist_branch_trip(bfault[0], bfault[1], '1')
                psspy.run(0, 0.67)
                psspy.dist_branch_close(bfault[0], bfault[1], '1')
                psspy.run(0, 0.77)
                psspy.dist_branch_trip(bfault[0], bfault[1], '1')
                psspy.run(0, 1.09)
                #Convert Output (.out) file to MATLAB (.dat) input.
                chan = dyntools.CHNF(outfile)
                data = chan.get_data()
                psspy.close_powerflow()


                # Sum Load Power #
                sumP = []
                sumQ = []
                for subL in range(nLDs):
                  sumP = np.sum([data[2][6+subL], sumP], axis=0)
                  sumQ = np.sum([data[2][6+nLDs+subL], sumQ], axis=0)
                #sumP = [sum(sublist) for sublist in itertools.izip(data[2][6:6+nLDs])]
                #sumQ = [sum(sublist) for sublist in itertools.izip(data[2][6+nLDs:6+nLDs+nLDs])]

                # Calculate Current from P, Q and V
                curm = []
                cura = []
                langle = []
                for pitem,qitem,vmitem,vaitem,raitem in
itertools.izip(sumP,sumQ,data[2][1],data[2][2],data[2][4]):
    #         for pitem,qitem,vmitem,vaitem,raitem in
itertools.izip(data[2][6],data[2][7],data[2][1],data[2][2],data[2][4]):
                  loadangle = vaitem - raitem
                  langle.append(loadangle)
                  current = (pitem + 1j*qitem)/(vmitem*np.exp(1j*np.radians(loadangle)))
                  currentm = np.abs(current)
                  currenta = -1*np.angle(current, deg=True) #Complex Conjugate in degrees
                  curm.append(currentm)
                  cura.append(currenta)

                fid = open(datfile, "w")
                fid.writelines("%s\n" % item for item in data[2][1])       #Load volt mag
                fid.write('EOF\n')
                fid.writelines("%s\n" % item for item in langle)           #Load volt ang minus Ref
                fid.write('EOF\n')
                #fid.writelines("%s\n" % item for item in data[2][2])       #Load volt ang
                #fid.write('EOF\n')
                #fid.writelines("%s\n" % item for item in data[2][3])       #Ref volt mag
                #fid.write('EOF\n')
                #fid.writelines("%s\n" % item for item in data[2][4])       #Ref volt ang
                #fid.write('EOF\n')
```

```
            fid.writelines("%s\n" % item for item in data[2][5])       #Load Freq
            fid.write('EOF\n')
            fid.writelines("%s\n" % item for item in curm)            #Load Current mag
            fid.write('EOF\n')
            fid.writelines("%s\n" % item for item in cura)            #Load Current Ang
            fid.close()

      except (RuntimeError, TypeError, NameError), e: #Error:
         print e
         print 'Error on line ',sys.exc_info()[-1].tb_lineno
         psspy.close_powerflow()
      else:
         psspy.close_powerflow()
   else:
      print "Start Error"
      psspy.close_powerflow()
   os.remove(outfile)
print "END OF PROGRAM"
```

F.3 Initial Neural Network Algorithm – MATLAB

```
%Parameter Estimation of Dynamic Load Model
%03/02/2015
%Hema Retty

clc
close all
clear all

ltime = 1.09;

% Parameters %
% filenamep = 'machparam_h.txt';
% fidp = fopen(filenamep);
% param1 = dlmread(filenamep);
% fclose(fidp);

filenamep = 'machparam_h2.txt';
fidp = fopen(filenamep);
param2 = dlmread(filenamep);
fclose(fidp);

param = param2; %[param1; param2];

% Size of Data Set %
if(fidp)
    n = size(param,1);
```

```
else
    prompt = 'Enter number of data sets:';
    n = input(prompt);
end

deletep = 0;
d = 1;
m = n;
for i=1:n
    %filename = uigetfile('*dat');     % gets the file name of contingency
case
    if round(param(i,1))==param(i,1)
        LMp = [num2str(param(i,1)) '.0'];
    else
        LMp = num2str(param(i,1));
    end
    if round(param(i,2))==param(i,2)
        XMp = [num2str(param(i,2)) '.0'];
    else
        XMp = num2str(param(i,2));
    end
    if round(param(i,5))==param(i,5)
        Hp = [num2str(param(i,5)) '.0'];
    else
        Hp = num2str(param(i,5));
    end
    %filename = ['118bus_load11_lines11-4_' num2str(param(i,1)) '_'
num2str(param(i,2)) '_' num2str(param(i,3)) '_' num2str(param(i,4)) '_'
num2str(param(i,5)) '.dat'];
    filename = ['118bus_load11_lines11-4_' LMp '_' XMp '_'
num2str(param(i,3)) '_' num2str(param(i,4)) '_' Hp '.dat'];
    if exist(filename, 'file')
        fid = fopen(filename);
        while ~feof(fid)
            data = textscan(fid, '%f');
            volt(i, :) = downsample(data{1,1},36);
            fgets(fid);
            data = textscan(fid, '%f');
            theta(i, :) = downsample(data{1,1},36);
            fgets(fid);
            data = textscan(fid, '%f');
            voltref(i, :) = data{1,1};
            fgets(fid);
            data = textscan(fid, '%f');
            thetaref(i, :) = data{1,1};
        end
        fclose(fid);
        % thetald(i, :) = theta(i, :)-thetaref(i, :);
    elseif(i==n)
        volt(i,:) = 0;
        theta(i,:) = 0;
        voltref(i,:) = 0;
        thetaref(i,:) = 0;
        m = m - 1;
        deletep(d) = i;
        d = d + 1;
    else
        m = m - 1;
        deletep(d) = i;
        d = d + 1;
    end
end
param(deletep, :) = [];
volt(deletep, :) = [];
theta(deletep, :) = [];
voltref(deletep, :) = [];
thetaref(deletep, :) = [];
ldata = size(volt, 2);
% Machine Learning %
[net, outputs, errors, performance] = neural_1(volt, param(1:m,:));   %
Neural
%network
% Test Case %
%filenamet = uigetfile('*dat');     % gets the file name of contingency case
paramt = [90, 3.1, 0.013, 0.08, 1]; %[23.0, 3.8, 0.013, 0.067, 1.0]; %[80,
3.71, 0.0139, 0.1605, 1]
filenamet = ['118bus_load11_lines11-4_' num2str(paramt(1)) '.0_'
num2str(paramt(2)) '_' num2str(paramt(3)) '_' num2str(paramt(4)) '_'
num2str(paramt(5)) '.0.dat']; %'118bus_load11_lines11-
4_23.0_3.8_0.013_0.067_1.5.dat';
fidt = fopen(filenamet);
while ~feof(fidt)
    datat = textscan(fidt, '%f');
    voltt = downsample(datat{1,1},36);
    fgets(fidt);
    datat = textscan(fidt, '%f');
    thetat = downsample(datat{1,1},36);
    fgets(fidt);
    datat = textscan(fidt, '%f');
    voltreft = datat{1,1};
    fgets(fidt);
    datat = textscan(fidt, '%f');
    thetareft = datat{1,1};
end
fclose(fidt);
% test = predict(rtree, voltt);
test = net(voltt); % Test with Neural Network
% test = voltt*beta'/(beta*beta');   % Test with MV Linear Reg.

%Error
paramt'
test
Error = (abs(paramt - test')./(max(param, [], 1) - min(param, [], 1)))*100;
Error'

% Plot Fault Data %
rate = ltime/(ldata-1);
simtime(:,1) = 0:rate:(rate*ldata-rate);
```

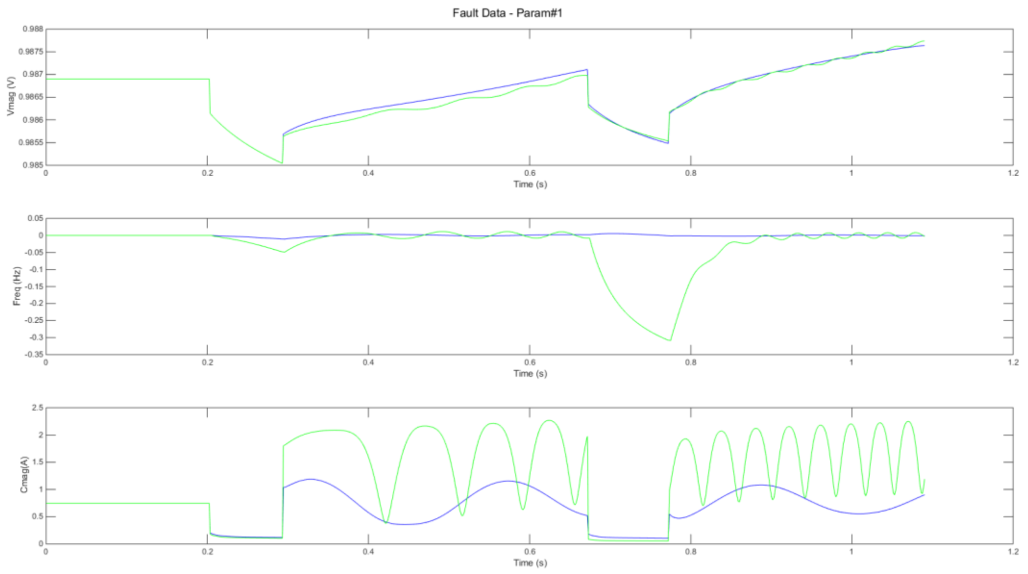
```
figure
suptitle('Fault Data - PSS/E');
subplot(2,1,1)
plot(simtime,volt(1,:), 'g');
hold on
plot(simtime,volt(40,:), 'b--');
plot(simtime,volt(200,:), 'r:');
hold off
xlabel('Time (s)');
ylabel('Vmag (V)');
axis([0 1.4 0 1.1]);
subplot(2,1,2);
plot(simtime,theta(1,:), 'g');
hold on
plot(simtime,theta(2,:),'b--');
plot(simtime,theta(3,:),'r:');
hold off
xlabel('Time (s)');
ylabel('Vang (theta)');
axis([0 1.4 0 55]);



% Solve an Input-Output Fitting problem with a Neural Network
% Script generated by NFTOOL
% Created Tue Mar 03 22:52:35 EST 2015
%
% This script assumes these variables are defined:
%
%   volt - input data.
%   param - target data.
function [net, outputs, errors, performance] = neural_2(volt, param)

    inputs = volt;
    targets = param;

    % Create a Fitting Network
    hiddenLayerSize = 50; %30;
    net = fitnet(hiddenLayerSize);
    net.trainFcn = 'trainbr';
    %net.trainFcn = 'trainscg';

    % Setup Division of Data for Training, Validation, Testing
    net.divideParam.trainRatio = 70/100;
    net.divideParam.valRatio = 15/100;
    net.divideParam.testRatio = 15/100;

    % Number of Iterations
    net.trainParam.epochs = 10;
    net.trainParam.max_fail = 3;

    % Train the Network
```

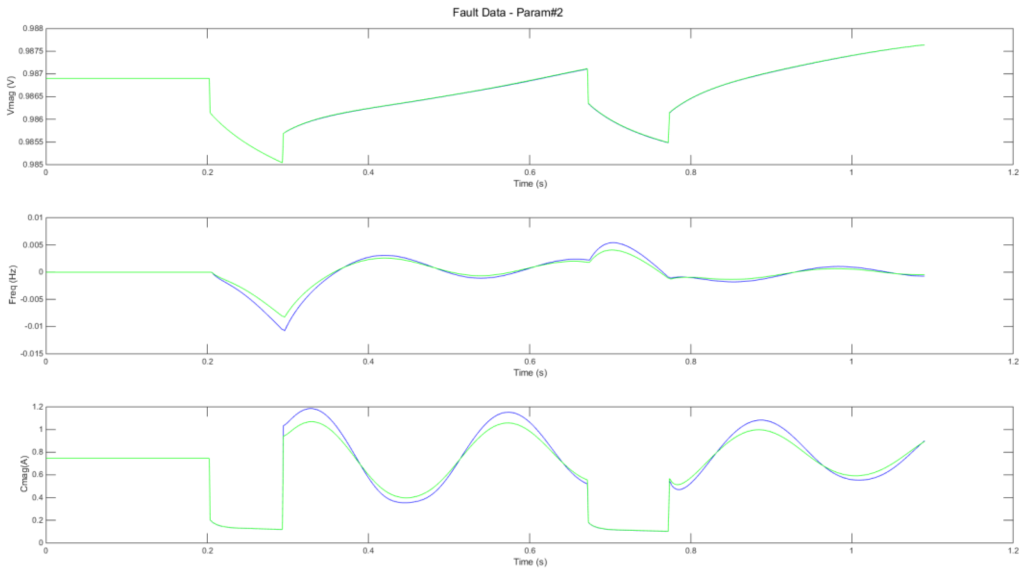
```
matlabpool open
[net,tr] = train(net,inputs,targets, 'useParallel', 'yes');
matlabpool close

% Test the Network
outputs = net(inputs);
errors = gsubtract(targets,outputs);
performance = perform(net,targets,outputs);

% View the Network
%view(net)

% Plots
% Uncomment these lines to enable various plots.
figure, plotperform(tr)
%figure, plottrainstate(tr)
%figure, plotfit(net,inputs,targets)
%figure, plotregression(targets,outputs)
%figure, ploterrhist(errors)
```

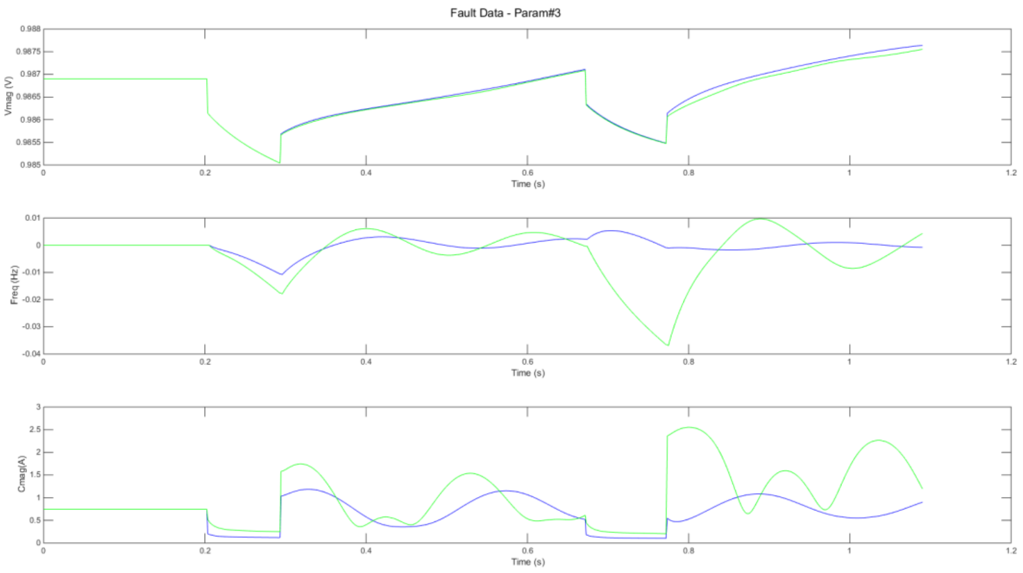
Appendix G. Ensemble Averaging with Neural Networks - MATLAB

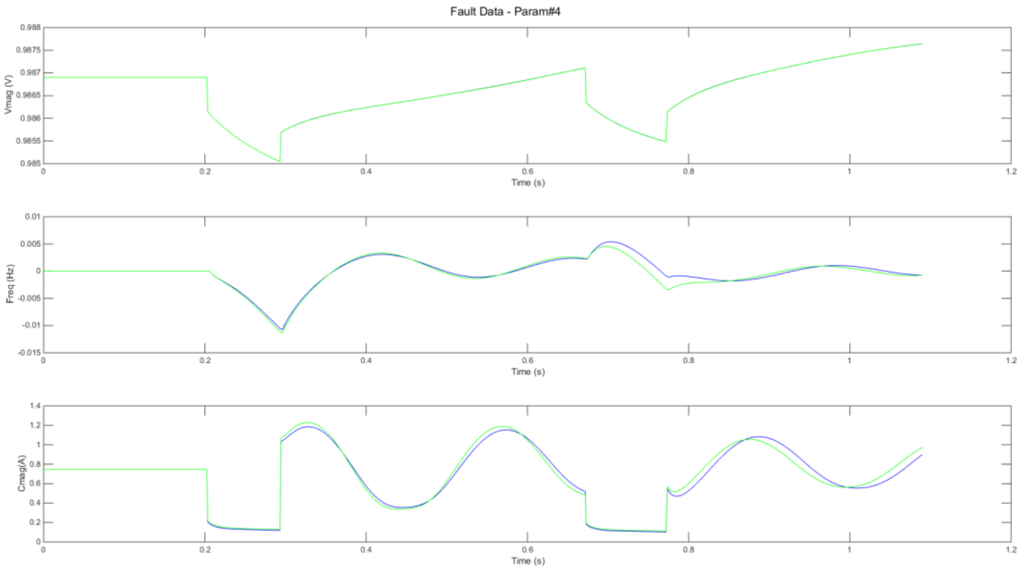
```
% Solve an Input-Output Fitting problem with a Neural Network
%
% Created Tue Mar 03 22:52:35 EST 2015
%
% This script assumes these variables are defined:
%
%   meas - input data.
%   param - target data.
function [net, avgerrs, avgperf] = neural_std4(meas, param, hlayers, mep,
numnet)
   % Test Set  - Removed from Training & Validation Set %
    dsize = size(param,2);
    tsize = dsize - roundn(dsize - round(.2*dsize),-2);
    tperm = randperm(dsize);
    tinputs = meas(:,tperm(1:tsize));
    ttargets = param(:,tperm(:,1:tsize));
    meas(:, tperm(:, 1:tsize)) = [];
    param(:, tperm(:, 1:tsize)) = [];

    % Variables %
    ndata = size(meas,2);
    nperm = ndata;
    unq = 0.7; %Percentage of unique samples
    %totouts = 0;
    toterrs=  0;
    totperf = 0;
    %matlabpool open
    parpool;

    parfor i=1:numnet
        %indperm = randperm(ndata);
        %indperm = randi(nperm, [nperm,1]); %1 was i?

        % Bagging w/ Replacement %
        % Duplicates only once. Replace 'randsample' with 'datasample' to
have
        % random number of duplicates for 1-unq percentage of data
        indperm1 = randperm(nperm);
        indperm2 = [indperm1(1:round(unq*nperm))
randsample(indperm1(1:round(unq*nperm)),nperm-round(unq*nperm))];
        indperm = randsample(indperm2,nperm);
        inputs = meas(:,indperm); %Can concatanate tinputs and targets if
test set must be defined
        targets = param(:,indperm);

        % Create a Fitting Network
        net{i} = fitnet(hlayers); %[hiddenLayer1Size hiddenLayer2Size]
        net{i}.trainFcn = 'trainbr';
        %net(i).trainFcn = 'trainscg';
        %net(i).trainFcn = 'trainrp';
        % Number of Iterations
        net{i}.trainParam.epochs = mep;
        net{i}.trainParam.max_fail = 5;

        % Setup Division of Data for Training, Validation, Testing
        net{i}.divideFcn = 'divideind';
        net{i}.divideParam.trainInd = indperm(1:round(0.85*nperm));
        net{i}.divideParam.valInd = indperm(round(0.85*nperm+1):nperm);
        %net{i}.divideParam.testInd = indperm(round(0.85*nperm+1:nperm));

        % Train the Network
        net{i} = train(net{i},inputs,targets, 'useParallel', 'yes');

        % Test the Network
        outputs(:,:,i) = sim(net{i},tinputs, 'useParallel', 'yes');
        errors(:,:,i) = abs(gsubtract(ttargets,outputs(:, :,i)));
%outputs(i,:,:))); %./(repmat(max(param,[],2)-min(param,[],2),1,nperm))*100;
        performance(i) = perform(net{i},ttargets,outputs(:,:,i));
%outputs(i,:,:));

        % Bagging Outputs %
        %totouts = totouts + outputs{i,1:p,1:ndata};
        avgperror(:,i) = (sum(errors(:,:,i),2))./nperm;
        toterrs = toterrs + avgperror(:,i);
        totperf = totperf + performance(i);

    end
    % Avg Ensemble Output %
    %avgouts = totouts./numnet;
    avgerrs = toterrs./numnet;
    avgperf = totperf./numnet;
    %matlabpool close
    delete(gcp);

    % View the Network
    %view(net)

    % Plots
    % Uncomment these lines to enable various plots.
    %figure, plot(sum(avgperror,1), '*')
    %xlabel('Neural Networks')
    %ylabel('Avg. Abs. Error')
    %figure, plotperform(avgtr)
    %figure, plottrainstate(tr)
    %figure, plotfit(net,inputs,targets)
    %figure, plotregression(targets,outputs)
    %figure, ploterrhist(errors)
```

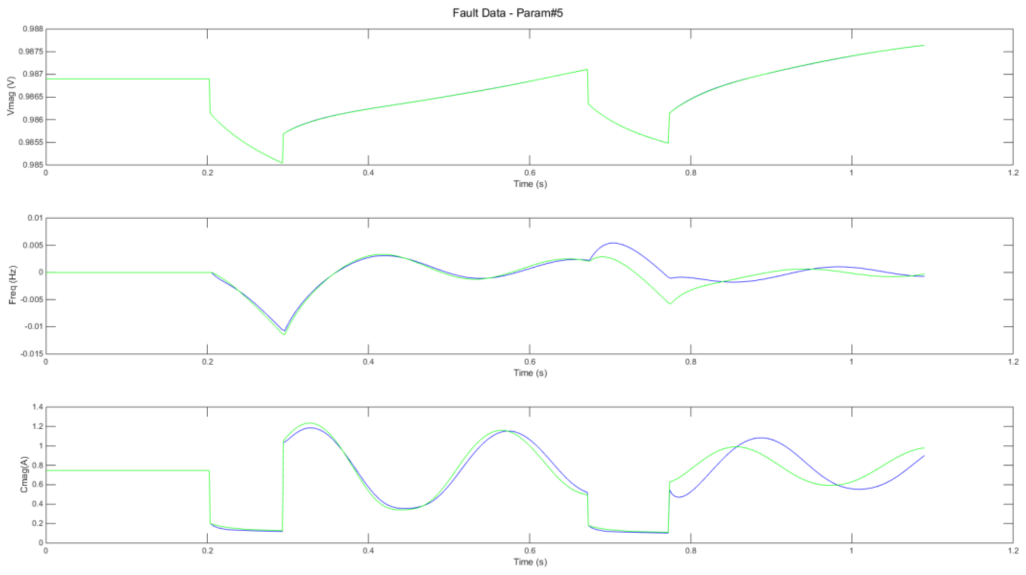
Appendix H. Comparison of Initial Six Parameters



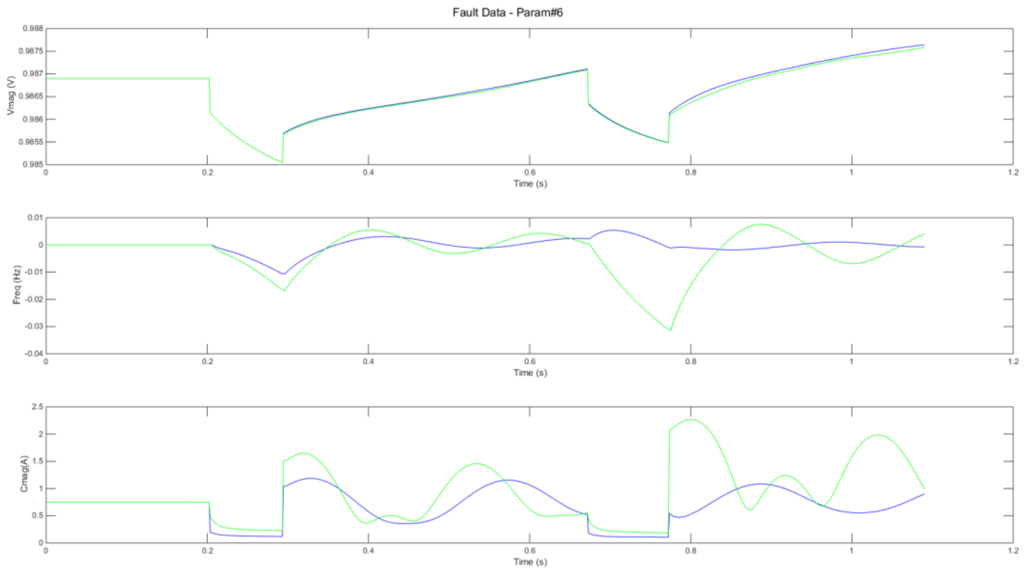



Fault Data - Param#3
Vmag (V)
Freq (Hz)
Cmag(A)
Time (s)


Fault Data - Param#4
Vmag (V)
Freq (Hz)
Cmag(A)
Time (s)



Fault Data - Param#5
Vmag (V)
Freq (Hz)
Cmag(A)
Time (s)



Fault Data - Param#6
Vmag (V)
Freq (Hz)
Cmag(A)
Time (s)

Appendix I. Neural Network Optimization – MATLAB

```
%Parameter Estimation of Dynamic Load Model
%05/13/2015
%Hema Retty

clc
close all
%clear all

ltime = 1.09;
ilayers = [55 25 16];
eps = 100;
numnet = 10;
load FaultData4

% Remove Constant Parameters %
param = param(1:4,:);

%-- Machine Learning --%

% Normalize Parameters for MSE %
param = normr(param);
% d = size(param,2);
minp = min(param(:,2:end), [], 2);
maxp = max(param(:,2:end), [], 2);
% param = (param - repmat(minp,1,d))./repmat((maxp - minp),1,d);
% param(5,:) = repmat(1.0,1,d);

% PCA - Preprocessing %
maxfrac = 0.000002;
%[voltnorm, ps] = mapstd(psdv);
[voltPCA, PS] = processpca(volt, maxfrac); %psdv
%[freqnorm, ps3] = mapstd(psdf);
[freqPCA, PS3]= processpca(freq, maxfrac); %psdf
%[curmagnorm, ps4] = mapstd(curmag);
[curmagPCA, PS4]= processpca(curmag, maxfrac); %psdc
vpca = [voltPCA; freqPCA; curmagPCA];
% vpca = [psdv; psdf; psdc];

lfix = 0;
err = 1;
i = 1;
layers(i,:) = ilayers;
errstor(i) = err;
AvgError(:,i) = [1;1;1;1];
save AvgError4param_optlayer3_3-3clu AvgError errstor layers
% Change layer index and size to improve optimization
while(err>0.1)
    errp =  err;
    tic
    % Network Training %
    [net, errors, performance] = neural_std42(vpca, param, layers(i), eps,
numnet);   % Neural
    toc

    % Error Calculation %
    i = i + 1;
    layers(i, :) = layers(i-1, :);
    AvgError(:,i) = (errors./(maxp-minp)).*100;
    err = sum(AvgError(:,i-1) - AvgError(:,i));
    errstor(i) = err;
    if(err>0.1)
        layers(i,3) = layers(i,3) + 1;
    elseif(err<-0.1)
        layers(i,3) = layers(i,3) - 1;
    else
        err = 0;
    end
%     disp('Iteration: ');
%     disp(i);
%     disp('err: ');
%     disp(err);

    % Save Output %
    save AvgError4param_optlayer3_3-3clu AvgError errstor layers -append
end
% Save Output %
save AvgError4param_optlayer3_3-3clu net errors performance -append
%save AvgError4param_optlayer3_3clu AvgError net errors performance layers
```