MODELING POWER SYSTEM LOAD USING INTELLIGENT METHODS

by

SHENGYANG HE

B.S., University of Waterloo, 2006

A THESIS

submitted in partial fulfillment of the requirements for the degree

MASTER OF SCIENCE

Department of Electrical Engineering
College of Engineering

KANSAS STATE UNIVERSITY
Manhattan, Kansas

2011

Approved by:

Major Professor
Shelli K. Starrett

Abstract

Modern power systems are integrated, complex, dynamic systems. Due to the complexity, power system operation and control need to be analyzed using numerical simulation. The load model is one of the least known models among the many components in the power system operation. The two different load models are the static and dynamic models.

The ZIP load model has been extensively studied. This has widely applied to composite load models that could maintain constant impedance, constant current, and/or constant power. In this work, various Neural Networks algorithms and fuzzy logic have been used to obtain these ZIP load model coefficients for determining the percentage of constant impedance, current, or power for the various load buses. The inputs are a combination of voltage, voltage change, and power change, or voltage and power, and the outputs consist of the ZIP load model coefficients for determining the type and the percentage of load at the bus. The trained model is used to predict the type and percentage of constant load at other buses using simulated transient data from the 16-generator system. A small study was also done using a dynamic induction machine model in addition to the ZIP load model. As expected, the results show that the dynamic model is more difficult to determine than the static model.

Table of Contents

List of Figures

List of Tables

List of Equations

Acknowledgements

I would like to gratefully acknowledge and thank my advisor Shelli K. Starrett, for her valuable contribution, advice, guidance, encouragement, and corrections during the entire study and research of this work.

I would like to thank Dr. Stanton for his valuable guidance and advice for this research. I would also like to thank Dr. Das and Dr. Miller for being on my committee and for their valuable advice and support.

I gratefully acknowledge the financial support provided in part by the Kansas State University Power Affiliates Project.

I would also give thanks to my parents, my sister, and friends for their encouragement and support throughout my Master's degree.

Dedication

I would like to dedicate this research to God and to my parents who have always been there to support and encourage me.

CHAPTER 1 - INTRODUCTION

1.1 Introduction and Motivation

On August 14, 2003, a blackout took place in the North American Eastern Interconnection that included most of New York State, parts of Pennsylvania, Ohio, Michigan, and Ontario, Canada. About 63 GW of load was interrupted. This magnitude equals approximately 11% of the total load distributed in the Eastern Interconnection of the North American system. Another important discovery was that there were significant reactive power supply problems in the states of Indiana and Ohio prior to noon. The first important event was the outage that took place in First Energy's Eastlake unit 5 generator as the area was generating high levels of reactive power. As a result, the Eastlake unit 5 voltage regulator tripped because of over-excitation.

Due to the cascading loss of major tie lines in Ohio and Michigan, a huge 3700 MW reverse power flow was serving load in the Ohio and Michigan system and caused heavy loading on the transmission around the region. In the end, this whole series of events resulted in a cascading outage of several hundred lines and generators tripped in the entire region.

The primary causes were summarized as inadequate understanding of the system, inadequate level of situation awareness, inadequate level of vegetation management (tree trimming), and an inadequate level of support from the reliability coordinator. The figure for the August 14, 2003 cascading outage is shown below.

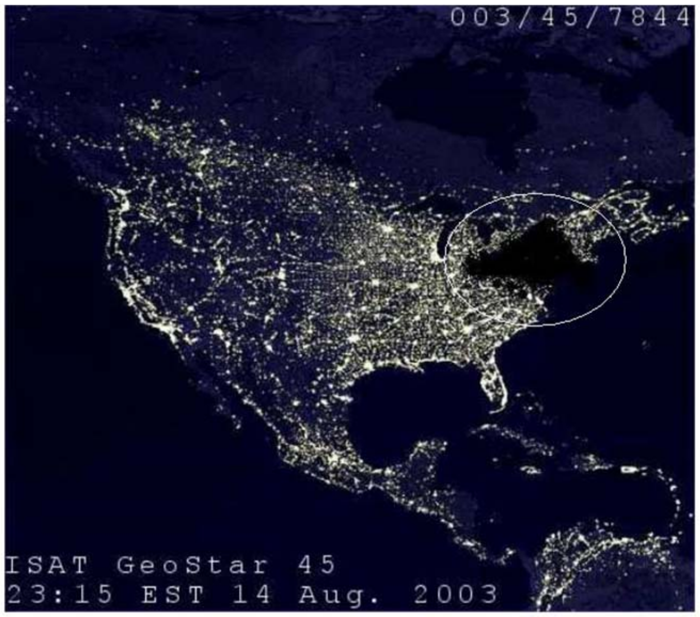


Figure 1.1 Display of the blackout on August 2003 [1]

Similarly, on September 23, 2003, voltage collapse took place in Southern Sweden and eastern Denmark, and led to the separation of a region of the Southern Swedish and Eastern Denmark system. This collapse was caused by both voltage and frequency.

Subsequently, an Italian blackout took place on September 28, 2003. The phase angle became too large due to heavy power import into Italy, and a 380 kV line was tripped on the same border. These outages left the Italian system with a shortage of 6400 MW of power, and the system collapse caused a nationwide blackout. It was the worst blackout in the history of its nation.

The Power System Stability and Power System Dynamic Performance Committee gathered experts from around the globe to have different kinds of sessions to discuss these issues. The root causes and necessary steps to reduce the risk of blackouts were the main focus of the panel session [2]. Recommendations for these blackouts by some of the presenters [2] announce that there were also data management issues. It seems there are definitely needs to improve the calibration of recording instruments, data collection, and establishment of an infrastructure to support a centralized blackout investigation. Moreover, there is a need to facilitate better insights into the cause of blackouts. As an example, WAMS or wide area measurement systems need to be established. Related studies were recommended and include observing Eigenanalysis, HVDC, and FACTS equipments to examine the system response to test inputs. Furthermore, tracking the implementation of recommended actions to improve reliability was also suggested. The application of automatic controls such as automatic voltage regulators and applicable power system stabilizers should be mandatory for generators. It is also important to build rapid system restoration to minimize the impact of a blackout.

Transient stability and load model are closely related to the power systems collapse. In reference [3], the author stated that as the load demand gets higher, the varying load becomes unstable as we try to control the power. The reduction in load impedance also will decrease the power. The load characteristics play a vital role in determining whether the voltage will progressively decrease or the system will stay stable. For constant impedance static load characteristics, the desired power and voltage will be higher than the actual power and voltage levels. Furthermore, the load-power factor affects the power-voltage characteristics of the system. As the voltage declines in the transmission line, the value will be a function of active and reactive transfer. We will describe voltage stability relies on the relationships between P, Q, and v [3]. Specifically, the P-V and Q-V curves are used extensively in voltage stability study. As power system stable operation relies on the capability to match the electrical load on the system to the electrical output, the parameters data and values for load model obtained can be very useful in determining the stability of the power systems [3].

1.2 Objective of the Research

Load modeling is not an easy task in a complex power system. A load bus representation in stability research consists of a combination of devices including fluorescent and incandescent lamps, refrigerators, heaters, compressors, motors, furnaces, and so on. The exact components of a load are very hard to describe mathematically or physically. The composition of the load will also change during the day and season, as well as with the weather conditions and the state of the economy. There are virtually millions of components in the total load supplied by the power systems. Hence, it becomes quite impractical to display all load components in a load model. To effectively study the load representation in a power system, we therefore really need to have a simplified model [3]. This thesis will describe simple load modeling concepts, load composition and characteristics as well as the obtaining of load model parameters. Load models from measurements of simulated system data will be provided using different intelligent computing techniques. Obtaining the load model parameters from measurement data will be the main goal. Having intelligent systems to quickly and accurately determine load parameters to be fed to voltage stability algorithms and others can help operators to have a more realistic picture of current system conditions and vulnerabilities. By doing so, these load parameters can be useful to provide analysis to prevent voltage collapse, to enable solutions to the power system collapse, and to allow future study of the system in smart grid. It is important to find accurate numerical methods and accurate data from the load buses, and to solve problems quickly and to obtain the solutions to aim in solving and preventing power systems from collapsing. The important step is to choose a suitable load model to represent one or two main power system areas for study.

1.3 Overview of Research

This research used various intelligent techniques to do a load modeling. The two preliminary work categories included using pseudo inverse and manual calculations to find load parameters. The later sections used various techniques including Levenberg-Marqauardt algorithm, Widow-Hoff backpropagation, Default Scaled Conjugate gradient Algorithm, Adaptive-Neural Fuzzy Inference Systems (ANFIS), and traditional neural networks method to find the parameters of the load model both in static and dynamic models. The objectives and approaches to achieve the goals are outlined as below.

Simple Modeling:

- Manual calculation of the parameters from a simple load constructed by the Microsoft Excel spreadsheet
- Inverse matrix or the simple pseudo inverse to calculate the over-determined parameters from the known values
- Linear programming approach

Intelligent Modeling:

- Utilize data from simulated, multi-machine systems to develop load models. The load voltages, real and reactive power, and the parameters that are associated with constant impedance, current and power are used as the input and output for the training.

- The above variables and parameters are used with the power changes and voltage changes at the different time steps for training and testing.
- The range of load parameters determining levels of constant impedance, current, and power and later percentage of dynamic load values are interchanged, and the different snap shots are taken for different simulation time steps to construct 15,552 or more load cases.
- Training, validating, and testing with the intelligent techniques were done to use the trained information to predict the information on the other load buses for both static and dynamic loads.
- Various strategies are used to training and testing the load buses, including combining some load buses from each case and test against all the load buses to determine the best data for training.
- The different intelligent methods and different strategies of testing and training and their MSE efficiency were recorded and compared to determine the best training strategy and method that can be used.

1.4 Organization of Thesis

The structure of the thesis is outlined as follows.

This chapter describes the power system load and the power system failure or collapse in recent years. Also, it gives the motivation and the need for the work and research.

Chapter 2 discusses the background of load modeling. The recent study and research done by others on load modeling are reviewed. The reviews of voltage stability aspects are described.

Chapter 3 talks about the problems and the preparation for this research. It explains the models and methods used at the early stage of the research and the shortcomings for these early experiments. Test cases are shown to prepare for this early research. It introduces different methods to solve load models. ZIP model parameters and various other forms of representation are introduced. Methods including pseudo-inverse, manual equation methods and inverse matrix methods are described. The advantages and disadvantages of these methods will be discussed.

Chapter 4 deals with a larger load model problem with the focus on the load voltage, load real and reactive power, and the frequency at the load. The transient data from the 16-generator and 52-bus system is extensively studied, and is used for neural networks training. The coefficients p_1, p_2, p_3, q_1, q_2, and q_3 from the ZIP load model are obtained through the well trained neural networks. The focus is on the static power system loads. A combination of load bus training and testing was done here with several intelligent methods including Levenberg-Marquardt, Widow-Hoff backpropagation, Default Scaled Conjugate gradient algorithm, and ANFIS.

Chapter 5 used the similar approach as in Chapter 4 but the work is focused on testing the approach for dynamic power system loads.

Chapter 6 concludes the present thesis work with some suggestions and for improvement of the study as well as some proposed future research work based on the current knowledge and work.

CHAPTER 2 - BACKGROUND

2.1 Introduction

This chapter gives us a background of load model structure and the recent research and knowledge of load model. The primary objective of the current research work is to have a good knowledge of the power system load and to be able to learn and predict the relations between the load variables such as real and reactive power and voltages and the different parameters that determine the amount of constant current, power, and current of the load. Finding the type of load is important to give us a deeper understanding of the power system model for computer simulations.

2.2 Voltage and Load Stability

The constant change in the electricity industry creates a new feature of our power systems. It is represented by complex interconnections, and the applying of a large variety of controllers for improving the system operation and the utilizing of available sources. Furthermore, the deregulation also causes the interconnected power network to be more complex. Therefore, the need for power networks to understand channels for the transfer of electricity from points of production to points of consumption is crucial. This process depends on a competitive system and time varying factors. The complexity of the system, the nature of the dynamics that cause it and the external factors interfering simultaneously require extra care, in order to maintain a suitably operated power system. The system must supply high reliability at minimum cost and ensure the smallest impact on the natural environment. To avoid inconvenience to customers and technical problems which lead to higher costs, the system needs to handle the frequent variations in active and reactive load. High levels of system security, availability of "spinning" reserve of active and reactive power, high quality in the design of the system components and availability of different paths for the delivery of the energy to the customers are very important [4]. The load representation in these stability studies is discussed later on in this chapter.

As described in [3], "voltage stability is the ability of a power system to maintain steady acceptable voltage at all buses in the system at normal operating conditions, and after being subjected to a disturbance." The important message includes not only maintain the steady acceptable voltage, but to sustain the voltage after a disturbance and to restore it to a state similar to the pre-disturbance situation. When the voltage in the system is not controllable and decreases linearly, variations in load or inappropriate voltage control devices may be applied as the system becomes voltage unstable. It is a requirement of the power system to stay in equilibrium under normal scenarios. Referring to [2], "voltage instability is the absence of voltage stability, and results in progressive voltage collapse (or increase)". The primary cause of this unstable situation in the power system is the lack of ability to satisfy the reactive load demand under heavily stressed conditions. Voltage collapse causes the system to have below normal voltage in a large part of the power system, and therefore will result in collapse of the power system. The load characteristic and its dynamics dictate the dependency between the load and the voltage, and therefore affect the voltage stability phenomenon. A voltage decrease at the beginning will cause decay in load, and after few seconds, a load restoration process needs to be started. To make it more complicated, the restoration can lead to heavily stressed loaded conditions. The restoration will cause voltage instability and even voltage collapse if the conditions remain for too long or suitable control decisions are not taken or the system is not able to resolve the reactive load demand.

Furthermore, power system stability can be categorized into angle stability and voltage stability as described in [3]. Voltage stability can be divided into short and long-term voltage stability. The 'short-term' is a time frame of about a few seconds, and describes the dynamics of components such as induction motors, static VAR compensators and excitation of synchronous generators. When the dynamics of the system react in slower time frames, around several minutes, this is termed 'long-term stability", two kinds of stability problems can take place as well as problems in frequency and voltage. Frequency stability problems are the result of power not balanced between generators and loads after a large disturbance, and can result in systems isolated from the main system [3]. Studies of each of these stability problems require some sort of load model.

2.3 History of Load Modeling

As the current trend shows, load modeling has gained interest as the power system load becomes the new area of research within power system stability. A couple of studies, [5 and 6] have demonstrated the key effect of load representation in voltage stability studies. Because of that, there is a need to find more accurate load models than those already used traditional ones (e.g. all constant impedance or all constant power).

As voltage collapses only took several minutes in the past "real world" cases, the older load modeling work was focused on induction machines, critical in the range of some seconds after a disturbance. The load response was taken as a function of voltage [7]. The use of dynamic load models has become increasingly popular compared to the static load models. Although knowledge has been acquired from power system load in the recently years, it is one of the most difficult and unknown areas of study in the midst of the power system models. This is because of the diverse and complex load components, the high distribution and variation during the time of day and year, weather, and the lack of information for the load. The new techniques for the load modeling will result in better understanding of the load and better representation of the load in simulations of the system. This will help to have a positive impact for the control, operation, and reliability of the power system. The accurate load model and a real-time monitoring application will help to introduce more competiveness for the electric industry and contribute to the development of smart grid information structure [8].

A model that uses a fundamental engineering knowledge and describes the physical phenomena of a system is called a physical model. The model uses the elementary laws that will give accurate results when simulating. However, for a more complex system, using the physical laws to solely find the specific parameters could be difficult. Instead, developing the model based on empirical laws may be easier and more practical. When developing the load model with empirical laws, only input and output signals are needed to form a relation because usually sufficient knowledge is not provided to form a physical model. However, the input and output can still be used to form a mathematical relation to model the system. This approach is also called a "black box model" as there is limited amount of available data to find the relation between the input and output of the system. The physical model for power system loads will be utilized in the work of this thesis.

As referenced by [9], the author showed that the load model consists of two different characteristics: the static and dynamic, where static part is commonly modeled by the ZIP load model and the dynamic characteristics can be more complex, often involving induction machine models. The voltage variation of the system is affected by the load variations. Two major categories for the study include the physical models and the non-physical models. The physical model includes the widely applied composite load model that needs the system to maintain the constant impedance, the constant current, or the constant power (e.g. ZIP load model is one kind of physical one). The non-physical load models include the exponential load model, the difference equations model, and the neuron-net model etc. In [9], the non-linear least square method worked well for the simple load model with measured active load.

The two different composite load characteristics used for load modeling are the measurement-based approach and the component-based approach. The measurement-based approach requires people to measure the data at substations and feeders directly. The voltage, frequency, sensitivity of the active P and reactive Q load were recorded through this approach [9]. The data was received from measurements on site and included the voltage and frequency variations as well as the changes in active and reactive load. In order to create this model by putting the measured data to an approximate model, the parameters would be generated. This is also called gray-box modeling as a structure model was constructed from measured data. For static models, this technique is much easier compared to a dynamic model. The advantage is that this technique is fairly accurate and can be applied as economical investments to observe the most important loads.

From [6], on the other hand, the component-based approach focuses on constructing a composite load model from data on the constituent parts. For instance, information can be obtained from a combination or composition of substations, classes, or load components. The load composition data is important in this approach and it defines the percentage of each load component. The load characteristic data is associated to the physical characteristics for those load components. According to [10], similarly, component-based approach works well with the static load model, but does a poor job of describing the dynamic load characteristics. Load parameter representation for dynamic performance can be very complicated as the load varies along with time. It is a big challenge to change the parameters to follow the variation. The load model parameters need to be adjusted as the load composition changes. Hence, additional changes need to be done for the dynamic load model. The main advantage of using this technique is that no field measurements are required. Also, this model can adapt to different systems and conditions easier as well as applying it to use. However, because of the dependence on weather and time, the model needs to be constantly updated to be more accurate. For the research purpose and the need to simplify the study, off-line data is usually studied as it corresponds to a period of time.

2.4 The Structure of Load and Load Model

The mathematical representation of the relationship between a bus voltage (magnitude and frequency) and the power (active and reactive) or current flowing into the bus load is called a load model [9]. It is important for the design, planning, and operation of a power system. Since the load parameters are usually non-linear, it is a challenging problem to describe the dynamic characteristics of the load.

As discussed previously, load models can be constructed based on two different approaches. One approach is measuring the voltage and frequency sensitivity of the active and reactive powers at the substation or load bus. The second approach constructs a composite load model for a given substation or load bus, according to the mix of load classes at the substation [11].

The voltages referred to in this work are the per unit values where the v is the voltage magnitude.

$$\bar{v} = \frac{v}{v_0} \qquad \textbf{(2.1)}$$

In the general structure of the ZIP load model used in this work, the active and reactive power have three components: constant impedance (Z), a constant current (I) and a constant power (P) injections. The general model, which represents the voltage dependency of loads, is of the form given below. In this thesis, the parameters given equations (2.2) and (2.3) will be used.

$$P_{ZIP} = P_0[p_1\bar{v}^2 + p_2\bar{v} + p_3] \qquad \textbf{(2.2)}$$

$$Q_{ZIP} = Q_0[q_1\bar{v}^2 + q_2\bar{v} + q_3] \qquad \textbf{(2.3)}$$

The parameters of the model are p_1 to p_3 and q_1 to q_3, which define the proportion of each component respectively. The frequency dependency of load characteristics is represented in the polynomial ZIP model by a factor including Δf as indicated below.

$$P_{ZIP} = P_0[p_1\bar{v}^2 + p_2\bar{v} + p_3](1 + k_{pf}\Delta f) \quad \textbf{(2.4)}$$

$$Q_{ZIP} = Q_0[q_1\bar{v}^2 + q_2\bar{v} + q_3](1 + k_{qf}\Delta f) \quad \textbf{(2.5)}$$

Δf is the frequency deviation (f-f0). K_{pf} ranges from 0 to 3, and K_{qf} ranges from -2 to 0 [5]. The frequency in the study is found from time step data using the form below:

$$\Delta f = \frac{\theta(i+1) - \theta(i)}{t(i+1) - t(i)} \frac{1}{2\pi} \quad \textbf{(2.6)}$$

Aside from the ZIP model or polynomial model, there is also an exponential load model in which the voltage depends on the load power exponentially. For the exponential model, the real power and reactive power can be expressed in equations (2.7) and (2.8) below:

$$P_{ZIP} = P_0[\bar{v}]^a \quad \textbf{(2.7)}$$

$$Q_{ZIP} = Q_0[\bar{v}]^b \quad \textbf{(2.8)}$$

Similarly, the frequency dependent components can be added to the exponential model as shown in equations (2.9) and (2.10) below [3].

$$P_{ZIP} = P_0[\bar{v}]^c (1 + k_{pf}\Delta f) \quad \textbf{(2.9)}$$

$$Q_{ZIP} = Q_0[\bar{v}]^d (1 + k_{qf}\Delta f) \quad \textbf{(2.10)}$$

The exponential components a, b, c, d have common values for various load components such as air conditioners, resistance space heater, small industrial motors, or fluorescent lightening [6].

In [3], the author pointed out that the dynamic load models are more complex because the response of the loads to voltage and frequency variations is a lot faster. The static models used before will be ineffective models in this fast response case. However, the detecting of parameters in this case can still rely on measurement based and component-based approaches. More accurate measurements need to be taken as the system's response changes rapidly. Due to the complexity, long term stability, inter-area oscillations, voltage stability become important criterion to be modeled. The study of systems with large motor loads may require them to be represented in load dynamics as 60 to 70% of the total energy was consumed. Therefore motors are usually the most important part of dynamic system modeling. Other load components that will affect the stability study are as below:

- Discharging the lamps and restart the lamp. The lamps that have mercury vapor, sodium vapor, and fluorescent lamps will affect the voltage recovery or delay the recovery.
- Protective relays especially thermal or over-current relays
- Thermostatic control of loads (i.e. heaters/coolers, water heaters, and refrigerators.). The voltage will drop once these devices connect to the system.
- ULTCs on distribution transformers, voltage regulators, and voltage-controlled capacitor banks. Although these are not modeled in a lot studies. They make a difference in the load. Also these devices help to mitigate a disturbance and aim to make the system return to the pre-disturbance levels.

Static and dynamic load models are shown in Figure 2.1 below. It represents a wide range of characteristics shown by many load components. This is an aggregated or complex load model including small induction motors, large induction motors, static load characteristics, etc.

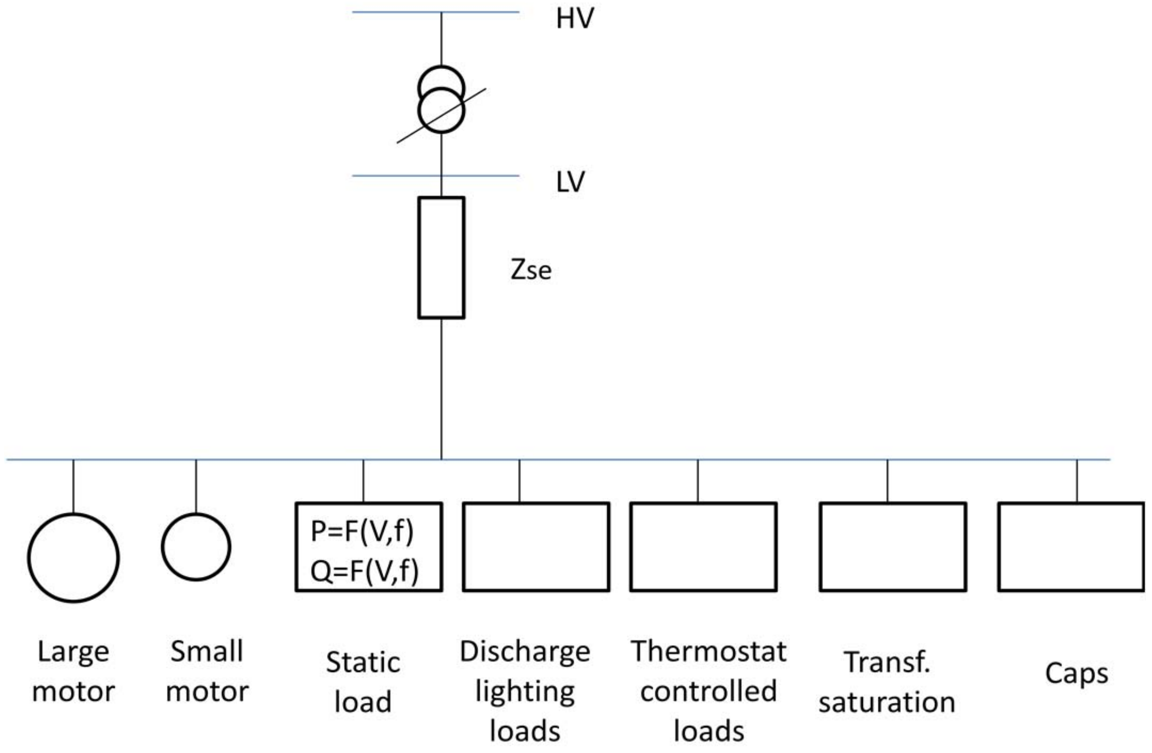


Figure 2.1 Static and dynamic load model [3]

The induction load model is an important component in dynamic load modeling. In Figure 2.2, a simple induction motor model is displayed.

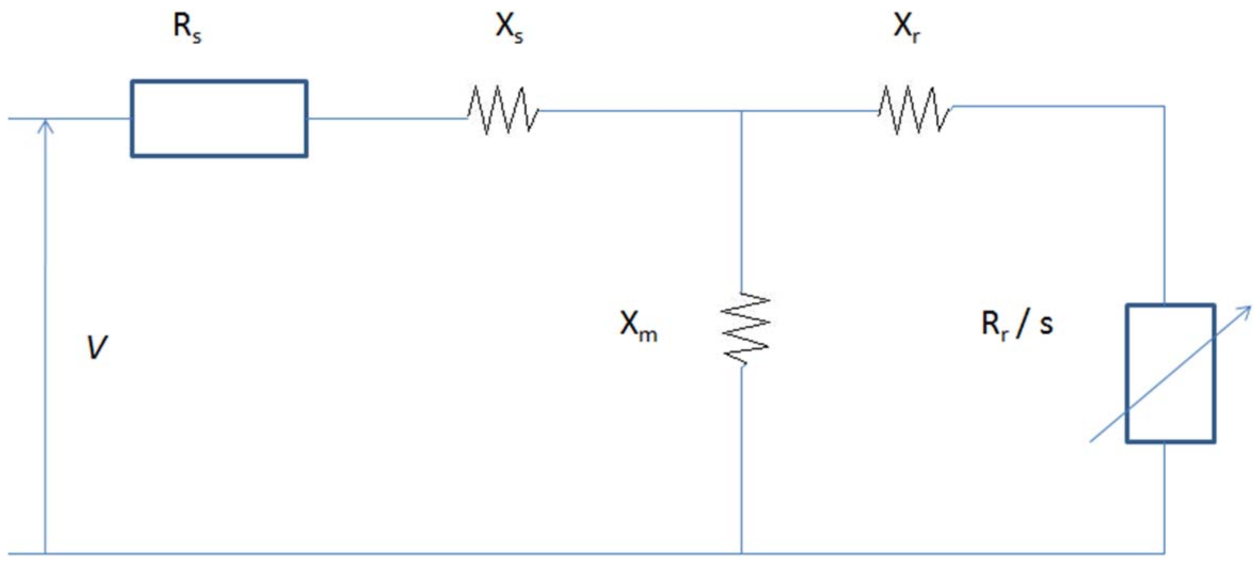


Figure 2.2 Induction motor model [12]

The variables for the induction motor model are given below.

Rs : stator resistance

Xs: stator leakage reactance

Xm: magnetizing reactance

Xr: rotor leakage reactance

Rr: rotor resistance

S = (ws-w)/ws : rotor slip

According to [12], dynamic load can be displayed as a combination of ZIP load model and induction motor model. Aggregate load models are frequently used, and are the combinations of a static load model and dynamic load model. The four aggregate load model structures are as below.

- ZIP augmented with induction motor
- ZIP augmented with another type of equation (second order)

- Exponential augmented with induction motor (three-state)
- Exponential augmented with another type of equation (second order)

There are many dynamic load models. For a specific type of measurement data, one model structure will be more preferred than the other one. Whichever model that gives the best results will also be the best model structure for the specific type of measurements. In this thesis, the ZIP static model augmented with an induction motor model is used for the load model structure. The frequency part is ignored in here, and the aggregate dynamic load model is shown below [12].

$$P_s = \gamma_1 * P_{ZIP} + \lambda_1 * P_{MOT} \tag{2.11}$$

$$Q_s = \gamma_2 * Q_{ZIP} + \lambda_2 * Q_{MOT} \tag{2.12}$$

The P_s, P_{ZIP}, P_{MOT} are the real power loads that are aggregated. Now, γ and δ are the percentages of static and dynamic load. The more extensive representation of the induction motor model can be shown in equations (2.13) through (2.18) below [12].

$$\frac{dE_d^{'}}{dt} = -\frac{1}{T^{'}}\left[E_d^{'} + (X - X')I_q\right] - (\omega - 1)E_q^{'} \tag{2.13}$$

$$\frac{dE_q^{'}}{dt} = -\frac{1}{T^{'}}\left[E_q^{'} - (X - X')I_d\right] + (\omega - 1)E_d^{'} \tag{2.14}$$

$$\frac{d\omega}{dt} = -\frac{1}{2H}\left[T_0(A\omega^2 + B\omega + C + D\omega^E) - (E_d^{'}I_d + E_q^{'}I_q)\right] \tag{2.15}$$

$$I_d = \frac{1}{R_s^2 + X'^2}\left[R_s(U_d - E_d^{'}) + X^{'}(U_q - E_q^{'})\right] \tag{2.16}$$

$$I_q = \frac{1}{R_s^2 + X'^2}\left[R_s(U_q - E_q^{'}) - X^{'}(U_d - E_d^{'})\right] \tag{2.17}$$

$$T_L = T_0(A\omega^2 + B\omega + C + D\omega^E) \tag{2.18}$$

In these equations X' is equivalent to X_s+((X_m)(X_r))/(X_m+X_r), and it is the transient reactance or the blocked-rotor (short-circuit) reactance. X or X_s+X_m is the open circuit reactance or the motor no-load reactance. The rest of the variables are listed below.

H: rotor inertia constant

T_L: load torque equation

T_0: steady state mechanical torque

ω: rotor rotation speed

E'_d: d-axis transient EMF of motor

E'_q: q-axis transient EMF of motor

U_d: d-axis bus voltage

U_q: q-axis bus voltage

I_d: d-axis stator current

I_q: q-axis stator current

The A,B,C,D,E coefficients in load torque equation need to satisfy

$$C = 1 - A\omega_0^2 - B\omega_0 - D\omega_0^E$$

To better understand the system, we also need to find the percentages of each individual load in the total load. The ZIP load model equations (2.2) and (2.3) have the parameters p_1, p_2, p_3, q_1, q_2, q_3. From the load composition equations (2.11) and (2.12), the parameters are α_1, α_2, β_1, β_2. Lastly, the parameters from the induction motor model are R_s, X_s, R_r, X_r, X_m, H, A, B, C, D, and E. Finding the load parameters will give us the best fit between the measurement data and the actual load model outputs. This work will be the main focus of this thesis. The intelligent techniques were applied to find these parameters that will give the smallest mean square errors between the measured data and the actual data.

2.6 The State of Art of Load Modeling

2.6.1 Traditional and Black Box Approaches

The traditional approach used manual calculation to find the best equations or physical structure to represent the relations between the output and input data. This approach can sometimes be quite efficient for small sized power systems and can provide high accuracy. However, for more complex systems and dynamic load systems, this approach is unreliable. In order to solve more complicated power systems, the black-box approach has been often used [10]. Due to the complexity of the system, several intelligence techniques were used to predict the output based from the input without representing the model in equation format. This method can be used to merely predict more complicated systems, but it does not allow the readers to really understand the power system loads nor does it provide the parameters for use in other applications. In [14], the writers specified that for general non-linear curve fitting at a precision problem, back propagation artificial neural network (BP ANN) was a good choice. However, one of the disadvantages of the BP ANN load model was that it lacked sufficient feedback. A more advanced method was needed. Hence, for dynamic load system, more complicated ANNs such as Elman Artificial Neural Network or Hopfield network were used to solve the time dependant dynamic system. Using field tests to determine the load model parameters indicated that the ULTC's and feeder voltage regulators are the main source of load recovery dynamics [14]. In [16], the least square method was sufficient to find the closest curve fit for the power system load. In paper [16], other practical tests have shown that for an overly conservative load model, inaccurate data will be an issue for the system.

As stated in paper [17], a general ANN model included voltages at the input, bias and weights at each neuron, with the output as power. One or more feedback signals went back to the input as well to form a closed loop system. This model can accurately provide the characteristics of the load dynamics. The ANN approximation will allow them to find the error bounds and justify the choice of functions with smaller errors. In [18], we can see that for the Matlab neural network test, it was concluded that the training algorithm was the most important factor in the accuracy and performance of the network. The performance for each training algorithm also depended on the size of the power system network.

In [19], neural network fuzzy dynamic programming approach was another method used. The procedures can be described as below.

- The input signal is first fed to the artificial neural network.
- Second step is to go through the ANN to construct the fuzzy dynamic programming rules. It also is helpful to include current status of capacitors and tap positions as ANN inputs.
- Then solve using the rules for the output. The combined approach was efficient as it only took a little time for ANN to reach the targeted solution.

As stated in the article [19], in order to improve the performance when dynamic power systems are designed and maintained with less stability margin, power system engineers need to consider the accuracy of loads and data and proper load models for the distribution equipment. In paper [20], fuzzy inference was suggested by the researchers to solve dynamic load at the transient case. Adaptive-network-based fuzzy inference system or ANFIS was developed to construct the basis of fuzzy if-then rules for combining learning the rules of adaptive network to model the no-linear system performance. The results show that the dynamic load modeling ANFIS structure emulated the P and Q load response with input voltage and frequency fairly well.

Another approach as shown in article [21] said that the node-load model by the application of an indistinct fuzzy logic approach allowed the modeling of a disturbance during transient conditions and also accounted for the structure of node-loads that were not completely clear.

2.6.2 Parameter Identification Approach

Parameter identification approach is an essential technology in measurement-based load modeling. According to [3, 22, 23, 24, and 25], precise load modeling can avoid any miscalculation or wrong operations. A contrary conclusion can be caused by inappropriate types of load models [26, and 27]. However, in article [24], the author declared that if a "standard" load model was used then the load model would not recreate the unstable behavior for a fault despite of reconstruction. Today, as reported by Southern California Edison and Florida Power & Light Company in article [28], load modeling has improved because of an increase in air conditioner load in some areas. Air conditioner load sometimes would cause short-term voltage instability, quick voltage collapse, and also would slow down voltage recovery. Although there is a lot of literature related to finding the load model parameters as stated in paper [29], to get the accurate load model from these parameters can be difficult. Several problems could be shown below:

- Time variance and stochastic variables are associated with loads
- Some issues with aggregate loads
    - Massive diverse load components

    - New load components getting into the system
    - Not enough composition information for certain loads
    - Insufficient details to show various load components
- Not enough actual measurements to verify the load models

We can see there are numerous problems with the load models, and an improved load model has become a challenge for power system analysis and control. The state-of-the-art techniques for load modeling should be taken advantage of to develop a systematic approach to represent the aggregate load for power system stability study purposes.

As discussed before, the load modeling methods are categorized by the component-based approach and the measurement-based approach. The latter one will be more real-time and more associated with the dynamic characteristics [30]. Measurement-based approaches are based on system identification and are more applicable to linear models. For nonlinear models, it could become more challenging [31]. Analytical-based approaches have been used to derive parameters in 1977 according to test results. Field tests were recorded and parameters of a simplified induction motor load model were solved [32]. A step/staged/controlled test could use this method although measurement error will be detected fairly easily.

On the other hand, an optimization-based approach can be used to find the best parameters that minimize an error function between the measured data and the simulated ones. Many of these approaches have been done to find the best parameter estimation. For instance, as shown in paper [12], the search algorithms are:

- Search algorithms with statistical techniques
    - Least square-based parameter estimation that involved induction motor models [33].
    - Weighted least square-based estimation with Unequal Square [34].
    - Instrumental variable-based estimation that minimizes the sum of absolute residues [35].
    - Maximum likelihood-based using a probability density function [36].
    - Gradient-based parameter estimation [37].
- Search algorithms with heuristic techniques

- Simulated annealing-based parameter estimation used adaptive simulated annealing [38].
- Neural network-based parameter estimation used measurements to train and update the load model continuously [17].
- Genetic algorithm-based parameter estimation [39].

Other methods include:

- Multistage algorithm for load parameter identifications [40].
- Nonlinear parameters calculated directly from the linear identification results [41].

Lastly, the stochastic based approach also was used and this approach makes use of the error function and therefore is more confined [12].

2.7 Software and Power System Model

As described in previous chapters, static models are commonly used in today's research. The models can be used in industries to predict dynamic behaviors of active/reactive loads. In this research, load bus 3 in a 16-generator power system from a Matlab coded power systems toolbox (by Cherry Tree software [42]) was used. The load bus voltage, active and reactive powers were used as the key information in this research. The 16-generator and 52-bus system is shown in the figure below.

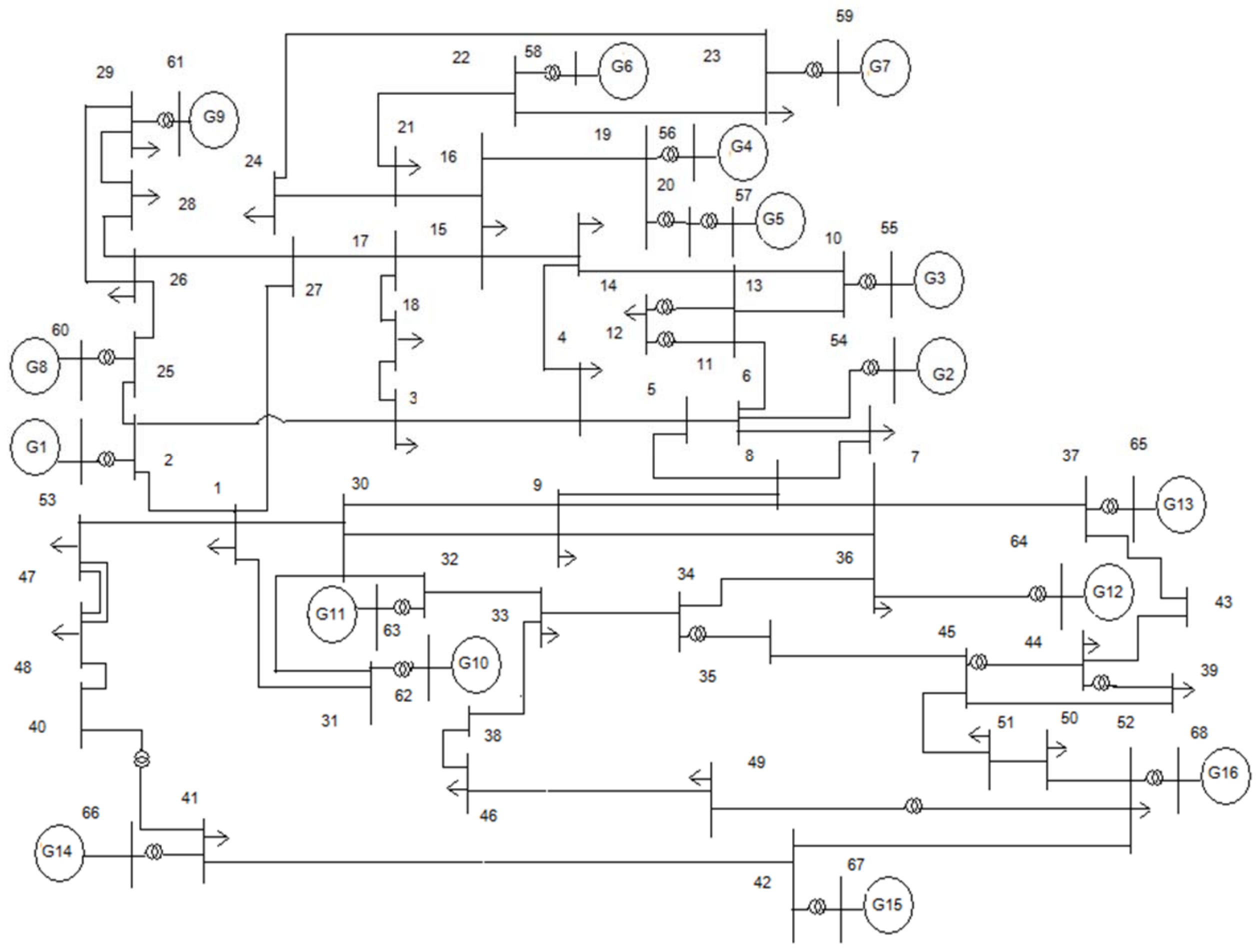


Figure 2.3 16-generator and 52-bus system used in the research

For this research, Matlab was used extensively. It helped to alleviate the amount of matrix calculations needed for the research. Calculations involved data analyzing, accumulating, and storing. As the Power System Toolbox was coded in Matlab, therefore Matlab software is a good choice for this research. The Microsoft Excel program was also utilized in the research as well. This software application helped in calculating large amounts of data as well as displaying the data in an organized manner. Plots and graphs can be generated fairly readily by this software as well. In this chapter, the first part focuses on a small simple power system. We used simple spreadsheet calculations and manipulation to show that the constant power, constant current, and constant impedance parameters can be obtained and verified using Microsoft Excel. In the second part, curve fitting and parameter finding by matrix manipulation was used. As some of the matrices are seen as over-determined systems, methods such as pseudo-inverse were used.

2.8 Problem Statement

The interest in this thesis is to develop methods to readily find the necessary parameters to represent the load model using "measurement" data from a power system load. The work utilized the static ZIP load model, as shown in equations (2.2) and (2.3) before and rewritten below for convenience.

$$P_{ZIP} = P_0[p_1\overline{v}^2 + p_2\overline{v} + p_3]$$

$$Q_{ZIP} = Q_0[q_1\overline{v}^2 + q_2\overline{v} + q_3]$$

We notice that P_0 and Q_0 are the pre-disturbed active and reactive powers. The representations for p_1 to p_3 and q_1 to q_3 can be summarized in the Table 2.1 below.

Table 2.1The parameter representation for ZIP Load Model

Variable	Representation
p_1	Percentage of constant real Impedance or Admittance
p_2	Percentage of constant real current
p_3	Percentage of constant active power
q_1	Percentage of constant reactive Impedance or Admittance
q_2	Percentage of constant reactive current
q_3	Percentage of constant reactive power

The goal of this work is to find these p_1, p_2, p_3, q_1, q_2, and q_3 variables for the load model. As mentioned before, a simple power system and the 16-genetaor power system from the Matlab power system tool box were used, and these parameters were identified. Their voltage, real powers and reactive powers were used as measurements during transient simulation to assist in finding the parameters. In a more complex power system where 339 time steps were run, an over determined matrix would be formed. To demonstrate this matrix, we re-arrange the equations (2.2) and (2.3), and get:

$$P_{norm} = \frac{P_{ZIP}}{P_0} = p_1\overline{v}^2 + p_2\overline{v} + p_3 \qquad \textbf{(2.19)}$$

$$Q_{norm} = \frac{Q_{ZIP}}{Q_0} = q_1\bar{v}^2 + q_2\bar{v} + q_3 \qquad \textbf{(2.20)}$$

As we finished running the transient simulation, data for load voltage, active, and reactive powers were obtained at different time steps. Let us represent these values as v_1, v_2, v_3, v_4, $P_{norm}\,1$, $P_{norm}\,2$, $P_{norm}\,3$, $Q_{norm}\,1$, $Q_{norm}\,2$, $Q_{norm}\,3$ and etc., for their respective value at different time steps. As we re-arrange the equations, the matrices would be:

$$\begin{bmatrix} v_1^2 & v_1 & 1 \\ v_2^2 & v_2 & 1 \\ v_3^2 & v_3 & 1 \\ \dots & \dots & \dots \\ v_n^2 & v_n & 1 \end{bmatrix} \begin{bmatrix} p_1 \\ p_2 \\ p_3 \end{bmatrix} = \begin{bmatrix} \mathrm{P}_{\mathrm{norm}\,1} \\ \mathrm{P}_{\mathrm{norm}\,2} \\ \mathrm{P}_{\mathrm{norm}\,3} \\ \dots \\ \mathrm{P}_{\mathrm{norm}\,n} \end{bmatrix} \qquad \textbf{(2.21)}$$

$$\begin{bmatrix} v_1^2 & v_1 & 1 \\ v_2^2 & v_2 & 1 \\ v_3^2 & v_3 & 1 \\ \dots & \dots & \dots \\ v_n^2 & v_n & 1 \end{bmatrix} \begin{bmatrix} q_1 \\ q_2 \\ q_3 \end{bmatrix} = \begin{bmatrix} \mathrm{Q}_{\mathrm{norm}\,1} \\ \mathrm{Q}_{\mathrm{norm}\,2} \\ \mathrm{Q}_{\mathrm{norm}\,3} \\ \dots \\ \mathrm{Q}_{\mathrm{norm}\,n} \end{bmatrix} \qquad \textbf{(2.22)}$$

These two matrices above are over-determined systems and the parameters would not be 100 percent accurately solved. Moreover, the sum of both active and reactive power parameters is 1. This is because the total amount of active or reactive power should be 1 or 100% as below:

$$p_1 + p_2 + p_3 = 1 \qquad \textbf{(2.23)}$$

$$q_1 + q_2 + q_3 = 1 \qquad \textbf{(2.24)}$$

CHAPTER 3 - PARAMETER IDENTIFICATION APPROACH USING MANUAL CALCULATIONS AND MATRIX MANIPULATION METHODS

3.1 Mathematical Manipulation and Microsoft Excel Approaches

In this problem, we focused on building a small power system load with real and reactive loads in parallel. Any impedance or even combinations of impedance and admittance could be converted to the format below. The small power system load is displayed below in Figure 3.1.

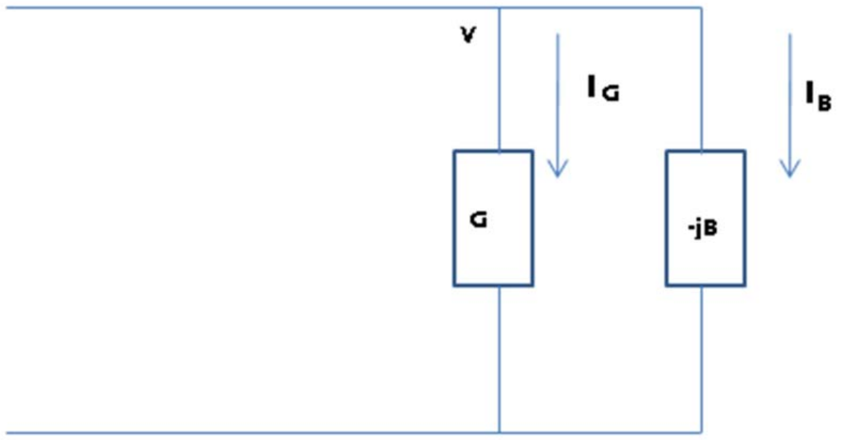


Figure 3.1 Simple Power System Load Used for Study

The parameters were calculated based on this simple schematic diagram to establish and verify the simple relations between the power system loads values. These simple concepts and insights would be carried over and be used to construct models of loads in more complicated power systems later on.

As seen in Figure 3.1, the admittance or the conductance, G, and susceptance, B, were used in parallel instead of impedance of resistance and reactance in series. This modification was needed to better understand the load voltage and the corresponding load powers. In fact, in order to convert the Impedance (Z) to an admittance (Y) form, we would do the following:

$$Y = Z^{-1} = 1/(R + jX) = G - jB \qquad \textbf{(3.1)}$$

To better understand the conductance (G) and the susceptance (B), we convert the resistance and reactance as shown below.

$$G = R/(R^2 + X^2) = R/|Z|^2 \tag{3.2}$$

$$B = X/(R^2 + X^2) = X/|Z|^2 \tag{3.3}$$

$$Y = 1/(R + jX) = \frac{R - jX}{R^2 + X^2} = \frac{R}{R^2 + X^2} - j\frac{X}{R^2 + X^2} = G - jB \tag{3.4}$$

Notice that to simplify the whole mathematical manipulation, we assume V to be the reference voltage with a zero angle.

3.1.1 Constant Impedance or Admittance

Following the Figure 3.1, for the constant impedance case, we could find the relation between powers and square of voltages as shown below.

$$\begin{aligned}
S &= VI^* = V\left(\frac{V}{Z}\right)^* \\
&= \frac{V.V^*}{Z^*} \\
&= \frac{|V|^2}{Z^*} \\
&= \frac{|V|^2}{\left(\frac{1}{Y}\right)^*} \\
&= |V|^2\left(Y^*\right) \qquad\qquad (3.5) \\
&= |V|^2\left(G - jB\right)^* \\
&= |V|^2\left(G + jB\right) \\
&= |V|^2 G + j|V|^2 B \\
&= P + jQ
\end{aligned}$$

Therefore, we conclude that:

$$P = |V|^2 G \tag{3.6}$$

$$Q = |V|^2 B \tag{3.7}$$

Looking back at Equations (2.2) and (2.3), we see that if the load has constant impedance, $p_2=p_3=0$ and $p_1=1$. As we substitute these values in, the simplified equations are as below. Note they have similar forms as equations (3.6) and (3.7) above.

$$P_{ZIP} = P_0 \bar{v}^2 \tag{3.8}$$

$$Q_{ZIP} = Q_0 \bar{v}^2 \tag{3.9}$$

3.1.2 Constant Current

Similarly, for the constant current problem, we look for the relationship between powers and real and reactive currents. First, we define conductance current and susceptance current in Equations (3.10) and (3.11) respectively as below.

$$I_G = VG \tag{3.10}$$

$$I_B = -jVB \tag{3.11}$$

After defining these equations, we solve the complex power as below.

$$\begin{aligned} S &= VI^* = V(I_G + I_B)^* = V(VG - jVB)^* \\ S &= V\left[VG + jVB\right] \\ S &= V\left(I_G - I_B\right) \\ P + jQ &= V\left(I_G - I_B\right) \end{aligned} \tag{3.12}$$

P and Q can be written separately as in equations (3.13) and (3.24).

$$P = VI_G \tag{3.13}$$

$$Q = \frac{-VI_B}{j} = jVI_B \tag{3.14}$$

Looking back at equations (2.2) and (2.3), we see that if the load has constant current, $p_1=p_3=0$ and $p_2=1$. As we substitute these values into the ZIP load equation, the equations are simplified as below with a similar structure as the equations above. However, we note that only the magnitude for the reactive power Q is used.

$$P_{ZIP} = P_0 \bar{v} \tag{3.15}$$

$$Q_{ZIP} = Q_0 \bar{v} \tag{3.16}$$

3.1.3 Constant Power

For constant power case, P and Q can be perceived fairly readily. Because P and Q need to be constant, they will be equivalent to the pre-disturbed values of P_0 and Q_0. These values for P and Q are not changed throughout the entire operation as shown in (3.17) and (3.18).

$$P = P_0 \tag{3.17}$$

$$Q = Q_0 \tag{3.18}$$

From Equations (2.2) and (2.3), if we have constant power, then $p_1=p_2=0$ and $p_3=1$. As we submit these values in, the equations are simplified as below in (3.19) and (3.20) which are equivalent to the equations above.

$$P_{ZIP} = P_0 \tag{3.19}$$

$$Q_{ZIP} = Q_0 \tag{3.20}$$

3.1.4 Microsoft Excel Approach

Microsoft Excel was used as an extended approach to verify the credibility of ZIP load equations. The main goal is to prove that ZIP load Equations (2.2) and (2.3) would still hold when a stream of voltage data was fed into the simple power system in Figure 3.1.

Several key points need to be taken into consideration as outlined below.

- The constant impedance, constant current, and constant power equations are used for calculating important values on the load such as the equations in (3.6), (3.7), (3.13), (3.14), (3.17), and (3.18).
- A set of changing voltages are fed into the simple power system in Figure 3.1, and their respective active, reactive powers were generated from these equations.
- In order to prove the credibility of ZIP load model, the P_0 and Q_0 values should be constant during the calculations as this is also the assumption in ZIP load equations.

Now, to first construct the model, we define the admittance Y from the simple power system to be Y=2.68+j28 per unit. This can also be expressed as:

B=2.68 per unit

G=28 per unit

The sections below calculate the P_0 and Q_0 and use the changing voltage from 0.95 to 1.08 per unit as the independent variable in the system.

3.1.4.1 Constant Impedance

Let us work with the constant impedance case first. The table for constant impedance or admittance below shows the injected voltages to be from 0.95 to 1.08 per unit. The B and G values would stay constant. The calculations for obtaining P, Q, P_0, and Q_0 are also shown in the first row of the Table 3.1 below.

Table 3.1 Constant impedance load calculations

$Q=V^2*B$	$P=V^2*G$		$P_0=P/(V^2)$	$Q_0=Q/(V^2)$			
Q	P	V	P_0	Q_0	V^2	Q/Q_0	P/P_0
25.27	2.4187	0.95	2.68	28	0.9025	0.9025	0.9025
25.8048	2.46989	0.96	2.68	28	0.9216	0.9216	0.9216
26.3452	2.52161	0.97	2.68	28	0.9409	0.9409	0.9409
26.8912	2.57387	0.98	2.68	28	0.9604	0.9604	0.9604
27.4428	2.62667	0.99	2.68	28	0.9801	0.9801	0.9801
28	2.68	1	2.68	28	1	1	1
28.5628	2.73387	1.01	2.68	28	1.0201	1.0201	1.0201
29.1312	2.78827	1.02	2.68	28	1.0404	1.0404	1.0404
29.7052	2.84321	1.03	2.68	28	1.0609	1.0609	1.0609
30.2848	2.89869	1.04	2.68	28	1.0816	1.0816	1.0816
30.87	2.9547	1.05	2.68	28	1.1025	1.1025	1.1025
31.4608	3.01125	1.06	2.68	28	1.1236	1.1236	1.1236
32.0572	3.06833	1.07	2.68	28	1.1449	1.1449	1.1449
32.6592	3.12595	1.08	2.68	28	1.1664	1.1664	1.1664

We notice that P_0, and Q_0 are constant throughout the calculation. Therefore, these calculations portray consistent characteristics with the ZIP load equations (P_0 and Q_0 unchanged). Also, the parameters $p_2=p_3=0$, and $p_1=1$, so the ZIP load equation will generate the same results. Notice that, as described from previous section, the power factor will also affect the stability of the system. To simplify the work, we avoid the power factor difference between the active and reactive powers. Therefore, we assume that the parameters for $q_1 = p_1$, $q_2 = p_2$, and $q_3 = p_3$ when we talk about constant impedance, constant current, constant power, or mixed load. This entire thesis generally follows this rule.

3.1.4.2 Constant Current

Similarly, for constant current, we calculate the constant currents as:

I_G = V*G = 1*2.68 = 2.68 per unit

I_B = -V*B = -1*28 = -28 per unit (where only the magnitude of the current were used)

By following the calculation on the first row of the Table 3.2 below, we would have the values shown.

Table 3.2 Constant current load calculations

Q = | V*I_B |	P = V*I_G		P_0=P/V	Q_0=Q/V			
Q	P	V	P0	Q0	V^2	Q/Q0	P/P0
26.6	2.546	0.95	2.68	28	0.9025	0.95	0.95
26.88	2.5728	0.96	2.68	28	0.9216	0.96	0.96
27.16	2.5996	0.97	2.68	28	0.9409	0.97	0.97
27.44	2.6264	0.98	2.68	28	0.9604	0.98	0.98
27.72	2.6532	0.99	2.68	28	0.9801	0.99	0.99
28	2.68	1	2.68	28	1	1	1
28.28	2.7068	1.01	2.68	28	1.0201	1.01	1.01
28.56	2.7336	1.02	2.68	28	1.0404	1.02	1.02
28.84	2.7604	1.03	2.68	28	1.0609	1.03	1.03
29.12	2.7872	1.04	2.68	28	1.0816	1.04	1.04
29.4	2.814	1.05	2.68	28	1.1025	1.05	1.05
29.68	2.8408	1.06	2.68	28	1.1236	1.06	1.06
29.96	2.8676	1.07	2.68	28	1.1449	1.07	1.07
30.24	2.8944	1.08	2.68	28	1.1664	1.08	1.08

As we can see, the P_0, and Q_0 are constant throughout the calculation. These calculations are consistent with the ZIP load equations as the values P_0 and Q_0 do not change. Since p_1=p_3=0, and p_2=1, the ZIP load equation will yield the same conclusion.

3.1.4.3 Constant Power

For the constant power case, the P and Q need to be constant. Since P_0 and Q_0 ideally should be the same number. These two values were consistent in the last two cases before, we will start using the same number P_0 = 2.68 per unit and Q_0 = 28 per unit initially to be consistent with the last two sections. From the Equations (3.17) and (3.18), we would get initial P and Q to be:

$P = P_0$ =2.68 per unit

$Q = Q_0$ =28 per unit

Therefore the first P and Q equal to 2.68 per unit and 28 per unit respectively initially.
Also, by reversing the Equations (3.21) and (3.22) above, we get:

$$P_0 = P \tag{3.21}$$

$$Q_0 = Q \tag{3.22}$$

As the P and Q stay the same, therefore from Equations (3.21) and (3.22) above, all the rest of P_0 and Q_0 will also be equal to 2.68 per unit and 28 per unit respectively. Therefore P_0 and Q_0 will still be consistent throughout the constant power calculations. Table 3.3 summarizes the results below.

Table 3.3 Constant power load calculations

Q (constant)	P (constant)		P_0=P	Q_0=Q			
Q	P	V	P0	Q0	V^2	Q/Q0	P/P0
28	2.68	0.95	2.68	28	0.9025	1	1
28	2.68	0.96	2.68	28	0.9216	1	1
28	2.68	0.97	2.68	28	0.9409	1	1
28	2.68	0.98	2.68	28	0.9604	1	1
28	2.68	0.99	2.68	28	0.9801	1	1
28	2.68	1	2.68	28	1	1	1
28	2.68	1.01	2.68	28	1.0201	1	1
28	2.68	1.02	2.68	28	1.0404	1	1
28	2.68	1.03	2.68	28	1.0609	1	1
28	2.68	1.04	2.68	28	1.0816	1	1
28	2.68	1.05	2.68	28	1.1025	1	1
28	2.68	1.06	2.68	28	1.1236	1	1
28	2.68	1.07	2.68	28	1.1449	1	1
28	2.68	1.08	2.68	28	1.1664	1	1

As the P_0 and Q_0 stay constant, they are consistent with the ZIP load equations as the values P_0 and Q_0 do not change. Since p_1=p_2=0, and p_3=1, the ZIP load equation will perform the same calculations as shown in the table above.

3.1.4.4 Equally Distributed Load

Moving forward, we look at a new load that has 1/3 constant impedance, 1/3 constant current, and 1/3 constant power components. The voltage here still goes from 0.95 to 1.08 per unit. The P and Q values from previous 3 sections were used in the calculations here. The calculations for P and Q are shown below.

$$P_{EDL}=0.3333*(P_{CI}+P_{CC}+P_{CP}) \tag{3.23}$$

$$Q_{EDL}=0.3333*(Q_{CI}+Q_{CC}+Q_{CP}) \tag{3.24}$$

The representations can be summarized below:

- EDL refers to the value for equally distributed load
- CI refers to constant impedance

- CC refers to constant current
- Finally CP refers to constant power

For instance, for voltage at 0.95 per unit, the P_{EDL} and Q_{EDL} were obtained as below.

P_{EDL}=0.3333*(P_{CI}+P_{CC}+P_{CP}) = 0.3333 * (2.4187+2.546+2.68) = 2.54798

Q_{EDL}=0.3333*(QP_{CI}+Q_{CC}+Q_{CP}) = 0.3333 * (25.27+26.6+28) = 26.6200671

Since we do not have the manual calculations for a mixed load, we use the ZIP load equations to get the values for P_0 and Q_0. As this is the case where the load components were evenly distributed, parameters p_1=p_2=p_3=q_1=q_2=q_3=0.3333. Hence the zip load equations are as below.

$$p_{ZIP} = p_0[0.3333\bar{v}^2 + 0.3333\bar{v} + 0.3333]$$

$$Q_{ZIP} = Q_0[0.3333\bar{v}^2 + 0.3333\bar{v} + 0.3333]$$

Solve for P_0 and Q_0, we get:

$$p_0 = \frac{p_{ZIP}}{0.3333\bar{v}^2 + 0.3333\bar{v} + 0.3333} = \frac{p_{ZIP}}{0.3333(\bar{v}^2 + \bar{v} + 1)}$$

$$Q_0 = \frac{Q_{ZIP}}{0.3333\bar{v}^2 + 0.3333\bar{v} + 0.3333} = \frac{Q_{ZIP}}{0.3333(\bar{v}^2 + \bar{v} + 1)}$$

Again, we solve the first case when V=0.95 per unit.

$$p_0 = \frac{2.54798}{0.3333[(0.95)^2 + 0.95 + 1]} = 2.68 \text{ per unit}$$

$$Q_0 = \frac{26.620671}{0.3333[(0.95)^2 + 0.95 + 1]} = 28 \text{ per unit}$$

These values are the values for constant power, constant impedance, and constant current loads. The other instances of P_{EDL} and Q_{EDL}, P_0 and Q_0 can be obtained as shown in the Table 3.4 below. We observe that the P_0 and Q_0 stay the same at 2.68 and 28 per units respectively which are consistent like the other sections.

Table 3.4 Equally distributed load calculations

Q	P	V	V^2	P_0	Q_0	P/P_0	Q/Q_0
26.620671	2.54798	0.95	0.9025	2.68	28	0.95073825	0.95073825
26.8922438	2.57397	0.96	0.9216	2.68	28	0.96043728	0.96043728
27.1656832	2.60014	0.97	0.9409	2.68	28	0.97020297	0.97020297
27.440989	2.62649	0.98	0.9604	2.68	28	0.98003532	0.98003532
27.7181612	2.65302	0.99	0.9801	2.68	28	0.98993433	0.98993433
27.9972	2.67973	1	1	2.68	28	0.9999	0.9999
28.2781052	2.70662	1.01	1.0201	2.68	28	1.00993233	1.00993233
28.560877	2.73368	1.02	1.0404	2.68	28	1.02003132	1.02003132
28.8455152	2.76093	1.03	1.0609	2.68	28	1.03019697	1.03019697
29.1320198	2.78835	1.04	1.0816	2.68	28	1.04042928	1.04042928
29.420391	2.81595	1.05	1.1025	2.68	28	1.05072825	1.05072825
29.7106286	2.84373	1.06	1.1236	2.68	28	1.06109388	1.06109388
30.0027328	2.87169	1.07	1.1449	2.68	28	1.07152617	1.07152617
30.2967034	2.89983	1.08	1.1664	2.68	28	1.08202512	1.08202512

3.1.4.5 Mixed Load

Similarly to the last section, we would have a mixed load that contains 20% constant impedance, 30% constant current, and 50% constant power. The P and Q values can be obtained as below.

P_{MIXL}=0.2*P_{CI}+0.3*P_{CC}+0.5*P_{CP}

Q_{MIXL}=0.2*Q_{CI}+0.3*Q_{CC}+0.5*Q_{CP}

In here, MIXL refers to mixed load.

Similar to the last section, we will do a demonstration by calculating the first case where the voltage is at 0.95 per unit.

P_{MIXL}=0.2*2.4187+0.3*2.546+0.5*2.68 = 2.58754

Q_{MIXL}=0.2*25.27+0.3*26.6+0.5*28 = 27.034

Similar to the last example, the ZIP load equations (2.2) and (2.3) are used to get P_0 and Q_0. In here, p_1=q_1=0.2, p_2=q_2=0.3, and p_3=q_3=0.5. After this modification, the ZIP load equations become:

$$P_{ZIP} = P_0[0.2\overline{v}^2 + 0.3\overline{v} + 0.5]$$

$$Q_{ZIP} = Q_0[0.2\overline{v}^2 + 0.3\overline{v} + 0.5]$$

Solve for P_0 and $Q_{0,}$ we get:

$$P_0 = \frac{P_{ZIP}}{[0.2\overline{v}^2 + 0.3\overline{v} + 0.5]}$$

$$Q_0 = \frac{Q_{ZIP}}{[0.2\overline{v}^2 + 0.3\overline{v} + 0.5]}$$

Again, we solve the first case when V=0.95 per unit.

$$P_0 = \frac{2.58754}{[0.2*(0.95)^2 + 0.3*(0.95) + 0.5]} = 2.68 \text{ per unit}$$

$$Q_0 = \frac{27.034}{[0.2*(0.95)^2 + 0.3*(0.95) + 0.5]} = 28 \text{ per unit}$$

Like the last section, P_0 and Q_0 stay the same at 2.68 and 28 per units respectively. The rest of P_0 and Q_0 are also constant in Table 3.5 below. Therefore, we conclude that the ZIP load equations as we have seen, work fairly well with the small power system load in Figure 3.1.

Table 3.5 Mixed load calculations

Q	P	V	V^2	P_0	Q_0	P/P_0	Q/Q_0
27.034	2.58754	0.95	0.9025	2.68	28	0.9655	0.9655
27.22496	2.60582	0.96	0.9216	2.68	28	0.97232	0.97232
27.41704	2.6242	0.97	0.9409	2.68	28	0.97918	0.97918
27.61024	2.64269	0.98	0.9604	2.68	28	0.98608	0.98608
27.80456	2.66129	0.99	0.9801	2.68	28	0.99302	0.99302
28	2.68	1	1	2.68	28	1	1
28.19656	2.69881	1.01	1.0201	2.68	28	1.00702	1.00702
28.39424	2.71773	1.02	1.0404	2.68	28	1.01408	1.01408
28.59304	2.73676	1.03	1.0609	2.68	28	1.02118	1.02118
28.79296	2.7559	1.04	1.0816	2.68	28	1.02832	1.02832
28.994	2.77514	1.05	1.1025	2.68	28	1.0355	1.0355
29.19616	2.79449	1.06	1.1236	2.68	28	1.04272	1.04272
29.39944	2.81395	1.07	1.1449	2.68	28	1.04998	1.04998
29.60384	2.83351	1.08	1.1664	2.68	28	1.05728	1.05728

3.2 Curve Fitting using Inverse Matrix and Pseudo Inverse Approaches

After the study of the simple power system load as in Figure 3.1, we move on to a bigger power system. As discussed in previous sections, a 16-generator and 52-bus power system in Figure 2.3 was used for the study.

In a real power system, after a disturbance, the system tries to recover for itself. If parts of the system fail to recover to its pre-disturbed conditions, then a part of the system will collapse and might even trigger the surrounding power system to collapse. Hence, a blackout takes place eventually in cases like this. The 16-genetator system was chosen so a more realistic system can be studied. In this section, load Bus 3 was where the load under study was located.

As the 16-generator system is coded in Matlab in power system toolbox, various changes could be adjusted in the program to place the disturbance (fault) at the desired load bus. In this case, the system trips at the 3-phase fault between the lines connecting between Bus 3 and Bus 18. A 5-second interval was observed during the simulation after the fault. Consequently, the fault occurs at 0.1 second and the clearing times are 0.35 second and 0.4 second for near end fault and far end fault, respectively. The load bus voltage magnitude varies during this disturbance and fights back to the pre-disturbance value. Figure 3.2 below displays the voltage magnitude during the fault.

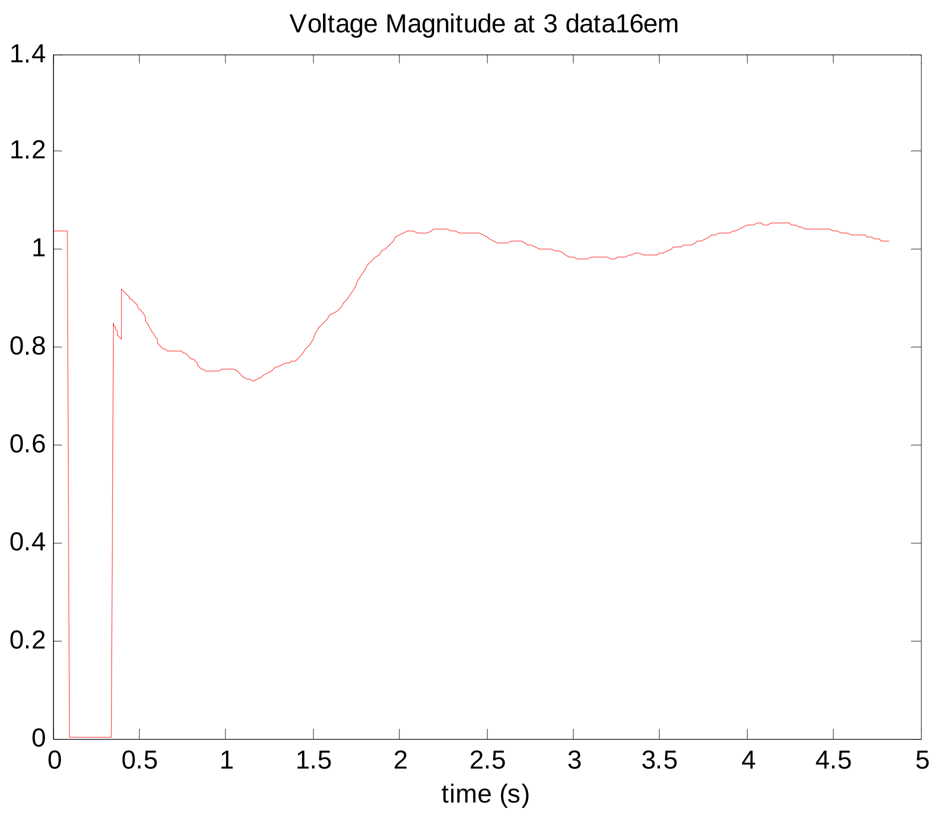


Figure 3.2 Voltage magnitude of Bus 3 during the fault

The real and reactive powers at load Bus 3 were calculated from the power flows that surround this bus. For instance, while we zoom in on the 16-generator diagram in Figure 2.3, Bus 3 can be show as below in Figure 3.3.

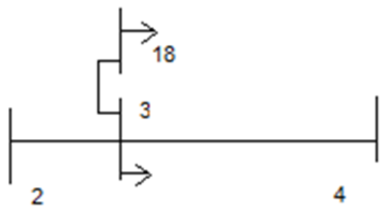


Figure 3.3 Load Bus 3

The load power and reactive power were calculated from the power flows surround it.

For instance, according to the diagram, the load active and reactive powers at Bus 3 are:

$P_{load3} = -(P_{3-2} + P_{3-4} + P_{3-18})$ **(3.25)**

$Q_{load3} = -(Q_{3-2} + Q_{3-4} + Q_{3-18})$ **(3.26)**

As we run the simulation, the total running time is 5 seconds and there are 339 time steps during this interval. Hence, we have 339 different voltages, active, and reactive powers. From matrix Equations (2.18) and (2.19), we can construct over-determined matrix equations. Recall these equations below.

$$\begin{bmatrix} v_1^2 & v_1 & 1 \\ v_2^2 & v_2 & 1 \\ v_3^2 & v_3 & 1 \\ \cdots & \cdots & \cdots \\ v_n^2 & v_n & 1 \end{bmatrix} \begin{bmatrix} p_1 \\ p_2 \\ p_3 \end{bmatrix} = \begin{bmatrix} \mathrm{p}_{\mathrm{norm}\,1} \\ \mathrm{p}_{\mathrm{norm}\,2} \\ \mathrm{p}_{\mathrm{norm}\,3} \\ \cdots \\ \mathrm{p}_{\mathrm{norm}\,n} \end{bmatrix}$$

$$\begin{bmatrix} v_1^2 & v_1 & 1 \\ v_2^2 & v_2 & 1 \\ v_3^2 & v_3 & 1 \\ \cdots & \cdots & \cdots \\ v_n^2 & v_n & 1 \end{bmatrix} \begin{bmatrix} q_1 \\ q_2 \\ q_3 \end{bmatrix} = \begin{bmatrix} \mathrm{Q}_{\mathrm{norm}\,1} \\ \mathrm{Q}_{\mathrm{norm}\,2} \\ \mathrm{Q}_{\mathrm{norm}\,3} \\ \cdots \\ \mathrm{Q}_{\mathrm{norm}\,n} \end{bmatrix}$$

The matrix in here can also be represented by the variable, A, that contains columns of square of voltage, voltage, and 1. x and y represent the three p parameters and three q parameters that need to be determined. Lastly, P_{norm} and Q_{norm} are the two vectors on the right hand side of equations.

$Ax=P_{norm}$

$Ay=Q_{norm}$

To construct an inverse matrix of A, we need to have a square matrix A. Therefore limited information needs to be taken from the 339 time steps. If we have the matrix A as a square matrix, we can rearrange the equations and get x and y as:

$x=A^{-1}\,P_{norm}$

$y=A^{-1}\,Q_{norm}$

As mentioned above n=339 in this case, and we have several options to solve this problem. Because it becomes an over-determined system that has more equations than unknown variables, we can select three different time steps with different voltages and their respective P_{norm} and Q_{norm} to solve the problem. From the 339 sets of values generated by the simulation, three time steps at time 0.5, 1, and 1.5 or at steps 89, 139, and 164 were selected. Solve the x and y using inverse matrix in Matlab, we get $p_1 = -121.3506$, $p_2 = 120.0284$, $p_3 = 4.2373$, $q_1 = 68.8928$, $q_2 = -56.1995$, and $q_3 = -4.0010$.

Although the above methods will solve the equations, only three time steps were chosen. Therefore, we can still solve the x and y by a proposed pseudo inverse method where all 339 cases will be used. The over-determined system will be solved to yield x and y that would match the matrix equations as closely as possible. After solving the problem using pseudo inverse, the parameter values are $p_1 = -110.2734$, $p_2 = 111.2523$, $p_3= 0.0273$, $q_1= 42.0395$, $q_2 = -42.7255$, and $q_3 = 0.0013$. Using these parameters with the 339 voltages input, the active and reactive powers look like the ones in Figure 3.4 and Figure 3.5 respectively. Notice that the red lines are the original power generated by the system, and the blue lines are the simulated lines using these newly determined parameters.

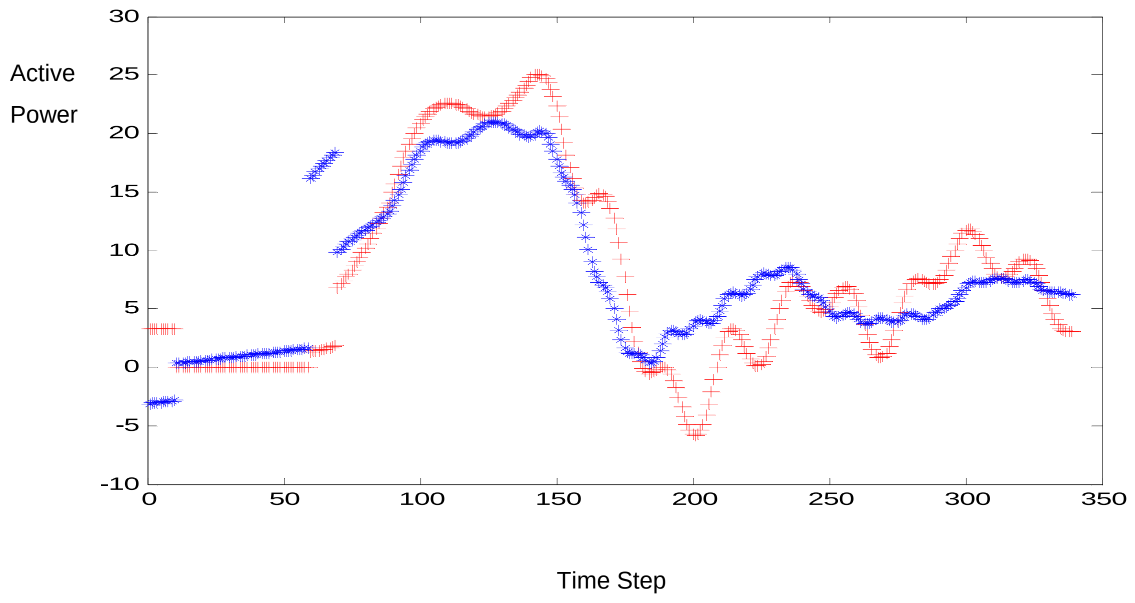


Figure 3.4 Actual and simulated active power of load bus 3RR

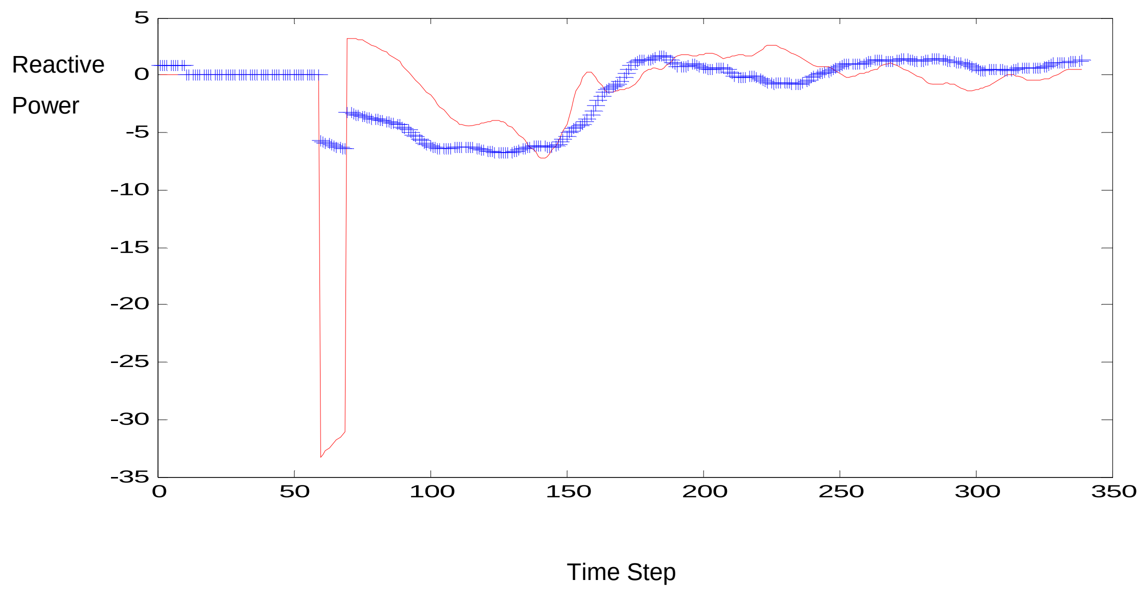


Figure 3.5 Actual and simulated reactive power of load bus 3

Although the curves for the most part were matched fairly well, the parameters do not seem to provide feasible or meaningful solutions. The problem lies with these p and q parameter values as they are large numbers instead of decimal fractions that are supposed to represent the percentage of constant load. Sometimes, these parameters are even negative. Hence, a better strategy needs to be used to predict these parameters. Other similar techniques such as linear programming were used and similar values were obtained for these parameters. The aggregated load buses and power systems do not seem to work as well as the simple power system.

In order to see how well the simulated load Bus 3 works with the ZIP load equations, we can try to solve for P_0 and Q_0 with constant impedance, constant current, and constant power. From Figure 3.6 below, we can see that as power systems get large and more complicated, the values for P_0 and Q_0 in each case no longer stay constant as we follow the power system closely.

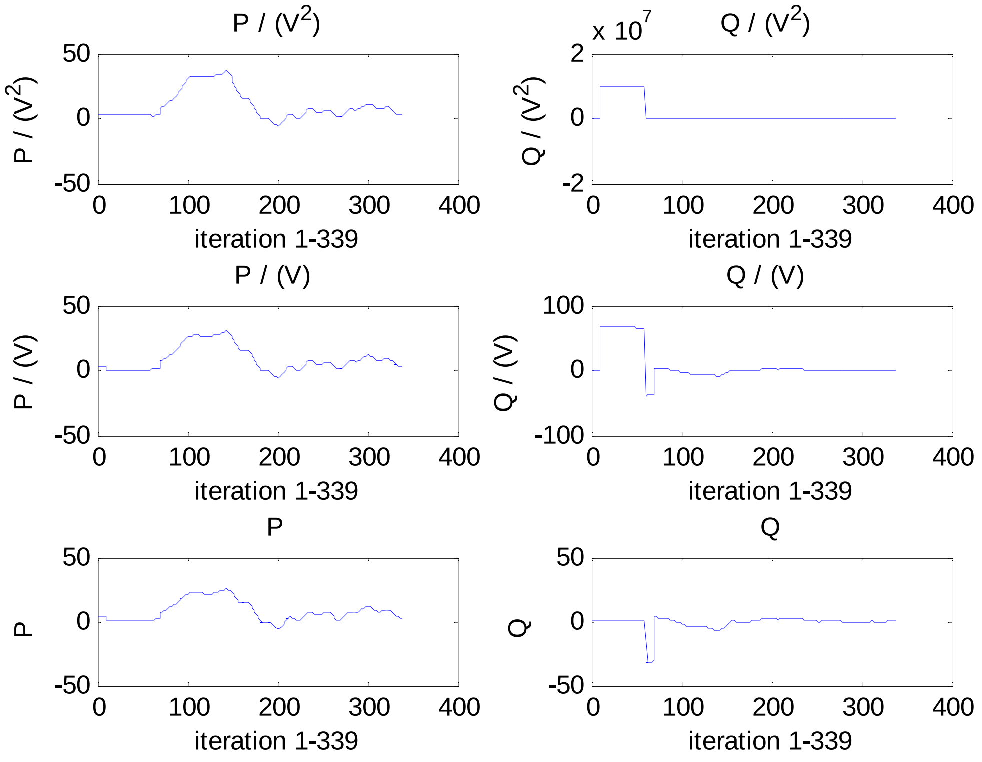


Figure 3.6 Values for P_0 and Q_0 for load bus 3

As demonstrated in this section, the parameters of p's and q's obtained using matrix manipulation for the ZIP load model is not feasible for realistic load models. Due to the complexity of the power systems, other parameter identification methods will be discussed in the next section.

CHAPTER 4 - PARAMETER ESTIMATION USING INTELLIGENT TECHNIQUES FOR STATIC LOAD MODELS

4.1 Introduction

As briefly touched on in the last section, the conventional matrix calculations do not work too well for power system load parameter identification. Other techniques need to be sought to solve the problems. The main idea here is to find the parameters p_1, p_2, p_3, q_1, q_2, q_3 that will form the components of the load model. The general search and use of load parameters are shown below.

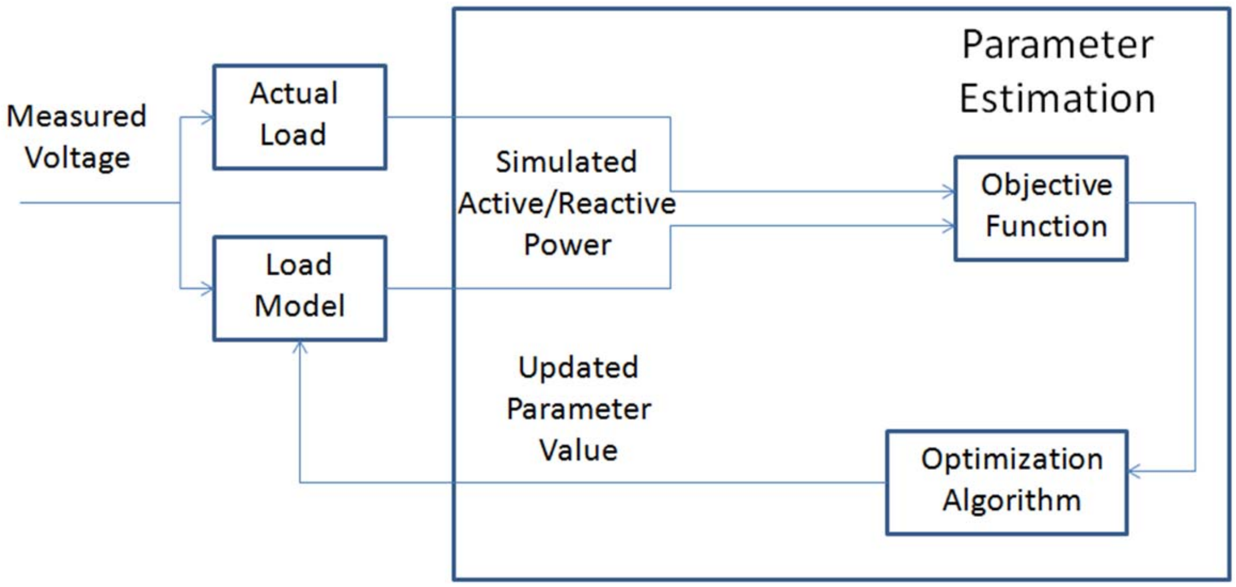


Figure 4.1 Parameter estimation procedures

As shown above, the parameters can be estimated based on difference between the measured active/reactive power and the simulated active/reactive power. Another approach is to train the system with inputs as the voltage, active/reactive power, and the outputs as the parameters p_1, p_2, p_3, q_1, q_2, and q_3. We use the already given system's information including the parameters, and train a model that will learn the system to be used with other unknown systems for finding the load parameters. Figure 4.2 below demonstrates a simple model that will train the system to find the parameters for power systems with unknown parameters.

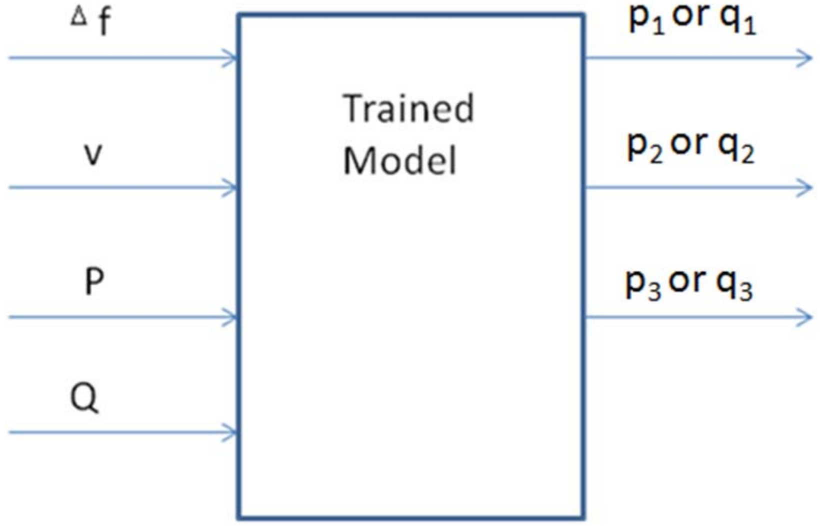


Figure 4.2 Simple parameter identification training

In order for the training model to work well, we need to have a set of rules for the training and testing. The trained model should be able to predict the parameters for itself if the inputs are given. Training and validation need to be done on the same system making the training model legitimate and useful. For instance, a set of data is used to train the model, and the rest of the set of data is used to validate and to observe the mean square error between real and simulated outputs. Consequently, the trained model should be applicable to the other load buses in other parts of the power systems. The trained model is useful if it can be used to predict the unknown parameters for the other load buses based on their values of v, P, and Q. Since compared to the other inputs, the frequency had less effect on the system; we neglect this component for simplifying our study in this thesis.

4.2 Initial Approaches

4.2.1 Simple Power System

First, we would choose the simple power system that was used in the excel spreadsheet before to be tested. This simple system would be fairly feasible as it works well with the ZIP load model. There are only 5 cases of combinations of p_1, p_2, p_3, q_1, q_2, and q_3, and as the p's and q's are assumed to be the same, we will ignore the q's for now. The five cases of variations in the parameters for percentage of constant real impedance p_1, percentage of constant real current p_2, and percentage of constant real active power p_3 are displayed in the table below.

Table 4.1 Parameters for simple power system

Status	p_1	p_2	p_3
Constant Impedance	1	0	0
Constant Current	0	1	0
Constant Power	0	0	1
Evenly Distributed Load	0.333	0.333	0.333
Mixed Load	0.2	0.3	0.5

After few trials and errors, we observed that instead of training all the inputs and outputs at once, the model will be more accurate if we split the training into three different sets for each p_1, p_2, and p_3. We can go to Appendix A.1 to verify the effectiveness of individual training compared to the original combined training. An illustration of the split training for each p parameter is shown in Figure 4.3 below.

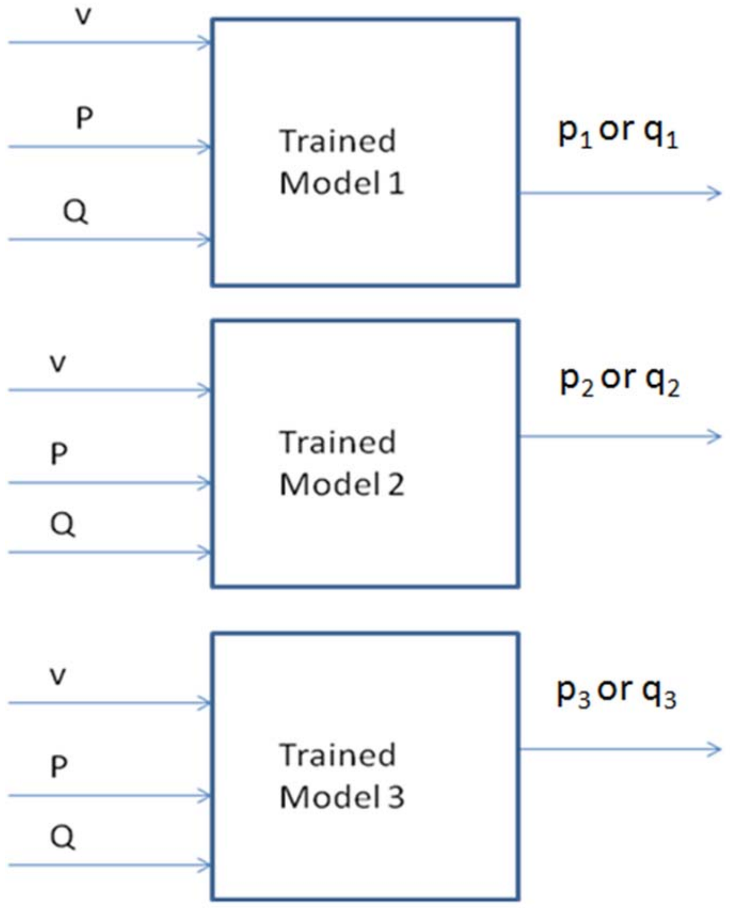


Figure 4.3 Split training method

Training this simple model yields the mean square error to be below 4%. Appendix A.2 shows the difference between the actual and simulated (tested) six parameters in more detail. Notice also that although p_1, p_3, q_1, and q_3 were trained well, p_2 and q_2 were a bit off. p_2 and q_2 do not seem to work as well as the other outputs in training. Another method uses v, Δv, ΔP or ΔQ as the inputs, and the outputs are one of the six p or q coefficient parameters like the previous part. The result for this method gives similar results to those of the last example; the simulated p_1, p_3, q_1, and q_3 worked well, but p_2 and q_2 were a bit off.

4.2.2 16-generator power system

Now, we move on to test a more complicated 16-generator power system. The written code of a backpropagation neural network and an adaptive neuro-fuzzy inference system

(ANFIS) were used to initially train and validate the inputs and outputs. Due to the nature of ANFIS and while declaring sufficient member functions, the training will be fairly slow. Therefore, the backpropogation neural network was initially used to train and test the system to see which method worked the best. The training methodology will be discussed in the next section. The 3-phase fault locations and their relative load buses are shown in Table 4.2 below. The faults between 41-42 and 43-44 that are highlighted are more sensitive. Therefore, we focus on the load buses surrounding these two fault locations. To determine the best data sets for training, we find out whether the selected load bus or buses are close to the fault location. A combination of load data from various bus or buses at the same or different fault or non fault location was considered.

Table 4.2 The fault locations and their load buses

3-phase fault locations (bus-bus)	Fault between buses 41-42	Location from the fault	Fault between buses43-44	Location from the fault
bus	41	at the fault	43	at the fault
	42	at the fault	44	at the fault
	37	far from the fault	36	far from the fault
	52	Close to the fault	51	Close to the fault

As a result, we can have various training strategies with their respective chosen bus positions shown in table below. This way, we can identify the best selections and combinations of buses to choose from for training.

Table 4.3 The training strategies with their respective training locations

Training strategy (Trained on:)	**Training locations**
A1 (37)	from fault 41-42
A2 (41)	at the fault 41-42
A3 (42)	at the fault 41-42
A4 (52)	close to fault 41-42
A5 (36)	far from fault 43-44
A6 (44)	at the fault 43-44
A7 (43)	at the fault 43-44
A8 (51)	close to fault 43-44
B1 (41, 42)	both at the fault 41-42
B2 (44, 43)	both close to fault 41-42
B3 (37, 41)	one far from, one at the fault 41-42
B4 (36, 44)	one far from, one at the fault 43-44
B5 (37, 42)	one far from, one at the fault 41-42
B6 (36, 43)	one far from, one at the fault 43-44
B7 (41, 52)	one close to, one at the fault 41-42
B8 (44, 51)	one close to, one at the fault 43-44
B9 (37, 52)	one close to, one far from fault 41-42
B10 (36, 51)	one close to, one far from fault 43-44

Several attempts and approaches were used to train for the parameter identifications. One approach (row) in Table 4.3 was used for each test. A list of load characteristic training cases is listed below to be used for outputs. These values represent the percentage of type of load used in training.

Table 4.4 Load characteristics cases for training

p_1 (or q_1)	p_2 (or q_2)	p_3 (or q_3)
0	0	1
0	1	0
1	0	0
0.8	0	0.2
0.6	0.2	0.2
0.4	0.4	0.2
0.2	0.6	0.2
0	0.8	0.2
0.6	0	0.4
0.4	0.2	0.4
0.2	0.4	0.4
0	0.6	0.4
0.4	0	0.6
0.2	0.2	0.6
0	0.4	0.7
0.2	0	0.8
0	0.2	0.8
0.8	0.2	0
0.6	0.4	0
0.4	0.6	0
0.2	0.8	0
0.5	0.3	0.2
0.33	0.33	0.33

Once these cases are trained against a reference load at Bus 41, we tested or verified them through the load cases in Table 4.5. The trained model needs to be used to test against all other load buses.

Table 4.5 Load characteristic cases for testing

p_1 (or q_1)	p_2 (or q_2)	p_3 (or q_3)
0.5	0.2	0.3
0.3	0.2	0.5
0.2	0.3	0.5
0.8	0.1	0.1
0.3	0.4	0.3
0.2	0.1	0.7

For more details, please refer to Appendix A.3. From the appendix, we see that although the training works well, the testing against all 29 cases works only well with the p_1 and p_3 output parameters. p_2 could not be verified very well. This might be explained considering that the constant current component is not very sensitive compared to the other components.

The weakness with the previous approach is that 339 time steps were trained together versus one load component case, and then another 339 time steps were trained with the next load component case and so on. Another weakness is that all the buses' load components changed the same way. For instance, when we run the simulations for fault 41-42, load components for load buses 41, 42, 37, 52 would all change exactly the same way as described on Table 4.3. Moreover, there is also a lack of finding the relations between these each time steps. Another method was proposed to train the network only 29 times (29 cases), but with each input taking voltages and powers at a few time steps each instance. The figure below demonstrates this training case. To simplify the demonstration, only training with the output parameter p_1 is shown.

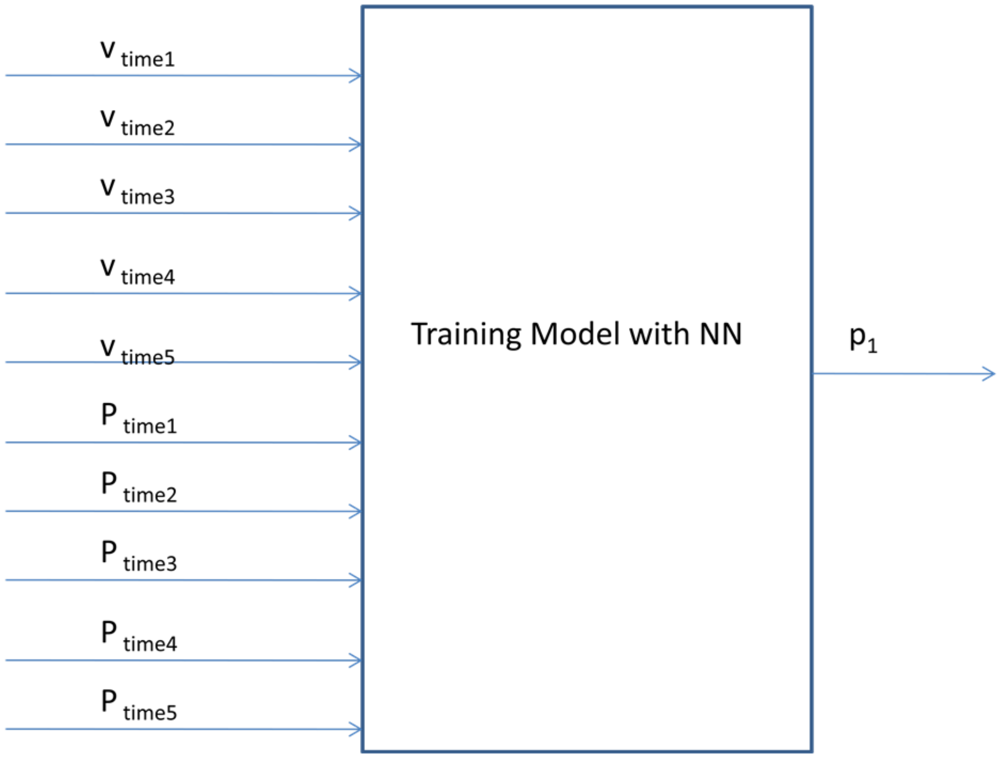


Figure 4.4 Parallel training with bus 41

The training accuracy is similar to the training method in Figure 4.3, however. Parameters p_2 and q_2 still did not get verified well after training.

In studying the previous approaches of finding the parameters, it was noted that an important component in the training that was missing was the sensitivity of the active and reactive power values to changes in voltage. The question is how the change of voltages and reactive or active powers through a fixed time would relate to the load components. To use the mathematical expressions to find the relations between $\Delta v, \Delta P, \Delta Q$, we do the following derivation for Δv and ΔP as Δv and ΔQ will have the similar derivation.

Recall from the simplified ZIP load model, we replace the parameters p_1, p_2, and p_3 with k_1, k_2, and k_3, and i represents the initial time for the value. The derivation is as follows.

$P_i = k_1 v_i^2 + k_2 v_i + k_3$

Let $v_{i+1} = v_i + \Delta v_i$

$P_{i+1} = k_1 v_{i+1}^2 + k_2 v_{i+1} + k_3$

Therefore, we can express ΔP_i as:

$$
\begin{aligned}
\Delta P_i &= k_1\left(v_{i+1}^2 - v_i^2\right) + k_2\left(v_{i+1} - v_i\right) + k_3 \\
&= k_1\left(v_{i+1}^2 - v_i^2\right) + k_2\left(v_{i+1} - v_i\right) + k_3 \\
&= k_1[v_i^2 + 2(\Delta v_i)(v_i) + \Delta v_i^2 - v_i^2] + k_2(v_i + \Delta v_i - v_i) + k_3 \qquad \mathbf{(4.1)} \\
&= k_1[2(\Delta v_i)(v_i) + \Delta v_i^2] + k_2(\Delta v_i) + k_3 \\
&= k_1 \Delta v_i^2 + (2k_1 v_i + k_2)\Delta v_i + k_3
\end{aligned}
$$

Hence, ΔP_i is a function of Δv_i and $\left(\Delta v_i\right)^2$. Depending on the value of v_i, and if v_i is a known or a constant value, we would also have a similar ZIP load equation in terms of Δv_i .

The training method related to $\Delta v, \Delta P, \Delta Q$ can be shown in Figure 4.5 below. For simplicity, only training with output p_1 is shown. The time steps taken from the simulations are also shown in Table 4.6 below.

Table 4.6 Time steps for the training system

	Δv_1 or Δp_1	Δv_2 or Δp_2	Δv_3 or Δp_3	Δv_4 or Δp_4	Δv_5 or Δp_5
initial time (seconds)	2.5	3	3.5	4	4.5
time steps during this Δ range	214-215	239-240	264-265	289-290	314-315

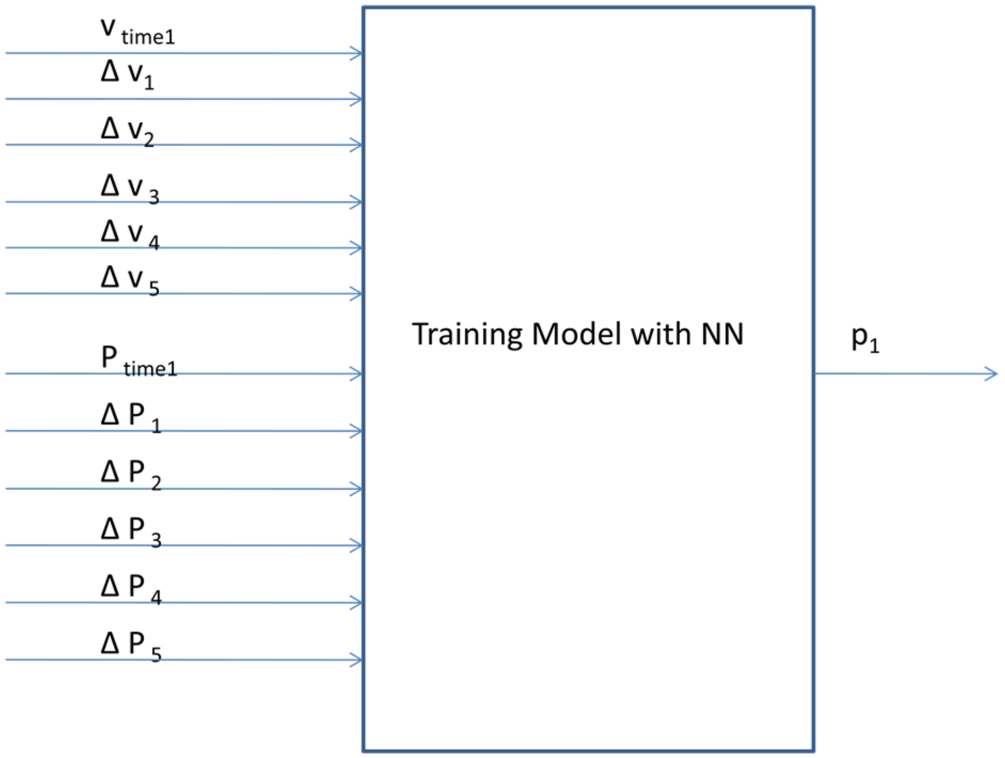


Figure 4.5 Training method with inputs $\Delta v, \Delta P, \Delta Q$

The results did not improve, however this system makes a little more sense for the power system training. After a series of trial and errors, two problems stand out most.

- After training, parameters for p_2 and q_2 still do not get validated well with the generated load model.
- Insufficient load characteristics cases for the training. A more diverse and random set of load components would better represent a real power system.

We need to develop a much larger set of inputs and outputs with more varied and irregular load components to resemble the real power system data. Also, steps need to be taken for the load model to do a better job predicting the p_2 and q_2 parameters. The final approach in the next section talks about the procedures to train, validate, and test the 16-generator power system.

Normalization is also important for the process of passing data from the power system to the intelligent system. The intelligent systems can better understand and find the pattern for the system if it is expressed in ratio instead of the actual values. As the output parameters are in the range of 0 to 1, they do not need to be normalized. The input parameters of v, P, Q, or the delta values certainly need to be normalized. Assuming that we already know the maximum value, max, minimum value, min, and the actual value; the normalized value can be expressed as:

$$\text{normalized value} = \frac{\text{actual value - min}}{\text{max - min}} \tag{4.2}$$

Note that for some system, the maximum and minimum values are noise that is either too large or too small. In case like this, we need to filter out these extreme values.

In order to solve the problem of generating better models for testing p_2 or q_2, we may go back to the definition of these parameters. As we know from Equations (2.20) and (2.21), we have:

$$p_1 + p_2 + p_3 = 1$$

$$q_1 + q_2 + q_3 = 1$$

Therefore, if we get good training results for p_1, p_3, or q_1, q_3, we could use the information to predict the p_2 or q_2 values. During the training, after normal models were developed for outputs p_1 (or q_1) and p_3 (or q_3), the value for p_1 (or q_1), p_3 (or q_3), and as well as $1-p_1-p_3$ (or $1-q_1-q_3$) could be feed in as inputs to develop the load model for output p_2 (or q_2). Figure 4.6 displays the basic strategy for developing a better artificial neural network load model for p_2 (or q_2).

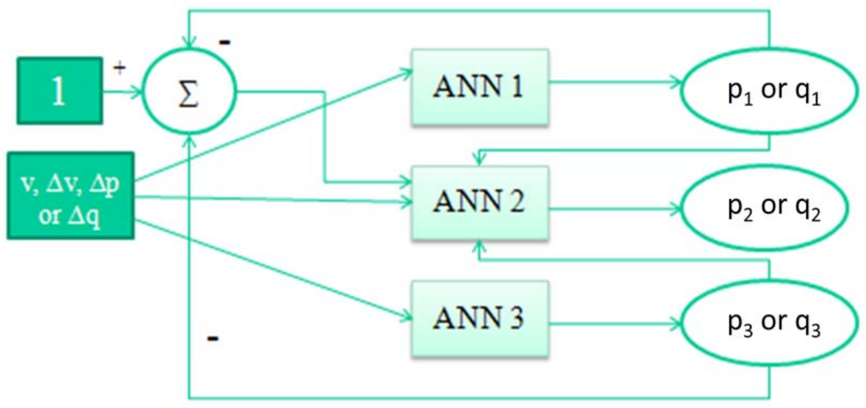


Figure 4.6 Strategy to develop a better load model for output p_2 (or q_2)

Once we established this special training method for ANN2, we could predict the values for p_2 (or q_2) fairly well. To look at this case in further detail, please refer to Appendix A.7. After these trials and errors, we are ready to set a standard to train and test the load buses. This standard is presented in the next section.

4.3 Training Methods

4.3.1 Adaptive Neural Fuzzy Logic Method

Fuzzy logic has two theories. One theory states that the fuzzy logic is an extension of logic with multi-values. The other theory can be used to describe the non-sharp boundaries where the memberships are matters of degrees. Fuzzy logic is used commonly in solving various engineering problems as it is flexible and not too difficult to learn. The input-output data set can be matched with the created fuzzy system. This process can be trained by an ANFIS (adaptive-neural network-based fuzzy inference system). The Sugeno fuzzy model is used for a three-input and single output system similar to the model presented in [6]. The fuzzy inference system is the method of mapping from given inputs to an output using fuzzy logic. Several components of the system include “membership functions”, “logic operations”, and “if then rules.”

The Neuro-Adaptive learning methods are very similar to neural networks. The learning technique provides a method for the fuzzy modeling procedure to learn information about a data set. ANFIS is a sample of this learning technique where the membership function parameters are adjusted by a backpropagation algorithm and in combination with the least squares method [43].

In this thesis, the Sugeno method was used; the algorithm was coded with the Matlab code "genfis1" to construct the membership functions, and the ANFIS was used to train the model to recognize the load patterns. The trained fuzzy system has its own membership functions and rules and generates the model output after feeding in the new inputs. Unlike the other intelligent techniques (which include randomness in the initial conditions), ANFIS models do not have to run multiple times in order to choose the best result as all the runs will yield the same MSE value. However, in this research, ANFIS method is strongly discouraged to use. The ability for ANFIS to predict parameters of unknown buses has been shown to be fairly poor relative to the other methods. More membership functions need to be used to yield better results. However, the simulation process to finish the training already takes a lot of time. By adding more membership functions, the training will require even more time to finish.

4.3.2 Levenberg- Marquardt algorithm Method

In general, this method does a great job of fitting any simple practical function. Levenberg-Marquardt algorithm is used in this work [44]. In [45], the author demonstrated for Levenberg-Marquardt method, we first need to look at the Gauss-Newton method to find the equation needed. However, if the desired solving system J(x) is rank-deficient, the Gauss-Newton method would not be able to converge or converge to a stationary point. Therefore, true-region technique was considered. The least squares problem were included to solve by rearranging the equations for J(x) [45, 46].

One disadvantage of using the neural networks Levenberg Marquardt algotirhm method is that sometimes the process does not converge very well. A few runs need to be executed to get a good result. In this thesis, ten runs were done with different starting points, and the model that contains the smallest MSE or Mean Square Error was used. In another comparison between the methods, the learning rates, parameters, number of hidden neurons, and number of epochs for these methods were all set to be identical. Each method will have distinctive 10 trials for training, and MSE for the testing these simulated data against the real data were recorded in each trial. All these results were used to compare between these methods.

In our research, Levenberg-Marquardt algorithm approximates the data more accurately compared to the ANFIS method. However, the training will not always converge to the small MSE value desired for the Levenberg-Marquardt algorithm. Using the Levenberg-Marquardt algorithm to train with a large set of data also requires large amount of computation time. Even though, it has an advantage over the Adaptive Neural Fuzzy Logic method, this method, after a load model is formed from training, did not do a very good job of validating simulated or predicted data against actual data in other unknown buses.

4.3.3 Widow-Hoff Backpropagation Method

In reference [44], the author used Widow-Hoff delta learning rule with multiple-layer networks and non-linear transfer functions are used in this Matlab backpropagation method. A gradient descent algorithm where the network weights are moved along the negative of the gradient of the function is the standard backpropagation. Inputs are applied to the network, the outputs are calculated, and the resulting error is used to adjust the weights in a back to front type order. The general structure is shown below in Figure 4.12. The backpropagation usually uses MSE to evaluate the real outputs and the generated outputs. The difference is recorded and will be used to refine the backward calculation until the real and generated results match closely. Generally the rule of thumb is the more hidden neurons and the more layers, the better the results. However, in a very big training set, more layers and hidden neurons will also greatly delay the computational training time. Figure 4.12 below shows a diagram of a Widow-Hoff backpropagation method.

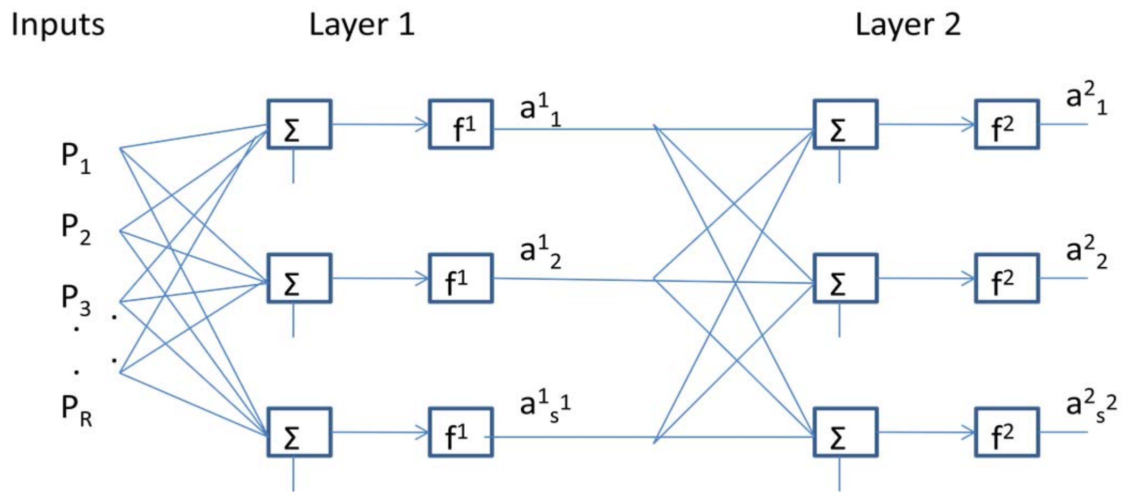


Figure 4.7 Widow-Hoff Backpropagatioin method with the inputs and outputs [44]

The training methodology is as below:

- Input and output data are gathered from the transient fault simulation.
- Normalization is applied to the input and output data.
- The initial weights are randomized at the beginning for each run.
- Now do ten runs and in each run do 500 epochs of training the networks with the updated weights using the backpropagation algorithm. The error will be recorded each time, and the weights will be used in the test data to produce the test error at the same time.
- For each epoch, all data from the input data and output data are trained, the weight is updated through the backpropagation function with the input and output data.
- In backpropagation, the forward method was used to obtain the net output from the second layer of the neural networks. The backpropagation is used to get the deltas in the first and second layers. These values are used to update the weights for both the first and second layers.
- The final weights were used to calculate the mean square error by comparing the actual output versus the calculated outputs using the forward method.
- Finally, the plots for both the training and test errors are produced from the above information.
- During the process, there were times we needed to take off the bias, and add back the bias and the momentum. The learning rate and momentum factors were adjusted and optimized by hand after few trial and errors to achieve the smallest error for both the training and test data.
- Also, to get the confusion matrices, a set of if statements were used and number of correct or mistaken counts were recorded in order to obtain the percentages of correction or mistakes in each output case.

Note that each p_1, p_2, p_3, q_1, q_2, q_3, or β were trained separately with the same input information except p_2 and q_2, where more care needed to be taken. Figure 4.8 below summarizes the training procedure.

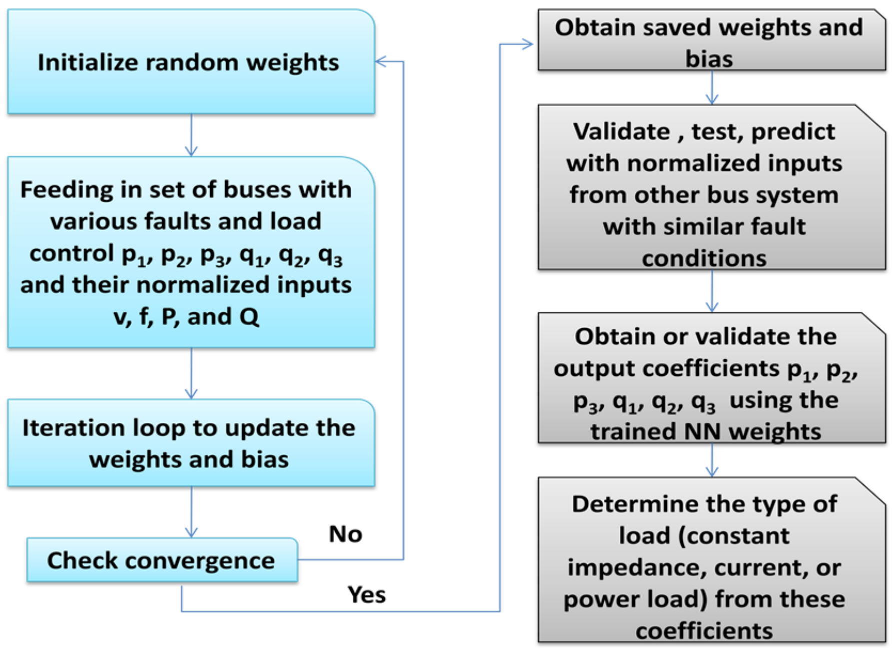

Initialize random weights
Feeding in set of buses with various faults and load control $p_1, p_2, p_3, q_1, q_2, q_3$ and their normalized inputs v, f, P, and Q
Iteration loop to update the weights and bias
Check convergence
No
Yes
Obtain saved weights and bias
Validate , test, predict with normalized inputs from other bus system with similar fault conditions
Obtain or validate the output coefficients $p_1, p_2, p_3, q_1, q_2, q_3$ using the trained NN weights
Determine the type of load (constant impedance, current, or power load) from these coefficients

Figure 4.8 Procedure for backprogation algorithm training

Also, ten runs for the whole program were needed, and the best one was used for constructing the load model. In another section in the thesis, all the trials were recorded and the results were compared with the other methods. Similar to the previous method, although this Widow-Hoff method trains the data well for the training set, the weights obtained will not do an excellent job in predicting the parameters for the unknown buses.

4.3.4 Default Scaled Conjugate Gradient Algorithm Method

As shown in the Matlab manual [44] and discussed in [47], a basic backpropagation algorithm adjusts the weights according to the direction of the steepest descent or negative of the gradient. The basic backpropagation algorithm does not produce the fastest convergence. The conjugate gradient algorithm on the other hand allows a search to be performed along conjugate directions, and converges faster than the steepest descent directions. Even though the results for conjugate gradient algorithms varies from one problem to another, this algorithms only needs a little more storage compared to the basic algorithms. Hence, it works well with networks with a large amount of weights. Furthermore, Scaled Conjugatge Gradient (SCG) method avoids the line search per learning iteration. SCG does not include user dependent parameters as the values will be important for the success of the algorithm. Compared to the line search based algorithms which include these kinds of parameters, avoiding these depending parameters in SCG seems to be a significant advantage [48].

To do a good job in training with this algorithm, more neurons and more hidden layers could be used with a cost in computation time. In our work, SCG method worked very well for training, testing, and validating unknown buses based on load model that was developed through training. This technique is highly recommended for parameter identification in power system load model.

4.4 Analyze Strategies and Methods with Best Training Approach

In this section, we trained the data from selected buses (assumed to be the known buses) with different training methods and strategies. The different training methods are described in a later part of this section. The training strategies described here involve constructing the model by selecting data from various combinations of buses from Table 4.7 on the next page for training. With the specified strategy and method, the weights and models obtained from the training will be used to test and predict the data (load parameters p_1, p_2, p_3, q_1, q_2, and q_3) from the unselected buses (assumed to be the unknown buses) as well as the known buses. The mean square errors between the simulated data and the actual data were recorded from ten trials in each method and strategy. In this Section 4.4, only the results from the best trials were recorded and used for comparison between the effectiveness of each strategy and method. Parameters from these methods, especially ANFIS method were optimized by hand. In ANFIS method, the member function and epoch numbers were also adjusted in order to train the data more accurately and with an allowable reasonable time. Section 4.5 later on will do a slightly different comparison.

4.4.1 Test Cases and Procedures

Recall from the previous sections that in order to train and validate the bus systems well, we need to have:

- Large size of training sets with irregular and randomized combinations of load components.

- Sensitivities or amount of increase or decrease of voltages, active/reactive powers need to be used in training
- Normalization needs to be used on the same type of input. Filtering out the excess of noise should be used but with caution.
- Special training methods need to be used for training outputs p_2 or q_2.

From the previous lessons, we would train the system with a large training set and include considerations for all the bullets listed above. After the training, we also realized that the early time steps that are close to the fault (and thus with larger variations in the variables) need to be used as these will better represent and can better reflect the load and its characteristics. In this approach, six different time steps are used for the training.

Table 4.7 Time steps for the final approach

Δ t=0.2 seconds	Interval 1	Interval 2	Interval 3	Interval 4	Interval 5	Interval 6
initial time (seconds)	1.02	1.22	1.42	1.62	1.82	2.02
Final time after this Δ range	1.22	1.42	1.62	1.82	2.02	2.04

However, the six different time steps will be used to create six input cases, therefore, if there are, for instance, 20 test cases then we would have 20 *6=120 sets of inputs. The structure of the training is shown below in Figure 4.7. Note that in order to simplify the demonstration, only output p_1 is shown here.

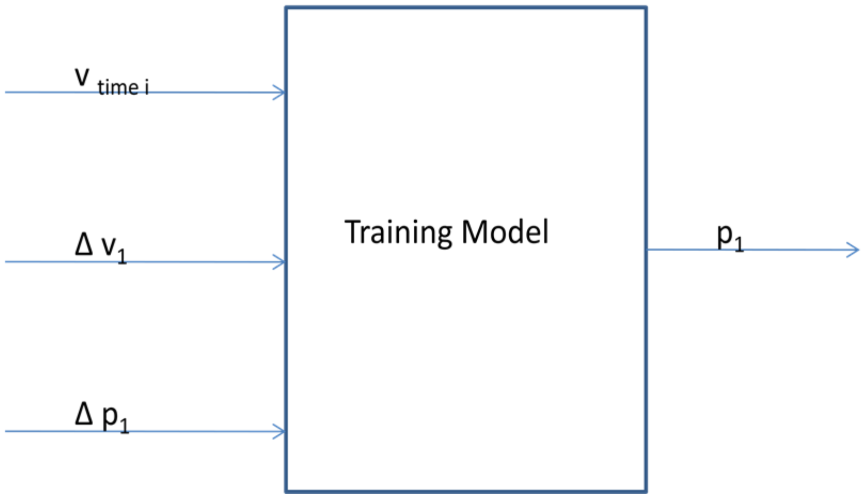


Figure 4.9 Training Model for 3 inputs and one output

The load faults 41-42, 43-44 and their respective load buses used in this approach are shown below.

Table 4.8 The faults 41-42, 43-44, and their respective load buses

3-phase Fault Locations	Fault at buses 41-42	Fault at buses 43-44
buses	41	43
	42	44
	37	36
	52	51

In this section, we have a large set of data. The transient fault simulation is constructed so that four load buses are used to represent various the load characteristic components of the power system. Only p_3 and p_2 need to be expressed, as p_1 can be calculated by $1-p_3-p_2$. One sample set of load components is shown below in Table 4.9.

Table 4.9 One instance of load component for simulation at fault 41-42

Fault at 41-42	p_3 or q_3	p_2 or q_2
37	0.1	0.2
41	0.3	0.7
42	0.1	0.3
52	0.6	0.3

In the Matlab software, a matrix named "load_con" is used to define the ZIP model for certain loads. Now, we need to construct a large "load_con" matrix so that almost any constant load component set can be considered and used for training for the intelligent system. For instance, the Table 4.8 above only has shown one instance of distribution of constant load components. To get a large amount of instances, the p_3 or q_3 can be generated by letting load bus increment from 0 to 1 by a step size of 0.2. Then all these increments will only associated with the load Bus 41 at 0, and load Bus 41 will also increment from 0 to 1, and each increment will experience Bus 37 increment from 0 to 1. Consequently, all these changes will happen when Bus 42 is at 0, and then there will be other values for Bus 42 at which all these described changes take place. Figure 4.8 shows the illustration of different types of load components for faults at Buses 41 to 42. As each load bus can have six values from 0 to 1, therefore, there can be 6^4 or 1296 cases of instances for p_3 alone.

The value for p_2 can be generated by a random number generator that gives a value less than the value of $1-p_3$. The p_1 value is obtained from $1-p_2-p_3$.

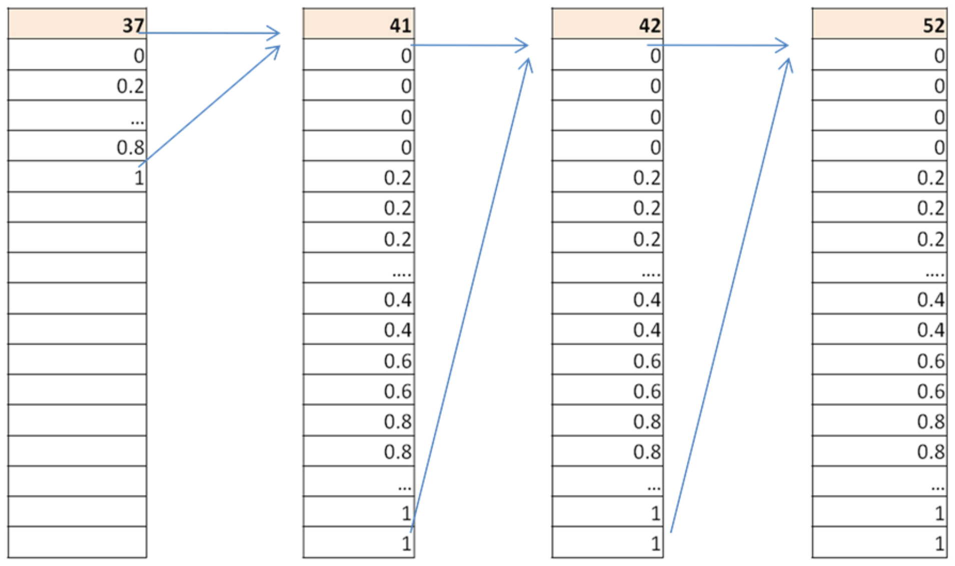


Figure 4.10 Various load components for load buses at fault 41-42

However, we can still generate more load cases. We would create a new set of data from the pre-existing data that swaps the p_3 and p_2 values. Also, remember that for each case, there are six time steps. Hence, the number of instances of load components can be as large as:

1296 x 2 x 6 = 15552 load case instances

Since each simulation for a transient fault at buses 41-42 can record six different sets of inputs and outputs at six time steps, the simulation for a transient fault at 41-42 needs to be executed 15552 divide by 6 times or 2592 times.

For a standard XP computer in the computer lab, about 9 seconds is need for each run.

Time generating inputs and outputs = 2592 x 9= 23328 seconds = 388.8 min =6.48 hour

A similar method for generating the inputs and outputs information at faults 43-44 was performed.

4.4.3 Performance Evaluation of Various Methods and Strategies

As mentioned before, in order to evaluate the best case and strategy used, training was only done with output p_1. To test against itself, at fault 41-42, this load bus was trained and tested. The first 10,000 load cases were used to train while the 5,552 load cases were used to test or validate the load model. Starting at fault 41-42, load bus was trained and tested with the 15,552 cases with different training methods, and the results are summarized in Table 4.10. Discussions of the different training methods are presented in Section 4.4. After comparisons against the previous methods, we see that the ANFIS algorithm does the best training job compared to the other methods.

Table 4.10 Different methods of training and testing against itself

Fault 41-42 , load bus: 41				
MSE for the different methods on load bus 41 (Full training and testing: all 15552 cases)				
	Neuro Fuzzy method: ANFIS	Levenberg-Marquardt Algorithm	Widow-Hoff Backpropagation Algorithm	Default Scaled Conjugate Gradient Algorithm
Fully trained MSE values	0.0068	0.0069	0.0069	0.0077

To validate and test against itself, at fault 41-42, the first 10,000 load cases were used to train while the 5,552 load cases were used to test or validate the load model. The results are shown below in Table 4.11. Notice that for a smaller sample size of data, the ANFIS algorithm does the best job on validating the load model against itself and validating against other load cases.

Table 4.11 Training with 1000 load cases and testing against itself with 5552 load cases

Fault 41-42 , load bus: 41				
Partial training and testing (Train on the first 10000 cases, test on 5552 cases)				
	Neuro Fuzzy method: ANFIS	Levenberg-Marquardt Algorithm	Widow-Hoff Backpropagation method	Default Scaled Conjugate Gradient Algorithm
Trained MSE values	0.0056	0.0058	0.0057	0.0063
Validated or tested MSE values	0.0095	0.0098	0.0103	0.0106

As the previous test was only done with a small sample size of data, we will test our system with a much larger set of data. As in Table 4.8 before, we will focus on fault at buses 41-42, and fault at buses 44-43. We will do a single load bus training and testing, with combinations of 2, 3, or even 4 load buses shown above for validating and testing. The standard rule is that we train a set of single or combination of various buses, but we will test the load model on all 8 load buses shown in Table 4.8 above and record the mean square errors. For instance, for training A1 strategy (training on load Bus 37 only), we have the MSE values for the different kinds of methods shown below. The testing will be done on all the eight load buses and the average error of these eight cases will be shown at the end. We can see at the end of the A1 section demonstrated that the Default Scaled Conjugate Gradient Algorithm method gives the most accurate training results whereas the ANFIS method does a comparably poor job. The training MSE values for training by using A1 strategy is shown below.

Table 4.12 Training strategy trained on load bus 37

Training Strategy Trained on:	**Tested on buses (MSE)**	**Levenberg-Marquardt Algorithm**	**Widow-Hoff Backpropagation Algorithm**	**Default Scaled Conjugate Gradient Algorithm**	**Neuro Fuzzy method: ANFIS**	**MSE Column Average**	**MSE Column Average (exclude anfis)**
A1 (37)	error37	0.0087041	0.0086853	0.0509954	0.0082321		
	error41	0.4742836	0.0674742	0.0474777	10.141724		
	error42	0.1751635	0.0756422	0.047743	44.152189		
	error52	0.741254	0.1381595	0.0502348	22.352412		
	error36	0.5841999	0.1367574	0.0509954	86.243		
	error44	0.2247336	0.1361067	0.0477431	118.05378		
	error43	1.6351558	0.21696	0.0476465	30.097273		
	error51	0.1283085	0.0921727	0.0502353	161.51818		
	average error	**0.4964754**	**0.1089948**	**0.0491339**	**59.070848**	14.93136	0.218201

Due to the large size of the spreadsheet and instead of showing all the eight tested parameter errors with the trained load model, we will only show the average mean square error for each training strategy that combines all the testing mean square error on each of the eight load buses. Table 4.13 summarizes all the average tested errors for the strategies and methods below.

Table 4.13 Training strategies and training methods for static load model

Training Strategy Trained on:	Tested on buses (MSE)	Levenberg-Marquardt Algorithm	Widow-Hoff Backpropagation method	Default Scaled Conjugate Gradient Algorithm	Neuro Fuzzy method: ANFIS	MSE Column Average	MSE Column Average (exclude anfis)
A1 (37)	**average error**	0.4964754	0.1089948	0.0491339	59.070848	14.931363	0.21820137
A2 (41)	**average error**	0.8134029	0.2605863	0.0482783	3478.4961	869.904592	0.37408917
A3 (42)	**average error**	0.3510092	0.4398576	0.0491547	409.95358	102.6984	0.28000717
A4 (52)	**average error**	0.1971639	0.0981685	0.0491175	319.20747	79.88798	0.11481663
A5 (36)	**average error**	8.7042359	0.5151541	0.0491551	943.42346	238.173001	3.08951503
A6 (44)	**average error**	6.6085212	0.8437232	0.1515104	174.54681	45.5376412	2.53458493
A7 (43)	**average error**	0.0990519	0.0551485	0.0328347	182.73939	45.7316063	0.06234503
A8 (51)	**average error**	2.3565249	0.9344518	0.1089633	76.365919	19.9414648	1.13331333
B1 (41, 42)	**average error**	0.4834025	0.2509	0.0686306	21.891033	5.67349153	0.26764437
B2 (44, 43)	**average error**	0.7291294	0.3800662	0.0559012	292.34947	73.3786417	0.3883656
B3 (37,41)	**average error**	0.1495188	0.0725869	0.0486374	64.995663	16.3166015	0.0902477
B4 (36, 44)	**average error**	0.3257481	0.3300078	0.0813284	425.79405	106.632784	0.24569477
B5 (37, 42)	**average error**	0.2192818	0.1482025	0.0562259	76.784213	19.3019808	0.14123673
B6 (36, 43)	**average error**	0.8097711	0.2396523	0.0469443	2097.6085	524.676217	0.3654559
B7 (41, 52)	**average error**	0.5096352	0.1888463	0.0500835	86.772676	21.8803103	0.24952167
B8 (44, 51)	**average error**	0.2505352	0.6354287	0.0652582	564.86153	141.453188	0.31707403
B9 (37, 52)	**average error**	0.1333552	0.1274686	0.0491529	30.166944	7.61923018	0.10332557
B10 (36, 51)	**average error**	1.6286079	0.5343869	0.048831	351.86842	88.5200615	0.73727527
C1 (37, 41, 42)	**average error**	0.1430585	0.1154196	0.0491513	56.642425	14.2375136	0.10254313
C2 (36, 44, 43)	**average error**	0.1908778	0.3944991	0.0589443	989.85096	247.62382	0.21477373
C3 (41, 42, 52)	**average error**	0.184652	0.2037786	0.0536802	147.30009	36.9355502	0.14737027
C4 (44, 43, 51)	**average error**	0.361787	0.127378	0.0480125	131.5802	33.0293444	0.17905917
D1 (36, 41, 42, 52)	**average error**	0.1339861	0.0786759	0.0494302	46.438301	11.6750983	0.08736407
D2 (36, 44, 43, 51)	**average error**	0.1824207	0.1274614	0.0491551	1762.1668	440.631459	0.11967907
E1 (41, 42, 44, 43)	**average error**	0.0618988	0.057713	0.0386157	9.6149073	2.4432837	0.0527425
E2 (37, 52, 36, 51)	**average error**	0.0819655	0.057171	0.039818	277.67509	69.4635111	0.0596515
E3 (37, 41, 36, 43)	**average error**	0.1173513	0.0780337	0.0436818	100.74114	25.2450517	0.07968893
E4 (42, 52, 44, 51)	**average error**	0.1239898	0.0625158	0.0365435	710.24098	177.616007	0.0743497
	MSE Row Average (averages of the average_error in each)	0.9445485	0.2666528	0.0562919	496.04096	124.327113	0.42249773

In order to better analyze the training strategies, we plot the different strategies with their respective mean square error values in Figure 4.11 below.

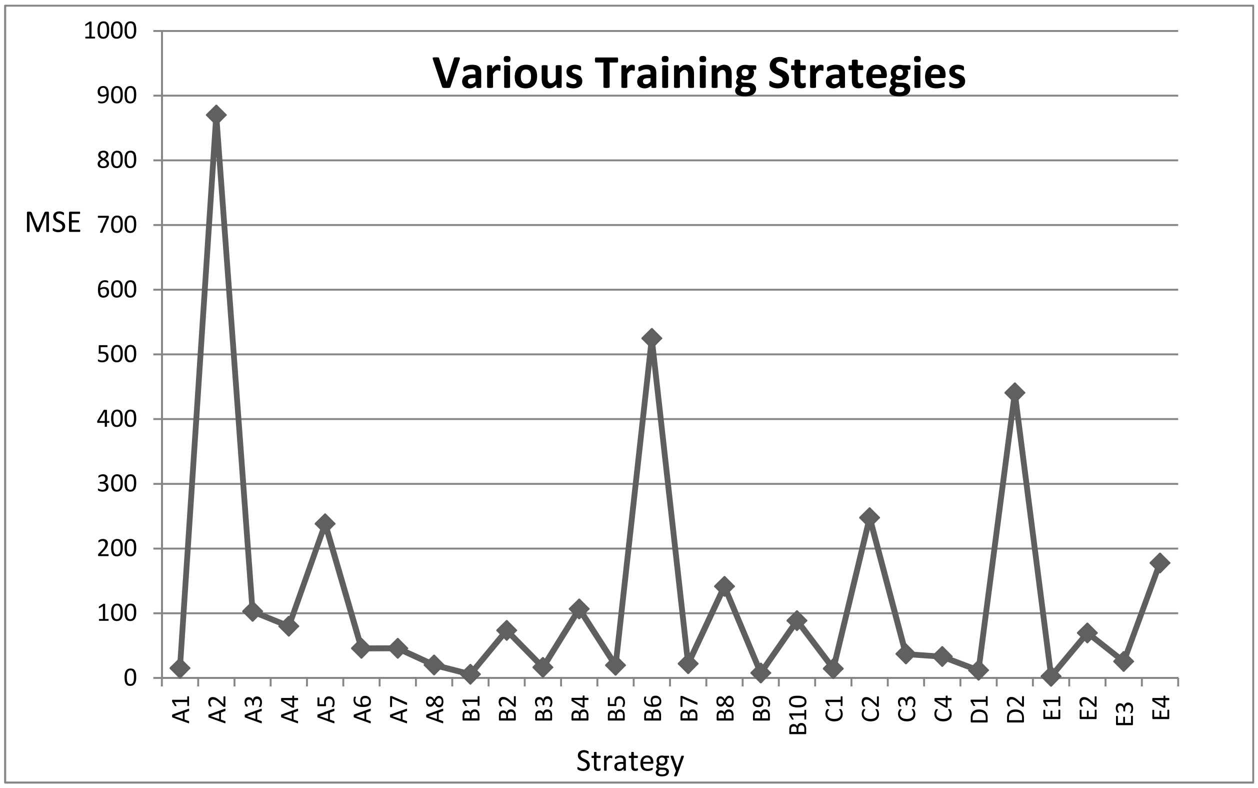


Figure 4.11 Various training strategies

Also, from Table 4.12, we realize that ANFIS method is fairly inefficient in validating the model against other load buses. Hence, we exclude ANFIS method, and replot the training strategies versus their respective MSE values as shown below in Figure 4.12.

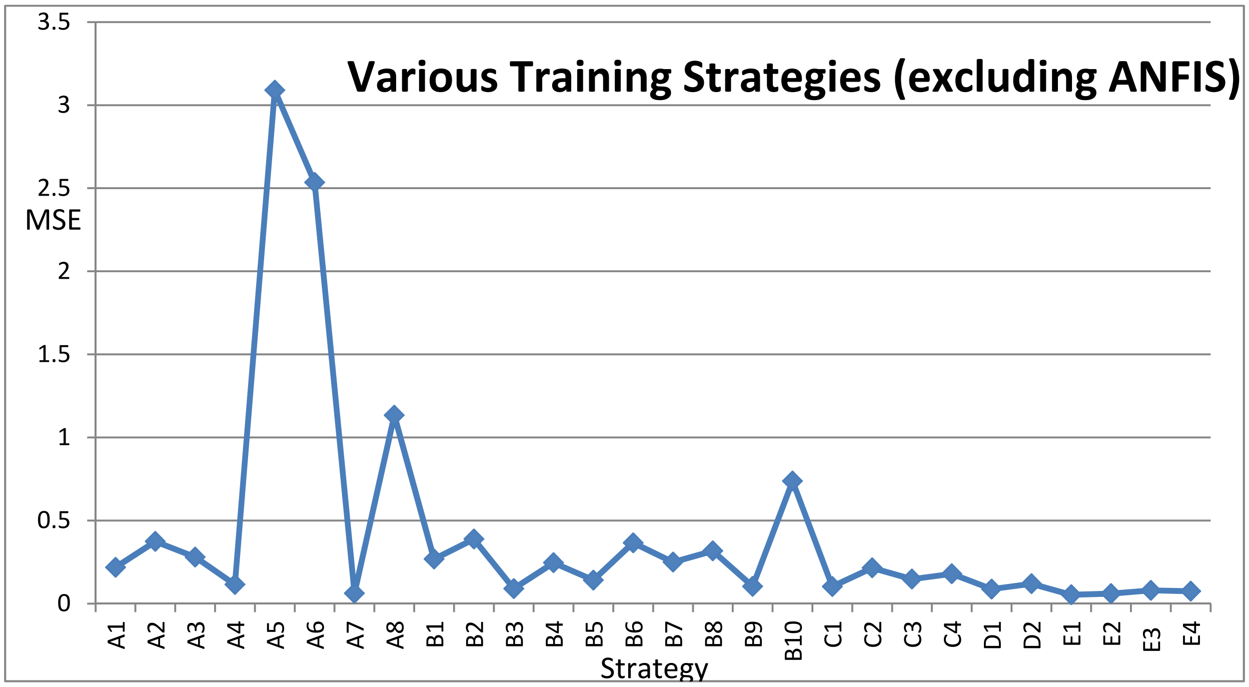


Figure 4.12 Various training strategies without using ANFIS

We can see from the Figure 4.12 that the strategies E1, E2, A7, E4, and E3 do a fairly good job in training and testing against various load buses. Now let us look at the various training methods. Since it is fairly obvious that ANFIS does not do well in testing, we ignore this method and plot the other methods in Figure 4.13 below.

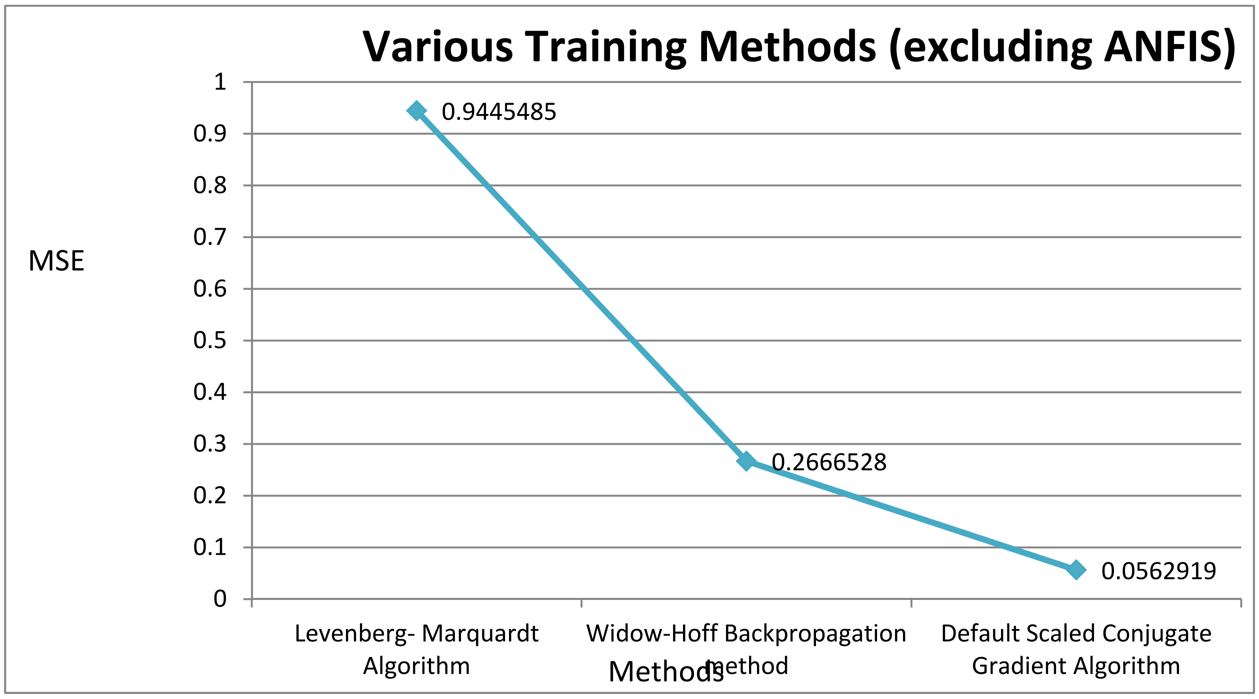


Figure 4.13 Various Training Methods excluding ANFIS method

From the diagram above, we see that the Default Scaled Conjugate Gradient algorithm method works the best here. Therefore, to train a good load model in the future, from the experience we acquired, we use Default Scaled Conjugate Gradient algorithm and E1, E2, A7, E4, and E3 strategies. For instance, the $p_{1,}$ p_2 and p_3 load models can be trained using E2 strategy and pattern recognition method below. The MSE between the simulated parameters $p_{1,}$ p_2 and p_3 and the actual parameters in each bus are shown below in Table 4.14. In this case, by looking at the average MSE values, these simulated parameters seem to closely match the actual parameters as shown below.

Table 4.14 Load model training for p_1, p_2, and p_3

E2	MSE	E2	MSE	E2	MSE
error37_p_1	0.0147255	error37_p_2	0.035449	error37_p_3	0.01957
error41_p_1	0.07612826	error41_p_2	0.073415	error41_p_3	0.0758
error42_p_1	0.09427535	error42_p_2	0.075667	error42_p_3	0.107402
error52_p_1	0.01057614	error52_p_2	0.030533	error52_p_3	0.012899
error36_p_1	0.01453636	error36_p_2	0.038002	error36_p_3	0.028068
error44_p_1	0.09376044	error44_p_2	0.078855	error44_p_3	0.112588
error43_p_1	0.11227931	error43_p_2	0.079245	error43_p_3	0.105747
error51_p_1	0.00955511	error51_p_2	0.028468	error51_p_3	0.0091
average_error_p_1	0.00119439	**average_error_p_2**	0.054954	**average_error_p_3**	0.058897

Although this section seems to draw a reasonable comparison between the strategies and methods, several shortcomings for comparing training methods include:

- Parameters and learning rate are not equivalent among these methods
- The number of training epochs are determined by effective convergence and are not equivalent
- Best training error in each method was used to do comparison

Therefore, in the next section, we will equate the parameters, number of training epochs, and record the range for methods comparison. Average MSE values, median, and range data will be recorded for all 10 trials in every method in the next section to draw a fairer comparison between the training methods.

4.5 Analyze Methods with Equivalent Training Criteria

As discussed in the ending of last section, we would work on equate the training parameters, learning rate, number of training epochs, and number of hidden neurons in each training parameters. To simplify the problem, an effective strategy E2 or load model obtained by training buses 37, 52, 36, and 51 was used for each training method. By doing so, we would have a fairer comparison between these methods. Except ANFIS, the other methods all have the following training criteria:

- The number of epochs for training is set to be 500.
- The learning rates are all set to be 0.3.

- 20 hidden neurons were used

The average MSE obtained by using trained models to predict or validating p_1, p_2 and p_3 data for all eight load buses (buses 37, 41, 42, 52, 36, 44, 43, and 51) was recorded in each of the 10 trials in Table 4.15 below. In order to better analyze the data, we calculate the average, median, maximum, minimum MSE, and range in each method and in each trial.

Table 4.15 Average MSE for p_1, p_2, and p_3 with equivalent training criteria

MSE average for its p_1, p_2, and p_3	**Fuzzy ANFIS**	**Levenberg-Marquardt**	**Widow-Hoff Backpropagation**	**Default Scaled Conjugate Gradient**
error 1	414.2555557	0.063198515	0.063860668	0.037668188
error 2	414.2555557	0.079539853	0.078574135	0.039167675
error 3	414.2555557	0.053574795	0.059783156	0.039513893
error 4	414.2555557	0.049195436	0.059881022	0.042653777
error 5	414.2555557	0.066969889	0.083464599	0.047261529
error 6	414.2555557	0.085755512	0.13119704	0.039839317
error 7	414.2555557	0.195083291	0.097044005	0.043701718
error 8	414.2555557	0.082345747	0.043798357	0.042379409
error 9	414.2555557	0.161564714	0.112865941	0.047141321
error 10	414.2555557	0.050235716	0.055168696	0.042298997
average error	414.2555557	0.088746347	0.078563762	0.042162582
median error	414.2555557	0.073254871	0.071217402	0.042339203
maximum error	414.2555557	0.195083291	0.13119704	0.047261529
minimum error	414.2555557	0.049195436	0.043798357	0.037668188
range	0	0.145887855	0.087398683	0.009593341

As the ANFIS method is fairly inaccurate, we will ignore this method for the purpose of avoiding confusion. The descending order to validate the data from the best method to the worst method based on average and median error is as follows:

- Default Scaled Conjugate Gradient
- Widow-Hoff Backpropagation
- Levenberg-Marquardet

The descending order to validate the data from the smallest range to the largest range based on is as follows:

- Default Scaled Conjugate Gradient
- Widow-Hoff Backpropagation
- Levenberg-Marquardet

Based on the given information in Table 4.14, we plot the graph for all the MSE points for these three training methods in all 10 trials below in Figure 4.14.

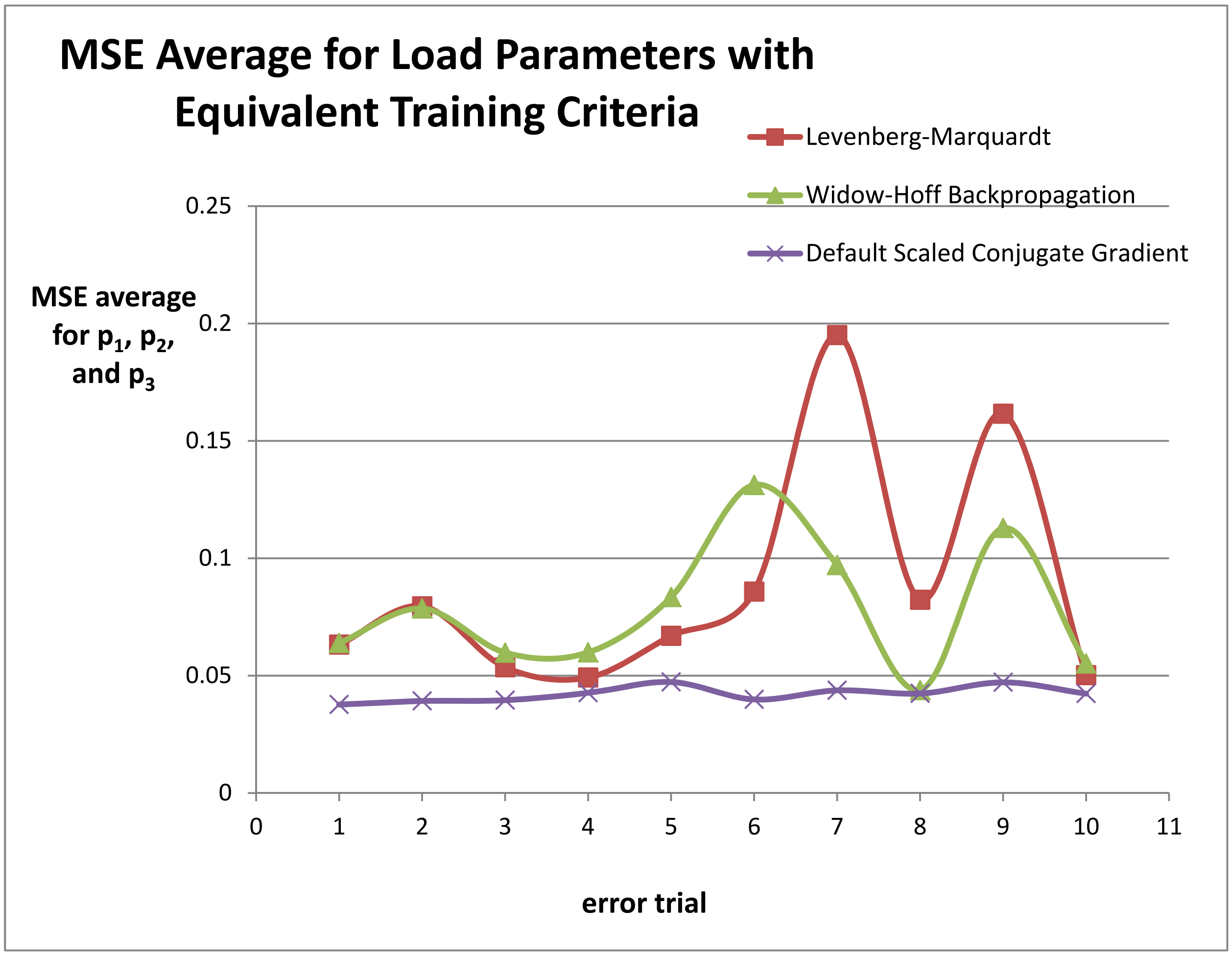


Figure 4.14 MSE average for load parameters with equivalent training criteria

From the analysis and graph above, we notice similar pattern as in Section 4.4 before. The model obtained by training Default Scaled Conjugate Gradient method works the best in validating or predicting the load parameters for unknown buses as the range is small and its median and average MSE is the smallest compared to the other methods. Therefore, we can also conclude that the descending order to validate the data from the best to the worst as:

- Default Scaled Conjugate Gradient

- Widow-Hoff Backpropagation
- Levenberg-Marquardet

4.6 Conclusion

In Chapter 4, different approaches to train and test the power system were used in order to solve the parameter identification problem. These approaches include:

- Training the load model for the simple power system with three inputs: v, P, and Q.
- Parallel training and testing of the 16-generator power system with 29 load characteristic cases, and the input are v, P or Q at 5 different time steps for each load characteristic.
- Sensitivity training and testing the more complex power system with inputs as $\Delta v, \Delta P, \Delta Q$ during five time periods as well as the initial time for v, and P or Q.
- In the final approach, $\Delta v, \Delta P, \Delta Q$ and initial time for 6 time steps were fed into the inputs for 6 times per load characteristic. The model was constructed based on observing the smallest MSE in each method.
- Lastly, equivalent training criteria including learning rate, number of epochs, , and number of hidden neurons in all training methods were established for the analysis.

Normalizing the inputs and using more strategies to train p_2 or q_2 outputs are essential for training as shown in Figure 4.6. This is also because normalizing the load values allows us to better train the load model. Throughout these trials and errors, we realized the importance to have sensitivity in changing voltages versus changing powers in the training. In our final approach, we decided to use a large set of load characteristics (15552 cases) to generate the inputs. The final approach would be the best training approach as the large set of load characteristics makes the training model more adaptable to different cases. For the approach to construct a load model based on best training mean square error, six changes in voltages with respect to the changes powers and their initial voltage and powers were fed into the training model giving the load model the ability to recognize the various time zones. After the training and testing, the results for different techniques and strategies were recorded. From Table 4.13, highlighted yellow is the average error for testing each training model on each load bus (including Buses 37, 41, 42, 52, 36, 44, 43, and 51). Furthermore, a fairer comparison between methods was conducted by equating various training criteria, and a similar pattern was observed. From studies in Sections 4.4 and 4.5, we realize that Default Scaled Conjugate Gradient method is very useful to construct load models in training and validating data for known and unknown buses. The strategies that were successful in obtaining these load parameters include A7, E1, E2, E3, or E4. From the studies, one thing to note is that the q parameters were shown to be less accurate than the p parameters in general. Since the p parameters and q parameters are equivalent to each other in order to reduce the power factor unbalance issue, we consider only the parameters p_1, p_2, and p_3 for load parameter identification. In Table 4.13 from before, we see that the testing MSE for p_2 and p_3 are fairly small as well (0.54954, 0.058897). Although the MSE for p_1 is best (or smallest), the MSE for p_2 and p_3 are acceptable. To better display the difference between parameter identifications between methods, we will show results from two training methods for parameter p_1 below. Figure 4.14 and Figure 4.15 below represent the training results from Levenberg-Marquardt and Widow-Hoff backpropagation methods respectively. From these two enlarged figures, we can see that, Widow-Hoff backpropagation algorithm method obtains better training results compared to the other one. For extended comparisons, we can refer to Appendix A.4.

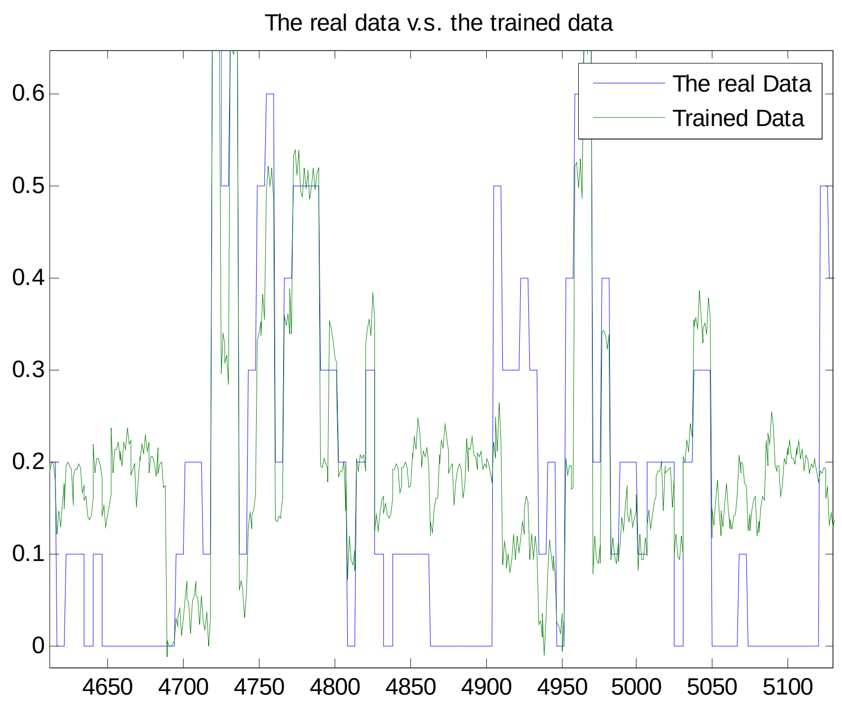


Figure 4.14 Levenbeg-Marquardt method used for validating data on Bus 41

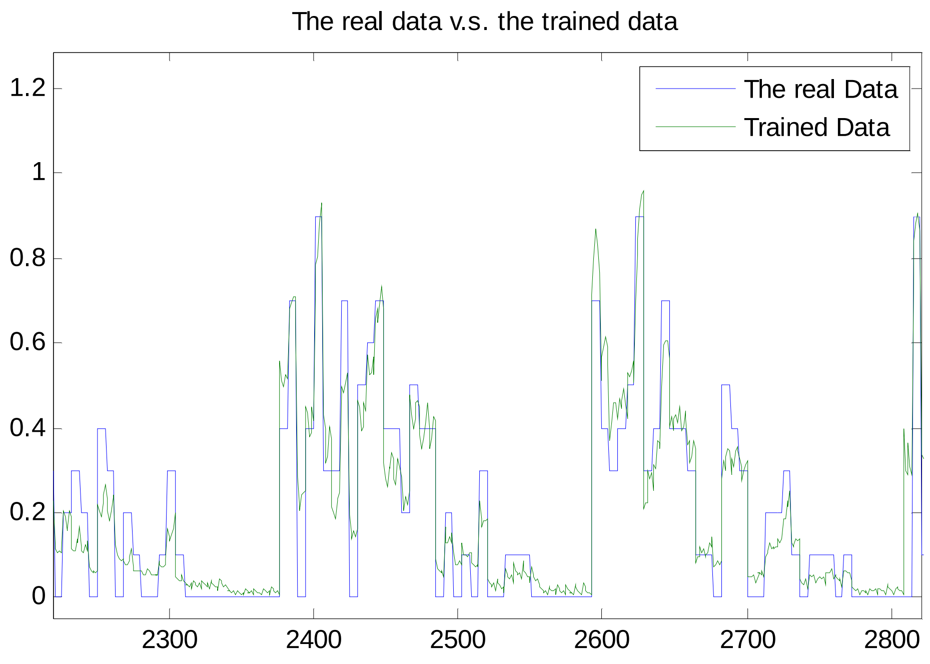


Figure 4.15 Widow-Hoff backpropagation algorithm method used for validating data on Bus 41

CHAPTER 5 - Parameter Estimation using Intelligent Techniques for Dynamic Load Models

Recall from Equations 2.11 and 2.12, the dynamic ZIP load model can be expressed as below:

$p_s = \gamma_1 * P_{ZIP} + \lambda_1 * P_{MOT}$

$Q_s = \gamma_2 * Q_{ZIP} + \lambda_2 * Q_{MOT}$

In the dynamic model, we would have to consider the amount of active/reactive induction motor load power that will be considered to be located at various busses within the system. Hence, as a starting point, training intelligent systems to yield the percentage of induction load to be modeled at a given bus will be studied in the thesis. The determining of the other parameters describing the induction motor loads will be left to future work.

In this case:

$\gamma_1 + \lambda_1 = 1$ **(5.1)**

$\gamma_2 + \lambda_2 = 1$ **(5.2)**

The equations above represent the total percentage of static ZIP power and dynamic induction motor power. In our static load model, the active and reactive parameters are assumed to be equivalent to avoid the stability issues with power factor. (i.e. $p_1=q_1$, $p_2=q_2$, $p_3=q_3$)

To make this rule consistent here or to set constant power factor as before, we will also assume $\gamma_1 = \gamma_2$ and $\lambda_1 = \lambda_2$.

From the previous static load model, we will still have 15,552 load component cases. Now, we will incorporate addition λ values in training. Note that the γ can be calculated by 1-λ. These λ values are 0.05, 0.1, 0.15, 0.175, and 0.2. These λ values are fairly small. One problem observed with a slightly larger λ value is that it causes the simulation for faults on 44-43 to have invalid voltage values at the load bus or other non-convergence problems. Therefore, these 5 λ values were chosen. With these additional λ values, the number of load component cases is 15,552 x 5=77,760 cases.

About 32.4 hours is needed to generate the inputs and outputs for each faulted load bus. After the values were obtained, an approach similar to Section 4 was used to train for the dynamic load model. The diagram for training is shown in Figure 5.1 below.

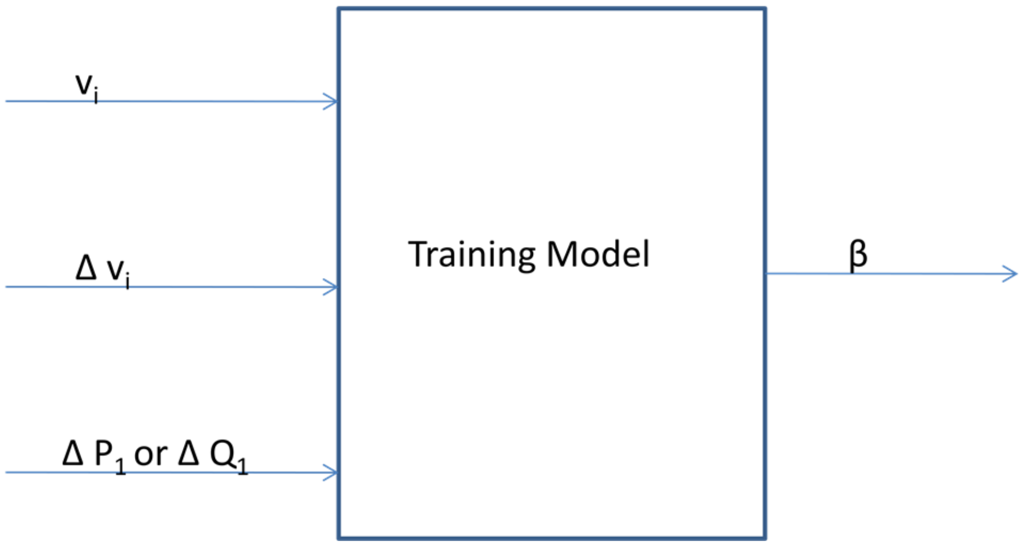


Figure 5.1 Training Model for dynamic load

The best approach, the Default Scaled Conjugate Gradient method and the best strategies E1, E2, E3, and E4 used in the static model training were used in this load model training. The MSE values obtained are displayed below in Table 5.1

Table 5.1 Training and validating MSE values for dynamic load model

Training procedures Trained on:	Tested on buses (MSE)	Default Scaled Conjugate Gradient
E1 bus(es): 41, 42, 44, 43	error37	0.105346953
	error41	0.025872072
	error42	0.049237347
	error52	0.12347474
	error36	0.145889447
	error44	0.051996439
	error43	0.051535485
	error51	0.124772609
	average error	**0.084765636**
E2 bus(es): 37, 52, 36, 51	error37	0.040199252
	error41	0.097029386
	error42	0.084719725
	error52	0.05831685
	error36	0.059616769
	error44	0.086380158
	error43	0.093925466
	error51	0.062820319
	average error	**0.072875991**
E3 bus(es): 37, 41, 36, 43	error37	0.038623464
	error41	0.02954644
	error42	0.139100605
	error52	0.122561639
	error36	0.059202945
	error44	0.192161932
	error43	0.052066952
	error51	0.138394401
	average error	**0.096457297**
E4 bus(es): 42, 52, 44, 51	error37	0.060768605
	error41	0.059869587
	error42	0.06299373
	error52	0.063834559
	error36	0.070227263
	error44	0.063485102
	error43	0.076367186
	error51	0.064627779
	average error	**0.065271726**
	MSE Row Average (averages of the average error in each)	0.079842663

We observe that the MSE average for the dynamic load in the end is worse than the MSE average for the respective static load. The intelligent techniques, however, seem to do a better job compared to the previous mathematical manipulation method such as pseudo inverse because at least feasible solutions are obtained from this approach.

CHAPTER 6 - CONCLUSIONS AND FUTURE WORK

6.1 Conclusions

In this thesis, we discussed ways to solve parameter identification problems. For instance, the simple power system was developed to work with mathematical calculations and ZIP load model verification. Although pseudo inverse and linear programming will solve for the parameters, the parameters will have infeasible solutions that do not represent the percentage of constant power, constant current and constant impedance. As a result, intelligent techniques were used to train intelligent systems that will be used to determine the models for loads that do not have information about their parameters.

Several intelligent methods were used in this research, included:

- Levenberg-Marquardt
- Widow-Hoff Backpropagation
- ANFIS (or Adaptive Network-Based Fuzzy Inference System)
- Default Scaled Conjugate gradient

A series of combined strategies were used to train and test the load buses as well. These strategies are trained on two main fault conditions. Namely load buses at fault 41-42 and fault 44-43. The buses that were trained are listed below with their symbol of representation.

- Single bus training: A1(37), A2(41), A3(42), A4(52), A5(36), A6(44), A7(43), and A8(51).
- Double bus training: B1(41, 42), B2(44, 43), B3(37,41), B4(36,44), B5(37,42), B6(36,43), B7(41,52), and B8(44, 51).
- Triple bus training: C1(37, 41, 42), C2(36, 44, 43), C3(41, 42, 52), and C4(44, 43, 51).
- Quadruple bus training: D1(36, 41, 42, 52) and D2(36, 44, 43, 51).
- Mixed bus training: E1(41,42,44,43), E2(37, 52, 36, 51), E3(37,41,36,43), and E4(42,52, 44, 51).

Through various training and testing, the Default Scaled Conjugate Gradient method and A7, E1, E2, E3, E4 training strategies are the most efficient training techniques. The dynamic load model uses the same techniques and the percentage of induction motor load can be detected using similar training techniques. Although in the end, the training and testing MSEs can be fairly small, but we have to realize that several key components need to be considered while training the set of inputs and outputs:

- Normalization needs to be applied for the similar information inputs.
- For training models that involve p_2, extra inputs from p_3 and p_1 need to be fed into the training models for $p_{2.}$
- Large sets of load data need to be used to make the model efficient for use.
- The sensitivities of voltages, active/reactive powers need to be used for the inputs of the training model.
- ANFIS method does not seem to be a good method for validating the data although it trains well against itself. Therefore, ANFIS method is not recommended for this research.

Throughout the research, we tried many different ways to train and test the systems. The final approach seems to be a fairly reasonable one. The results in Table 4.13 and Table 4.15 demonstrate that there are at least several methods and strategies which are fairly reliable in training and testing of load bus models. The Figure 4.12, Figure 4.13, and Figure 4.14 further elaborate the best to worst training methods to obtain a power load model. However, by choosing the right techniques, we still have to be careful in training and testing the data as some data are not sensitive to the intelligent techniques and therefore will not predict the unknown data from unknown buses readily. Overall, this research helps to identify the approaches and ways to solve load parameters identification problems. It can be used to construct load models, and it can also be used to predict the load characteristic for unknown buses. It will become useful when we use the constructed load models through right training to predict data in unknown buses in a real power system, and to take actions to possibly prevent any power system related failures from taking place.

6.2 Recommendations and Future Work

Sensitivities in voltages and active and reactive powers have been used extensively in this load model training. However, we regretted that the more recent techniques such as genetic algorithms were not used in this research. This is something that can be used in the future or maybe even a combination of artificial intelligence and genetic algorithm techniques can be used in training. Moreover, the assumption of parameters $p_1=q_1$, $p_2=q_2$, and $p_3=q_3$, $\alpha_1=\alpha_2$, and $\beta_1=\beta_2$ limit the changing load parameters in this thesis. Although by doing so, we would not have to worry about the power factor related voltage stability issues, the model also becomes less effective in a real complex system. This can also be said for including frequency variations parameters that we neglected in our research. Therefore, in the future, we encourage future students who are interested in this field to choose a varying power factor power system load and adding frequency components for load model training. In addition, real data is encouraged to be used in this load modeling research. It will be a big accomplishment to use a developed or even a more enhanced load model to test and validate on a real power system in the future.

References

[1] *Canada-U.S. Power System Outage Task Force*. Nov. 2003. Web. Aug. 2010. <http://www.knowledgerush.com/kr/encyclopedia/Canada-U.S._Power_System_Outage_Task_Force/>.

[2] Martins, N., Paserba, J., Pourbeik, P., Sanchez-Gasca, J., Schulz, R., Stankovic, A., Taylor, C., Vittal, V. (2003). *Causes of the 2003 major grid blackouts in North America and Europe, and recommended means to improve system dynamic performance*, Power Systems, IEEE Transactions, 20(4), 1922 – 1928, doi: 10.1109/TPWRS.2005.857942

[3] Kundur, P., Balu., N.J., Lauby, M.G. (1994). *Power System Stability and Control*. University of Michigan.

[4] Machowski, J., Bialek, J.W., Bumby, J.R. (1997). *Power System Dynamics and Stability*. Jonh Wiley & Sons. England, 235-253.

[5] (1990). IEEE Stability Special: *Voltage Stability of Power Systems: Concepts, Analytical Tools and Industry Experience*, IEEE Special Publication, doi: 90TH0358-2-PWR

[6] Taylor, C.W. (1994). *Power System Voltage Stability*, Electric Power Research Institute, McGraw-Hill, 17-135

[7] Karlsson, D., Hill, D.J. (1994). *Modeling and identification of nonlinear dynamic loads in power systems*. IEEE Transactions on Power Systems, 9(1), 157-166

[8] Willis, H.L.. Finley, L.A., Buri, M.J. (1995). *Forecasting Electric Demand of Distribution Planning in Rural and Sparsely Populated Regions*, IEEE Transactions on Power Systems, 10(4), 2008-2013, doi: 10.1109/59.477100

[9] Ma, J., Dong, Z., He, R., Hill, D.J. (2007). *Measurement-based Load Modeling using Genetic Algorithms*. IEEE Congress on Evolutionary Computation, 2909 - 2916. doi: 10.1109/CEC.2007.4424841

[10] Shi, J.H., Renmu, H. (2003). *Measurement-based Load Modeling - Model Structure*. 2003 IEEE Bologna PowerTech Conference, 2, 1-5. doi: 10.1109/PTC.2003.1304621

[11] Mota, L.T.M., Mota, A.A. (2004). *Load modeling at electric power distribution substations using dynamic load parameters estimation*. International Journal of Electrical Power And Energy Systems, 26(10), 805-811, doi:10.1016/j.ijepes.2004.07.002

[12] Bai, H., Zhang, P., Ajjarapu, V. (2009). *A Novel Parameter Identification Approach via Hybrid Learning for Aggregate Load Modeling*, IEEE Transactions on Power Systems, 24(3), 1145-1154, doi: 10.1109/TPWRS.2009.2022984

[13] Wen, J.Y., Jiang, L., Wu, Q.H. Cheng, S.J. (2003). *Power system load modeling by learning based on system measurement*. IEEE Transactiosn on Power Delivery, 18(2), 364-371. doi: 10.1109/TPWRD.2003.809730J

[14] Li, X., Wang, L., Li, P. (2008). *The Study on Composite Load Model Structure of Artificial Neural Network*. Electric Utility Deregulation and Restructuring and Power Technologies, 1564-1570. doi:10.1109/DRPT.2008.4523654

[15] Xu, W., Vaahedi, E., Mansour, Y., Tamby, J. (1997). *Voltage Stability Load Parameter Determination from Field Tests on B.C. Hydro's system*, Power Systems, IEEE Transactions on, 12(3), 1290-1297. doi: 10.1109/59.630473

[16] Coker, M., Kgasoane, H. (1999). *Load Modeling*, SAPSSI, ESKOM Technology Group, 663-668.

[17] Hiyama, T., Tokieda, M., Hubbi, Walid. (1997). *Artificial neural network based dynamic load modeling*, Power Systems, IEEE Transactions on, 12(4), 1576-1583. doi: 10.1109/59.627861

[18] Thukaram, D., Kamalasadan, S., Ghandakly, *A Matlab based artificial neural network algorithm for voltage stability assessment*, University of Toledo, 1-4.

[19] Hsu, Y., Lu, F. (1998). *A combined Artificial Neural Network – Fuzzy Dynamic Programming approach to reactive power voltage control in a distribution*, National Taiwan University, 1265-1271.

[20] Oonsivilai, A., El-Hawary, M.E. (1999). *Power System Dynamic Load Modeling using Adaptive-Network-Based Fuzzy Inference System*, Electrical and Computer Engineering, 1999 IEEE Canadian Conference on, 3, 1217-1222. doi: 10.1109/CCECE.1999.804864

[21] Radaideh, O.Y. (2003). *Selection of Load Node Model in Power Systems: Fuzzy Logic Approach*, Asian Network for Scientific Information, 2(2), 148-153.

[22] (1993). IEEE Task Force on Load Representation for Dynamic Performance, *Load representation for dynamic performance analysis*, IEEE Trans. Power Syst., 8(2), 472–482.

[23] Makarov, Y., Maslennikov, V., and Hill, D. (1996). *Revealing loads having the biggest influence on power system small disturbance stability*, IEEE Trans. Power Syst., 11(4), 2018–2023

[24] Craven, R H., George, T., Price, G.B., (1994). *Validation of dynamic modeling methods against power system response to small and large disturbances*

[25] (1995). IEEE Task Force on Load Representation for Dynamic Performance: *Standard load models for power flow and dynamic performance simulation*, IEEE Trans. Power Syst., 10(3)

[26] Kosterev, D.N., Taylor, C.W., Mittelstadt, W.A. (1999). *Model validation for the August 10, 1996 WSCC system outage*, IEEE Trans. Power Syst., 14(3), 967–979, doi: 10.1109/59.780909

[27] Kao, W.S., Lin, C.J., Huang, C.T., Chen, Y.T., Chiou, C.Y. (1994). *Comparison of simulated power system dynamics applying various load models with actual recorded data*, 9(1), 248-254, doi: 10.1109/59.317604

[28] (2006). *Load Model Parameter Derivation Based on Measurement*. Palo Alto, CA: EPRI

[29] (1979). *Determining Load Characteristics for Transient Performances*, EPRI EL-849, 3, Project 849-3

[30] Lin, C.-J., Chen, A.Y.-T., Chiou, C.-Y., Huang, C.-H., Chiang, H.-D., Wang, J.-C., Fekih-Ahmed, L. (1993). *Dynamic load models in power systems using the measurement approach*, IEEE Trans. Power Syst., 8(1), 309–315

[31] Nelles, O. (2001). *Nonlinear System Identification: From Classical Approaches to Neural Networks and Fuzzy Models. Heidelberg*, Germany

[32] Shackshaft, G., Symons, O.C., Hadwick, J.G. (1997). *General purpose model of power system loads*, Proc. Inst. Elect. Eng., 124, 715–723

[33] Qisheng, L., Yunping, C., Dunfeng, D., (2002). *The load modeling and parameter identification for voltage stability analysis*, Int. Conf. Power System Technology, 4(4), 2030–2033, doi: 10.1109/ICPST.2002.1047137

[34] Hiskens, I. A. (2001). *Nonlinear dynamic model evaluation from disturbance measurements*, IEEE Trans. Power Syst., 16(4), 702–710

[35] Zhu, S.Z., Zheng, J.H., Shen, S.D., Luo, G.M. (2000). *Effect of load modeling on voltage stability*, IEEE Power Eng. Soc. Summer Meeting, 1, 395–400, doi: 10.1109/PESS.2000.867617

[36] Kamoun S., Malham, R. P. (1992). *Convergence characteristics of a maximum likelihood load model identification scheme*, Automatica, 28(5), 885–896.

[37] De Kock, J.A., Van der Merwe, F.S., Vermeulen, H.J. (1994). *Induction motor parameter estimation through an output error technique*, IEEE Trans. Energy Convers., 9(1), 69–76

[38] Knyazkin, V., Canizares, C.A., Soder, L.H. (2004). *On the parameter estimation and modeling of aggregate power system loads*, IEEE Trans. Power Syst., 19(2), 1023–1031, doi: 10.1109/TPWRS.2003.821634

[39] Ju, P., Handschin, E., Karlsson, D. (1996). *Nonlinear dynamic load modeling: Model and parameter estimation*, IEEE Trans. Power Syst., 11(4), 1689–1697, doi: 10.1109/59.544629

[40] Jazayeri, P., Rosehart, W., Westwick, D.T. (2007). *A multistage algorithm for identification of nonlinear aggregate power system loads*, IEEE Trans. Power Syst., 22(3), 1072–1079

[41] Karlsson, D., Hill, D.J. (1994). *Modeling and identification of nonlinear dynamic loads in power systems*, IEEE Trans. Power Syst., 9(1), 157–166

[42] (1997). *Power System Toolbox for Matlab*, Cherry Tree Scientific Software

[43] (2011). *Fuzzy Logic Toolbox User's Guide.* Retrieved. July 11, 2011, from The MathWorks: Accelerating the pace of engineering and science ,Web Site: http://www.mathworks.com/access/helpdesk/help/pdf_doc/fuzzy/fuzzy.pdf

[44] (2011). *Neural Network Toolbox User's Guide*. Retrieved. June 11, 2011, from The MathWorks: Accelerating the pace of engineering and science, Web Site: http://www.mathworks.com/access/helpdesk/help/pdf_doc/nnet/nnet.pdf

[45] Chen, W., Gong, Q.W., Zhang, L., Yanng, M. (2008). *Parameters identification of twelve-phase synchronous generator based on Levenberg-Marquardt algorithm*, International Conference on Electrical Machines and Systems, 3992-3995

[46] Stan, O., Kamen, E.W. (1999). *New block recursive MLP training algorithms using the Levenberg-Marquardt algorithm*, 1672-1677, doi: 10.1109/IJCNN.1999.832625

[47] Bartkowiak, A. (2004). *Neural Networks and Pattern Recognition*, 1-56

[48] Rpley, B.D. (1996), Pattern Recognition and Neural Networks, 1-401

[49] Tchaban, T. , Griffin, J.P., Taylor, M.J. (1997). *A comparison between single and combined backpropagation neural networks in the prediction of turnover*, First International Conference on Knowledge-Based Intelligent Electronic Systems, 347-354, 2, doi: 10.1109/KES.1997.619408

[50] Sugiyama, S. (1995). *Twofold type of backpropagation neural network*, Conference on Neural Networks, 1535-1540, 3, doi: 10.1109/ICNN.1995.487391

[51] Bishop, C.M. (1995), *Neural networks for pattern recognition,* 1-477

[52] Jang, J.R. (1993), ANFIS: Adaptive-Network-Based Fuzzy Inference System, IEEE transactions on systems, man, and cybernetics, 23(3), 1-21

[53] He, S., Starrett, S. (2009). *Modeling Power System Load using Adaptive Neural Fuzzy Logic and Artificial Neural Networks*. NAPS 2009, 1-5.

[54] CIGRE TF 38-02-10 (1993), Modeling of Voltage Collapse Including Dynamic Phenomena

[55] Morison, K., Hamadani, H., Wang, L. (2007). *Load Modeling for Voltage Stability Studies*, Power Systems Conference and Exposition 2006, 564-568, doi: 10.1109/PSCE.2006.296379

[56] (2011). *Conjugate Gradient Algorithms.* Retrieved. June 11, 2011, from Neural Network Toolbox, Web Site: http://www.kxcad.net/cae_MATLAB/toolbox/nnet/backpr14.html

Appendix A - Graphs Generated by the Simulations

A.1 Individual training v.s. Combined Training

Load bus 41 training with each parameter, and the combined outputs training

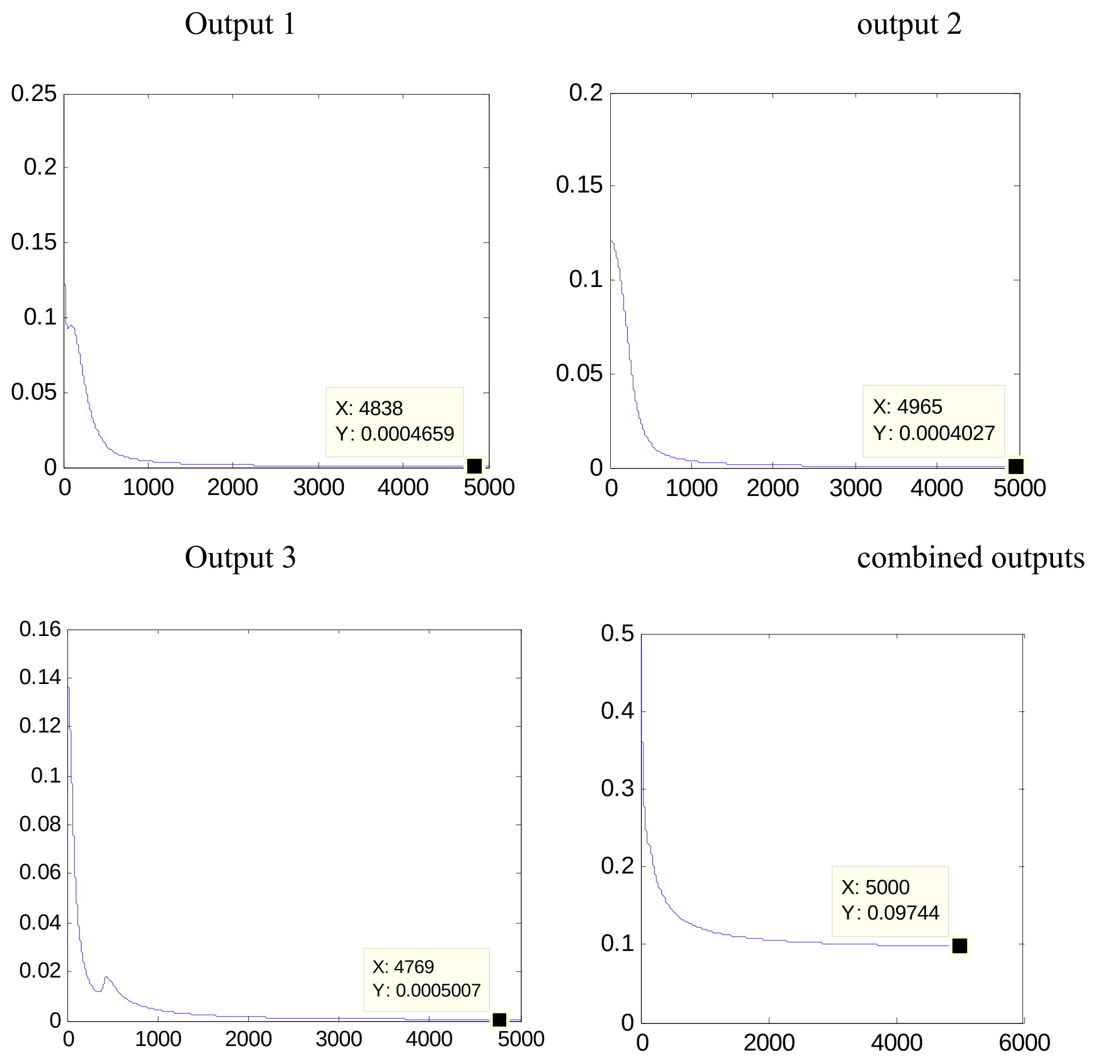


Figure A.1 The individual and combined training for bus 41

A.2 Simple Power System Training and validating

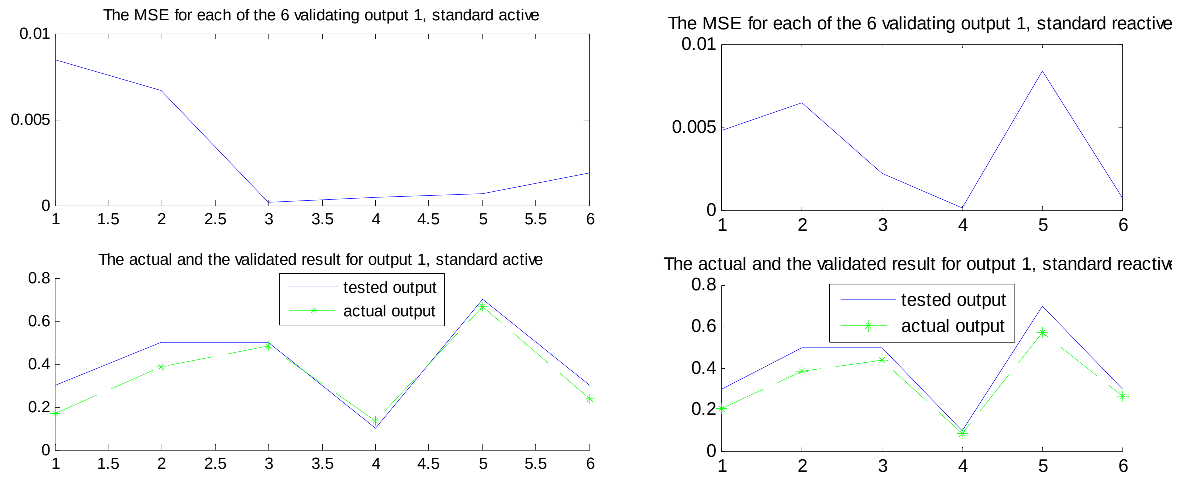


Figure A.2 The simple power system training for output 1

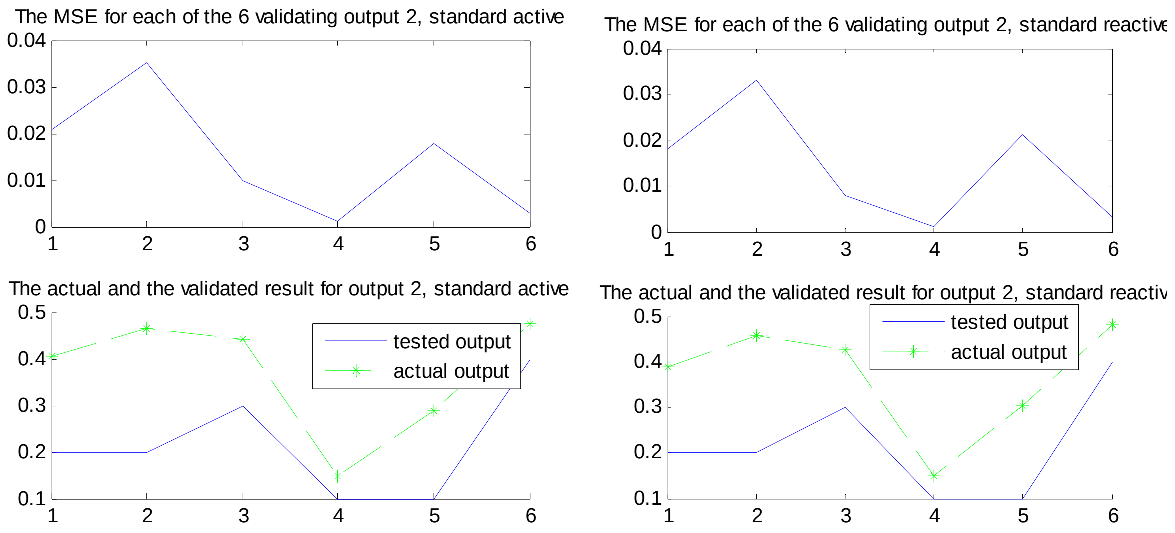


Figure A.3 The simple power system training for output 2

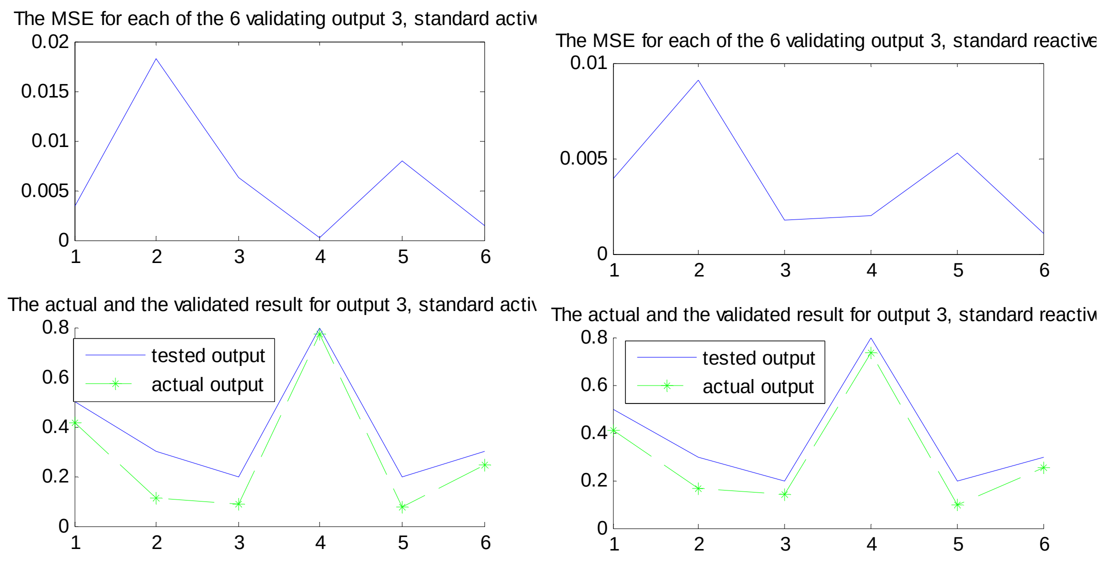


Figure A.4 Simple power system training for output 3

A.3 Training and validating on Bus 41

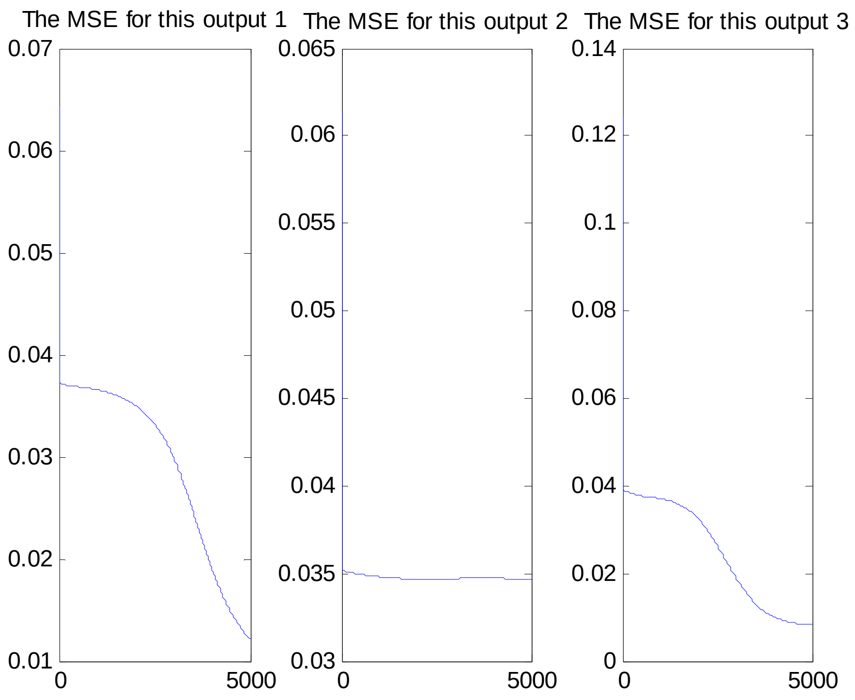


Figure A.5 Training for Bus 41 output 3

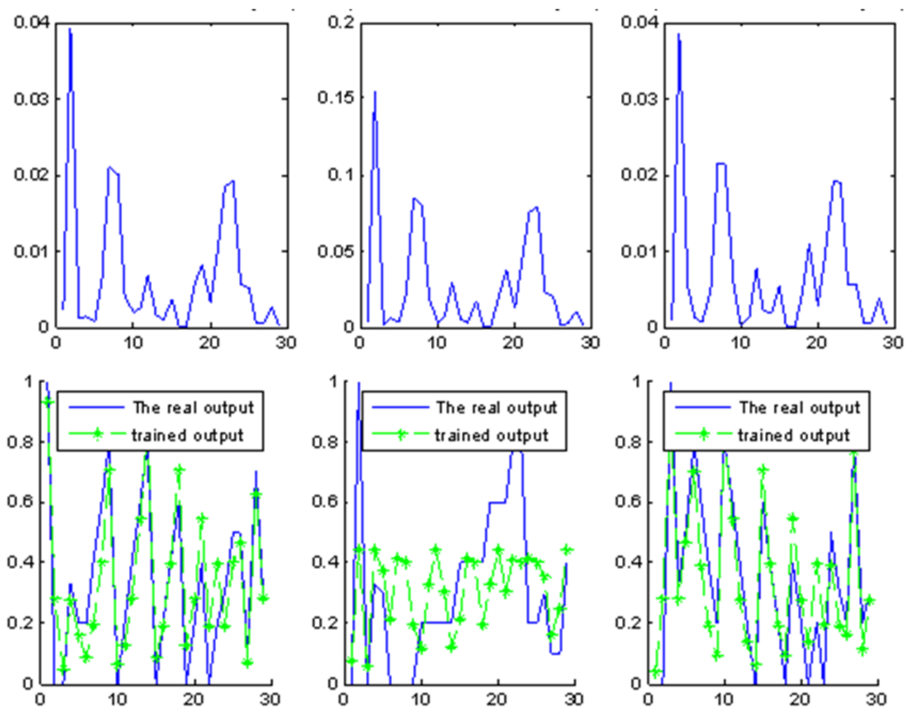


Figure A.6 validating for Bus 41 for output 1 to output 3 from left to right

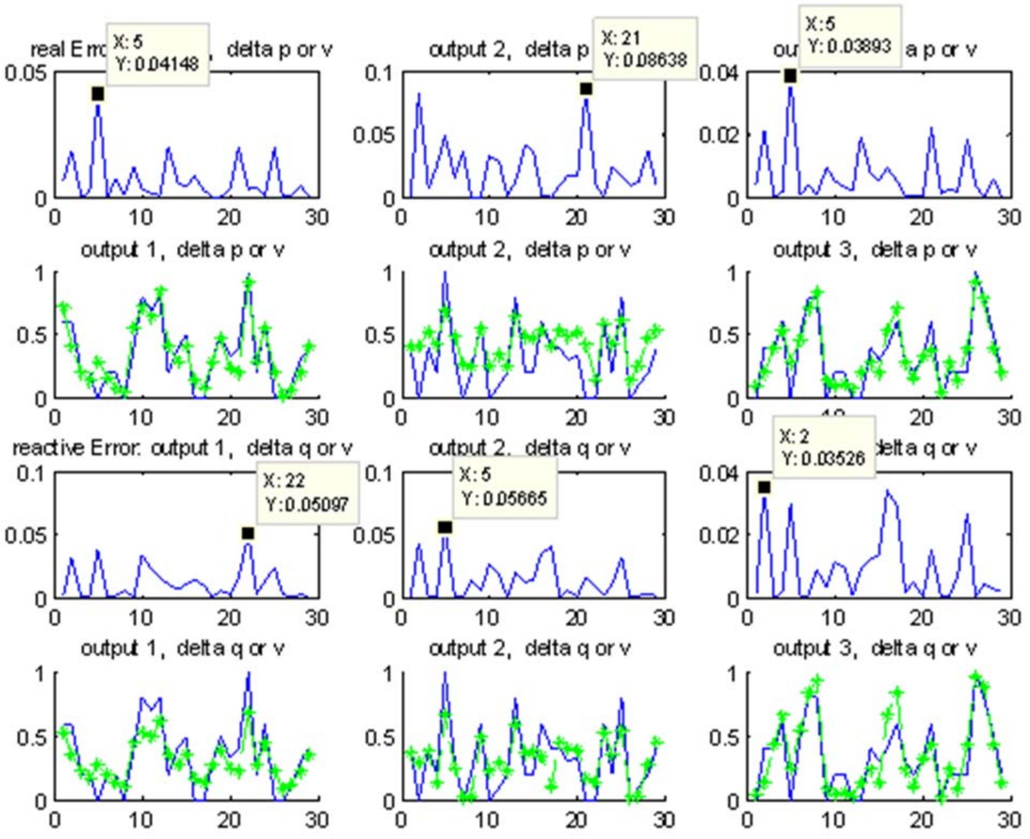


Figure A.7 Validation on bus 1 using trained model for bus 41

A.4 Intelligent methods on Bus 41

ANFIS training on fault 41-42 with load bus information at Bus 41

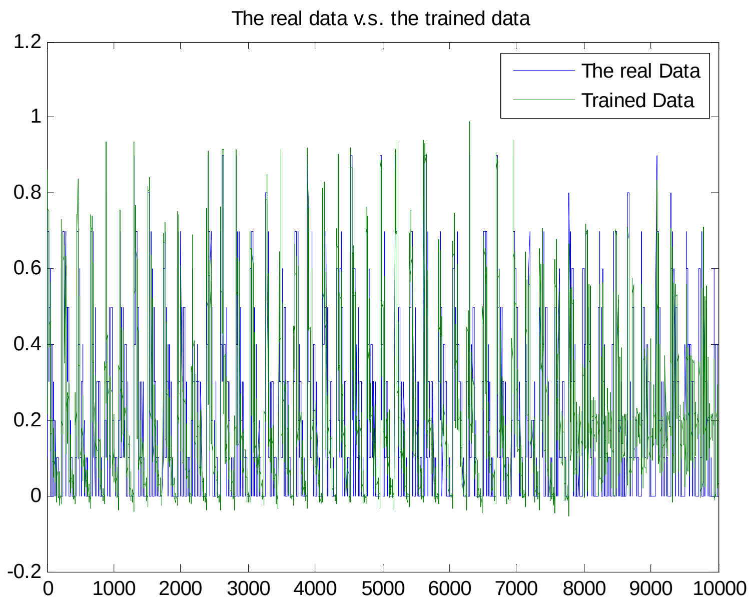


Figure A.8 Enlarged training vs real data for Bus 41

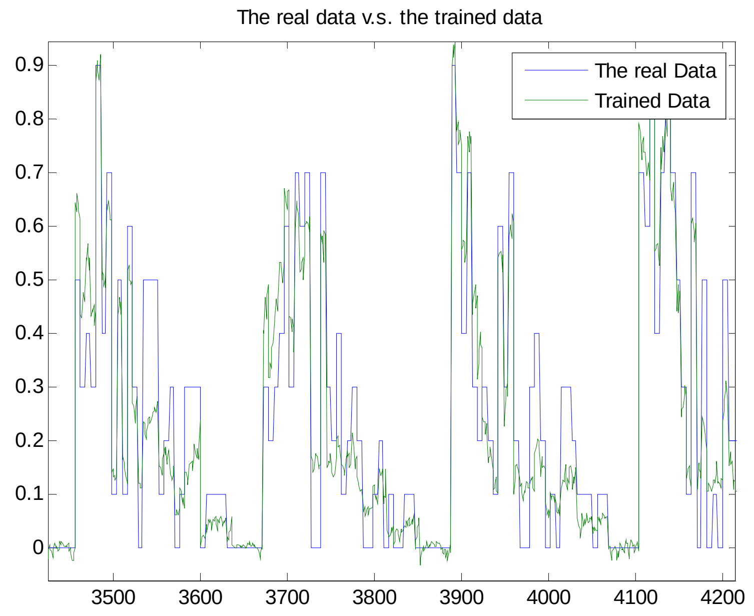


Figure A.9 Normal view for training vs real data for Bus 41

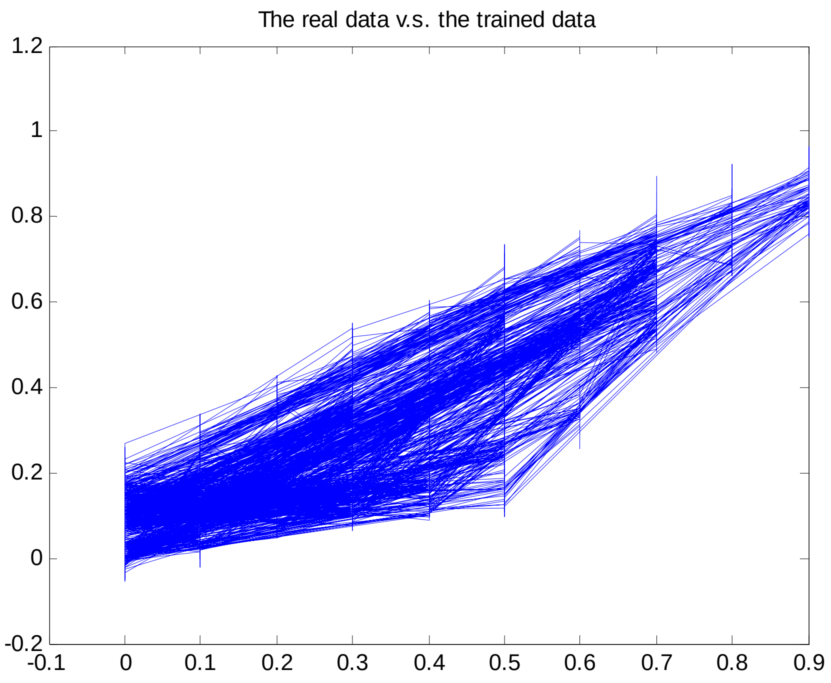


Figure A.10 The plot for real and trained data in 2-axis for bus 41

With the trained info, we test the Bus 41results using the remaining 5552 points:

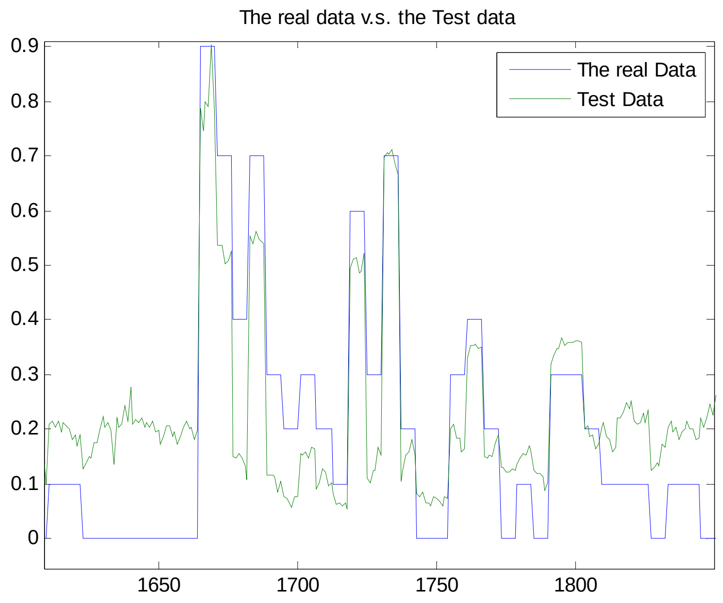


Figure A.11 The tested real data v.s. test data for load bus 41 (enlarged version)

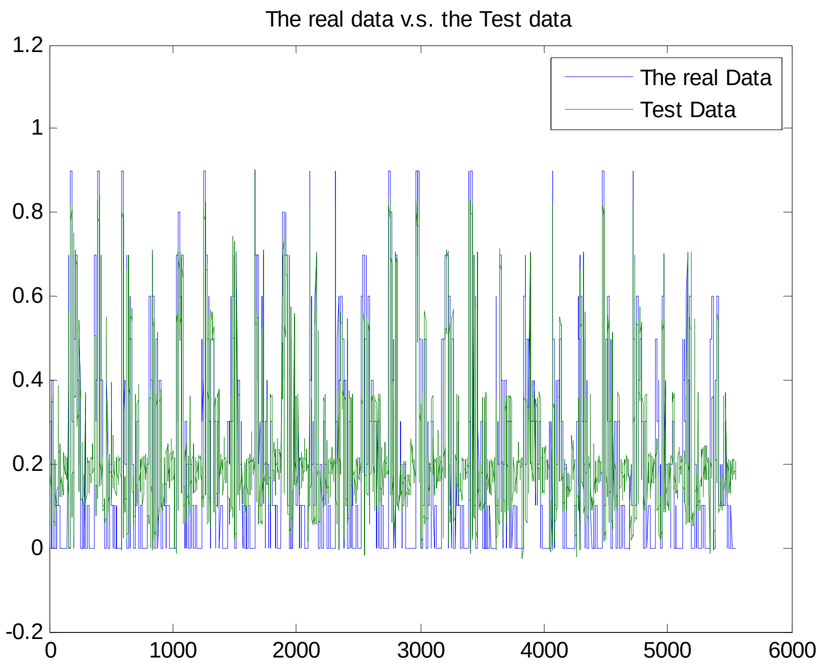


Figure A.12 The tested real data v.s. test data for load bus 41

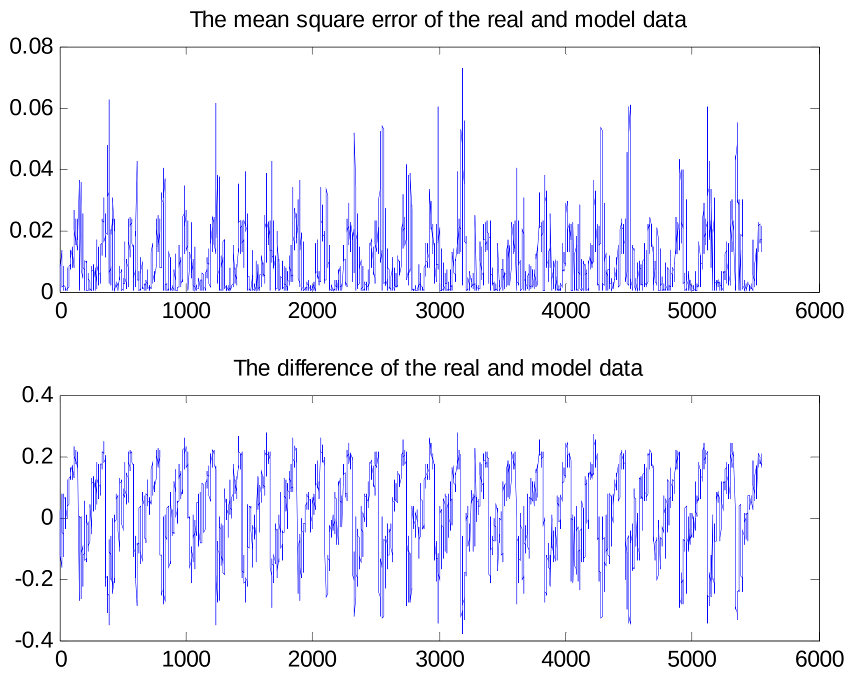


Figure A.13 The MSE for tested data and the real data

Levenberg-Marquardt Method

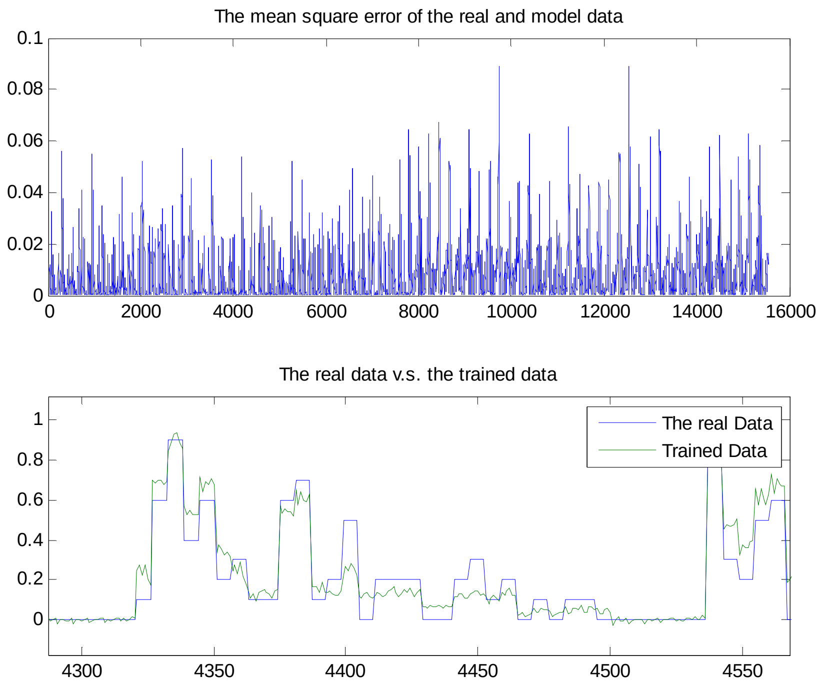


Figure A.14 Levenberg-Marquardt method trained on 10000 points

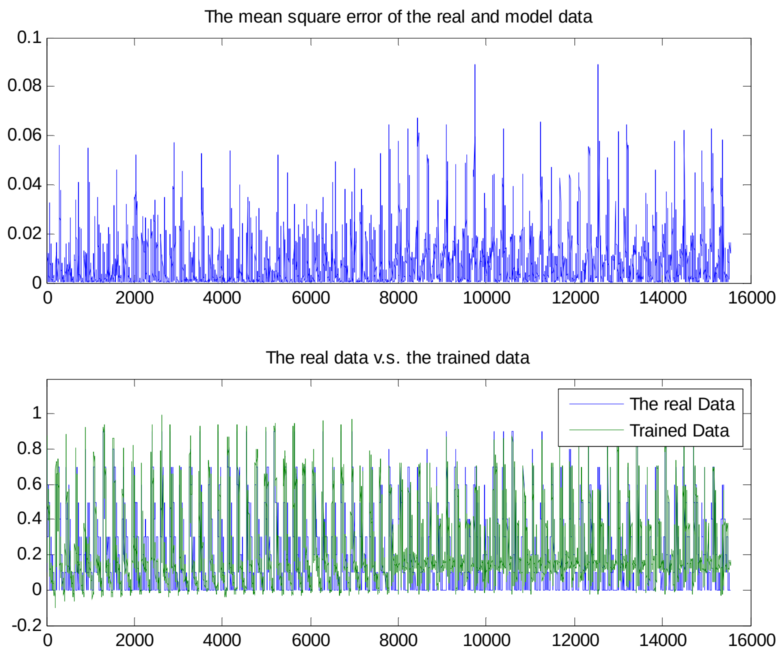


Figure A.15 MSE for Levenberg-Marquardt training

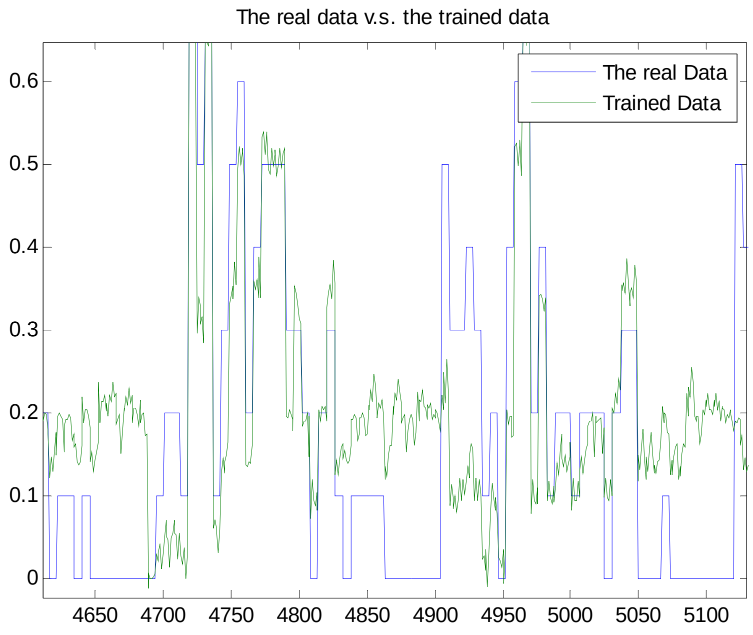


Figure A.16 Levenberg-Marquardt method validate on the 5552 points (enlarged)

ANN backpropagation full training on 10000 points

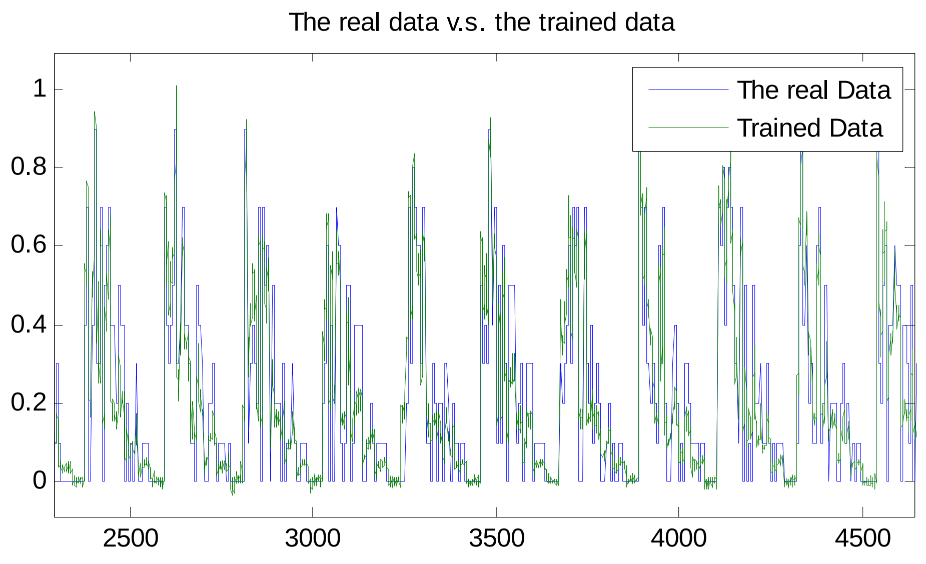


Figure A.17 ANN backpropagation traininig

Backpropagation tested on 5552 points:

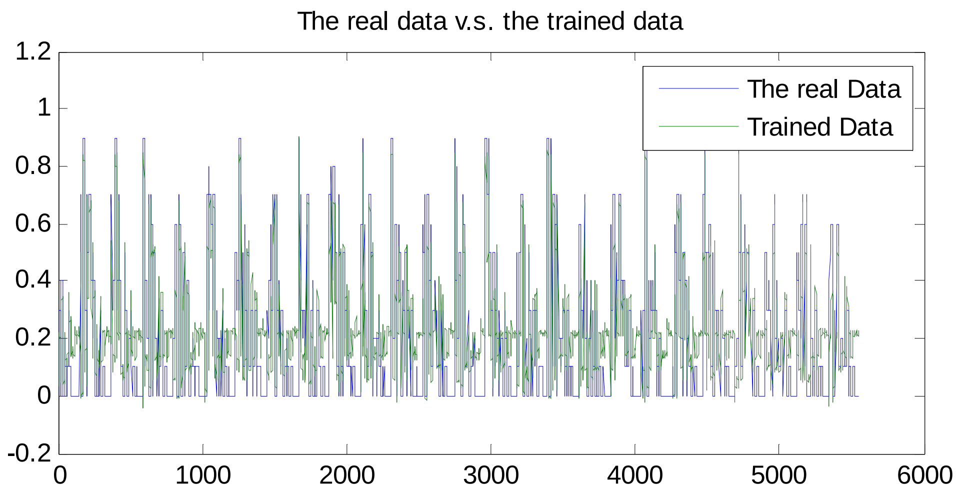


Figure A.18 ANN backpropagation testing

Pattern recognition, full trained on 10000 points

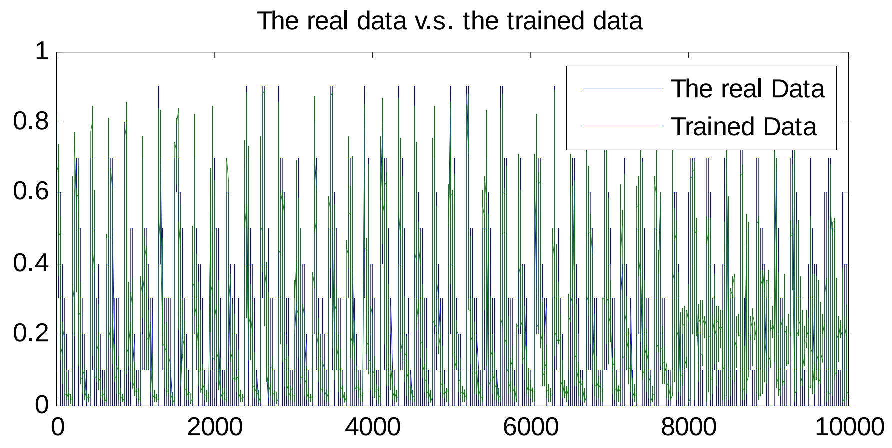
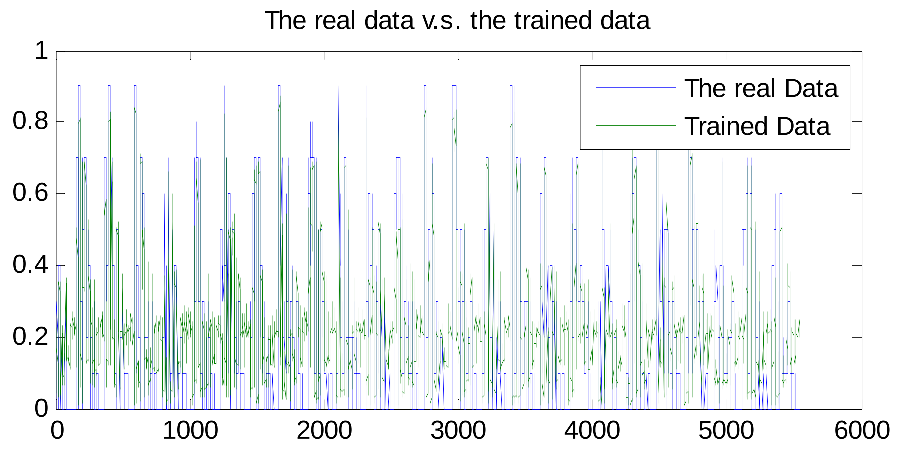


Figure A.19 Default Scaled Conjugate Gradient training

Default Scaled Conjugate Gradient, validated on 5552 points



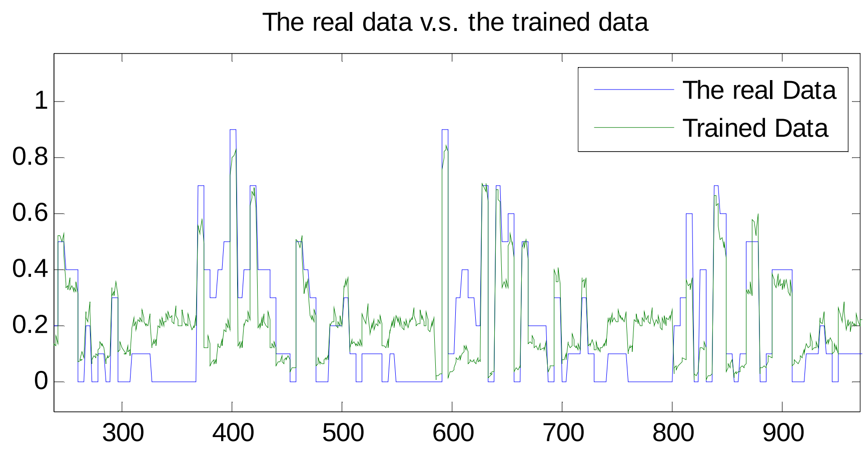
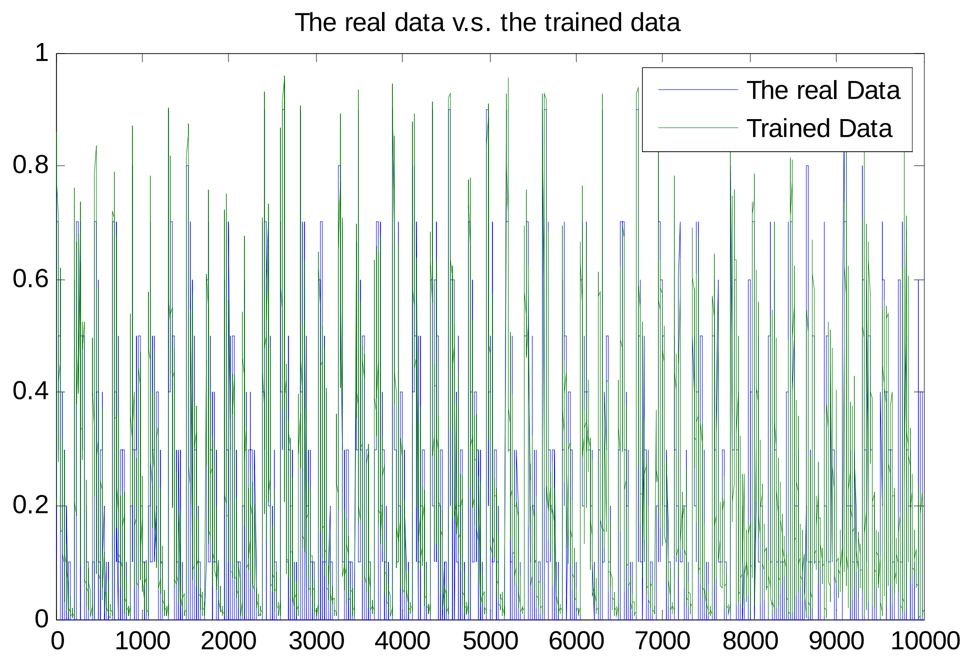


Figure A.20 Default Scaled Conjugate Gradient testing

Widow-Hoff backpropagation algorithm code, trained on the first 10000 points



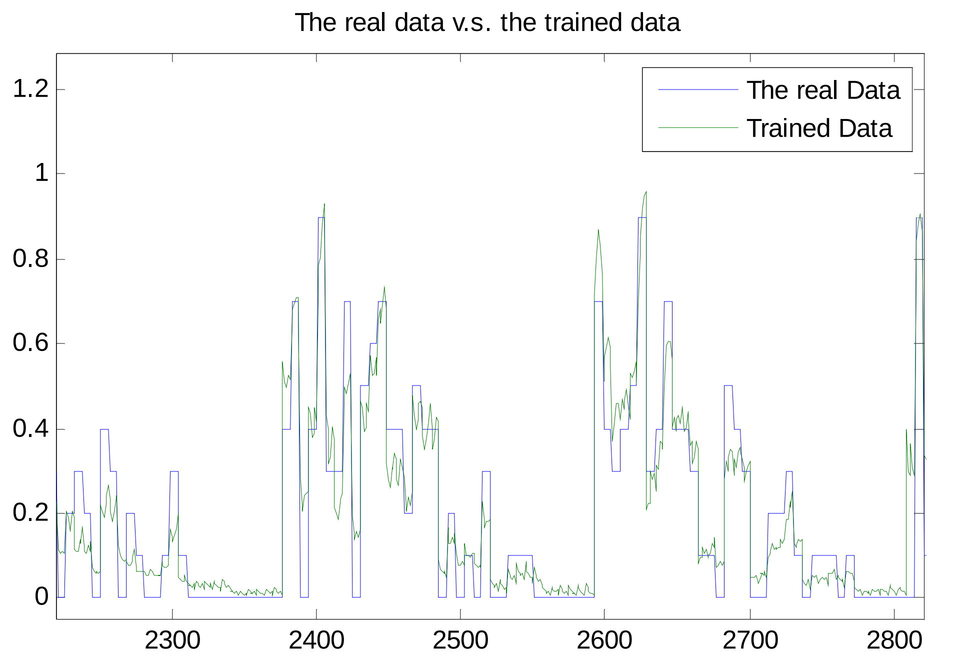


Figure A.21 Widow-Hoff backpropagation training

Widow-Hoff backpropagation validated on the later 5552 points

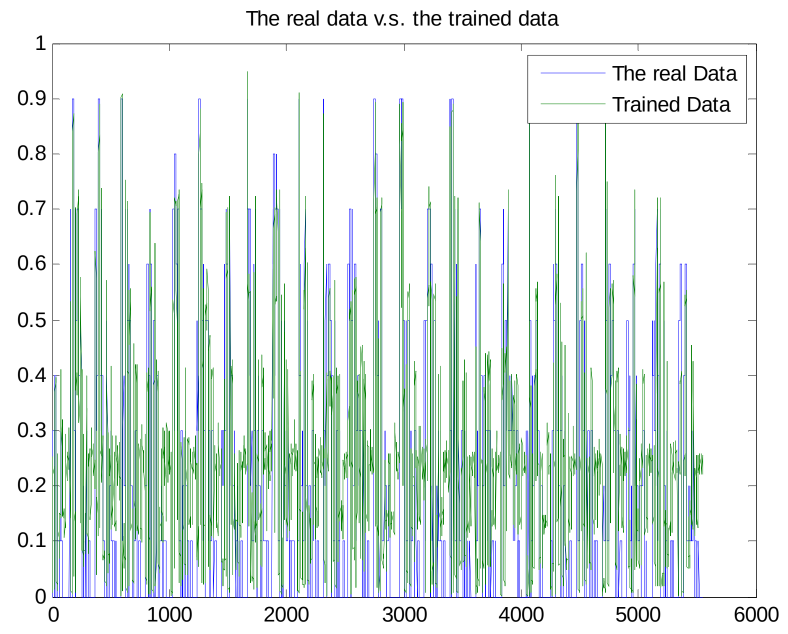


Figure A.22 Widow-Hoff backpropagation testing

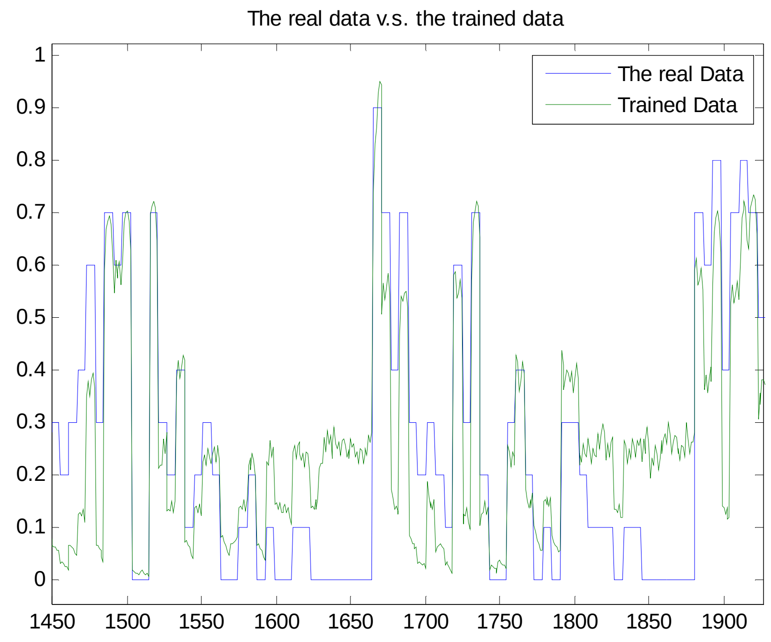


Figure A.23 Widow-Hoff backpropagation testing (enlarged)

Appendix B - Software Code

B.1 Code for Pseudo Inverse method and Inverse Matrix

```
% Input and output data manipulating for the

%This only caucluates the stuff on bus 3 for trip at line 3-18
%Section 1
% data_combine will include all the training sets with iter =
% focus at bus 3, and the 3-phase fault at line 3-18
%  0 constant impedance
%  1 single constant current load
%  2 single constant power  load
%  3 mixed single load
%  4 multiple constant current load
%  5 multiple constant power load
%  6 mixed multiple loads
%  7 yet another mixed loads

%Section 2
%data_combine, the columns are
% 1 - voltage
% 2 - voltage^2
% 3 - frequency
% 4 - real power
% 5 - reactive power
% 6 - time steps t
% line information
% record line information "S1_combine (from), S2_combine (away)"  for 1 to 86
lines

%iter=0 for the first case and so on see Section 1 above
iter=0;

data_combine(1+339*iter:339+339*iter, 1)=abs(bus_v(3, 1:339));
data_combine(1+339*iter:339+339*iter, 2)=abs(bus_v(3, 1:339)).*abs(bus_v(3,
1:339));

frequency(1)=0;
for index= 2 : 339
    frequency(index)=(angle(bus_v(3, index))-angle(bus_v(3, index-
1)))/(t(index)-t(index-1))/2/pi;
end

data_combine(1+339*iter:339+339*iter, 3)=frequency;

data_combine(1+339*iter:339+339*iter, 4)=-real(S1(6, :))-real(S1(7, :))-
real(S2(3, :));

data_combine(1+339*iter:339+339*iter, 5)=-imag(S1(6, :))-imag(S1(7, :))-
mag(S2(3, :));
```

```
data_combine(1+339*iter:339+339*iter, 6)=t;
%pseudo inverse method

b=data_combine(:, 4)

br=data_combine(:, 5)

A(:,1)=data_combine(:, 2)
A(:,2)=data_combine(:, 1)
A(1:339,3)=1

x=pinv(A)*b

y=pinv(A)*br

%inverse matrix method, 3 points at time steps 119, 158, 171 were chosen

index=[119 158 171];
for i=1:3
    b(i,1)=data_combine(index(i), 4);
    br(i,1)=data_combine(index(i), 5);
    A(i,1)=data_combine(index(i), 2)
    A(i,2)=data_combine(index(i), 1)
end

A(1:3,3)=1;

x=inv(A)*b

y=inv(A)*br
```

B.2 Generate 15552 cases of load_con matrix for fault condition

```
% p3 values for load buses
clear all;
clc;

index=[0 0.2 0.4 0.6 0.8 1];

%first column for p3 values
for i=1:216
    k=i-1;
    p3values(1+k*6:6+k*6,1)=index(1:6);
end

%second column for p3 values
for i=1:36
    k=i-1;
    p3values(1+k*36:6+k*36,2)=index(1);
    p3values(7+k*36:12+k*36,2)=index(2);
    p3values(13+k*36:18+k*36,2)=index(3);
    p3values(19+k*36:24+k*36,2)=index(4);
    p3values(25+k*36:30+k*36,2)=index(5);
    p3values(31+k*36:36+k*36,2)=index(6);
end

%third column for p3 values
for i=1:6
    k=i-1;
    p3values(1+k*216:36+k*216,3)=index(1);
    p3values(37+k*216:72+k*216,3)=index(2);
    p3values(73+k*216:108+k*216,3)=index(3);
    p3values(109+k*216:144+k*216,3)=index(4);
    p3values(145+k*216:180+k*216,3)=index(5);
    p3values(181+k*216:216+k*216,3)=index(6);
end

%fourth column for p3 values
    p3values(1:216,4)=index(1);
    p3values(217:432,4)=index(2);
    p3values(433:648,4)=index(3);
    p3values(649:864,4)=index(4);
    p3values(865:1080,4)=index(5);
    p3values(1081:1296,4)=index(6);


for i=1:4
  for j=1:1512
    total_allowed=10-p3values(j,i)*10;
    if total_allowed==0
        p2values(j,i)=0;
    else
        p2_integer=randi(total_allowed);
        p2values(j,i)=p2_integer/10;
    end
  end
end

%Assign p3 and p2 values to the load con
load_con_test(1:1512,1:4)=p3values;
load_con_test(1:1512,5:8)=p2values;

%Now, swap p3 and p2 values for the next 1512 values
load_con_test(1+1512:1512+1512,1:4)=p2values;
load_con_test(1+1512:1512+1512,5:8)=p3values;

save load_con_test_fault41_42_mediumversion.mat load_con_test
```

B.3 Generate the inputs and outputs for fault 41-42, 44-43 and etc

First, we need to use an existing transient fault simulation. Add the extra components to the transient fault simulations

```
%added the load_con matrix
load load_con_test_fault41_42_medianversion.mat


%Now add the following components to change the load_con or the load
characteristic
count_iteration=0;

%change the load bus
%use the one below for fault at 41-42
loadbus_index=[37 41 42 52 0];

%use the one below for fault at 44-43
%loadbus_index=[33 34 36 37 0];
%stop changing the load bus

for iteration=1:2592

load_info=[];
load_con=[];
load_info=load_con_test(iteration, :);

load_con(:,1)=[33 34 36 37]';
load_con(1, 2:3)=load_info(1, 1);
load_con(1, 4:5)=load_info(1, 5);
load_con(2, 2:3)=load_info(1, 2);
load_con(2, 4:5)=load_info(1, 6);
load_con(3, 2:3)=load_info(1, 3);
load_con(3, 4:5)=load_info(1, 7);
load_con(4, 2:3)=load_info(1, 4);
load_con(4, 4:5)=load_info(1, 8);
```

<<<<<<Now the part below is added at the bottom of the program>>>

```
%To calculate the complex power flow from the transient simulation results

V1= bus_v(From_idx,:); V2= bus_v(To_idx,:);
[S1,S2] = line_pq(V1,V2,R,X,B,tap,phi);

jjj=0;
loadP=zeros(size(bus,1),size(S1,2));
loadQ=zeros(size(bus,1),size(S1,2));
```

```
while jjj<size(line,1)
    jjj=jjj+1;

    iii=0;
    while iii<size(bus,1)
        iii=iii+1;
        if busnum(iii)==FromBus(jjj);
            loadP(iii,:)=loadP(iii,:) - real(S1(jjj,:));
            loadQ(iii,:)=loadQ(iii,:) - imag(S1(jjj,:));
         end
        if busnum(iii)==ToBus(jjj);
            loadP(iii,:)=loadP(iii,:) - real(S2(jjj,:));
            loadQ(iii,:)=loadQ(iii,:) - imag(S2(jjj,:));
        end
    end
end



keepsaving=0;

faultbus= sw_con(2,2);

casedata=[1:14]';

%added components to display this
count_i=0;
% loadbus_index=[37 41 42 52 0];
% loadbus_index=[41 0];
% loadbus_index=[33 0];
% loadbus_index=[44 0];
%stop displaying

while keepsaving<1
count_i=count_i+1;
%   loadbus=input('Enter the bus for which you want to save data >>');

% altered results for loadbus
loadbus=loadbus_index(count_i);
%stop altering


  if loadbus==0;
    keepsaving=1;
  end

  if isempty(loadbus);
    keepsaving=1;
  end
  iii=0;
  while iii<size(bus,1)
    iii=iii+1;
    if loadbus==busnum(iii);

        kkk = 0;
        while kkk<size(load_con,1)
            kkk = kkk+1;

            if load_con(kkk,1)==loadbus;

               ccc=0;
               ttt=100;

               while ccc<10

                 ccc=ccc+1;

                 ttt=ttt+10;

                 newdata=[abs(bus_v(busnum(iii),ttt))
abs(bus_v(busnum(iii),ttt+10)-bus_v(busnum(iii),ttt)) ...
                          (loadP(iii,ttt+10)-loadP(iii,ttt))/loadP(iii,1)
(loadQ(iii,ttt+10)-loadQ(iii,ttt))/loadQ(iii,1) ...
                           load_con(kkk,2) load_con(kkk,3) load_con(kkk,4)
load_con(kkk,5) ...
                            t(ttt) t(ttt+10)-t(ttt) loadP(iii,1) loadQ(iii,1)
loadbus faultbus]';

                 casedata=[casedata newdata];
               end
           end
        end

    end
  end

end

% disp(' ')
% disp('Data is in matrix called casedata.  ')
% disp('    Each column is one data set.  ')
% disp('    There are 10 data sets per bus entered and found in load_con')
% disp(' ')
% disp('In casedata matrix, ')
% disp('  Row 1 is the pu bus voltage magnitude')
% disp('  Row 2 is change in bus pu voltage magnitude')
% disp('  Row 3 is change in load real power over initial real power')
% disp('  Row 4 is change in load reactive power over initial reactive
power')
% disp('  Row 5 is fraction const active power load from load_con')
% disp('  Row 6 is fraction const reactive power load from load_con')
% disp('  Row 7 is fraction const active current load from load_con')
% disp('  Row 8 is fraction const reactive current load from load_con')
% disp('  Row 9 is simultation time at beginning of interval')
% disp('  Row 10 is change in time')
% disp('  Row 11 is initial real power')
% disp('  Row 12 is initial reactive power')
% disp('  Row 13 is the load bus number')
% disp('  Row 14 is bus number of the faulted bus in the simulation')





%the input and output training data obtained from here
%this is for load bus 37
for bbb=1:6

count_iteration=count_iteration+1;
datamatrixl37(count_iteration, 1)=casedata(1, bbb+4);
datamatrixl37(count_iteration, 2)=casedata(2, bbb+4);
datamatrixl37(count_iteration, 3)=casedata(3, bbb+4);
datamatrixl37(count_iteration, 4)=casedata(4, bbb+4);
datamatrixl37(count_iteration, 5)=1-casedata(7, bbb+4)-casedata(5, bbb+4);
datamatrixl37(count_iteration, 6)=casedata(7, bbb+4);
datamatrixl37(count_iteration, 7)=casedata(5, bbb+4);

%this is for load bus 41
datamatrixl41(count_iteration, 1)=casedata(1, bbb+14);
datamatrixl41(count_iteration, 2)=casedata(2, bbb+14);
datamatrixl41(count_iteration, 3)=casedata(3, bbb+14);
datamatrixl41(count_iteration, 4)=casedata(4, bbb+14);
datamatrixl41(count_iteration, 5)=1-casedata(7, bbb+14)-casedata(5, bbb+14);
datamatrixl41(count_iteration, 6)=casedata(7, bbb+14);
datamatrixl41(count_iteration, 7)=casedata(5, bbb+14);

%this is for load bus 42
datamatrixl42(count_iteration, 1)=casedata(1, bbb+24);
datamatrixl42(count_iteration, 2)=casedata(2, bbb+24);
datamatrixl42(count_iteration, 3)=casedata(3, bbb+24);
datamatrixl42(count_iteration, 4)=casedata(4, bbb+24);
datamatrixl42(count_iteration, 5)=1-casedata(7, bbb+24)-casedata(5, bbb+24);
datamatrixl42(count_iteration, 6)=casedata(7, bbb+24);
datamatrixl42(count_iteration, 7)=casedata(5, bbb+24);

%this is for load bus 52
datamatrixl52(count_iteration, 1)=casedata(1, bbb+34);
datamatrixl52(count_iteration, 2)=casedata(2, bbb+34);
datamatrixl52(count_iteration, 3)=casedata(3, bbb+34);
datamatrixl52(count_iteration, 4)=casedata(4, bbb+34);
datamatrixl52(count_iteration, 5)=1-casedata(7, bbb+34)-casedata(5, bbb+34);
datamatrixl52(count_iteration, 6)=casedata(7, bbb+34);
datamatrixl52(count_iteration, 7)=casedata(5, bbb+34);
```

```
end
count_iteration
end

%normalized matrix for load bus 37
datamatrix_normalizedl37(:, 1)=mat2gray(datamatrixl37(:,1));
datamatrix_normalizedl37(:, 2)=mat2gray(datamatrixl37(:,2));
datamatrix_normalizedl37(:, 3)=mat2gray(datamatrixl37(:,3));
datamatrix_normalizedl37(:, 4)=mat2gray(datamatrixl37(:,4));
datamatrix_normalizedl37(:, 5)=datamatrixl37(:,5);
datamatrix_normalizedl37(:, 6)=datamatrixl37(:,6);
datamatrix_normalizedl37(:, 7)=datamatrixl37(:,7);

%normalized matrix for load bus 41
datamatrix_normalizedl41(:, 1)=mat2gray(datamatrixl41(:,1));
datamatrix_normalizedl41(:, 2)=mat2gray(datamatrixl41(:,2));
datamatrix_normalizedl41(:, 3)=mat2gray(datamatrixl41(:,3));
datamatrix_normalizedl41(:, 4)=mat2gray(datamatrixl41(:,4));
datamatrix_normalizedl41(:, 5)=datamatrixl41(:,5);
datamatrix_normalizedl41(:, 6)=datamatrixl41(:,6);
datamatrix_normalizedl41(:, 7)=datamatrixl41(:,7);

%normalized matrix for load bus 42
datamatrix_normalizedl42(:, 1)=mat2gray(datamatrixl42(:,1));
datamatrix_normalizedl42(:, 2)=mat2gray(datamatrixl42(:,2));
datamatrix_normalizedl42(:, 3)=mat2gray(datamatrixl42(:,3));
datamatrix_normalizedl42(:, 4)=mat2gray(datamatrixl42(:,4));
datamatrix_normalizedl42(:, 5)=datamatrixl42(:,5);
datamatrix_normalizedl42(:, 6)=datamatrixl42(:,6);
datamatrix_normalizedl42(:, 7)=datamatrixl42(:,7);

%normalized matrix for load bus 52
datamatrix_normalizedl52(:, 1)=mat2gray(datamatrixl52(:,1));
datamatrix_normalizedl52(:, 2)=mat2gray(datamatrixl52(:,2));
datamatrix_normalizedl52(:, 3)=mat2gray(datamatrixl52(:,3));
datamatrix_normalizedl52(:, 4)=mat2gray(datamatrixl52(:,4));
datamatrix_normalizedl52(:, 5)=datamatrixl52(:,5);
datamatrix_normalizedl52(:, 6)=datamatrixl52(:,6);
datamatrix_normalizedl52(:, 7)=datamatrixl52(:,7);

save input_output_fault41_42_4loadbuses_mediumversion.mat datamatrixl37
datamatrix_normalizedl37 datamatrixl41 datamatrix_normalizedl41 datamatrixl42
datamatrix_normalizedl42 datamatrixl52 datamatrix_normalizedl52
```

B.4 Intelligent Methods

B.4.1 Levenberg-Marquardt, Widow-Hoff backpropagation, and Default Scaled Conjugate Gradient methods to train strategy E's

```
clear all;
clc;
close all;
load input_output_fault41_42_4loadbuses_mediumversion.mat;
load input_output_fault44_43_4loadbuses_mediumversion.mat;

count=0;
number_of_points=15552*4;
% number_of_points=15552;
% number_of_points=10000;

% dataPinput=(datamatrix_normalizedl41(1:number_of_points, 1:3))';
% dataPoutput=(datamatrix_normalizedl41(1:number_of_points, 5))';
% datamatrix_normalized1=datamatrix_normalizedl37;
% datamatrix_normalized2=datamatrix_normalizedl41;


% dataPinput=(datamatrix_normalized(1:number_of_points, 1:3))';
% dataPoutput=(datamatrix_normalized(1:number_of_points, 5))';



% dataPinput(1:3, 1:15552)=(datamatrix_normalized1(1:15552, 1:3))';
% dataPinput(1:3, 15553:31104)=(datamatrix_normalized2(1:15552, 1:3))';

% dataPoutput(1, 1:15552)=(datamatrix_normalized1(1:15552, 5))';
% dataPoutput(1, 15553:31104)=(datamatrix_normalized2(1:15552, 5))';

for Atrial=4
    datamatrix_normalized1=[];
    datamatrix_normalized2=[];
    datamatrix_normalized3=[];
    datamatrix_normalized4=[];
    dataPinput=[];
    dataPoutput=[];
    if Atrial==1
        datamatrix_normalized1=datamatrix_normalizedl41;
        datamatrix_normalized2=datamatrix_normalizedl42;
        datamatrix_normalized3=datamatrix_normalizedl44;
        datamatrix_normalized4=datamatrix_normalizedl43;
    elseif Atrial==2
        datamatrix_normalized1=datamatrix_normalizedl37;
        datamatrix_normalized2=datamatrix_normalizedl52;
        datamatrix_normalized3=datamatrix_normalizedl36;
        datamatrix_normalized4=datamatrix_normalizedl51;
    elseif Atrial==3
        datamatrix_normalized1=datamatrix_normalizedl37;
        datamatrix_normalized2=datamatrix_normalizedl41;
        datamatrix_normalized3=datamatrix_normalizedl36;
        datamatrix_normalized4=datamatrix_normalizedl43;
    elseif Atrial==4
        datamatrix_normalized1=datamatrix_normalizedl42;
        datamatrix_normalized2=datamatrix_normalizedl52;
        datamatrix_normalized3=datamatrix_normalizedl44;
        datamatrix_normalized4=datamatrix_normalizedl51;
    end

dataPinput(1:3, 1:15552)=(datamatrix_normalized1(1:15552, 1:3))';
dataPinput(1:3, 15553:31104)=(datamatrix_normalized2(1:15552, 1:3))';
dataPinput(1:3, 31105:46656)=(datamatrix_normalized3(1:15552, 1:3))';
dataPinput(1:3, 46657:62208)=(datamatrix_normalized4(1:15552, 1:3))';
dataPoutput(1, 1:15552)=(datamatrix_normalized1(1:15552, 5))';
dataPoutput(1, 15553:31104)=(datamatrix_normalized2(1:15552, 5))';
dataPoutput(1, 31105:46656)=(datamatrix_normalized3(1:15552, 5))';
dataPoutput(1, 46657:62208)=(datamatrix_normalized4(1:15552, 5))';

average_error=100000;


for trial=1:10

count=count+1;
count
%NNtool Levenberg-Marquardt
% net=newfit(dataPinput, dataPoutput, 20);
% net=train(net, dataPinput, dataPoutput);
% traiendoutput=sim(net, dataPinput);

%NNtool Widow-Hoff backpropagation
% net=newff(dataPinput, dataPoutput, 20);
% net=init(net);
% net=train(net, dataPinput, dataPoutput);

%Default Scaled Conjugate Gradient
net=newpr(dataPinput, dataPoutput, 20);
net=train(net, dataPinput, dataPoutput);

traiendoutput=sim(net, dataPinput);

%error calculation
for i=1:number_of_points
    error(i)=0.5*((traiendoutput(i)-dataPoutput(i))^2);
    difference(i)=traiendoutput(i)-dataPoutput(i);
end
error_sum_original=sum(error)/number_of_points;

% x=(1:1:number_of_points)';
% plot(x, dataPoutput, x, traiendoutput);
% legend('The real Data', 'Trained Data');
% title('The real data v.s. the trained data');
% figure;
% plot(dataPoutput, traiendoutput);
% title('The real data v.s. the trained data');
%
% figure;
% subplot(2,1,1);
% plot(error);
% title('The mean square error of the real and model data');
% subplot(2,1,2);
% plot(difference);
% title('The difference of the real and model data');

% trained_net41_42=net;
trained_net37=net;
% save NNtoolFitting_bus37.mat trained_net37
% save NNtoolFitting_a_f_trainedfault41_42.mat trained_net41_42
% save NNtoolFitting_a_f_trainedfault41_42f10000.mat trained_net41_42f10000

%Test on the other buses
sumbus_error=0;
number_of_points2=15552;
for kkk=1:8
    datamatrix_normalized=[];
    error=0;
    error_sum=0;
    if kkk==1
        datamatrix_normalized=datamatrix_normalizedl37;
    elseif kkk==2
        datamatrix_normalized=datamatrix_normalizedl41;
    elseif kkk==3
        datamatrix_normalized=datamatrix_normalizedl42;
    elseif kkk==4
        datamatrix_normalized=datamatrix_normalizedl52;
    elseif kkk==5
        datamatrix_normalized=datamatrix_normalizedl36;
    elseif kkk==6
        datamatrix_normalized=datamatrix_normalizedl44;
    elseif kkk==7
        datamatrix_normalized=datamatrix_normalizedl43;
    elseif kkk==8
        datamatrix_normalized=datamatrix_normalizedl51;
    end

%initializing
dataPinput2=[];
dataPoutput2=[];
trainedoutput2=[];

dataPinput2=(datamatrix_normalized(1:number_of_points2, 1:3))';
dataPoutput2=(datamatrix_normalized(1:number_of_points2, 5))';
```

```
traiendoutput2=sim(trained_net37, dataPinput2);

%error calculation
for i=1:number_of_points2
    error(i)=0.5*((traiendoutput2(i)-dataPoutput2(i))^2);
    difference(i)=traiendoutput2(i)-dataPoutput2(i);
end
error_sum=sum(error)/number_of_points2;

    if kkk==1
        error37_trial=error_sum;
    elseif kkk==2
        error41_trial=error_sum;
    elseif kkk==3
        error42_trial=error_sum;
    elseif kkk==4
        error52_trial=error_sum;
    elseif kkk==5
        error36_trial=error_sum;
    elseif kkk==6
        error44_trial=error_sum;
    elseif kkk==7
        error43_trial=error_sum;
    elseif kkk==8
        error51_trial=error_sum;
    end
    sumbus_error=sumbus_error+error_sum;
end
average_error_trial=sumbus_error/8;

if average_error_trial<average_error
    average_error=average_error_trial;
    error37=error37_trial;
    error41=error41_trial;
    error42=error42_trial;
    error52=error52_trial;
    error36=error36_trial;
    error44=error44_trial;
    error43=error43_trial;
    error51=error51_trial;
end

end
    average_error
    d = {'error37' error37; 'error41' error41; 'error42' error42; 'error52'
error52; 'error36' error36; 'error44' error44; 'error43' error43; 'error51'
error51; 'average_error' average_error}

%     xlswrite('testedbuses', d, 'Testing buses', 'A1');

    if Atrial==1
        xlswrite('testedbusespE', d, 'Testing buses E1', 'A1');
    elseif Atrial==2
        xlswrite('testedbusespE', d, 'Testing buses E2', 'A1');
    elseif Atrial==3
        xlswrite('testedbusespE', d, 'Testing buses E3', 'A1');
    elseif Atrial==4
        xlswrite('testedbusespE', d, 'Testing buses E4', 'A1');
    end
end
```

B.4.2 Widow-Hoff Backpropagation Method on Strategy B's

```
clear all;
clc;
close all;
load input_output_fault41_42_4loadbuses_mediumversion.mat;
load input_output_fault44_43_4loadbuses_mediumversion.mat;

count=0;
number_of_points=15552*2;


for Atrial=1:10
    datamatrix_normalized1=[];
    datamatrix_normalized2=[];
    dataPinput=[];
    dataPoutput=[];
    if Atrial==1
        datamatrix_normalized1=datamatrix_normalizedl41;
        datamatrix_normalized2=datamatrix_normalizedl42;
    elseif Atrial==2
        datamatrix_normalized1=datamatrix_normalizedl44;
        datamatrix_normalized2=datamatrix_normalizedl43;
    elseif Atrial==3
        datamatrix_normalized1=datamatrix_normalizedl37;
        datamatrix_normalized2=datamatrix_normalizedl41;
    elseif Atrial==4
        datamatrix_normalized1=datamatrix_normalizedl36;
        datamatrix_normalized2=datamatrix_normalizedl44;
    elseif Atrial==5
        datamatrix_normalized1=datamatrix_normalizedl37;
        datamatrix_normalized2=datamatrix_normalizedl42;
    elseif Atrial==6
        datamatrix_normalized1=datamatrix_normalizedl36;
        datamatrix_normalized2=datamatrix_normalizedl43;
    elseif Atrial==7
        datamatrix_normalized1=datamatrix_normalizedl41;
        datamatrix_normalized2=datamatrix_normalizedl52;
    elseif Atrial==8
        datamatrix_normalized1=datamatrix_normalizedl44;
        datamatrix_normalized2=datamatrix_normalizedl51;
    elseif Atrial==9
        datamatrix_normalized1=datamatrix_normalizedl37;
        datamatrix_normalized2=datamatrix_normalizedl52;
    elseif Atrial==10
```

```
        datamatrix_normalized1=datamatrix_normalizedl36;
        datamatrix_normalized2=datamatrix_normalizedl51;
    end

dataPinput(1:3, 1:15552)=(datamatrix_normalized1(1:15552, 1:3))';
dataPinput(1:3, 15553:31104)=(datamatrix_normalized2(1:15552, 1:3))';
dataPoutput(1, 1:15552)=(datamatrix_normalized1(1:15552, 5))';
dataPoutput(1, 15553:31104)=(datamatrix_normalized2(1:15552, 5))';
%used for 10 trials
numberInputTrain=3;
NumberOfInputs=3;
MAX_Epochs=500;
N=number_of_points;
%Number of neurons in the hidden layer

HiddenNeurons=20;
epsilon=0.3;
momentum_factor=0.05;

%used for 10 trials
average_error=100000;

%10 trials
for trial=1:10

count=count+1;
count



w1r1=[];
w2r1=[];
    w1r1=(rand(HiddenNeurons,NumberOfInputs+1)-0.5)*3;
    w2r1=(rand(1,(HiddenNeurons+1))-0.5)*3;
    deltaw1r1=0;
    deltaw2r1=0;

for epoch = 1: MAX_Epochs
    for i = 1: N
% FOr output 1--------------------------------------------------------
%watch where to put the p or q for p, it's 1:3, for q, it is 1:2, 4
        xr1=dataPinput(:,i);
        tr1=dataPoutput(1,i);
        [dw1r1, dw2r1, outr1]=backpropagation_single_output(xr1, w1r1, w2r1,
tr1, HiddenNeurons, epsilon);
        deltaw1r1=momentum_factor*deltaw1r1+dw1r1;
        deltaw2r1=momentum_factor*deltaw2r1+dw2r1;
        w1r1=w1r1+deltaw1r1;
        w2r1=w2r1+deltaw2r1;
        errr1(i)=0.5*(tr1-outr1)^2;
        output_1(i)=outr1;
   end
    errorr1(epoch)=sum(errr1(1:N))/N;
end
```

```
MSE_error=errorr1(epoch)






%error calculation
% for i=1:number_of_points
%     error(i)=0.5*((dataPoutput(i)-output_1(i))^2);
%     difference(i)=dataPoutput(i)-output_1(i);
% end
% error_sum_original=sum(error)/number_of_points;

% x=(1:1:number_of_points)';
% plot(x, dataPoutput, x, traiendoutput);
% legend('The real Data', 'Trained Data');
% title('The real data v.s. the trained data');
% figure;
% plot(dataPoutput, traiendoutput);
% title('The real data v.s. the trained data');
%
% figure;
% subplot(2,1,1);
% plot(error);
% title('The mean square error of the real and model data');
% subplot(2,1,2);
% plot(difference);
% title('The difference of the real and model data');

% trained_net41_42=net;






% save NNtoolFitting_bus37.mat trained_net37
% save NNtoolFitting_a_f_trainedfault41_42.mat trained_net41_42
% save NNtoolFitting_a_f_trainedfault41_42f10000.mat trained_net41_42f10000

%Test on the other buses
sumbus_error=0;
number_of_points_2=15552;
for kkk=1:8
    datamatrix_normalized=[];
    error=0;
    error_sum=0;
    if kkk==1
        datamatrix_normalized=datamatrix_normalizedl37;
    elseif kkk==2
        datamatrix_normalized=datamatrix_normalizedl41;
    elseif kkk==3
```

```
        datamatrix_normalized=datamatrix_normalizedl42;
    elseif kkk==4
        datamatrix_normalized=datamatrix_normalizedl52;
    elseif kkk==5
        datamatrix_normalized=datamatrix_normalizedl36;
    elseif kkk==6
        datamatrix_normalized=datamatrix_normalizedl44;
    elseif kkk==7
        datamatrix_normalized=datamatrix_normalizedl43;
    elseif kkk==8
        datamatrix_normalized=datamatrix_normalizedl51;
    end

%initializing
dataPinput2=[];
dataPoutput2=[];
trainedoutput2=[];

dataPinput2=(datamatrix_normalized(1:number_of_points_2, 1:3))';
dataPoutput2=(datamatrix_normalized(1:number_of_points_2, 5))';

%ntool method
% traiendoutput2=sim(trained_net37, dataPinput2);

w1=[];
w2=[];
w1=w1r1;
w2=w2r1;
N2=number_of_points_2;
for i = 1: N2

        x=dataPinput2(:,i);
        t=dataPoutput2(1,i);
        [dw1, dw2, out]=backpropagation_single_output(x, w1, w2, t,
HiddenNeurons, epsilon);
        output_1(i)=out;
        err(i)=0.5*(t-out)^2;
end


error(1)=sum(err)/N2;

% traiendoutput=sim(net, dataPinput);

traiendoutput2=output_1;

%error calculation
for i=1:number_of_points_2
    error(i)=0.5*((traiendoutput2(i)-dataPoutput2(i))^2);
    difference(i)=traiendoutput2(i)-dataPoutput2(i);
end
error_sum=sum(error)/number_of_points_2;
```

```
    if kkk==1
        error37_trial=error_sum;
    elseif kkk==2
        error41_trial=error_sum;
    elseif kkk==3
        error42_trial=error_sum;
    elseif kkk==4
        error52_trial=error_sum;
    elseif kkk==5
        error36_trial=error_sum;
    elseif kkk==6
        error44_trial=error_sum;
    elseif kkk==7
        error43_trial=error_sum;
    elseif kkk==8
        error51_trial=error_sum;
    end
    sumbus_error=sumbus_error+error_sum;
end
average_error_trial=sumbus_error/8;

%find the smallest error among the 10 trials
if average_error_trial<average_error
    average_error=average_error_trial;
    error37=error37_trial;
    error41=error41_trial;
    error42=error42_trial;
    error52=error52_trial;
    error36=error36_trial;
    error44=error44_trial;
    error43=error43_trial;
    error51=error51_trial;
end

%not finding the smallest error coz only 1 trial for anfis
%
%     average_error=average_error_trial;
%     error37=error37_trial;
%     error41=error41_trial;
%     error42=error42_trial;
%     error52=error52_trial;
%     error36=error36_trial;
%     error44=error44_trial;
%     error43=error43_trial;
%     error51=error51_trial;

% end for the 10 trial
end
    average_error
    d = {'error37' error37; 'error41' error41; 'error42' error42; 'error52'
error52; 'error36' error36; 'error44' error44; 'error43' error43; 'error51'
error51; 'average_error' average_error}
```

```
%     xlswrite('testedbuses', d, 'Testing buses', 'A1');

    if Atrial==1
        xlswrite('testedbuses_NB', d, 'Testing buses B1', 'A1');
    elseif Atrial==2
        xlswrite('testedbuses_NB', d, 'Testing buses B2', 'A1');
    elseif Atrial==3
        xlswrite('testedbuses_NB', d, 'Testing buses B3', 'A1');
    elseif Atrial==4
        xlswrite('testedbuses_NB', d, 'Testing buses B4', 'A1');
    elseif Atrial==5
        xlswrite('testedbuses_NB', d, 'Testing buses B5', 'A1');
    elseif Atrial==6
        xlswrite('testedbuses_NB', d, 'Testing buses B6', 'A1');
    elseif Atrial==7
        xlswrite('testedbuses_NB', d, 'Testing buses B7', 'A1');
    elseif Atrial==8
        xlswrite('testedbuses_NB', d, 'Testing buses B8', 'A1');
    elseif Atrial==9
        xlswrite('testedbuses_NB', d, 'Testing buses B9', 'A1');
    elseif Atrial==10
        xlswrite('testedbuses_NB', d, 'Testing buses B10', 'A1');
    end
end
```

B.4.3 ANFIS Method on Strategy C's

```
clear all;
clc;
close all;
load input_output_fault41_42_4loadbuses_mediumversion.mat;
load input_output_fault44_43_4loadbuses_mediumversion.mat;

count=0;
number_of_points=15552*3;

for Atrial=1:4
    datamatrix_normalized1=[];
    datamatrix_normalized2=[];
    datamatrix_normalized3=[];
    dataPinput=[];
    dataPoutput=[];
    trnData=[];
    final_out_fis=[];
    out_fis=[];

    if Atrial==1
        datamatrix_normalized1=datamatrix_normalizedl37;
        datamatrix_normalized2=datamatrix_normalizedl41;
        datamatrix_normalized3=datamatrix_normalizedl42;
    elseif Atrial==2
        datamatrix_normalized1=datamatrix_normalizedl36;
        datamatrix_normalized2=datamatrix_normalizedl44;
        datamatrix_normalized3=datamatrix_normalizedl43;
    elseif Atrial==3
        datamatrix_normalized1=datamatrix_normalizedl41;
        datamatrix_normalized2=datamatrix_normalizedl42;
        datamatrix_normalized3=datamatrix_normalizedl52;
    elseif Atrial==4
        datamatrix_normalized1=datamatrix_normalizedl44;
        datamatrix_normalized2=datamatrix_normalizedl43;
        datamatrix_normalized3=datamatrix_normalizedl51;
    end

dataPinput(1:3, 1:15552)=(datamatrix_normalized1(1:15552, 1:3))';
dataPinput(1:3, 15553:31104)=(datamatrix_normalized2(1:15552, 1:3))';
dataPinput(1:3, 31105:46656)=(datamatrix_normalized3(1:15552, 1:3))';
dataPoutput(1, 1:15552)=(datamatrix_normalized1(1:15552, 5))';
dataPoutput(1, 15553:31104)=(datamatrix_normalized2(1:15552, 5))';
dataPoutput(1, 31105:46656)=(datamatrix_normalized3(1:15552, 5))';
%used for 10 trials
% average_error=100000;

%10 trials
% for trial=1:10

count=count+1;
count
%NNtool fit a function
% net=newfit(dataPinput, dataPoutput, 20);
% net=train(net, dataPinput, dataPoutput);
% traiendoutput=sim(net, dataPinput);

%NNtool feedforward feedback backpropagation
% net=newff(dataPinput, dataPoutput, 20);
% net=init(net);
% net=train(net, dataPinput, dataPoutput);
%
% traiendoutput=sim(net, dataPinput);


%important trained_net37
% trained_net37=net;

%ANFIS method
trnData(:, 1:3)=dataPinput(1:3, :)';
trnData(:, 4)=dataPoutput(1, :)';
% x=(1:1:number_of_points)';
%it was 15
numMFs=5;
mfType='gbellmf';
epoch_n=20;
```

```
in_fis=genfis1(trnData, numMFs, mfType);
out_fis=anfis(trnData, in_fis, epoch_n);

traiendoutput=evalfis(trnData(:,1:3), out_fis);


final_out_fis=out_fis;


%error calculation
for i=1:number_of_points
    error(i)=0.5*((traiendoutput(i)-dataPoutput(i))^2);
    difference(i)=traiendoutput(i)-dataPoutput(i);
end
error_sum_original=sum(error)/number_of_points;

% x=(1:1:number_of_points)';
% plot(x, dataPoutput, x, traiendoutput);
% legend('The real Data', 'Trained Data');
% title('The real data v.s. the trained data');
% figure;
% plot(dataPoutput, traiendoutput);
% title('The real data v.s. the trained data');
%
% figure;
% subplot(2,1,1);
% plot(error);
% title('The mean square error of the real and model data');
% subplot(2,1,2);
% plot(difference);
% title('The difference of the real and model data');

% trained_net41_42=net;






% save NNtoolFitting_bus37.mat trained_net37
% save NNtoolFitting_a_f_trainedfault41_42.mat trained_net41_42
% save NNtoolFitting_a_f_trainedfault41_42f10000.mat trained_net41_42f10000

%Test on the other buses
sumbus_error=0;
number_of_points=15552;
for kkk=1:8
    datamatrix_normalized=[];
    error=0;
    error_sum=0;
    if kkk==1
        datamatrix_normalized=datamatrix_normalizedl37;
    elseif kkk==2
```

```
        datamatrix_normalized=datamatrix_normalizedl41;
    elseif kkk==3
        datamatrix_normalized=datamatrix_normalizedl42;
    elseif kkk==4
        datamatrix_normalized=datamatrix_normalizedl52;
    elseif kkk==5
        datamatrix_normalized=datamatrix_normalizedl36;
    elseif kkk==6
        datamatrix_normalized=datamatrix_normalizedl44;
    elseif kkk==7
        datamatrix_normalized=datamatrix_normalizedl43;
    elseif kkk==8
        datamatrix_normalized=datamatrix_normalizedl51;
    end

%initializing
dataPinput2=[];
dataPoutput2=[];
trainedoutput2=[];

dataPinput2=(datamatrix_normalized(1:number_of_points, 1:3))';
dataPoutput2=(datamatrix_normalized(1:number_of_points, 5))';

%ntool method
% traiendoutput2=sim(trained_net37, dataPinput2);

traiendoutput2=evalfis(dataPinput2', final_out_fis);

%error calculation
for i=1:number_of_points
    error(i)=0.5*((traiendoutput2(i)-dataPoutput2(i))^2);
    difference(i)=traiendoutput2(i)-dataPoutput2(i);
end
error_sum=sum(error)/number_of_points;

    if kkk==1
        error37_trial=error_sum;
    elseif kkk==2
        error41_trial=error_sum;
    elseif kkk==3
        error42_trial=error_sum;
    elseif kkk==4
        error52_trial=error_sum;
    elseif kkk==5
        error36_trial=error_sum;
    elseif kkk==6
        error44_trial=error_sum;
    elseif kkk==7
        error43_trial=error_sum;
    elseif kkk==8
        error51_trial=error_sum;
    end
    sumbus_error=sumbus_error+error_sum;
```

```
end
average_error_trial=sumbus_error/8;

%find the smallest error among the 10 trials
% if average_error_trial<average_error
%     average_error=average_error_trial;
%     error37=error37_trial;
%     error41=error41_trial;
%     error42=error42_trial;
%     error52=error52_trial;
%     error36=error36_trial;
%     error44=error44_trial;
%     error43=error43_trial;
%     error51=error51_trial;
% end

%not finding the smallest error coz only 1 trial for anfis

    average_error=average_error_trial;
    error37=error37_trial;
    error41=error41_trial;
    error42=error42_trial;
    error52=error52_trial;
    error36=error36_trial;
    error44=error44_trial;
    error43=error43_trial;
    error51=error51_trial;

% end for the 10 trial
% end
    average_error
    d = {'error37' error37; 'error41' error41; 'error42' error42; 'error52'
error52; 'error36' error36; 'error44' error44; 'error43' error43; 'error51'
error51; 'average_error' average_error}

%     xlswrite('testedbuses', d, 'Testing buses', 'A1');

    if Atrial==1
        xlswrite('testedbusesC', d, 'Testing buses C1', 'A1');
    elseif Atrial==2
        xlswrite('testedbusesC', d, 'Testing buses C2', 'A1');
    elseif Atrial==3
        xlswrite('testedbusesC', d, 'Testing buses C3', 'A1');
    elseif Atrial==4
        xlswrite('testedbusesC', d, 'Testing buses C4', 'A1');
    end
end
```

B.4.4 Levenberg-Marquardt, Widow-Hoff backpropagation, and Default Scaled Conjugate Gradient methods training with equivalent training criteria

```
clear all;
clc;
close all;
load input_output_fault41_42_4loadbuses_mediumversion.mat;
load input_output_fault44_43_4loadbuses_mediumversion.mat;

count=0;
average_error=[];
number_of_points=15552*4;

for Atrial=2
    datamatrix_normalized1=[];
    datamatrix_normalized2=[];
    datamatrix_normalized3=[];
    datamatrix_normalized4=[];
    dataPinput=[];
    dataPoutput=[];
    if Atrial==1
        datamatrix_normalized1=datamatrix_normalizedl41;
        datamatrix_normalized2=datamatrix_normalizedl42;
        datamatrix_normalized3=datamatrix_normalizedl44;
        datamatrix_normalized4=datamatrix_normalizedl43;
    elseif Atrial==2
        datamatrix_normalized1=datamatrix_normalizedl37;
        datamatrix_normalized2=datamatrix_normalizedl52;
        datamatrix_normalized3=datamatrix_normalizedl36;
        datamatrix_normalized4=datamatrix_normalizedl51;
    elseif Atrial==3
        datamatrix_normalized1=datamatrix_normalizedl37;
        datamatrix_normalized2=datamatrix_normalizedl41;
        datamatrix_normalized3=datamatrix_normalizedl36;
        datamatrix_normalized4=datamatrix_normalizedl43;
    elseif Atrial==4
        datamatrix_normalized1=datamatrix_normalizedl42;
        datamatrix_normalized2=datamatrix_normalizedl52;
        datamatrix_normalized3=datamatrix_normalizedl44;
        datamatrix_normalized4=datamatrix_normalizedl51;
    end

%get all the P parameters

dataPinput(1:3, 1:15552)=(datamatrix_normalized1(1:15552, 1:3))';
dataPinput(1:3, 15553:31104)=(datamatrix_normalized2(1:15552, 1:3))';
dataPinput(1:3, 31105:46656)=(datamatrix_normalized3(1:15552, 1:3))';
dataPinput(1:3, 46657:62208)=(datamatrix_normalized4(1:15552, 1:3))';
dataPoutput(1, 1:15552)=(datamatrix_normalized1(1:15552, 5))';
dataPoutput(1, 15553:31104)=(datamatrix_normalized2(1:15552, 5))';
dataPoutput(1, 31105:46656)=(datamatrix_normalized3(1:15552, 5))';
dataPoutput(1, 46657:62208)=(datamatrix_normalized4(1:15552, 5))';

dataPoutput_P3(1, 1:15552)=(datamatrix_normalized1(1:15552, 7))';
dataPoutput_P3(1, 15553:31104)=(datamatrix_normalized2(1:15552, 7))';
dataPoutput_P3(1, 31105:46656)=(datamatrix_normalized3(1:15552, 7))';
```

```
dataPoutput_P3(1, 46657:62208)=(datamatrix_normalized4(1:15552, 7))';

dataPoutput_P2(1, 1:15552)=(datamatrix_normalized1(1:15552, 6))';
dataPoutput_P2(1, 15553:31104)=(datamatrix_normalized2(1:15552, 6))';
dataPoutput_P2(1, 31105:46656)=(datamatrix_normalized3(1:15552, 6))';
dataPoutput_P2(1, 46657:62208)=(datamatrix_normalized4(1:15552, 6))';

%get all the q parameters

% dataQinput(1:2, 1:15552)=(datamatrix_normalized1(1:15552, 1:2))';
% dataQinput(1:2, 15553:31104)=(datamatrix_normalized2(1:15552, 1:2))';
% dataQinput(1:2, 31105:46656)=(datamatrix_normalized3(1:15552, 1:2))';
% dataQinput(1:2, 46657:62208)=(datamatrix_normalized4(1:15552, 1:2))';
%
% dataQinput(3, 1:15552)=(datamatrix_normalized1(1:15552, 4))';
% dataQinput(3, 15553:31104)=(datamatrix_normalized2(1:15552, 4))';
% dataQinput(3, 31105:46656)=(datamatrix_normalized3(1:15552, 4))';
% dataQinput(3, 46657:62208)=(datamatrix_normalized4(1:15552, 4))';
%
% dataQoutput(1, 1:15552)=(datamatrix_normalized1(1:15552, 5))';
% dataQoutput(1, 15553:31104)=(datamatrix_normalized2(1:15552, 5))';
% dataQoutput(1, 31105:46656)=(datamatrix_normalized3(1:15552, 5))';
% dataQoutput(1, 46657:62208)=(datamatrix_normalized4(1:15552, 5))';
%
% dataQoutput_q3(1, 1:15552)=(datamatrix_normalized1(1:15552, 7))';
% dataQoutput_q3(1, 15553:31104)=(datamatrix_normalized2(1:15552, 7))';
% dataQoutput_q3(1, 31105:46656)=(datamatrix_normalized3(1:15552, 7))';
% dataQoutput_q3(1, 46657:62208)=(datamatrix_normalized4(1:15552, 7))';
%
% dataQoutput_q2(1, 1:15552)=(datamatrix_normalized1(1:15552, 6))';
% dataQoutput_q2(1, 15553:31104)=(datamatrix_normalized2(1:15552, 6))';
% dataQoutput_q2(1, 31105:46656)=(datamatrix_normalized3(1:15552, 6))';
% dataQoutput_q2(1, 46657:62208)=(datamatrix_normalized4(1:15552, 6))';



%start training

average_error=100000;
for trial=1:10

count=count+1;
count



%Levenberg-Marquardt
% %p1
%
%
% net_p1=newfit(dataPinput, dataPoutput, 20);
%
% net_p1.trainParam.epochs=500;%(number of epochs)
% net_p1.trainParam.lr=0.3;%(learning rate)
%
% net_p1=train(net_p1, dataPinput, dataPoutput);
%
% traiendoutput_p1=sim(net_p1, dataPinput);
%
% %q1
% % net_q1=newfit(dataQinput, dataQoutput, 20);
% % net_q1=train(net_q1, dataQinput, dataQoutput);
%
% % traiendoutput_q1=sim(net_q1, dataQinput);
%
% %p3
% net_p3=newfit(dataPinput, dataPoutput_P3, 20);
%
% net_p3.trainParam.epochs=500;%(number of epochs)
% net_p3.trainParam.lr=0.3;%(learning rate)
%
% net_p3=train(net_p3, dataPinput, dataPoutput_P3);
%
% traiendoutput_p3=sim(net_p3, dataPinput);
%
% %q3
% % net_q3=newfit(dataQinput, dataQoutput_q3, 20);
% % net_q3=train(net_q3, dataQinput, dataQoutput_q3);
% %
% % traiendoutput_q3=sim(net_q3, dataQinput);
%
% %p2
% % dataPinput_p2(1:3, 1:15552)=(datamatrix_normalized1(1:15552, 1:3))';
% dataPinput_p2(1:3, 1:15552)=(datamatrix_normalized1(1:15552, 1:3))';
% dataPinput_p2(1:3, 15553:31104)=(datamatrix_normalized2(1:15552, 1:3))';
% dataPinput_p2(1:3, 31105:46656)=(datamatrix_normalized3(1:15552, 1:3))';
% dataPinput_p2(1:3, 46657:62208)=(datamatrix_normalized4(1:15552, 1:3))';
%
% dataPinput_p2(4, 1:62208)=traiendoutput_p1;
% dataPinput_p2(5, 1:62208)=traiendoutput_p3;
% dataPinput_p2(6, 1:62208)=1-traiendoutput_p1-traiendoutput_p3;
%
% net_p2=newfit(dataPinput_p2, dataPoutput_P2, 20);
%
% net_p2.trainParam.epochs=500;%(number of epochs)
% net_p2.trainParam.lr=0.3;%(learning rate)
%
% net_p2=train(net_p2, dataPinput_p2, dataPoutput_P2);
%
% traiendoutput_p2=sim(net_p2, dataPinput_p2);
%
% %q2
% % dataQinput_q2(1:2, 1:15552)=(datamatrix_normalized1(1:15552, 1:2))';
% % dataQinput_q2(1:2, 15553:31104)=(datamatrix_normalized2(1:15552, 1:2))';
% % dataQinput_q2(1:2, 31105:46656)=(datamatrix_normalized3(1:15552, 1:2))';
% % dataQinput_q2(1:2, 46657:62208)=(datamatrix_normalized4(1:15552, 1:2))';
% %
% % dataQinput_q2(3, 1:15552)=(datamatrix_normalized1(1:15552, 4))';
% % dataQinput_q2(3, 15553:31104)=(datamatrix_normalized2(1:15552, 4))';
% % dataQinput_q2(3, 31105:46656)=(datamatrix_normalized3(1:15552, 4))';
% % dataQinput_q2(3, 46657:62208)=(datamatrix_normalized4(1:15552, 4))';
% %
% % dataQinput_q2(4, 1:62208)=traiendoutput_q1;
% % dataQinput_q2(5, 1:62208)=traiendoutput_q3;
% % dataQinput_q2(6, 1:62208)=1-traiendoutput_q1-traiendoutput_q3;
% %
% % net_q2=newfit(dataQinput_q2, dataQoutput_q2, 20);
% % net_q2=train(net_q2, dataQinput_q2, dataQoutput_q2);
% %
% % traiendoutput_q2=sim(net_q2, dataQinput_q2);



%Widow-Hoff Backpropagation
% %p1
%
%
% net_p1=newff(dataPinput, dataPoutput, 20);
% net=init(net_p1);
%
% net_p1.trainParam.epochs=500;%(number of epochs)
% net_p1.trainParam.lr=0.3;%(learning rate)
% net_p1.trainParam.mc=0.05;%(momentum)
%
% net_p1=train(net_p1, dataPinput, dataPoutput);
%
% traiendoutput_p1=sim(net_p1, dataPinput);
%
% %q1
% % net_q1=newff(dataQinput, dataQoutput, 20);
% % net_q1=train(net_q1, dataQinput, dataQoutput);
%
% % traiendoutput_q1=sim(net_q1, dataQinput);
%
% %p3
% net_p3=newff(dataPinput, dataPoutput_P3, 20);
% net=init(net_p3);
%
% net_p3.trainParam.epochs=500;%(number of epochs)
% net_p3.trainParam.lr=0.3;%(learning rate)
% net_p3.trainParam.mc=0.05;%(momentum)
%
% net_p3=train(net_p3, dataPinput, dataPoutput_P3);
%
% traiendoutput_p3=sim(net_p3, dataPinput);
%
% %q3
% % net_q3=newff(dataQinput, dataQoutput_q3, 20);
% % net_q3=train(net_q3, dataQinput, dataQoutput_q3);
% %
% % traiendoutput_q3=sim(net_q3, dataQinput);
%
```

```
% %p2
% % dataPinput_p2(1:3, 1:15552)=(datamatrix_normalized1(1:15552, 1:3))';
% dataPinput_p2(1:3, 1:15552)=(datamatrix_normalized1(1:15552, 1:3))';
% dataPinput_p2(1:3, 15553:31104)=(datamatrix_normalized2(1:15552, 1:3))';
% dataPinput_p2(1:3, 31105:46656)=(datamatrix_normalized3(1:15552, 1:3))';
% dataPinput_p2(1:3, 46657:62208)=(datamatrix_normalized4(1:15552, 1:3))';
%
% dataPinput_p2(4, 1:62208)=traiendoutput_p1;
% dataPinput_p2(5, 1:62208)=traiendoutput_p3;
% dataPinput_p2(6, 1:62208)=1-traiendoutput_p1-traiendoutput_p3;
%
% net_p2=newff(dataPinput_p2, dataPoutput_P2, 20);
% net=init(net_p2);
%
% net_p2.trainParam.epochs=500;%(number of epochs)
% net_p2.trainParam.lr=0.3;%(learning rate)
% net_p2.trainParam.mc=0.05;%(momentum)
%
% net_p2=train(net_p2, dataPinput_p2, dataPoutput_P2);
%
% traiendoutput_p2=sim(net_p2, dataPinput_p2);
%
% %q2
% % dataQinput_q2(1:2, 1:15552)=(datamatrix_normalized1(1:15552, 1:2))';
% % dataQinput_q2(1:2, 15553:31104)=(datamatrix_normalized2(1:15552, 1:2))';
% % dataQinput_q2(1:2, 31105:46656)=(datamatrix_normalized3(1:15552, 1:2))';
% % dataQinput_q2(1:2, 46657:62208)=(datamatrix_normalized4(1:15552, 1:2))';
% %
% % dataQinput_q2(3, 1:15552)=(datamatrix_normalized1(1:15552, 4))';
% % dataQinput_q2(3, 15553:31104)=(datamatrix_normalized2(1:15552, 4))';
% % dataQinput_q2(3, 31105:46656)=(datamatrix_normalized3(1:15552, 4))';
% % dataQinput_q2(3, 46657:62208)=(datamatrix_normalized4(1:15552, 4))';
% %
% % dataQinput_q2(4, 1:62208)=traiendoutput_q1;
% % dataQinput_q2(5, 1:62208)=traiendoutput_q3;
% % dataQinput_q2(6, 1:62208)=1-traiendoutput_q1-traiendoutput_q3;
% %
% % net_q2=newff(dataQinput_q2, dataQoutput_q2, 20);
% % net_q2=train(net_q2, dataQinput_q2, dataQoutput_q2);
% %
% % traiendoutput_q2=sim(net_q2, dataQinput_q2);


%Default Scaled Conjugate Gradient Descent

%p1

net_p1=newpr(dataPinput, dataPoutput, 20);

net_p1.trainParam.epochs=500;%(number of epochs)
net_p1.trainParam.lr=0.3;%(learning rate)

net_p1=train(net_p1, dataPinput, dataPoutput);
```

```
traiendoutput_p1=sim(net_p1, dataPinput);

%q1
% net_q1=newpr(dataQinput, dataQoutput, 20);
% net_q1=train(net_q1, dataQinput, dataQoutput);

% traiendoutput_q1=sim(net_q1, dataQinput);

%p3
net_p3=newpr(dataPinput, dataPoutput_P3, 20);

net_p3.trainParam.epochs=500;%(number of epochs)
net_p3.trainParam.lr=0.3;%(learning rate)

net_p3=train(net_p3, dataPinput, dataPoutput_P3);

traiendoutput_p3=sim(net_p3, dataPinput);

%q3
% net_q3=newpr(dataQinput, dataQoutput_q3, 20);
% net_q3=train(net_q3, dataQinput, dataQoutput_q3);
%
% traiendoutput_q3=sim(net_q3, dataQinput);

%p2
% dataPinput_p2(1:3, 1:15552)=(datamatrix_normalized1(1:15552, 1:3))';
dataPinput_p2(1:3, 1:15552)=(datamatrix_normalized1(1:15552, 1:3))';
dataPinput_p2(1:3, 15553:31104)=(datamatrix_normalized2(1:15552, 1:3))';
dataPinput_p2(1:3, 31105:46656)=(datamatrix_normalized3(1:15552, 1:3))';
dataPinput_p2(1:3, 46657:62208)=(datamatrix_normalized4(1:15552, 1:3))';

dataPinput_p2(4, 1:62208)=traiendoutput_p1;
dataPinput_p2(5, 1:62208)=traiendoutput_p3;
dataPinput_p2(6, 1:62208)=1-traiendoutput_p1-traiendoutput_p3;

net_p2=newpr(dataPinput_p2, dataPoutput_P2, 20);

net_p2.trainParam.epochs=500;%(number of epochs)
net_p2.trainParam.lr=0.3;%(learning rate)

net_p2=train(net_p2, dataPinput_p2, dataPoutput_P2);

traiendoutput_p2=sim(net_p2, dataPinput_p2);

%q2
% dataQinput_q2(1:2, 1:15552)=(datamatrix_normalized1(1:15552, 1:2))';
% dataQinput_q2(1:2, 15553:31104)=(datamatrix_normalized2(1:15552, 1:2))';
% dataQinput_q2(1:2, 31105:46656)=(datamatrix_normalized3(1:15552, 1:2))';
% dataQinput_q2(1:2, 46657:62208)=(datamatrix_normalized4(1:15552, 1:2))';
%
```

```
% dataQinput_q2(3, 1:15552)=(datamatrix_normalized1(1:15552, 4))';
% dataQinput_q2(3, 15553:31104)=(datamatrix_normalized2(1:15552, 4))';
% dataQinput_q2(3, 31105:46656)=(datamatrix_normalized3(1:15552, 4))';
% dataQinput_q2(3, 46657:62208)=(datamatrix_normalized4(1:15552, 4))';
%
% dataQinput_q2(4, 1:62208)=traiendoutput_q1;
% dataQinput_q2(5, 1:62208)=traiendoutput_q3;
% dataQinput_q2(6, 1:62208)=1-traiendoutput_q1-traiendoutput_q3;
%
% net_q2=newpr(dataQinput_q2, dataQoutput_q2, 20);
% net_q2=train(net_q2, dataQinput_q2, dataQoutput_q2);
%
% traiendoutput_q2=sim(net_q2, dataQinput_q2);







%error calculation for the initial training
for i=1:number_of_points
    error1(i)=0.5*((traiendoutput_p1(i)-dataPoutput(i))^2);
    error3(i)=0.5*((traiendoutput_p3(i)-dataPoutput_P3(i))^2);
    error2(i)=0.5*((traiendoutput_p2(i)-dataPoutput_P2(i))^2);

%     errorq1(i)=0.5*((traiendoutput_q1(i)-dataQoutput(i))^2);
%     errorq3(i)=0.5*((traiendoutput_q3(i)-dataQoutput_q3(i))^2);
%     errorq2(i)=0.5*((traiendoutput_q2(i)-dataQoutput_q2(i))^2);

%     difference(i)=traiendoutput_p1(i)-dataPoutput(i);
end
error_sum_original_p1=sum(error1)/number_of_points
error_sum_original_p3=sum(error3)/number_of_points
error_sum_original_p2=sum(error2)/number_of_points

% error_sum_original_q1=sum(errorq1)/number_of_points
% error_sum_original_q3=sum(errorq3)/number_of_points
% error_sum_original_q2=sum(errorq2)/number_of_points



% x=(1:1:number_of_points)';
% plot(x, dataPoutput, x, traiendoutput);
% legend('The real Data', 'Trained Data');
% title('The real data v.s. the trained data');
% figure;
% plot(dataPoutput, traiendoutput);
% title('The real data v.s. the trained data');
%
% figure;
% subplot(2,1,1);
% plot(error);
% title('The mean square error of the real and model data');
% subplot(2,1,2);
% plot(difference);
% title('The difference of the real and model data');

% trained_net41_42=net;
% trained_net37=net;
% save NNtoolFitting_bus37.mat trained_net37
% save NNtoolFitting_a_f_trainedfault41_42.mat trained_net41_42
% save NNtoolFitting_a_f_trainedfault41_42f10000.mat trained_net41_42f10000

%Test on the other buses
%for p1
sumbus_error=0;
sumbus_error_p3=0;
sumbus_error_p2=0;
number_of_points2=15552;
for kkk=1:8
    datamatrix_normalized=[];
    error=0;
    error_sum=0;
    if kkk==1
        datamatrix_normalized=datamatrix_normalizedl37;
    elseif kkk==2
        datamatrix_normalized=datamatrix_normalizedl41;
    elseif kkk==3
        datamatrix_normalized=datamatrix_normalizedl42;
    elseif kkk==4
        datamatrix_normalized=datamatrix_normalizedl52;
    elseif kkk==5
        datamatrix_normalized=datamatrix_normalizedl36;
    elseif kkk==6
        datamatrix_normalized=datamatrix_normalizedl44;
    elseif kkk==7
        datamatrix_normalized=datamatrix_normalizedl43;
    elseif kkk==8
        datamatrix_normalized=datamatrix_normalizedl51;
    end

%initializing
dataPinput2=[];
dataPoutput2=[];
trainedoutput2=[];

dataPoutput2_P3=[];
trainedoutput2_P3=[];

dataPinput2_p2=[];
dataPoutput2_P2=[];
trainedoutput2_P2=[];


dataPinput2=(datamatrix_normalized(1:number_of_points2, 1:3))';
dataPoutput2=(datamatrix_normalized(1:number_of_points2, 5))';
traiendoutput2=sim(net_p1, dataPinput2);
```

```
dataPoutput2_P3(1,
1:number_of_points2)=(datamatrix_normalized(1:number_of_points2, 7))';
traiendoutput2_P3=sim(net_p3, dataPinput2);

dataPinput2_p2(1:3,
1:number_of_points2)=(datamatrix_normalized(1:number_of_points2, 1:3))';
dataPinput2_p2(4, 1:number_of_points2)=traiendoutput2;
dataPinput2_p2(5, 1:number_of_points2)=traiendoutput2_P3;
dataPinput2_p2(6, 1:number_of_points2)=1-traiendoutput2-traiendoutput2_P3;

dataPoutput2_P2(1,
1:number_of_points2)=(datamatrix_normalized(1:number_of_points2, 6))';
traiendoutput2_P2=sim(net_p2, dataPinput2_p2);





error_p1=[];
error_p2=[];
error_p3=[];
%error calculation
for i=1:number_of_points2
    error_p1(i)=0.5*((traiendoutput2(i)-dataPoutput2(i))^2);
    error_p3(i)=0.5*((traiendoutput2_P3(i)-dataPoutput2_P3(i))^2);
    error_p2(i)=0.5*((traiendoutput2_P2(i)-dataPoutput2_P2(i))^2);
    error_combined(i)=error_p1(i)+error_p3(i)+error_p2(i);

%      difference(i)=traiendoutput2(i)-dataPoutput2(i);
end
error_sum_p1=sum(error_p1)/number_of_points2;
error_sum_p3=sum(error_p3)/number_of_points2;
error_sum_p2=sum(error_p2)/number_of_points2;
error_sum=sum(error_combined)/number_of_points2/3;


    if kkk==1
        error37_trial=error_sum_p1;
        error37_trial_p3=error_sum_p3;
        error37_trial_p2=error_sum_p2;
    elseif kkk==2
        error41_trial=error_sum_p1;
        error41_trial_p3=error_sum_p3;
        error41_trial_p2=error_sum_p2;
    elseif kkk==3
        error42_trial=error_sum_p1;
        error42_trial_p3=error_sum_p3;
        error42_trial_p2=error_sum_p2;
    elseif kkk==4
        error52_trial=error_sum_p1;
        error52_trial_p3=error_sum_p3;
```

```
            error52_trial_p2=error_sum_p2;
        elseif kkk==5
            error36_trial=error_sum_p1;
            error36_trial_p3=error_sum_p3;
            error36_trial_p2=error_sum_p2;
        elseif kkk==6
            error44_trial=error_sum_p1;
            error44_trial_p3=error_sum_p3;
            error44_trial_p2=error_sum_p2;
        elseif kkk==7
            error43_trial=error_sum_p1;
            error43_trial_p3=error_sum_p3;
            error43_trial_p2=error_sum_p2;
        elseif kkk==8
            error51_trial=error_sum_p1;
            error51_trial_p3=error_sum_p3;
            error51_trial_p2=error_sum_p2;
        end
        sumbus_error_p1=sumbus_error+error_sum_p1;
        sumbus_error_p3=sumbus_error_p3+error_sum_p3;
        sumbus_error_p2=sumbus_error_p2+error_sum_p2;
    end
    average_error_trial_p1=sumbus_error_p1/8;
    average_error_trial_p3=sumbus_error_p3/8;
    average_error_trial_p2=sumbus_error_p2/8;

    average_error_trial=(average_error_trial_p1+average_error_trial_p2+average_er
    ror_trial_p3)/3;

    % if average_error_trial<average_error
        average_error(trial)=average_error_trial;


    end

    end
        average_error
        d = {'error1' average_error(1); 'error2' average_error(2); 'error3'
    average_error(3); 'error4' average_error(4); 'error5' average_error(5);
    'error6' average_error(6); 'error7' average_error(7); 'error8'
    average_error(8); 'error9' average_error(9); 'error10' average_error(10)}

        xlswrite(Default_Scaled_Conjugated_Gradient_caseE2', d, 'Testing buses
    E2', 'A1');


    %     xlswrite('testedbuses', d, 'Testing buses', 'A1');

    %     if Atrial==1
    %         xlswrite('testedbuses_E', d, 'Testing buses E1', 'A1');
    %         xlswrite('testedbuses_E', d3, 'Testing buses E1', 'D1');
    %         xlswrite('testedbuses_E', d2, 'Testing buses E1', 'G1');
    %         save pattern_recognition_caseE1 net_p1 net_p2 net_p3
```

```
%     elseif Atrial==2
%         xlswrite('testedbuses_E2', d, 'Testing buses E2', 'A1');
%         xlswrite('testedbuses_E2', d3, 'Testing buses E2', 'D1');
%         xlswrite('testedbuses_E2', d2, 'Testing buses E2', 'G1');
%         save pattern_recognition_caseE2 net_p1 net_p2 net_p3
%     elseif Atrial==3
%         xlswrite('testedbuses_E3', d, 'Testing buses E3', 'A1');
%         xlswrite('testedbuses_E3', d3, 'Testing buses E3', 'D1');
%         xlswrite('testedbuses_E3', d2, 'Testing buses E3', 'G1');
%         save pattern_recognition_caseE3 net_p1 net_p2 net_p3
%     elseif Atrial==4
%         xlswrite('testedbuses_E4', d, 'Testing buses E4', 'A1');
%         xlswrite('testedbuses_E4', d3, 'Testing buses E4', 'D1');
%         xlswrite('testedbuses_E4', d2, 'Testing buses E4', 'G1');
%         save Default_Scaled_Conjugated_Gradient_caseE2 net_p1 net_p2 net_p3
%
%     end
% end
```