Power System Fault Detection Using the Discrete Wavelet Transform and Artificial Neural Networks

By

Kevin Keegan

Advisor: Xiao-Hua Yu

Senior Project

ELECTRICAL ENGINEERING DEPARTMENT

California Polytechnic State University

San Luis Obispo

June 2014

TABLE OF CONTENTS

List of Tables

List of Figures

Abstract:

This project focuses on detecting various phase to ground faults in three phase power systems. In this research, the faults are generated using a power distribution system simulator; and the three phase voltage waveforms are analyzed using the discrete wavelet transform. Multi-layer feed forward neural networks are employed for fault detection and classification. The effectiveness of this approach is demonstrated by computer simulation results.

I. Introduction:

Power system fault detection has always been an area of importance in transmitting signals across power distribution systems. The systems operate in the kV range, thus have significant current flowing through the lines. This results in high power. A single fault, even lasting for a fraction of a second, can cause huge losses and manufacturing downtime in industrial applications [1]. This creates high demand for improvements in power fault detection systems and ways to reduce or avoid the occurrence of problems with power distribution systems. Intensive studies are done in the area of power system fault detection in order to avoid any economic losses caused by a power system fault.

The primary goal of this project is to find sufficient ways to detect fault conditions correctly as quickly as possible so that action can be taken to minimize the damage caused by the fault. The process begins with generating known faults at known times using the Simulink and SimPowerSystems MATLAB add-ons. The discrete wavelet transform is used as the feature extraction technique for the project. Once feature extraction is performed, the power residing in the decomposition is calculated, and input and output files of the neural network are created. Training of the neural network is accomplished with these files.

Comparisons were done on the feature extraction capabilities of different detail levels, as well as the success of the two different mother wavelets. Different mother wavelets have different filter coefficients, which result in different wavelet decompositions. A conclusion was made as to the best mother wavelet and decomposition level for the feature extraction aspect of this project.

Two tests will be performed. The first test consists of changing the double phase faults to double phase to ground faults with various ground resistances. The performance of the neural network will be analyzed under the chosen ground resistances. The second test consists of analyzing the performance of the neural network under different noise conditions. Different amounts of noise will be added to the neutral line of the three-phase source in the Simulink diagram. Again, the performance of three different neural networks will be analyzed under these different conditions.

II. Literature Review

[1] Settipalli, Praveen. May 2007. "Automated Classification of Power Quality Disturbances Using Signal Processing Techniques and Neural Networks." University of Kentucky, 2007.

Praveen Settipalli was a graduate student at the University of Kentucky who wrote his master's thesis titled "Automated Classification of Power Quality Disturbances Using Signal Processing Techniques and Neural Networks." The paper was completed in 2007, and offers a recent synopsis of power system fault detection techniques. One of the techniques used for feature extraction in the project was the wavelet transform. The proposed general method for power system fault detection was signal generation, feature extraction, neural network training, classification, and decision-making. For the classification of fault conditions, many different approaches can be used. The most common are fuzzy logic, adaptive fuzzy logic, and artificial neural networks. Fuzzy logic and adaptive fuzzy logic use different combinations of wavelet transforms and Fourier transforms.

[2] Patel, Mamta. June 2012. "Fault Detection and Classification on A Transmission Line using Wavelet Multi Resolution Analysis and Neural Network."

Patel has a PhD from the Government Polytechnic Durg, in India, from the department of Electrical Engineering. He pushes that multi-resolution analysis is a popular method for feature extraction. The wavelet transform is a widely used for multi-resolution analysis. Patel describes how a fault initiates a transient condition, which results in high frequency components in the voltage or current fault signals. The various ways proposed to extract the important information in these high frequency components include, but are not limited to, Fourier Transforms, Wavelet Transforms, Neural networks, or fuzzy logic. Wavelets have been proven to be a phenomenal trade-off between time accuracy and frequency resolution. The fact that the user has useful information about both time and frequency simultaneously labels this method as a superior method for frequency extraction.

[3] Kasinathan, Karthikeyan. 2007. "Power System Fault Detection and Classification by Wavelet Transforms and Adaptive Resonance Theory Neural Networks." University of Kentucky, 2007.

Karthikeyan Kasinathan has a master's degree from the University of Kentucky and wrote the thesis on power system fault detection using wavelet transforms and neural networks in 2006. Kasinathan uses two different unsupervised adaptive resonance theory neural networks for the classification stage of the system. The two neural networks used were back propagation neural network and fuzzy logic classification. The faults in this thesis are classified via 3 different characteristics: fault type, fault location, and fault resistance. In the project in this paper, faults will only be classified via the type of fault that has occurred. In the thesis, Kasinathan highlights that self-organizing neural networks can train and learn independently from external feedback. This is an attractive property for the classification stage of the design. A general description of the operation of fuzzy logic is also offered in the paper. An entire data set is processed once, forming unstable clusters, and the unstable clusters are iteratively processed until all clusters are stable. The means of association is the Euclidean distance between past and present input data.

[4] Sathiya priya, K. Geethanjali, M. “Combined Wavelet Transforms and Neural Network (WNN) Based Fault Detection and Classification in Transmission Lines.” 2006.

Priya and Geethanjali use MATLAB and Simulink to simulate many different types of faults, where this project only utilized one type of fault, specifically single (and double) phase to ground faults. Each phase fault was simulated by a fault to ground. They also simulate single phase to ground fault, line-to-line fault, double line to ground, three-phase short circuit, and capacitor switching and breaker operation. The classification scheme is a three layer neural network utilizing the back-propagation learning algorithm. The training and testing of the neural network is done using the Discrete Wavelet Transform. This is the process understood in this project to create the input files to the feed-forward neural network for training. Sathiya and Geethanjali highlight HTL (High Voltage Transmission Line) fault detection and classify this detection process into three different methods: circuit theory, travelling theory, and intelligent systems.

[5] Lampley, Glenn C. “Fault Detection and Location on Electrical Distribution System.” IEEE. Carolina Power & Light. 2002.

Lampley offers the detection of faults and the fault location in the electrical distribution system. In this project, fault conditions are intentionally generated, and thus known, at a specific point along a transmission line. As a result, the location of the fault occurrences in this project is fixed. Lampley implemented a different approach for fault detection and location detection. Lampley used a Feeder Monitoring System, an Automated Outage Management System, and a Distribution SCADA system in order to detect faults and their locations. Using these systems provided a graphical display of possible locations for faults that have locked out feeder circuit breakers [5].

[6] Xiaohua, Yang. Yadong, Zhang. Zhongmei, Xi. “Wavelet Neural Network Based Fault Detection Method in Power System.” IEEE. 2003.

Yang, Lai Wu, and Zhang talk about fault detection methods in a power system using the wavelet transform. They state that wavelet analysis is a significant mathematical tool that has gained a lot of momentum in recent years and is being more widely used for certain applications such as neural networks. Applications of the wavelet transform are used in signal analysis containing fault conditions, and detecting faults and harmonics. The project uses the wavelet transform and artificial neural networks together. The wavelet transform requires significant construction and storage space for large-scale applications, and neural networks are strong at handling large data sets and problems. The wavelet neural network used in the project combines the properties of the two components, both with good time and frequency localization, in order to detect faults in a power system. It is shown in the paper that wavelet neural networks have faster convergence rates (smaller training times) than a regular artificial neural network.

[7] Blumenstein, Michael, Xin Yu Liu, and Brijesh Verma. "Investigation of the Modified Direction Feature for Cursive Character Recognition." *ScienceDirect*. Elsevier, 14 May 2006. Web. 20 Oct. 2013. <http://dl.acm.org/citation.cfm?id=1221195>

Blumenstein, Liu, and Verma published this paper on efficient techniques of character recognition on ScienceDirect. It describes a feature extraction technique for the recognition of not plain block letters, but letters more in the style of handwriting, i.e. segmented or cursive characters. The feature extraction technique is a form of a modified direction feature technique that is used for extracting general attributes from characters written in a cursive style. They use local vectors alongside global information to provide integrated features to a neural network for training in pattern recognition. The directional vectors are obtained by a character outline tracing technique, while the global information is obtained by comparing the location of background to foreground pixel transitions [7]. The paper offers an additional use and application of neural networks, and lays a solid foundation for the idea of feature extraction, as in this project feature extraction techniques were also used (namely, the discrete wavelet transform).

[8] Huang, Shyh-Jier, and Cheng-Tao Hsieh. "High-Impedance Fault Detection Utilizing A Morlet Wavelet Transform Approach." *IEEE*. N.p., Oct. 1999. Web. 14 Nov. 2013.

Huang and Hsieh analyze the application of specific types of wavelets: the Morlet wavelets. The Morlet wavelets are used to analyze high-impedance fault generated signals. The wavelet transform is used to distinguish these high-impedance faults from normal switching events. Normal switching events are events that may also happen under normal system conditions such as transient conditions or arcs. It is important to distinguish between faults and these cases. The current most common detection technique for high impedance faults involves overcurrent protective devices. However, this causes unexpected service interruptions, because the current variations are not significantly different than that of harmless events during operation. A brief background of the wavelet transform and its properties is presented. The wavelet transform is a powerful tool that allows for knowledge of both time and frequency simultaneously. This time localization proposes it as a superior method for knowledge of time varying signals than the fast Fourier transform, as the Fourier transform offers no knowledge of time domain characteristics.

[9] Kennedy, James, and Russell Eberhart. "Particle Swarm Optimization." IEEE. 1995. Web.

Kennedy and Eberhart introduce a method known as particle swarm optimization for optimizing non-linear functions that are continuous. They suggest that particle swarm optimization has ties to common artificial intelligence methodologies such as bird flocking, fish schooling, and swarming theory [9]. The paper highlights the simplicity of such methodologies, stating that particle swarm optimization can be implemented cheaply, quickly, and with trivial mathematical operations. Another application that served as a motive for developing swarm optimization was human social behavior. It is more abstract than the previously mentioned examples, which have largely predictable factors. Humans do not usually turn in unison, as a flock of birds may. Swarm optimization also has ties in evolutionary computation, and more specifically genetic algorithms. The approach used in this paper uses the particle swarm optimization algorithm to train the weight functions within an artificial neural network. It offers a possible alternative training method to that used in this project: the back-propagation training algorithm.

[10] Narendra, Kumpati S., and Kannan Parthasarathy. "Identification and Control of Dynamical Systems Using Neural Networks." *IEEE*. N.p., Mar. 1990. Web. 5 Nov. 2013.

Narendra and Parthasarathy offer another application for artificial neural networks. The application for neural networks used in this paper was for the identification and control of nonlinear dynamical systems. Similar training techniques were used in this paper as were used in this project. They implemented both static and dynamic back propagation training techniques for parameter adjustment. The paper highlights two popular types of artificial neural networks, multilayer neural networks and recurrent networks. Multilayer neural networks have been commonly applied with high success in pattern recognition applications, along with static nonlinear maps. Recurrent networks have been successfully applied to optimization problems and embody dynamic feedback systems. The primary goal of the paper is to use neural networks to suggest identification and control of nonlinear dynamical systems. Most advancements in this area have only been made in the design of controllers for linear systems, not nonlinear systems. This paper offers an alternate application for neural networks, utilizing a highly desirable property of the neural network: the ability to adapt and deal with highly nonlinear mathematical functions.

[11] Tayeb, Eisa Bashier M. "Faults Detection in Power Systems Using Artificial Neural Network." *American Journal of Engineering Research* 2320-0847 02.06 (2013): 69-75. Web. 30 Dec. 2013.

Tayeb writes about using artificial neural networks to detect faults in electrical systems. The faults that occur in electrical systems directly cause discontinuities in the supply of electricity. Fault detection techniques employed by artificial neural networks offer a protection system that detects and isolates faults as quickly as possible in order minimize the damage caused to the power distribution system. This is ultimately one of the goals of the fault detection techniques also used in this project. A three phase power distribution system is created with known fault conditions in order to train a neural network to detect the fault conditions as soon as possible. The paper points out that various different types of faults can occur anywhere along the transmission lines in the electrical system. The paper classifies faults into two main areas: active and passive. The majority of short-circuit faults tend to occur on overhead lines [11]. The faults considered in the paper were single phase-to-ground, double phase and double phase-to-ground faults.

[12] Poungponsri, Suranai, and Xiao-Hua Yu. "Electrocardiogram (ECG) Signal Modeling and Noise Reduction Using Wavelet Neural Networks." Proceedings of the IEEE International Conference on Automation and Logistics, Aug. 2009. Web. Nov. 2013.

Poungponsri and Yu wrote this paper about accomplishing electrocardiogram signal modeling and noise reduction through the use of the wavelet transform and neural networks. The feature extraction technique used in this project is also the discrete wavelet transform, which is used to determine the inputs to the artificial neural network. The Electrocardiogram signal has been used in cardiac pathology quite frequently as a means to reveal heart disease. It does so through a process

similar to the process used in this project: feature extraction and neural network training and validation. This paper offers insight into applications and diversity of the wavelet transform, and a different learning algorithm was used for the neural network than was used in this project. Because the wavelet transform offers both information in time and frequency, it has a multi-resolution property. The approach used in the paper combines this property of wavelets along with training the neural network using Adaptive Diversity Learning Particle Swarm Optimization. The method of training used in this project was the Levenberg-Marquardt back-propagation training algorithm. The swarm optimization technique known as Gradient Descent Optimization was also used in training of the neural network. This optimization technique is used to find local minimums by stepping in the direction of the steepest descent.

[13] Wang, Xiao-bin, Guang-yuan Yang, Yi-chao Li, and Dan Liu. "Review on the Application of Artificial Intelligence in Antivirus Detection System." University of Electronic Science and Technology of China. IEEE. 2008.

Wang, Yang, Li, and Liu offer another application for artificial intelligence techniques. Artificial intelligence continues to play an expanding role in anti-virus detection. Artificial intelligence will improve the performance of such detection systems and promote the growth of new algorithms to be implemented in anti-virus detection techniques. The paper briefly talks about five main artificial intelligence techniques applied in the field of anti-virus detection: Heuristic technique, data mining, Agent technique, artificial immune, and artificial neural networks [13]. The paper illustrates that artificial neural networks is the most prominent method, because it solves a problem with lack of associative memory and the capacity of real-time calculation. The paper highlights properties of the neural network that give it the ability to deal with these problems. The artificial neural network has parallel storage and processing, the ability to self organize, adaptive capabilities, and self-learning abilities. Neural networks used in this paper contained one hidden layer. In this project, one and two layer hidden layer configurations were analyzed.

[14] Lyons, Richard. "Understanding Cascaded-Integrator Comb Filters" Embedded. 31 March. 2005. 30 March. 2014 <http://www.embedded.com/design/configurable-systems/4006446/Understanding-cascaded-integrator-comb-filters>

Cascaded-integrator-comb (CIC) filters were used in the Simulink model for the neural network processing after training of the neural network. The purpose of the CIC filters is to take each of the detail level 4 coefficients and provide a moving average as a way to track the occurrence of a fault. CIC filters are efficient implementations of low pass filters used for interpolation or decimation. In this project, the CIC filter will be used for decimation to decrease the data rate input to the neural network and provide more efficient computations. Because the magnitude response is that of a sinc function, the phase is non-linear. To compensate for this, an FIR filter is placed either before or after the CIC filter to flatten the pass band and provide a more linear phase response. CIC filters are also used for anti-imaging filtering for interpolated signals. With an increase in beneficial applications for the CIC filter, the interest for the CIC filter in wireless communications and signal processing has increased substantially.

III. Approach/Algorithm:

As mentioned in [1] and [2], there are many types of faults in a power distribution system, including single phase to ground faults, line-to-line faults, double line to ground faults, and three-phase short circuit faults. A particular fault was chosen to be the focus of this project: phase to ground faults (both single and double phase). A fault occurs when the signal momentarily connects to a component of the power distribution system other than where it should be. In the case of a phase to ground fault, a certain phase of the power distribution network is connected directly to ground, unless there is some small amount of ground resistance. All of the single phase-to-ground faults were chosen (A faults, B faults, and C faults). Only one double phase to ground fault was chosen: the AC fault. A "no" fault was also simulated as a reference point. This makes for five different conditions. The top-level block diagram of the approach is shown in Figure 1. This is similar to that proposed in [1].

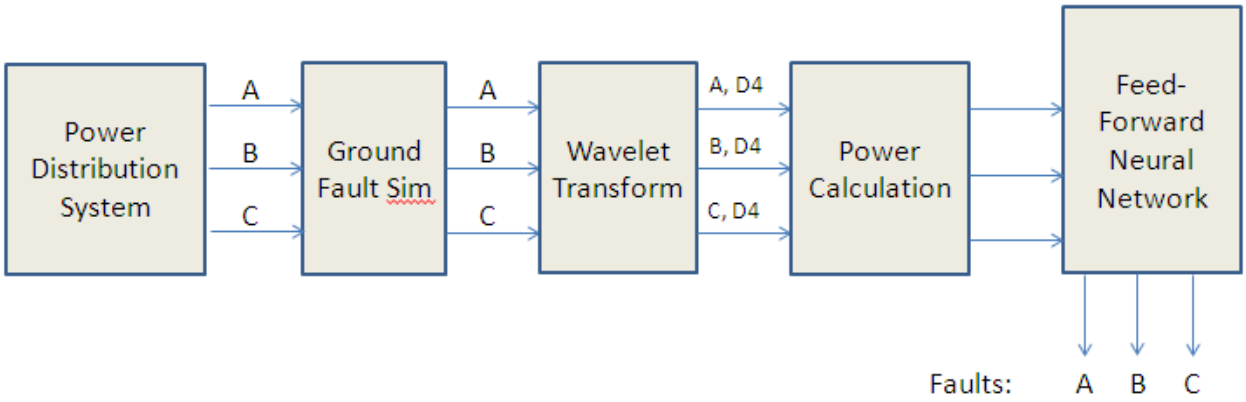

Figure 1: Top-Level Block Diagram

Simulink Diagram:

The first two blocks of figure 1 are accomplished by the Simulink design. The 3-phase power distribution system has been created and the phase to ground faults have been simulated and recorded. Simulink and the SimPowerSystems Matlab toolbox were used to generate 20 phase to ground faults from each of five different phase-to-ground faults: A-Phase, B-Phase, C-Phase, AC-Phase, and no phase faults. This creates a data set of 100 faults in total. By extending the simulation time, more faults can be simulated and obtained. Figure 2 shows a picture of the complete system designed in Simulink to both create and capture these faults.

The Three-Phase Source generates a sinusoidal signal with a frequency of 50 Hz. It is a balanced source, so each phase is 120 degrees out of phase with each other. The phase-to-phase voltage of the sinusoidal signal generated by this source is 10,000 V_{RMS}. The internal connection is a 'Yn' connection, meaning the three voltage sources are connected in a 'Y' configuration and the neutral line is an input to the three phase source that can be manually controlled. If no noise is to be introduced to the system, the neutral line is connected to the Simulink constant '0', emulating ground.

The controlled voltage source labeled 'Control' is an interface that allows a Simulink block to control the three-phase power source, which is from SimPowerSystems. Blocks from SimPowerSystems all need an input and output interface to other Simulink blocks. This controlled voltage source allows the

introduction of noise from the band-limited white noise source to the neutral line of the three-phase power source. The manual switch allows a choice between the Simulink constant '0' (for no noise) and the white noise source. To observe the neutral line, and thus the noise applied to the source, a SimPowerSystems Voltage Measurement block is required to interface to a scope.

The powergui block specifies a sampling time of 61.035µs for the system. This corresponds to a sample rate of 16384 Hz (2^{14}). There are other analysis functions in the powergui that are unused, however it is a required block to implement the functions within the SimPowerSystems toolbox.

The first distribution parameter line is representing a long transmission line from the three-phase source to the point where the phase-to-ground faults are being created. This section of the transmission line is 150km long. It is a 3-phase transmission line with 50 Hz used for RLC specifications of the line, in order to match-up with the 3-phase source parameters. Positive, negative, and zero sequence impedances are specified because the line consists of symmetrical components. This is a very important property for the power lines to have to allow one to analyze the operation of the power system during unbalanced states, such as the occurrence of faults. For reference, the positive and zero sequence impedances of the transmission lines used in the Simulink model are given below. This transmission line is continually transposed, meaning the positive and negative sequences are equal.

The second distribution line in the system is representing a transmission line 50 km long, and is also a 3-phase transmission line with 50 Hz used for RLC specifications of the line. The resistance, capacitance, and inductance per unit length for the positive, negative, and zero sequences of both transmission lines are also those described in table 1. These are the standard parameter values used in the Simulink model, and are used during this simulation. All of these parameters can be varied upon user's choice.

Parameter	Positive Sequence	Zero Sequence
Resistance per unit length (Ohms/km)	0.01273	0.3864
Capacitance per unit length (H/km)	12.74×10^{-9}	7.751×10^{-9}
Inductance per unit length (F/km)	0.9337×10^{-3}	4.1264×10^{-3}
Line Length (km)	150, 50	

Table 1: Parameters of Transmission Lines.

There are six blocks that are creating a fault 150 km down the transmission line (between transmission lines 1 & 2). One block simulates a fault on the transmission line for each of the 6 different faults being considered: A, B, C, AB, BC, and AC faults. There is no block simulating no faults, as this condition is the condition of the system when no fault is present. The fault simulator labeled 'A Phase Fault' is set to provide a phase to ground fault in the A phase of the 3-phase system upon an external fault control input to the fourth port of the block. The fault simulators for each single phase fault are programmed to provide the corresponding phase to ground fault via an external triggering source. In order to properly simulate a fault to ground, the user must input a small non-zero ground resistance Rg. In the case of this Simulink model, the ground resistance was 0.001Ω. The double phase faults are phase to phase faults and not faults to ground. For instance, the AB fault is a fault between phases A and B.

Each fault simulator has a fault generator that is labeled 'Pulse Generator 1-6' on the Simulink Diagram of figure 2. Each generator corresponds to the simulator that takes the pulse as its fourth input port, via the diagram. All six pulses share parameters specified below in table 2. The period is 7.002 seconds, and the pulse width is 0.5% of the period. At the sampling rate of 16.384 kHz, the fault occurs for 574 samples (0.005x7.002x16.384k). Each fault is generated when the pulse from these fault generators is high. The phase delays for the six pulse generators are 1 second apart from each other. This specifies when the fault will occur. The period is 7 seconds to allow 1 more second for the occurrence of the no fault condition. Thus, each type of fault occurs at 7-second intervals for 140 seconds. This creates 20 faults of each type of fault, or 140 faults, as the sample size. The phase delays were arbitrarily chosen to be 0.29 seconds into each 1-second interval, and in order to make sure the fault began and ended within that 1-second interval, the pulse width (duration of the fault) was chosen to be 0.5% of the period. Another reason the pulse width was arbitrarily chosen as 0.5% is because it is a small duration of occurrence, and the smaller the fault duration the harder the fault is to detect.

Amplitude	1
Period [seconds]	7.002
Pulse Width (% of period)	0.5
Phase Delay (Pulse Generator 1) [sec]	0.29
Phase Delay (Pulse Generator 2) [sec]	1.29
Phase Delay (Pulse Generator 3) [sec]	2.29
Phase Delay (Pulse Generator 4) [sec]	3.29
Phase Delay (Pulse Generator 5) [sec]	4.29
Phase Delay (Pulse Generator 6) [sec]	5.29

Table 2: Parameters for Fault Generators

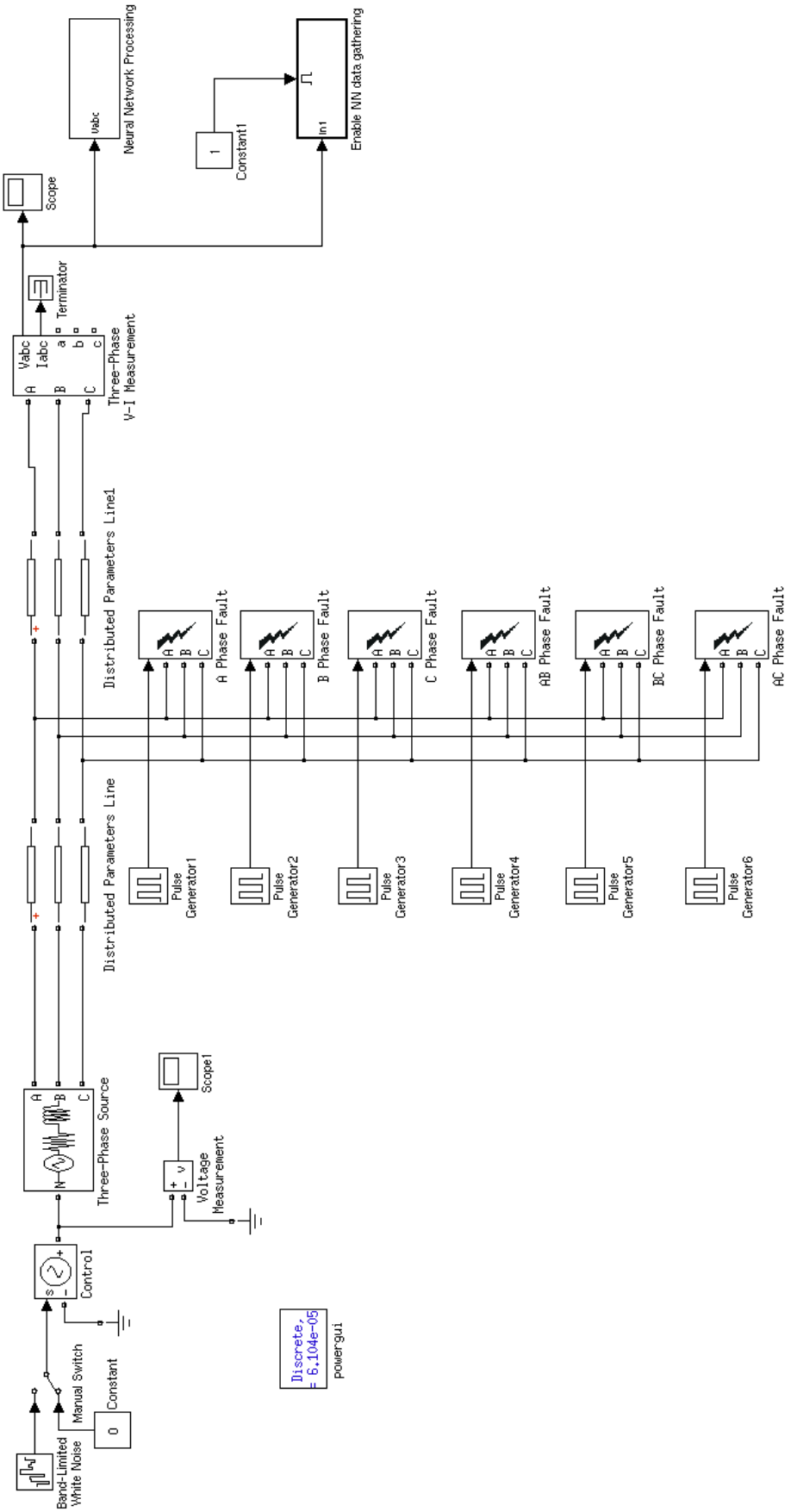

Figure 2: Simulink Model of Network created using SimPowerSystems

At the end of the second transmission line, there is a line termination that functions as the measurement interface. This block allows voltage and current measurements of all three phases to be taken simultaneously. The block is needed to take measurements from a SimPowerSystems block. The three inputs to the block are the three phases of the transmission line, and the output of interest is the 1x3 vector of voltages V_{ABC}. Data of the simulated faults on these voltages will be recorded. This signal, V_{ABC}, is processed by two subsystems. The first is the data-gathering block, labeled 'Enable NN data gathering' in figure 2. This subsystem provides known fault data for training the neural network. It is enabled by a second input to allow for disabling during long neural network processing intervals. This reduces the amount of data written to the workspace and makes the simulation run faster. The second block labeled 'Neural Network Processing' in figure 2, which takes the signal V_{ABC} and processes the data using the discrete wavelet transforms for feature extraction and neural networks to provide fault detection.

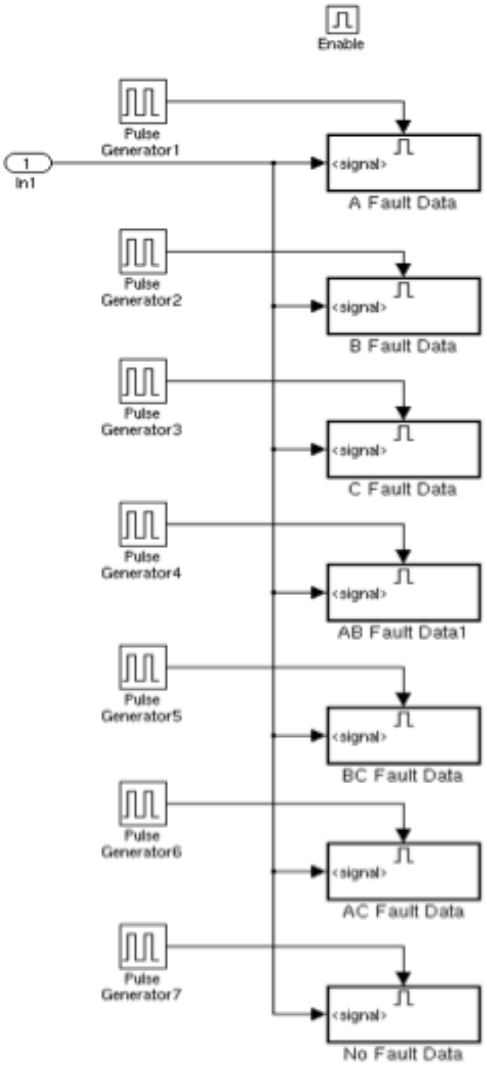

Figure 3: Enable NN Data Gathering

The subsystem for data gathering is shown in figure 3. The sample memories are blocks that record data on the faults created by the fault simulators and save this data to the workspace. Each of the seven different phase faults (including the no-fault condition) requires a sample memory to save the data recorded to the workspace. Each block is also externally controlled by a pulse generator that acts as the enable for the data capture. The seven structures created were used as inputs to the wavelet network. Each structure characterizes each phase of the power system. Take the first structure as an example. This represents the A, B, and C phases of the system during an A fault. Thus, it is in a matrix of 3 columns: one for the A phase, B phase, and C phase respectively, representing the 20 phase faults for each fault type. These 7 sample memory blocks capture known errors and fault conditions that can be used to train the neural network.

The 'Pulse Generators 1-7' on the subsystem diagram (figure 3) are the enable signals to allow the sample memories to record data during a specific fault. The period of the pulse is 7 seconds, and the duration of the pulse is 1 second (1/7 of the period, or 100/7%). The data gathering for the fault lasts for 1 second, or 16384 samples per fault. The phase delays for these enable signals are 0 to have the observation time be the full second the fault condition will occur, ensuring the entire fault is captured.

Note that the period of the fault generators is 7.002 seconds, while the periods of the sample memory enable signals is 7 seconds. The difference in these periods was intentional to cause each of the faults to occur at different times, or different phases, within the sample period. The transmission line parameter values are held constant throughout all of the faults. To illustrate the difference between some of the faults, 4 different A-phase faults spread throughout the 20 A-phase faults were captured and are displayed in figures 4-7.

The A-phase is the blue sinusoid in figures 4-7. It can be seen that the behavior of phases A, B, and C after the occurrence of the A fault largely depends upon the angular phase of the A-phase when the fault occurs. An interesting observation is that the amount of distortion in all three phases as a result of the fault is greater the larger the amplitude of the A-phase signal (for an A-fault). The amount of distortion in the 3 phases also has a proportional correlation with the amplitude of the phase where the fault occurs at the point of the fault.

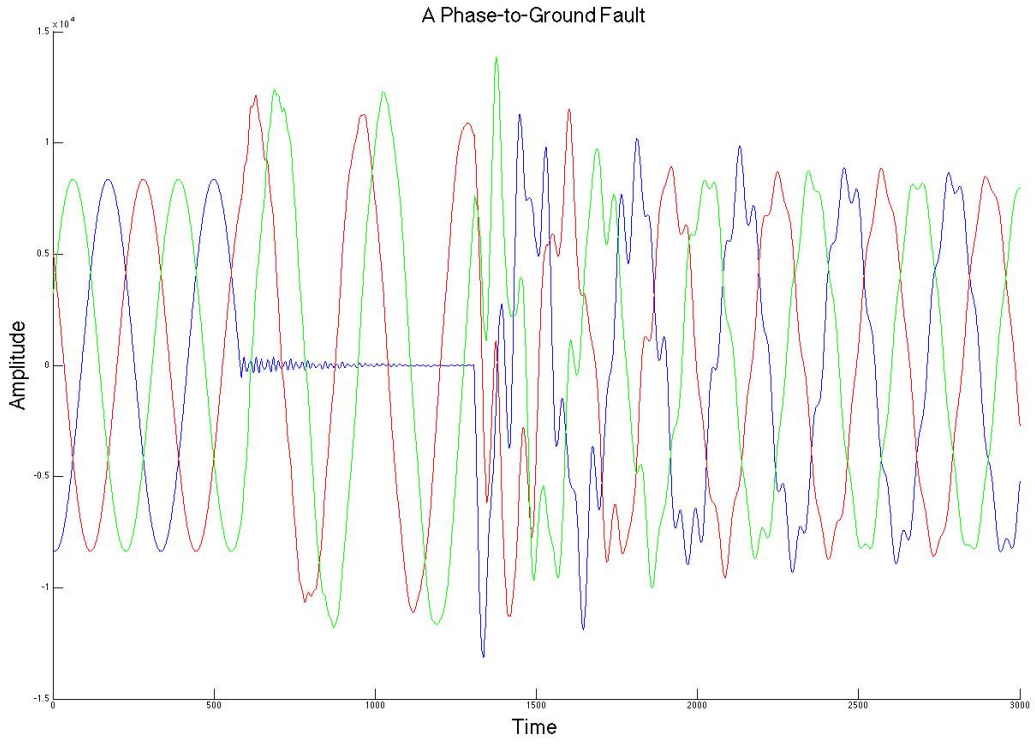

Figure 4: Sample A-fault #1

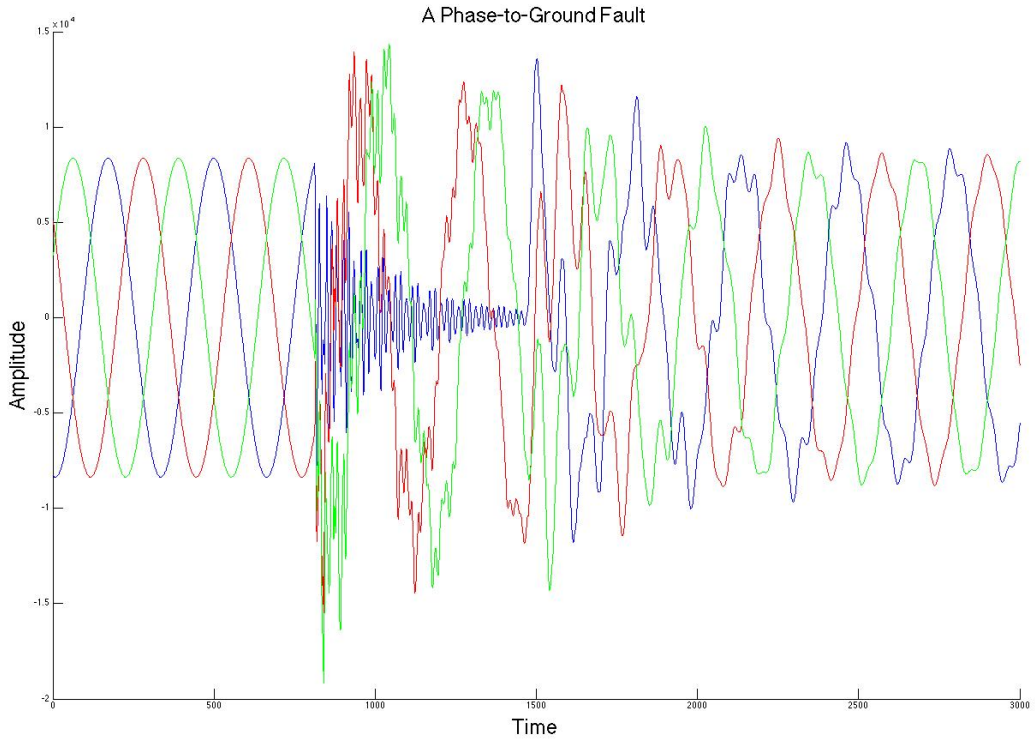

Figure 5: Sample A-fault #2

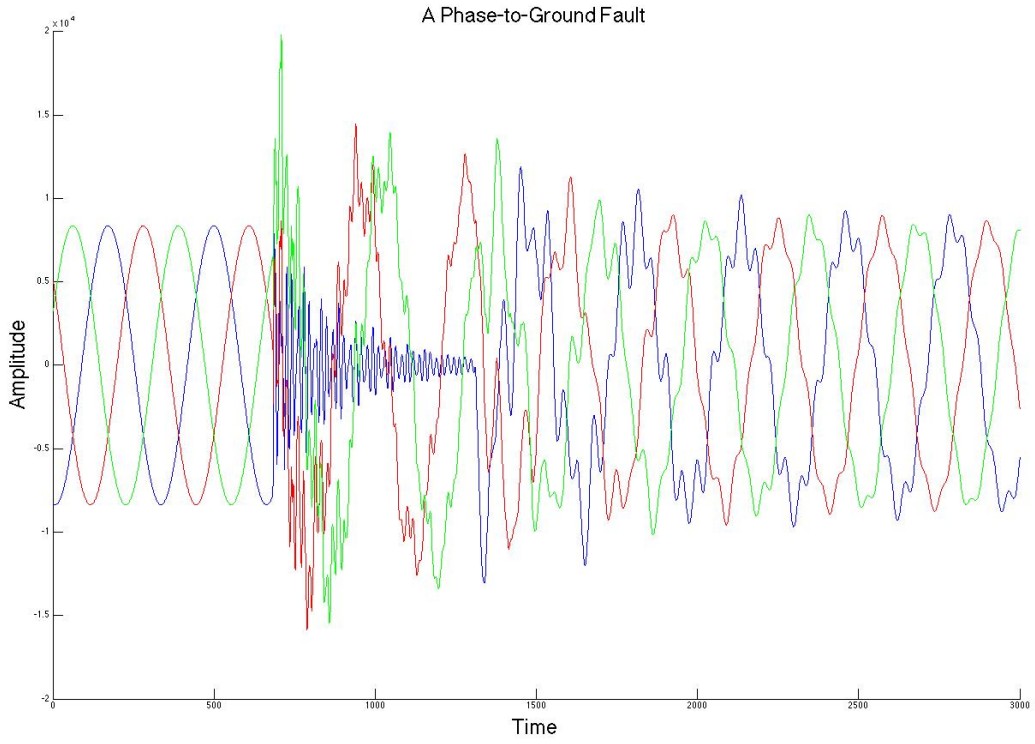

Figure 6: Sample A-fault #3

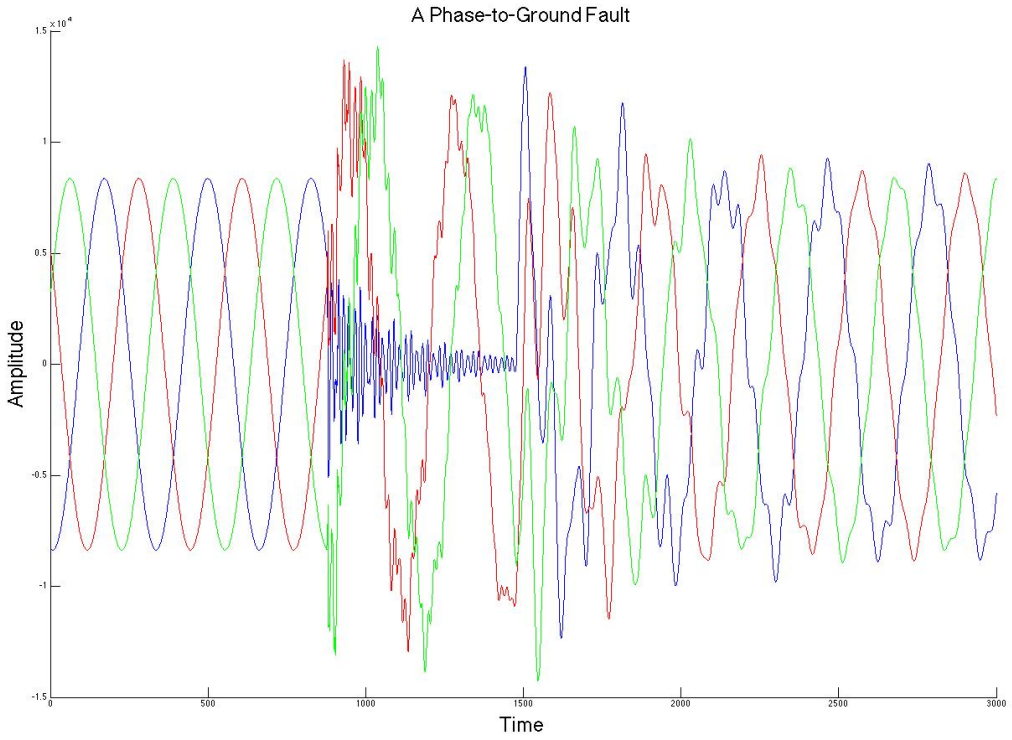

Figure 7: Sample A-fault #4

Figure 8 shows the Neural Network Processing subsystem. This subsystem provides the fault detection and classification. The input is fed into the demultiplexor (Simulink Demux). The demux separates the A, B, and C phase signals of the transmission line, which provides a phase of the transmission line to each one of the DWT blocks through a buffer. The buffer is a serial to parallel interface that takes a time series and converts it to a 1 second block of data to be processed by each DWT block. Each DWT block has parameters for the filter type, the mother wavelet, and the decomposition level to be applied. The discrete wavelet employed uses mother wavelet DB4, decomposition level 6, and Debauchee filter types. A setting of the block can enable access to each of the decomposition levels, as well as the resulting approximation level. The diagram shows that the further processing will be done on detail level 4, as the outputs coincide top to bottom with detail levels 1-6, and finally approximation 6.

The detail level 4 outputs for each of the DWT blocks has length 1024, since there are 16384 samples per second and detail level four divides that number of samples by 2^4. For more information, refer to figure 18. This frame of data is serialized by the unbuffer blocks at the outputs to each of the 3 DWT blocks. The serialized data is fed into 1 block for each phase that squares the input, giving a metric proportional to power. The power estimate is input into a cascaded integrator-comb filter (CIC), 1 for each phase, that develops a moving average of the power. Details of the CIC filter will be discussed later.

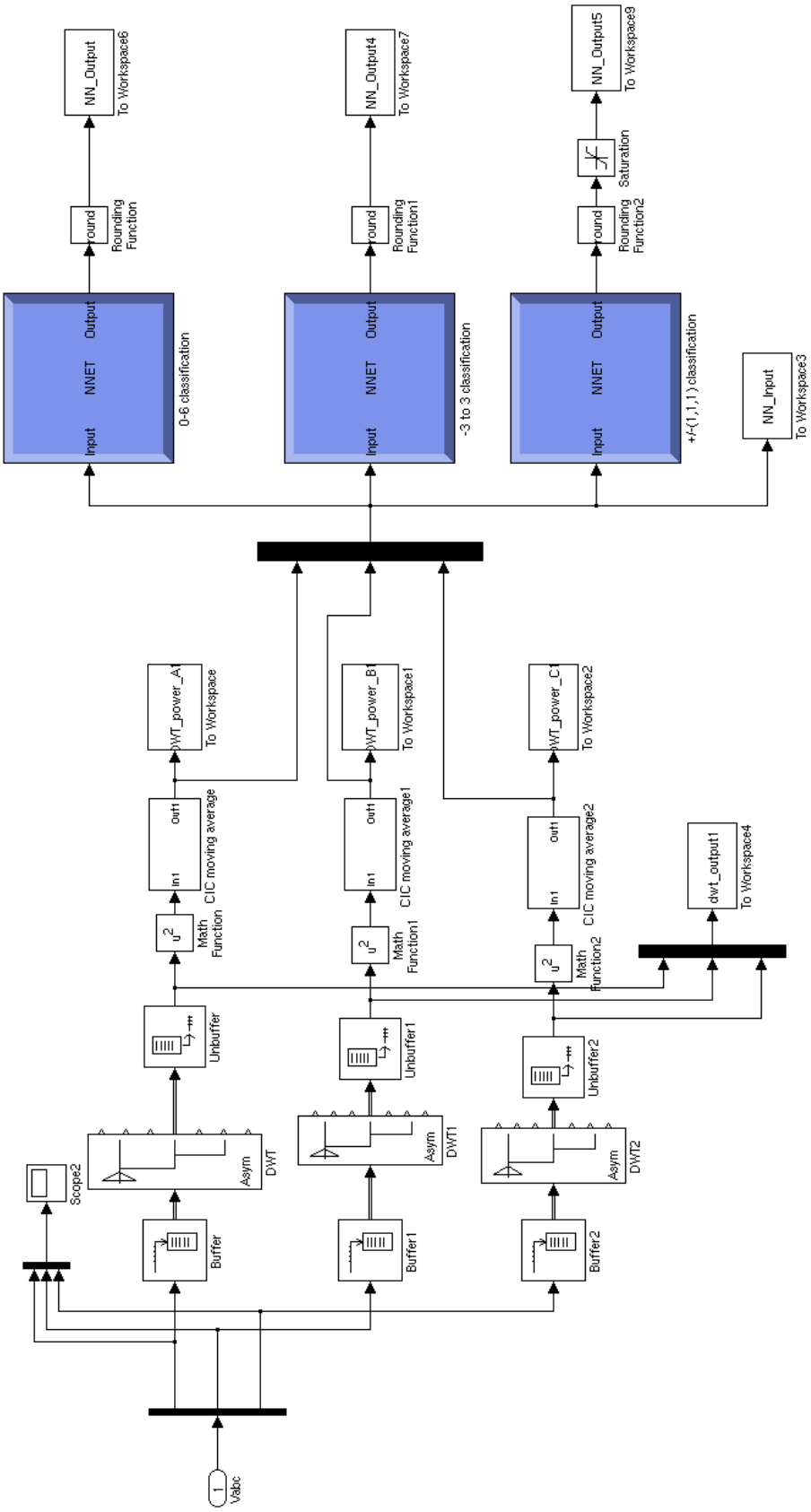

Figure 8: Neural Network Processing Subsystem

The outputs of the three CIC filters are multiplexed into a single vector as the input to the neural network. Three different Neural Networks are shown, each using the same input but providing different classification techniques. The first neural network labeled '0-6 classification' classifies a no-fault condition as a '0' and then 1-6 are the classification for A fault, B fault, C fault, AB fault, BC fault, and AC fault, respectively. An example is shown in figure 9a. The second neural network labeled '-3 to 3 classification' still classifies a no-fault condition as '0'. 1 to 3 classifies A faults, B faults, and C faults respectively, while -1 classifies AB faults, -2 classifies BC faults, and -3 classifies AC faults. An example is shown in figure 9b. The third neural network labeled '+/-(1,1,1) classification' classifies a fault condition for each of the three phases. A 1 signifies a fault condition while a -1 signifies no fault condition. An example is shown in figure 9c. For clarity, the three phases have been offset by +3, 0, -3. All three neural network outputs are rounded to the nearest integer, and the last one is also saturated to +/- 1. The last one appears to achieve the best performance so further analysis will only be done on that neural network. All three neural networks were trained on the same set of input data and target data.

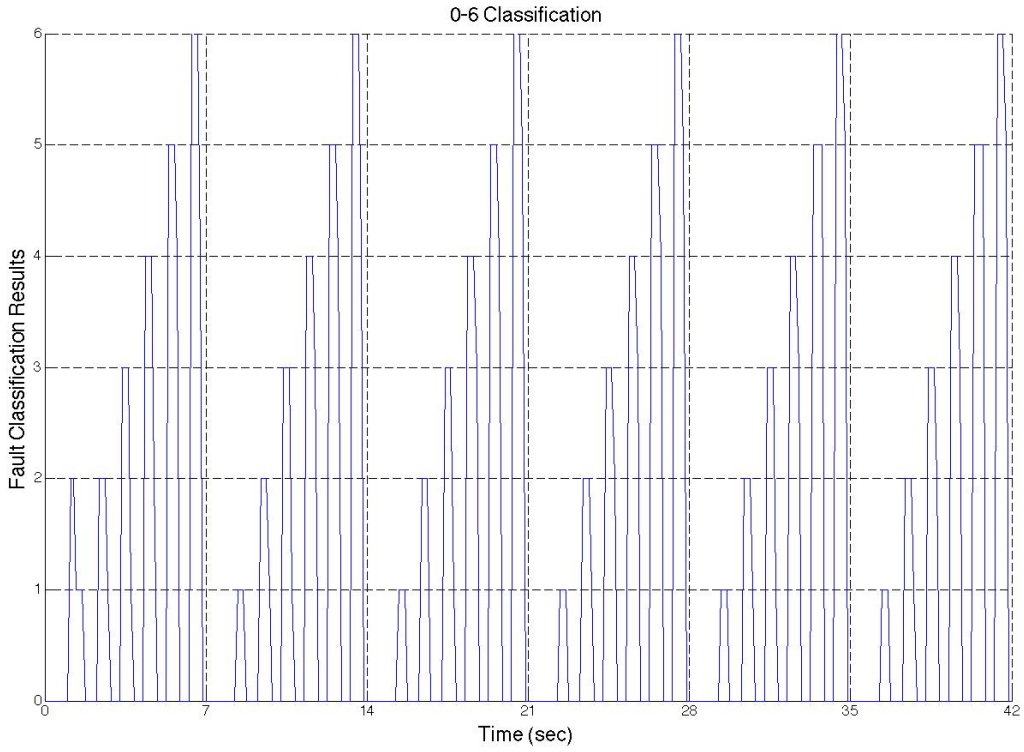

Figure 9a: Neural Network #1: 0-6 Fault Classification

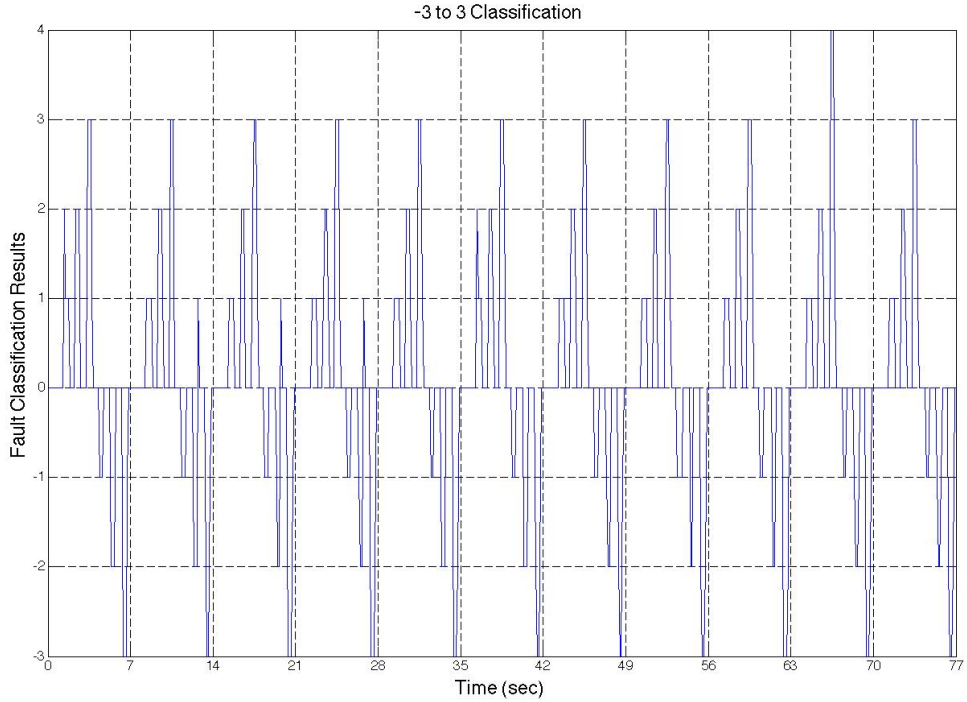

Figure 9b: Neural Network #2: (-3 to 3) Fault Classification

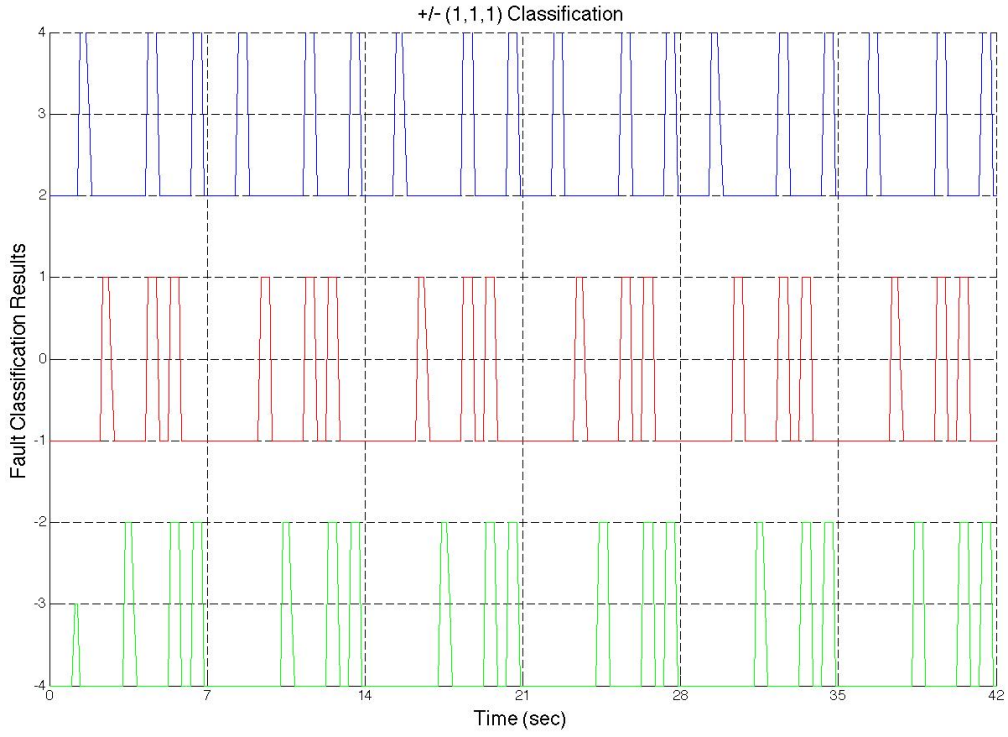

Figure 9c: Neural Network #3: +/- (1,1,1) Fault Classification. (Phase ABC = 'BRG')

There are three CIC filters shown in figure 8. A CIC filter operates on the power estimate from the DWT from each phase. The details of the CIC are shown in figure 10. The comb portion of the CIC filter (left half) indicates that the moving average filter is 256 samples in length. The integrator follows the comb portion of the CIC filter. The classic down-sampled CIC filter has a configuration in which the integrator is followed by a down sampler and then a comb filter. The input is a square function, and any integrator that follows this function will eventually saturate, thus the integrator must be placed after the comb filter. The input rate is 1024 samples/sec, as mentioned before, and the down sampler causes the output rate to be 16 samples/sec (1024/64). This is the input rate for the neural network, so the neural network operates on 16 samples/second. By referencing figures 9a-c, the time of the fault occurrence can be inferred to 1/16 of a second, since the DWT preserves time information.

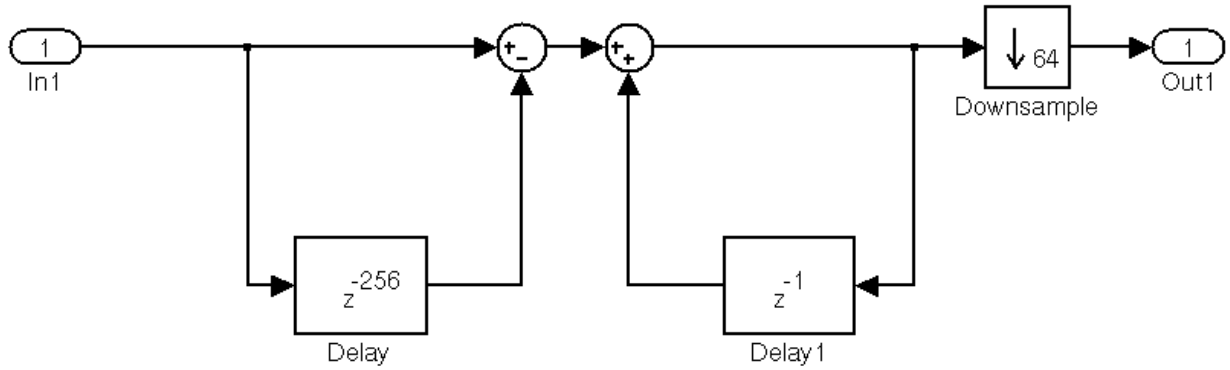

Figure 10: Cascaded Integrator-Comb Filter (1 each phase)

SimPowerSystems and Simulink were used to create the 3-phase power distribution system previously described. This simulation was used to intentionally create A faults, B faults, C faults, AB faults, BC faults, AC faults, and no faults. The generated signals were 50 Hertz sampled at 16384 Hertz for 1 second. The 1-second covers the duration of the fault. A fault occurred at 1-second intervals, cycling through the types of faults. Thus, each type of fault occurred at 7-second intervals for 140 seconds. This creates 20 faults (this number is increased later for further training trials) of each type of fault, or 140 faults, as our sample size. It is important for later to note that according to Nyquist's Sampling theorem, with a 16.384 kHz sampling rate, the highest frequency signal that can be constructed without loss of information is 8192 Hz.

Wavelet Transform Block

The output of the power distribution system is 7 structures used as inputs to the wavelet network. Each file characterizes each phase of the power system. Take the first structure (for A-faults) as an example. This represents the A, B, and C phases of the system *during* an A fault. Thus, it is in a matrix of 3 columns: one for the A phase, B phase, and C phase respectively. In order to analyze single faults, an arbitrary fault was analyzed before and after the wavelet network, as well as comparing each type of fault in each phase. This was done for comprehension and demonstrational purposes.

The next step is analyzing a particular fault. An arbitrary fault of each type was chosen and graphed for observation. This resulted in seven graphs. Each graph depicts the A phase, B phase, and C phase of a given fault condition. Through all graphs, phase A is illustrated in blue, phase B is illustrated in red, and phase C is illustrated in green. The graphs clearly illustrate what happens when a fault occurs. Looking at figure 11 below, the fault clearly occurs in the A phase, when the voltage shorts (grounds) during samples 750-1750. Phases B and C are only slightly altered because the fault occurring in the A phase was at about 0 volts, thus the distortion in the other two phases is not significant. When the fault is removed, there is a large step in voltage and thus a large spike in current. This causes a transient condition that causes distortion in the A, B, and C phases before returning to the no-fault condition. Similar events are evident in figures 12 and 13, representing B and C faults. Looking at the AC fault (figure 16), both the A and C phases exhibit the same behavior characteristic of a fault. Finally, looking at figure 17, this is the no fault condition. It is quite evident why this is the case. The figure depicts a balanced three-phase system with each phase 120^0 out of phase of each other, with no faults at either stage, showing three unharmed sinusoids.

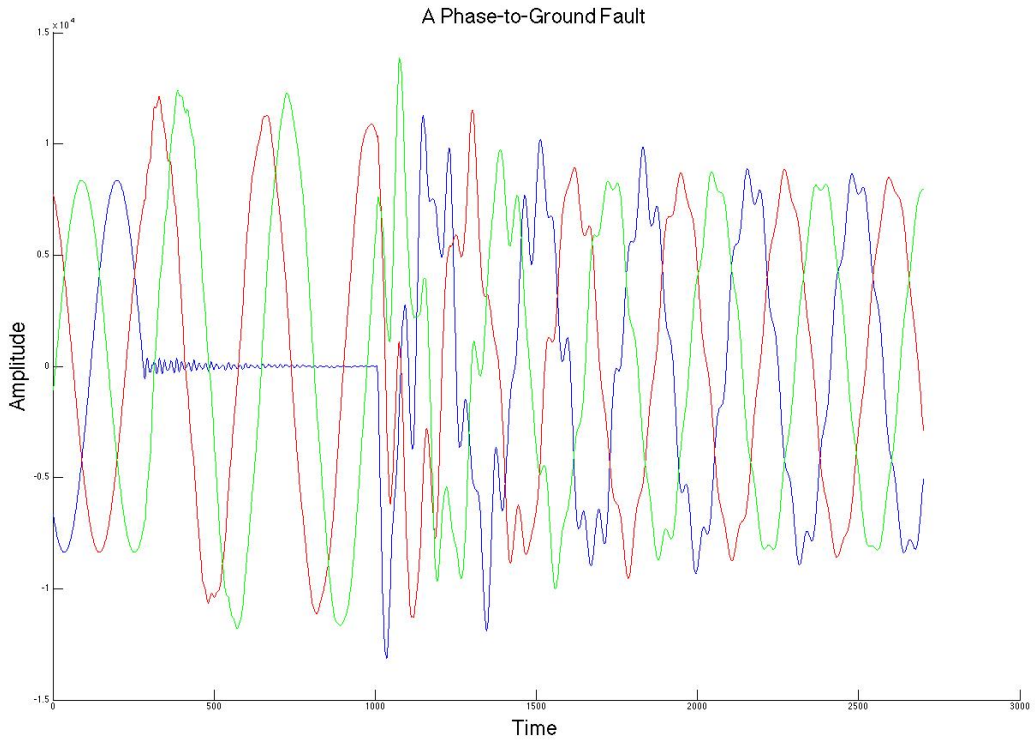

Figure 11: A single Phase A Fault, Phases ABC = “BRG”

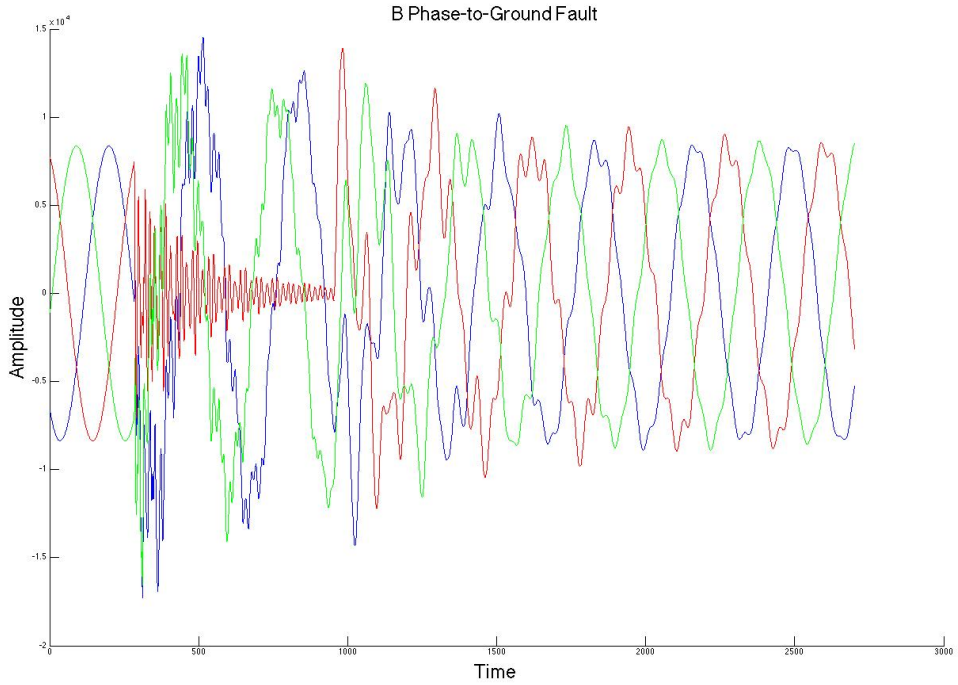

Figure 12: A single Phase B Fault, Phases ABC = “BRG”

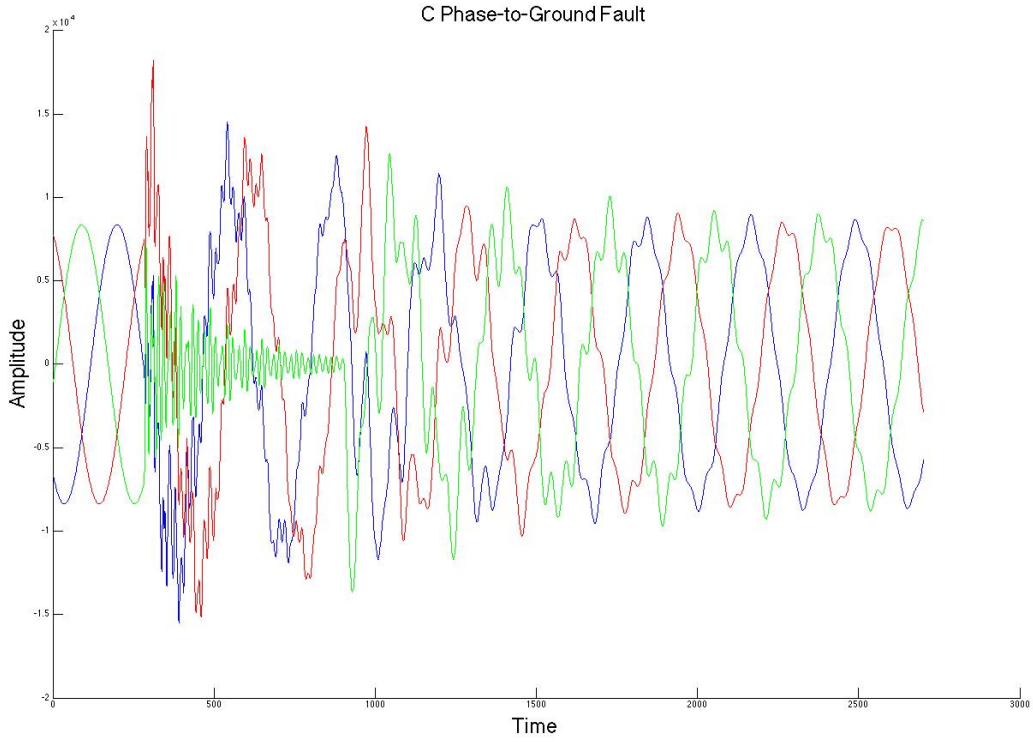

Figure 13: A single Phase C Fault, Phases ABC = “BRG”

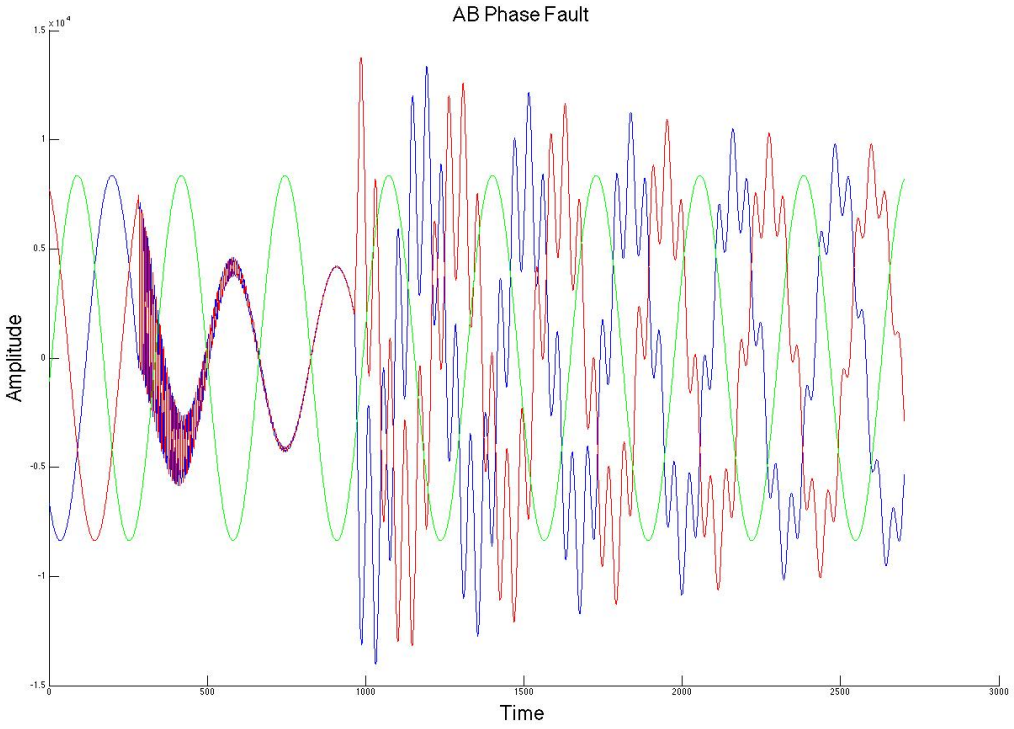

Figure 14: A single Phase AB Fault, Phases ABC = “BRG”

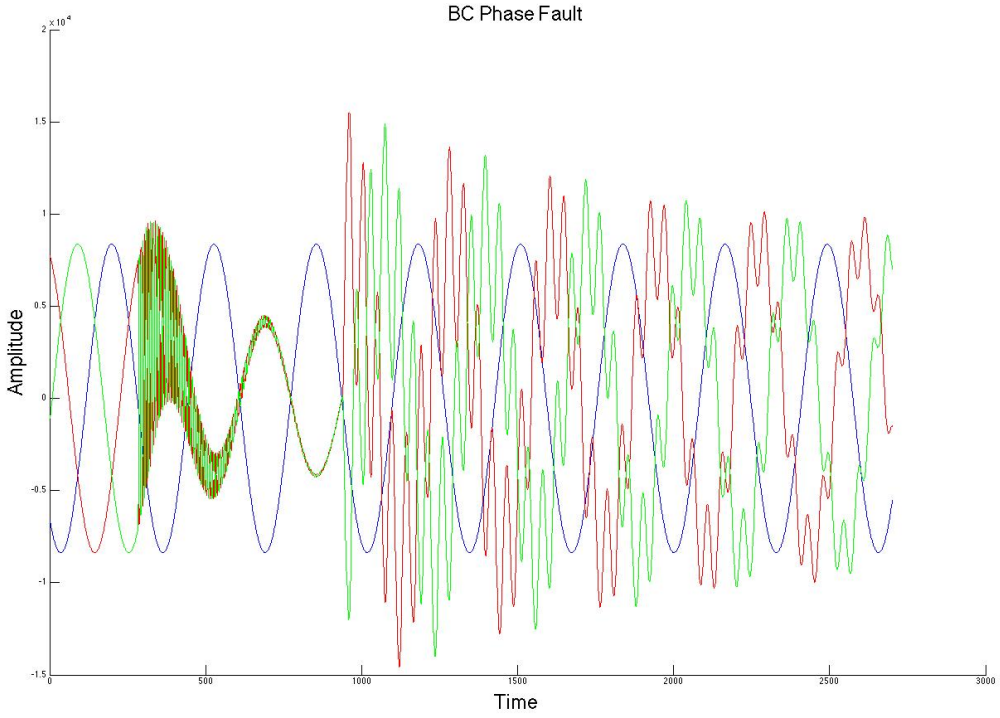

Figure 15: A single Phase BC Fault, Phases ABC = “BRG”

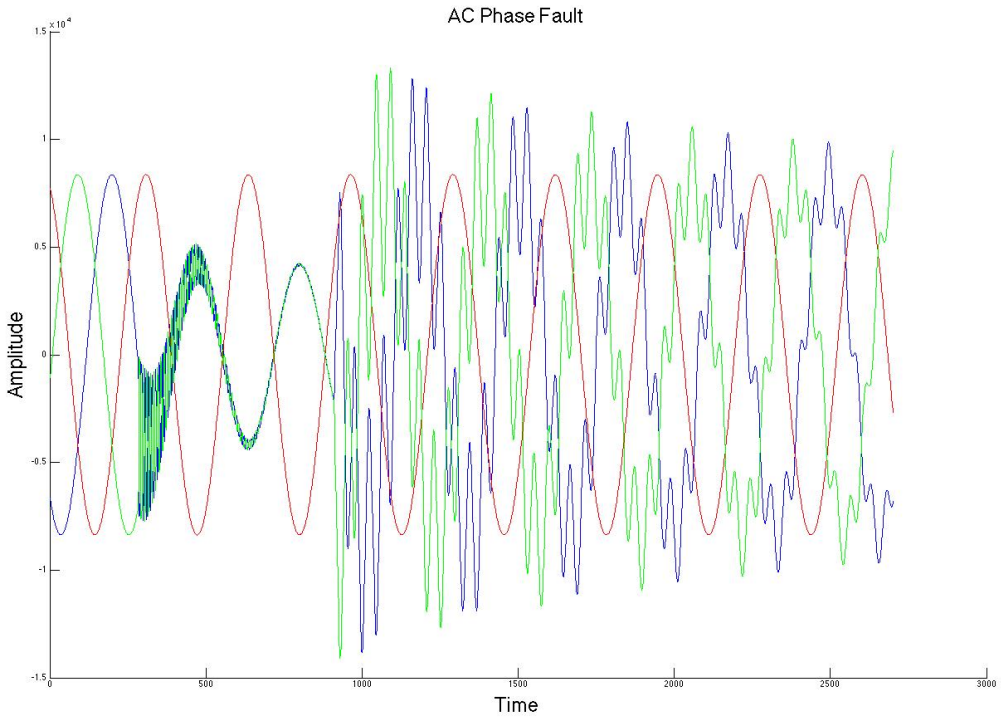

Figure 16: A single Phase AC Fault, Phases ABC = “BRG”

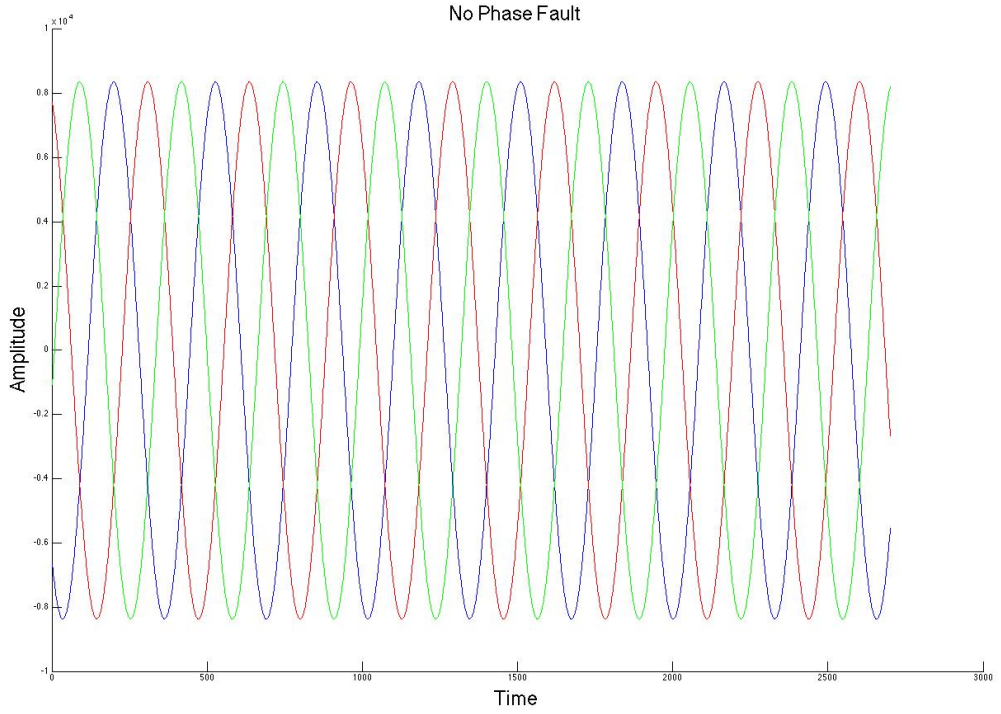

Figure 17: A single No Fault, Phases ABC = “BRG”

The discrete wavelet transform is an extremely powerful decomposition tool that provides a trade-off between time accuracy and frequency resolution. As mentioned before, this transform allows for knowledge of both time and frequency simultaneously. The well-known Fourier Transform provides only frequency information, with no time discrimination at all. At the top level of the decomposition, the time accuracy is at a maximum, thus the frequency resolution is at a minimum. As more wavelet decomposition levels are performed, the elements span the same time frame with less time accuracy. Figure 18 shows an intuitive illustration of the first levels of wavelet decomposition.

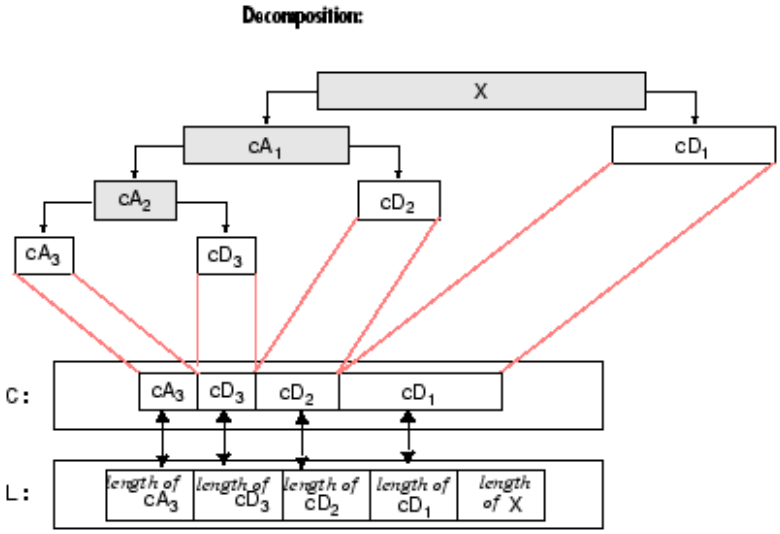

Figure 18: Wavelet Decomposition: *MATLAB Wavedec Help

Every time decomposition is performed, there are less data points represented for the same duration of time, thus less time accuracy. This is because the decomposition effectively performs decimation on the time scale by a factor of 2. However, the frequency band of a particular element gets smaller so greater frequency discrimination has been obtained. Each level further down spans half as much frequency, but also half the time. This is in effect pinpoints the frequency because if it occurs in a certain bin (a bin is referred to as a box on the diagram), the span of frequency is known. The more decompositions, the smaller the frequency span of a particular bin, thus the more the frequency is known if the fault occurs in that frequency band. To reiterate, each decomposition decreases the frequency span of a bin so the frequency is known more accurately, but has less data points so there is an increase in time ambiguity. However, the time of occurrence is still known because of where in the span it occurs.

For further clarification on the frequency span, let f_S be the sampling frequency. By Nyquist's Sampling theorem, the frequency span of the bin labeled "X" in the diagram above is $f_S/2$. The decomposition is actually done by applying a low pass filter and a high pass filter at the center frequency of the span to obtain A1 and D1, respectively. As a result, the frequency span of the bins labeled A1 and D1 are 0- $f_S/4$ and $f_S/4$ - $f_S/2$. D1 is now kept as information and A1 goes through the same wavelet

decomposition into A2 and D2. A2 spans frequencies of 0 - $f_S/8$, and D2 spans frequencies $f_S/8$ - $f_S/4$. [2] Provides a similar version of this explanation.

Throughout the project, it was determined that the best wavelet decomposition level was 4. It provided the most power discrimination between the A, B, and C phases and corresponded the most with the actual fault condition. For further detail about how the best mother wavelet and decomposition level was chosen, see 'Comparison Between Detail Levels and 2 Mother Wavelets.' Because the decomposition level 4 was used, the frequency spans of interest were 0 - $f_S/32$ for A4 and $f_S/32$ - $f_S/16$. During the literature review, it was found that the wavelet that provided the best test results was DB4. This refers to the Debauchee level 4 wavelet. Debauchee was a brilliant mathematician that determined the most widely used coefficients that elegantly satisfied the numerous mathematically intensive requirements of a wavelet transform. These coefficients represent the filter coefficients needed to decompose a signal into wavelets. Two mother wavelets were tested in this project: DB2, and DB4. DB4 worked well as the mother wavelet for feature extraction.

The next block in the design is the wavelet network. The wavedec Matlab function mentioned above in figure 18 was the function used to perform the discrete wavelet transform (DWT). For analysis purposes, the same faults that were analyzed before the wavelet network will be analyzed after. After the 4^{th} level wavelet decomposition was completed on each of the 140 faults, the same seven faults were grabbed and plotted. Each phase of a given fault was plotted on the same graph for comparison and realizing the classification of a fault condition. For each of the seven faults under analysis, A4 and D4 were plotted, as well as D4 alone. This is to analyze the detail information and compare the scales between the two graphs. These figures are shown below in figures 17-32.

Analysis of figures 19 and 20 show A4 and D4 levels of a phase A fault. As depicted in the diagram in figure 18, the first half of the graph is A4, and the second half is D4. Thus, figure 20 is the right half of the graph in figure 19. The reason the two figures look so different is because of the scale. The scale of figure 19 is 10 times larger than that of only D4. D4 has picked out the *detail* of the fault. This is one of the 140 elements that will be outputted by the wavelet block of the design. These 140 elements are the D4 decompositions of each of the 140 faults. It is understood that the occurrence of a fault causes a transient behavior, which directly causes high frequency components in the signal. Thus, the high frequency components of the band must contain the information of the fault. This information will show up eventually in a detail wavelet at some decomposition level. Here is an important note: A4 stands for "approximation" level 4. This is because the A4 section of this figure looks like figure 11. The difference is the time axis is scaled down by a factor of 2^4 (2 for each decomposition). Now note figure 20. The figure is dominated by blue spikes. This is characteristic of a phase A fault. The power calculation for the A phase in a phase A fault will be much higher than that in phases B and C. This is ultimately how the desired output is determined and how the neural network is trained. Furthermore, the power calculation in the B phase of a B fault will be much higher than the other two phases, and the power calculation in the C phase of a C fault will be the largest. The 16 figures below support this argument; the phases where the fault occurs dominate the 4^{th} level detail wavelets.

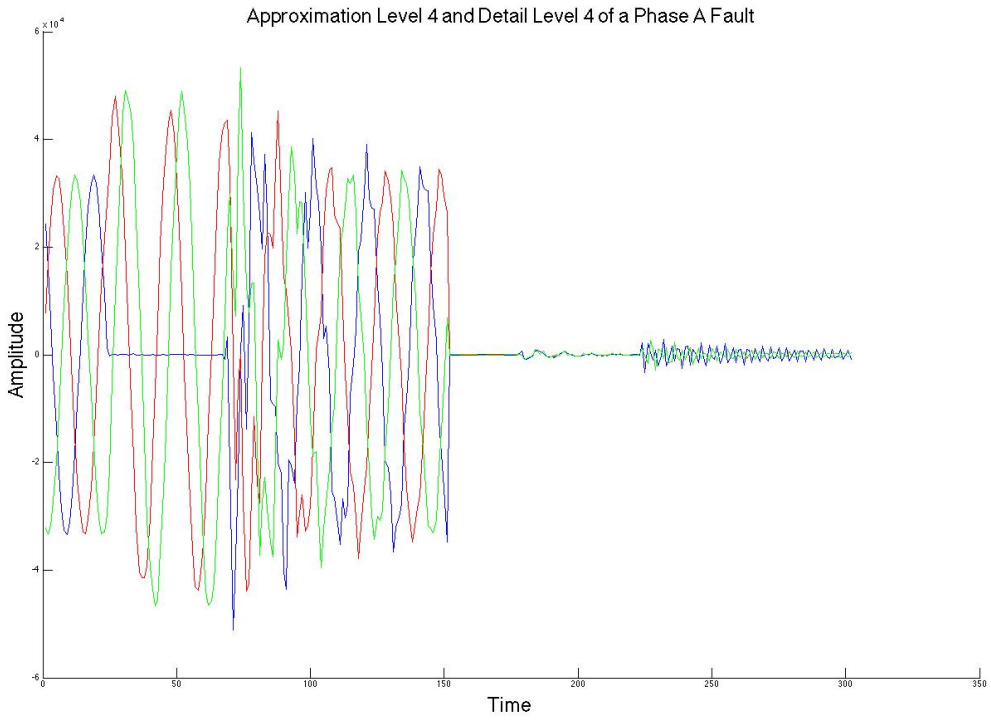

Figure 19: A Fault A4 and D4 Wavelets

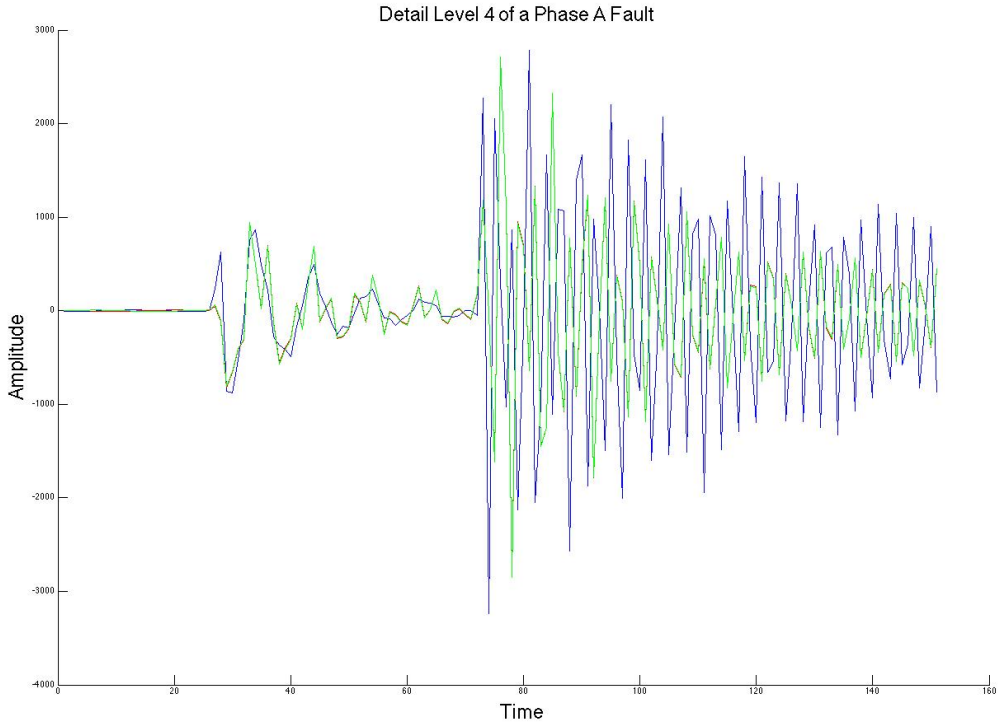

Figure 20: A Fault D4 Wavelet

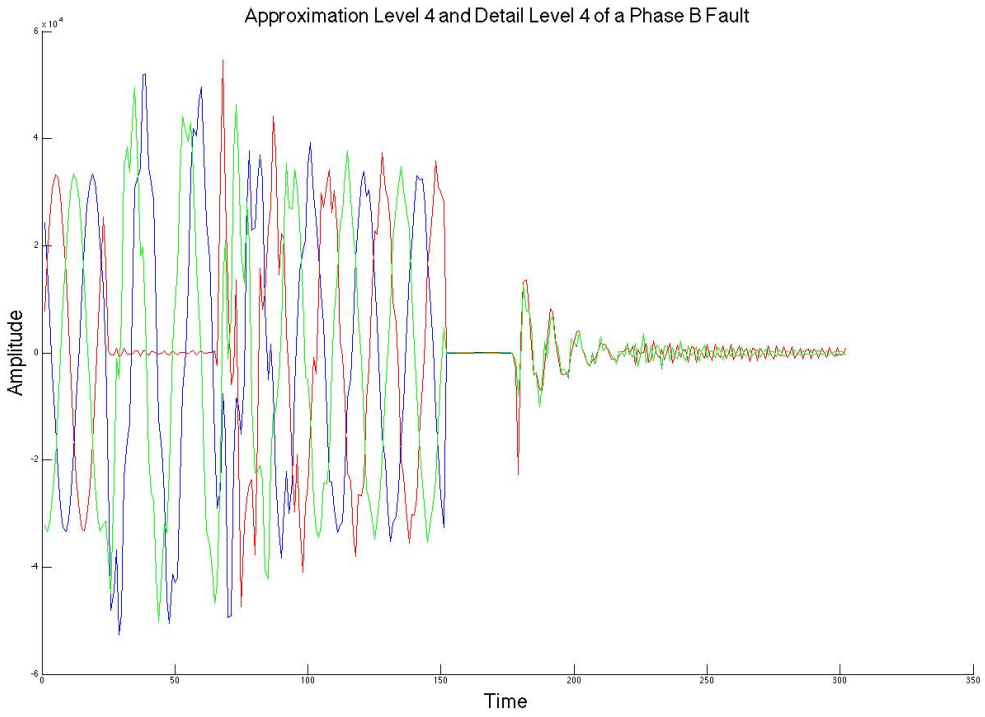

Figure 21: B Fault A4 and D4 Wavelets

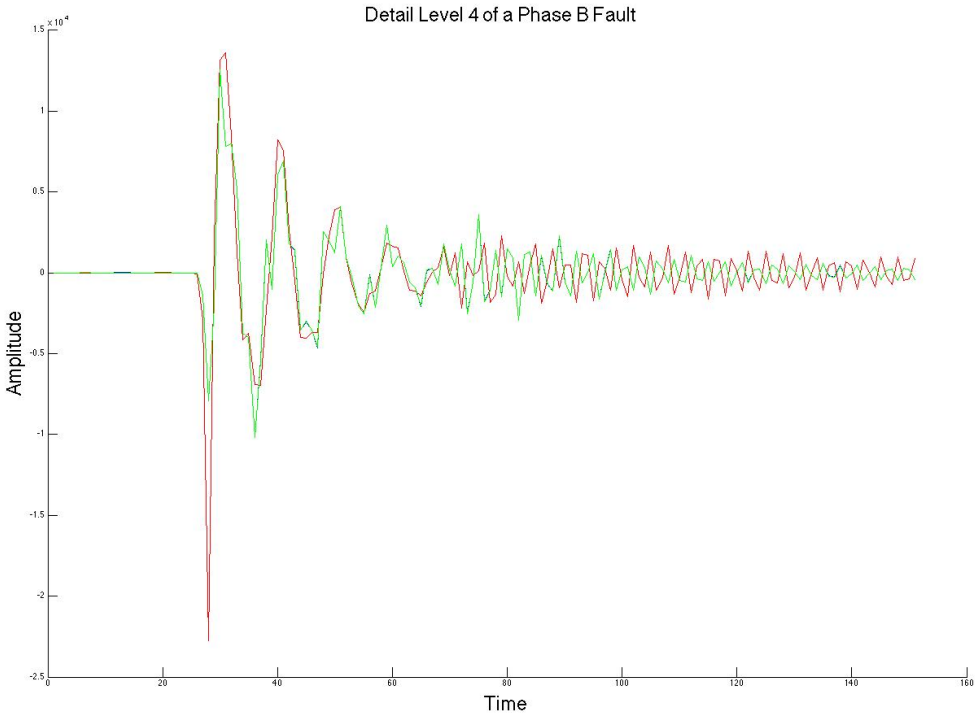

Figure 22: B Fault D4 Wavelet

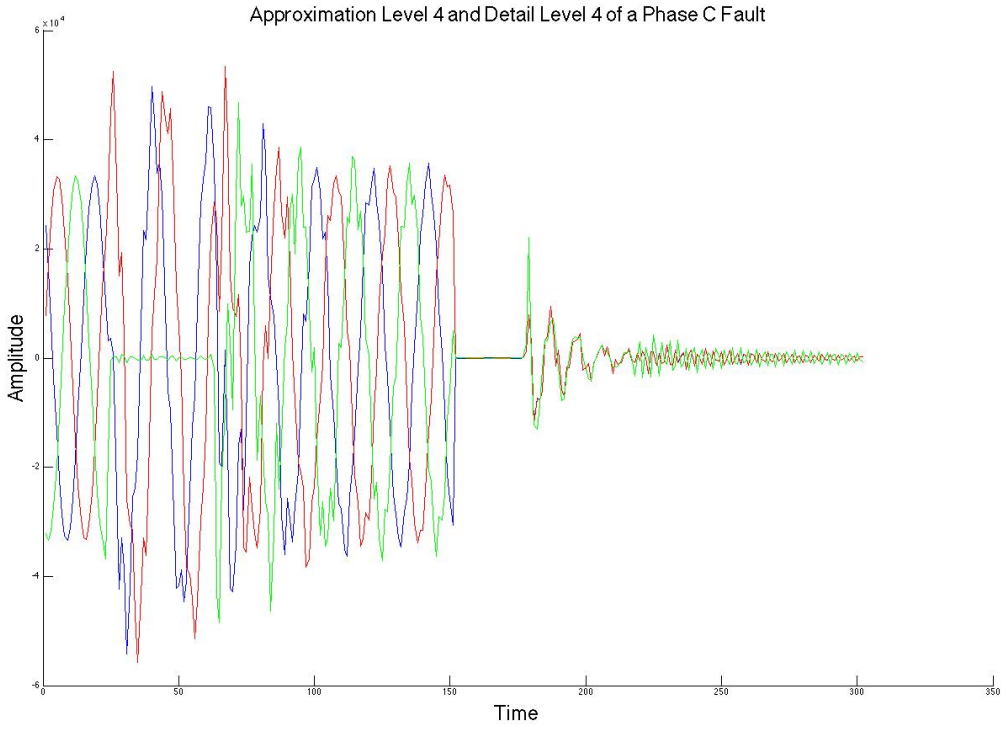

Figure 23: C Fault A4 and D4 Wavelets

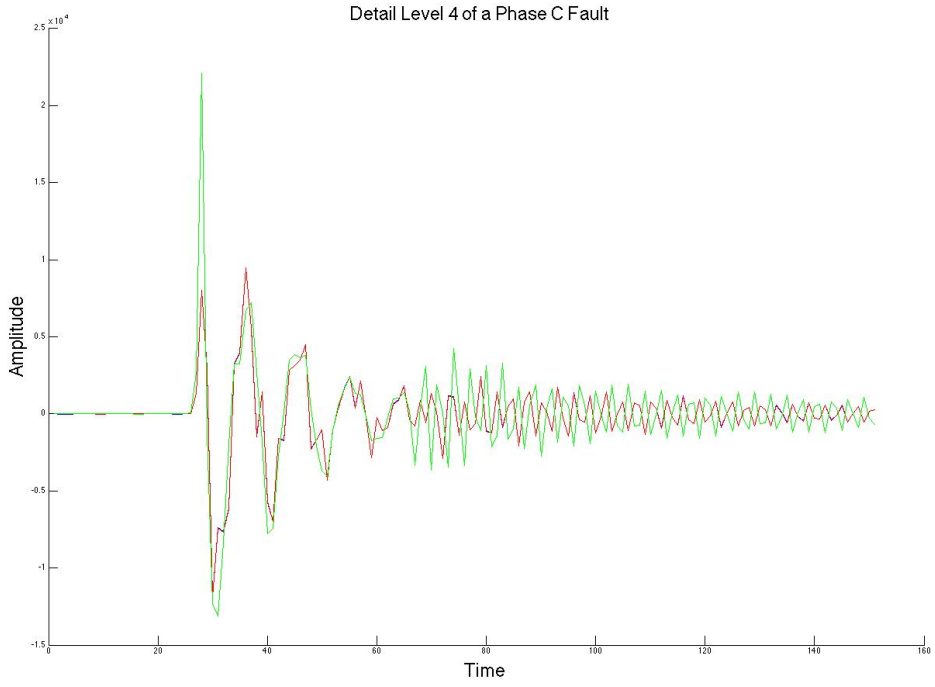

Figure 24: C Fault D4 Wavelets

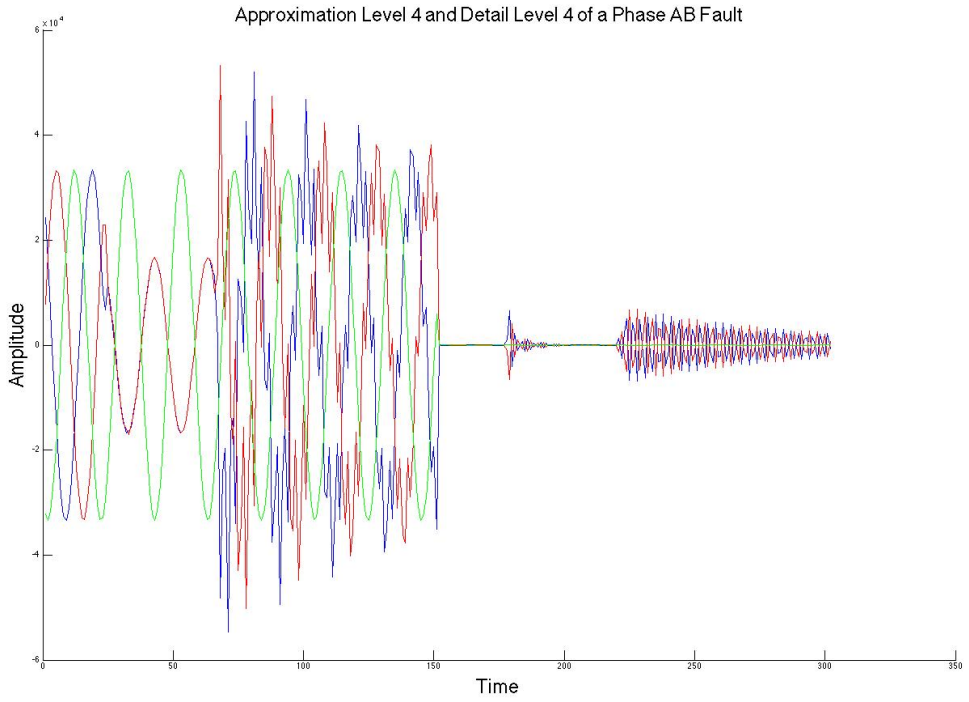

Figure 25: AB Fault A4 and D4 Wavelets

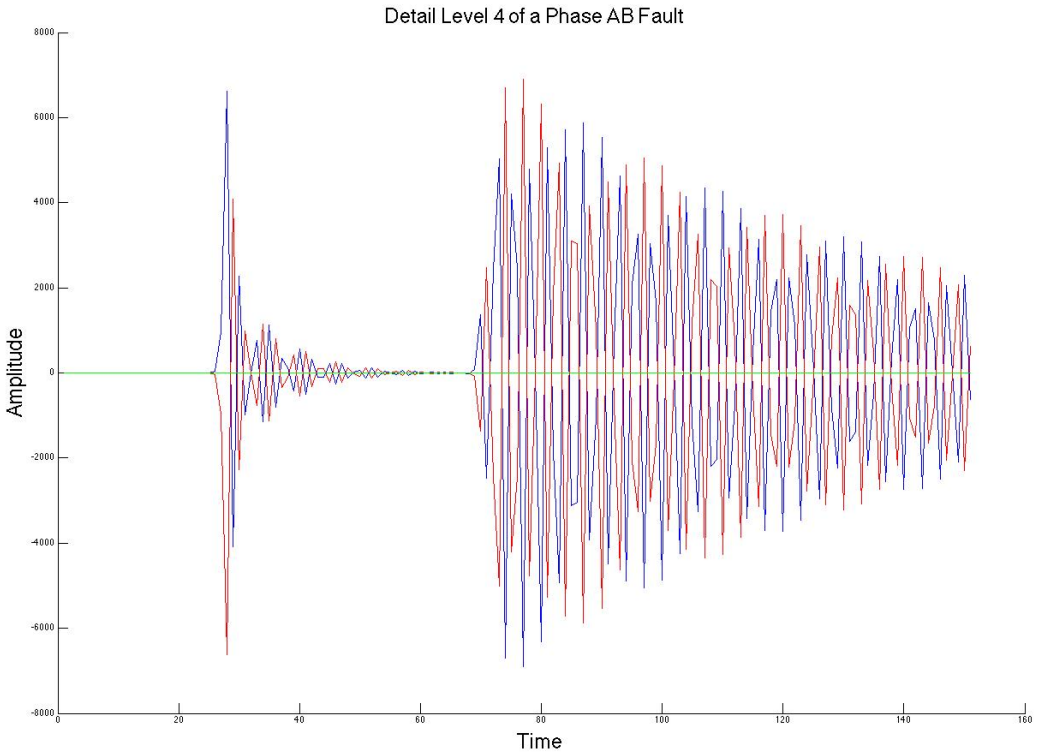

Figure 26: AB Fault D4 Wavelet

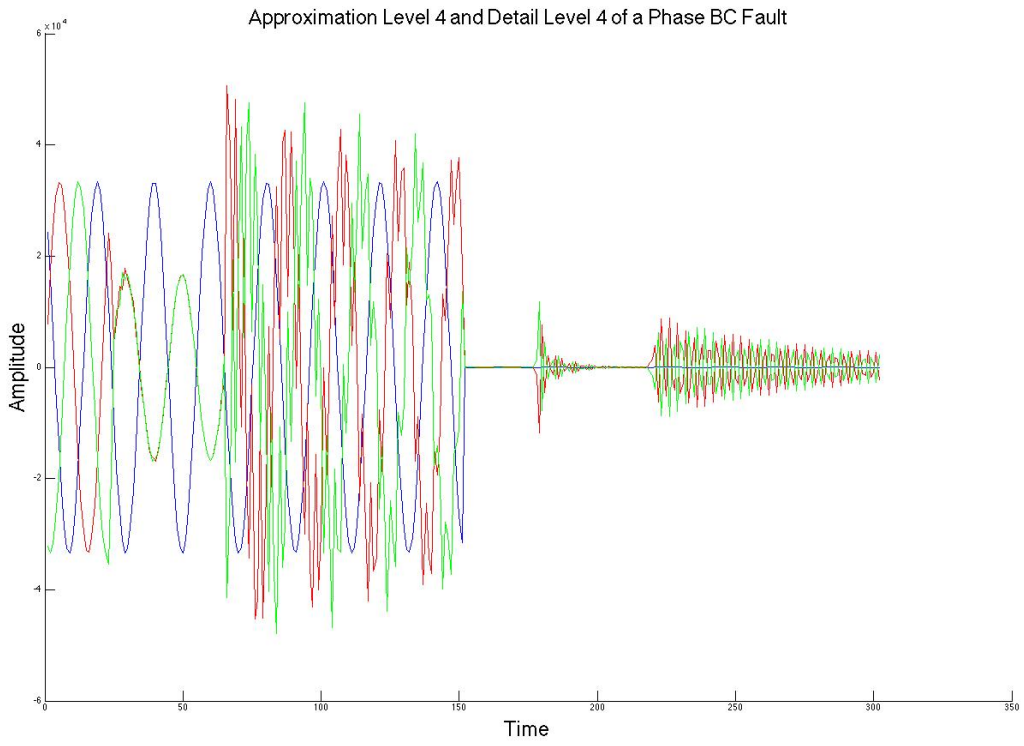

Figure 27: BC Fault A4 & D4 Wavelets

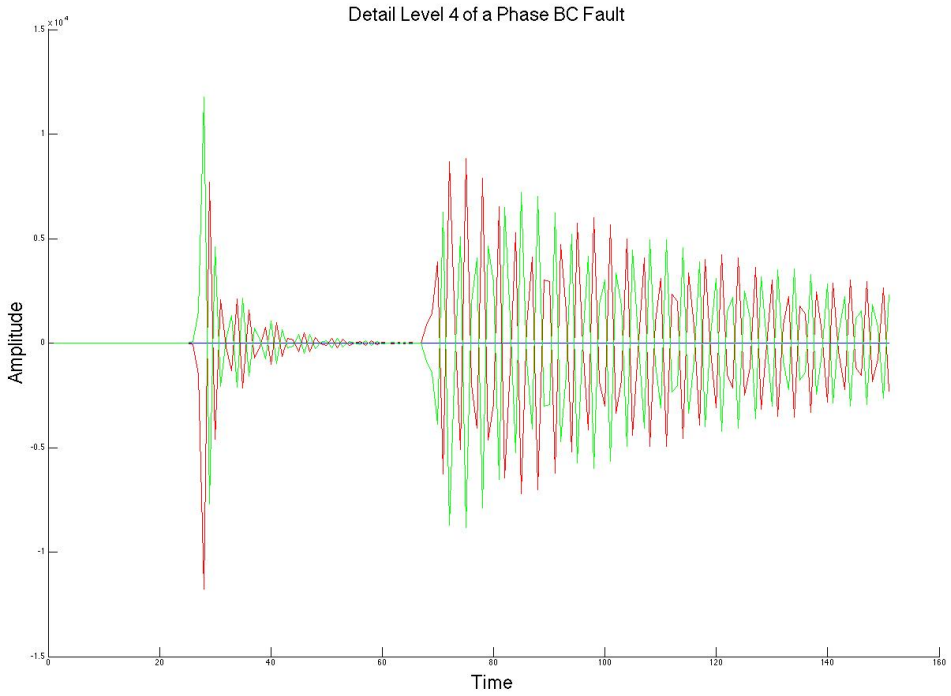

Figure 28: BC Fault D4 Wavelet

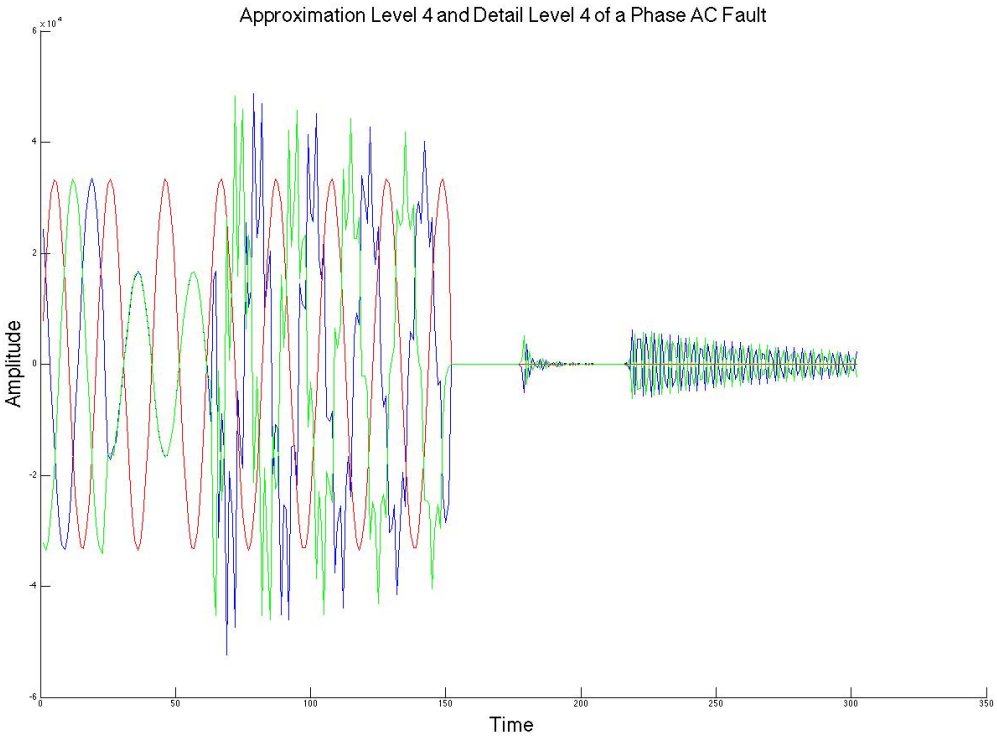

Figure 29: AC Fault A4 & D4 Wavelets

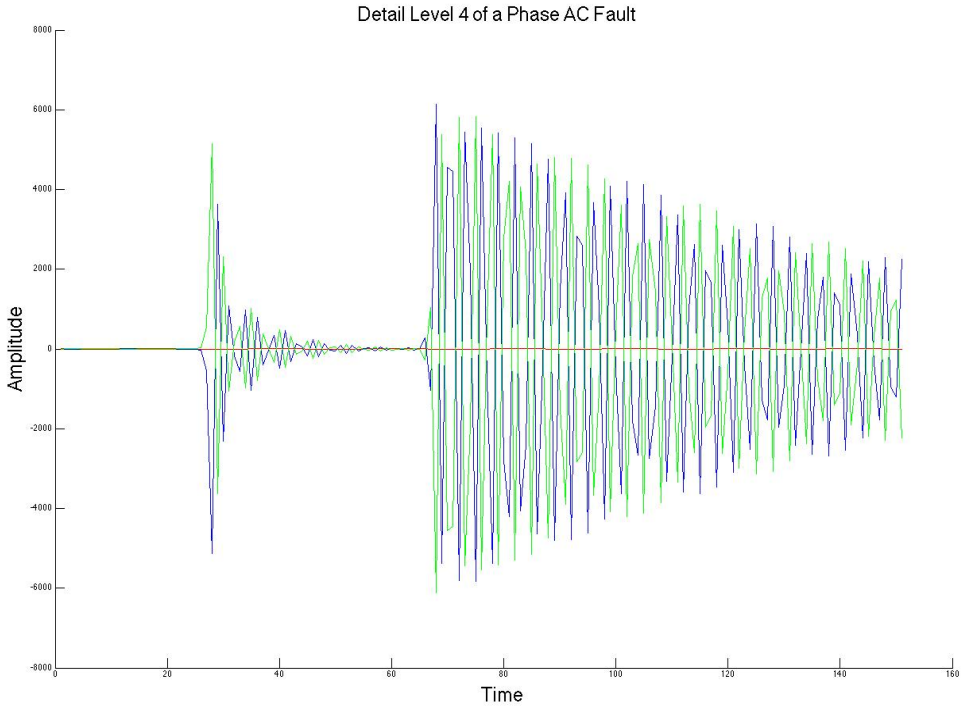

Figure 30: AC Fault D4 Wavelet

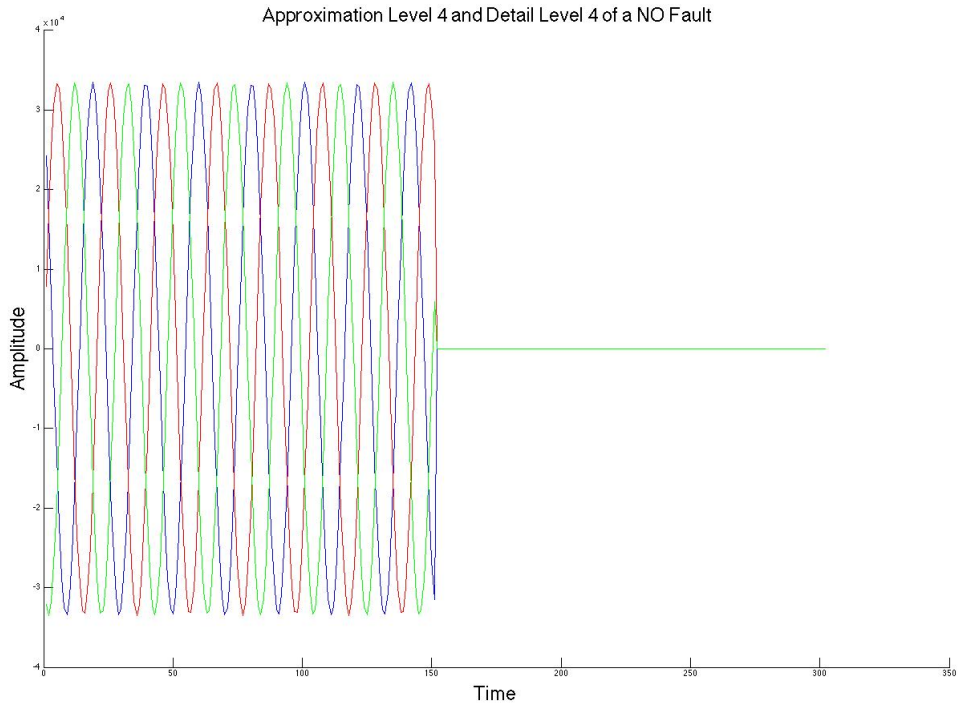

Figure 31: No Fault A4 and D4 Wavelets

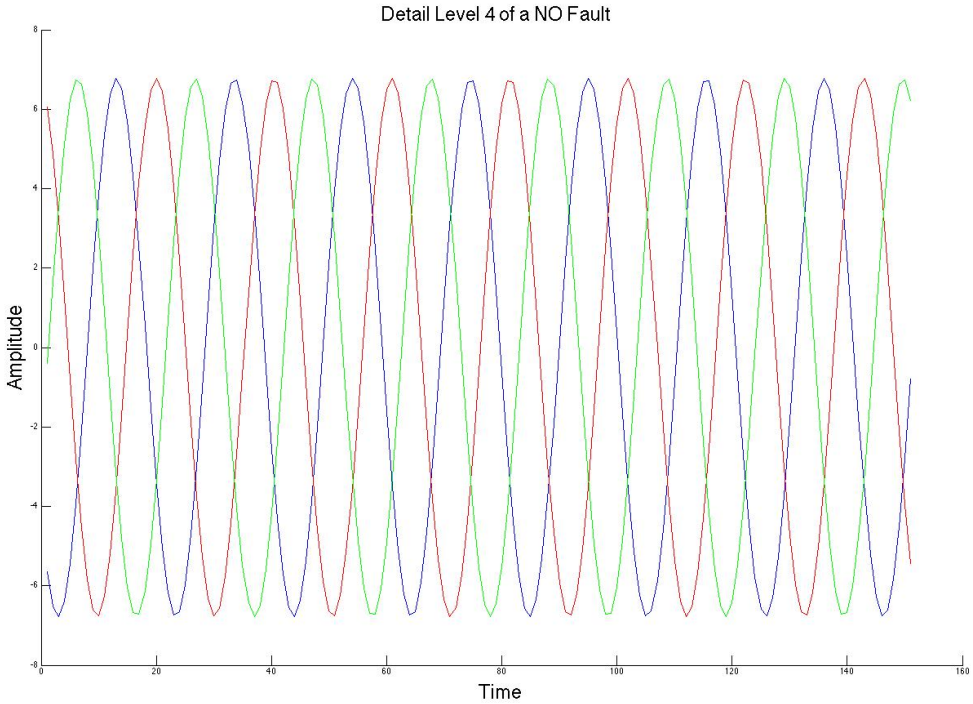

Figure 32: No Fault D4 Wavelets

Power Calculations

The next block of the design is the power calculations. This is an intuitive block that calculates the power of the 140 input elements and outputs the 140 results. Because each input contains the wavelet decomposition of the A, B, and C phases, the input of this block is three vectors representing each phase of a fault. The entire input set is 140 sets of these three input vectors. The output of this block is a 140x3 matrix containing the power calculated in each phase, of each fault. Note that *a particular element of the output matrix is the power calculation of one vector input*. The calculations are done by squaring each element of a vector input and summing the results. As mentioned before, wherever a fault occurs, that phase will have a larger power calculation than the other phases. This was also the basis by which the different decomposition levels were evaluated on which level best suits the feature extraction for this project, as the power calculations are what determine the output of the neural network. In the instance of a "no" fault, all three of the power calculations will be small.

For emphasis, there is a significant difference in power between when a fault occurs, and when a fault does not occur. Table 3 illustrates this difference. It makes sense that the power calculation for a no fault condition would be so low in comparison to the power calculations for when a fault occurs. The calculations are being performed on the level 4 decomposition, because that is what contains the best information of the fault. Since no fault is occurring, there is no high frequency content for the wavelet decomposition to extract. The power is an integration over an interval of the detail level 4 DWT, synchronized to the fault condition, not an instant.

Fault Type	Phase A Power	Phase B Power	Phase C Power
AC	2.319001957832834e+09	1.049866242851910e+09	1.960353712468527e+09
A	2.263222746737017e+07	1.319521306549184e+07	1.312957736662839e+07
B	1.748454672943847e+09	2.859689416692212e+09	1.748329683589568e+09
C	1.632908627127061e+09	1.633098361043096e+09	2.722518342687527e+09
None	83.932510953162490	79.139887463154850	79.591765545812660

Table 3: Sample Power Calculations of Detail Level 4 in one of each Fault

Outputting all of the power calculations into a text file creates the input file for the neural network used for training. The data size is of course 140 samples, and 70% of the samples were used for training, 15% for validation, and the last 15% for testing of the neural network. The output file was created by simultaneously writing a known fault condition to another file as each power calculation was written to the input file. For example, during an A fault output, the power calculations were written to the neural network input file, and the known fault conditions of [1,-1,-1] were written to the neural network target output file. A 1 represents a fault is present, while a -1 represents that no fault has occurred. The three elements of the vector represent the A, B, and C phases of the event respectively.

Neural Network Block

The last element of the block design is the neural network. The neural network fitting tool was used to implement the network. This is a feed-forward neural network using a common back propagation training algorithm known as Levenberg-Marquardt back propagation. There were

advantages and disadvantages to this feature in the Matlab neural network toolbox. A script did not have to be written; it could just be generated upon a successful training and validation process. Also, the tool can automatically read in input and target files for the neural network to obtain a solution. However, the fitting tool did not allow more than one hidden layer. In order to implement more than one hidden layer, a script had to be generated using the neural network tool, and the hidden layer configuration had to be modified accordingly in the script. As a result, the project simulations included 3 trials: one of 10, 15, and 20 hidden neurons in the hidden layer. Figures 33-35 highlight the results of the trial implementing 10 hidden neurons.

Figure 33 plots the error, which is the difference between the outputs of the neural network and the pre-determined target data set. The more data elements close to zero the better, because this means the neural network converged to an accurate solution, and the neural network output is very close to the target output. Figure 34 analyzes the MSE (mean square error) as the neural network is being trained. The neural network stops training when the stop criterion is met and when the MSE for the validation data stops decreasing. An epoch is an iteration of the neural network going through the training, validation, and testing samples. Figure 34 shows that the mean square error continually decreases at a quick rate as the neural network learns. The most important line on this graph is the red line; this is how the network performed while undergoing testing, which is an indication of how well the network generalizes to new data. The network performed even better under testing than it did under training. That shows a high success rate. However, every time the neural network is trained, the performance differs because the initial conditions of the network vary with every trial run. Trial and error is the only approach to date that can determine an optimal configuration of the hidden layers and the number of neurons in each layer in order to obtain the most efficient convergence to a solution.

Figure 35 plots the target data against the output of the neural network over all epochs. Ideally the equation of this line should be output = target (y=x), because a one to one correlation between the two is desired. Looking at figure 35, which contains the graphs of the target output versus testing, validation, training, and the entire input data set, all of the equations are displayed on the left of each graph. They are all extremely close to the desired relationship. The “R” value displayed on the chart is a measurement that determines the amount of correlation between two variables. If this value is zero, the two variables are said to be random and there is no way of predicting the value of one given another. If the value is 1, the value of the other variable can be perfectly predicted. As the R-value is very close to 1, this shows that the neural network can accurately predict the output given an input, with an extremely high confidence rate. Overall, the neural network outputs showed high success.

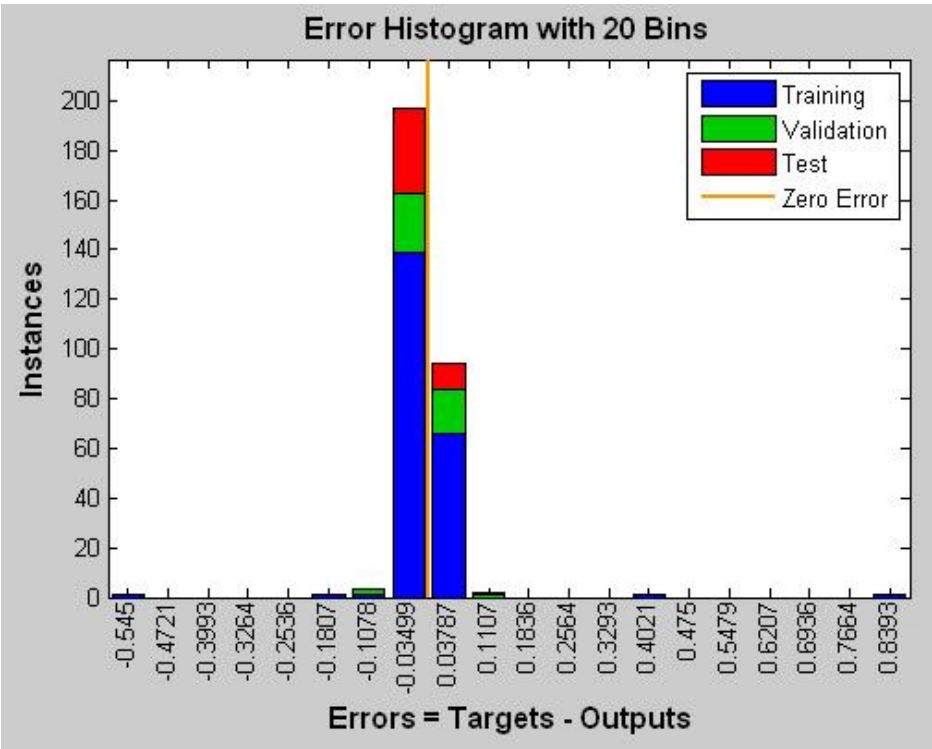

Figure 33: Error Histogram of Neural Network, 10 Hidden Neurons

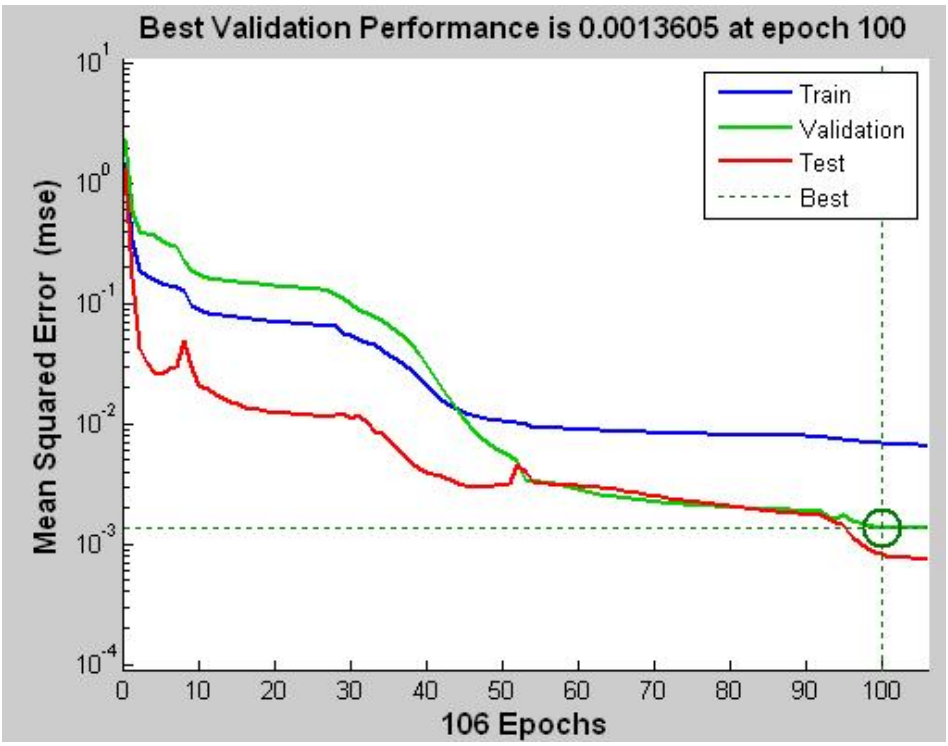

Figure 34: Validation Performance of Neural Network, 10 Hidden Neurons

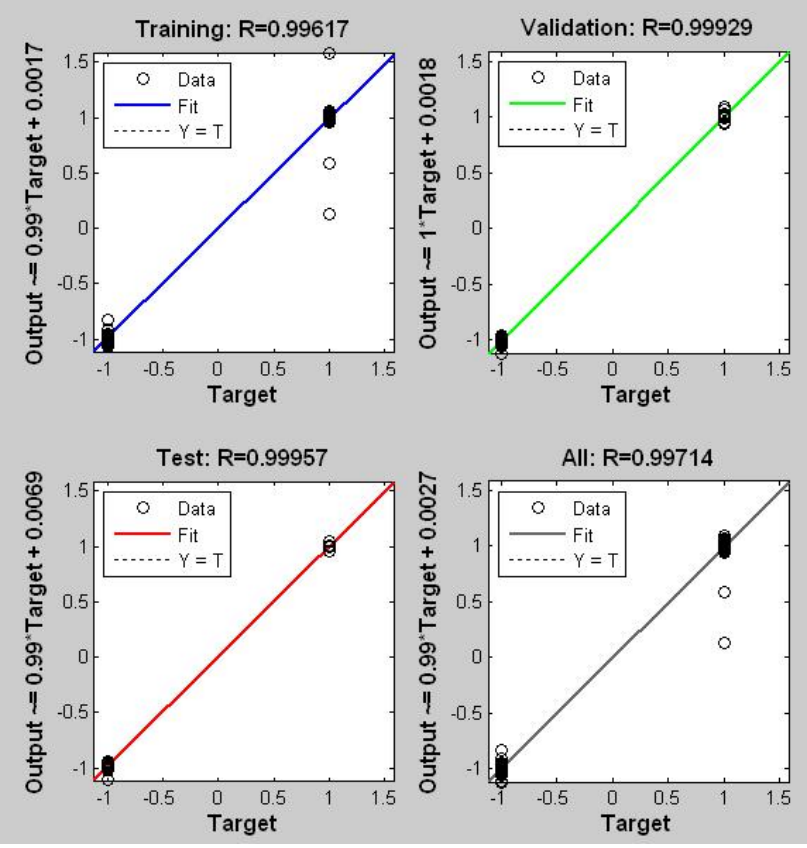

Figure 35: Regression Plot of Neural Network, 10 Hidden Neurons

IV. Results

Comparison Between Detail Levels and 2 Mother Wavelets

In the literature review, multiple sources mentioned that performing 4 levels of decomposition using the mother wavelet DB4 (4^{th} order Debauchee filter) provided adequate power discrimination for the feature extraction aspect of the project. This section attempts to analyze the different decomposition levels from detail level 2 to detail level 6, along with using 2 different, popular mother wavelets: DB2 and DB4. 6^{th} level decomposition was used because of the manner in which the wavedec function stores the results. It can be seen clearly from figure 18 that performing decomposition with the wavedec function also gives the previous decompositions. Hence, a 6^{th} level decomposition gives D1, D2, D3, D4, D5, and D6 (as well as A6). As a side note, the frequency of the source is seen at higher decomposition levels because the decomposition is accomplished with low pass and high pass filters. When the source frequency becomes too obvious, it contributes significant energy to the power calculations. This occurs as early as decomposition level 5.

The functions power_arrays.m and powercalculation.m (shown in Matlab scripts in the Appendix) were used to calculate the powers in each phase, of each fault simulated, for detail levels 2-6. Once this was done, the power calculations for each phase, of each fault, were plotted against the fault. Faults 21-49 were taken as observation points. Thus, 4 faults of each type are in each of figures 36-45. The fault types are repeating every 5 faults. So, faults 21, 28, 35, and 42 are A faults, thus a blue dot should show the largest power calculation for each A fault. Faults 22, 29, 36, and 43 are B faults, so a red dot should show the largest power calculation for each B fault. For the C faults (Faults 23, 30, 37, 44), the largest power calculation should be green for each C fault. For the AC fault, the largest should be both blue and green. For the AB fault, the largest should be both blue and red. For the BC fault, the largest should be both red and green. Based on discussion of figures 4-7, I anticipate the difference between these for the AC faults is directly correlated with the amplitude of the signal when the fault occurred: i.e. the phase with the larger power calculation had larger amplitude when the fault occurred. For the no fault condition, all 3 dots should be small. This is the basis used to analyze figures 36-45 and determine which mother wavelet and decomposition level is best for feature extraction of the faults.

By quick inspection of the figures, it can be determined that D2, D3, and D6 decomposition levels do not match up with the expected outcomes at all. The D5 decomposition level matches up with most of them. It is particularly good at distinguishing AC Faults. However, some single-phase faults come very close in power calculations between phases. The DWT and neural network was able to distinguish between the single-phase fault and a double phase fault. The best detail level, for both mother wavelets, is clearly D4. Now consider figures 38 and 43. These figures are the detail level 4 decompositions for DB2 and DB4, respectively. Both of these outcomes match the expected phase-to-ground fault conditions. Either one accomplishes feature extraction.

It is important to note that double phase-faults caused too much of a transient effect in the third phase, making it indistinguishable from a 3-phase fault. To compensate for this, the double phase faults were made to be phase-to-phase faults and not double phase-to-ground faults.

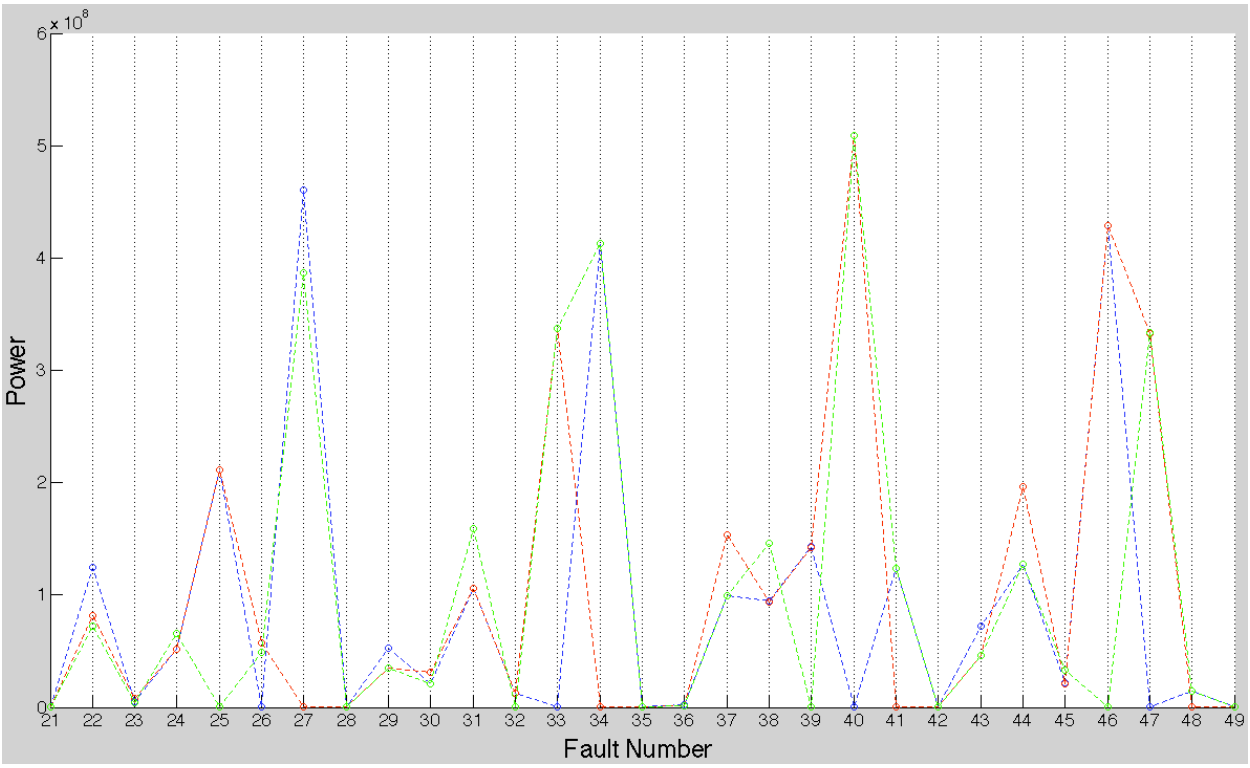

Figure 36: DB2 Detail Level 2 Power Comparison. Phase ABC = "BRG"

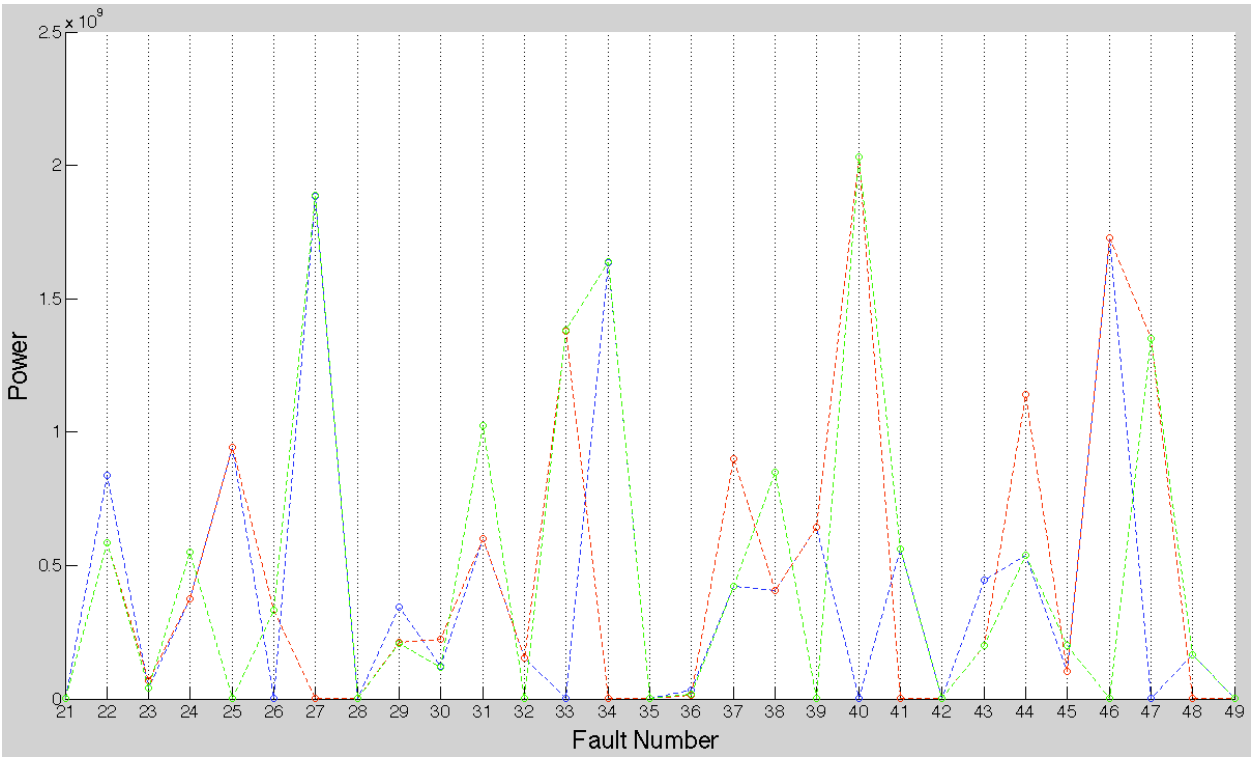

Figure 37: DB2 Detail Level 3 Power Comparison. Phase ABC = "BRG"

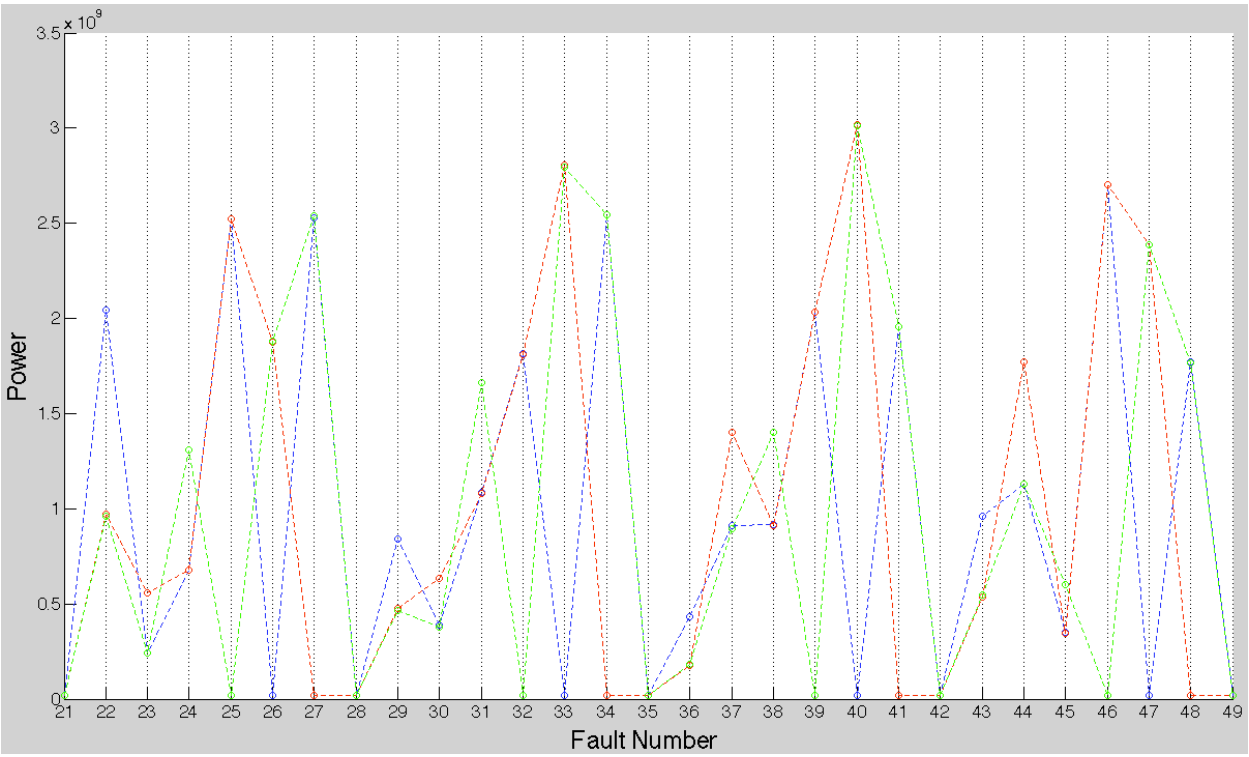

Figure 38: DB2 Detail Level 4 Power Comparison. Phase ABC = "BRG"

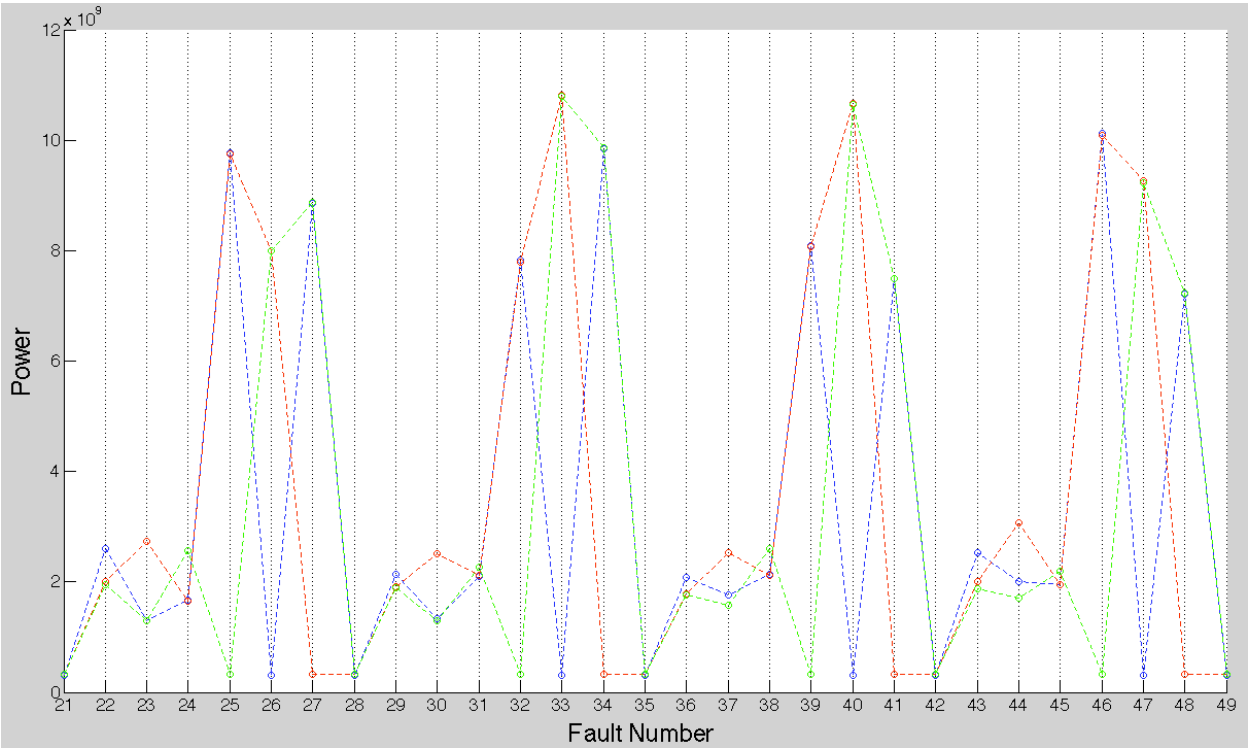

Figure 39: DB2 Detail Level 5 Power Comparison. Phase ABC = "BRG"

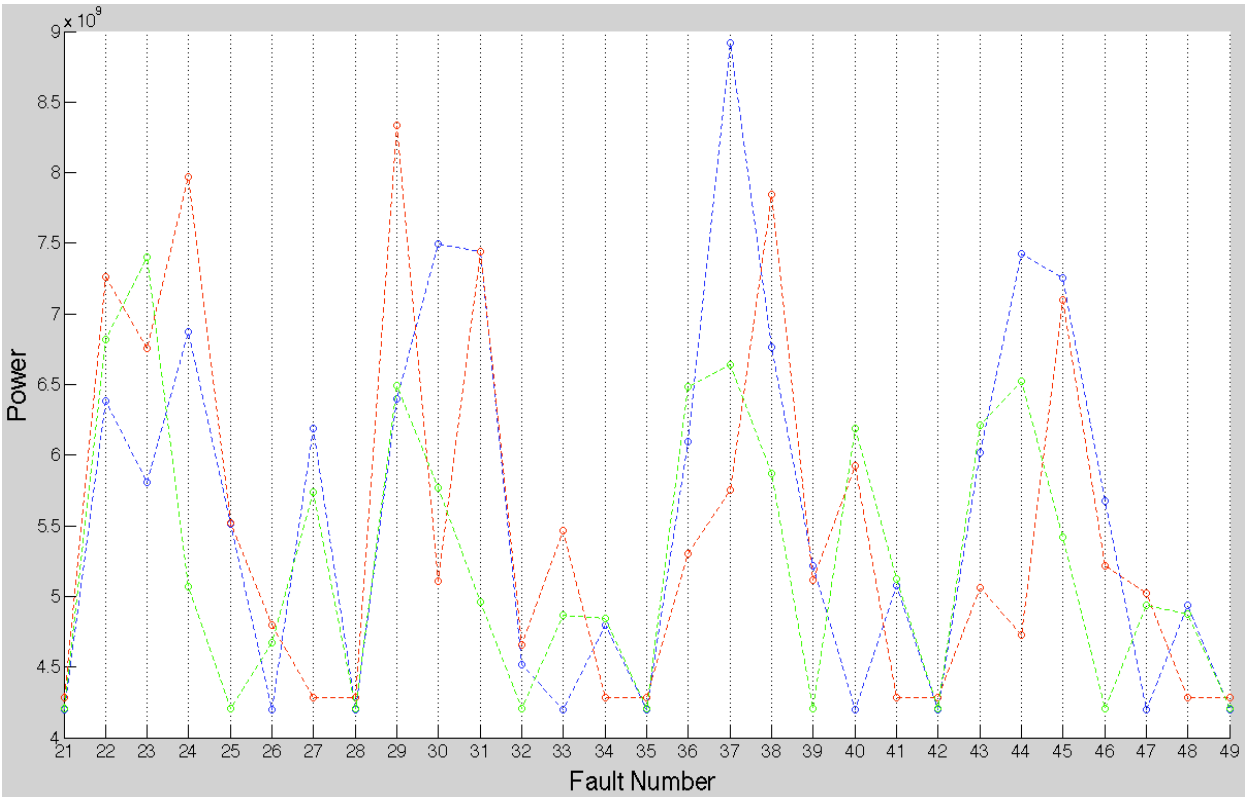

Figure 40: DB2 Detail Level 6 Power Comparison. Phase ABC = "BRG"

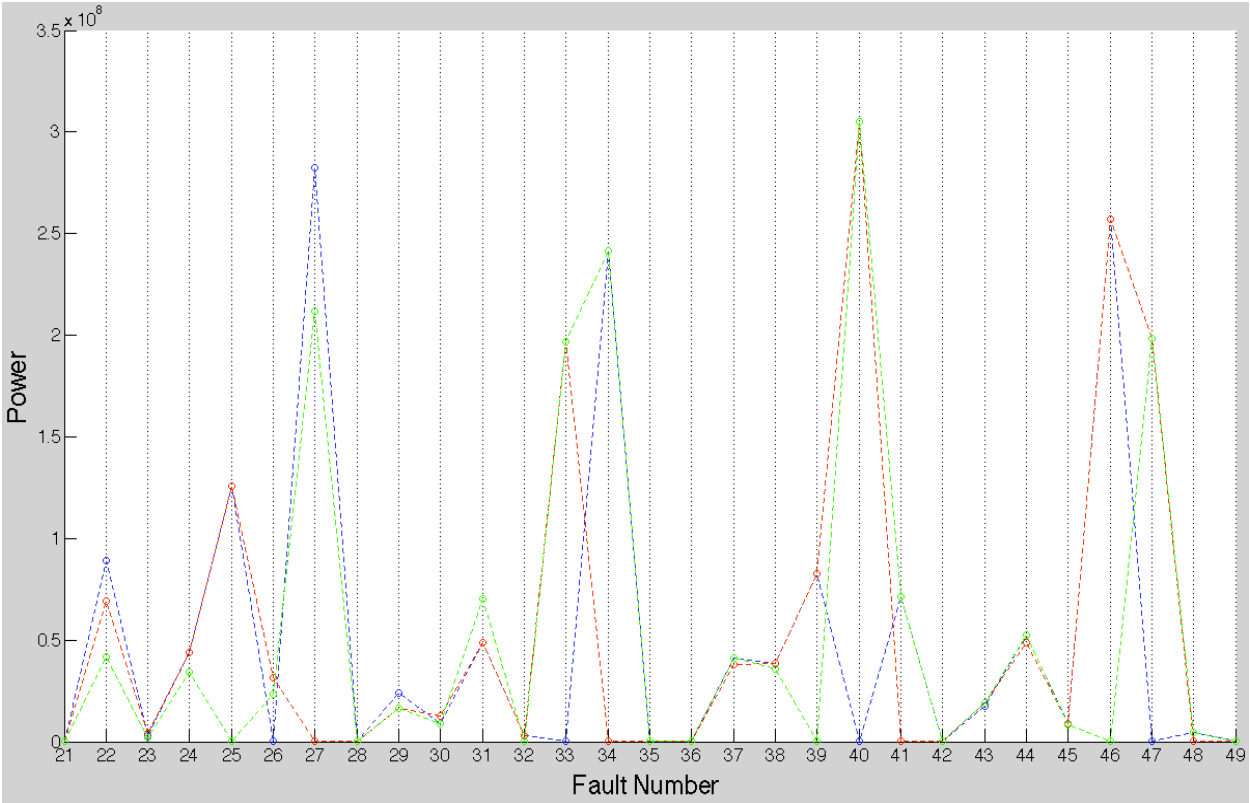

Figure 41: DB4 Detail Level 2 Power Comparison. Phase ABC = "BRG"

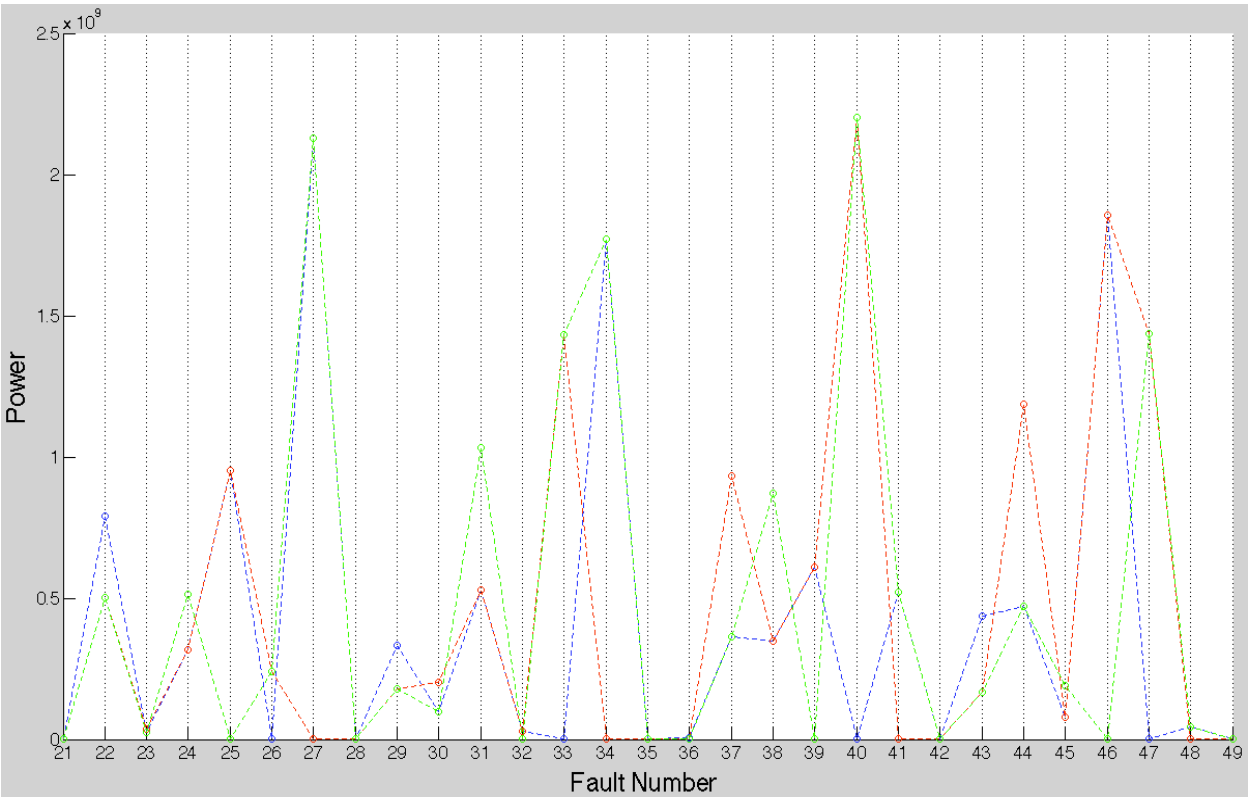

Figure 42: DB4 Detail Level 3 Power Comparison. Phase ABC = “BRG”

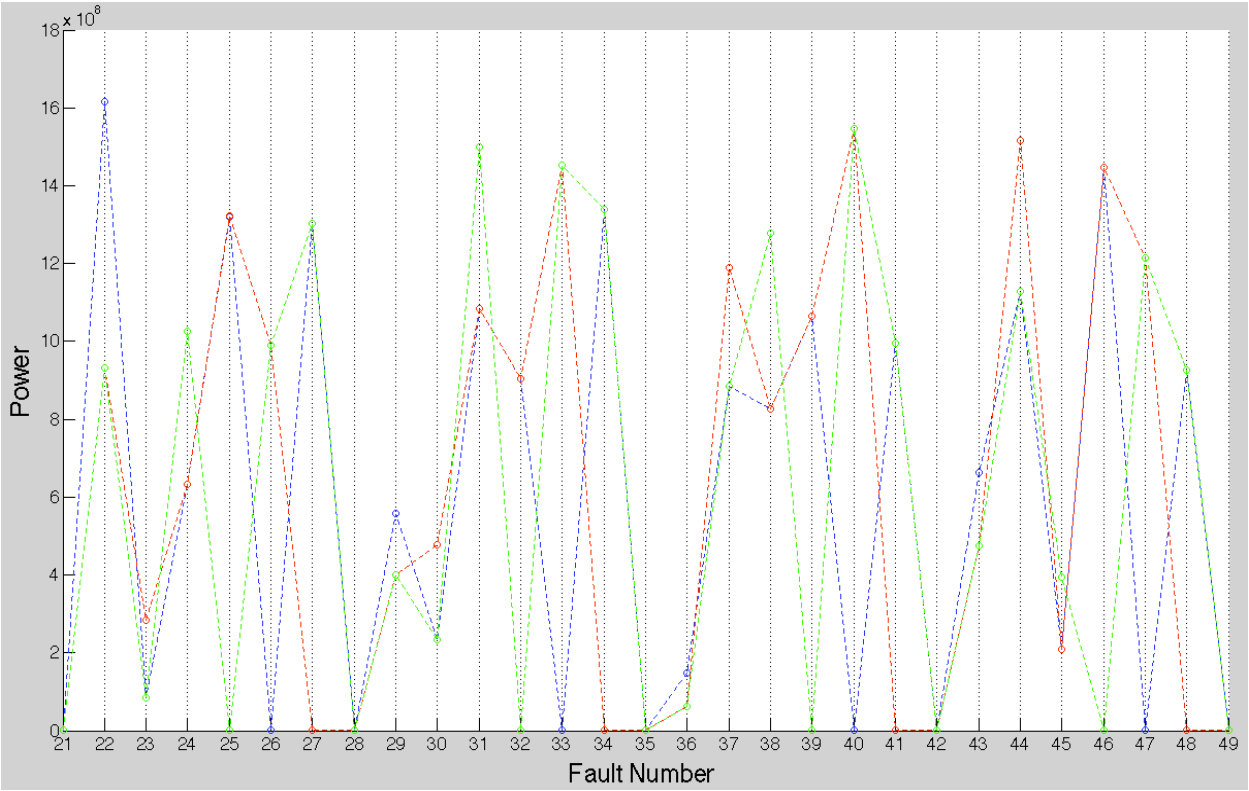

Figure 43: DB4 Detail Level 4 Power Comparison. Phase ABC = “BRG”

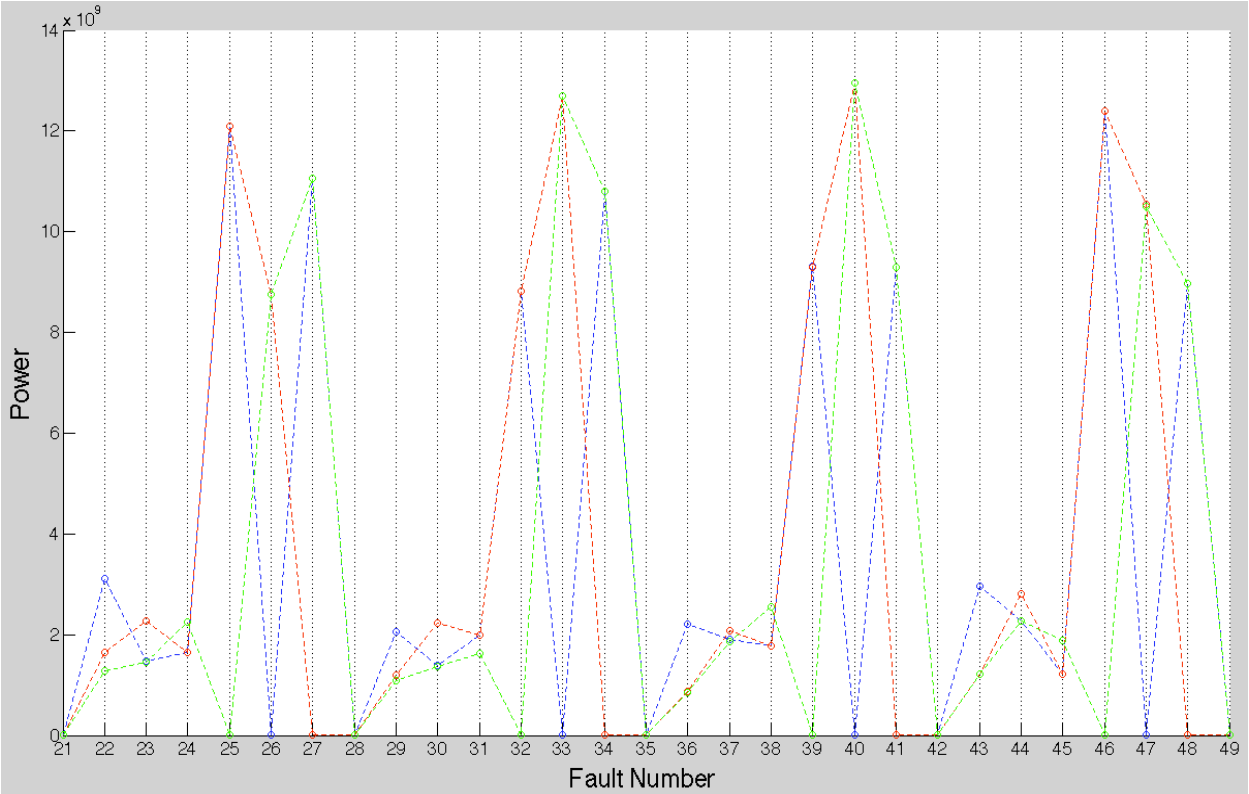

Figure 44: DB4 Detail Level 5 Power Comparison. Phase ABC = “BRG”

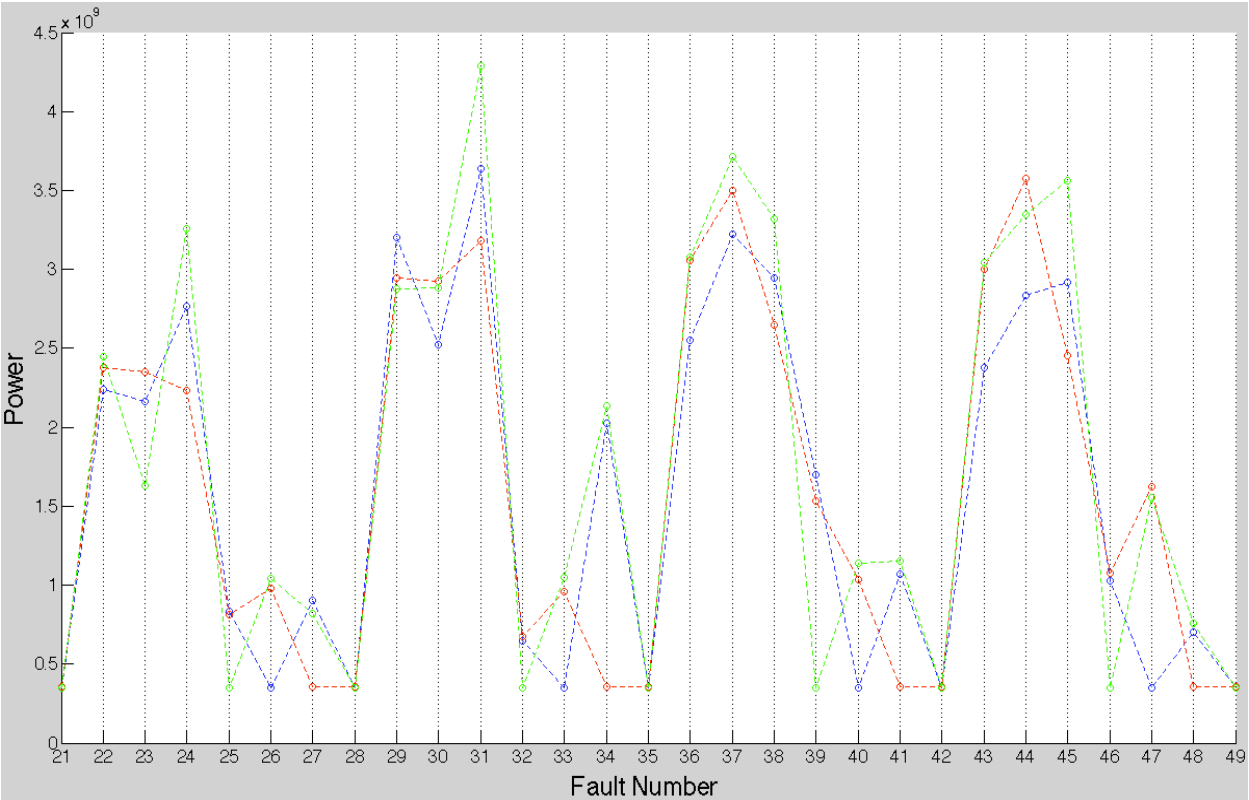

Figure 45: DB4 Detail Level 6 Power Comparison. Phase ABC = “BRG”

Testing Neural Network Capabilities

Now that the neural network is trained to classify faults correctly, a larger data set was sent through the neural network to see how the neural network would perform on more known fault conditions. First, a 2100 second (35 minute) data set was created and input to the neural network with no noise added to the system, in order for comparison purposes after creating data sets with noise added. This 2100-second data set becomes the truth reference by which the noisy data will be compared to assess performance. This will be called the good set in the following. Two tests were performed. The first test was adding noise to the system. The second test was changing the double phase faults to double phase to ground faults with different ground resistances. Two ground resistances were chosen and analyzed: 50 Ω and 100 Ω. In the multiphase test to ground, the neural network would detect multi phase to ground faults but could not discern which 2 phases were being faulted very successfully. The test for both of the ground resistances were successful in determining that faults occur, but not successful in determining which phases were faulted. Therefore, the remainder of the testing focused on single phase to ground faults and multi phase-to-phase faults under noise conditions.

The first test done was to assess the capabilities of the neural network without multi-phase to ground faults. This was accomplished by adding noise to the neutral of the three-phase source. Three different noise powers were added to the system for comparison against a known good set. The three noise powers used were 0.5, 1, and 2. Examples of how much white noise was being added to the system for each of the three different noise powers are shown in figures 46-48. Since it is unclear in Simulink the units this noise power corresponds to, 30 seconds of the noise was generated and the standard deviation was calculated on that 30-second interval in order to give an RMS value for the noise levels. The three corresponding RMS values for each noise level can be seen in table 4. An example plot of the three-phase system with noise added is shown in figure 49. For comparison, a plot of the same time frame with no noise added to the three-phase system is also shown in figure 50. Note that since the phase-to-phase RMS value of the Simulink source is 10,000V, this is equivalent to a phase-to-ground voltage of approximately 5800V.

Noise Level	RMS Value (V_{RMS})
0.5	90.5745
1	128.0917
2	181.1490

Table 4: RMS Values of each Noise Power Level Tested

After the three 2100 second data sets were created and pushed through the neural networks – one for each noise level – each of these 3 data sets were compared to the good set. The comparison showed that the performance of the neural network was not adequate as many faults were being missed and too many faults that were not actually faults were being detected. The neural network was trained with the 140-sample data set, which is 20 faults of each condition (A, B, C, AB, BC, AC, no fault), which is the same neural network as the one used in figure 8 labeled *+/-(1,1,1) fault classification.*

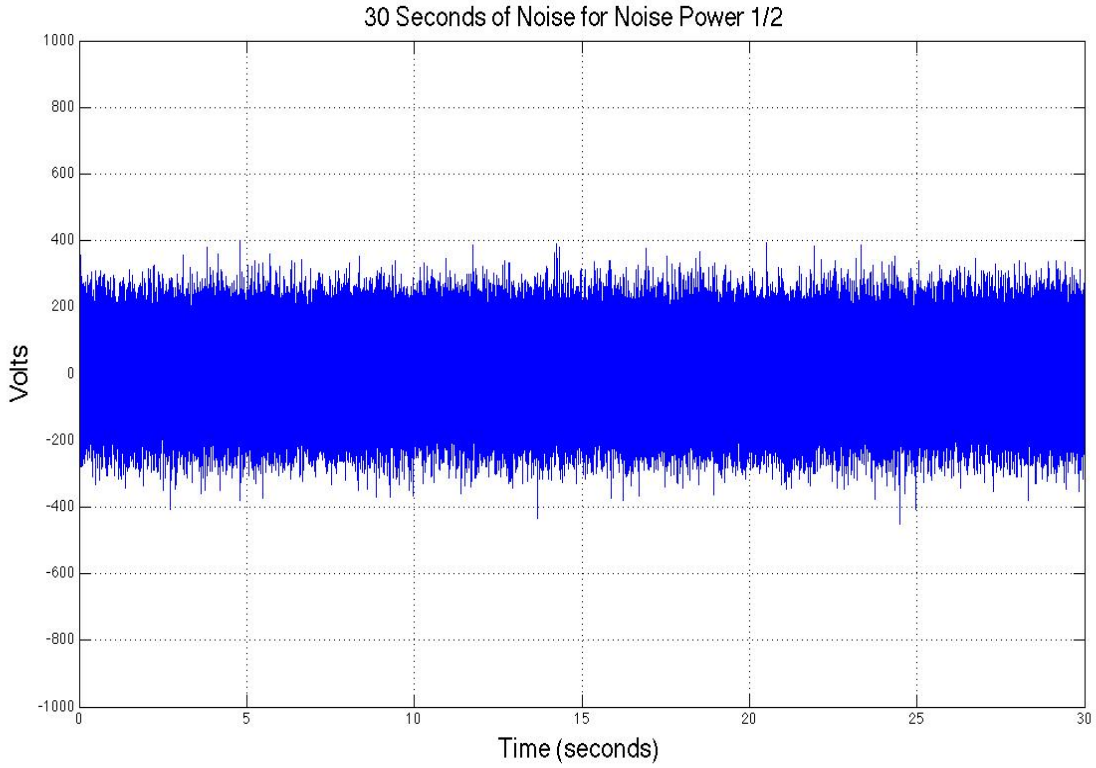

Figure 46: Noise Power ½ With an RMS Value of 90.6 V_{RMS}

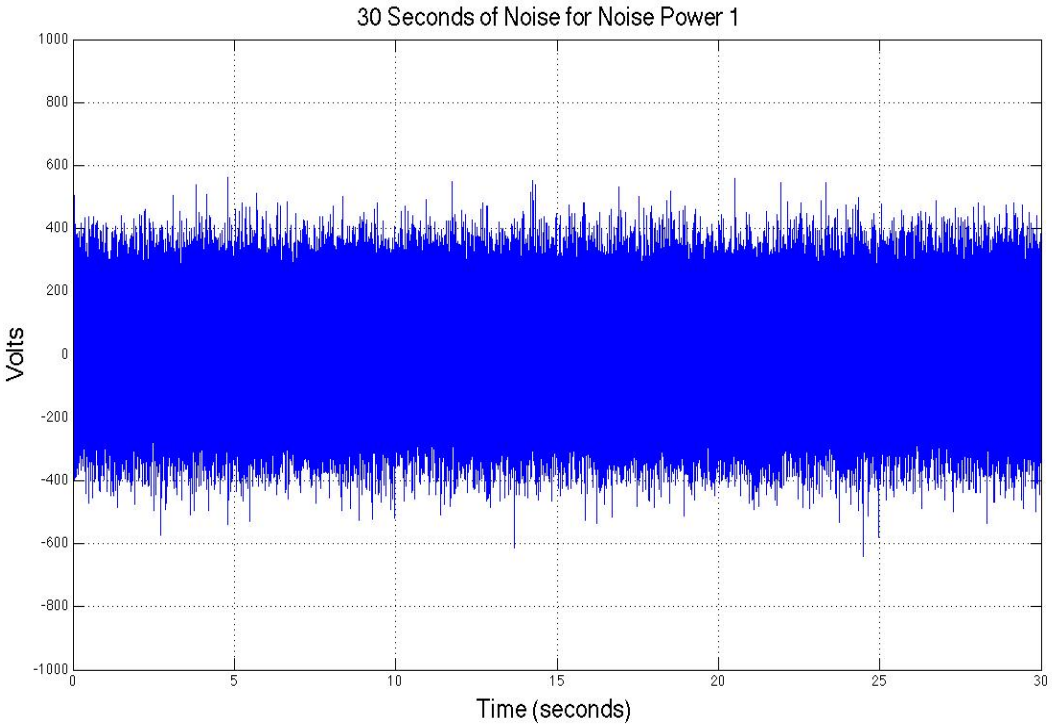

Figure 47: Noise Power 1 With an RMS Value of 128.1 V_{RMS}

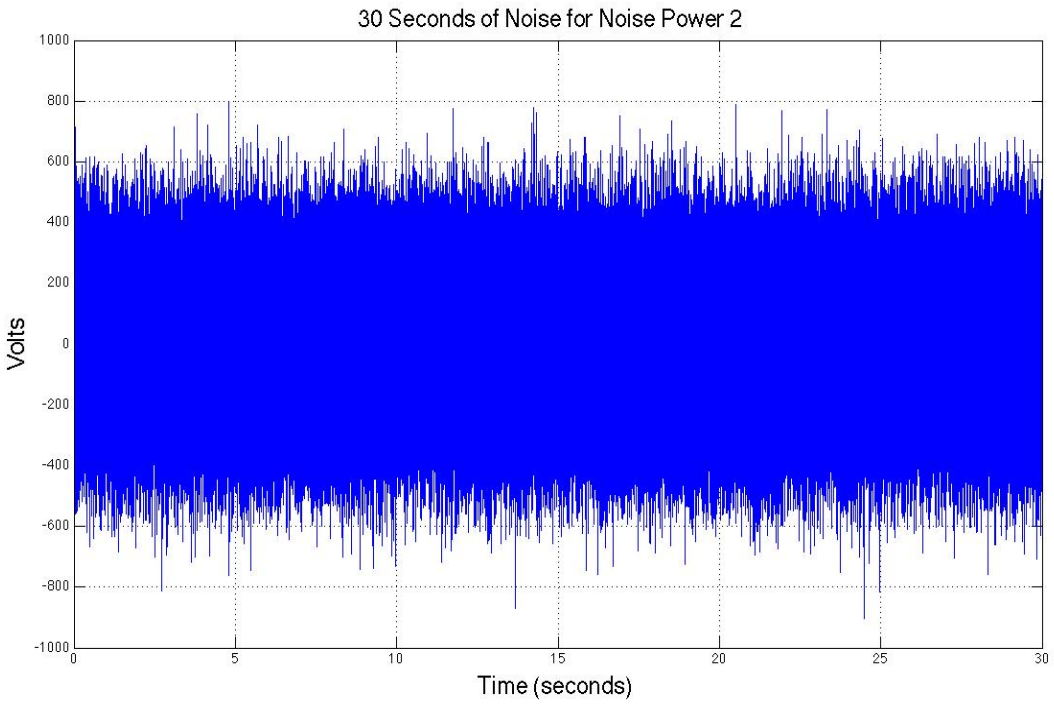

Figure 48: Noise Power 2 With an RMS Value of 181.1 V_{RMS}

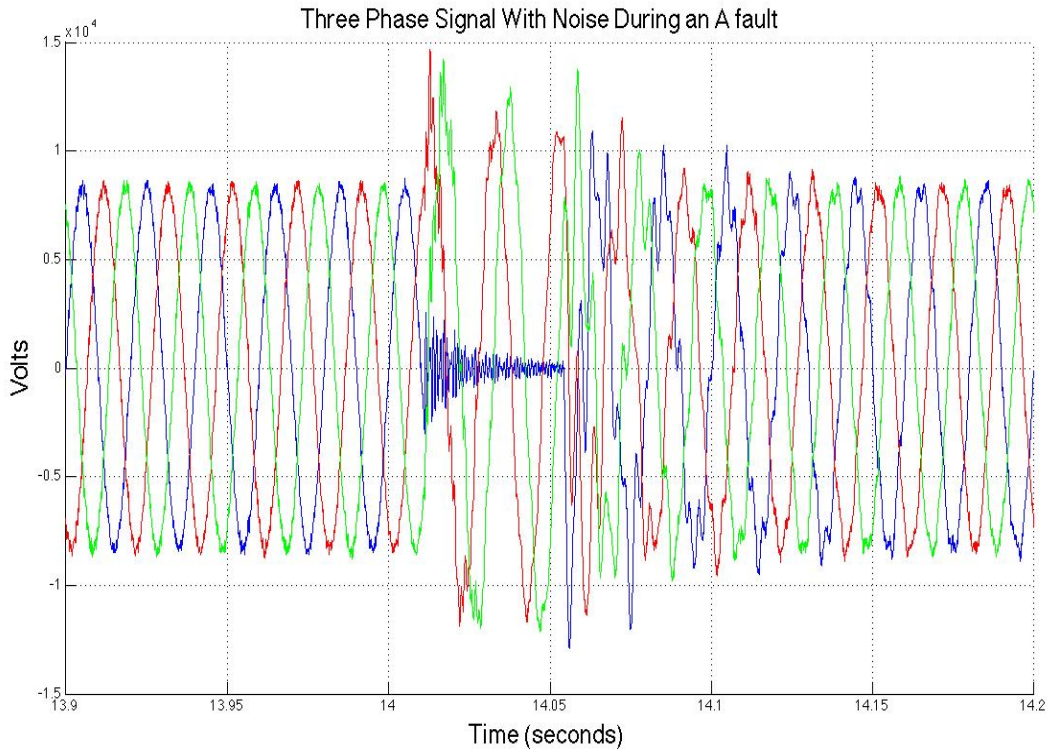

Figure 49: Three Phase Signal With Noise Power ½ Added

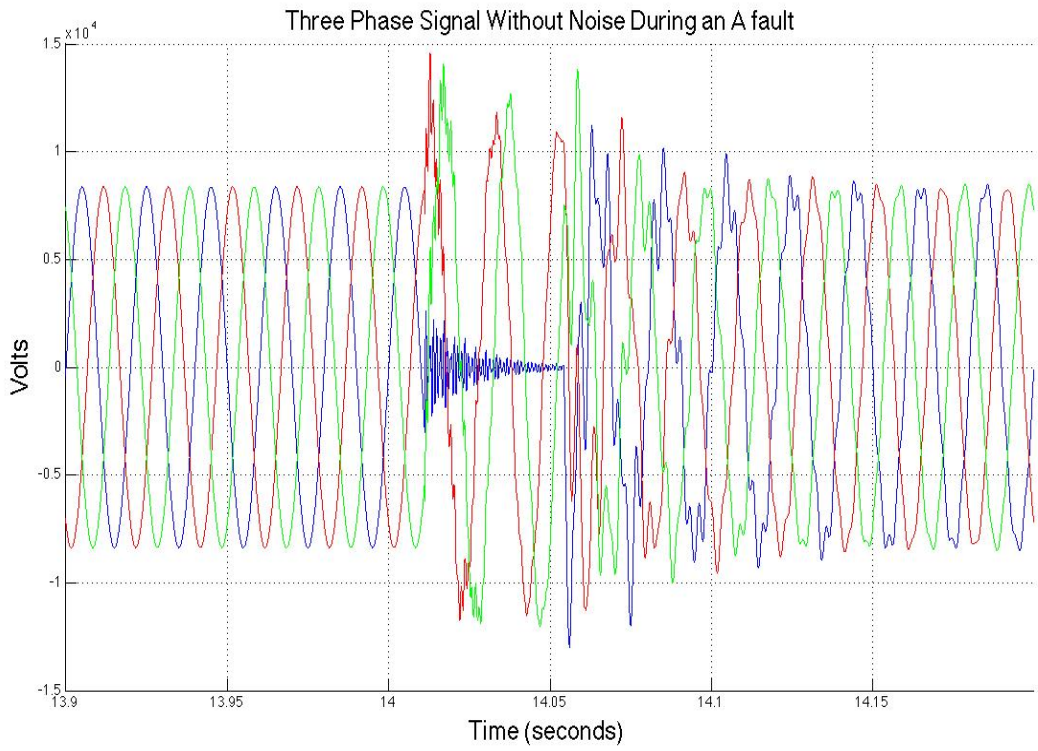

Figure 50: Three-Phase Signal Without Noise Added

Because the neural network trained with 140 faults had too many incorrect fault classifications, the neural network was retrained with a larger data set. The data set was expanded to 350 faults, or in other words 50 faults per fault condition analyzed. Two neural networks were created from this data set. The first neural network was one with 1 hidden layer of 10 neurons. The second neural network was one with 2 hidden layers of 10 neurons each. These two neural networks were inserted into the Simulink Diagram illustrated in figure 8 in place of the top 2 neural networks, because the previous 2 neural networks were deemed insufficient at fault classification with no noise applied to the system. The performance of the 2 neural networks created with the data set of 350 faults both had better performance parameters than the third neural network, which is kept as the third neural network for comparison purposes. As shown in figures 51 and 52, the MSE of the performance converges to values on the order of 10^{-5} and 10^{-6}, respectively. These are both smaller MSE by at least two orders of magnitude compared to figure 34, the neural network trained with the data set of 140 faults.

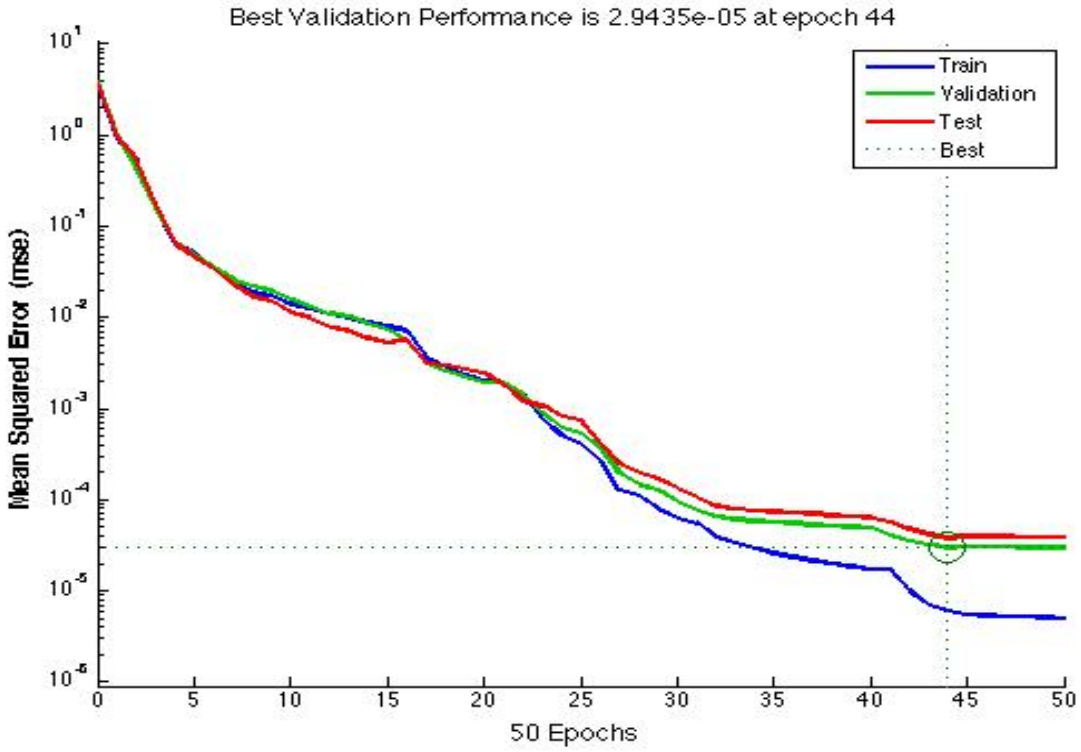

Figure 51: Performance of 1 Hidden Layer Neural Network

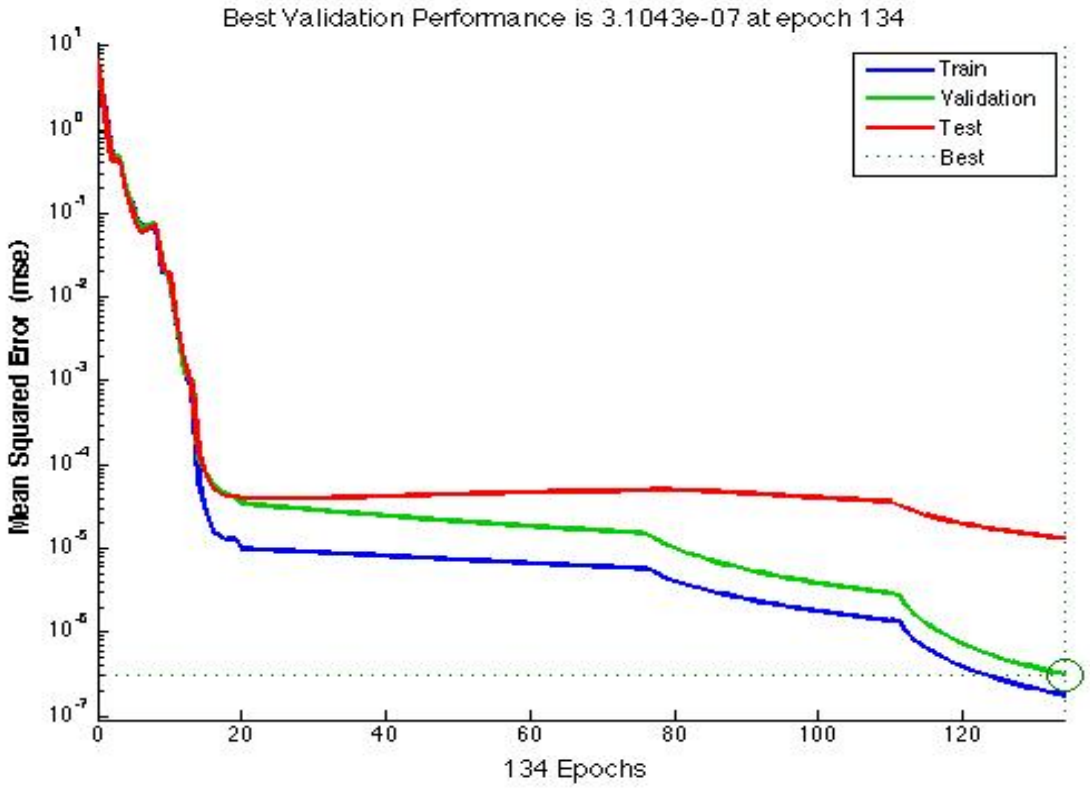

Figure 52: Performance of 2 Hidden Layer Neural Network

Tests were run on three different neural networks. As a result, one good set for each neural network, for a total of 3, had to be created with no noise applied for comparison. The comparison with respect to the good sets is with respect to two parameters. The first parameter is false alarms, where a fault was detected when a fault did not actually occur. The second parameter is a missed fault, where a fault occurred but was not detected. Each phase was compared for each neural network on each noise level, resulting in 27 different comparisons. False alarms and missed faults were counted and assembled into the three tables below.

The first neural network was trained using 350 faults and the network contained a single hidden layer with 10 neurons. The performance with respect to the two parameters described in the previous paragraph, when compared with the good set for the three power levels, is shown in table 5.

Noise Level	Phase	Missed Faults	False Alarms
0.5	A	8	8
0.5	B	1	9
0.5	C	11	11
1	A	11	17
1	B	7	32
1	C	15	41
2	A	18	62
2	B	7	110
2	C	14	103
Total		92	393

Table 5: Performance of Neural Network with 1 Hidden Layer Trained with 350 Faults

The second neural network was trained using 350 faults and the network contained 2 hidden layers with 10 neurons each. The performance with respect to the two parameters described above, when compared with the good set for the three power levels, is shown in table 6.

Noise Level	Phase	Missed Faults	False Alarms
0.5	A	0	87
0.5	B	0	7
0.5	C	34	0
1	A	0	167
1	B	1	19
1	C	48	0
2	A	1	296
2	B	4	62
2	C	72	7
Total		160	645

Table 6: Performance of Neural Network with 2 Hidden Layers Trained with 350 Faults

The third neural network was trained using 140 faults and the network contained a single hidden layer with 10 neurons. The performance with respect to the two parameters described above, when compared with the good set for the three power levels, is shown in table 7.

Noise Level	Phase	Missed Faults	False Alarms
0.5	A	15	11
0.5	B	0	7
0.5	C	3	2
1	A	14	30
1	B	4	17
1	C	11	8
2	A	17	105
2	B	7	80
2	C	23	19
Total		94	279

Table 7: Performance of Neural Network with 1 Hidden Layer Trained with 140 Faults

Of the two parameters being measured, false alarms are much worse than miss detections. This is because, if the neural network outputs a false alarm, unnecessary resources will be engaged to take action to address a fault that did not occur. A missed detection is less severe because there would be many chances to detect a true fault and thus very little chance to miss the fault. Thus, in the aggregate, very little chance to miss the fault.

By quick analysis of tables 5-7, noise level 2 is difficult for all neural networks, because there are too many false alarms being detected. A reasonable comparison will not include noise level 2, because all three neural networks do not operate sufficiently under that much noise. This amount of noise being introduced to the system provides enough high frequency content to skew the DWT calculations and provide significant amounts of miss-classifications. The worst performance in the noise level 2 was the 2 hidden layer neural network. By further looking at two lesser noise levels for the 2 hidden layer neural network, it can also be seen that there are many more false alarms.

Since the performance of all three neural networks was insufficient at noise level 2, in order to compare the two neural networks with one hidden layer, the missed detections and false alarms occurring at noise level 2 was subtracted from the total. This leaves the neural network described by table 5 with 53 missed detections and 118 false alarms during noise levels ½ and 1. The neural network described by table 7 contained 47 missed detections and 75 false alarms in noise levels ½ and 1. By comparing these numbers, the neural network trained with less data performed the best.

After comparing the results of the three neural networks at the three different noise levels, it is interesting to note that the neural network trained with the most data and with the most complicated hidden layer configuration performed the worst at all three noise levels. It is also interesting to note that although the two neural networks with single hidden layers had the same amount of missed detections,

the neural network trained with less data had the least false alarms. This makes it the best performing neural network of the three. The two neural networks with 1 hidden layer operate sufficiently under noise levels ½ and 1. All three neural networks perform unacceptably at noise level 2.

An interesting point to note is that all the false alarms are single point events, so by eliminating these single point events, all false alarms would be eliminated. However, this increases the number of missed detections because some of the fault detections were also single events in noise cases. In the noiseless case, a fault occurred for a longer period of time, so it is reasonable to increase the number of missed events while simultaneously eliminating all false alarms. This is because false alarms are significantly more resource costly than a missed event. Figure 53 illustrates a missed detection, a single point correct detection, and a false alarm. The blue is a graph of the good set for the corresponding neural network offset by +3, the red plot is the output of the same neural network with noise applied, and the green is the two multiplied together offset by -3. Thus, -1 indicates differences (-4 on the plot). The missed detection is at time 347, the false alarm is at time 317, and the single point correct detection is seen at time 340.

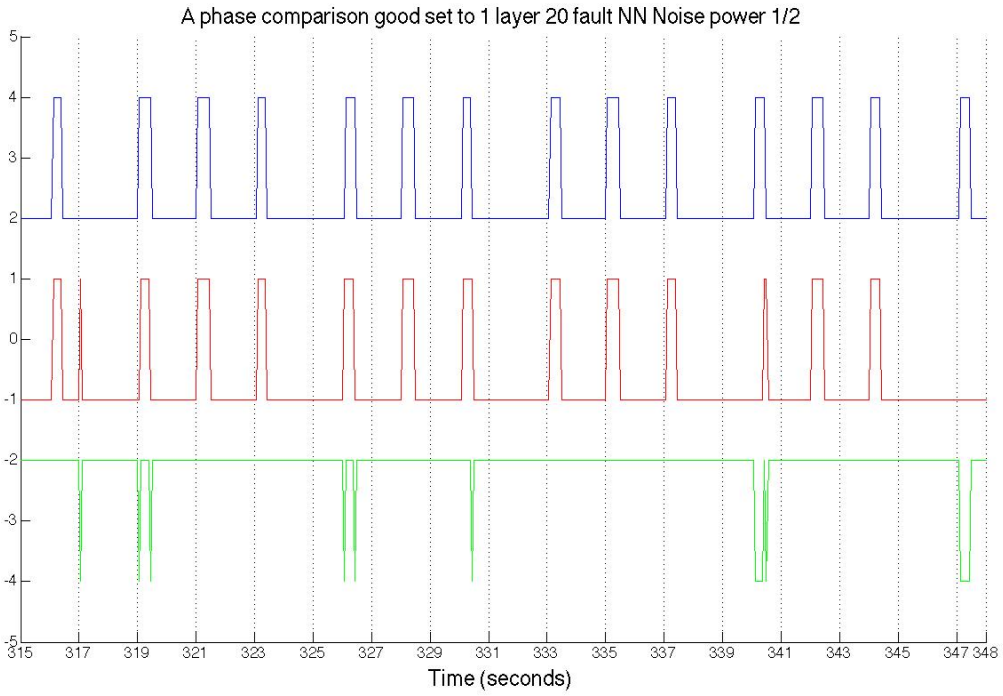

Figure 53: Example of False Alarm and Missed Detection

V. Conclusion and Future Works:

The training of the neural network with no noise was successful. There was high correlation between the power calculations of the detail level 4 wavelet coefficients and the faults in which a fault condition occurred. The neural network converged to a solution that produced proper classifications of all fault conditions generated when the multi-phase faults were phase-to-phase faults. When the multi-phase faults were phase-to-ground, the neural network had a problem distinguishing between double phase-to-ground faults and a fault in all three phases. It was determined that the neural network classifies faults adequately when noise levels of 90.5 V_{RMS} and 128.1 V_{RMS} are applied to the neutral line of the three phase source. The highest performing neural network contained 1 hidden layer with 10 neurons and training set of 140 faults. This neural network performed more successfully than the neural network of equivalent hidden layer configurations and a larger training set of 350 faults, which was not expected.

The Wavelet Transform is a mathematically intensive operation that requires complex and advanced linear algebra techniques to meet fundamental requirements. For instance, the mathematical operation called the wavelet transform must consist of a basis of vectors that are all orthonormal to each other. This requirement ensures that any signal decomposition using the transform can be reversed and the signal can be *completely* reconstructed. In other words, an inverse operation exists. In linear algebra terms, the matrix of coefficients given by the operation is invertible. With an orthonormal basis, this can be accomplished instantly by transposing the matrix. The mathematical details behind the operation are not yet understood. For future works, I will use my knowledge of linear algebra to further understand the mathematics behind the wavelet transform.

As far as the neural network implementation goes, I plan on further studying the different algorithms and network topologies to see if they converge to solutions faster and more effectively than the feed-forward neural network using back propagation weight updates accomplished throughout this project. Even more so, I would like to execute trial and error to obtain a better-hidden layer configuration that converges to an optimal solution the quickest. It was mentioned in the literature review that there are many different kinds of power line system faults, and I chose to simulate two of them: single phase-to-ground and double phase-to-phase faults. The more common power system faults are line to ground and line-to-line faults. A common example of a line-to-line fault is a bird landing on the transmission line and causing two of them to touch. This can occur very frequently. As a result, I plan on investigating the other types of power distribution system faults mentioned here.

The results for changing the multi-phase-to-phase faults into multi-phase-to-ground faults were unsuccessful. As another test, the neural network could be trained with both multiphase-to-phase faults and multiphase-to-ground faults to see if training to classify both faults at the same time provides sufficient discrimination between the two types of faults for the neural network to provide proper classification. Another interesting result to investigate would be to apply a single point eliminator to the outputs of the neural network under noise conditions to assess the impact of improved false alarm performance at the penalty of increased missed detections.

References

[1] Settipalli, Praveen. May 2007. “Automated Classification of Power Quality Disturbances Using Signal Processing Techniques and Neural Networks.” University of Kentucky, 2007.

[2] Patel, Mamta. June 2012. “Fault Detection and Classification on A Transmission Line using Wavelet Multi Resolution Analysis and Neural Network.”

[3] Kasinathan, Karthikeyan. 2007. “Power System Fault Detection and Classification by Wavelet Transforms and Adaptive Resonance Theory Neural Networks.” University of Kentucky, 2007.

[4] Sathiya priya, K. Geethanjali, M. “Combined Wavelet Transforms and Neural Network (WNN) Based Fault Detection and Classification in Transmission Lines.” 2006.

[5] Lampley, Glenn C. “Fault Detection and Location on Electrical Distribution System.” IEEE. Carolina Power & Light. 2002.

[6] Xiaohua, Yang. Yadong, Zhang. Zhongmei, Xi. “Wavelet Neural Network Based Fault Detection Method in Power System.” IEEE. 2003.

[7] Blumenstein, Michael, Xin Yu Liu, and Brijesh Verma. "Investigation of the Modified Direction Feature for Cursive Character Recognition." *ScienceDirect*. Elsevier, 14 May 2006. Web. 20 Oct. 2013. <http://dl.acm.org/citation.cfm?id=1221195>

[8] Huang, Shyh-Jier, and Cheng-Tao Hsieh. "High-Impedance Fault Detection Utilizing A Morlet Wavelet Transform Approach." *IEEE*. N.p., Oct. 1999. Web. 14 Nov. 2013.

[9] Kennedy, James, and Russell Eberhart. "Particle Swarm Optimization." IEEE. 1995. Web.

[10] Narendra, Kumpati S., and Kannan Parthasarathy. "Identification and Control of Dynamical Systems Using Neural Networks." *IEEE*. N.p., Mar. 1990. Web. 5 Nov. 2013.

[11] Tayeb, Eisa Bashier M. "Faults Detection in Power Systems Using Artificial Neural Network." *American Journal of Engineering Research* 2320-0847 02.06 (2013): 69-75. Web. 30 Dec. 2013.

[12] Poungponsri, Suranai, and Xiao-Hua Yu. "Electrocardiogram (ECG) Signal Modeling and Noise Reduction Using Wavelet Neural Networks." Proceedings of the IEEE International Conference on Automation and Logistics, Aug. 2009. Web. Nov. 2013.

[13] Wang, Xiao-bin, Guang-yuan Yang, Yi-chao Li, and Dan Liu. "Review on the Application of Artificial Intelligence in Antivirus Detection System." University of Electronic Science and Technology of China. IEEE. 2008.

[14] Lyons, Richard. "Understanding Cascaded-Integrator Comb Filters" Embedded. 31 March. 2005. 30 March. 2014 <http://www.embedded.com/design/configurable-systems/4006446/Understanding-cascaded-integrator-comb-filters>

Appendix A – Analysis of Senior Project Design

Power System Fault Detection with the Discrete Wavelet Transform and Artificial Neural Networks

Kevin Keegan

Advisor: Xiao-Hua Yu

1) Summary of Functional Requirements
    a) Describe the overall capabilities or functions of your project or design. Describe what your project does.
        i) The three-phase power distribution system generates various fault conditions to train a feed-forward artificial neural network using back propagation algorithm. Computer simulation results show this neural network based approach can successfully detect various fault conditions.
2) Primary Constraints
    a) Describe significant challenges or difficulties associated with your project or implementation. For example, what were limiting factors, or other issues that impacted your approach?
        i) It is difficult to determine which mother wavelet allows for best feature extraction of the fault as well as which decomposition level accomplishes the best feature extraction. Knowledge of Matlab and Simulink impacted the design approach. As more knowledge about Matlab and Simulink was attained, more and more components of the design were implemented in Simulink. A Simulink Matlab function could not be used because I could not integrate the XCode compiler with Simulink.
3) Economic
    a) What Economic Impacts will result?
        i) Human Capital
            (1) The implementation of the neural network design would be a convenience for employees of power distribution companies. The neural network accomplishes one of many tasks that these employees have and with striking efficiency.
        ii) Financial Capital
            (1) The primary purpose of the implementation of the neural network is that it is a solution to a problem. That problem is not being able to detect fault conditions in a 3-phase power system. When a fault occurs, a lot of money and manufacturing time is lost. Neural networks offer a way to detect the fault before a power outage and all that time and money is lost.
        iii) Manufactured Capital

(1) The neural network design implementation will ultimately reduce the amount of time lost to power outages caused by fault conditions in the transmission line. As a result, this will increase the overall time the power is on and increase the manufactured capital.

iv) Natural Capital

(1) Benefits occur indirectly. The neural network provides a solution to the problem of losing power, time, and money because of a fault in the transmission line. By detecting the fault before it happens, the benefit is a reduction of resources lost because of the fault.

4) Costs
   a) This project is solely simulation-based project.
   b) MATLAB and Simulink Student Suite - $99.00
   c) Neural Network Toolbox - $29.00
   d) Wavelet Toolbox - $29.00
   e) SimPowerSystems - $29.00
   f) SimScape - $29.00
5) Environmental
   a) Because faults cause a loss of power, many industries would benefit from knowing that a power failure was imminent. For example, chemical plants and sewage treatment plants could have dire consequences should they lose power even for a few seconds. Any industry that may release toxic agents into the environment if safety mechanisms were disabled because of loss of power will benefit from the neural network design.
6) Manufacturability
   a) The neural network implementation helps minimize or eliminate manufacturing downtime as a result of a fault condition in the three-phase system. Also, manufacturing of the device will be limited, as only power distribution companies need them. Since the risk of unpredictable faults creates such huge potential losses for these companies, the demand for the neural network system will be strong enough to allow for significantly high prices.
7) Sustainability
   a) Describe any issues or challenged associated with maintaining the completed device, or system.
      i) Sustainability problems will mostly be due to external physical apparatus. The implementation of the design must be done across transmission lines many kilometers long. These transmission lines exposed to weather, animals, and time. As transmission lines lose their integrity, more external forces will have more profound effect upon voltages and/or currents flowing through the transmission lines allowing for possible skewed readings. To compensate for this, when any significant changes

in integrity or condition of the transmission lines occur, the neural network should be retrained for optimal fault detection.

b) Describe any upgrades that would improve the design of the project.
   i) One property of neural networks is that any configuration of hidden layers and neurons has the possibility of converging to a solution faster and better than one currently used. To improve the design could be as simple as finding a better hidden layer configuration. There can also be a more effective way of distinguishing the different fault conditions from each other.

c) Describe any issues or challenges associated with upgrading the design.
   i) Because one possible upgrade to the design is finding a hidden layer configuration, the search can be endless. Also, depending on the data size used to train the neural network, it could take an extensive amount of time to train the network.

8) Ethical
   a) According to IEEE code of ethics part 3, an engineer is required "to be honest and realistic in stating claims or estimates based on available data." This is particularly crucial concerning the development of the neural network to classify fault conditions in a three-phase power system. The validity of neural network classification of fault conditions must be well known to all users of the system. If a wrong classification is made, the safety of power distribution company employees can be put at risk.
   b) According to Utilitarian ethics, decisions are based upon the decision that brings about the highest good for all. The implementation of the neural network for fault management has the ability to minimize power grid failures. This maximizes the availability of power to the greatest number of people.

9) Health and Safety
   a) The environmental issues associated with the neural network design pose health and safety problems. If a chemical plant or sewage treatment facility experiences a power failure, safety mechanisms may lose the ability to keep toxic agents from entering the environment. Furthermore, hospitals depend on power. Any emergency facility is affected if a power grid failure occurs. Traffic lights are another example that would affect a lot of people if the power grid fails.

10) Social and Political
   a) Describe social and political issues associated with design, manufacture, and use.
      i) Power distribution companies provide power to everyone; hence it is a public utility. Being a public utility, it is political by nature, and has a significant impact on society. The power grid is widely regulated, and anything that would improve its availability would improve its stature in this political structure.

11) Development

a) Describe any new tools or techniques, used for either development or analysis that was learned independently during the course of your project.

i) CIC filters: Cascaded Integrator Comb filters were used to calculate the moving average power that allows me to apply the neural network to entire data sets of fault conditions in real time.

ii) DWT: The Discrete Wavelet Transform was used as the technique for feature extraction throughout the design. The DWT is a mathematical operation that involves applying high pass and low pass filters to a signal in order to extract higher frequency components. This method was used to extract the higher frequency components that fault conditions cause on the power line.

iii) Simulink: Simulink is the simulation program used for the design. In the diagram, known fault conditions are generated at known times. It has the functionality to change types of faults to generate and change the amount of band-limited white noise in the simulation.

The inputs to this function are the three phases of the output of the DWT blocks from the Simulink diagram, as well as the level of decomposition being performed. The function outputs the powers calculated for each fault in each of the decomposition levels D2, D3, D4, and D5. The mother wavelet used was DB4 for this function. This also generates figures 36-45.

```
function [ PA,PB,PC ] = power_arrays(dwt_output1,level)

dwt = squeeze(dwt_output1.signals.values);

% Separating the three phases because they were multi-plexed together in
% the Simulink diagram prior to being output to the Workspace
dwt_A = dwt(1:16384,:);
dwt_B = dwt(16385:32769,:);
dwt_C = dwt(32770+2:49152,:);

% dwt_A = squeeze(dwt_A);
% dwt_B = squeeze(dwt_B);
% dwt_C = squeeze(dwt_C);

num_of_faults = size(dwt_A,2);

PA = zeros(1,level);
PB = zeros(1,level);
PC = zeros(1,level);

for j = 2:num_of_faults
    [P_A,P_B,P_C]=powercalculation(dwt_A(:,j),dwt_B(:,j),dwt_C(:,j),level);
    PA = [PA; P_A];
    PB = [PB; P_B];
    PC = [PC; P_C];
end

PA(1,:) = [];
PA(:,level) = [];
PB(1,:) = [];
PB(:,level) = [];
PC(1,:) = [];
PC(:,level) = [];

axn = 1:140;


for k = 1:5
    figure(k)

    hold on

    plot(axn(21:49),PA(21:49,k), '--ob')
    plot(axn(21:49),PB(21:49,k), '--or')
    plot(axn(21:49),PC(21:49,k), '--og')

    hold off

    set(gca,'XLim',[21 49]);
```

```
    set(gca,'XTick',[21:1:49],'XGrid','on');

    xlabel('Fault Number','Fontsize',24)
    ylabel('Power','Fontsize',24)

end
end
```

This function takes as inputs the discrete wavelet transforms of phases A, B, and C. It calculates the power in each of the decomposition levels by summing the squares of the values in each detail level by the indices determined directly from plotting a DWT of any arbitrary fault, and observing where the fault occurs.

```
function [ P_A,P_B,P_C ] = powercalculation( DWT_A,DWT_B,DWT_C,level )

% These indices give the range where the fault information is occuring in
% each detail level - starting from level 2 at the first index, and ending
% with level 6.

sndx = [9408, 12890, 14638, 15520, 15936];
endx = [9728, 13120, 14976, 15696, 16016];

level = length(endx);

sndx(level+1) = 0;
endx(level+1) = 0;
P_A = zeros(1,level+1);
P_B = zeros(1,level+1);
P_C = zeros(1,level+1);


for i = 1:level
    P_A(i) = sum(DWT_A(sndx(i):endx(i)).^2);
    P_B(i) = sum(DWT_B(sndx(i):endx(i)).^2);
    P_C(i) = sum(DWT_C(sndx(i):endx(i)).^2);
end

end
```

This script is an analysis script used for testing the Neural Network Capabilities under noise conditions, more specifically the three noise levels of ½, 1, and 2 described in the results section. It generated figure 53.

```
% Noise power 1/2
x1 = NN_Output_50faults1layer2100powerhalf.signals.values;
x2 = NN_Output_50faults2layer2100powerhalf.signals.values;
x3 = NN_Output_20faults1layer2100powerhalf.signals.values;

y1 = NN_Output_50faults1layer2100power1.signals.values;
y2 = NN_Output_50faults2layer2100power1.signals.values;
y3 = NN_Output_20faults1layer2100power1.signals.values;

z1 = NN_Output_50faults1layer2100power2.signals.values;
z2 = NN_Output_50faults2layer2100power2.signals.values;
z3 = NN_Output_20faults1layer2100power2.signals.values;
```

```
t = NN_Output_50faults1layer2100powerhalf.time;

GoodSet1r = GoodSet1.signals.values; % GoodSet for NN1
GoodSet2r = GoodSet2.signals.values; % NN2
GoodSet3r = GoodSet3.signals.values; % NN3

errorsx1 = GoodSet1r.*x1;
errorsx2 = GoodSet2r.*x2;
errorsx3 = GoodSet3r.*x3;
errorsy1 = GoodSet1r.*y1;
errorsy2 = GoodSet2r.*y2;
errorsy3 = GoodSet3r.*y3;
errorsz1 = GoodSet1r.*z1;
errorsz2 = GoodSet2r.*z2;
errorsz3 = GoodSet3r.*z3;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Noise power 1/2

% x1
errorNDXa = find(errorsx1(:,1) == -1);
errorNDXb = find(errorsx1(:,2) == -1);
errorNDXc = find(errorsx1(:,3) == -1);

terrorA = t(errorNDXa);
terrorB = t(errorNDXb);
terrorC = t(errorNDXc);

deltatimeA = diff(terrorA);
deltatimeB = diff(terrorB);
deltatimeC = diff(terrorC);

terrorA(1) = [];
terrorB(1) = [];
terrorC(1) = [];

figure(1)
plot(terrorA,deltatimeA,'b');
figure(2)
plot(terrorB,deltatimeB,'r');
figure(3)
plot(terrorC,deltatimeC,'g');

figure(28)
title('A phase comparison good set to 1 layer 50 fault NN Noise power 1/2')
hold on
plot(t,GoodSet1r(:,1)+3,'b')
plot(t,x1(:,1),'r')
plot(t,errorsx1(:,1)-3,'g')
hold off

figure(29)
title('B phase comparison good set to 1 layer 50 fault NN Noise power 1/2')
hold on
plot(t,GoodSet1r(:,2)+3,'b')
plot(t,x1(:,2),'r')
```

```
plot(t,errorsx1(:,2)-3,'g')
hold off

figure(30)
title('C phase comparison good set to 1 layer 50 fault NN Noise power 1/2')
hold on
plot(t,GoodSet1r(:,3)+3,'b')
plot(t,x1(:,3),'r')
plot(t,errorsx1(:,3)-3,'g')
hold off

% x2
errorNDXa = find(errorsx2(:,1) == -1);
errorNDXb = find(errorsx2(:,2) == -1);
errorNDXc = find(errorsx2(:,3) == -1);

terrorA = t(errorNDXa);
terrorB = t(errorNDXb);
terrorC = t(errorNDXc);

deltatimeA = diff(terrorA);
deltatimeB = diff(terrorB);
deltatimeC = diff(terrorC);

terrorA(1) = [];
terrorB(1) = [];
terrorC(1) = [];

figure(4)
plot(terrorA,deltatimeA,'b');
figure(5)
plot(terrorB,deltatimeB,'r');
figure(6)
plot(terrorC,deltatimeC,'g');

figure(31)
title('A phase comparison good set to 2 layer 50 fault NN Noise power 1/2')
hold on
plot(t,GoodSet2r(:,1)+3,'b')
plot(t,x2(:,1),'r')
plot(t,errorsx2(:,1)-3,'g')
hold off

figure(32)
title('B phase comparison good set to 2 layer 50 fault NN Noise power 1/2')
hold on
plot(t,GoodSet2r(:,2)+3,'b')
plot(t,x2(:,2),'r')
plot(t,errorsx2(:,2)-3,'g')
hold off

figure(33)
title('C phase comparison good set to 2 layer 50 fault NN Noise power 1/2')
hold on
plot(t,GoodSet2r(:,3)+3,'b')
plot(t,x2(:,3),'r')
```

```
plot(t,errorsx2(:,3)-3,'g')
hold off

% x3
errorNDXa = find(errorsx3(:,1) == -1);
errorNDXb = find(errorsx3(:,2) == -1);
errorNDXc = find(errorsx3(:,3) == -1);

terrorA = t(errorNDXa);
terrorB = t(errorNDXb);
terrorC = t(errorNDXc);

deltatimeA = diff(terrorA);
deltatimeB = diff(terrorB);
deltatimeC = diff(terrorC);

terrorA(1) = [];
terrorB(1) = [];
terrorC(1) = [];

figure(7)
plot(terrorA,deltatimeA,'b');
figure(8)
plot(terrorB,deltatimeB,'r');
figure(9)
plot(terrorC,deltatimeC,'g');

figure(34)
title('A phase comparison good set to 1 layer 20 fault NN Noise power 1/2')
xlabel('Time (seconds)')
hold on
plot(t,GoodSet3r(:,1)+3,'b')
plot(t,x3(:,1),'r')
plot(t,errorsx3(:,1)-3,'g')
hold off

figure(35)
title('B phase comparison good set to 1 layer 20 fault NN Noise power 1/2')
hold on
plot(t,GoodSet3r(:,2)+3,'b')
plot(t,x3(:,2),'r')
plot(t,errorsx3(:,2)-3,'g')
hold off

figure(36)
title('C phase comparison good set to 1 layer 20 fault NN Noise power 1/2')
hold on
plot(t,GoodSet3r(:,3)+3,'b')
plot(t,x3(:,3),'r')
plot(t,errorsx3(:,3)-3,'g')
hold off
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Noise power 1
```

```
% y1
errorNDXa = find(errorsy1(:,1) == -1);
errorNDXb = find(errorsy1(:,2) == -1);
errorNDXc = find(errorsy1(:,3) == -1);

terrorA = t(errorNDXa);
terrorB = t(errorNDXb);
terrorC = t(errorNDXc);

deltatimeA = diff(terrorA);
deltatimeB = diff(terrorB);
deltatimeC = diff(terrorC);

terrorA(1) = [];
terrorB(1) = [];
terrorC(1) = [];

figure(10)
plot(terrorA,deltatimeA,'b');
figure(11)
plot(terrorB,deltatimeB,'r');
figure(12)
plot(terrorC,deltatimeC,'g');

figure(37)
title('A phase comparison good set to 1 layer 50 fault NN Noise power 1')
hold on
plot(t,GoodSet1r(:,1)+3,'b')
plot(t,y1(:,1),'r')
plot(t,errorsy1(:,1)-3,'g')
hold off

figure(38)
title('B phase comparison good set to 1 layer 50 fault NN Noise power 1')
hold on
plot(t,GoodSet1r(:,2)+3,'b')
plot(t,y1(:,2),'r')
plot(t,errorsy1(:,2)-3,'g')
hold off

figure(39)
title('C phase comparison good set to 1 layer 50 fault NN Noise power 1')
hold on
plot(t,GoodSet1r(:,3)+3,'b')
plot(t,y1(:,3),'r')
plot(t,errorsy1(:,3)-3,'g')
hold off


% y2
errorNDXa = find(errorsy2(:,1) == -1);
errorNDXb = find(errorsy2(:,2) == -1);
errorNDXc = find(errorsy2(:,3) == -1);

terrorA = t(errorNDXa);
```

```
terrorB = t(errorNDXb);
terrorC = t(errorNDXc);

deltatimeA = diff(terrorA);
deltatimeB = diff(terrorB);
deltatimeC = diff(terrorC);

terrorA(1) = [];
terrorB(1) = [];
terrorC(1) = [];

figure(13)
plot(terrorA,deltatimeA,'b');
figure(14)
plot(terrorB,deltatimeB,'r');
figure(15)
plot(terrorC,deltatimeC,'g');

figure(40)
title('A phase comparison good set to 2 layer 50 fault NN Noise power 1')
hold on
plot(t,GoodSet2r(:,1)+3,'b')
plot(t,y2(:,1),'r')
plot(t,errorsy2(:,1)-3,'g')
hold off

figure(41)
title('B phase comparison good set to 2 layer 50 fault NN Noise power 1')
hold on
plot(t,GoodSet2r(:,2)+3,'b')
plot(t,y2(:,2),'r')
plot(t,errorsy2(:,2)-3,'g')
hold off

figure(42)
title('C phase comparison good set to 2 layer 50 fault NN Noise power 1')
hold on
plot(t,GoodSet2r(:,3)+3,'b')
plot(t,y2(:,3),'r')
plot(t,errorsy2(:,3)-3,'g')
hold off


% y3
errorNDXa = find(errorsy3(:,1) == -1);
errorNDXb = find(errorsy3(:,2) == -1);
errorNDXc = find(errorsy3(:,3) == -1);

terrorA = t(errorNDXa);
terrorB = t(errorNDXb);
terrorC = t(errorNDXc);

deltatimeA = diff(terrorA);
deltatimeB = diff(terrorB);
deltatimeC = diff(terrorC);
```

```
terrorA(1) = [];
terrorB(1) = [];
terrorC(1) = [];

figure(16)
plot(terrorA,deltatimeA,'b');
figure(17)
plot(terrorB,deltatimeB,'r');
figure(18)
plot(terrorC,deltatimeC,'g');

figure(43)
title('A phase comparison good set to 1 layer 20 fault NN Noise power 1')
hold on
plot(t,GoodSet3r(:,1)+3,'b')
plot(t,y3(:,1),'r')
plot(t,errorsy3(:,1)-3,'g')
hold off

figure(44)
title('B phase comparison good set to 1 layer 20 fault NN Noise power 1')
hold on
plot(t,GoodSet3r(:,2)+3,'b')
plot(t,y3(:,2),'r')
plot(t,errorsy3(:,2)-3,'g')
hold off

figure(45)
title('C phase comparison good set to 1 layer 20 fault NN Noise power 1')
hold on
plot(t,GoodSet3r(:,3)+3,'b')
plot(t,y3(:,3),'r')
plot(t,errorsy3(:,3)-3,'g')
hold off
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%



% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Noise power 2

% z1
errorNDXa = find(errorsz1(:,1) == -1);
errorNDXb = find(errorsz1(:,2) == -1);
errorNDXc = find(errorsz1(:,3) == -1);

terrorA = t(errorNDXa);
terrorB = t(errorNDXb);
terrorC = t(errorNDXc);

deltatimeA = diff(terrorA);
deltatimeB = diff(terrorB);
deltatimeC = diff(terrorC);

terrorA(1) = [];
terrorB(1) = [];
```

```
terrorC(1) = [];

figure(19)
plot(terrorA,deltatimeA,'b');
figure(20)
plot(terrorB,deltatimeB,'r');
figure(21)
plot(terrorC,deltatimeC,'g');

figure(46)
title('A phase comparison good set to 1 layer 50 fault NN Noise power 2')
hold on
plot(t,GoodSet1r(:,1)+3,'b')
plot(t,z1(:,1),'r')
plot(t,errorsz1(:,1)-3,'g')
hold off

figure(47)
title('B phase comparison good set to 1 layer 50 fault NN Noise power 2')
hold on
plot(t,GoodSet1r(:,2)+3,'b')
plot(t,z1(:,2),'r')
plot(t,errorsz1(:,2)-3,'g')
hold off

figure(48)
title('C phase comparison good set to 1 layer 50 fault NN Noise power 2')
hold on
plot(t,GoodSet1r(:,3)+3,'b')
plot(t,z1(:,3),'r')
plot(t,errorsz1(:,3)-3,'g')
hold off

% z2
errorNDXa = find(errorsz2(:,1) == -1);
errorNDXb = find(errorsz2(:,2) == -1);
errorNDXc = find(errorsz2(:,3) == -1);

terrorA = t(errorNDXa);
terrorB = t(errorNDXb);
terrorC = t(errorNDXc);

deltatimeA = diff(terrorA);
deltatimeB = diff(terrorB);
deltatimeC = diff(terrorC);

terrorA(1) = [];
terrorB(1) = [];
terrorC(1) = [];

figure(22)
plot(terrorA,deltatimeA,'b');
figure(23)
plot(terrorB,deltatimeB,'r');
figure(24)
plot(terrorC,deltatimeC,'g');
```

```
figure(49)
title('A phase comparison good set to 2 layer 50 fault NN Noise power 2')
hold on
plot(t,GoodSet2r(:,1)+3,'b')
plot(t,z2(:,1),'r')
plot(t,errorsz2(:,1)-3,'g')
hold off

figure(50)
title('B phase comparison good set to 2 layer 50 fault NN Noise power 2')
hold on
plot(t,GoodSet2r(:,2)+3,'b')
plot(t,z2(:,2),'r')
plot(t,errorsz2(:,2)-3,'g')
hold off

figure(51)
title('C phase comparison good set to 2 layer 50 fault NN Noise power 2')
hold on
plot(t,GoodSet2r(:,3)+3,'b')
plot(t,z2(:,3),'r')
plot(t,errorsz2(:,3)-3,'g')
hold off

% z3
errorNDXa = find(errorsz3(:,1) == -1);
errorNDXb = find(errorsz3(:,2) == -1);
errorNDXc = find(errorsz3(:,3) == -1);

terrorA = t(errorNDXa);
terrorB = t(errorNDXb);
terrorC = t(errorNDXc);

deltatimeA = diff(terrorA);
deltatimeB = diff(terrorB);
deltatimeC = diff(terrorC);

terrorA(1) = [];
terrorB(1) = [];
terrorC(1) = [];

figure(25)
plot(terrorA,deltatimeA,'b');
figure(26)
plot(terrorB,deltatimeB,'r');
figure(27)
plot(terrorC,deltatimeC,'g');

figure(52)
title('A phase comparison good set to 1 layer 20 fault NN Noise power 2')
hold on
plot(t,GoodSet3r(:,1)+3,'b')
plot(t,z3(:,1),'r')
plot(t,errorsz3(:,1)-3,'g')
hold off
```

```
figure(53)
title('B phase comparison good set to 1 layer 20 fault NN Noise power 2')
hold on
plot(t,GoodSet3r(:,2)+3,'b')
plot(t,z3(:,2),'r')
plot(t,errorsz3(:,2)-3,'g')
hold off

figure(54)
title('C phase comparison good set to 1 layer 20 fault NN Noise power 2')
hold on
plot(t,GoodSet3r(:,3)+3,'b')
plot(t,z3(:,3),'r')
plot(t,errorsz3(:,3)-3,'g')
hold off
```

This testing script generated figures 46-50.

```
figure(1)
plot(NoiseScopehalf.time,NoiseScopehalf.signals.values)
title('30 Seconds of Noise for Noise Power 1/2')
ylabel('Volts')
xlabel('Time (seconds)')
figure(2)
plot(NoiseScopeone.time,NoiseScopeone.signals.values)
title('30 Seconds of Noise for Noise Power 1')
ylabel('Volts')
xlabel('Time (seconds)')
figure(3)
plot(NoiseScopetwo.time,NoiseScopetwo.signals.values)
title('30 Seconds of Noise for Noise Power 2')
ylabel('Volts')
xlabel('Time (seconds)')

figure(4)
hold on
plot(phaseswithnoise.time,phaseswithnoise.signals.values(:,1),'b')
plot(phaseswithnoise.time,phaseswithnoise.signals.values(:,2),'r')
plot(phaseswithnoise.time,phaseswithnoise.signals.values(:,3),'g')
hold off
title('Three Phase Signal With Noise During an A fault')
ylabel('Volts')
xlabel('Time (seconds)')

figure(5)
hold on
plot(phaseswithnonoise.time,phaseswithnonoise.signals.values(:,1),'b')
plot(phaseswithnonoise.time,phaseswithnonoise.signals.values(:,2),'r')
plot(phaseswithnonoise.time,phaseswithnonoise.signals.values(:,3),'g')
hold off
title('Three Phase Signal Without Noise During an A fault')
ylabel('Volts')
xlabel('Time (seconds)')
```

This script is used to create the input and output files for the Neural Networks. By changing the range of the for loop variable NDX, the user can accommodate different data sizes. The largest value of NDX should be the number of faults of each fault condition being generated. This script also generates figures 11-17 and 19-32, by capturing the 11th fault and plotting before and after wavelet decomposition.

```
in = fopen('nnInputnew.txt','w');        % input to NN
out1 = fopen('nnTargetsnew1.txt','w');  % expected Output from NN
out2 = fopen('nnTargetsnew2.txt','w');
out3 = fopen('nnTargetsnew3.txt','w');

% grabbing fault content from simulink model for each fault type
AF = afault.signals.values;
BF = bfault.signals.values;
CF = cfault.signals.values;
ABF = abfault.signals.values;
BCF = bcfault.signals.values;
ACF = acfault.signals.values;
NF = nofault.signals.values;

% choosing indices to capture information for power calculations

I1 = 1034;
I2 = 1260;

xl = 'Time';
yl = 'Amplitude';

for NDX = 1:50;
    N1 = (NDX-1)*16384+1;
    N2 = NDX*16384;

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Process Fault on Phase A

    AFA = AF(N1: N2,1);
    AFB = AF(N1: N2,2);
    AFC = AF(N1: N2,3);

    % Extract Detail level 4 from wavelet
    [CAFA, LAFA] = wavedec(AFA, 4, 'db4');
    [CAFB, LAFB] = wavedec(AFB, 4, 'db4');
    [CAFC, LAFC] = wavedec(AFC, 4, 'db4');

    % Capture the 11th fault for analysis
    if (NDX == 11)
        figure(NDX+25)
        hold on
        plot(AFA(4800:7500))
        plot(AFB(4800:7500), 'r')
        plot(AFC(4800:7500), 'g')
        hold off
        title('A Phase-to-Ground Fault','Fontsize', 24)
        xlabel(xl,'Fontsize', 24)
```

```
        ylabel(yl,'Fontsize', 24)

        interimA = [CAFA(300:450); CAFA(1024+300:1024+450)];
        interimB = [CAFB(300:450); CAFB(1024+300:1024+450)];
        interimC = [CAFC(300:450); CAFC(1024+300:1024+450)];

        figure(NDX+26)
        hold on
        plot(interimA)
        plot(interimB, 'r')
        plot(interimC, 'g')
        hold off
        title('Approximation Level 4 and Detail Level 4 of a Phase A
Fault','Fontsize', 24)
        xlabel(xl,'Fontsize', 24)
        ylabel(yl,'Fontsize', 24)

        figure(NDX+27)
        hold on
        plot(CAFA(1024+300:1024+450))
        plot(CAFB(1024+300:1024+450),'r')
        plot(CAFC(1024+300:1024+450),'g')
        hold off
        title('Detail Level 4 of a Phase A Fault','Fontsize', 24)
        xlabel(xl,'Fontsize', 24)
        ylabel(yl,'Fontsize', 24)
    end

    % Calculate power
    powerAFA = sum(CAFA(I1:I2).^2);
    powerAFB = sum(CAFB(I1:I2).^2);
    powerAFC = sum(CAFC(I1:I2).^2);
    pAfault = [powerAFA powerAFB powerAFC];

    fprintf(in,'%f %f %f \n',pAfault);
    fprintf(out1,'1 \n');
    fprintf(out2,'1 \n');
    fprintf(out3,'1 -1 -1 \n');

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Process Fault on Phase B

    BFA = BF(N1: N2,1);
    BFB = BF(N1: N2,2);
    BFC = BF(N1: N2,3);

    % Extract Detail level 4 from wavelet
    [CBFA, LBFA] = wavedec(BFA, 4, 'db4');
    [CBFB, LBFB] = wavedec(BFB, 4, 'db4');
    [CBFC, LBFC] = wavedec(BFC, 4, 'db4');

    % Capture the 11th fault for analysis
    if (NDX == 11)
        figure(NDX+28)
        hold on
        plot(BFA(4800:7500))
```

```
        plot(BFB(4800:7500), 'r')
        plot(BFC(4800:7500), 'g')
        hold off
        title('B Phase-to-Ground Fault','Fontsize', 24)
        xlabel(xl,'Fontsize', 24)
        ylabel(yl,'Fontsize', 24)

        interimA = [CBFA(300:450); CBFA(1024+300:1024+450)];
        interimB = [CBFB(300:450); CBFB(1024+300:1024+450)];
        interimC = [CBFC(300:450); CBFC(1024+300:1024+450)];

        figure(NDX+29)
        hold on
        plot(interimA)
        plot(interimB, 'r')
        plot(interimC, 'g')
        hold off
        title('Approximation Level 4 and Detail Level 4 of a Phase B
Fault','Fontsize', 24)
        xlabel(xl,'Fontsize', 24)
        ylabel(yl,'Fontsize', 24)

        figure(NDX+30)
        hold on
        plot(CBFA(1024+300:1024+450))
        plot(CBFB(1024+300:1024+450),'r')
        plot(CBFC(1024+300:1024+450),'g')
        hold off
        title('Detail Level 4 of a Phase B Fault','Fontsize', 24)
        xlabel(xl,'Fontsize', 24)
        ylabel(yl,'Fontsize', 24)
    end

    % Calculate power
    powerBFA = sum(CBFA(I1:I2).^2);
    powerBFB = sum(CBFB(I1:I2).^2);
    powerBFC = sum(CBFC(I1:I2).^2);
    pBfault = [powerBFA powerBFB powerBFC];

    fprintf(in,'%f %f %f \n',pBfault);
    fprintf(out1,'2 \n');
    fprintf(out2,'2 \n');
    fprintf(out3,'-1 1 -1 \n');

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Process Fault on Phase C

    CFA = CF(N1: N2,1);
    CFB = CF(N1: N2,2);
    CFC = CF(N1: N2,3);

    % Extract Detail level 4 from wavelet
    [CCFA, LCFA] = wavedec(CFA, 4, 'db4');
    [CCFB, LCFB] = wavedec(CFB, 4, 'db4');
    [CCFC, LCFC] = wavedec(CFC, 4, 'db4');
```

```
    % Capture the 11th fault for analysis
    if (NDX == 11)
        figure(NDX+31)
        hold on
        plot(CFA(4800:7500))
        plot(CFB(4800:7500), 'r')
        plot(CFC(4800:7500), 'g')
        hold off
        title('C Phase-to-Ground Fault','Fontsize', 24)
        xlabel(xl,'Fontsize', 24)
        ylabel(yl,'Fontsize', 24)

        interimA = [CCFA(300:450); CCFA(1024+300:1024+450)];
        interimB = [CCFB(300:450); CCFB(1024+300:1024+450)];
        interimC = [CCFC(300:450); CCFC(1024+300:1024+450)];

        figure(NDX+32)
        hold on
        plot(interimA)
        plot(interimB, 'r')
        plot(interimC, 'g')
        hold off
        title('Approximation Level 4 and Detail Level 4 of a Phase C
Fault','Fontsize', 24)
        xlabel(xl,'Fontsize', 24)
        ylabel(yl,'Fontsize', 24)

        figure(NDX+33)
        hold on
        plot(CCFA(1024+300:1024+450))
        plot(CCFB(1024+300:1024+450),'r')
        plot(CCFC(1024+300:1024+450),'g')
        hold off
        title('Detail Level 4 of a Phase C Fault','Fontsize', 24)
        xlabel(xl,'Fontsize', 24)
        ylabel(yl,'Fontsize', 24)
    end

    % Calculate power
    powerCFA = sum(CCFA(I1:I2).^2);
    powerCFB = sum(CCFB(I1:I2).^2);
    powerCFC = sum(CCFC(I1:I2).^2);
    pCfault = [powerCFA powerCFB powerCFC];

    fprintf(in,'%f %f %f \n',pCfault);
    fprintf(out1,'3 \n');
    fprintf(out2,'3 \n');
    fprintf(out3,'-1 -1 1 \n');

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Process Fault on Phases A & B

    ABFA = ABF(N1: N2,1);
    ABFB = ABF(N1: N2,2);
    ABFC = ABF(N1: N2,3);

    % Extract Detail level 4 from wavelet
```

```
    [CABFA, LABFA] = wavedec(ABFA, 4, 'db4');
    [CABFB, LABFB] = wavedec(ABFB, 4, 'db4');
    [CABFC, LABFC] = wavedec(ABFC, 4, 'db4');

    % Capture the 11th fault for analysis
    if (NDX == 11)
        figure(NDX+34)
        hold on
        plot(ABFA(4800:7500))
        plot(ABFB(4800:7500), 'r')
        plot(ABFC(4800:7500), 'g')
        hold off
        title('AB Phase Fault','Fontsize', 24)
        xlabel(xl,'Fontsize', 24)
        ylabel(yl,'Fontsize', 24)

        interimA = [CABFA(300:450); CABFA(1024+300:1024+450)];
        interimB = [CABFB(300:450); CABFB(1024+300:1024+450)];
        interimC = [CABFC(300:450); CABFC(1024+300:1024+450)];

        figure(NDX+35)
        hold on
        plot(interimA)
        plot(interimB, 'r')
        plot(interimC, 'g')
        hold off
        title('Approximation Level 4 and Detail Level 4 of a Phase AB
Fault','Fontsize', 24)
        xlabel(xl,'Fontsize', 24)
        ylabel(yl,'Fontsize', 24)

        figure(NDX+36)
        hold on
        plot(CABFA(1024+300:1024+450))
        plot(CABFB(1024+300:1024+450),'r')
        plot(CABFC(1024+300:1024+450),'g')
        hold off
        title('Detail Level 4 of a Phase AB Fault','Fontsize', 24)
        xlabel(xl,'Fontsize', 24)
        ylabel(yl,'Fontsize', 24)
    end

    % Calculate power
    powerABFA = sum(CABFA(I1:I2).^2);
    powerABFB = sum(CABFB(I1:I2).^2);
    powerABFC = sum(CABFC(I1:I2).^2);
    pABfault = [powerABFA powerABFB powerABFC];

    fprintf(in,'%f %f %f \n',pABfault);
    fprintf(out1,'4 \n');
    fprintf(out2,'-1 \n');
    fprintf(out3,'1 1 -1 \n');

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Process Fault on Phases B & C

    BCFA = BCF(N1: N2,1);
```

```
    BCFB = BCF(N1: N2,2);
    BCFC = BCF(N1: N2,3);

    % Extract Detail level 4 from wavelet
    [CBCFA, LBCFA] = wavedec(BCFA, 4, 'db4');
    [CBCFB, LBCFB] = wavedec(BCFB, 4, 'db4');
    [CBCFC, LBCFC] = wavedec(BCFC, 4, 'db4');

    % Capture the 11th fault for analysis
    if (NDX == 11)
        figure(NDX+37)
        hold on
        plot(BCFA(4800:7500))
        plot(BCFB(4800:7500), 'r')
        plot(BCFC(4800:7500), 'g')
        hold off
        title('BC Phase Fault','Fontsize', 24)
        xlabel(xl,'Fontsize', 24)
        ylabel(yl,'Fontsize', 24)

        interimA = [CBCFA(300:450); CBCFA(1024+300:1024+450)];
        interimB = [CBCFB(300:450); CBCFB(1024+300:1024+450)];
        interimC = [CBCFC(300:450); CBCFC(1024+300:1024+450)];

        figure(NDX+38)
        hold on
        plot(interimA)
        plot(interimB, 'r')
        plot(interimC, 'g')
        hold off
        title('Approximation Level 4 and Detail Level 4 of a Phase BC
Fault','Fontsize', 24)
        xlabel(xl,'Fontsize', 24)
        ylabel(yl,'Fontsize', 24)

        figure(NDX+39)
        hold on
        plot(CBCFA(1024+300:1024+450))
        plot(CBCFB(1024+300:1024+450),'r')
        plot(CBCFC(1024+300:1024+450),'g')
        hold off
        title('Detail Level 4 of a Phase BC Fault','Fontsize', 24)
        xlabel(xl,'Fontsize', 24)
        ylabel(yl,'Fontsize', 24)
    end

    % Calculate power
    powerBCFA = sum(CBCFA(I1:I2).^2);
    powerBCFB = sum(CBCFB(I1:I2).^2);
    powerBCFC = sum(CBCFC(I1:I2).^2);
    pBCfault = [powerBCFA powerBCFB powerBCFC];

    fprintf(in,'%f %f %f \n',pBCfault);
    fprintf(out1,'5 \n');
    fprintf(out2,'-2 \n');
    fprintf(out3,'-1 1 1 \n');
```

```
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Process Fault on Phases A & C

    ACFA = ACF(N1: N2,1);
    ACFB = ACF(N1: N2,2);
    ACFC = ACF(N1: N2,3);

    % Extract Detail level 4 from wavelet
    [CACFA, LACFA] = wavedec(ACFA, 4, 'db4');
    [CACFB, LACFB] = wavedec(ACFB, 4, 'db4');
    [CACFC, LACFC] = wavedec(ACFC, 4, 'db4');

    % Capture the 11th fault for analysis
    if (NDX == 11)
        figure(NDX+40)
        hold on
        plot(ACFA(4800:7500))
        plot(ACFB(4800:7500), 'r')
        plot(ACFC(4800:7500), 'g')
        hold off
        title('AC Phase Fault','Fontsize', 24)
        xlabel(xl,'Fontsize', 24)
        ylabel(yl,'Fontsize', 24)

        interimA = [CACFA(300:450); CACFA(1024+300:1024+450)];
        interimB = [CACFB(300:450); CACFB(1024+300:1024+450)];
        interimC = [CACFC(300:450); CACFC(1024+300:1024+450)];

        figure(NDX+41)
        hold on
        plot(interimA)
        plot(interimB, 'r')
        plot(interimC, 'g')
        hold off
        title('Approximation Level 4 and Detail Level 4 of a Phase AC
Fault','Fontsize', 24)
        xlabel(xl,'Fontsize', 24)
        ylabel(yl,'Fontsize', 24)

        figure(NDX+42)
        hold on
        plot(CACFA(1024+300:1024+450))
        plot(CACFB(1024+300:1024+450),'r')
        plot(CACFC(1024+300:1024+450),'g')
        hold off
        title('Detail Level 4 of a Phase AC Fault','Fontsize', 24)
        xlabel(xl,'Fontsize', 24)
        ylabel(yl,'Fontsize', 24)
    end

    % Calculate power
    powerACFA = sum(CACFA(I1:I2).^2);
    powerACFB = sum(CACFB(I1:I2).^2);
    powerACFC = sum(CACFC(I1:I2).^2);
    pACfault = [powerACFA powerACFB powerACFC];

    fprintf(in,'%f %f %f \n',pACfault);
```

```
    fprintf(out1,'6 \n');
    fprintf(out2,'-3 \n');
    fprintf(out3,'1 -1 1 \n');

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Process NO Fault

    NOFA = NF(N1: N2,1);
    NOFB = NF(N1: N2,2);
    NOFC = NF(N1: N2,3);

    % Extract Detail level 4 from wavelet
    [CNOFA, LNOFA] = wavedec(NOFA, 4, 'db4');
    [CNOFB, LNOFB] = wavedec(NOFB, 4, 'db4');
    [CNOFC, LNOFC] = wavedec(NOFC, 4, 'db4');

    % Capture the 11th fault for analysis
    if (NDX == 11)
        figure(NDX+43)
        hold on
        plot(NOFA(4800:7500))
        plot(NOFB(4800:7500), 'r')
        plot(NOFC(4800:7500), 'g')
        hold off
        title('No Phase Fault','Fontsize', 24)
        xlabel(xl,'Fontsize', 24)
        ylabel(yl,'Fontsize', 24)

        interimA = [CNOFA(300:450); CNOFA(1024+300:1024+450)];
        interimB = [CNOFB(300:450); CNOFB(1024+300:1024+450)];
        interimC = [CNOFC(300:450); CNOFC(1024+300:1024+450)];

        figure(NDX+44)
        hold on
        plot(interimA)
        plot(interimB, 'r')
        plot(interimC, 'g')
        hold off
        title('Approximation Level 4 and Detail Level 4 of a NO
Fault','Fontsize', 24)
        xlabel(xl,'Fontsize', 24)
        ylabel(yl,'Fontsize', 24)

        figure(NDX+45)
        hold on
        plot(CNOFA(1024+300:1024+450))
        plot(CNOFB(1024+300:1024+450),'r')
        plot(CNOFC(1024+300:1024+450),'g')
        hold off
        title('Detail Level 4 of a NO Fault','Fontsize', 24)
        xlabel(xl,'Fontsize', 24)
        ylabel(yl,'Fontsize', 24)
    end

    % Calculate power
    powerNOFA = sum(CNOFA(I1:I2).^2);
    powerNOFB = sum(CNOFB(I1:I2).^2);
```

```
    powerNOFC = sum(CNOFC(I1:I2).^2);
    pNOfault = [powerNOFA powerNOFB powerNOFC];

    fprintf(in,'%f %f %f \n',pNOfault);
    fprintf(out1,'0 \n');
    fprintf(out2,'0 \n');
    fprintf(out3,'-1 -1 -1 \n');

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
end;

fclose(in);
fclose(out1);
fclose(out2);
fclose(out3);
```