NEURAL NETWORK FAULT DIAGNOSIS SYSTEM FOR A DIESEL-ELECTRIC LOCOMOTIVE'S CLOSED LOOP EXCITATION CONTROL SYSTEM

Master's Dissertation

for the qualification towards

Master of Engineering in Mechatronics

Department of Mechatronics

NMMU

"Submitted in fulfilment of the requirements for the Master of Engineering in Mechatronics at the Nelson Mandela Metropolitan University"

Compiler: Morne Barnard (s207073708)

April 2017

Supervisor: Prof. Theo van Niekerk

Declaration

Morne Barnard

14 Eendrag Street

Despatch, 6220

South Africa

I, Morne Barnard, student number s207073708, hereby declare that this dissertation for the Master of Engineering in Mechatronics is my own work and that it has not previously been submitted for assessment or completion of any postgraduate qualification to another university or for another qualification.

_____ _____

Morne Barnard Date:

Abstract

In closed loop control systems fault isolation is extremely difficult due to the fact that if feedbacks are corrupted or actuators can't produce a desired output, a system reacts due to an increase in error between the measured variable and the set input variable, which can cause oscillations.

The goal of this project is to develop a fault detection and isolation system for the isolation of faults, which cause oscillatory conditions on a GE Diesel-Electric Locomotive's excitation control system. The proposed system will illustrate the use of artificial neural networks as a replacement to classical analytical models. The artificial neural network model's design will be based on model-based dedicated observer theory to isolate sensor, as well as component faults, where observer theory will be utilised to effectively select input-output data configurations for detection of sensor and component faults causing oscillations.

Owing to the nature of the locomotive's data acquisition abilities, the model-based observer design will utilise historical data to design an effective model of the system which will be used to perform offline sampled fault detection. This method is proposed as an alternative to trend checking, data mining, etc. Faults are thus detected through the use of an offline model-based dedicated observer residual generator.

With the use of a neural network a number of parameters affect the accuracy of the network where the primary source of ensuring an accurate model is training. The project highlights and experiments with these parameters to ensure an accurate model is trained with the use of the gradient descent training algorithm. The parameters which are considered are learning rate, hidden layer neurons, momentum and data preparation.

The project will also provide a literature review on residual evaluation techniques used in practice and describe and evaluate the proposed method to perform residual evaluation for this specific application. The proposed method for residual evaluation was based on two principles, namely the moving average, as well as the simple thresholding techniques.

The developed FDI system's performance was measured against known faults and produced 100% accuracy for the detection and isolation of sensor and components causing oscillatory conditions on the locomotive's excitation system.

Table of Contents

Abbreviations

AC – Alternator Current

ANN – Artificial Neural Network

BN – Biological Neuron

BSS – BrightStar (Locomotive's Propulsion Control System)

D34 – Diesel 34 Class locomotive

DAMADICS - Development of Applications and Methods for Actuator Diagnosis in Industrial Control Systems

DC – Direct Current

D-E – Diesel-Electric

EXACT – Exciter Armature Current Transducer

EXFM – Exciter Field Module

FDI – Fault Detection and Isolation

FFNN – Feedforward Neural Network

GD – Gradient Descent (Neural Network Training Algorithm)

GE – General Electric (OEM for D34 class Locomotives used by Transnet)

GUI – Graphical User Interface

KBNNFD – Knowledge Based Neural Network Fault detection

LCP – Load Control Potentiometer

MAE – Mean Absolute Error

MSE – Mean Squared Error

NN – Neural Network

OEM – Original Equipment Manufacturer

RMS – Root Mean Square

RPM – Revolutions per Minute

SCADA – Supervisory Control and Data Acquisition

List of Figures

List of Tables

Chapter 1: Introduction

An unappealing characteristic of real world control systems is the fact that they are vulnerable to faults, malfunctions and unexpected modes of operations due to component and/or sensor failures. These failures affect operations in industrial plants negatively in terms of production or a plant's operating time.

In any production line, service centre or really any business, time plays a very important role. Time can be considered one of the key factor which determines delivery and quality, as well as profitability. One of the key factors affecting the operating time of machinery is unscheduled breakdowns which in turn requires unscheduled maintenance. Within the maintenance environment, "fault isolation" time, which can be defined as the time taken to isolate a faulty component, can be considered one of the most important factors affecting production or a plant's operating time.

Owing to the increasing demands on the reduction of this time, research in the field of fault detection and isolation has received increasing interest over the years, especially in automated environments, which has led to a significant improvement in the process of fault detection and isolation, with a large reduction in the need for limit checking or trend analysis, which requires expert knowledge of systems, in order to perform fault detection and isolation [1, pp.22-23].

In order to avoid the heavy economic losses involved in halted production, due to the replacement of elements, parts and fault isolation, literature on methods to perform fault diagnosis are mostly aimed at performing FDI with the use of model-based techniques, which are created with the use of analytical approaches. The problem with the analytical approach is that most industrial systems cannot easily be modelled due to their sheer size, complexity, unavailability of component data of the design, measurements being corrupted by noise and unreliable sensors within the control system. Owing to this, a number of researchers have focussed their research on the use of neural networks to produce models of industrial processes. This is due to the fact that neural networks have the ability to filter out noise and disturbances, thus providing a stable and highly sensitive model of an industrial system without the use of a mathematical model.

This chapter presents a breakdown on the proposed concept of an FDI system to be used on a GE Diesel-Electric Locomotive's excitation control system with the use of neural networks. The first section focusses on the importance of the need for an FDI system on the locomotive's excitation system. The second section provides the reader with the overall objective, which the research aims to achieve. Section 3 highlights the Hypothesis. Section 4 provides the reader with the delimitation of the research to be done. The organization of the dissertation is then given in section 5.

1.1 Rationale and Motivation

Transnet Engineering's Maintenance department currently spends up to 3 hours to isolate oscillatory faults occurring on the D34 class Diesel-Electric Locomotive's excitation system. The reason for the large amount of time spend, is due to the fact that the current control

system, does not perform fault detection and isolation for oscillatory control loop faults. Owing to this, the method followed to isolate faults is by elimination of possibilities.

This method eliminates the possible faulty components, one at a time on the locomotive, by replacing each component which has an impact on the excitation control loop, with a new one. This method is effective but takes up a lot of time, due to the location, as well as the structure of the components.

With time being one of the most important aspects within the maintenance environment and especially in the field of fault isolation, it is of great importance to research a method, which facilitates faster fault isolation on the excitation system of the locomotive.

As previously mentioned, time is of great concern, but is not the only factor affected by this lengthy process. Figure 1.1 below shows that an increase in time, affects the overall costs due to an increase in the labour hours. Quality is not affected negatively in this case, due to the fact that the locomotive still leaves the depot in an excellent working condition.

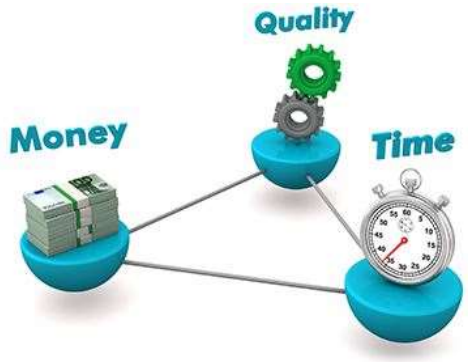


Figure 1.1: Project Management Triangle [38]

Another important factor, which is not listed in Figure 1.1, is safety. Safety of Transnet's most valuable assets, namely the workforce, could have been put at the top of the list as the replacement of the sensors used within the excitation system is labour intensive and located in confined spaces, increasing the risk of injuries.

The research and design of an FDI system would not only be able to significantly reduce fault isolation times and labor costs, but would also provide fault isolation of locomotives not in a maintenance depot. This will reduce the out-of-service time of failed locomotives, hence improving the availability and reliability of the D34 class GE fleet.

1.2 Overall Objective

The overall aim of the research is to design and develop a software-based fault detection and isolation system, to be used on the D34 Class Diesel-Electric locomotive's excitation control system for the detection and isolation of oscillatory faults.

1.2.1 Sub-Objective

The following objectives were accordingly specified for this project:

- Research current Model-Based FDI methods

  - Research current Model-Based Residual Generation methods
  - Gain an understanding of the use of Neural networks in the plant/system model design
  - Research Model-Based Residual Evaluation methods also known as the decision making stage
- Design Neural Network models of the GE Diesel-Electric Locomotive's Excitation System, which will be used as a residual generator to isolate component and sensor faults
  - Research various performance and training parameters of a Neural Network
  - Analyse the locomotive's excitation control system (Neural Network input-output selection)
  - Experiment with different parameters to develop the optimal model of the plant for sensor and component fault detection
- Design a residual evaluation technique to isolate component and sensor faults
  - Provide a literature review on different residual evaluation techniques used in practice
  - Evaluate residual evaluation technique performance
- Test the FDI system's ability to isolate faults.

1.3 Hypothesis

A software-based fault detection and isolation system will be developed to improve the productivity lost due to the time taken to perform fault finding and diagnosis on a GE D34 Class Diesel-Electric Locomotive's Excitation control system. The software will utilise intelligent systems to improve the fault detection and isolation time for dynamic faults on the excitation system of the locomotive.

1.4 Delimitation of the Research

The following limitations to the dissertation should be noted:

- The FDI system will be limited to GE D34 class Diesel-Electric Locomotives operating with Brightstar constant horse power excitation control systems
- The OEM Brightstar control software will remain standard with no alterations done to it
- The FDI software will be an offline/data-model-based fault detection system
- The FDI system will be designed to perform only fault detection and isolation on oscillatory faults occurring on the closed loop excitation control system of the locomotive
- No additional Hardware will be added to the current Hardware-Architecture
- Locomotive Notch Control will not be done manually.

1.5 Organization of the Dissertation

This section provides an overview of the organization of the dissertation. The dissertation is divided into 7 chapters, namely:

Chapter 1 – This chapter states the Rationale and Motivation, Overall Objective, Hypothesis and Delimitation of the research to be done.

Chapter 2 – This chapter describes the proposed FDI system, literature reviews on up to date fault detection and isolation techniques and model-based fault detection, as well as different FDI and model-based applications. A brief review on the application of Artificial Neural Networks in FDI systems is also presented. In conclusion, this chapter will provide the reader with an idea of the proposed system and link relevant literature to the proposed system.

Chapter 3 – The chapter starts with a brief history of artificial neural networks which is then followed by a review on general applications of artificial neural networks; literature review on real world applications of artificial neural networks in the field of FDI systems; analysis on feedforward neural networks and a literature review on the use of feedforward neural networks in FDI systems. Artificial neural network training is then further discussed with the focus being on supervised learning and the gradient descent learning algorithm. Different activation functions which are associated with their use in the back propagation training algorithm are also reviewed.

Chapter 4 – This chapter sets the theoretical basis for the effective development of an optimum neural network model of the excitation control system of the locomotive, to be used as per the proposed FDI system. The theoretical basis covers the following: review on relevant literature on the implementation of the gradient descent training algorithm in FDI systems; overview on the performance measures of a neural network to consider in order to successfully model the locomotive's excitation system; mathematical analysis on the locomotive's excitation control system to determine the input-output data selection; literature review on data preparations, learning parameters, neural network configurations and training data.

Chapter 5 – In this chapter the development of 7 neural networks is presented, where 1 neural network is developed as a dedicated observer for component fault detection and a bank of six neural networks is developed for the modelling of dedicated observers for the detection of sensor failures. The development is done with the aid of the theoretical research done in Chapter 4, where the following training performance parameters were varied and experimented with in order to create accurate models; hence residual generators; Number of hidden layered neurons, scaling and stopping conditions. The chapter documents the results of the training done on each neural network and discusses the findings. The chapter ends with a detailed breakdown on the developed residual evaluation technique used in the FDI application where the two main components for fault isolation, used in this application, namely, moving average filter and simple thresholding, are discussed in detail.

Chapter 6 – The chapter will highlight the overall designed FDI system and performance thereof, in terms of its ability to isolate oscillatory faults occurring in the excitation system of the locomotive. Results on the performance of the FDI system to detect specific faults will be tabulated and discussed. The chapter will also illustrate the developed FDI Application software and will provide the reader with an insight into the operation thereof. An additional design for the isolation of total component failures will also be presented.

Chapter 7 – This chapter presents a conclusion to the research done in terms of the outputs achieved and knowledge gained in the field of fault detection and isolation systems. It also discusses possible future research to be done.

Chapter 2 – Review on Fault Detection and Isolation methods and Model-based fault detection and isolation

In this chapter the general idea of the proposed FDI system will be presented, followed by an in-depth literature study on different FDI methods and in particular model-based fault detection and isolation. The literature study will be used to direct the proposed idea and to find possible correlations between literature methods used and the proposed FDI system.

2.1 Proposed FDI System

This section highlights the proposed FDI system for fault detection and isolation on a Diesel-Electric locomotive's excitation control system. Firstly, an analysis on the locomotive's data acquisition abilities was done which indicated that only recorded data could be obtained from the locomotive, thus rendering online fault diagnosis impossible. It was also found that the OEM's controlling software was *copyright* protected and did not allow any online interface whatsoever. The recording function of the locomotive's software provided data on the nominal operation of the excitation control system. Owing to this, an offline approach could be used where the FDI data receiving function was done as indicated in Figure 2.1 below, where the following basic steps were followed:

- Connect Laptop to BSS control system via USB interface
- Operate the locomotive for 30 seconds whilst Hyperterminal records data plus creates a data file
- Decode data file and import into Matlab
- Create residuals in Matlab with the use of a neural network residual generator
- Perform residual evaluation using a residual evaluation technique
- Isolated fault will then be displayed in a window.

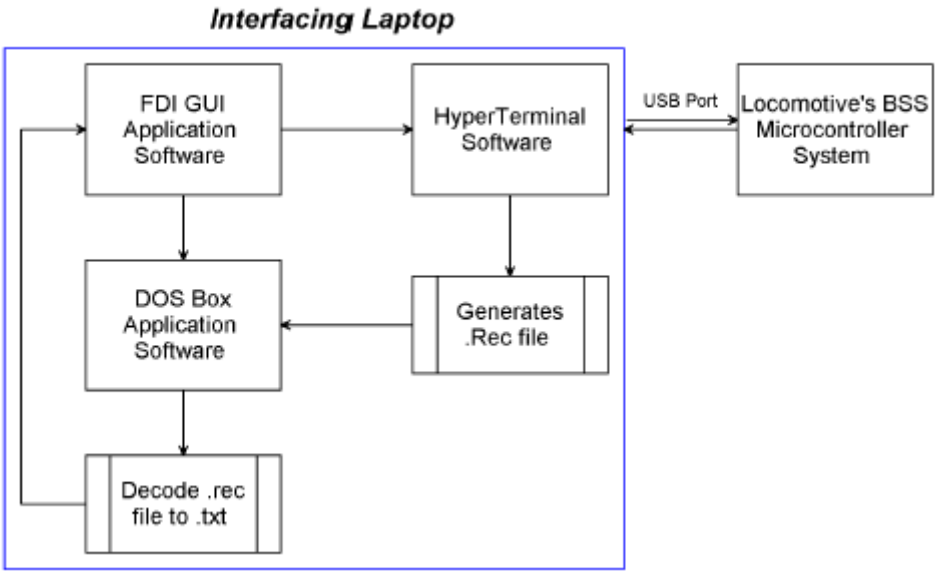


Figure 2.1: System Configuration

Figure 2.2 shows the overall intention of the proposed neural network FDI system which will be designed in Matlab.

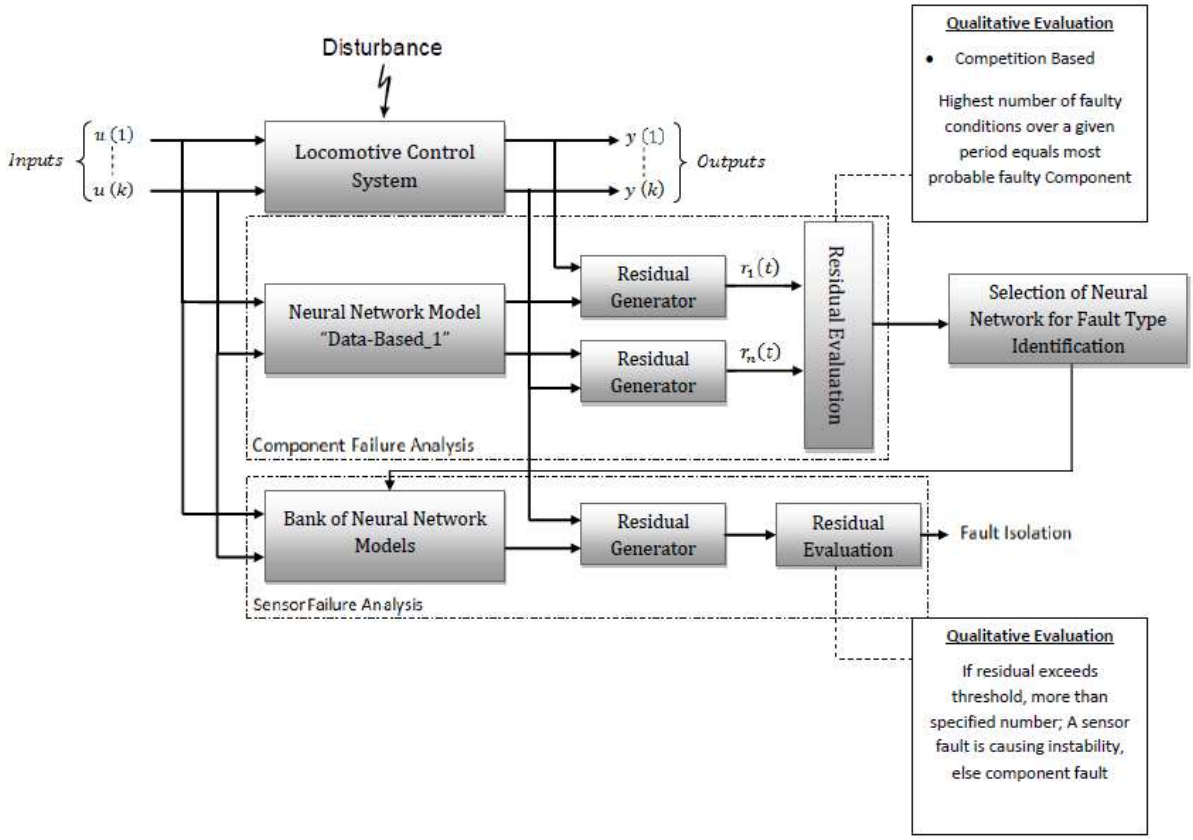


Figure 2.2: General Idea of the FDI System to be Developed

It should be noted that the proposed FDI system illustrated in Figure 2.2, will be based on nominal models running in parallel with the plant, where the first model will be used to detect and isolate component failures, with the second set of models being used for sensor fault isolation.

The input-output configuration of the plant as illustrated in Figure 2.2 will be in the form of a recording file, which will display the locomotive's sensor readings for a specified time, in predefined time steps. Figure 2.3 below shows an example of a decoded recording file. This file will then be imported into Matlab, where a simulation of the system will be run through the FDI process.

engNotchCmd	pwrNotchCmd	mainGenVolts	mainGenFldAmps	excFldAmps	loadControlPot	engineWaterTemp	hump
2	0	5	0	0.057566	71.914062	100	0.19541
2	0	5	0	0.042925	71.914062	100	0.19541
2	0	5	0	0.028283	71.914062	100	0.19541
2	0	5	0	0.072235	71.914062	100	0.19541
2	0	5	0	0.057566	71.71875	100	0.19541
2	0	5	0	0.072235	72.109375	100	0
2	0	5	0	0.042925	71.914062	100	0.19541
2	0	5	0	0.028283	71.914062	100	0.19541
2	0	5	0	0.057566	71.71875	100	0.19541
2	0	5	0	0.057566	71.71875	100	0.19541
2	0	5	0	0.013641	71.71875	100	0.19541
2	0	5	0	0.072235	71.914062	100	0.19541
6	0	5	0	0.072235	71.71875	100	0.19541
6	0	1	1	-0.000999	71.71875	100	0.19541
6	0	2	3	0.057566	71.523438	100	0.19541
6	1	2	5	0.057566	71.328125	100	0.19541
6	1	2	6	0.057566	71.71875	100	0.19541
6	1	3	7	0.057566	71.914062	100	0.19541
6	1	3	8	0.042925	73.476562	100	0.19541
6	1	3	8	0.057566	73.476562	100	0.19541

Figure 2.3: Decoded Recording File

As mentioned above, the first step will be to compare the model outputs with those of the system (which will be recorded and imported), to generate residuals. These residuals will then be evaluated in a residual evaluation process for each time step. Figure 2.3 above indicates the time steps. This process is indicated as component failure analysis in Figure 2.2. A count is made of how many times the residual for a specific reading exceeds a threshold value, where the highest number indicates the highest probability for the cause of the oscillation. With the possible cause isolated a neural network model is then selected in order to determine whether the fault is caused by a sensor failure. This process is illustrated as sensor failure analysis in Figure 2.2. The same evaluation process is followed as in the first step. The end result uses qualitative reasoning to determine whether the fault is a sensor or component failure.

In order to effectively develop the proposed system, more research was necessary in different FDI methods, as well as in the field of model-based fault detection and isolation. Thus the succeeding sections will provide a review on relevant theories, with regard to FDI systems.

2.2 Background on Fault Detection and Isolation Systems

Because of increased demands on reliability, availability and safety of technical plants, research on improving the supervision and monitoring abilities of systems to increase their fault detection abilities, has received much attention over the years. Owing to this, there exists a lot of research in the field of fault diagnostics, where Bagajewicz and Chmielewski [2, p.288] indicates that fault diagnosis methods can be categorised in three main groups, namely: qualitative, quantitative and process history-based methods (See Figure 2.4 below).

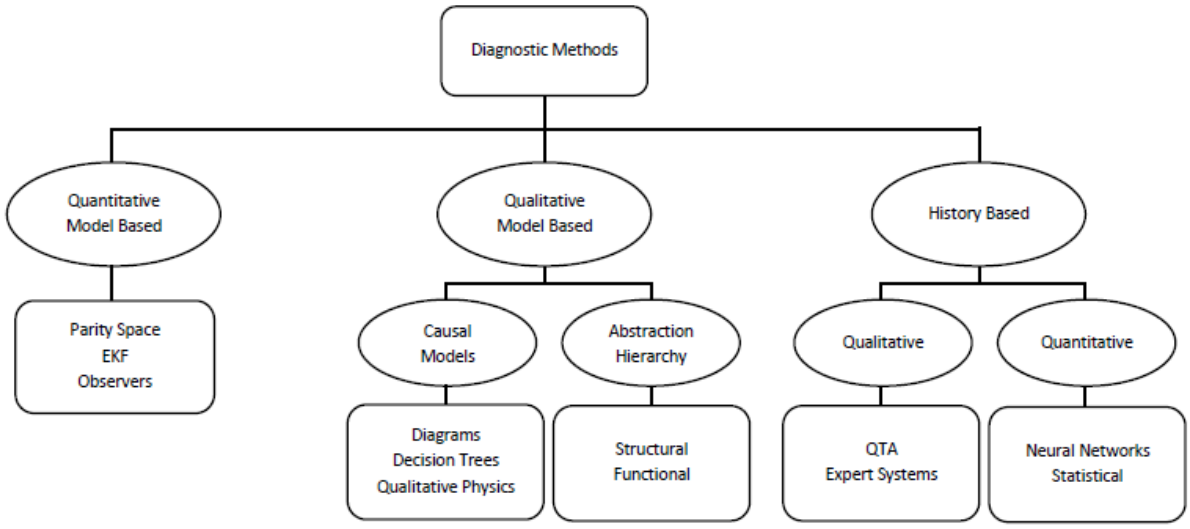


Figure 2.4: Diagnostic Methods

2.2.1 Qualitative Methods

These methods use symbolic reasoning to perform fault detection and isolation. The symbolic reasoning generally combines different kinds of knowledge with graph theory to analyse variables of a system. This method does not require an explicit model of the system and includes the following methods:

- Fault Trees: This method traces the fault back to possible causes, to identify the cause effect, but is not useful in identifying the cause [2, p.288]
- Rule-Based Approach: This approach utilises signed directed graphs, to determine the effect of a fault on a variable [2, p.288]
- Failure Propagation Networks: Bagajewicz and Chmielewski [2, p.289] describes this as follows: "*These are based on the notion that the effect of the fault propagates through equipment. Diagraphs with propagation times and failure probabilities are used.*"
- Knowledge-Based Approach: The knowledge-based approach makes use of available knowledge to either derive a qualitative description of the system in the form of a qualitative model or a rule-based representation.

2.2.2 Quantitative Methods

Quantitative Model-based methods require the use of an analytical model, where differences between plant and model behaviour are used to detect faults; thus the method is known as a *quantitative* Model-Based approach. Bagajewicz and Chmielewski [2, p.288] describes these methods as follows: "*These are quantitative models (usually linear) used for fault detection and isolation. They use dynamic models (Kalman filters, diagnostic observers, parity relations, parameters estimation, etc.) that help generate estimators of measured and unmeasured variables and parameters. Then, measurements and estimators are used to generate residuals, which are used for diagnostics (Gertler, 1988; Frank, 1990; and Patton, 1995) as well as the books by Himmelblau (1978), Patton et al. (1989), and Gertler (1998), among others*."

2.2.3 Process History-Based Methods

Process History-based methods require historical data of a process, which serves as prior knowledge of the system. This knowledge can be in the form of feature extraction which could be qualitative or quantitative. Tools used within this method include the following

- Artificial Neural Networks:
- Multivariate Statistical Methods: These methods rely on principle component analysis, inverse least squares and principle component regression, as well as partial least squares.

Frank, Garcia and Koppen-Seiliger [3] stated that the most powerful approaches are the ones which utilise a process model where quantitative, qualitative, knowledge-based, data-based models or a combination thereof are used. Data-based models can also be defined as a Model-based FDI where the historical data of a system is used to create a nominal model of a system to generate residuals from comparison with plant readings.

There are different approaches to model-based FDI systems, where system modelling is the key concept. Most of the earliest successful approaches developed over the past 40 years were with the use of analytical models, which incorporated the use of mathematical models. Where work done by Chen & Patton, 1999; Frank,1990; Gertler, 1998; Himmelblau, 1978; Isermann, 1984, 1977; Patton, Frank,& Clark, 2000; Willsky, 1976 had contributed majorly towards FDI systems with the use of analytical models [4].

Figure 2.5, illustrates the general scheme of process model-based fault-detection and diagnosis used by Isermann [4, p.72].

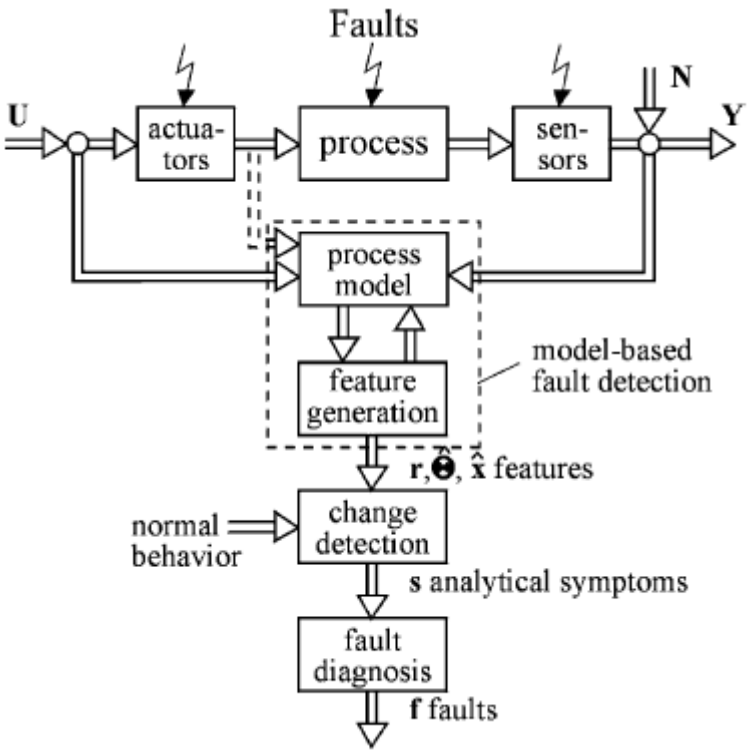
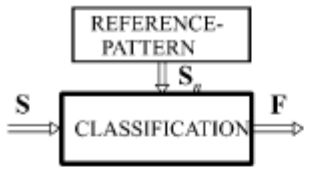
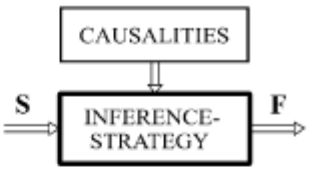
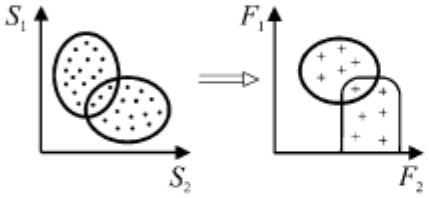
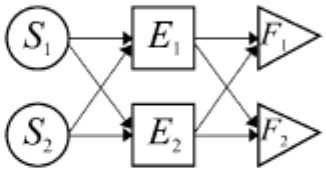
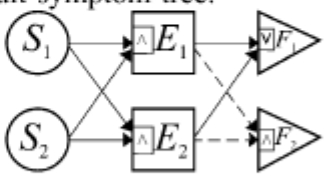


Figure 2.5: General Scheme of Process Model-Based Fault-Detection and Diagnosis [4]

The fault detection and diagnosis system illustrated in Figure 2.5, has four main sections, namely: Process, Model-based fault detection, change detection and fault diagnosis. Where faults in the process can be detected with the aid of detection methods which generate residuals r, parameter estimates or state estimates, which are based on measured input signals U and output signals Y. The features are then compared with nominal features and if changes in them are detected, it leads to the generation of analytical symptoms **s**. The analytical symptoms, **s**, are then further analysed to isolate faults. Table 2.1 below shows the different methods which utilises NN to analyse **s,** to perform fault diagnosis.

Table 2.1: Fault Diagnosis Methods [4, p.77]

Classification methods	Inference methods
	
Without a-priori knowledge on symptom-causalities. Mapping:  $\mathbf{S}^T = [\, S_1, S_2 \ldots S_n \,]$ $\mathbf{F}^T = [\, F_1, F_2 \ldots F_m \,]$	With a-priori knowledge on symptom-causalities. Causal network:  Fault-symptom-tree: 
Classification: - statistical - geometrical - neural nets - fuzzy clusters	Rules: If $< S_1 \wedge S_2 >$ Then $< E_1 >$ Diagnostic reasoning: - Boolean logic: facts binary - Approximative reasoning: - Probabilistic facts: probability densities - Fuzzy facts: fuzzy sets

With the above mentioned approaches, a number of fault detection and diagnosis systems have been designed, for example: fault diagnosis of a DC motor actuator used in an aircraft cabin control pressure system, where a combination of parameter estimation and parity equations were applied for the detection of several parametric and additive faults by using four measurements, followed by fuzzy logic interfacing; for fault detection and isolation on diesel engines, three different detection models were proposed to generate symptoms based on mainly production-type sensors. The symptoms were generated with non-linear input and output error parity equations. 20 symptoms were then generated, which allowed for in-depth diagnosis, e.g. by a fuzzy logic inference scheme [4, pp.77-84].

It could be noted that the above mentioned approaches used mathematical models to perform fault detection and isolation. The development of mathematical models is dependent on the availability of component data of all components used within the process or plant. This data is not always readily available, which makes mathematical modelling challenging.

To overcome the difficulties of using mathematical models and to make FDI algorithms more applicable to real systems, neural networks can be used to model system behaviour to both generate residuals, as well as isolate faults [5].

2.3 Neural Network Model-Based Fault Diagnosis

By using Neural Networks to model a system, the problems associated with FDI methods being sensitive to modelling errors, parameter variation, noise and disturbance, experienced with the use of mathematical models are eliminated or reduced as no mathematical model is needed. Figure 2.6 illustrates a general scheme of a model-based fault diagnosis system which utilises neural networks as replacement to mathematical/analytical models.

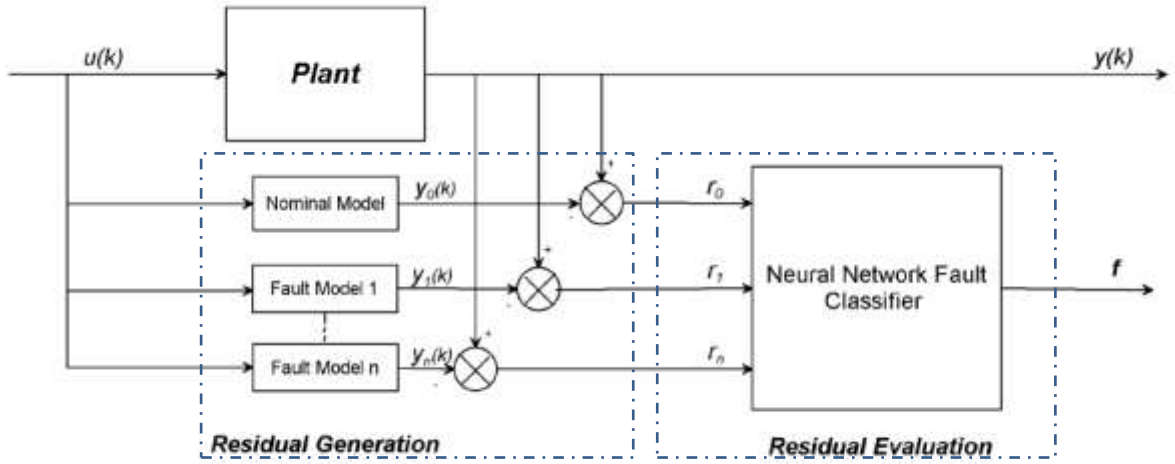


Figure 2.6: General Scheme of Model-Based Fault Diagnosis using Neural Networks

From Figure 2.6 it could be noted that there are different models running in parallel, where each model represents one class of the system or plant's behaviour. One model represents the system under normal operating conditions and each successive model thereafter represents a specific faulty condition.

The inputs $u(k)$ are fed into each model where the models then outputs $y_0(k), y_1(k), \ldots . y_n(k)$. These outputs are then compared to the plant's output $y(k)$ to produce residual vectors $[r_0, r_{1,\ldots .} r_n]$, which characterizes a suitable class of system behaviour. This process is referred to as a residual generation function. The residual vector r is then transformed by a classification Neural Network to determine the location and time of the fault.

One drawback to this method is the fact that it is impossible to model all potential system faults and normally only data containing normal operation is available, whereas data for faulty conditions have to be simulated, thus when designing faulty models using neural networks, serious problems can be encountered. [6, p.11]

Even with the above mentioned challenges, Neural Networks have been successfully applied to many applications to perform FDI, for example:

- In chemical processes: batch polymerisation and the distillation column, where a neural model prediction error was used as a residual, for fault diagnosis of sensors or components. Training time of the Neural Process model was reduced by the use of an input feature extraction process [6, p.22]
- Singh and Murthy [7] presented a Neural Network-based Sensor Fault detection for flight control systems, where two different approaches were used and analysed. The

first approach was based on an algorithmic method which dealt with Luenberger observers, whereas the second approach utilised knowledge-based neural network fault detection (KBNNFD) which used a feedforward Neural Network trained with the gradient descent back propagation training algorithm

- N.P. Srivastava, R.K. Srivastava and Vashishtha [5] highlighted a number of applications in which neural networks were successfully implemented which included fault diagnosis of nonlinear dynamic systems. The applications are as follows:
  - The use of a multilayer neural network to detect leakages in an electro-hydraulic cylinder drive of a fluid system [5, p.82]
  - The use of neural networks to detect internal leakages in control valves and motor faults in process plants [5, p.82]
- Mhamdi, Dhouibi, Liouane and Simeu-Abazi [8] described the use of four independent artificial neural networks which were used to perform fault detection and classification on power transmission lines. The method used consecutive voltage and current readings as inputs to an ANN, where the ANN's outputs were then used to indicate the presence and type of fault.

The above mentioned research makes it clear that neural networks can be successfully applied to perform fault diagnosis using different approaches. The different approaches can be defined as *Pattern recognition* and *residual generation and evaluation*.

In the field of residual generation and evaluation, thresholding techniques are of great concern. The decision whether or not a faulty condition exists, can be a daunting task, as signals are corrupted by noise and disturbances. These corrupted signals have a huge impact on the magnitude of residuals, where residuals can be defined as follows:

$$r_{k0}(t) = z_k(t) - z'_k(t), \qquad k = 1, \dots, n \tag{2.1}$$

Residuals in fault free cases should be close to zero and should be far from zero in the case of a fault. Thus some threshold value is needed to determine whether a residual value indicates a fault or not. This threshold value can be defined by the following equation [9, p.3187]:

$$|r_{k0}(t)| \leq S_{k0} : \text{No Fault} \tag{2.2}$$

$$|r_{k0}(t)| > S_{k0} : \text{Fault Detected} \tag{2.3}$$

The challenge is in selecting a threshold value which is not affected by corrupted signals, but is still large enough to avoid false alarms, but small enough to still be sensitive to faults to prevent non-detections [9, p.3187].

Kourd, Lefebvre and Guersi [9] proposed a fault diagnosis method which is based on a three valued residuals system, where analysis of the residuals $R_j(t), j = 0, \dots, p$ are done with parallel computation of fault-free and faulty ANNs models (*See Figure 2.7*). Detection and diagnosis are achieved according to a decision-making block, where diagnosis results are either from the use of usual thresholding or from online determination of fault probabilities and confidence factors [9, pp.3187-3189].

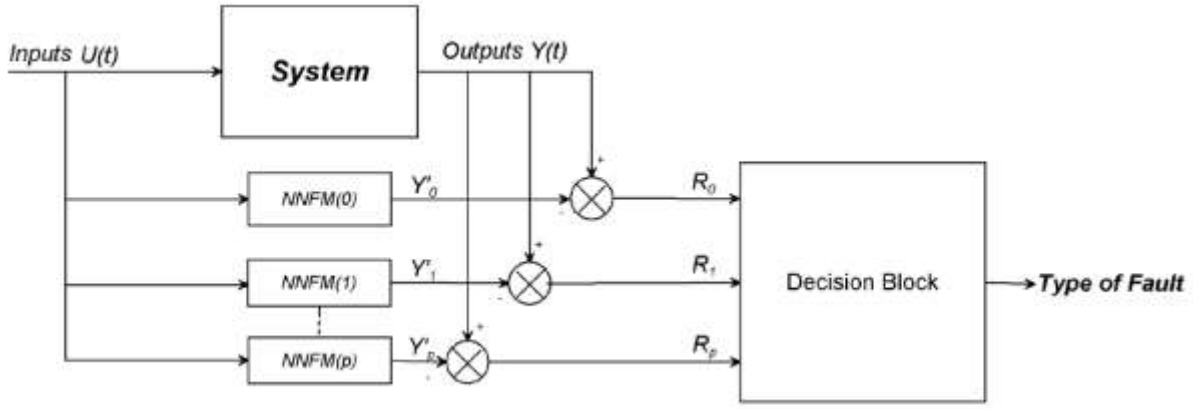


Figure 2.7: FDI design with Models of Faulty Behaviours

It could be noted that the main difference between Figures 2.6 and 2.7 is the fault diagnosis method, where Figure 2.6 uses a classification Neural Network and Figure 2.7 uses a decision block which utilises a fault probability estimation technique to perform FDI.

Kourd, Lefebvre and Guersi [9] successfully applied the above mentioned FDI method on a DAMADICS system which consisted of a control valve, a pneumatic servomotor and a positioner [9, pp.3189-3195].

Other thresholding techniques frequently used are: Simple Thresholding, Density Estimation, adaptive thresholds and Fuzzy Threshold Adaptation [6, pp.124-136]. Some of these will be discussed in section 2.4.1.2.

2.3.1 Conclusion

It could be noted from the above mentioned literature review that a number of different approaches exist for the detection and isolation of faults in processes or plants. It was also noted that the type of fault detection and isolation method relies on knowledge of the system and data availability, as well as online or offline fault detection requirements; thus it could be said that the method is dependent on the application.

When analysing the locomotive's data acquisition abilities it was noted that historical data could be used to create a nominal model of the system with the use of a data-based approach which simply replaces the analytical approach.

The literature review done in the above section showed that the use of a model to develop an FDI system has been implemented successfully into a number of applications and is known as model-based FDI systems. The proposed system design, as indicated in Figure 2.2, indicates that only nominal models were used in the FDI design; thus in order to effectively develop the proposed system, more research is necessary in the field of model-based fault detection and isolation. The next section will highlight different methods of model-based fault detection and isolation.

2.4 Review on Model-based Fault Detection and Isolation

From the previous sections it could be noted that Model-based fault detection and isolation has been successfully applied to a number of different applications. It could also be

gathered that the main requirement for a model-based fault detection and isolation system is a model of the system or process, which could be either in the form of a mathematical or neural network model. A number of model-based FDI systems have been developed which have yielded excellent results, where:

- Xu, Lee, Zhou, and Yang [10] developed a model-based fault detection and isolation system for a rudder servo system, which used a mathematical model, with an observer scheme to detect actuator as well as sensor faults
- Hashimoto, Kawashima and Oba [11] developed a model-based fault detection and diagnosis for internal sensors used on a mobile robot. Faults were grouped under three failure modes, namely: hard failure modes, noise failure modes and scale failure modes. The detection and isolation of faults were based on interacting estimators where the residual evaluation was done with the aid of kalman filters
- Addel-Geliel, Zakzouk and Sengaby [12] used a model-based observer scheme to detect abnormalities in an industrial boiler. Different fault scenarios were simulated on an identification model. To validate sensor readings an observer-based fault detection algorithm was used, where after a fault detection algorithm was applied to detect abnormal behaviours
- Zafira and Rahman [13] used a model-based observer system which utilised a genetic algorithm for optimization of the membership function to develop a fault detection and diagnosis system for a process control rig. The residuals generated from the observer model were used by an artificial neural network to classify faults
- Odgaard, Lin, and Jorgensen [14] presented and compared three different approaches to fault detection and isolation in power plant mills. The three approaches analysed were: observer, data-based fault detection and a combination of the two. The model development was done with the use of a mathematical model.

It could be noted that most of the above mentioned examples performed residual generation with the use of observer schemes. Observers are models of a system or process which can be in either a mathematical or neural network model form, where the neural network model can be developed with the use of historical process data or real time data.

The principle of design for each model-based fault diagnostic system is similar and will be discussed in this section. The section will further provide a review on applied and known theories surrounding model-based fault detection systems, where a more in depth analysis will be done on dedicated observer schemes, which were found to be used with great success in many applications [12],[13],[14],[16].

Research in the classical approaches of model-based fault detection, which utilise mathematical models, is necessary due to the fact that the only difference is the model used, which could be either mathematical or neural network based (which could be developed online or with the use of historical data), whereas all other residual generation and evaluation techniques are exactly the same.

2.4.1 Model-Based Fault Diagnosis

Model-based fault diagnosis depends on the analysis of the deviation between a model and real system responses. Figure 2.8 below shows a breakdown of model-based fault detection methods, which can be grouped into two main groups namely [12]:

- Quantitative Model-Based fault detection, and
- Qualitative Model-Based fault detection.

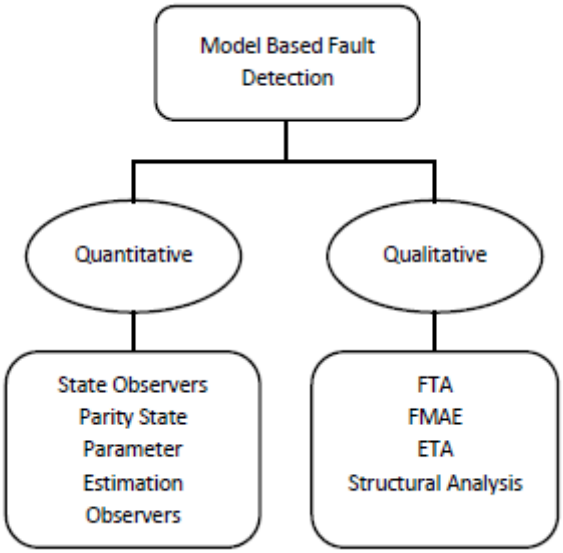


Figure 2.8: Model-Based Fault Diagnosis

The general idea of a *Quantitative model-based fault diagnosis* system is to compare measurements from a monitored system with that of predicted values obtained from a nominal model of the system. The difference between these two values is known as a residual. Figure 2.9 below shows the general idea of a Model-Based Fault Diagnosis System as described above.

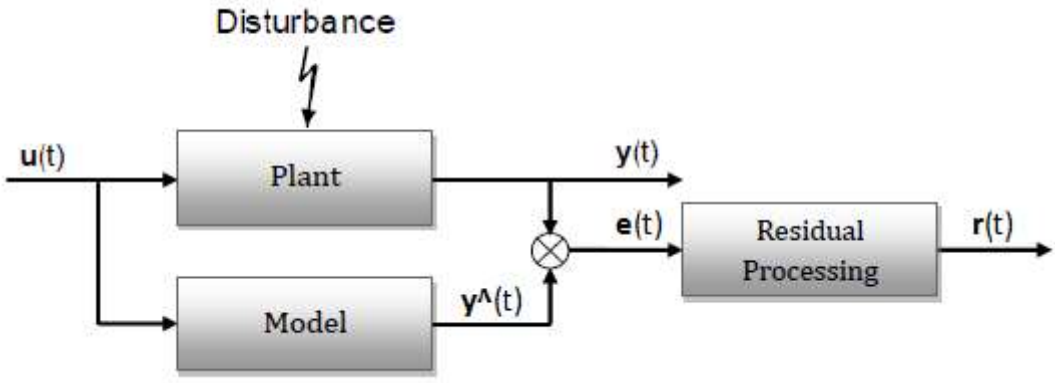


Figure 2.9: Principle of Operation of a Model-Based Fault Detection System

Figure 2.9 indicates that the system consists of two parts to perform fault detection and isolation, namely:

- *Residual Generation (e(t)), and*
- *Residual Evaluation (r(t)) also known as a Decision-Making System.*

The residual generation process is performed by the comparison of the measured signal *y(t)* and the predicted output signal *y^(t)* from the model. Under ideal conditions the residual will be zero, which should give a fault-free condition, whereas a faulty condition will produce a deviation from zero. The residuals are normally calculated with the use of analytical methods such as parity equations, observers and parameter estimators. These will be discussed in more detail in section 2.4.1.1.

The second part of the fault detection and isolation is known as residual evaluation where the *Primary Residual* is analyzed for likelihood of faults. The evaluation process may consist of the following methods:

- simple thresholding,
- moving average of the residual, and
- Methods based on statistical decision making theory; e.g. generalized likelihood ratio testing or sequential probability ratio testing [15].

These methods will be discussed in more detail in section 2.4.1.2.

Qualitative Model-Based fault detection methods on the other hand use symbolic reasoning to analyse the relationships between variables of a system. The symbolic reasoning combines different kinds of knowledge with graph theory to analyse the relationships between the variables. This type of model-based fault detection does not require an explicit model of the system; thus it is advantageous in systems where analytical models of a system is not available [3].

2.4.1.1 Residual Generation Methods

As illustrated in Figure 2.8 above, quantitative model-based fault diagnosis can be done using three main analytical processes, namely:

- Parameter Estimation
- Parity Equations
- Observers.

This section will highlight the differences of the different quantitative model-based residual generation techniques, where observers will be discussed in more detail due to its use in this project.

2.4.1.1.1 Parameter Estimation

Parameter Estimation can be defined as the process of estimating some or all of a system model's parameters using its input and output measurements. Residuals are then created when the estimated parameters are compared with the measured values [16]. This can only be done if the basic structure of the process is known. K Patan [6] explained parameter estimation by considering a process described by

$$y(k) = \Psi^T \theta \qquad (2.4)$$

Where Ψ and θ are the regressive and parameter vector respectively and can be given by:

$$\Psi = [-y(k-1) \dots, \dots., -y(k-m), u(k) \dots, \dots, u(k-n)]^T \qquad (2.5)$$

$$\theta = [a_1, \dots, a_m, b_0, \dots, b_n]^T \quad (2.6)$$

Patan [6] stated that if "*θ is assumed to have physical meaning, the task consists in detecting faults in a system by measuring the input **u**(k) and the output y(k), and then giving the estimate of the parameters of the system model $\hat{\theta}$"* [6, p.12].

If the fault acting on the system is modelled as an additive term f, then θ can be expressed as follows:

$$\theta = \theta_{nom} + f \quad (2.7)$$

where θ_{nom} represents the nominal (fault-free) parameter vector. The change detection can then be expressed as

$$\Delta\theta = \theta - \hat{\theta} \quad (2.8)$$

The change detection is then put through an evaluation process to determine whether a faulty condition is present. This is done by means of limit checking, where checks are done to ascertain whether the parameter change is greater than a predefined threshold value.

Patan [6] highlighted a drawback of this method where he stated:

"The main drawback of this approach is that the model parameters should have physical meaning, i.e. they should correspond to the parameters of the system. In such situations, the detection and isolation of faults is very straightforward. If this is not the case, it is usually difficult to distinguish a fault from a change in the parameter vector θ resulting from time-varying properties of the system. Moreover, the process of fault isolation may become extremely difficult because the model parameters do not uniquely correspond to those of the system. It should also be pointed out that the detection of faults in sensors and actuators is possible but rather complicated [6, p.12]."

2.4.1.1.2 Parity Relations

In order to define the parity relation let's consider Figure 2.10, which was set up by Isermann [1].

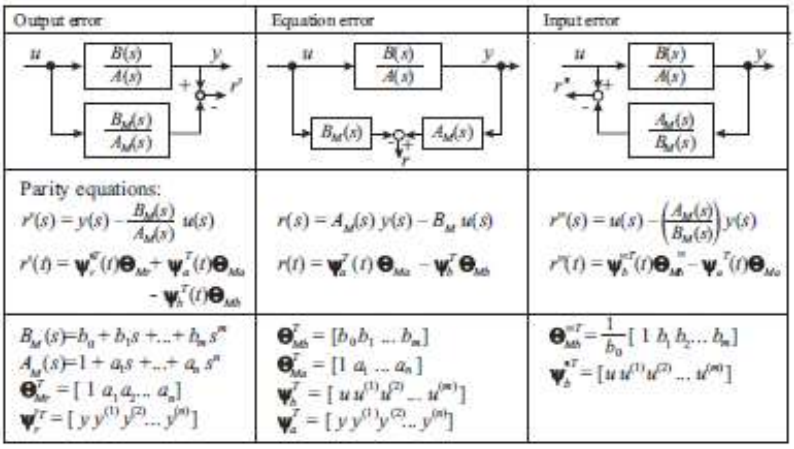


Figure 2.10: Parity Relations [1]

It can be seen that the Parity Relation, can be described by considering the following linear process's transfer function: [6, pp.12-13], [16, p.9]

$$G_p(s) = \frac{B_p(s)}{A_p(s)} \quad (2.9)$$

Now if the parameters as well as the structure of the process, are known, the process model can be represented as:

$$G_M(s) = \frac{B_M(s)}{A_M(s)} \quad (2.10)$$

And if we assume that $f_u(t)$ and $f_y(t)$ are addictive faults acting on the input and output of the system and $G_p(s) = G_M(s)$, the output error can be expressed as follows:

$$e'(s) = y(s) - G_M(s)\, u(s) = G_P(s)\, f_u(s) + f_y(s) \quad (2.11)$$

The residual $e'(t)$ changes with different transients due to faults which influence the input and or output of the process. $G_M(s)'s$ polynomials can also be used to form a polynomial error, which can be expressed as:

$$e(s) = A_M(s)y(s) - B_m(s)u(s) = A_P(s)f_y(s) + B_p(s)f_u(s) \quad (2.12)$$

The above equations are parity equations. Fault isolation for sensor faults can be easily realized, with the use of parity equations, where the general scheme for a dedicated fault isolation scheme is used to create parity equations to isolate faults for single sensor faults. This can be done by assuming that there are no actuator faults present in the system [6, p.13].

When considering actuator fault isolation, it can be done in a similar manner as sensor faults, by a process named the single-actuator parity relations scheme. Patan [6] indicates that although the single-actuator parity relations scheme is possible, not all actuator faults are isolatable using this scheme [6, p.13]. It could also be noted that some form of mathematical modeling is necessary to use this method.

2.4.1.1.3 State Estimators (Observers)

Observer Model-based fault detection systems have been successfully implemented in a number of applications [12], [13], [14]. The method allows for the detection of actuator as well as sensor faults and has different configurations for the detection of sensor and actuator faults. A requirement for this project is the isolation of actuator and sensor faults causing oscillations, thus making this method suitable as a residual generation technique. When considering the applications to which this method has been successfully applied, an important factor to be taken into account is data availability and knowledge about the process. In this section an overview on the operation of observers will be given.

State Estimators are used to reconstruct measurable or unmeasured state variables, from measuring input and output variables. Additive faults can be easily detected using this technique whereas multiplicative faults are more complicated to detect due to the fact that residual changes can be affected by changes in parameter, input and state variables; thus it is not easily detected in the output. State estimators also require an accurate mathematical model of the process; thus the performance of the estimation is dependent on the accuracy of the model [4].

To provide a mathematical explanation of an observer system let's consider the following state equations of a system [16, p.10]:

$$x(m+1) = Ax(m) + Bu(m), \tag{2.13}$$

$$y(m) = Ex(m) = \hat{y}(m) \tag{2.14}$$

Where:

$A = State\ Transition\ Matrix,$

$B = Input\ Matrix,$

$E = Output\ Matrix,$

$x = State\ Vector,$

$u\ and\ y\ are\ Input\ and\ Output\ vectors$

The estimates of the measured signals are then compared with their originals to compute the residuals as follows:

$$r(m) = y(m) - E\hat{x}(m) \tag{2.15}$$

It could be noted from the above equation that state estimators' main objective is to estimate the outputs of a system, thus estimating the entire state vector is unnecessary. This is due to the fact that reduced order observers can be employed facilitating state estimation significantly.

Figure 2.11 below shows the general idea of an observer-based system as per the above discussed.

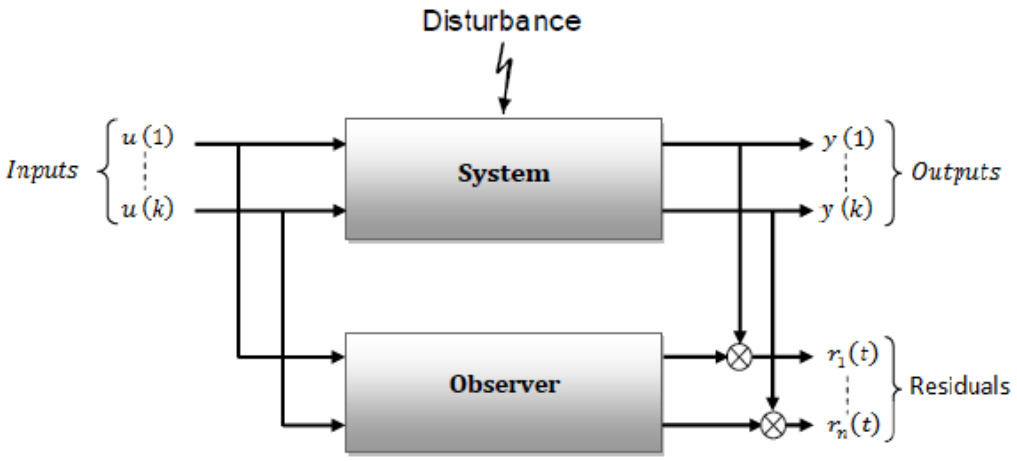


Figure 2.11: General idea of an Observer-Based System

Different types of observers have been developed over the years to enable different types of fault detections. These observers can be classified as follows:

- Generalized Observer Scheme [8]
- Output observers [8]
- Dedicated Observer Scheme.

For this project only the generalized and dedicated observer schemes will be analysed. Where the generalized observer scheme is formed by a bank of observers of reduced order; where for sensor faults (IFD) each observer uses all the inputs and n-1 outputs and for actuator fault detection each observer uses all the outputs and m-1 inputs. Where *m* is the number of inputs. [16, p.10] The generalized observer scheme can be described in layman terms as a bank of observers excited by all outputs or inputs except one which is then supervised.

In a dedicated observer scheme several observers constitute a bank of reduced order observers, where for the detection of sensor faults each observer uses all the inputs and just one output to detect faults. Here the number of observers equals the number of outputs which is also equal to the number of sensors. Dedicated observer schemes can also be used to detect actuator faults, where each observer uses one input and all outputs. The DOS scheme allows for the localization of multiple faults for either sensor or actuator faults [8].

The above mentioned dedicated observer schemes can be further grouped into the following:

a) Observer excited by one input: this was mentioned above but simply means that one observer is excited by a single input while the other outputs are estimated and compared to measured signals. This allows for single sensor faults (additive faults).
b) Kalman Filter excited by all outputs: Here a hypothesis test is used to detect residual changes of which changes in the characteristics of zero mean white noise with known covariance when a fault occurs.

It could be noted that a lot of observer-based systems exist, which works on the same principle as the above mentioned. An example of the application of the dedicated observer scheme was highlighted in [14] where a dedicated observer scheme was used to isolate sensor and actuator faults in a power plant coal mill.

Application of Observers within this Dissertation:

When considering the aim of this project, the detection and isolation of sensor and component faults are of primary concern in the event of oscillatory failing conditions which cause the entire close loop control system to oscillate, hence making fault isolation extremely difficult.

When considering Figure 2.2 which illustrated the proposed system, the dedicated observer scheme would provide a method to isolate both sensor and component faults. The two sections specified in the proposed system, namely: sensor and component fault analysis will be done with the use of the dedicated observer system, but observation will be performed offline on recorded data and not in real time.

This section showed that an observer scheme is dependent on a model of the system, which is normally a mathematical model. In the next section we will give an alternative to the mathematical model, namely: a Neural Network model to perform residual generation.

2.4.1.1.4 Neural Network Residual Generator

As mentioned above, the models discussed in section 2.4.1.1 used for residual generation or fault diagnosis required the use of mathematical models of the process. Even though software packages such as Matlab have applications which can generate mathematical models from input and output data, accurate mathematical models which models all system's physical conditions influencing a system are not easy to create. When considering technological plants, which are normally described by non-linear high-order differential equations, simplification is inevitable to accommodate quantitative modelling for residual generation. Difficulties arise from this due to reduction of dynamics order and linearization, as well as the problems which arise from unknown process parameters which have the effect that effective residual generation cannot be done with the use of conventional analytical models which are not accurate enough [6].

In these cases an alternative to analytical models are artificial neural networks, which can replace the analytical models by learning the process under normal operating conditions. Figure 2.12 shows a model-based neural network FDI system [6].

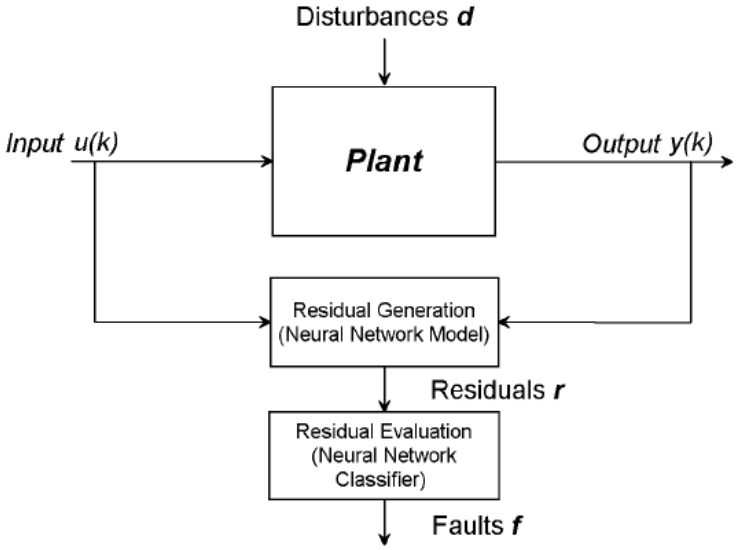


Figure 2.12: Model-Based Neural Network FDI System

In order for an artificial neural network to learn the process under normal conditions, data is required from the plant. Dependent on the system or process, the data can be gathered either directly from the plant or from a simulated model that represents the process as realistically as possible. Training can then either be on-line or off-line. The type of training is dependent on the availability of data.

Once the NN is trained, it represents the process under normal operating conditions and can be used to generate residuals using the below mentioned equation:

$$r(m) = y(m) - \hat{y}(m) \qquad (2.16)$$

Where $r(m)$ is the residual generated from comparing the measured output $y(m)$ with the NN predicted output $\hat{y}(m)$. Thus it shows that NNs can be used to replace mathematical models.

It is this method which will be used within this project to model a system and generate residuals. As per section 2.4.1.1, a dedicated observer scheme will be used in conjunction with a neural network to generate the residuals. The neural network model will be developed from recorded process data from the locomotive's excitation system. Thus it will be designed as a model but more as a data-based model.

Accuracy of a model used for residual generation is one of the most important factors within the application of FDI; thus the training of ANNs will be discussed in more detail in Chapter 3 and residual evaluation techniques will be discussed in the next section.

2.4.1.2 Residual Evaluation Techniques

Any Model-Based fault detection system, whether analytical, ANN or fuzzy logic, consists of a residual generation process and a decision-making process, where the residuals are evaluated to make a decision whether or not a fault occurred. The decision-making process, is responsible for alerting a user of the occurrence of a fault. Residual evaluation is basically a logical decision-making process which transforms quantitative knowledge into qualitative Yes-No statements. To perform the transformation from quantitative knowledge into qualitative statements, some measures need to be put in place to enable the FDI system to perform robust decisions.

In order to fully consider the necessity of a robust decision-making system let's revisit section 2.4.1.1 where the residual generator was defined as a deviation between a measured signal and a model estimate of the signal. The residual $\bar{r}_k$ can then be defined as [16, p.47]:

$$\bar{r}_k = (y_{realk} - \hat{y}_k)^2 \qquad (2.17)$$

Where y_{realk} and $\hat{y}_k$ are the sensor measurements and model estimations of plant characteristics at time instant k. Ideally the model estimate and the sensor measurement should be equal, leading to a residual of zero for fault-free conditions, thus indicating that a threshold value of zero could be used for the decision-making process. However in practice signals or measurements are affected by unknown inputs, which are due to the presence of noise contained within a signal. The model estimate can also be affected by modelling errors and these factors cause the residual to be greater than zero even in a fault free case, hence causing false alarms. A simple solution to this is to raise the threshold value; however this increases the risk of not detecting faults. Owing to this, a number of well-established approaches to residual evaluation and/or threshold calculations have been developed over the years where the objective was to have the decision-making process or threshold values be insensitive to uncontrolled effects such as modelling errors and other disturbances in the system. In this section we will discuss the following residual evaluation techniques which are used to calculate thresholds and or evaluate residuals [16, p.47]:

- Simple Thresholding
- Adaptive Thresholding
- Moving Average
- Neural Networks
- Statistical decision-making theory, e.g. generalized likelihood ratio testing or sequential probability ratio testing [15].

2.4.1.2.1 Simple Thresholding

Simple thresholding is one of the simplest methods used for residual evaluation. The theoretical analysis of a healthy system is defined as follows: if the residual generated is smaller than a threshold value, the process is considered healthy, otherwise it is faulty. In theory a fault-free case refers to a condition in which the residual value is zero. However, in practice this is not feasible due to modelling errors and noisy signals; thus thresholds need to be larger than zero to prevent false alarms. In order to select a threshold range, let's assume that a residual satisfies the following:

$$r(k,\theta) = \epsilon(k), \qquad k = 1, \dots, N, \tag{2.18}$$

Where $\epsilon(k)$ is equal to $\mathcal{N}(m,v)$, which are random variables with a mean value m and standard deviation v. N specifies the number of samples which are used to calculate m and v. θ represents the vector of the model parameters and β the significance level which corresponds to the probability that a random value t_β is exceeded by a residual with $\mathcal{N}(0,1)$. [6, pp.124-125]The significance level β can be expressed as:

$$\beta = P\left(\left|\frac{r(k)-m}{v}\right| > t_\beta\right) \tag{2.19}$$

t_β is tabulated in most statistical books and can be obtained by assuming a significance level β and then a threshold, T, can be calculated using the following equation:

$$T = t_\beta v + m \tag{2.20}$$

Residual evaluation is then done by comparing the absolute residual value and comparing it to its assigned threshold T. [6, pp.124-125] A diagnostic signal is then created and assigned a value, according to the following:

$$s(r) = \begin{cases} 0 \; if & |r(k)| < T \\ 1 \; if & |r(k)| > T \end{cases} \tag{2.21}$$

The diagnosis signal $s(r)$ is assigned a value of zero if the residual's absolute value is less than the threshold T, and one if it is greater.

Thresholds can also be calculated using the following threshold calculation, which eliminates the need for the determination of t_β. [6, pp.124-125] The threshold value is derived using ζ- standard deviation where the residual is assumed to be a random variable $\mathcal{N}(m,v)$. Thresholds are then calculated as follows:

$$T = m \pm \zeta v \tag{2.22}$$

Where $m \; and \; v$ is defined as follows:

$$m = \frac{1}{N}\sum_{i=1}^{N} r_i \tag{2.23}$$

And

$$v = \frac{1}{N-1}\sum_{i=1}^{N}(r_i - m)^2 \tag{2.24}$$

With N being the number of samples and r_i being the residual value. ζ defines the probability that a sample exceeds the threshold value T, where the probability is equal to 0.15866 for $\zeta = 1$, 0.02275 for $\zeta = 2$ and 0.00135 for $\zeta = 3$ [6, p.125]. The effect of changing ζ is indicated in Figure 2.13 [6, p.125].

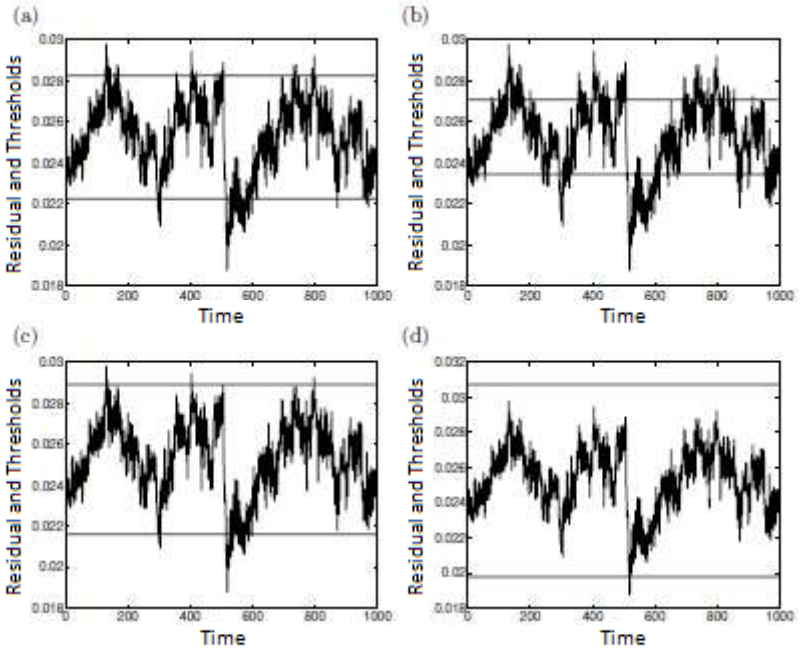


Figure 2.13: Residual Thresholds calculated with $\zeta = 1\ (b); \zeta = 2\ (c)\ and\ \zeta = 3\ (d)$ [6, p.125]

When analysing the above figure it could be noted that for smaller values of ζ the confidence level is narrow, which makes the decision-making system sensitive to faults and produces a large number of faults. By increasing the value of ζ, the confidence interval becomes larger, thus making the system less sensitive to faults. Thus it could be noted that the choice of the threshold value is a compromise between fault detection sensitivity and false alarm rates.

The above method works well and gives satisfying results when the residual is assumed to be normal. In order to determine whether residuals are normal, a normality test is needed. One of the most used methods used to test normality is to compare the cumulative distribution function of a residual $F_r(x)$ with the normal distribution $F(x)$ [6, pp.125-126]. With this method the residual first needs to be normalised. Patan [6] showed that this can be done as follows:

$$r_n(k) = \frac{r(k)-m}{v}, \qquad k = 1, \dots, n, \tag{2.25}$$

Where $m = mean\ value\ of\ r(k)\ and\ v\ is\ the\ standard\ deviation\ of\ r(k)$. The residual can then be ordered by indexing time instants as follows:

$$r_n(k_1) \leq r_n(k_2) \leq \cdots \leq r_n(k_n) \tag{2.26}$$

The cumulative distribution function can then be defined as:

$$F_r(x) = \begin{cases} 0 & if \quad x < r_n(k_1) \\ \frac{i}{n} & if \quad r_n(k_i) \leq x < r_n(k_{i+1}) \\ 1 & if \quad r_n(k_n) \leq x \end{cases}$$

$F_r(x)$ can then be plotted against the cumulative distribution function of a normal variable $\mathcal{N}(0,1)$ - $\mathrm{F}(\mathrm{x})$ [6, p.126]. Figure 2.14 below shows the Normality testing of the residual using the above mention method.

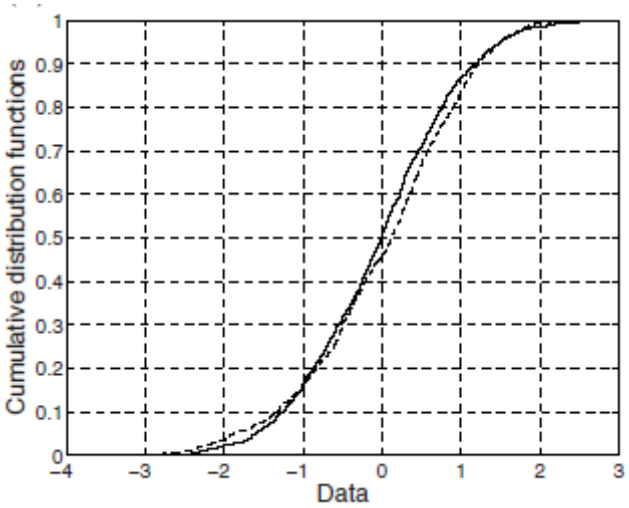


Figure 2.14: Normality Test with the use of the Cumulative Function [6, p.127]

It could be noted that when comparing the dashed cumulative distribution function of the residual with the normal distribution, that the empirical cumulative distribution function of the residual is not symmetric thus indicating that the normality function is not valid in this case and by assuming that the residual has normal distribution, and applying a confidence level will cause significant mistakes in the decision making process [6].

An alternative method for normalised testing is what is known as probability plots, where $F(x)$ is plotted as a function of i/n. In this method the normality assumption is accepted if the plot is close to a straight line [6, p.127]. Let's consider Figure 2.15 below, which shows an example of a probability plot.

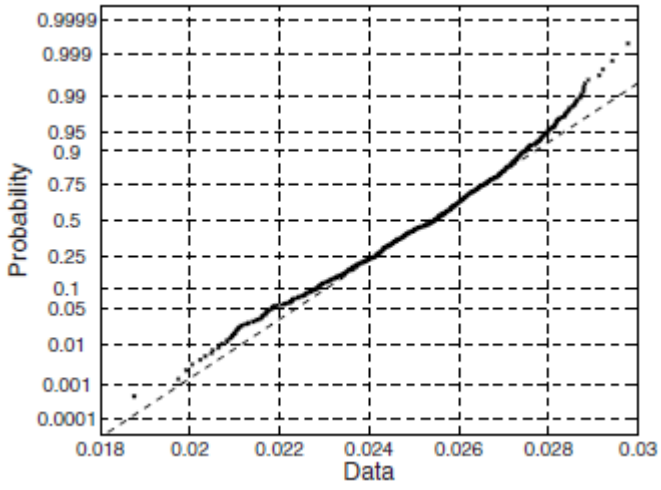


Figure 2.15: Normality Test with the use of the Probability Plot [6, p.127]

It can be seen that there are large deviations from the normal distribution, thus indicating that the normality assumption is invalid.

As mentioned above, if a normality test fails and normal distribution statistics for decision making is used, it can cause significant mistakes in the fault detection process. In order to prevent the latter, the selection of a threshold should be done in a proper way by determining the distribution of the residual or by the transformation of the residual to another known distribution should be performed [6].

2.4.1.2.2 Adaptive Thresholds

As mentioned above, in practice modelling uncertainty and measurement noise, makes it necessary to set threshold values larger than zero to avoid false alarms. Owing to this, a compromise between fault detection sensitivity and false alarms needs to be reached and for this reason, adaptive thresholds are applied. The main idea of adaptive thresholds is that they should vary with time due to the fact that disturbances and other uncontrolled inputs can also vary with time. Adaptive thresholding can be constructed based on the estimation of statistical parameters which is based on historical observed residual data [6, p.133].

If the residual can be an approximation of normal distribution then the values of the mean can be estimated over the past n sample as follows:

$$m(k) = \frac{1}{n}\sum_{i=k-n}^{k} r(i), \tag{2.27}$$

$$v(k) = \frac{1}{n-1}\sum_{i=k-n}^{k}\big(r(i) - m(k)\big)^2 \tag{2.28}$$

Where $v(k)$ is the variance and $0 < n < k$. Substituting these two equations into the following equation a threshold can be calculated:

$$T(k) = t_\beta v(k) + m(k), \tag{2.29}$$

A problem with equation 2.27 and 2.28 is the selection of the length of the time window n. If n is too small, the threshold adapts very quickly to any change in the residual caused by any factor (*These factors includes disturbances, noise or a fault*). On the other hand, if n is too large the threshold does not adapt sufficiently and acts as a constant one, hence decreasing fault sensitivity. [6, p.133] To prevent too fast an adaptation, a weighted sum of current and previous residual statistics is used as follows:

$$T(k) = t_\beta \bar{v}(k) + \bar{m}(k), \tag{2.30}$$

Where $\bar{v}(k)\ and\ \bar{m}(k)$ can be calculated as follows:

$$\bar{v}(k) = \zeta v(k) + (1-\zeta)v(k-1), \tag{2.31}$$

And

$$\bar{m}(k) = \zeta m(k) + (1-\zeta)m(k-1), \tag{2.32}$$

Where ζ is a momentum parameter, which is used to control the influence of the current and previous value of the standard deviation on the threshold. It is used for the same function on $\bar{m}(k)$.

Figure 2.16 below shows the operation of the adaptive threshold calculation as per the above mentioned.

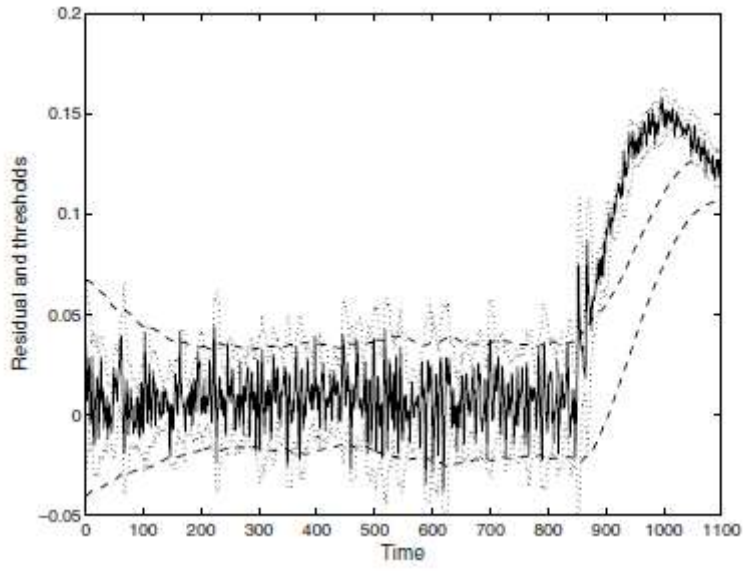


Figure 2.16: Adaptive Thresholding [6, p.135]

Figure 2.16 shows the residual as a solid line and the adaptive threshold as a dotted line when using equation 2.29 and a dashed line which represents the use of equation 2.30. It could be noted that when using equation 2.29, the threshold value takes the form of the residual itself rendering it useless. This is due to the short time window which makes the threshold adapt to residual changes quickly.

When considering the dashed threshold, it could be noted that the threshold is not influenced by fast changes in the residual statistics. This is due to the momentum term which is usually defined as $\zeta = 0.99$. The problem with this method is the selection of a proper or ideal momentum term, which can become troublesome [6, p.135].

More methods exist which take into account model uncertainties, which take into account measurable inputs and outputs [6].

A lot of different threshold adaptation methods exist such as Fuzzy Threshold adaptation which utilises fuzzy logic approach, but for this project we will only highlight the above mentioned [6, pp.123-140].

2.4.1.2.3 Moving Average Filter

The generated residual, can be filtered with the use of a moving average filter, as to sufficiently dampen the residual noise.[16, p.47] The residual can then be expressed as follows:

$$r_{kRGE} = \frac{\bar{\omega}}{\Omega} \sum_{j=k-(\Omega-1)}^{k} \bar{r}_j \qquad (2.33)$$

Where the arithmetic mean is chosen as the average and the weighted moving average of the past Ω residuals generated. $\bar{\omega}$ is equal to a user defined weight and r_{kRGE} is the residual generated at sample instant k. Now the generated residual can be evaluated with the use of a predefined threshold value. The moving average method is basically a moving average

filter which filters out noise from the generated residual as per equation 2.17. [16, p.47] Figure 2.17 (a) and (b) below shows the difference between a residual with and without the use of the moving average filter as per the above mentioned.

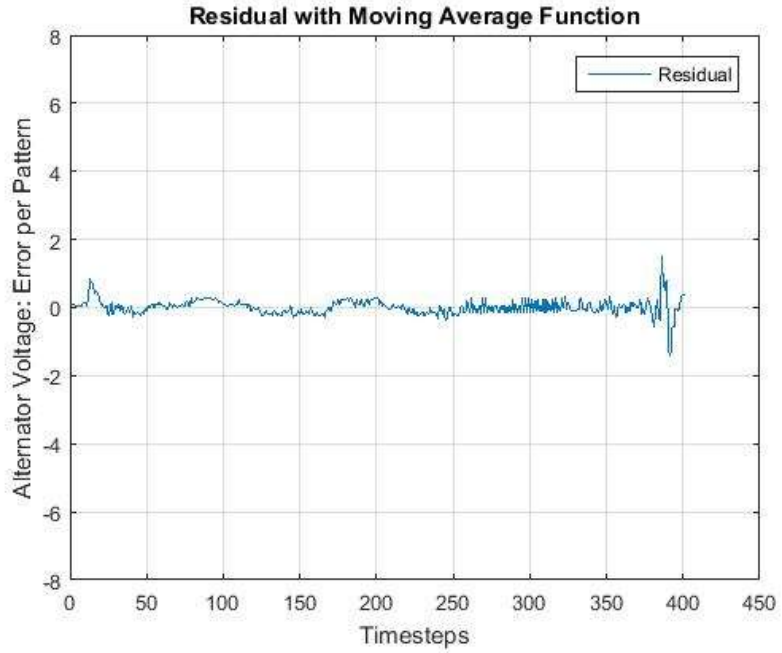


Figure 2.17 (a): Residual with a Moving Average Filter

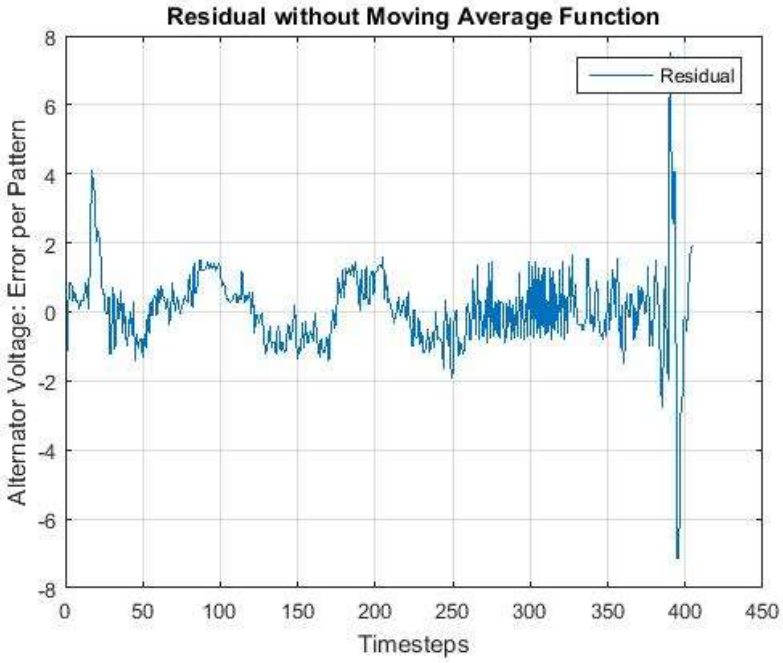


Figure 2.17 (b): Residual without a Moving Average Filter

It should be noted that the user defined weight, $\bar{\omega}$, for the above graph was set equal to 1. It could be noted from the graph that the overall noise is dramatically minimized by using the moving average filter. The filtered residual is so effectively done that the theoretical fault diagnosis principle of a residual of zero, indicating fault-free and greater than zero, indicating a fault can almost be realized.

The above mentioned method was further developed to reduce the effect of noise on the residual in order to further reduce the number of false alarms. The method as indicated by [16], uses the same steps as the above mentioned but introduces an additional artificial data parameter known as a padding factor. Equation 2.33 is then modified to give the following:

$$r_{kRGPE} = \frac{\bar{\omega}}{\Omega + P_{pad}} \left[\sum_{j=k-(\Omega-1)}^{k} \bar{r}_j + P_{pad} min\{\bar{r}_k - (\Omega - 1) : k\} \right] \quad (2.34)$$

Where

$r_{kRGPE} = residual\ generated\ at\ time\ instant\ k$

$P_{pad} = number\ of\ padding\ point\ to\ be\ added$

$\min \quad = minimum\ function$

The residual padding is used to extend the Ω samples from the original equation 2.33 with P_{pad} artificial data points before the moving average calculation is done. The added artificial data points are equal to the minimum value in the original samples [16, pp.47-48].

Residual padding is used as an attempt to dampen the effects of unknown inputs by assuming that:

- "the residual has a fast spike type effect and settles to a close-to-zero value" [16, p.48]
- It also assumes that faults have permanent effects on the residuals. In other words, the residual does not settle back to close-to-zero once a fault occurs.

It works on the principle of reducing the overall average of a residual when residual noise is present. This then reduces the chances of the residual exceeding a threshold value, resulting in false alarms. The average is reduced by extending the residual set by adding artificial data which is equal to the minimum data value from the residual set. This process is known as padding [16, pp.47-48].

The above mentioned algorithms are good at reducing noise which is located in a signal, but excessive padding and amplification using $\bar{\omega}$ should be avoided as this can increase fault detection time or completely dampen faults and increase the false alarm rate. Thus the tuning factors for the two methods mentioned above should be carefully chosen; these factors are; $\bar{\omega}, P_{pad}\ and\ \Omega$ respectively [16, pp. 48].

2.4.1.2.4 Residual Evaluation with Neural Networks

Residual evaluation methods using neural networks have become more and more attractive in recent years. Where a number of evaluation systems have been developed with the use of supervised Neural Networks which act as classifiers [17, p.352]. For these Neural Networks prior knowledge of the faults is needed to perform fault isolation. Koppen-Selige and Frank [18] used a multi-layered feedforward neural network to perform residual evaluation with the above mentioned technique.

Garcia, Schubert and Frank [19] used a Restricted Coloumb Energy Neural network to perform residual evaluation on a winding machine, where the Neural Network simultaneously evaluated 5 residuals in order to isolate a fault. This method also used a

scoring system where the highest probable cause of failure had the highest fault count number, thus isolating the fault.

2.5 Conclusion

With all the relevant literature reviews done it could be stated that the proposed FDI system is indeed plausible, due to the amount of similar methods used in a number of different applications. Thus for this project a neural network dedicated observer scheme will be utilized to detect and isolate oscillatory sensor and component faults. The observer scheme will utilize a neural network model of the system to perform residual generation. This method of residual generation was selected due to the nature of the locomotive's data acquisition abilities which only allows recorded data to be captured, which makes it perfect for modeling the excitation system under normal conditions.

Literature indicated that a number of residual evaluation techniques exist and for this project the threshold values will be calculated using the simple thresholding technique in conjunction with a moving average filter.The fault detection and isolation of component and sensor faults using a dedicated observer scheme, will work as per Figure 2.18 below.

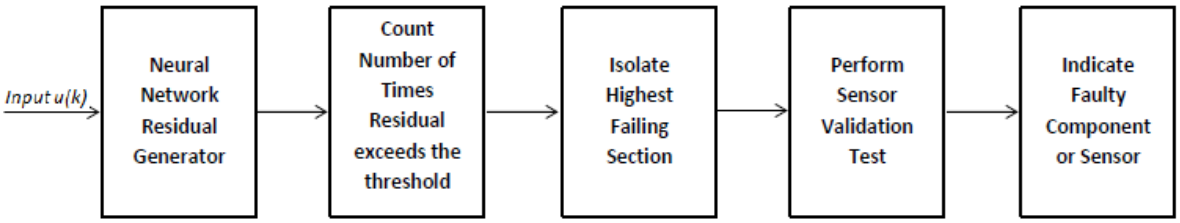


Figure 2.18: Breakdown on the Operation of the Proposed FDI System

As mentioned above, there are two types of faults to be isolated, which can be categorised as: sensory or component faults. These faults have a direct impact on the number of NN models to be used for the fault detection process. Thus let's revisit section 2.4.1.1.3, which stated that sensor faults are detected with the use of a dedicated observer scheme, where all inputs are used and just one output and where the output is equal to the number of sensors. Alternatively, component faults are isolated using one input and all outputs. Thus for this project, as indicated in section 2.1, the proposed system will have two stages of fault isolation. Firstly, the component fault isolation will be done where the primary controlling factor of the system will be the input and all affecting variables will be compared to measured readings. Then the highest failing section/component from the first test is evaluated further with the aid of the dedicated neural network observer scheme for sensor fault detection or sensor validation.

From the above mentioned it could be noted that two main factors affect the performance of any FDI system, which can be given as the accuracy of the model used to represent the plant and the threshold calculation which is used in the decision-making process. Owing to the fact that the accuracy of the model is of utmost importance to the successful implementation of a FDI system, Chapter 3 will discuss the artificial neural network in terms of its use in FDI systems, as well as a brief background on the operation and training thereof.

Chapter 3 – Artificial Neural Network Review and Applications in FDI systems

Chapter 2 highlighted the use of a neural network as a replacement to mathematical models within the application of Model-Based fault detection and isolation. This chapter will further elaborate on the artificial neural network and its use in real world FDI applications. The question as to why artificial neural networks and not classical statistical methods for FDI will also be discussed. First a brief history on the fundamentals of neural networks will be given.

3.1 History of Neural Networks

The human brain is an amazing but complex organism; it sets human beings apart from all species on earth due to its abilities which enable us to walk, talk, read, write, analyze odours, have learning abilities, control motion, recognize abilities, memorize, etc. It should be noted that the brain itself cannot perform all these functions alone, but works hand in hand with all senses of the human body, where the senses act as inputs to the brain. The brain then outputs commands to the body's different organs to execute a specific process. To perform all the above mentioned tasks, the brain uses what is known as biological neural networks.

The biological neural network is estimated to house in the order of 10-500 billion neurons with 60 trillion synapses, where the neurons are arranged in approximately 1000 main modules, each having 500 neural networks. It's these interconnected neural networks which give the brain the ability to solve several problems simultaneously, and more importantly give the brain its ability to learn, memorize and still generalize. It is the latter which prompted research in algorithmic modeling of biological neural systems, which is today known as artificial neural networks [20, pp.3-4].

Artificial neural networks can be described as mathematical models of real world systems which are equated by changing weights to obtain a satisfactory output or give a solution. Many different artificial neural networks exist and these are used for many different applications.

This chapter will consider the use of an artificial neural network to model a closed loop control system for a Diesel-Electric locomotive's excitation system, as well as highlight different approaches to training neural networks.

The chapter is divided into 6 sections where reviews are done on artificial neural networks, neural network architectures, learning paradigms, feedforward neural networks and activation functions. The chapter will then conclude in section 3.7, where it states which techniques will be used in the project.

3.1.1 Biological Neural Systems

To effectively define the operation of an artificial neural network, it is advantageous to first consider the basic building blocks of a biological neural system, which consist of neurons. Figure 3.1 below illustrates the anatomy of a neuron which consists of a cell body, dendrites and an axon. It could be noted that neurons are interconnected with the connections

known as synapses, which is between the axon of one neuron and the dendrite of another neuron. Signals are propagated from the dendrites through the cell body to the axon. From the axon the signal is propagated to all connected dendrites. A neuron either inhibits or excites a signal, where the signal is excited only when the neuron "fires" [20, pp.5-7].

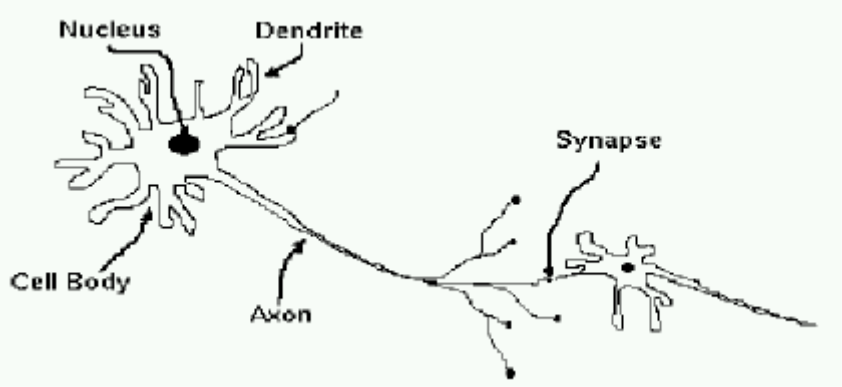


Figure 3.1: Biological Neural Network System [20, p.6]

3.1.2 Artificial Neural System

An artificial neuron (AN) is a mathematical model of a biological neuron (BN), which simulates a BN's principle of operation. Figure 3.2 illustrates an artificial neuron, where the input signals are inhibited or excited through numerical weights which are associated with each connection to the AN. The firing strength of the artificial neuron is controlled via a function known as an activation function [37, pp.8-10].

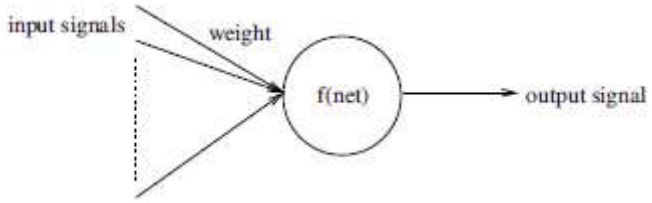


Figure 3.2: Artificial Neuron [37, p.8]

The AN computes the output signal as follows:

The input signals are multiplied by individual weights and then summed together. The summed inputs are then subtracted by a Bias (ɵ) to create the net.

The net input signal can then be represented as:

$$net = \sum_{i=1}^{I} x_i w_i - \theta \quad (3.1)$$

Where $x_i = input\ signal$, $w_i = weight$, I = Number of input vectors and ɵ = $x_{I+1}w_{I+1}$ where x_{I+1} is always -1.

$$net = \sum_{i=1}^{I} x_i w_i + x_{I+1}w_{I+1} \quad (3.2)$$

Thus the equation can be simplified to calculate the net as follows:

$$net = \sum_{i=1}^{i+1} x_i w_i \quad (3.3)$$

From this, the output signal is calculated using the equation

$$y = f(net) = f\left(\sum_{i=1}^{i+1} x_i w_i\right) \quad (3.4)$$

It is important to note that artificial neurons, which is also known as perceptrons, as described above, can be used to realize linearly separable functions without any error, but cannot learn non-linear functions with the same accuracy. To learn non-linear functions a layered neural network is needed. This will be discussed in the next section 3.2.

3.2 Artificial Neural Network – Overview

Artificial neural networks have the ability to extract patterns and detect trends by deriving meaning from complex, incomplete or imprecise datasets. These datasets are usually too complex to be analyzed by either humans or conventional computer techniques. A supervised neural network which is trained on a set of data can be used to approximate an output for a set of inputs, which was not in the training set. Thus the neural network could be considered an expert on the dataset which it has learned. Similarly, an unsupervised neural network can approximate outputs from a set of inputs for which it was never trained or where the training did not include a target vector to evaluate the neural network's performance.

Owing to the ability of neural networks to learn, memorize and generalize, a number of applications for artificial neural networks exist. Below is an incomplete list of a number of applications:

- Process Modeling
- Fault Detection and Isolation of Control Systems
- Diagnosis of Diseases
- Speech Recognition
- Data
- Composition of Music
- Image Processing
- Forecasting
- Robot Control
- Credit Approvals
- Classification
- Pattern Recognition
- Compression and many others.

The above mentioned applications can be divided into three main categories according to Edward and Jones [21]. The categories are as follows:

- Forecasting/Prediction: Where both quantitative and categorical input data are used to predict one or more quantitative outcomes
- Classification: where input data is classified into one of two or more categories
- Pattern Recognition: where patterns are uncovered among a set of variables.

Before neural networks these problems were solved using traditional statistical methods such as [21, pp.7-8]:

- Linear Regression
- Logistic Regression
- Principal Component Analysis
- Discriminant Analysis
- K-nearest Neighbour Classification
- ARMA and non-linear ARMA Time Series Forecasts.

However neural networks provided a single framework to solve many of these traditional problems and even extend the range of problems that can be solved in some cases. Research indicates that a simple neural network configuration yields the same result as most traditional statistical applications. It was also found that neural networks provide more accurate and robust results for problems, where tradition methods cannot be completely used or applied to [21, p.8].

When considering forecasting or prediction problems, traditional methods have been shown to be problematic when a time series [21, p.8]:

- Is non-stationary
- Data is noisy
- Or the data is too short, in other words not enough data presented.

Traditional methods have been shown to produce poor forecasting abilities when one of the above mentioned problems existed. Neural networks, on the other hand, can produce good results from data sets where the above mentioned problems exist, due to their ability to adapt to changes in a non-stationary series, filter out noise and train on less data. Thus it could be noted that neural networks provide a single tool for problems, where a variety of traditional statistical methods would be needed to solve a problem [21, p.8].

It could be noted that in order for neural networks to solve the problems associated with the different applications which were categorized as forecasting, pattern recognition and classification, a number of different types of neural networks have been developed over the years, for example:

- Single Layered Neural Networks
- Multi Layered Neural Networks
- Temporary Neural Networks, such as Recurrent Neural Networks and Time-Delayed Neural Networks
- Self-organizing Neural Networks, such as the Kohonen self-organizing maps
- Combined Supervised and Unsupervised Neural Networks.

The different neural networks discussed above are used to solve the three different problems as highlighted in this section. It should be noted that even though these networks solve different problems, they are always the result of computations that proceed from network inputs and outputs, where the network inputs are referred to as patterns and the outputs as classes.

3.2.1 Artificial Neural Networks Real World Applications

The project is based on FDI systems; thus all the applications will be listed with reference to FDI systems. Neural networks have been successfully implemented in many linear and non-linear systems to perform fault diagnosis. Below is a list of real world FDI systems:

- Watton and Pham used multi-layer perceptron networks to detect leakages in electro-hydraulic cylinder drive in a fluid power system[5]
- Sharif and Grosvenor applied neural networks to detect internal leakages in control valves and motor faults in process plants [5]
- Butler highlighted the use of a neural network to perform fault diagnosis for power distribution networks [5]
- James and Yu used a neural network to perform condition motoring and fault diagnosis of a high pressure air compressor valve [5].

These are just a few of real world applications which exist for the use of neural networks in FDI systems. Neural networks have also been successfully applied as residual generators as well as residual evaluators within dynamic systems [5]. These neural networks were used as follows:

- Neural networks were applied to state and parameter based FDI schemes by Han and Frank [5], where they proposed a parameter estimation based FDI system using neural networks to estimate physical parameters
- Fuente and Vega used neural networks for FDI on a biotechnological process [5]
- Yu et al. researched the use of a radial based neural network to generate residuals for diagnosing sensor faults in a reactor [5].

Not all of the neural networks used in the above applications have the same network architecture due to the fact that network architecture is application specific. In the next section network architectures will be discussed and highlighted.

3.3 Neural Network Architecture

When considering neural network architecture, three parameters are essential: topology, learning paradigm and learning algorithms. This section will concentrate on the network topology, whereas a learning paradigm and learning algorithms will be discussed in section 3.5. Before continuing it is important to define what a network topology is. A network topology can be referred to as the manner in which the nodes in the neural network are connected and organized, and the way data and error information travel from one layer of nodes to the next. Fundamentally, network topology can be grouped into two main groups, namely: feedforward and feedback. The main difference between these two topologies is the manner in which the inputs from preceding layers are combined in the hidden layers. A number of neural network designs have been developed over the years with different combinations of these three parameters [22]. As mentioned above, a number of FDI systems have been developed which utilize neural networks. When looking at a topology which is simple yet effective for fault detection and isolation, the feedforward network has been implemented successfully for modeling/residual generation, as well as for residual evaluation in the following systems:

- Guo, Liu, Xu and Chen [23] used a multi-layered feedforward neural network, trained using back propagation to isolate faults in a hydro turbine governor
- Taplak, Uzmay and Yıldırım [24] used a feedforward neural network with only one hidden layer to model a ball-bearing system, which was then further used to perform FDI using vibrations based fault diagnosis
- Mousavi and Khorasani [25] used a dynamic multi-layered feed forward neural network to perform residual generation, by modelling an altitude control subsystem of a formation flying satellite. The neural network incorporated an IIR filter, but was still based on the same principles as a normal feedforward neural network and was trained using the BP training algorithm
- Saravanan, Duyar, Guo and Merrill [26] used a feedforward neural network that was trained using back propagation to model a space shuttle main engine.

With the above mentioned systems and the use of a feedforward neural network as either a model/residual generator or a residual evaluator, it is clear that even though the feed forward neural network is considered the simplest neural network, it has the ability to model any non-linear or linear system with relatively high accuracy, using a back propagation training algorithm. Thus for the application of FDI on the locomotive's excitation control system, a standard multi-layered feedforward neural network will be used. Owing to its contribution in this project, the operation of a feedforward neural network will be discussed in the next section.

3.4 Feedforward Neural Network Model

A feedforward neural network consists of three layers, namely: input, hidden and output layers which form what is known as a layered network. The layered network has feedforward connections from the input layer to the hidden layer, which is then in turn connected to the output layer which displays the result. The term feedforward is used to indicate that the network operates in one direction and does not have any feedback loops.

Figure 3.3 illustrates a multi-layered feedforward neural network (FFNN). Here it can be seen that the network consists of three layers as mentioned in the above section (an input layer, a hidden layer and an output layer). It should be noted that a FFNN can have more than one hidden layer. The three layers serve the following functions:

- The input layer receives the user input or data input from a source.
- Each hidden layer processes the input layer's data net sum as well as its bias. Then it runs it through an activation function. No node on the hidden layer is interconnected.
- Each output layer neuron then receives the data input from each individual hidden layer neuron, to compute the net sum plus bias, before calculating the result by passing the net sum through an activation function.

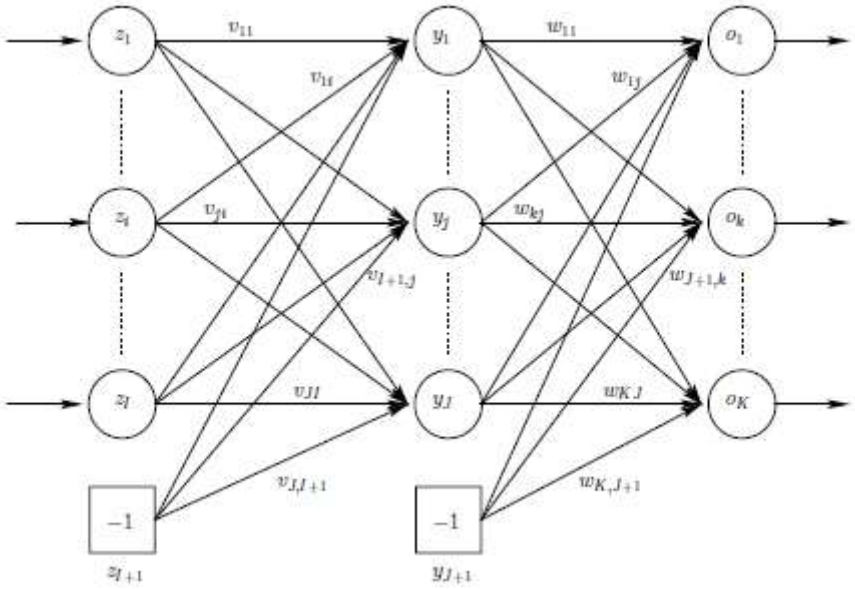


Figure 3.3: Feedforward Neural Network [37, p.28]

This process can be mathematically expressed as follows: the output for any input pattern Z_p can be calculated by a single forward pass through the network. Thus for each output O_k we have:

$$\begin{aligned} O_{k,p} &= f_{ok}(net_{ok,p}) \\ &= f_{ok}\left(\sum_{j=1}^{J+1} \omega_{kj} f_{y_j}\left(net_{y_{j,p}}\right)\right) \\ &= f_{ok}\left(\sum_{j=1}^{J+1} \omega_{kj} f_{y_j}\left(\sum_{i=1}^{I+1} v_{ji} z_{i,p}\right)\right) \end{aligned} \qquad (3.5)$$

Where f_{o_k} and f_{y_j} are activation functions for the hidden unit y_j and output unit o_k.There are different activation functions and these are highlighted in section 3.6. $\omega_{kj}\ and\ v_{ji}$ are the weights between the output unit o_k and hidden unit y_j, with $\omega_{k,J+1}\ and\ v_{j,I+1}$ being the weights from the biases to the respective hidden and output units [20, p.28].

In order for the neural network to predict an output from a set of input vectors, it needs to adjust its weight to enable its output to effectively provide an accurate result from the inputs. This is done by the use of what is known as neural network learning. The next section will describe neural network learning for a feedforward neural network.

3.5 Neural Network Learning

Before a neural network can perform any function, it first needs to be taught how to process inputs in order to produce a satisfying output. It does this by adjusting weights and bias values until a certain criterion is met. The criterion is normally a measure of the neural network's output versus a target value. The process of changing the weight values to approximate a target value is known as neural network learning. The ultimate aim of learning is to enable a neural network to accurately output a result from input data which was not part of the training set. In order to achieve optimal performance, network training

can be grouped into 4 groups namely: learning paradigm, learning rate and learning algorithm [22]. In this section the learning paradigms, learning algorithm and activation functions used for learning of feedforward neural networks will be discussed.

3.5.1 Learning Paradigm

Learning paradigms can be grouped into two main groups, namely:

- Unsupervised Learning
- Supervised Learning.

Where unsupervised learning was effectively defined by Gao [22] as follows:

"Unsupervised learning is based on local information. Data presented to the network are self-organized to detect their collective properties via competitive learning and hebbian learning. In competitive learning all output nodes compete with one another for the right to respond to a request. Hebbian learning minimizes the same error function that is equivalent to the sum of squared distances between each training case and a linear subspace of the input space (with distances measured perpendicularly), as an auto associative network with a linear hidden layer (Sarle, 2002). However, the distinction between supervised and unsupervised learning is not always so clear-cut because in unsupervised learning a summarized distribution of probability can be used to make predictions [22]."

With regard to this project, supervised learning will be evaluated in more detail in the next section due to the nature of the locomotive's human machine interface, which does not allow direct monitoring of sensor readings but rather recording data for a specified time, which supports supervised learning.

3.5.1.1 Supervised Learning

Supervised learning utilizes a data set which is also referred to as a training set which consists of input and target (Desired output) vectors associated with each input vector. A neuron or neural network is then provided with the training set where training then takes place by adjusting the weight values to minimize the error between the neuron or neural network output and the target output values.

Engelbrecht [20, p.37] classifies neural network learning further as follows:

- Stochastic/online Learning: where the weights are updated after each pattern presentation
- Batch/Offline learning: for this type of learning the weight updates are accumulated and used to adjust the weights only after all the training patterns have been presented.

Stochastic learning has both advantages and disadvantages and Gao [22] gives a thorough account thereof as follows:

"It is advantageous in that non-stationary environments where the best model gradually changes over time can be better monitored. There is less chance for noise to develop into local minima. Online learning is often faster than off-line learning if the training dataset has a high degree of redundancy (Orr et al., 1999). The downside of online learning is that it is

unable to take advantage of many network optimization measures that are widely practised in off-line learning, such as multiple random initialization, computation of a minimum of the objective function to any desired prevision, conjugate and second-order gradient methods, support vector machines, and bayesian methods. Thus, off-line learning is easier and more reliable than online learning. A compromise can be reached by combing online learning with off-line learning in which the weights are updated only after certain number of data points [22]."

Looking at the advantages and disadvantages as stated above, batch/offline learning will be implemented in the training of the neural network in this project. To apply batch learning, a learning algorithm is needed; thus it is necessary to discuss different learning algorithms used to train feedforward neural networks. This will be done in the next section.

3.5.1.1.1 Supervised Learning Algorithms

In this project a neural network will be used as an estimator, which estimates state variables (Sensor Measurements) of a control system. It could be noted that the performance and acceptability of a neural estimator is largely related to the performance of the learning algorithm used. There are a number of different training algorithms used for training feedforward neural networks of which we will evaluate the following due to their popularity in FDI system design.

- Gradient Descent Optimization

The Gradient Descent Optimization performance will be evaluated in Chapter 4, where it will be used to train a feedforward neural network to model the excitation system of a GE D-E locomotive's excitation control System. But first a discussion on the basic principle of operation of the back propagating learning algorithm will be done. Examples where the GD training algorithm was used will also be presented.

3.5.1.1.2 Gradient Descent Optimization

Gradient descent optimization has led to one of the most popular learning algorithms used for neural network learning, namely the backpropagation algorithm. Owing to its simplicity and accuracy, it has been used in many modeling and FDI applications, performed by training of neural networks. These applications include the following:

- Taplak, Uzmay and Yıldırım [24], used the GD backpropagation learning algorithm with momentum to model a rotor bearing system.
- The modeling of a space shuttle main engine, which used a feedforward neural network training with the use of the generalized delta rule, which is a type of GD back propagation [26]
- Napolitano, Silvestri, Windon, Casanova and Innocentri [27] used an extension of the back propagation learning algorithm to perform on-line sensor validation.

From the above mentioned applications, it could be noted that the basis of all neural networks used for modeling of systems used in the specified FDI systems, the GD learning algorithm or extensions thereof were used. Where a standard GD learning algorithm works as follows: each learning iteration consists of a feedforward pass and the backward

propagation. The feedforward pass simply calculates the output value as per equation 3.5. The backward propagation propagates the error signal back from the output layer towards the input layer. Weights are then updated in accordance to the error signal.

The error signal is determined using the sum of squared errors and is defined as

$$\varepsilon = \frac{1}{2} \sum_{p=1}^{p} (t_p - y_p)^2 \tag{3.6}$$

Where $t_p = desired\ output$

$y_p = actual\ output$

$p = number\ of\ patterns\ taught\ (training\ set)$

In order to minimise the error, the gradient of the error is then plotted and the weights are determined and the weight is moved along the negative gradient; therefore decreasing error. This is done by taking partial derivative of the error function with respect to w_i as shown in equation

$$\frac{\partial \varepsilon}{\partial w_{k,j}} = \frac{\partial \varepsilon}{\partial w_{ok}} \frac{\partial Ok}{\partial net_{ok}} \frac{\partial \varepsilon}{\partial w_{k,j}}$$

$$\frac{\partial \varepsilon}{\partial w_{k,j}} = -(t_k - o_k) \times f'(net) \times y_j \tag{3.7}$$

Let's assume the output activation function is linear, thus $f'(net) = 1$, then the above equation can be written as follows:

$$\frac{\partial \varepsilon}{\partial w_{k,j}} = -(t_k - o_k) \times y_j \tag{3.8}$$

If the weight update is given as

$$w_{k,j}(t+1) = w_{k,j}(t) - \left(\eta \times \frac{\partial \varepsilon}{\partial w_i}\right) \tag{3.9}$$

Then

$$w_{k,j}(t+1) = w_{k,j}(t) + \eta \times (t_k - o_k) \times y_j \tag{3.10}$$

When the hidden layer is added to this, the weights' updates of the hidden layer are calculated as follows:

$$v_{j,i}(t+1) = v_{j,i}(t) - \left(\eta \times \frac{\partial \varepsilon}{\partial v_{j,i}}\right) \tag{3.11}$$

Where $\frac{\partial \varepsilon}{\partial v_{j,i}}$ can be calculated as:

$$\frac{\partial \varepsilon}{\partial v_{j,i}} = \frac{\partial \varepsilon}{\partial o_k} \frac{\partial o_k}{\partial net_{ok}} \frac{\partial net_{ok}}{\partial f_{yj}} \frac{\partial f_{yj}}{\partial net_{yj}} \frac{\partial net_{yj}}{\partial v_{j,i}}$$

$$\frac{\partial \varepsilon}{\partial v_{j,i}} = -(t_k - o_k) \times o_k \times (1 - o_k) \times w_{k,j} \times y_j \times (1 - y_j) \times x_i \tag{3.12}$$

Thus the weight update is calculated as follows:

$$v_{j,i}(t+1) = v_{j,i}(t) - \eta \sum(-(t_k - o_k) \times o_k \times (1 - o_k) \times w_{k,j} \times y_j \times (1 - y_j) \times x_i) \quad (3.13)$$

Training is stopped by monitoring the neural network's mean squared error and the generalization errors. These can be calculated as follows:

$$\varepsilon_T = \frac{\sum_{p=1}^{PT} \sum_{k=1}^{K} (\frac{1}{2}(t_{k,p} - o_{k,p})^2)}{P_T K} \quad (3.14)$$

where ε_T is the training error, where P_T is the total number of training patterns in the training set and K is the number of output units. The generalization error ε_G is approximated in the same way.

$$\varepsilon_G = \frac{\sum_{p=1}^{PG} \sum_{k=1}^{K} (\frac{1}{2}(t_{k,p} - o_{k,p})^2)}{P_G K} \quad (3.15)$$

The only difference is that the first summation is over P_G patterns, which are the patterns of the generalization data set. Training is then stopped when ε_T still decreases but ε_G increases.

As mentioned above, the performance of the neural network is dependent on the performance training algorithm whereas the training algorithm has a set of performance parameters itself. These will be discussed in Chapter 4, where this algorithm will be used to train a neural network to model the excitation control system of the locomotive.

From the equations above it could be noted that there were references made to $f(net)$; this is called an activation function and is explained in depth in the next section.

3.6 The Activation Function

The activation function f_{AN} was first introduced in section 3.1.2, equation 3.1. From there it could be noted that the activation function f_{AN} receives the net inputs and the bias, from which it determines the output of a neuron. Activation functions are generally monotonically mappings where (Except for the linear function):

$$f_{AN}(-\infty) = 0 \; or \; f_{AN}(-\infty) = -1$$

$$and$$

$$f_{AN}(\infty) = 1$$

Theoretically any differential function can be used as an activation function, but it should be noted that the linear and sigmoid functions are used the most in neural networks [21, p.14]. A brief overview of the most typical ones used will be highlighted in this section.

3.6.1 The Linear Function

The linear activation function, also known as the identity function, is a flow through mapping of a neuron's potential to its output. These functions are normally used in the

output layer of a neural network used for forecasting or predictions [21, p.14]. Figure 3.4 below shows a linear activation function.

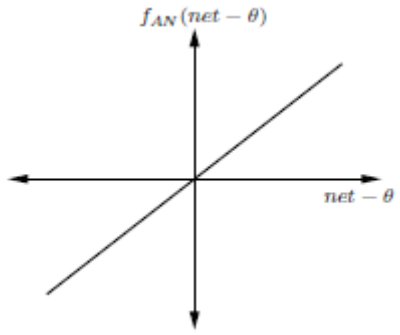


Figure 3.4: The Linear Activation Function [20]

Where the linear activation function can be mathematically expressed as follows:

$$f_{AN}(net-\theta) = \lambda(net-\theta)$$

The slope of the function is controlled or set via the constant λ and produces a linearly modulated output.

3.6.2 The Step Function

Figure 3.5 shows a step activation function. It could be noted that the output is a specific number. Thus the step function can be considered to be binary and is mostly used for binary classification schemes. It could also be used as a feature detector to solve clustering of classification problems.

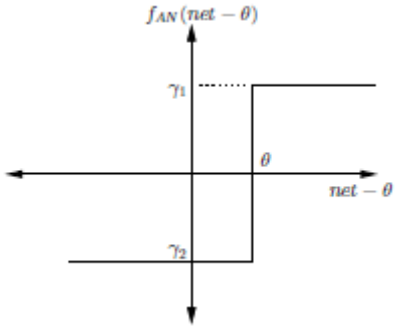


Figure 3.5: The Step Activation Function

Where the step function can be mathematically expressed as:

$$f_{AN}(net-\theta) = \begin{cases} \gamma_1 \; if \; net \geq \theta \\ \gamma_2 \; if \; net < \theta \end{cases}$$

Where the step function produces one of two scalar values, which is dependent on the value of the threshold θ. For binary output code, mentioned above $\gamma_1 = 1 \; and \; \gamma_2 = 0$ and if a bipolar output is needed the scalar values are set as $\gamma_1 = 1 \; and \; \gamma_2 = -1$. [21, p.15]

3.6.3 The Ramp Function

The ramp function is a combination of the step and identity functions. Figure 3.6 below represents the ramp function:

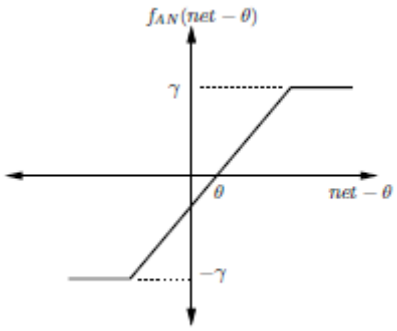


Figure 3.6: The Ramp Activation Function

Where the ramp function can be mathematically expressed as follows:

$$f_{AN}(net-\ \theta) = \begin{cases} \gamma & if\ \ net-\theta \geq\ \epsilon \\ net-\ \theta & if\ \ -\epsilon < net-\theta < \epsilon \\ -\gamma & if\ \ net-\theta\ \leq\ -\epsilon \end{cases}$$

The ramp function works on the principle that as long as the activation is smaller than or equal to the threshold value $-\epsilon$, the output will be the scalar value $-\gamma$. If the activation is smaller than the threshold value ϵ but bigger than $-\epsilon$ the output will be $net-\ \theta$. Then if the activation is greater than or equal to ϵ, then the output will be equal to the output scalar γ [20], [21].

3.6.4 The Sigmoid Function

The sigmoid activation functions are differential functions which map a neuron's potential outputs to a range of values. The sigmoid function can further be described as a continuous version of the ramp function. Figure 3.7 illustrates the sigmoid activation function.

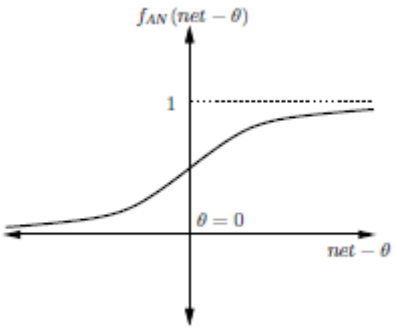


Figure 3.7: The Sigmoid Activation Function

The mathematical representation of the sigmoid function is as follows:

$$f_{AN}(net-\theta) = \frac{1}{1+e^{-\lambda(net-\theta)}}$$

Where $f_{AN}(net) \in (0,1)$and the parameter λ controls the steepness of the function and is usually equal to 1. The sigmoid function is normally used in the hidden layer of a neural network to scale the inputs between 0 and 1, after which a linear function is used to provide a scalar output [20].

3.6.5 The Hyperbolic Tangent Function

The Hyperbolic Tangent activation functions can be proven to be linearly related to the logistic activation functions. Thus forecasts or predictions produced by using the logistic activation functions should be close to those produced by hyperbolic tangent activation functions. Research shows that the use of hyperbolic tangent activation functions takes up less training time compared to logistic activation functions [21, p.15].

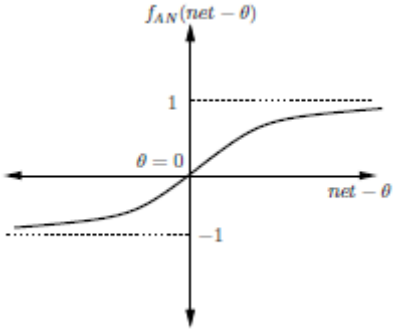


Figure 3.8: The Hyperbolic Tangent Function

Considering Figure 3.8 the mathematical representation of the hyperbolic tangent function can be given as follows:

$$f_{AN}(net-\theta) = \frac{2}{1+e^{-\lambda(net-\theta)}} - 1$$

The output of the hyperbolic tangent function is in the range (-1, 1).

3.6.6 The Gaussian Function

The Gaussian activation function can be expressed by the following equation:

$$f_{AN}(net-\theta) = e^{-(net-\theta)^2/\sigma^2}$$

Where $net-\theta$ is the mean and σ is the standard deviation of the Gaussian distribution. Figure 3.9 below illustrates a graphical representation of the Gaussian function.

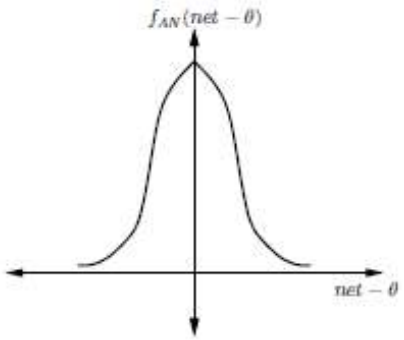


Figure 3.9: The Gaussian Function

3.7 Conclusion

From the above literature review of different applications of neural networks in the general field and more importantly the field of FDI, it was noted that the proposed system which required the use of a model to model the excitation control system of the locomotive could be done with the use of artificial neural networks. Chapter 2 indicated that the two models required by the proposed system could be realized with the use of dedicated observers and Chapter 3 showed that these observers could be designed with the use of a feedforward neural network, trained using supervised learning and the gradient descent training algorithm. The training algorithm has different parameters affecting its performance which will be discussed in the next chapter.

Chapter 4 – Neural Network Parameter Setup for the Modelling of the Locomotive's Excitation System

Chapters 2 and 3 indicated that an artificial neural network could be used as a replacement to the classical mathematical model of a process or plant. This makes modelling of processes or plants much easier when no analytical data of a plant is available. Neural networks' ability to generalize and filter out noise makes them perfect for this function. Thus in this chapter a neural network model of the locomotive's excitation control system will be developed. The gradient descent algorithm will be used as the primary training algorithm within this project due to its successful implementation in numerous modelling and FDI system developments [23],[24],[25],[26].

The gradient descent training algorithm has a number of different training parameters which increase the performance of a neural network. This chapter will highlight some of these performance measures and provide an overall view of parameters which will be considered when designing the neural network to model the system using the gradient descent training algorithm.

The chapter is divided into 5 main sections, namely: Overview of the performance measurements of a neural network, data preparation *(where an analysis of the excitation control system was done to determine the neural network's input-output configuration),* Learning Rate and momentum, training stopping conditions and network configuration.

4.1 Overview of Performance Measures of a Neural Network to consider

The performance of a neural network can be divided into two main groups namely; accuracy and complexity [20].

4.1.1 Accuracy

Accuracy of a neural network is extremely important and is normally measured in terms of its generalization ability. Generalization can be defined as the ability of a neural network to provide an output from patterns which were not part of the training set. The objective of training a neural network is to produce a low generalization error. Chapter 3 highlighted the performance measurements based on the generalization and training errors. It was noted that the training should stop at the instant where the generalization error increases while the training error still decreases. If training is not stopped at this instant, overfitting can occur [20, pp.95-98].

Overfitting is an important factor, due to the fact that when overfitting occurs, the neural network memorizes the training patterns and loses its ability to generalize. Thus the NN loses its ability to predict outputs from data not seen during training. Factors which were found to lead to overfitting are:

- Too Large Network Architecture
- Training Time – NN trained too long.

The point of overfitting is described as the point at which the training error continues to decrease while the generalization error starts to increase. Engelbrecht [20] stated that it is at this stage that training should be stopped and that if a set number of iterations or the training error alone, were to be used as a stopping condition, overfitting would occur.

To detect overfitting the data set is divided into three sets, namely: training set, generalization set and the validation set, where the validation set is used to estimate the generalization error. Determining overfitting is not straight forward due to both the training and validation errors fluctuating. Thus overfitting can be detected when the following conditions are met:

$$\varepsilon_v > \bar{\varepsilon}_v + \sigma\varepsilon_v$$

Where ε_v is the MSE on the validation set and $\bar{\varepsilon}_v \; and \; \sigma\varepsilon_v$ are the average of the MSE and the standard deviation of the validation set.

It could be noted from the above mentioned that the accuracy of a neural network is a trade-off between the training and generalization errors.

4.1.2 Complexity

There are two main factors which affect the computational complexity of a neural network namely:

- The Network Architecture: the larger the architecture the more feedforward calculations are needed and the more training calculations are needed per pattern during
- The Training Set Size: the larger the training set size the more training patterns are presented per epoch.

Both of these factors have a huge impact on the training time.

4.1.3 Performance Factors

In order to develop a successful neural network model of the excitation system, keeping in mind the performance measures discussed above, the following performance factors need to be evaluated:

- Data Preparation
- Learning Rate and Momentum
- Stopping Condition
- Network Configuration.

The remainder of the chapter will evaluate the above mentioned parameters in order to design a neural network model of the locomotive's excitation control system. The gradient descent training algorithm was discussed in Chapter 3, but before implementing the training algorithm, a system analysis will be done.

4.2 Data Preparation

Data preparation is the most important step in neural network training, as it defines which parameters from the data set should be used as inputs and outputs. Chapter 2 highlighted two types of dedicated observer schemes, where one was used for the detection of sensor faults and another for actuator faults. For the detection of sensor faults each observer used all the inputs and just one output to detect faults. The number of observers equalled the number of outputs which was also equal to the number of sensors. Actuator faults were isolated with the use of an observer system where each observer used one input and all outputs. The DOS scheme allows for the localization of multiple faults for either sensor or actuator faults [8].

In this section an analysis of the excitation control system will be done, in an effort to determine which parameters can be used to estimate other parameters. The controlling variables will be used to estimate all output values (sensor readings).

4.2.1 Locomotive's Excitation Control System

The GE U26C Diesel-Electric locomotive which is also known as the BSS locomotive is DC powered, and operates with a constant horsepower type excitation control system named "Micro-CHEC". The excitation system utilizes a microcomputer to optimize performance and provide on-board diagnostics. Figure 4.1 below shows the Diesel-Electric locomotive's energy flow. The excitation system is at the heart of the operation and controls the electrical power to the traction motors.

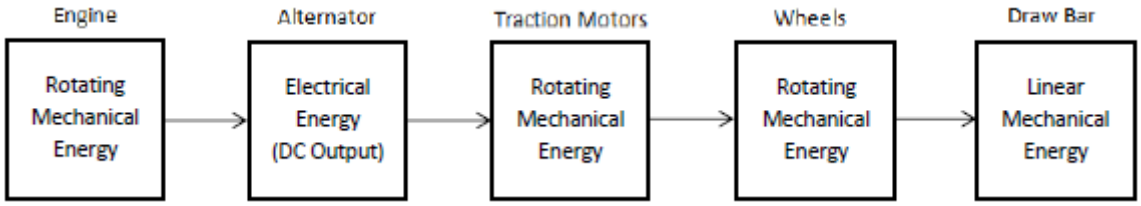


Figure 4.1: Diesel-Electric Locomotive's Energy Flow

The process starts with the conversion of heat energy from the chemical combustion of diesel fuel in the engine into work *(linear force x piston travel, changed to rotational force x circular movement via crankshaft action).* The rotational energy is then converted into electrical energy by the alternator, which then in turn supplies the traction motors with electrical power. The energy is then converted back to rotational mechanical energy and exerted onto the wheels, which provides linear forward mechanical energy.

The GE D34 class D-E locomotive on which the project is based, utilises 8 power levels where the locomotive's engine speed is at max RPM from notches 5-8, meaning the amount of fuel injected is increased while the RPM remains constant, in order to obtain higher power levels. The engine's RPM and torque are controlled via a mechanical engine governing unit, known as the Governor. The Governor controls the engine's RPM through 4 electrically actuated solenoid valves. Table 4.1 below shows the Governor's RPM selection via the solenoid valves.

Table 4.1: Governor Control Table

Condition	Throttle Notch Command	Governor Solenoid Valves				Equiv Notch	Engine Speed (RPM)
		AV	BV	CV	DV		
Shutdown	Neutral	0	0	0	1		0
Idling (Ready)	Neutral	1	0	0	0	2	519-549
Engine Speed Control Above Idling	1	1	1	0	0	4	685-702
	2	1	1	0	0	6	865-873
	3	0	1	1	0	7	960-968
	4	0	1	1	0	7	960-968
	5	1	1	1	0	8	1045-1055
	6	1	1	1	0	8	1045-1055
	7	1	1	1	0	8	1045-1055
	8	1	1	1	0	8	1045-1055
MU Eng. Stop	ANY	0	0	0	1		

It could be noted that the solenoid sequence is determined on the power or throttle notch selection, manually done by the train operator.

The Governor also incorporates a load sensing function where changes in the load conditions trigger a hydraulic mechanism that regulates oil pressure to either one side or the other of a vane servo motor which is in turn coupled to a circular load control potentiometer. The device provides the BSS control system with a voltage feedback, indicating the conditional mechanical power of the engine, hence balancing the electrical output power from the alternator and the mechanical power of the engine.

The alternator's rotor shaft is directly coupled to the engine's crankshaft, hence rotating at the same RPM as the engine so that the constant mechanical engine power is converted to constant electrical power for each of the 8 power levels. The alternator is excited by a separately excited DC generator, known as the exciter. The exciter's armature is mechanically rotated by the alternator via a gear train, with the exciter's field excited via the BSS excitation control system. Figure 4.2 below shows a block diagram of the BSS excitation control system.

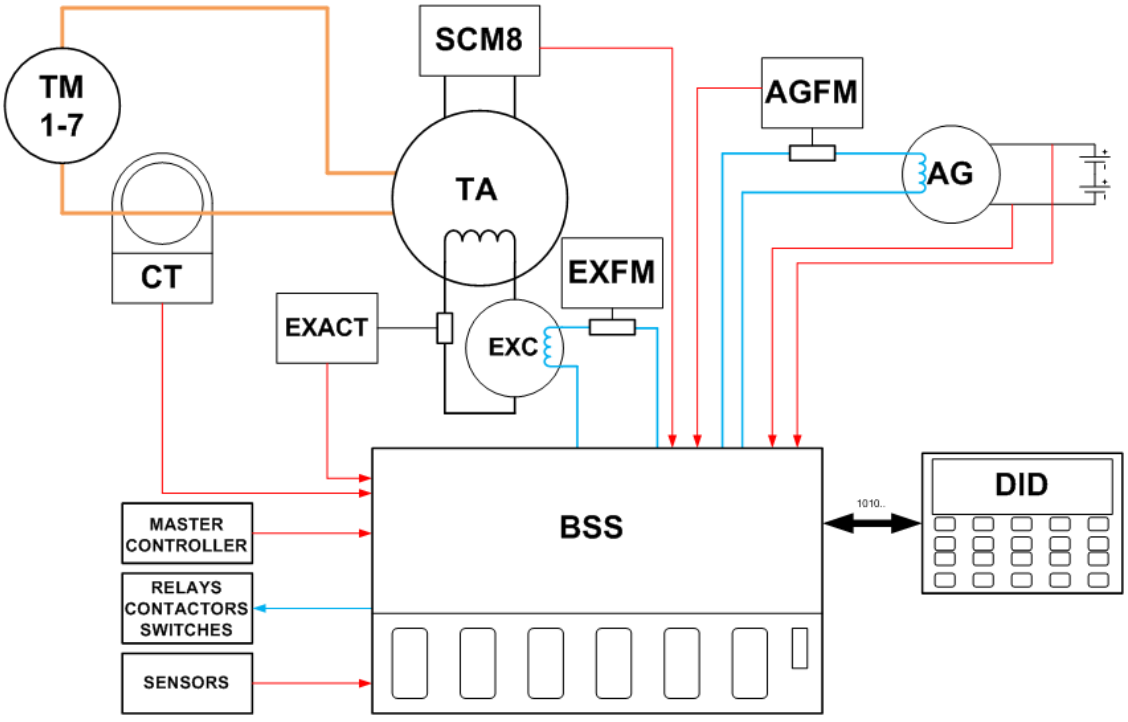


Figure 4.2: BSS Excitation Control System

Figure 4.2, indicates all the sensor feedbacks, used within the excitation system. The sensor feedbacks used are as follows:

- EXACT – Exciter Armature Current
- EXFM – Exciter Field Current
- SCM8 – Rectified Alternator Voltage
- TM1-7 CT's – Traction Motor Currents.

Figure 4.2 further illustrates the basic feedbacks used in the control of the exciter's field current. The next section will provide an analysis of the electrical power generating control system, which is controlled via the exciter's field current.

As per Figure 4.2, it could be noted that the excitation system can be grouped into three main components, namely: Separately Excited DC Generator, 3 Phase Alternator and 6 Separately Excited DC Motors. In order to further analyse the operation or principle of control of the electrical power generating system, each component's operation will be highlighted and analysed in the succeeding sections.

4.2.1.1 Separately Excited DC Generator (Exciter)

In a separately excited DC generator the field coils are excited by an external DC power source. This source maybe any dc source such as a battery, another dc generator or in this case the BSS control system which supplies a voltage across the field winding.

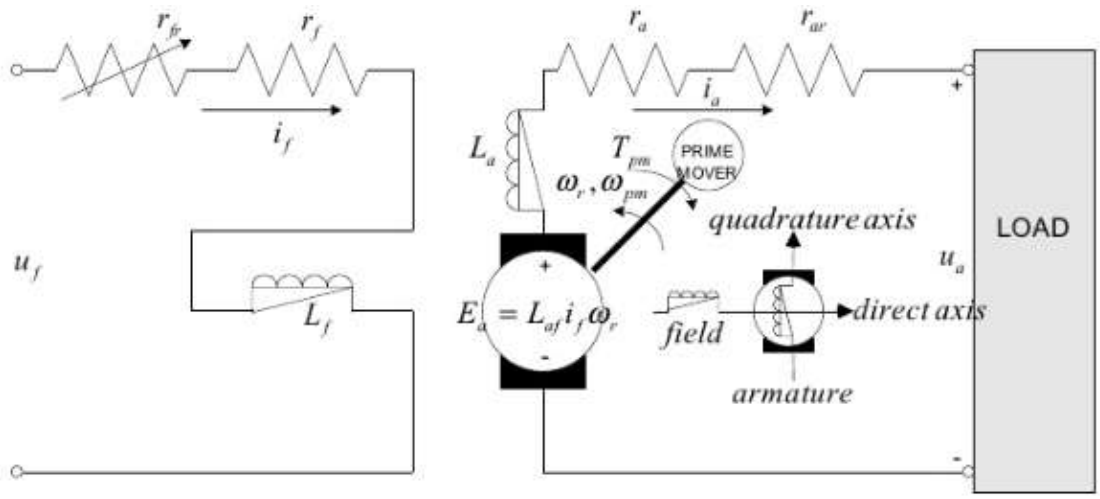


Figure 4.3: Separately Excited DC Generator Circuit

Figure 4.3 above shows the steady state model of a separately excited dc generator where:

I_f = Field Current
I_a = Armature Current
E_a = Generated EMF
I_L = Load Current
L_f = Field Coil Inductance
L_{af} = Mutual Inductance
R_f = Field Coil Resistance
R_a = Armature Coil Resistance
L_a = Armature Coil Inductance
U_a = Generated Voltage

$$U_f = I_f R_f + L_f \frac{dI_f}{dt} \quad (4.1)$$

$$U_a = -I_a R_a - L_a \frac{dI_a}{dt} + L_{af} I_f W_r \quad (4.2)$$

Where L_{af} is a constant which is equal to $\frac{N_f N_a}{R_m(90°)}$ and U_a is connected to the field windings of the three phase AC alternator. The voltage output of the alternator is increased with an increase in the speed rotation, thus increasing the number of force lines being cut per second. As the field excitation increases, in this case U_a, the magnetic field is increased to the point of saturation, for a given rotational speed.

In order to effectively control the alternator's voltage output, it needs to be driven at a constant speed while varying the field excitation. The alternator's frequency is a direct function of the field current and rotational speed, thus making it more feasible to drive the alternator at a constant speed, whilst varying the excitation current. Figure 4.4 below shows the principle of operation of a 3 phase synchronous ac alternator as indicated by [28, pp.17-19].

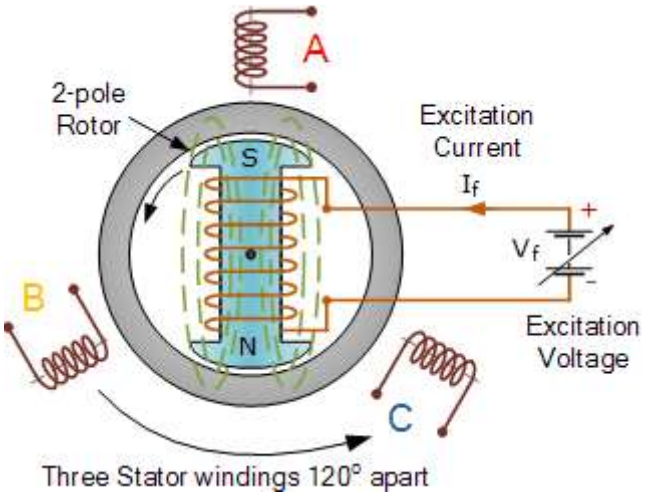


Figure 4.4: Three Phase Synchronous AC Machine [28, p.17]

The V_f displayed in Figure 4.4, can be defined by the following:

$$V_f = I_r R_r + \frac{dI_r}{dt} L_r \quad (4.3)$$

Where

V_f = Alternator Field Excitation Voltage

R_r =Alternator Excitation field resistance (Rotor Field Resistance)

L_r =Alternator Excitation field Inductance (Rotor Field Inductance)

As the exciter is electrically connected to the alternator's field winding it could be noted that $V_f = U_a$ and $I_r = I_a$. Equation 4.3 can then be rewritten as follows:

$$U_a = I_a R_r + \frac{dI_a}{dt} L_r \quad (4.4)$$

Substituting equation 4.4 into equation 4.2 gives us:

$$I_a R_r + \frac{dI_a}{dt} L_r = -I_a R_a - L_a \frac{dI_a}{dt} + L_{af} I_f W_r$$

$$I_a = \frac{L_{af} I_f W_r - \frac{dI_a}{dt}(L_r + L_a)}{(R_r + R_a)} \quad (4.5)$$

In the above equation it could be noted that the following are constants: $L_a, L_r, L_{af}, R_a, R_r$. Thus the only varying factors which affect the alternator's rotor current is the exciter's field current plus the angular velocity of the exciter's armature. At this point it could be noted that by monitoring the exciter's field current, I_f, as well as its armature's angular velocity, the alternator's rotor excitation current can be calculated.

4.2.1.2 Three Phase Synchronous Generator

To effectively model the electrical operation of the alternator, building on equation 4.5, Figure 4.5 below will be used to illustrate schematically the cross section of a three phase cylindrical rotor synchronous machine in order to derive dynamic equations for the alternator phase voltages.

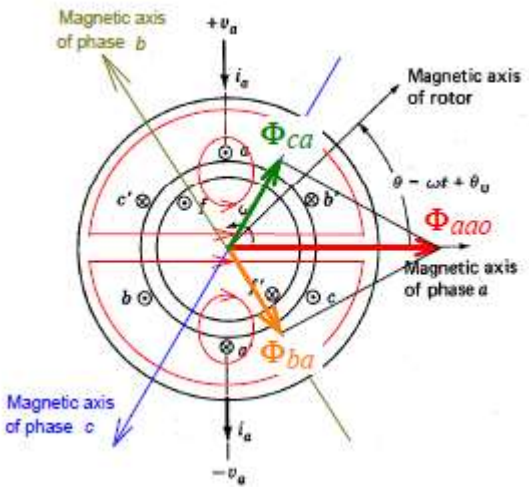


Figure 4.5: Schematic Diagram of a Three Phase Cylindrical Rotor Synchronous Machine [28, p.17]

Figure 4.5 represents the three stator windings (aa', bb' and cc') which produce sinusoidal mmf and flux density waves rotating in the air gap whereas ff' is used to represent the field winding on the rotor. This winding represents a distributed winding which produce sinusoidal mmf and flux density waves centred on its magnetic axis and rotating with the rotor [28, pp.17-20].

With reference to Figure 4.5 and [28], the electrical circuit equations for the three phase windings can be expressed with the use of Kirchhoff's Voltage law as follows:

$$V_a = -R_a i_a + \frac{d\lambda_a}{dt} \quad (4.6)$$

$$V_b = -R_b i_b + \frac{d\lambda_b}{dt} \quad (4.7)$$

$$V_c = -R_c i_c + \frac{d\lambda_c}{dt} \quad (4.8)$$

Where $V_a, V_b and\ V_c$ represent the voltages across the windings and $R_a, R_b and\ R_c$ being the winding resistances. $\lambda_a, \lambda_b and\ \lambda_c$ represent the total flux linkages of the windings of phases a, b and c. Assuming a symmetric three phase stator winding:

$$R_a = R_b = R_c$$

$\lambda_a, \lambda_b and\ \lambda_c$ can be expressed as follows:

$$\lambda_a = -[\lambda_{aa} + \lambda_{ab} + \lambda_{ac}] + \lambda_{af} = -[L_{aa}i_a + L_{ab}i_b + L_{ac}i_c] + L_{af1}i_f \quad (4.9)$$

$$\lambda_b = -[\lambda_{ba} + \lambda_{bb} + \lambda_{bc}] + \lambda_{bf} = -[L_{ba}i_a + L_{bb}i_b + L_{bc}i_c] + L_{bf}i_f \quad (4.10)$$

$$\lambda_c = -[\lambda_{ca} + \lambda_{cb} + \lambda_{cc}] + \lambda_{cf} = -[L_{ca}i_a + L_{cb}i_b + L_{cc}i_c] + L_{cf}i_f \quad (4.11)$$

Where the mutual and self-inductances are expressed as follows:

$$L_{aa} = L_{bb} = L_{cc} = L_{aao} + L_{al}$$

$$L_{ab} = L_{ba} = L_{ac} = L_{ca} = -L_{aao}/2$$

$$L_{af1} = L_{afm}\cos\theta$$

$$L_{bf} = L_{afm}\cos(\theta - 120°)$$

$$L_{cf} = L_{afm}\cos(\theta - 240°)$$

When considering a balanced three phase machine, $L_{aao} = \frac{\Phi_{aao}}{i_a}$, $L_{al} = \frac{\Phi_{al}}{i_a}$, Φ_{aao} is the flux that links all three phase windings. Φ_{al} represent the flux that links only the phase winding and $\theta = \omega t + \theta_0$ [28, p.18].

If the stator windings outputs balanced three phase currents, then:

$$i_{aphase} + i_{bphase} + i_{cphase} = 0$$

The total flux linkage of the phase '*a*' winding can be further written as follows:

$$\lambda_a = -(L_{aao} + L_{al})i_a + \frac{L_{aao}i_b}{2} + \frac{L_{aao}i_c}{2} + L_{afm}i_f\cos(\omega t + \theta_0)$$

$$= -(L_{aao} + L_{al})i_a + \frac{L_{aao}}{2}(i_b + i_c) + L_{afm}i_f\cos(\omega t + \theta_0)$$

$$= -(L_{aao} + L_{al})i_a - \frac{L_{aao}}{2}(i_a) + L_{afm}i_f\cos(\omega t + \theta_0)$$

$$= -\left(\frac{3L_{aao}}{2} + L_{al}\right)i_a + L_{afm}i_f\cos(\omega t + \theta_0)$$

$$= -L_s i_a + L_{afm}i_f\cos(\omega t + \theta_0) \quad (4.12)$$

The total flux linkages of phases *b* and *c* ($\lambda_b\ and\ \lambda_c$) can then be written as:

$$\lambda_b = -L_s i_b + L_{afm}i_f\cos(\omega t + \theta_0 - 120°) \quad (4.13)$$

$$\lambda_c = -L_s i_c + L_{afm}i_f\cos(\omega t + \theta_0 - 240°) \quad (4.14)$$

[28] highlighted that in this method the three phase windings can be mathematically de-coupled, when considering a balanced three phase synchronous machine; hence it is only necessary to solve the circuit equation from one phase. When substituting equation 4.12 into 4.6 we obtain

$$V_a = -R_a i_a + \frac{d(-L_s i_a + L_{afm} i_f \cos(\omega t + \theta_0))}{dt}$$

$$V_a = -R_a i_a + \frac{d(-L_s i_a)}{dt} + \frac{d(L_{afm} i_f \cos(\omega t + \theta_0))}{dt}$$

$$V_a = -R_a i_a - L_s \frac{d(i_a)}{dt} + L_{afm}[\cos(\omega t + \theta_0)\frac{di_f}{dt} - i_f \omega \sin(\omega t + \theta_0)] \quad (4.15)$$

The line to line voltage can then be written as follows:

$$V_{aline} = \sqrt{3}\{-R_a i_a - L_s \frac{d(i_a)}{dt} + L_{afm}[\cos(\omega t + \theta_0)\frac{di_f}{dt} - i_f \omega \sin(\omega t + \theta_0)]\} \quad (4.16)$$

Where i_f is equal to i_a and when the phase current i_a in the above equation is labelled as i_{aphase}, and $R_a = R_{aStator}$ then the above equation can be written as:

$$V_{aline} = \sqrt{3}\left\{-R_{astator} i_{aphase} - L_s \frac{d(i_{aphase})}{dt} + L_{afm}\left[\cos(\omega t + \theta_0)\frac{d\left[\frac{L_{af} I_f W_r - \frac{dI_a}{dt}(L_r + L_a)}{(R_r + R_a)}\right]}{dt} - \left[\frac{L_{af} I_f W_r - \frac{dI_a}{dt}(L_r + L_a)}{(R_r + R_a)}\right]\omega \sin(\omega t + \theta_0)\right]\right\}$$

By analysing the above equation it could be noted that the line to line voltage is dependent on the following variables:

$$i_f = Exciter's\ Field\ current$$

$$W_r = Exciter's\ Rotor\ Angular\ Velocity\ in\ rad/sec$$

$$\omega = electrical\ angular\ velocity\ in\ electrical\ rad/sec$$

ω can be expressed as follows [28, p.5]:

$$\omega = \frac{P}{2}\omega_m \quad (4.18)$$

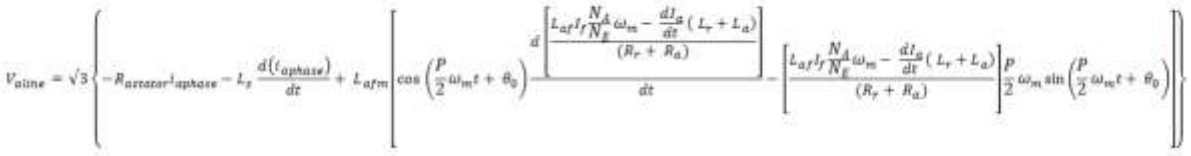
Where $P = Number\ of\ poles\ of\ the\ alternator$ and $\omega_m = mechanical\ rad/sec$. ω in the equation 4.16 can then be replaced with the mechanical angular velocity; hence V_{aline} is dependent on $i_f, W_r\ and\ \omega_m$. Where $W_r\ and\ \omega_m$ can be related by considering the mechanical linkages between the primary mover (*diesel engine*), alternator and the exciter. Where the primary mover mechanically drives the alternator which then in turn drivers the Exciter via a gear train:

$$\frac{N_A}{N_E} = \frac{W_r}{\omega_m} \quad (4.19)$$

Here $\frac{N_A}{N_E}$ is the gear ratio which is a constant, thus W_r can be expressed in terms of $\omega_m$as follows:

$$W_r = \frac{N_A}{N_E}\omega_m \tag{4.20}$$

When substituting the above mentioned equations into equation 4.17, V_{aline} is equal to the following:



[28, p.19] states that the above equation can be expressed in the steady state in terms of voltage and current phasors as:

$$\begin{aligned} V_{as} &= E_a - (R_a + j\omega L_s)I_a \\ &= E_a - (R_a + jX_s)I_a \end{aligned} \tag{4.21}$$

Where $X_s = \omega L_s$, which is known as the synchronous reactance and E_a can be expressed as follows:

$$E_a = j\frac{\omega L_{afm} I_f}{\sqrt{2}} = j\frac{2\pi}{\sqrt{2}} f k_w N_{ph} \Phi_f \tag{4.22}$$

Where

$I_f = Field\ Current$
$\Phi_f = Rotor\ Magnetic\ Flux\ in\ air\ gap$

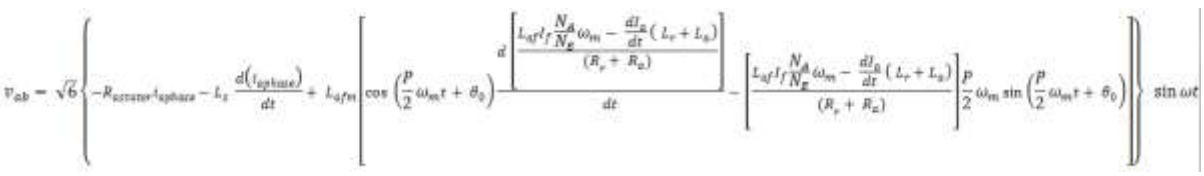
From the analysis of the above equations it could be noted that V_{aline} is dependent on two main variables, namely, $i_f\ and\ \omega_m$ respectively as all the other values are constants. Thus by monitoring $i_f\ and\ \omega_m$, V_{aline} can be determined.

4.2.1.3 Three Phase Bridge Rectifier

The alternator's star connected stator windings are connected to three rectifier banks on the locomotive, one for each phase. The output voltage is then connected across the traction motor's power circuit. Where the output voltage can be expressed as:

$$v_{ab} = \sqrt{2}\, V_{ab} \sin\omega t \quad \text{for} \quad \frac{\pi}{3} \leq \omega t \leq \frac{2\pi}{3} \tag{4.23}$$

Where V_{ab} is the line-to-line *rms* input voltage which is equal to V_{aline} or V_{as}. When substituting V_{aline} into equation 4.23 v_{ab} is equal to:



The output current can then be expressed as follows:

$$L\frac{di_o}{dt} + Ri_o + E_b = \left|\sqrt{2}V_{ab}\sin\omega t\right| \text{ for } i_0 \geq 0 \tag{4.24}$$

Where L, R and E_b will be defined in the next section when the load is discussed.

4.2.1.4 DC Traction Motors

As indicated in Figure 4.6 below, the alternator's ac output is rectified by means of a full wave bridge rectifier, which supplies DC power to 6 separately excited dc traction motors.

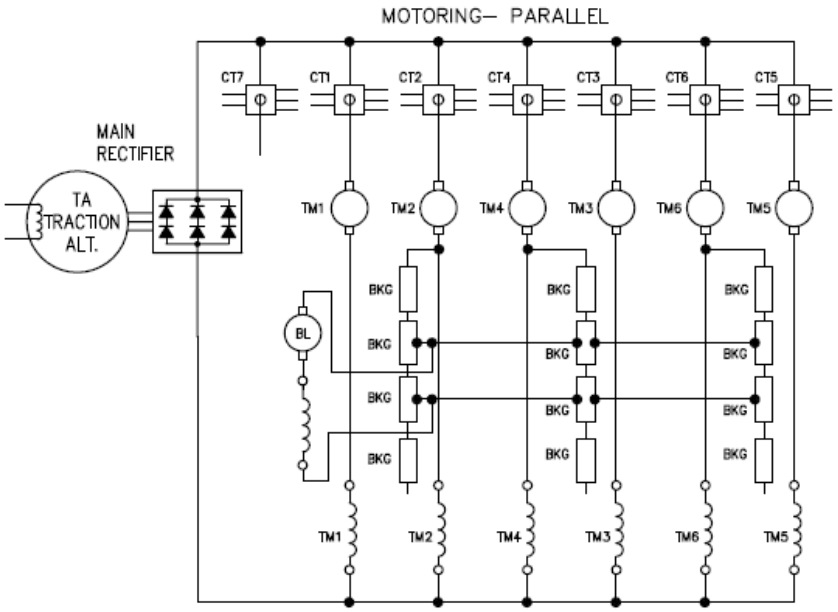


Figure 4.6: Schematic Diagram of the Power Circuit Connections [30]

Figure 4.6 also indicates that 6 separately excited DC traction motors are connected in a parallel combination, with their fields and armature windings connected in series. Modelling of one traction motor can thus be expressed as follows [29], [32]:

$$V_{dc} = I_m\left[L_{af1}\omega_{motor} + R_a + R_f\right] + \frac{dI_m}{dt}\left[L_a + L_f\right] \qquad (4.25)$$

Where $I_m = {}^{I_o}/_{6}$, if all 6 traction motor's field and armature windings are equal and $V_{dc} = v_{ab}$ which is the rectified voltage generated by the alternator.

Equation 4.23 can be rewritten as follows:

$$v_{ab} = I_o\frac{\left[L_{af1}\omega_{motor}+R_a+R_f\right]}{6} + \frac{dI_o}{dt}\frac{\left[L_a+L_f\right]}{6} \qquad (4.26)$$

Equation 4.26 can then be written in terms of the output current I_o:

$$I_o = \frac{6v_{ab}-\frac{dI_o}{dt}\left[L_a+L_f\right]}{L_{af1}\omega_{motor}+R_a+R_f} \qquad (4.27)$$

If L_a and L_f is neglectable, with L_{af1}, R_a and R_f being constants, the output current I_o, is dependent on ω_{motor} and $v_{ab.}$ Where v_{ab} was determined to be dependent on i_f and ω_m. Which are the exciter's field current and the velocity of the primary mover.

When considering the test conditions in which the locomotive will be tested, which would be a stationary condition, it could be noted that $\omega_{motor} = 0$, thus making it a constant, hence I_o is also dependent on i_f and ω_m.

Section Conclusion: i_f and ω_m can be used as inputs to a Neural Network to approximate Rectified Alternator Voltage (SCM8 Sensor) and Exciter Armature Current/Alternator Field Current (EXACT Sensor).

4.2.1.5 Engine Governing Unit

As mentioned in the previous section, the locomotive's electrical power is controlled by a constant horsepower excitation control system. The traction alternator excitation is initiated in response to throttle notch signals controlled by the operator. In order to achieve the electrical power output, a balance between the electrical power generated by the alternator and the mechanical power of the diesel engine is required. The governor acts as the primary feedback for the control of the electrical power, as its function is to control the mechanical engine power through an increase in fuel. The governor controls the fuel supply to the engine by sensing momentary drops in the engine's RPM due to the power demand from the alternator exceeding the engine's output. The alternator's power demand is increased due to an increase in the traction effort requirement selected by the operator, which causes the control system to increase excitation on the alternator's rotor windings [30].

The Governor reacts to (momentarily) supply more fuel to the engine to correct the RPM. During this event a pilot valve in the governor seeds oil under pressure to a vane servo motor, which drives a circular rheostat. This changes the electrical resistance of the rheostat and if power matching cannot be done, reduction in the governor's voltage feedback signal is made, via the rheostat. This then reduces the excitation on the alternator's rotor winding, thus decreasing electrical power to the traction motors. This condition is brought forward when the mechanical power cannot match the alternator's request [30].

The governor's power matching function can be expressed as follows:

$$\%\, LCP = \frac{Engine_p}{Alternator_p} \times 100 \qquad (4.28)$$

Where $Engine_p$ and $Alternator_p$ represent the mechanical and electrical powers and the LCP is the electrical voltage signal, which is connected to the rheostat situated on the engine governing unit. [31] stated that the engine's output is kept at a constant level for every notch, to enable the directly driven alternator to have the same constant electrical power (*Disregarding any losses*). Thus the engine power can be considered to be a constant when the locomotive is powering and stationary position. The alternator's output power which represents the electrical power can be expressed as follows:

$$P_a = I_{dc} V_{dc} \qquad (4.29)$$

Where $V_{dc} = v_{ab}$ and $I_{dc} = I_o$ from equations 4.25 and 4.27 respectively. If the locomotive is stationary with the mechanical power kept constant as per [31] the *%LCP* can be defined as follows:

$$\%\, LCP = \frac{Engine_p}{v_{ab} I_o} \times 100$$

From the previous sections it was proved that v_{ab} and I_o are both dependent on i_f and ω_m which is the exciter generator's field current and the engine's rotation in rad/sec; hence it could then be noted that the LCP is indirectly proportional to i_f and ω_m. Thus by monitoring i_f and ω_m with $Engine_p$ being constant, the *LCP* could be determined.

4.2.1.6 Engine Speed Selection

Let's revisit Table 4.1, which highlighted the functions which controls the engine's speed via 4 solenoid valves located within the engine's governor. The process of control was discussed in section 4.2.1.

Condition	Throttle Notch Command	Governor Solenoid Valves				Equiv Notch	Engine Speed (RPM)
		AV	BV	CV	DV		
Shutdown	Neutral	0	0	0	1		0
Idling (Ready)	Neutral	1	0	0	0	2	519-549
Engine Speed Control Above Idling	1	1	1	0	0	4	685-702
	2	1	1	0	0	6	865-873
	3	0	1	1	0	7	960-968
	4	0	1	1	0	7	960-968
	5	1	1	1	0	8	1045-1055
	6	1	1	1	0	8	1045-1055
	7	1	1	1	0	8	1045-1055
	8	1	1	1	0	8	1045-1055
MU Eng. Stop	ANY	0	0	0	1		

It could be noted that the engine's RPM is dependent on the notch command made by the operator, as well as the actuated solenoid valves. The engine's RPM remains constant for notches 3 and 4 and at 1045-1055RPM for notches 5 – 8.

The preceding sections proved that the two main controlling factors, affecting the alternator and exciter's outputs are their field currents and rotor/armature RPMs. Thus the output can be controlled by driving the shaft at a constant RPM while varying the field current or vice versa. This being said, it could be noted from the above table that both these principles are utilised by the locomotive's excitation control system; hence the throttle notch position as well as the actuated solenoid sequence could be determined by monitoring the exciter's field current (I_f) and primary mover's *(Engine)* speed in rad/sec (ω_m).

4.2.2 Neural Network Input-Output Selection

As per the above sections it could be noted that the input-output data selection for training the neural network model can be defined as illustrated in the Figure 4.7 below.

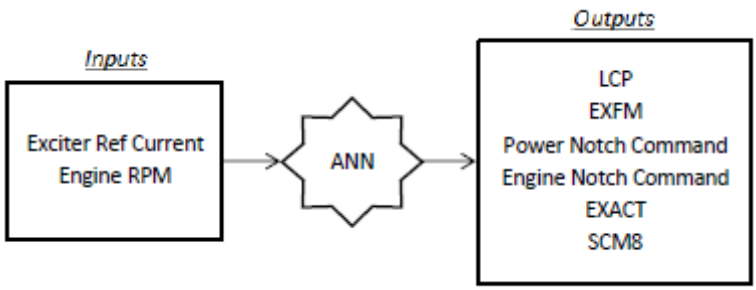


Figure 4.7: Neural Network Input-Output Configuration for a Dedicated Observer Scheme

When analysing the input-output signals to and from the neural network it can be noted that they closely resemble a fault detection and isolation system for a dedicated observer scheme for detection of actuator or component faults. In order to provide an accurate FDI system, which can detect sensor as well as actuator faults, an additional input-output training data set is needed. Mhamdi, Dhouibi, Liouane and Simeu-Abazi [8] describes an observer scheme for the detection of sensor faults as an observer which constitutes a bank of observers which receives all inputs and produces only one output, where the number of observers is equal to the number of sensors. Using this principle for the detection of sensor faults, the inputs and targets need to be different, thus indicating that additional observer models are needed, hence neural networks. In this case the number of additional neural networks is equal to 6, where all other measurements are used to predict a single sensor measurement.

According to [20],[22],[35] and [21], if all the inputs and targets of the neural network have been defined the following steps to data preparation should be done:

- Dealing with Missing Values
- Coding of Input Values
- Outliers
- Noise Injection
- Scaling and Normalization.

With reference to this project, missing values, coding of input values and noise injection will not be discussed due to the nature of the dataset, as well as the number of training parameters present. Outliers and Scaling of the input parameters will however be discussed.

4.2.2.1 Scaling and Normalization:

Scaling of data is not considered a necessity but scaling of the input values to the active domain of the activation functions can improve the performance of a neural network [20, pp. 102-105]. A number of different approaches exist, but for this project the focus will be on:

- Amplitude or Min-Max Scaling
- Mean Centering Scaling
- Variance Scaling
- Combination of Mean and Variance Scaling (Z-Score normalization).

4.2.2.2 Amplitude or Min-Max Scaling:

For bounded functions such as the sigmoid function and the hyperbolic tangent functions min-max scaling is used to scale the target values to the range of the activation functions used, for example (0,1) for the sigmoid function and (-1,1) for the hyperbolic tangent. Where amplitude scaling can be expressed as follows [20, p.102],[21],[22]:

$$t_s = \frac{t_u - t_{u,min}}{t_{u,max} - t_{u,min}}\left(t_{s,max} - t_{s,min}\right) + t_{s,min}$$

where

$t_s = Scaled\ Target$

$t_{s,min} = Minimum\ Scaled\ Target$

$t_{s,max} = Maximum\ Scaled\ Target$

$t_u = Unscaled\ Target$

$t_{u,min} = Minimum\ Unscaled\ Target$

$t_{u,max} = Maximum\ Unscaled\ Target$

This function then linearly maps the target range $[t_{u,min}, t_{u,max}]$ to the range $[t_{s,min}, t_{s,max}]$. It should be noted that scaling target values into smaller ranges have the disadvantage of increased training times. Engelbrecht [20] showed that a neural network must be trained longer to reach a desired accuracy when using the above mentioned method.

4.2.2.3 Z-Score normalization

Z-Score normalization will be discussed as it is essentially a combination of mean centering and variance scaling, which is useful for dealing with outliers in data. For z-score normalization:

$$Z_{i,p}^{MV} = \frac{Z_{i,p} - \bar{Z}_i}{\sigma_{Z_i}}$$

$$T_{k,p}^{MV} = \frac{T_{k,p} - \bar{T}_k}{\sigma_{t_k}}$$

where

$$\bar{Z}_i = \sum_{p=1}^{P} \frac{Z_{i,p}}{P}$$

And

$$\bar{T}_k = \sum_{p=1}^{P} \frac{T_{k,p}}{P}$$

And $\sigma_{t_k}\ and\ \sigma_{z_i}$ are the standard deviations of the target matrix T and the input matrix Z. This method will be used and its performance measured in Chapter 5. It has also been successfully used in the following reference:

- Saravanan, Duyar, Guo and Merrill [26] used normalization through the standardization of deviations from steady state values.

4.3 Learning Rate and Momentum

The learning rate is also directly proportional to the convergence speed of a neural network. The function of the learning rate is to control the size of the step toward the minimum of the objective function. It can become a tricky process due to the following challenges which may arise:

- Learning rate too small causes weight adjustments to also be small; hence more iterations would be necessary to reach the local minimum, thus increasing the training time. This type of learning rate closely approximates the gradient path and could also become trapped in a bad local minimum depending on its starting point. See Figures 4.8a and 4.8d.
- Alternatively, too large learning rates cause large weight adjustments. This could also cause the neural network to "jump" over a good local minimum during the training process. The advantage of a large learning rate is that convergence can be reached fast but then could become oscillatory. See Figures 4.8b and 4.8c

Two common methods used for selecting an optimum learning rate are:

- To start with a small learning rate and increase it if convergence is too slow, or decrease it if the error did not decrease fast enough.
- Plaut et al. proposed a method where the learning rate should be inversely proportional to the *fanin* of the neuron [37, pp.98-101].

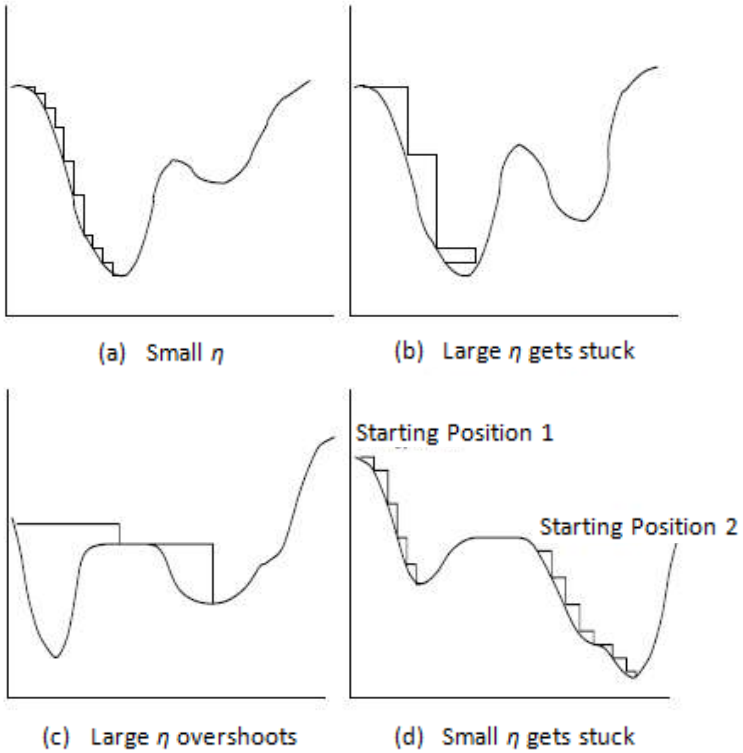


Figure 4.8: Learning Rate Adjustments [37, p.99]

An alternative to the above mentioned methods is the use of the dynamic learning rate [33]. Engelbrecht [20] defines a dynamic learning rate as follows:

"*One of the simplest approaches is to assume that each weight has a different learning rate ηkj. The following rule is then applied to each weight before that weight is updated: if the direction in which the error decreases at this weight change is the same as the direction in which it has been decreasing recently, then ηkj is increased; if not, ηkj is decreased [410]. The direction in which the error decreases is determined by the sign of the partial derivative of the objective function with respect to the weight. Usually, the average change over a number of pattern presentations is considered and not just the previous adjustment* [20]."

Research into the use of a dynamic learning rate with the use of a BP algorithm was reported in [33] and has proven to work well in most cases. Therefore, neural network training will be done with the use of a dynamic learning rate with the GD training algorithm, with the use of Matlab's build-in functions within this project. This method was also selected due to its use in a number of applications, specifically in the field of modelling of control systems and FDI systems [24].

4.3.1 Momentum

When considering stochastic learning, a network spends a huge amount of time going back and forth, unlearning what the previous steps have learned. A solution to this is what is known as the momentum term where the idea is to average the weight changes to ensure that the search path is in an average downhill direction. A static value of 0.9 is usually used for the momentum term [20]. Adaptive momentum rates have also been developed, where each weight has a different momentum term. Engelbrecht [20, p.109] highlighted this method.

For this project Matlab's build-in functions will be used to train a neural network with adaptive learning rate and static momentum.

4.4 The Stopping condition

The stopping condition is one of the most important factors to consider during training due to the occurrence of overfitting. This concept was briefly discussed in section 4.1.1. One of the earliest yet successful stopping condition is given as follows by [34]:

"The first subset is the training data set, which is used for computing the gradient and updating the network weights and biases. The second subset is used as a validation data set and the third subset is used to evaluate the final accuracy of the NN. The error on the validation data set is monitored during the training process. After some number of iterations, the NN begins to overfit the data and, consequently, the error on the validation data set begins to rise. In order to deal with this problem, when the validation error increases during a specified number of iterations, the algorithm stops the training section and applies the weights and biases at the minimum of the validation error in the NN model [34]."

The error is defined by the specific objective function used, where the objective function can be generally defined as the difference between the network's output and the target output, given in the training data. A number of different objective functions exist, where the most commonly used are:

- Mean Squared Error,
- Mean Absolute Error, and
- Sum of Squared Error.

Recently McMillan [35] highlighted the use of genetic algorithms and evolutionary computing techniques to train neural networks, where the objective functions were defined in the training algorithms.

A number of different approaches exist where the main objectives are aimed at modifying the objective function. A summary of the different methods which utilizes evolutionary computing techniques, modifications done to objective functions, etc., was done by [34].

For this project the focus will be on MSE and MAE as stopping conditions. These two objective functions will be experimented with in Chapter 5.

4.5 Network Configuration

In any artificial neural network the selection of an appropriate network model is of utmost importance, where the success of the network depends on the proper configuration of the network model. However the selection of a proper network configuration is more challenging due to the absence of generalized rules for defining a suitable network configuration. Thus it is difficult to define the number of hidden layers, hidden nodes and learning rate.

These can be determined from scratch through experiments, with the classical trial and error method. Engelbrecht [20] stated that if several network architectures fit a training set equally well, then on average the simplest one will give the best generalization performance. Sietsma and Dow tested and confirmed this in their journal. Engelbrecht [37] presented a simple method to determine the optimum model by training a few different network architectures and then choose the one with the lowest generalization error as estimated by the generalised predicted error. Gao [22] provided an additional check to determine whether there are too few hidden units, by monitoring the training error. If the training error is large then more hidden units are needed. From the above mentioned it could be noted that the selection of the number of hidden layers and hidden units is not easily determined and needs to be done experimentally to determine which architecture provides the best neural network predictions.

4.5.1 Hidden Layers

When considering the number of hidden layers, it is important to realize that the more hidden layers are used, the more feedforward calculations are needed and hence the more computational time is needed. As noted in the above section, it is important to note that the simplest NN construction is always the best to use if it provides similar results to those of larger network constructions. Another important aspect to consider is the use of more hidden layers which has the disadvantage of reduced ability to generalize from unseen patterns outside of the training set [36].

In determining the number of hidden layers, it has been shown that one hidden layer neural network is sufficient to uniformly approximate any continuous functions [36]. Therefore, a single hidden layered neural network with tan-sig activation function will be used.

4.5.2 Hidden Layer Nodes

When considering the number of hidden nodes, it is important to note that there is no definitive method for deciding *a priori* number of nodes. The number of nodes can be closely related to the complexity of the non-linearity of the function to be generated by the network. The same as with the number of hidden layers: if too many hidden nodes are used the network is prone to overfitting and if too few are selected the accuracy of the network is negatively impacted. Thus a general procedure for selecting the optimum number of nodes as used by Saravanan, Duyar, Guo and Merrill [26], which is also highlighted by Engelbrecht [20], is to start with a small number of nodes and increase the number of nodes up to the point that there is no significant change in the networks accuracy. This method was successfully implemented by Saravanan, Duyar, Guo and Merrill [26].

Gao [22] further indicated that when one hidden layer is used the number of nodes to be used should be equal to 20. Experiments will be done using the above mentioned methods to find the optimum network structure.

4.6 Conclusion

This chapter provided the theoretical analysis on the method to design the excitation model using the gradient descent method. It showed that experiments into the different parameters are necessary to effectively develop an accurate model. Examples presented in this chapter, as well as in Chapter 3 showed that the use of a neural network trained using the back propagation training algorithm is sufficient to be used as dedicated observers.

In the next chapter the above parameter analyses and training data will be used to train neural networks to model the system in its nominal form. This will be done with the use of the gradient decent training algorithm. Two main neural networks will be designed, namely, a neural network for the use of a dedicated observer scheme to detect actuator faults, as well as a separate bank of neural network observers for sensor faults.

Chapter 5 – Excitation Model Design for the detection of Sensor and Component faults

Chapter 4 highlighted all the parameters to be used and experimented with in order to successfully train a neural network for use as a residual generator. Let us review the proposed system as illustrated in Figure 5.1 below.

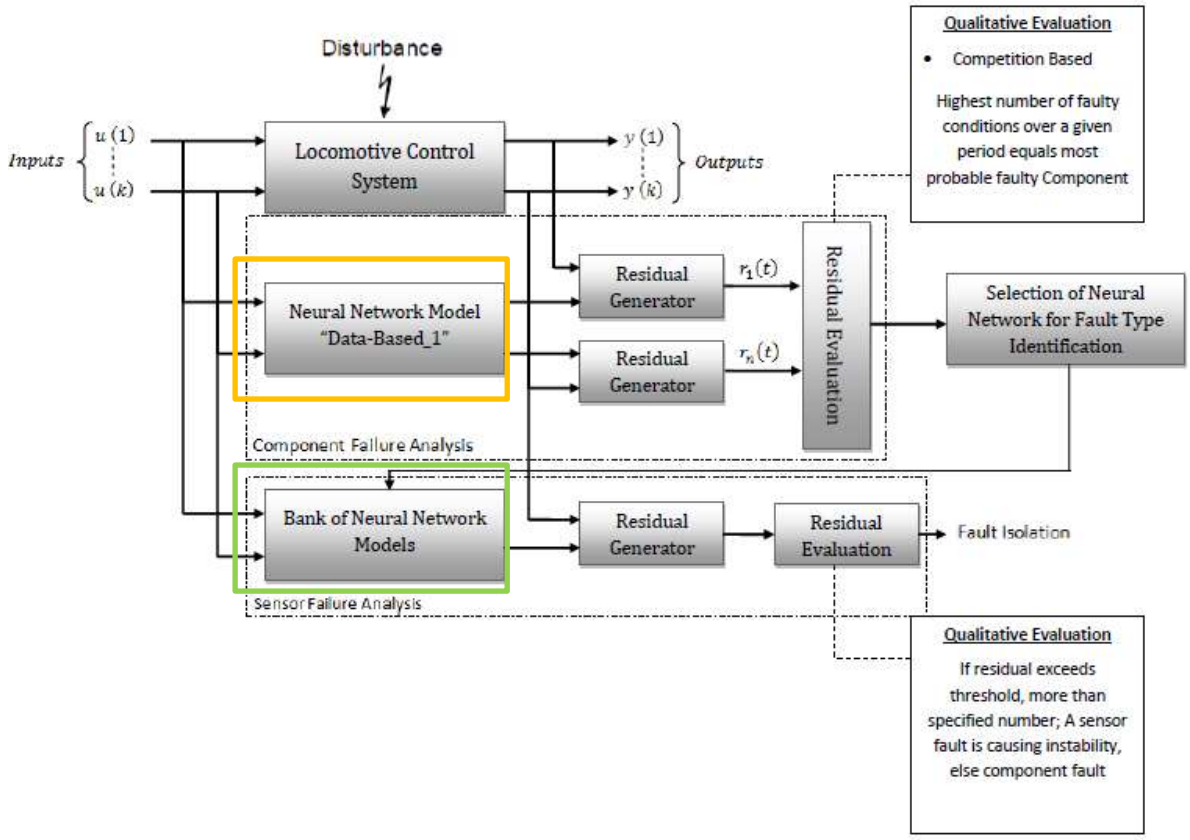


Figure 5.1: Review of Proposed System Design

Chapter 2 indicated that if the dedicated observer scheme is to be used as a residual generator, two separate banks of observers would be needed to isolate component and sensor faults. Figure 5.1 above shows these observers, with the component observer highlighted in orange and the sensor observer in green. Chapter 2 further indicated that these observers require different training sets in order to detect the different type of faults.

In this chapter the gradient descent training algorithm with momentum and dynamic learning rate will be used to train 2 sets of dedicated neural observers to perform as a residual generator for both sensor and component faults. Where the observer used for sensor faults will consist of 6 different neural networks, which constitutes a bank of observers.

The chapter will be divided into three main sections, namely: Neural Network Model Design for detection of component faults, model design for the detection of sensor faults and the

development of a residual evaluation technique with the use of thresholding and a moving average filter.

Experiments will be done in accordance with the neural network performance theory highlighted in Chapter 4, which highlighted the control parameters to be varied or set to static values as per literature reviews in the respective areas. The experiments will be done on all both the observer neural network types.

5.1 Neural Network Model design for Component faults

Using the analysis done in Chapter 4, a neural network will be developed and trained using Matlab's Build-in functions to facilitate an adaptive learning and momentum gradient descent learning algorithm. The training set was modified to satisfy the input-output configuration to isolate actuator faults. Figure 5.2 below recalls the input-output selection made in chapter 4.

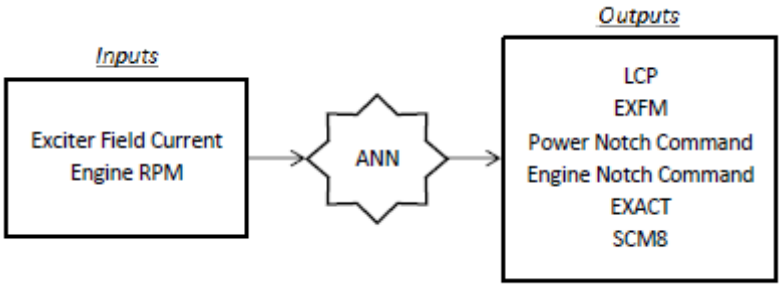


Figure 5.2: Input-Output Configuration for Component Fault Observer Model

In this section, experiments will be done with regard to the following:

- Number of Hidden Neurons
- Scaling
- Stopping Conditions (Objective Functions).

The parameters for the first experiment are illustrated in Table 5.1:

Table 5.1: Experiment 1 Training Parameters

Learning Rate	Learning Rate Increment	Learning Rate Decrement	Momentum	Network Architecture	Scaling
0.0001	1.0005	0.007	0.9	10	Min/Max Scaling

The network architecture for the first experiment was randomly selected at 10 hidden layer neurons and will be further developed later on in the analysis. The learning rate was made small as per the procedure set forth in Chapter 4. Performance was measured using the MSE performance measurement for the first couple of experiments. Where the data set was divided into three main datasets namely; training, validation and the test set. The division was done as follows: 70%, 15% and 15%.

The experiment results are shown in Table 5.2:

Table 5.2: Experiment 1 Results

Simulation	Training Set Error (MSE)	Validation Set Error (MSE)	Test Set Error(MSE)
1	5.330547255	5.586274834	5.277217197
2	5.303593464	4.899633586	5.312831778
3	5.013874924	5.852223293	6.28926204
4	5.278003262	5.525092302	6.311138052
5	5.42896673	4.983707799	5.66261327
6	5.51385784	4.085628528	4.695762704
7	5.633449078	4.812728948	3.935995461
8	5.260224245	5.645437511	5.585492427
9	5.571294812	4.759818946	5.904372127
10	5.322038561	5.617055016	4.918024054
11	5.318679907	4.737281792	5.656448638
12	5.985529554	5.245279095	5.377745021
13	5.404414247	5.640671188	5.041086677
14	5.365335679	4.982594391	5.194110872
15	5.328027436	5.408010372	4.785417975
16	5.499116649	4.812475263	4.674197483
17	5.011983508	5.311209601	6.39876134
18	5.098403564	5.873629913	6.44977643
19	5.395145317	5.586960712	4.510954197
20	4.80428168	6.662653305	6.123008951
21	5.43715457	5.045835634	5.602591809
22	5.351241865	5.787244191	4.811050139
23	5.507104322	5.311389176	5.527994761
24	4.793563179	7.451825961	5.23196447
25	5.175011071	5.857296045	4.786852679
26	5.243963999	6.003176953	4.570595661
27	5.485378762	5.429752638	5.215878109
28	5.20278036	5.088501434	5.487863401
29	5.092331846	5.524778806	5.484659296
30	5.021121503	7.321296004	5.678160949
Average Error	5.30588064	5.494982108	5.350060932

5.1.1 Z-Score Normalization Training Experiment

In this experiment the *Z*-Score normalization as discussed in Chapter 4 will be used in an effort to further improve the performance of the feedforward neural network. This experiment will be referred to as experiment 2. This method is also used to compensate for outliers in the training data [20]. The experimental setup will be as follows:

Table 5.3: Experiment 2 Training Parameters

Learning Rate	Learning Rate Increment	Learning Rate Decrement	Momentum	Network Architecture	Scaling
0.0001	1.0005	0.007	0.9	10	Z-Score

The experiment's results are displayed in Table 5.4 below. It could be noted that the overall average error is lower for all three of the measured MSE's, when compared with the min-max scaling method used in experiment 1.

Table 5.4: Experiment 2 Results

Simulation	Training Set Error (MSE)	Validation Set Error (MSE)	Test Set Error(MSE)
1	4.993645982	6.024774917	5.789848881
2	5.439486592	4.745341743	5.056765908
3	4.985683878	4.942098116	6.817434064
4	5.264175986	6.319690795	4.468169975
5	4.945919038	6.272820421	5.381287384
6	5.346911514	5.449227453	4.898826596
7	4.926470627	6.692524949	5.03560999
8	5.384815167	4.873275943	4.714247075
9	4.948042006	5.7562921	5.556065732
10	5.277533195	3.321207121	6.50831937
11	5.215327751	5.49441129	4.80426023
12	5.310232332	5.346592704	4.70311175
13	5.075252699	5.475634695	5.618012286
14	5.456961821	3.962200483	5.031286781
15	4.768614412	7.050491759	4.97540448
16	5.637553404	5.301822837	3.094273215
17	4.830792394	6.008942807	5.735395984
18	5.575277807	4.90252097	4.24212453
19	5.306151465	5.220063427	5.056982912
20	5.20119293	5.366820712	4.827136978
21	5.401679297	6.030997774	5.47952995
22	4.787632955	5.612888615	6.307488579
23	4.892325095	6.48556515	6.448027551
24	5.096961132	5.851769548	4.825849193
25	5.246037081	3.516159175	6.357946115
26	5.206682691	5.21861362	5.386553008
27	4.9932249	5.397237547	5.431478792
28	5.165834352	5.44927092	5.247784871
29	5.251096146	5.459896872	4.840255247
30	5.301809755	6.137852508	3.840440063
Average Error	5.174444147	5.456233566	5.21599725

5.1.2 Hidden Layer Neurons Experiment

Chapter 4 highlighted two methods to determine the optimum number of hidden layer neurons. The principle of method 1, which indicated that one should start with a small number of hidden neurons and then increase it until there is no remarkable increase in the accuracy of the neural network's output was experimented with and the results are indicated in Appendix A, where Table A5 highlights the results from executing the theory set forth by [22], which stated that when one hidden layer is used the number of nodes to be used should be equal to 20.

Table 5.5 below highlights the results from the tests done in Appendix A, where the number of hidden layer neurons was varied from 8 to 24. From the table it could be noted, that the mean training error decreased with an increase in the number of hidden layer neurons, but the network's generalization ability decreased with an increase in the number of neurons.

This was also the observation of Skidmore in his analysis on the effect of adding hidden layered neurons [20].

The generalization ability for the use of 20 hidden layer neurons, which were selected in accordance with [22], did not decrease and showed a significant improvement over the other neural network architectures with regard to the training and test results. However, the best generalization results were gathered from the network architecture of 10 hidden neurons.

Table 5.5: Average Error Comparison for Varying Hidden Layer Nodes (8-24)

Hidden Layer Nodes	Training Error (MSE)	Validation Error (MSE)	Test Error (MSE)
8	5.235	5.273	5.423
10	5.174	5.456	5.125
12	5.153	5.215	5.446
15	5.113	5.465	5.312
18	5.16	5.381	5.184
20	5.101	5.261	5.129
24	5.094	5.131	5.186

In an effort to validate the two best performing network architectures, training was done again to ensure that the results were accurate and that the correct architecture selection for further development could be made. Both architectures were trained 30 times again and their averages compared. Table 5.6 below shows the comparison.

Table 5.6: Average Error Comparison for Hidden Layer Nodes (10 and 20)

Hidden Layer Nodes	Simulations	Training Error (MSE)	Validation Error (MSE)	Test Error (MSE)
10	1	5.291	5.259	5.188
	2	5.245	5.125	5.308
20	1	5.038	5.49	5.328
	2	5.053	5.319	5.365

With the additional tests as per the above mentioned, the network architecture to be used in the next section will be as follows:

Inputs	**Hidden Layers**	**Hidden Layer Neurons**	**Outputs**	**Hidden Layer Activiation Function**	**Scaling**
2	1	10	6	Tansig	Z-Score

As the test error is used to evaluate the accuracy of the neural network's generalization ability, the decision was based on the fact that the test error is constantly lower for 10 hidden neurons as compared to a network architecture consisting of 20 hidden neurons in the hidden layer. Network complexity also played an important part but overall for an FDI system, an accurate model is of most importance. In the next experiment the stopping condition will be made changed to the MAE objective function.

5.1.3 Objective Function

In this experiment all the same training parameters will be used except for the objective function which will be the mean absolute error (MAE) instead of the mean squared error. The experiment will be referred to as experiment 3, where the two stopping conditions will be compared and the best trained neural network will be used as the system model for the

excitation control system. Table 5.7 below shows the neural network performance when using MAE as the stopping condition.

Table 5.7: Test Results using the MAE as Objective Function

Simulation	Training Set Error (MAE)	Validation Set Error (MAE)	Test Set Error(MAE)
1	0.631718384	0.656437029	0.687043248
2	0.790052889	0.836778804	0.732591615
3	0.796937879	0.795089958	0.831040151
4	0.750459893	0.716378003	0.740243744
5	0.729714621	0.633074676	0.717497811
6	0.718760043	0.696392883	0.740257296
7	0.772377236	0.734869649	0.765026613
8	0.680179549	0.685004955	0.738600055
9	0.75288602	0.744375287	0.746416961
10	0.880332805	0.826662698	0.868383932
11	0.677340575	0.717787281	0.647679243
12	0.701032	0.714333482	0.737112986
13	0.727084495	0.725044209	0.730831506
14	0.726335994	0.673518937	0.768084853
15	0.792848886	0.723338244	0.737708685
16	0.744971391	0.747726251	0.730158674
17	0.703282348	0.739544257	0.74813902
18	0.732864136	0.795855025	0.789711257
19	0.738344621	0.747613435	0.743654611
20	0.693846646	0.691205004	0.667825677
21	0.687640043	0.745492408	0.647089175
22	0.682276248	0.693803713	0.664054985
23	0.783687827	0.665107224	0.808965273
24	0.755681267	0.752951743	0.746786088
25	0.734940809	0.732784558	0.740905123
26	0.695236491	0.695428373	0.710813806
27	0.768846508	0.752622031	0.763646827
28	0.807282327	0.832248248	0.750554706
29	0.708335453	0.755844805	0.770046931
30	0.714820051	0.699488826	0.69934916
Average Error	0.736003915	0.7308934	0.739007334

When comparing the two methods, it is necessary to take the square root of the MSE objective function results. Table 5.8 highlights the square root average error from the best performing MSE results. It could be noted that the performance of the neural network with the use of MAE outperforms the MSE objection function. Thus the MAE will be used to develop the model of the system. In the next section the model will be trained and tested against untrained data.

Table 5.8: Test Results using the MSE as Objective Function with 10 Hidden Neurons

Simulation	Training Set Error (MSE)	Validation Set Error (MSE)	Test Set Error(MSE)
1	7.13513757	6.243367564	6.408967601
2	6.614764759	7.185734735	7.834411458
3	6.999417689	6.580587916	7.2712324
4	7.472850561	6.979292755	7.125984567
5	7.223198156	7.554414032	6.518684944
6	6.765265954	7.979145912	5.914255129
7	6.636864283	7.571433638	7.142409519
8	6.946766064	7.03583033	7.196978469
9	6.890212599	6.801672213	6.629602638
10	6.942303772	7.144529903	6.818721577
11	6.627625919	8.215745371	8.104436548
12	6.99611702	6.295543981	7.617274954
13	6.535668364	6.942778419	7.911566188
14	6.69124927	8.001679915	7.039079792
15	6.773964111	6.8138639	7.000092949
16	7.3948874	7.379583703	7.662407334
17	6.980704164	7.59019337	6.649124473
18	6.776704659	7.130022338	7.020446707
19	7.833692222	7.322152403	8.786507062
20	6.729297943	7.895906332	7.723821842
21	6.983963716	7.002703141	6.915247846
22	6.81947151	6.825227501	7.266004596
23	7.091536904	7.732001246	7.960833821
24	7.290288862	6.79360475	5.485859705
25	7.696144878	7.834515919	7.760804286
26	7.079569261	7.896731564	8.062195235
27	6.999937468	6.995264827	7.943375775
28	6.560836378	9.068201852	6.184948761
29	6.693450732	5.62414933	8.49696478
30	6.805055425	7.340266487	6.500985986
Average Error	2.639361966	2.694291158	2.689195834

5.1.4 Model design

The model was designed using the training parameters used in experiment 3. A neural network was trained 30 times and the best performance in terms of generalization was selected. Table 5.9 below shows the results. The best performing neural network is highlighted, with a testing error 0.627. Figure 5.3 below shows the performance of the training algorithm.

Table 5.9: Final Model Design for Component Fault Observer

Simulation	Training Set Error (MAE)	Validation Set Error (MAE)	Test Set Error(MAE)
1	0.715821678	0.73365324	0.747815455
2	0.791444226	0.777218225	0.741351103
3	0.770414371	0.73109854	0.775976244
4	0.727599435	0.711852136	0.716021753
5	0.701663912	0.762314096	0.711494772
6	0.726035492	0.682323222	0.679359616
7	0.771758682	0.75673137	0.691252163
8	0.746745626	0.710496697	0.82016818
9	0.764544324	0.753512696	0.706508842
10	0.680920671	0.70515299	0.654672033
11	0.774918359	0.722450459	0.753370662
12	0.688032275	0.795636151	0.697648982
13	0.712235368	0.743168124	0.637120864
14	0.775094977	0.794107564	0.7966061
15	0.68480608	0.653265746	0.686951586
16	0.755908685	0.699247796	0.814870698
17	0.795811172	0.804242042	0.864998941
18	0.674077269	0.72270708	0.791242436
19	0.734592863	0.768423871	0.745658849
20	0.664854018	0.689508365	0.627161423
21	0.671277205	0.780356595	0.713587328
22	0.696651569	0.722459298	0.681696253
23	0.781551238	0.706254479	0.735188745
24	0.748904117	0.757716561	0.73739184
25	0.811195359	0.797955965	0.836476216
26	0.856825052	0.847498818	0.863361037
27	0.646779791	0.681524673	0.735486271
28	0.717912183	0.699979078	0.741823558
29	0.750169932	0.747200586	0.740275376
30	0.859572789	0.858722664	0.813710408
Average Error	0.739937291	0.743892638	0.741974924

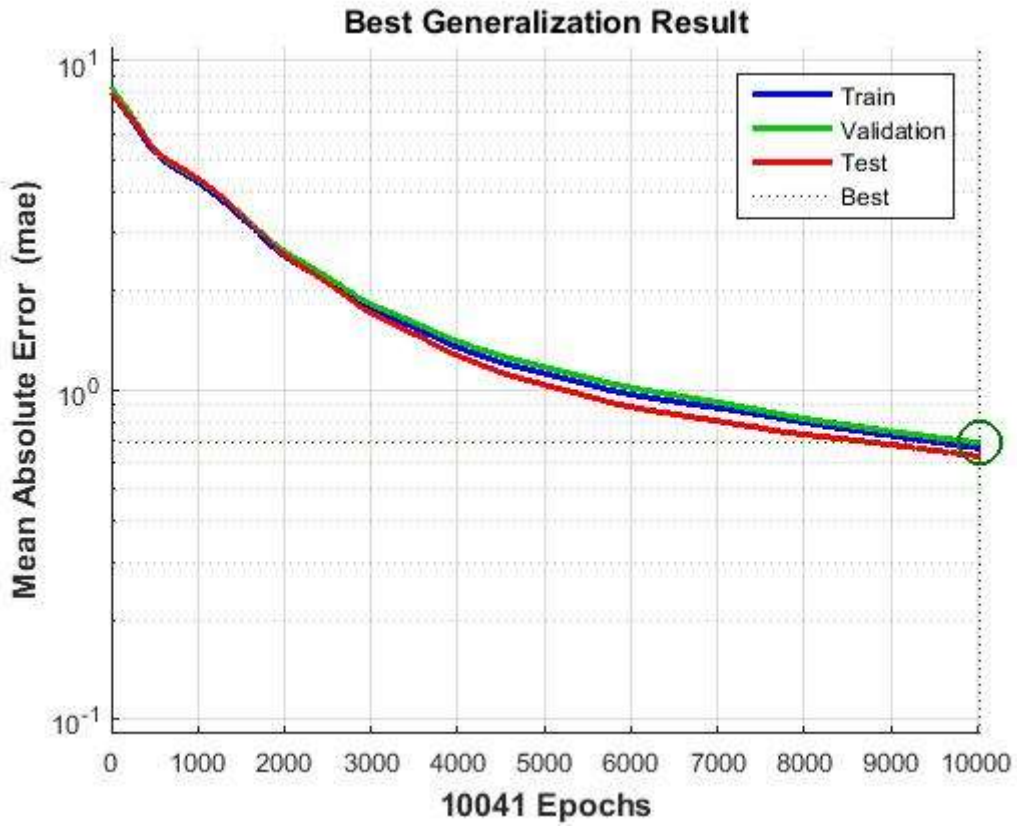


Figure 5.3: Training Test Result for Component Fault Observer Model

5.1.5 Conclusion

Through the analysis of a number of literatures on neural networks used in the field of model design and FDI systems, a model with a testing error of 0.627 was developed. This was done by varying training parameters, of which was recorded in literature. The successful implementation of a model for the excitation system, to be used as a residual generator for the detection of actuator faults was done and the performance thereof will be tested in Chapter 6. As illustrated in Figure 5.1, the primary model is then evaluated using a thresholding technique where the fault is isolated and evaluated further to determine whether it is a sensor or a component failure. In order to determine whether it is a sensor or component failure an additional neural network structure for each component is necessary. These are in the form of a dedicated observer scheme which constitutes a bank of observers to isolate sensor faults. Appendix N section N1 illustrates the code used for training the neural network which was experimented with in this section. The next section will focus on the development of these neural networks.

5.2 Neural Network Model design for Sensor faults

Using the analysis done in Chapter 4 a neural network will be developed and trained using Matlab's Build-in functions to facilitate an adaptive learning and momentum gradient descent learning algorithm. The training set was modified to satisfy the input-output configuration to isolate sensor faults. A total of 6 neural networks will be designed which constitutes a bank of observers used to isolate sensor faults. This section will be divided into 6 sections, where each section will represent an observer. These will be as follows:

- SCM 8
- EXACT
- EXFM
- LCP
- Engine Notch Command
- Power Notch Command.

The same principle of network design will be followed as in section 5.1, and for each section the training set will be displayed, as the training set will differ for each neural observer.

5.2.1 Neural Observer Design for the Rectified Voltage Sensor (SCM 8)

Figure 5.4 below shows the input-output configuration of the training data to be used for training a neural network as a residual generator, to be used as a sensor fault detector for the SCM8 voltage transducer.

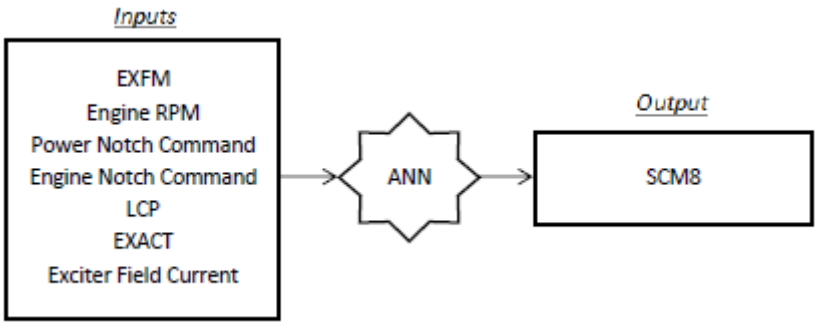


Figure 5.4: Input-Output Configuration for the SCM8 Sensor Neural Observer Model

It could be noted that the training data configuration in terms of the number of inputs differs from that of the actuator faults observer. This is due to the fact that all inputs were not randomly selected but were selected in terms of their effect on the output. Chapter 4 indicated the data preparation which highlighted the system parameters and dependencies of specific variables. The selection of the inputs was done in accordance with this. The following criteria were used to train the neural network.

Table 5.10: SCM8 Experiment Training Parameters

Learning Rate	Learning Rate Increment	Learning Rate Decrement	Momentum	Network Architecture	Scaling
0.0001	1.0005	0.007	0.9	2-30	Z-Score

Z-Score scaling was used; this is due to the results gained from the previous experiments, which showed that when using Z-Score and MAE the neural network gave the best results in terms of generalization. The network architecture was varied from 2 to 30. Where hidden layer with 20 nodes was selected as highlighted in [22]. The results for varying the network architecture with the above mentioned parameters are presented in Appendix B.

Table 5.11 below shows the results from experiments done in Appendix B where the average error of the training, testing and validation set was calculated for each variation in the number of hidden layer neurons. It could be noted that the best generalization abilities were at 24 and 30, but 24 was selected as the networks generalization ability decreased with the use of 30 neurons, thus indicating that the use of larger hidden layered neuron

structure decreased the generalization ability. The optimum network structure was at 24 hidden neurons.

Table 5.11: Average Error Comparison for Varying Hidden Layer Nodes (2-30) for SCM8 Neural Observer Model Design

Hidden Layer Nodes	Training Error (MAE)	Validation Error (MAE)	Test Error (MAE)
2	0.770565221	0.768202103	0.767971904
4	0.708832607	0.699527538	0.710097628
8	0.686970229	0.691215282	0.693393075
10	0.679214962	0.693920957	0.699474683
12	0.683395091	0.682636344	0.708430358
15	0.680100948	0.669396973	0.700614289
20	0.677945426	0.689987127	0.690598694
24	0.665720289	0.676433707	0.674228199
30	0.678319073	0.674845265	0.679630734

Table 5.11 above indicated that an accurate observer model for the detection of sensor faults could be sufficiently developed with the use of following training parameters:

Table 5.12: Optimum Training Parameters for SCM8 Neural Observer Model Design

Learning Rate	Learning Rate Increment	Learning Rate Decrement	Momentum	Network Architecture	Scaling
0.0001	1.0005	0.007	0.9	24	Z-Score

Table 5.13, below highlights the best trained neural network which produced a mean generalization error of 0.569. This was done with varying the training parameters, which were recorded in literature and which are highlighted in Table 5.12. The successful implementation of a model for the excitation system to be used as a residual generator for the detection of the SCM8 sensor faults was done and the performance thereof will be tested in Chapter 6.

Table 5.13: Final Model Design Results for SCM8 Neural Observer Model

Simulation	Training Set Error (MAE)	Validation Set Error (MAE)	Test Set Error(MAE)
1	0.711152778	0.692058592	0.663406303
2	0.665241522	0.642661134	0.67196563
3	0.660539967	0.666781822	0.686965569
4	0.691123963	0.646528152	0.710125137
5	0.669308487	0.777671831	0.709491469
6	0.650050981	0.700732451	0.689105511
7	0.689331004	0.69696175	0.702803379
8	0.702285029	0.649982301	0.717501259
9	0.66976497	0.675747499	0.670614464
10	0.689207329	0.613867054	0.652368049
11	0.691533943	0.639054555	0.67058114
12	0.63717792	0.755479317	0.72510463
13	0.682746259	0.642940413	0.625601941
14	0.68115507	0.708822848	0.682823757
15	0.672221595	0.672811242	0.627735408
16	0.7004104	0.66603741	0.608903358
17	0.699709988	0.621929878	0.674189953
18	0.679517259	0.592767655	0.663711906
19	0.682382944	0.694874242	0.766172432
20	0.674851978	0.672901934	0.786588871
21	0.703838668	0.703201956	0.605732155
22	0.6516365	0.7368768	0.608164349
23	0.664116545	0.661085315	0.674642997
24	0.672162482	0.807210607	0.689615537
25	0.686263807	0.685900781	0.651976198
26	0.664858593	0.661754287	0.607698321
27	0.684216019	0.747336914	0.61839335
28	0.662606401	0.68992917	0.611134732
29	0.647681051	0.686672762	0.654615132
30	0.689567249	0.62882628	0.569193443
Average Error	0.677555357	0.681313565	0.666564213

It was noted that the training parameters as indicated in Table 5.12, provided the best results for sensor fault neural network architectures where a hidden layer of 24 neurons was used. When comparing the training parameters with those of the actuator fault residual generator, it could be noted that the best results were achieved with the use of 10 hidden layered neurons; this is due to the difference in the input-output construction. Appendix N section N2 illustrates the code used for training the neural network.

5.2.2 Neural Observer Design for the Exciter Armature Current Sensor (EXACT)

Figure 5.5 below shows the input-output training configuration for the development of the neural network to be used as a dedicated observer for the detection of sensor faults which could occur due to EXACT sensor faults.

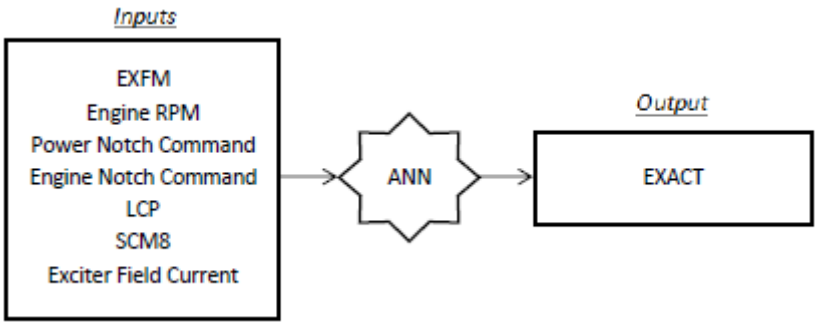


Figure 5.5: Input-Output Configuration for the EXACT Sensor Neural Observer Model

In this section the same experimental procedure was applied as illustrated in table 5.10. The experimental results from varying the neural network architecture are highlighted in Appendix C. Table 5.14, below, provides a breakdown on the results gathered from the experiments done in Appendix C, where the three main performance criteria are highlighted, namely: training, validation and test errors.

Table 5.14: Average Error Comparison for Varying Hidden Layer Nodes (2-30) for EXACT Neural Observer Model Design

Hidden Layer Nodes	Training Error (MAE)	Validation Error (MAE)	Test Error (MAE)
2	1.727022699	1.761281855	1.736610799
4	1.662718812	1.696843326	1.713621373
8	1.622961574	1.616008946	1.65513393
10	1.625604145	1.663400935	1.599714939
12	1.626314708	1.681595051	1.69524953
15	1.571445237	1.593553497	1.654360347
20	1.594914798	1.605752674	1.612801809
24	1.624352472	1.663974391	1.692291666
30	1.674346436	1.713416664	1.688187963

Table 5.14, indicates the optimum result in terms of the generalization ability of the neural network, which resulted from the use of a neural network architecture with 10 hidden layers. It was also noted that the average error in the model did not vary much with the use of different number of neurons, but due to the application for which the model is required, the most accurate model was selected. Table 5.15 below highlights the test results of the optimum EXACT observer model, which will be used as a residual generator within the proposed FDI system. Appendix N section N6 illustrates the code used for training the neural network.

Table 5.15: Final Model Design Results for EXACT Neural Observer Model

Simulation	Training Set Error (MAE)	Validation Set Error (MAE)	Test Set Error(MAE)
1	1.662088158	1.569816944	1.615536438
2	1.478081399	1.484494597	1.39948898
3	1.502027101	1.382701101	1.572712292
4	1.472190351	1.568652744	1.457692064
5	1.616250454	1.583574529	1.533592551
6	1.439879536	1.573255947	1.586474925
7	1.521149959	1.452567252	1.599602689
8	1.648115438	1.590245056	1.546331723
9	1.505785176	1.730534772	1.628846132
10	1.476816093	1.525330231	1.596260816
11	1.586099447	1.513693474	1.503454053
12	1.502363083	1.521950849	1.424640164
13	1.491151795	1.68453253	1.407643081
14	1.627143676	1.690451283	1.515475647
15	1.500361829	1.735631507	1.562113315
16	1.740845818	1.986936894	1.614396668
17	1.583000376	1.626076181	1.5257342
18	1.385691767	1.302460101	1.233806977
19	1.466817308	1.514314607	1.48517933
20	1.689528834	1.754139614	1.563854195
21	1.622583213	1.706871055	1.546680533
22	1.459042172	1.733079241	1.363992312
23	1.559528902	1.623517294	1.513574103
24	1.567664904	1.563526817	1.383680377
25	1.536506671	1.696618073	1.348006441
26	1.600244098	1.576840284	1.556031406
27	1.755907229	1.778396222	1.602833184
28	1.555368981	1.400758637	1.458188884
29	1.554256743	1.557303111	1.423130274
30	1.541909103	1.565353472	1.457630291
Average Error	1.554946654	1.599787481	1.500886135

5.2.3 Neural Observer Design for the Exciter Field Current Module (EXFM)

Figure 5.6 below shows the input-output training configuration for the development of the neural network to be used as a dedicated observer for the detection of a sensor fault which could occur in the EXFM module. The module provides the control system with a feedback of the exciter field current.

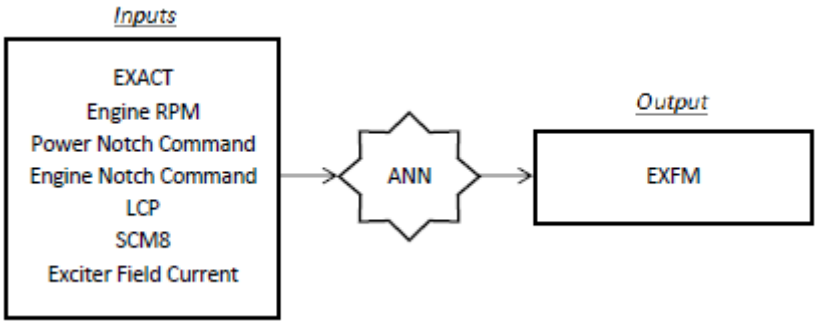


Figure 5.6: Input-Output Configuration for the EXFM Sensor Neural Observer Model

The experimental results from varying the neural network architecture are highlighted in Appendix D. Table 5.16, below, provides a breakdown on the results from the experiments done, where the three main performance criteria are highlighted as training, validation and test errors.

Table 5.16: Average Error Comparison for Varying Hidden Layer Nodes (2-30) for EXFM Neural Observer Model Design

Hidden Layer Nodes	Training Error (MAE)	Validation Error (MAE)	Test Error (MAE)
2	0.241604521	0.241471387	0.240154233
4	0.232418746	0.2335026	0.233177337
8	0.230330026	0.231662941	0.23304112
10	0.232665968	0.233097429	0.234477523
12	0.231228296	0.228213765	0.231829391
15	0.230995053	0.234026295	0.230822365
20	0.230459524	0.232681625	0.231756483
24	0.231264283	0.232744234	0.232708437
30	0.229539413	0.229645749	0.232336711

The table above highlights the network architecture which produced the best results in terms of generalization ability, which is one of the core principles needed for an effective residual generator. Thus for the detection of sensor faults, in the EXFM sensor a network architecture of 15 hidden neurons was selected.

Table 5.18 below highlights the results from training the neural network using the training parameters as mentioned above. The training yielded a testing error of 0.215 which is acceptable for the application of sensor validation within this project. Appendix N section N5 illustrates the code used for training the neural network.

5.2.4 Neural Observer Design for the Load Control Potentiometer (LCP)

Figure 5.7 below shows the input-output training configuration for the development of the neural network to be used as a dedicated observer for the detection of a sensor fault which could occur in the feedback of the load control potentiometer which provides the control system with a measurement of the balance between the electrical and mechanical power. The input-output selection was done as per the data preparation section in Chapter 4.

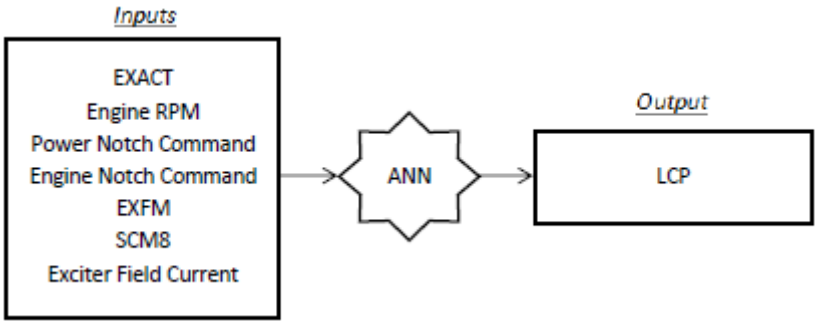


Figure 5.7: Input-Output Configuration for the LCP Neural Observer Model

Appendix E highlights the experimental results from varying the neural network architecture in the training of the model. Table 5.17, below shows the average errors, gathered from these experiments.

Table 5.17: Average Error Comparison for Varying Hidden Layer Nodes (2-30) for LCP Neural Observer Model Design

Hidden Layer Nodes	Training Error (MAE)	Validation Error (MAE)	Test Error (MAE)
2	0.658702283	0.657784819	0.67142268
4	0.651317164	0.6473576	0.649317713
8	0.644255169	0.632234932	0.661954604
10	0.651289991	0.633567137	0.649810316
12	0.653408599	0.626083662	0.668026748
15	0.652138067	0.661662163	0.602125805
20	0.646168418	0.674383678	0.668378295
24	0.639018843	0.661861834	0.636210317
30	0.645015016	0.650299479	0.639773112

The optimum result was obtained with a neural network architecture of 15 hidden neurons. It could be noted that the generalization ability of the neural network decreased with an increase in the neural network architecture greater than 15 neurons; thus an architecture of 15 hidden neurons was selected to develop the residual generator for fault detection of LCP sensor faults.

When considering the measurement range of the LCP sensor, which is 0-74Vdc, the overall error which the model could produce per pattern is approximately 0.813%, which is considerably small.

Table 5.19 below highlights the results from training the neural network 30 times with a network architecture of 15 hidden layer neurons. The best result in terms of generalization is highlighted and will be used in the proposed FDI system. Appendix N section N4 illustrates the code used for training the neural network.

Table 5.18: Final Model Design Results for EXFM Neural Observer Model

Simulation	Training Set Error (MAE)	Validation Set Error (MAE)	Test Set Error(MAE)
1	0.22071177	0.224664026	0.222114352
2	0.236087638	0.2364175	0.230479361
3	0.224506838	0.227356888	0.215558741
4	0.227597859	0.226512531	0.222241641
5	0.224977447	0.225323941	0.226325113
6	0.236818466	0.221077782	0.230285043
7	0.227857851	0.214947443	0.229748302
8	0.239610985	0.23858941	0.231887564
9	0.22777054	0.230632327	0.229181948
10	0.237202941	0.230857997	0.232651987
11	0.222804736	0.236953862	0.21524314
12	0.221084195	0.22118566	0.223910406
13	0.232855038	0.234621463	0.225460653
14	0.227976476	0.235630771	0.226751776
15	0.22860935	0.2398432	0.224035536
16	0.237345369	0.234306571	0.228066889
17	0.22560633	0.222284158	0.226143301
18	0.238995347	0.229527461	0.231971428
19	0.236880538	0.233964994	0.228106809
20	0.232521511	0.227105423	0.231686992
21	0.225788818	0.233477605	0.220121893
22	0.238280127	0.228002619	0.227890169
23	0.235111387	0.238063985	0.230695852
24	0.208555619	0.215719118	0.225387124
25	0.233525243	0.23952723	0.23229434
26	0.232291803	0.227403926	0.229471348
27	0.220732102	0.228084988	0.218824914
28	0.226888551	0.233112687	0.223913609
29	0.222851854	0.226228381	0.232120493
30	0.231782235	0.232817859	0.231936667
Average Error	0.229454299	0.22980806	0.226816913

Table 5.19: Final Model Design Results for LCP Neural Observer Model

Simulation	Training Set Error (MAE)	Validation Set Error (MAE)	Test Set Error(MAE)
1	0.596084527	0.605047293	0.613251651
2	0.630606824	0.777435816	0.586081199
3	0.602181185	0.586690803	0.59429399
4	0.589996878	0.716747011	0.636730676
5	0.646946484	0.639307369	0.559378377
6	0.619601757	0.883906529	0.487499848
7	0.634888918	0.671567896	0.599579846
8	0.603249602	0.652667252	0.635751878
9	0.719452368	0.637111793	0.622347656
10	0.693910919	0.5621524	0.509220423
11	0.618938981	0.496465019	0.63565375
12	0.688216617	0.514796372	0.617259189
13	0.564977659	0.777911841	0.595368453
14	0.656222861	0.547051685	0.533940507
15	0.656023388	0.787004168	0.44213671
16	0.638428511	0.731430067	0.542792421
17	0.629359348	0.589227348	0.542347873
18	0.667328165	0.764440937	0.509079415
19	0.590682348	0.626753667	0.424970109
20	0.594752129	0.550734932	0.567521547
21	0.67218783	0.633174568	0.615904231
22	0.669388061	0.704617924	0.623359292
23	0.664544956	0.775502962	0.502030504
24	0.673544041	0.430277232	0.506435562
25	0.62794679	0.663251425	0.606456524
26	0.671467179	0.564786814	0.605854975
27	0.688453387	0.519228849	0.638926381
28	0.63784188	0.495597035	0.574219598
29	0.651132073	0.535002575	0.567515714
30	0.683824725	0.675337268	0.609073952
Average Error	0.642739346	0.637174228	0.570166075

5.2.5 Neural Observer Design for the Power Notch Command

The power notch command is the primary interface between the user and the BSS control system. The selection of a notch provides the BSS control system with a reference to what the excitation should output to produce a specified traction effort. Figure 5.8 below shows the input-output training configuration for the development of the neural network to be used as a dedicated observer for the detection of sensor faults which could occur in the feedback from the cam controller.

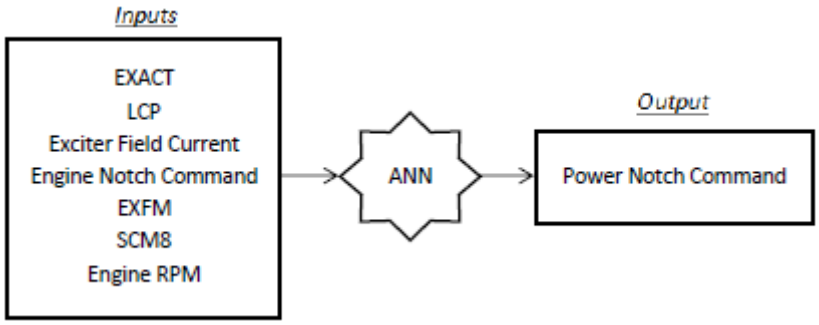


Figure 5.8: Input-Output Configuration for the Power Notch Command Neural Observer Model

The neural observer was trained using the same procedure as indicated in the previous observer designs where the main varying factor was the network architecture in terms of the number of hidden layer neurons. The results from these experiments, as illustrated in Appendix F, in terms of average training, generalization and validation errors are illustrated in Table 5.20 below.

Table 5.20: Average Error Comparison for Varying Hidden Layer Nodes (2-30) for Power Notch Command Neural Observer Model Design

Hidden Layer Nodes	Training Error (MAE)	Validation Error (MAE)	Test Error (MAE)
2	0.055110602	0.05617222	0.054935207
4	0.015757447	0.015664357	0.015649187
8	0.016489381	0.015898326	0.017628388
10	0.018612734	0.019180387	0.017944392
12	0.017569264	0.017391822	0.01824393
15	0.018607196	0.018428942	0.018256354
20	0.018518536	0.018838072	0.019648028
24	0.018923032	0.020192946	0.020090172
30	0.019323907	0.021056801	0.021507685

The best result was obtained from the use of network architecture of 4 hidden layer neurons. Table 5.21 below highlights the results from training the neural network 30 times with a network architecture as illustrated in Table 5.20. The best result in terms of generalization is highlighted and will be used in the proposed FDI system for the detection of sensor faults in the power notch command. Appendix N section N7 illustrates the code used for training the neural network.

Table 5.19: Final Model Design Results for Power Notch Command Neural Observer Model

Simulation	Training Set Error (MAE)	Validation Set Error (MAE)	Test Set Error(MAE)
1	0.013883725	0.012721601	0.015244943
2	0.015000018	0.013492382	0.013454159
3	0.013586017	0.01776692	0.011566301
4	0.015920405	0.010433298	0.012698814
5	0.01553687	0.010080354	0.011148366
6	0.015015079	0.01635825	0.013570081
7	0.015714371	0.012928998	0.013166063
8	0.019329127	0.011069761	0.014238121
9	0.016579305	0.01668431	0.014369854
10	0.013638965	0.022759821	0.013423068
11	0.015164771	0.015419828	0.014865452
12	0.015240179	0.009297374	0.008316444
13	0.014468926	0.020368682	0.012506617
14	0.016283927	0.017193926	0.012303558
15	0.013856161	0.014058314	0.012739515
16	0.015045866	0.016258387	0.012851845
17	0.014788618	0.013357377	0.010189717
18	0.013156784	0.021784261	0.014339475
19	0.015749191	0.012690641	0.011472479
20	0.013361141	0.012373665	0.015254229
21	0.01446528	0.016172094	0.014904323
22	0.016863361	0.015719625	0.015784017
23	0.014952424	0.009034401	0.013795891
24	0.01516484	0.017336303	0.009494061
25	0.013583137	0.016967935	0.012822094
26	0.014674483	0.016803832	0.014297713
27	0.013674311	0.012838726	0.015916868
28	0.016233364	0.017314619	0.014232728
29	0.01421185	0.018458173	0.011531994
30	0.01768771	0.012916788	0.012645922
Average Error	0.01509434	0.015022022	0.013104824

5.3 Residual Evaluation

Chapter 2 highlighted the proposed residual evaluation and provided a literature review on residual evaluation techniques. The proposed residual evaluation was then further reviewed and it was decided that simple thresholding in conjunction with a moving average filter would be used to perform residual evaluation. Figure 5.9 below shows the proposed system with the addition of the threshold calculation to be equated. The threshold calculation was done by analysing the generated residuals from the worst oscillating excitation system of a normal or fault free locomotive.

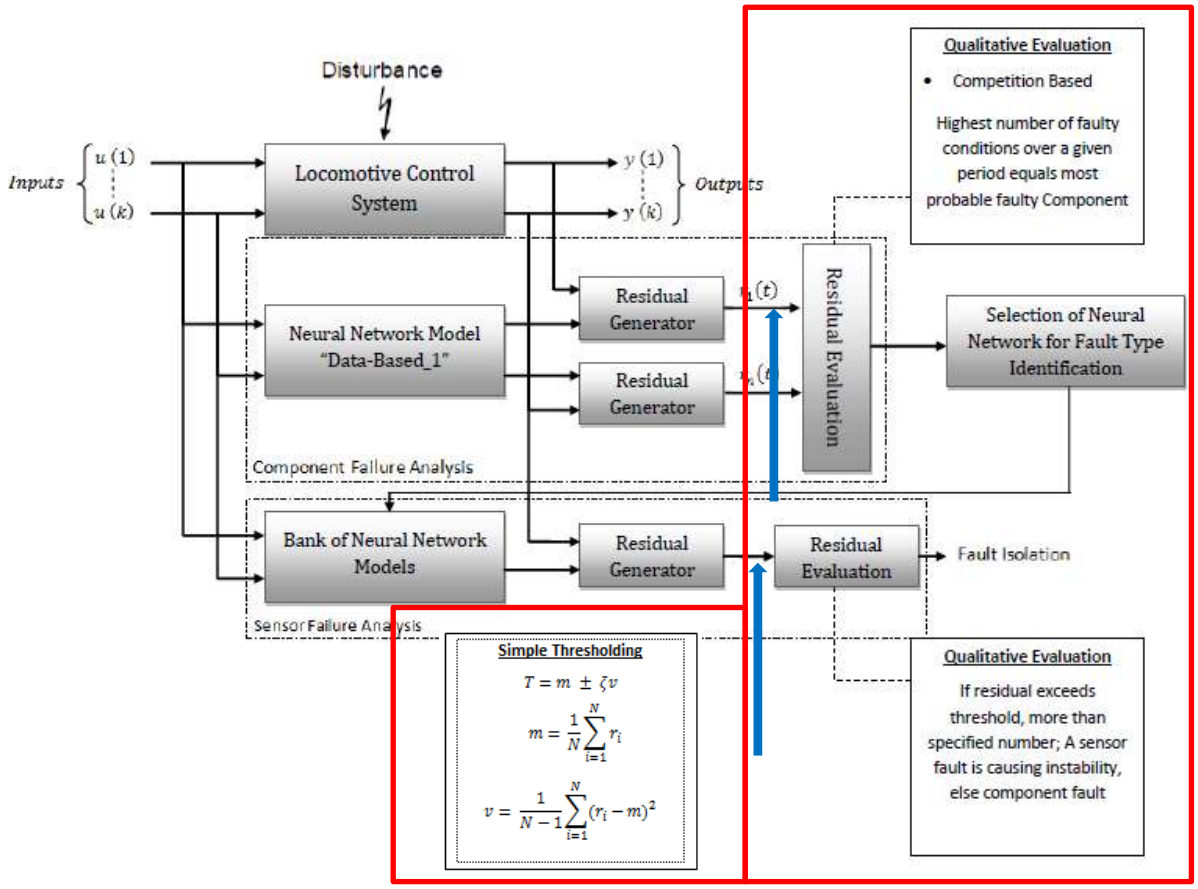


Figure 5.9: Proposed Residual Evaluation System

In order to develop the residual evaluation system the following steps had to be followed:

- Residual Scaling
- Filter Residual to eliminate Noise and Outliers
- Threshold limit calculation by using Simple Thresholding
- Fault Count Method.

This section will be divided into four sections, where section one will focus on the scaling method used; section two, the moving average filter design used to eliminate outliers and noise in the residual signal; section three will then focus on normality tests done on the residual data to determine whether the simple thresholding technique could be used to calculate and select the optimum threshold value. Section four then gives a breakdown of the fault count method to be used to detect and isolate faults. All threshold calculations and filtering source code is highlighted in Appendix O.

5.3.1 Residual Scaling

Owing to the difference in residual sizes of the measured signals, the hyperbolic tangent function was used to scale the residual data in the range of 0-1, by doing the following:

$$R_{j,o} = (y_{measured} - y_{Neural\ Network})^2 \qquad (5.1)$$

Where R is the residual and $y_{measured}$ and $y_{Neural\ Network}$ are the measured and predicted outputs of the monitored system. The square root is then used to ensure only positive

residual values are obtained. Residual scaling is then done using the hyperbolic tangent function as indicated below.

$$R_n = \frac{2}{1+ e^{-\sum_{j=k}^{J=1} R_{j,o}}} - 1 \qquad (5.2)$$

The scaled data is then filtered with the use of a moving average filter to filter out any noise and outliers in the data set. This will be discussed in the next section.

5.3.2 Moving Average Filter

The moving average filter is classified as the most common filter used in discrete sampled processes due to its simple design. Chapter 2 highlighted the use of this filter as a method of residual evaluation and in this section the filter will be applied to the proposed residual evaluation system. Let's review the moving average filter design which [16] gives as follows:

$$y[i] = \frac{1}{\mathrm{M}} \sum_{j=0}^{M-1} x[i+j] \qquad (5.3)$$

Where M is the number of points used in the moving average calculation, x is the input data and y is the filtered output. The degree of smoothing is determined by the number of points specified by M. A sample number (M) of 5 samples were used for smoothing the residual which consisted of 145 samples per locomotive recording, but with the use of the filter specified above, the total number of usable samples drops to 140, due to the filter being a forward moving average filter. Figure 5.10 (a) and (b) below shows an example of the effect of using the moving average filter on the scaled residuals generated for the EXACT.

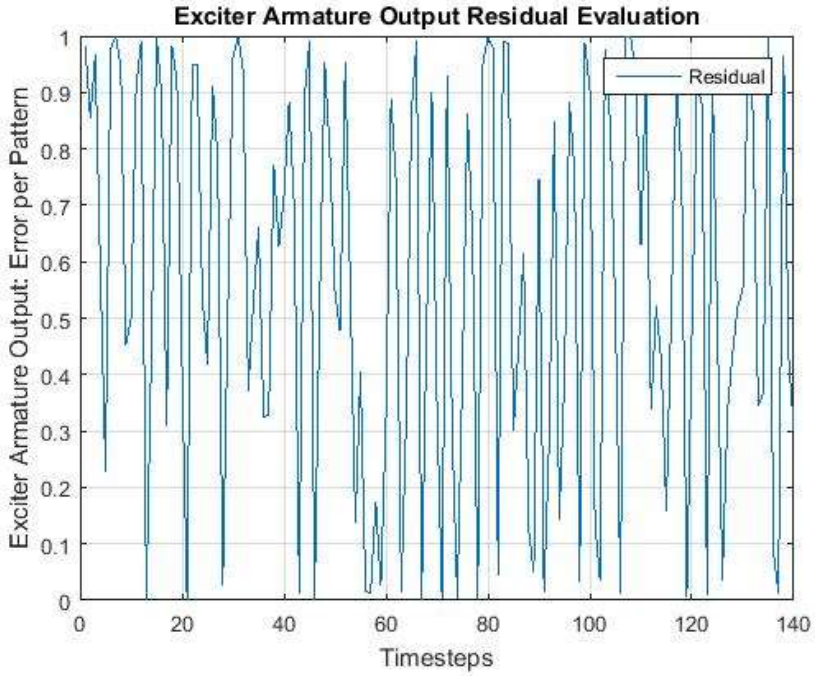


Figure 5.10 (a): EXACT Residual without a Filtering

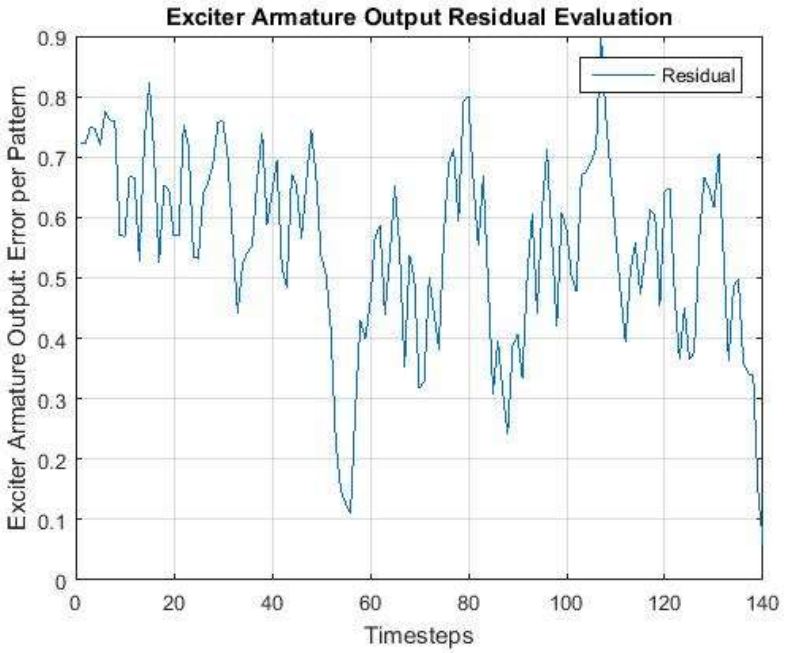


Figure 5.10 (b): EXACT Residual with a Moving Average Filter

It could be noted from the figure above that the moving average filter, removes most of the noise in the residual, which makes thresholding calculation easier and minimizes the possibility of false alarms due to noise. M was selected to be 5, through the experimentation done in Appendix G, where it could be noted that with an increase in M, there is a decrease in the amount of noise but also a decrease in the discreet time steps.

5.3.3 Simple Thresholding

Simple thresholding is a technique used to calculate a threshold value. The technique can only be used with a high confidence level if the normality assumption is satisfied. Thus it is important to verify whether the filtered residuals are normally distributed and satisfy the normality assumption. Patan [6] stated that if simple thresholding is to be used on data which is not normally distributed, the system will tend to give more false alarms. Thus in this section normality tests will be done on the residuals observed from the component and sensor fault detection sections. The thresholds will then be calculated and tabulated for each observed measurement.

5.3.3.1 Component Fault Detection

The component fault detection input-output configuration was highlighted in section 5.1, where 6 outputs were observed and compared to 6 neural network outputs, excited by two inputs. The difference between the neural network outputs and the observed outputs produces residuals and in this section analysis will be done to verify whether the data is normally distributed to verify whether the simple thresholding technique can be applied with high confidence. Two main checks, namely: a probability plot and the Chi-Square Goodness of fit test, were used to check if the residuals were normally distributed.

Appendix H highlights the probability tests done for the residual data generated for isolating component faults. Table 5.20 below highlights the results obtained from analysis done on the probability plots and Chi-Square Goodness of fit test.

Table 5.20: Normality Test Results for Component Neural Observer Model

Residual Data Test	**Visual Probability Plot**	**Chi-square Goodness of Fit**
EXACT	Skewed Distribution (Semi Long Tails)	Accepts the Null Hypothesis at a 1% significance Level
EXFM	Normal Distribution	Accepts the Null Hypothesis at a 5% significance Level
SCM8	Normal Distribution	Accepts the Null Hypothesis at a 5% significance Level
Engine Notch Command	Left Skewed Distribution	Accepts the Null Hypothesis at a 5% significance Level
Power Notch Command	Short Tails Distribution	Rejects the Null Hypothesis
Exciter Reference Field	Normal Distribution (One Outlier)	Accepts the Null Hypothesis at a 5% significance Level
LCP	Left Skewed Distribution	Accepts the Null Hypothesis at a 1% significance Level

From the table it could be noted that the Power Notch Command input is not normally distributed; hence the simple thresholding technique cannot be used to accurately calculate a threshold value. Owing to the function of the Power Notch Command, on the locomotive the threshold value could be easily selected manually without the use of statistical techniques.

With the normality tests done, the threshold values for the component or sectional fault isolation could be done, with the use of the simple thresholding technique. Appendix I shows the threshold limits for each measurement. Table 5.21 below gives a summary of the threshold values calculated for each observed measurement.

Table 5.21: Threshold Values for Component Residuals

Residual Observed	Engine Notch CMD	Power Notch CMD	SCM8	EXACT	EXFM	LCP
Lamda	1	N/A	1.5	1.5	1.5	1
Threshold Value	0.00378276	0.45	0.702682577	0.789902293	0.203869739	0.201130211

These values were calculated from data obtained from the worst oscillating locomotive under normal conditions, meaning that the oscillations were still within limits.

5.3.3.2 Sensor Fault Detection

The sensor fault detection system is divided into 6 main groups, where each group constitutes a sensor reading validation. The sensor to be validated is selected from the highest failing component or sectional failure determined by the component failure analysis section. Sensor validation is done with the use of a dedicated observer scheme for the detection of sensor faults which has the effect that the configuration of the fault detection system is different in terms of its input-output configuration, when compared to component or sectional fault detection. As the configurations and errors of the neural network are different from that of the component analysis section, the residuals would also be different; hence normality tests on each sensor observers residual output is necessary.

The same two methods for determining the distribution of the residuals in section 5.3.3.1 were used in this section. Appendix J illustrates the probability plots for each observed residual. Table 5.22 below highlights the results obtained from the tests done in Appendix J.

Table 5.22: Normality Test Results for Sensor Neural Observer Models

Residual Data Test	Visual Probability Plot	Chi-square Goodness of Fit
EXACT	Normal Distributed	Accepts the Null Hypothesis at a 5% significance Level
EXFM	Long Tails	Accepts the Null Hypothesis at a 5% significance Level
SCM8	Long Tails	Accepts the Null Hypothesis at a 1% significance Level
Engine Notch Command	Normal Distributed	Accepts the Null Hypothesis at a 5% significance Level
Power Notch Command	Normal Distributed	Accepts the Null Hypothesis at a 5% significance Level
LCP	Long Tails	Rejects the Null Hypothesis

It could be noted from the above table that all the residuals are normally distributed except for the LCP's residual. Thus for the cases where the normality assumption can be made, simple thresholding could be used to calculate a threshold value. For the LCP threshold value, system knowledge had to be incorporated and the effect of the LCP on the system, as discussed in Chapter 4, had to be taken into consideration. Owing to the fact that the LCP reading had to be constant during notch 1 at 72Vdc, the neural network model's average testing error was added to a user defined maximum allowable residual of 3% to calculate the threshold. Table 5.23 below highlights the threshold values for each observed residual. The threshold was calculated from the worst yet in limit oscillating control system. Appendix K shows the graphs of the threshold against the residual.

Table 5.23: Threshold Values for Sensor Residuals

Residual Observed	Engine Notch CMD	Power Notch CMD	SCM8	EXACT	EXFM	LCP
Lamda	3.2	2	1.1	1.2	1.5	N/A
Threshold Value	4.13E-08	9.97E-07	0.614165657	0.940231032	0.139353725	0.887494133

The threshold values in the table will be used to detect whether a residual value is out of limits. Owing to the fact that the FDI system is an offline approach and to prevent false alarms or false detections due to intermittent faults, a count process was implemented for fault isolation. This process will be discussed in the next section.

5.3.4 Fault Count Process

In its simplest form, the fault count process can be described as the number of times a residual exceeds a threshold value over a given time period. The reasoning behind using a count process is to eliminate false alarms caused by intermittent or abrupt signals by monitoring the system to see whether the fault persists.

The hypothesis seems to be simple and effective but does not work well with oscillatory faults within a closed loop control system, where the system's aim is to minimize the error between the output variable and the reference signal, causing all the sensor readings in the system to oscillate in excess of their threshold. It is for this reason that a simple count or penalty system could not be used for the isolation of faulty sensor or component faults causing oscillations.

The proposed fault counting process is indicated below, where it could be noted that the count is done over M samples, where the count increment is based on the percentage of how far above the threshold value the residuals are. Thus if the residuals are far above the threshold it will give a 1 count otherwise it will increment with the percentage above the threshold.

$$Count(a) = \sum_{m=1}^{M=140} \frac{R_n(m,a) - T(a)}{1 - T(a)} \qquad (5.4)$$

Where

R_n is the filtered residual

M is the number of samples

$T(a)$ is the threshold for the observed signal

$Count(a)$ is the fault count for the observed signal.

This method is based on the assumption that within a closed loop control system the greater the error the more system response is needed to correct the error, thus the residual which deviates the most from the norm, has a higher probability of causing the oscillation compared to that of a reading with a smaller deviation, which would cause a smaller error, hence less system response.

5.4 Conclusion

This chapter highlighted the successful development of residual generators, as well as residual evaluation techniques to be used within the proposed FDI system. The residual evaluation technique which incorporated a probability analysis was done and will be tested in the next chapter.

Up to this point all relevant theories, as well as the successful development of the proposed system were developed with great accuracy. Chapter 6 will use these principles and designed FDI components to develop a user friendly software package to perform FDI on GE diesel-electric locomotive's excitation control system. The performance of the FDI system will be assessed and discussed.

Chapter 6 – FDI Application Setup and Performance Evaluation

In this chapter all relevant theories, designs and developments dealt with in the previous chapters will be combined to develop a prototype user-friendly Matlab GUI application, to perform fault detection and isolation on sensor and component faults which cause oscillations in the excitation system of a 34 class GE Diesel-Electric locomotive.

The chapter is divided into three main sections namely: FDI application operation, FDI performance evaluation and total component failures analysis. These three sections gives a breakdown on the operation of the application software designed to perform FDI, analysis on the performance of the FDI's ability to isolate faults and an additional approach being followed for isolating total component failures.

6.1 FDI Application Operation

This section will discuss the developed FDI system, where Figure 6.1 below illustrates a flowchart of the developed FDI system's software configuration.

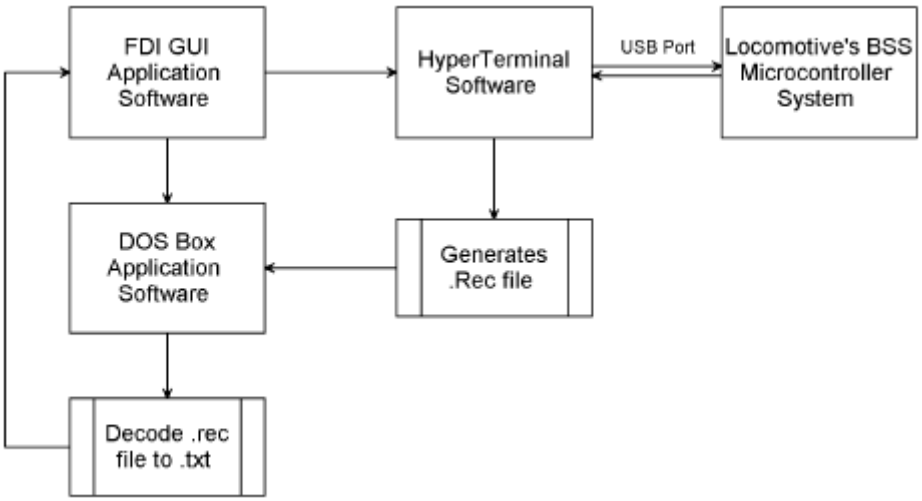


Figure 6.1: FDI GUI Application Flowchart

Here it could be noted that the FDI system incorporates the use of 3 software packages, namely: Matlab GUI, Hyperterminal and DOS Box, to perform fault detection and isolation on the locomotive's excitation system. The reasons for utilizing 3 software packages should be noted from their functions in the FDI Application. These functions are as follows:

- Matlab GUI Application software
  - Primary User Machine interface via a GUI
  - Performs all computations for residual generation and evaluations
  - Performs inter-program communications
  - Step-by-step instruction Guide for the FDI Process
  - Provides the user with the results of the FDI

- Hyperterminal Software
  - Communicates with the Locomotive's microcontroller system
  - Creates a .rec file from the recorded measurements
- DOS Box Software
  - Used to run the 16 bit decoding program on a 64bit Windows Operating System
  - Decoding Software decodes .rec file and outputs a .txt file.

From the functions highlighted above it could be noted that the Hyperterminal software is required to communicate with the locomotive's control system, whereas Dos Box is used to run the Decoding software which is a 16bit program and was found to be incapable of running on a 64bit Windows operating system. The Matlab GUI on the other hand is at the core, fulfilling the role of a master with the DOS Box and the Hyperterminal packages being the slaves. Figure 6.2 below illustrates the Matlab GUI application. Source Code for the GUI application is given in Appendix P.

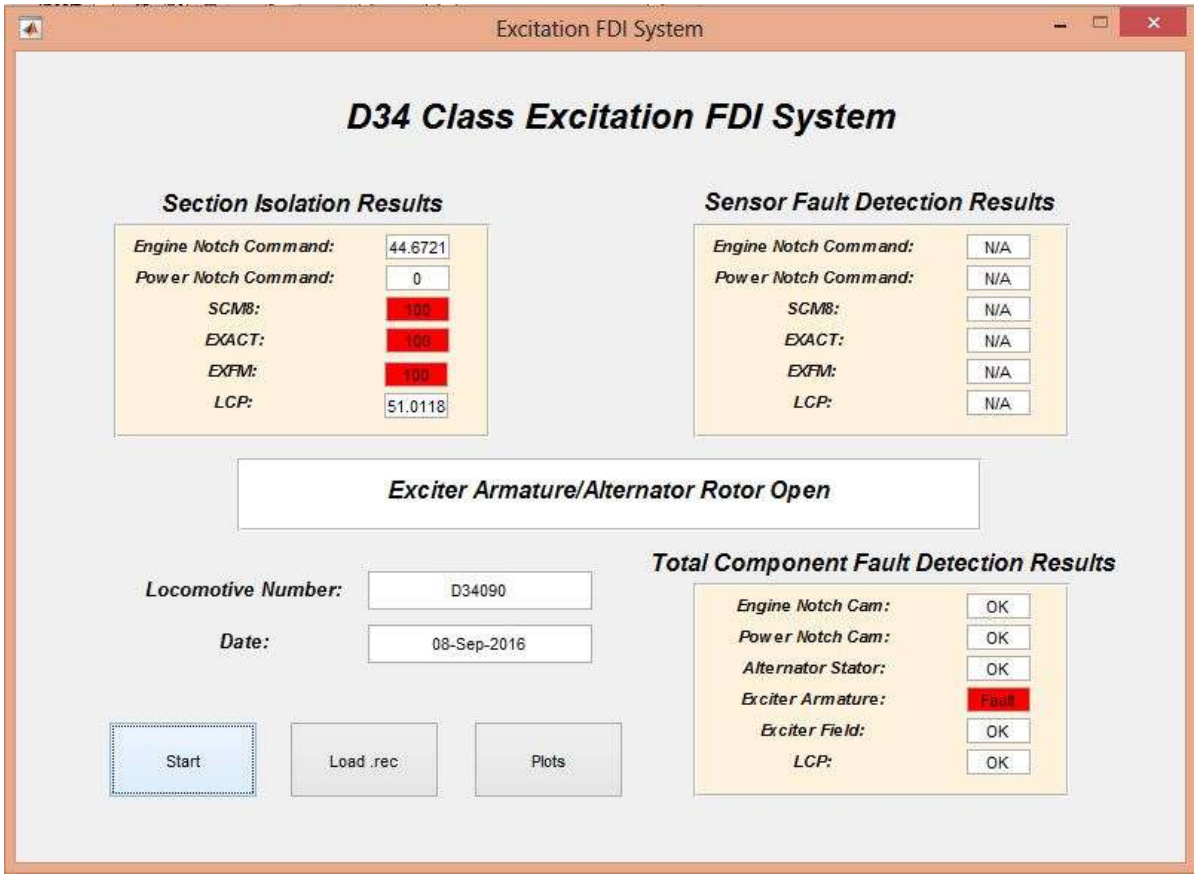


Figure 6.2: Developed FDI Matlab GUI Program

It could be noted that the FDI system requires user interface, whilst guiding the user via a status-bar located in the middle of the window and provides results for component as well as sensor faults. The general flow of the program as indicated by Figure 6.1 and illustrated in Figure 6.2, can be described in the following steps:

1. Matlab GUI requires a user input at the 'Locomotive Number' textbox.
2. The instruction box in the middle of the window instructs the user to notch the locomotive to notch one and then press the start button.

3. Matlab GUI sends control commands to Hyperterminal software to enable recording.
4. Hyperterminal then creates a .rec file.
5. Matlab GUI then sends control commands to close Hyperterminal and open DOS Box.
6. Matlab then sends commands to Dos Box to decode the .rec file to a .txt file.
7. When .txt file is created, Matlab closes Dos Box and imports the .txt file.
8. The imported .txt file is then saved in a matrix and sorted to allow input variables to be inputted into the component failure analysis neural network.
9. Matlab then performs residual generation by comparing the neural network output to the measured outputs (.txt).
10. Residuals are then evaluated with the use of thresholding via Matlab and the highest failing component is then sent to second stage for sensor validation.
11. For sensor validation the .txt file is again sorted to incorporate the input-output configuration for the sensor neural network.
12. Matlab then again performs residual generation by comparing the neural network output to the measured outputs (.txt).
13. Residuals are then evaluated with the use of thresholding via Matlab and if the first and second evaluation is high then it is a sensor fault but if the sensor validation shows low then it is a component causing the oscillation. See Figure 6.3 below.
14. The GUI then highlights the component and/or sensor readings which has the highest calculated probability of causing an oscillation fault in the control system.
15. The GUI also displays the fault in the textbox located in the centre of window.

Appendix M provides a flowchart representation of the internal operations of the above mentioned process. Appendix P highlights the code for the Matlab GUI application.

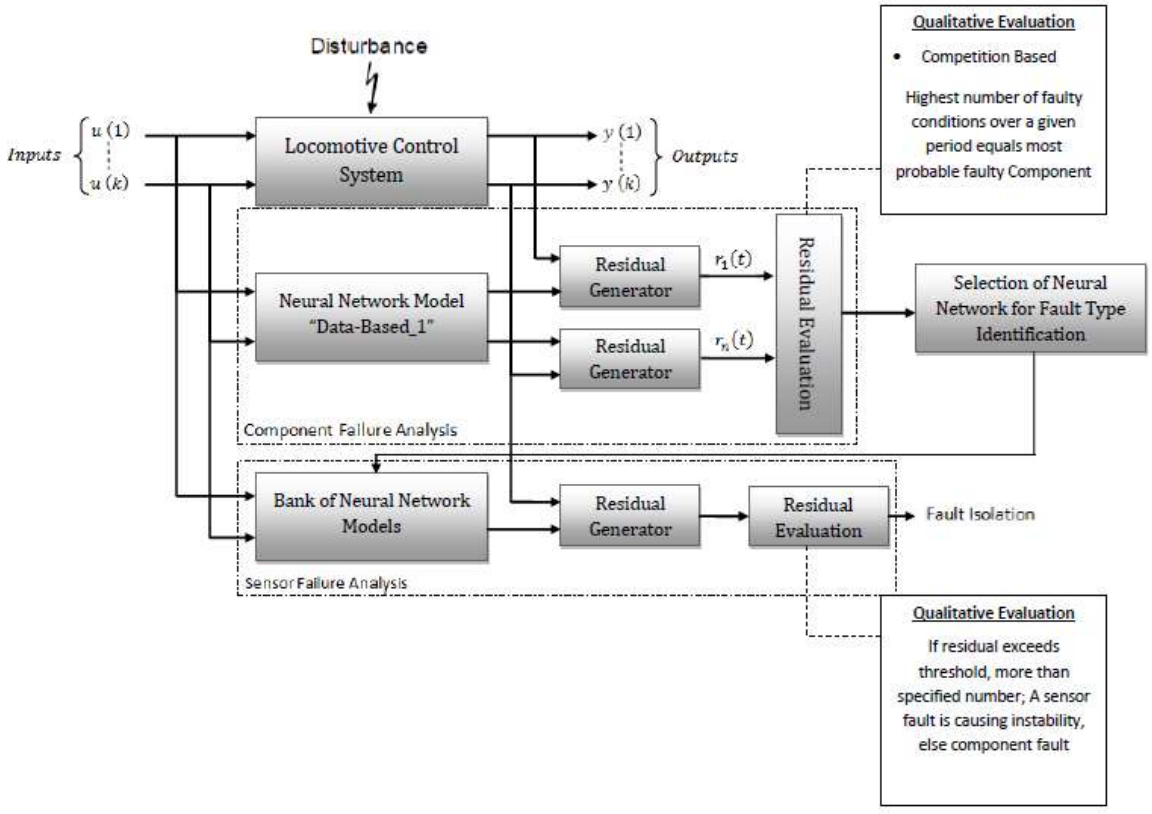


Figure 6.3: Developed FDI System Principle

6.2 FDI Performance Evaluation

In this section the performance of the FDI system will be evaluated in terms of its ability to detect and isolate faults in an oscillatory system. To effectively test the performance of the developed FDI system, tests were done on real faults occurring on locomotives and not simulated faults. Table 6.1 below shows the results of fault isolation with the use of the developed FDI system.

Table 6.1: FDI System Test Results

Fault Type	Test	Probability Engine Notch Command	Probability Power Notch Command	Probability SCM8	Probability EXACT	Probability EXFM	Probability LCP	Sensor Fault Probability	Component Failure(Complete)	Results
SCM8	1	3.32%	0.00%	96.74%	66.46%	8.58%	0.00%	90.43%	N/A	SCM8 Sensor Fault
	2	0.56%	0.00%	100.00%	26.20%	1.37%	0.00%	97.00%	N/A	SCM8 Sensor Fault
	3	0.00%	0.00%	100.00%	17.20%	0.00%	0.00%	96.98%	N/A	SCM8 Sensor Fault
	4	0.01%	0.00%	100.00%	29.83%	0.00%	0.00%	97.31%	N/A	SCM8 Sensor Fault
	5	0.03%	0.00%	100.00%	31.57%	0.00%	0.00%	97.54%	N/A	SCM8 Sensor Fault
	6	0.08%	0.00%	100.00%	23.15%	0.00%	0.00%	90.84%	N/A	SCM8 Sensor Fault
	7	0.01%	0.00%	99.99%	9.47%	0.00%	0.00%	87.09%	N/A	SCM8 Sensor Fault
	8	3.70%	0.00%	97.25%	68.22%	10.49%	0.00%	95.88%	N/A	SCM8 Sensor Fault
EXACT	1	19.98%	0.00%	8.70%	100.00%	10.60%	0.00%	96.93%	N/A	EXACT Sensor Fault
	2	25.80%	0.00%	15.53%	100.00%	22.47%	0.02%	96.53%	N/A	EXACT Sensor Fault
	3	32.17%	0.00%	23.35%	100.00%	26.76%	0.00%	97.14%	N/A	EXACT Sensor Fault
	4	28.18%	0.00%	15.07%	100.00%	21.36%	0.00%	97.14%	N/A	EXACT Sensor Fault
	5	44.67%	0.00%	100.00%	100.00%	100.00%	51.01%	N/A	3	Total Failure Component Fault
	6	44.68%	0.00%	100.00%	100.00%	100.00%	0.01%	N/A	3	Total Failure Component Fault
	7	44.68%	0.00%	100.00%	100.00%	100.00%	0.00%	N/A	3	Total Failure Component Fault
LCP	1	0.00%	0.00%	0.00%	0.00%	0.00%	100.00%	98.09%	N/A	Sensor Fault
	2	11.13%	0.00%	9.33%	16.40%	13.05%	99.99%	0.00%	N/A	Component Fault
	3	0.00%	0.00%	0.00%	0.00%	0.00%	100.00%	98.19%	N/A	LCP Sensor Fault
EXFM	1	0.00%	0.00%	0.00%	0.00%	85.37%	0.00%	96.43%	N/A	EXFM Sensor Fault
	2	57.98%	0.00%	78.42%	83.17%	100.00%	0.00%	100.00%	N/A	EXFM Sensor Fault
	3	0.00%	0.00%	0.00%	19.94%	92.13%	30.94%	100.00%	N/A	EXFM Sensor Fault
Normal	1	0.00%	0.00%	0.00%	0.32%	0.00%	0.00%	N/A	N/A	No Fault
	2	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	N/A	N/A	No Fault
	3	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	N/A	N/A	No Fault
	4	0.00%	0.00%	6.85%	0.12%	0.00%	0.30%	N/A	N/A	No Fault

Table 6.1 indicates that the developed FDI system accurately isolates component or sensor faults which cause oscillations in the locomotive's excitation control system. Where the average accuracy of isolating a fault is as follows:

- SCM8 – 99.25% for the sectional isolation stage and 94.13% on the sensor validation
- EXACT – 100% for the sectional isolation stage and 96.6% on the sensor validation stage
- LCP – 99.99% for the sectional isolation stage and 98.14% on the sensor validation stage
- EXFM - 92.37% for the sectional isolation stage and 98.81% on the sensor validation stage.

Appendix L, shows the graphs of the different residuals obtained from the tests done in Table 6.1.

6.2.1 SCM8 Results Analysis

When analyzing the SCM8 tests, it could be observed that the probability of failure for the EXACT is high in certain cases even though it is not as high as the SCM8's probability. This was noted when there was an increase in the magnitude of the oscillation, which then caused an increase in the probability of failure for the EXACT. Even at the worst oscillation the FDI system still isolated the faulty section and/or sensor with a high confidence level, thus indicating that the FDI system was accurate.

6.2.2 EXACT Results Analysis

Analysis on the EXACT results indicated that the FDI system detected and isolated faulty components or sensors with a high confidence level except for total component failure faults, where it was unable to detect or isolate the cause of failure. Section 6.3 discusses the reasons for this and also provides the solution to the problem.

For oscillatory faults within the system the proposed FDI system detected faults with a high confidence level and it was noted that oscillations in the system did have a major effect on the probabilities of the other monitored signals. Thus in conclusion it could be noted that the developed FDI system isolated EXACT sensor faults in an oscillatory system with high confidence.

6.2.3 LCP Results Analysis

For the LCP test the results component and sensor faults were isolated with high confidence levels. Two of the three faults which caused oscillations were sensor failures and one was a component failure. The developed FDI's performance on all of the different faults was satisfactory and it was noted that the other monitored readings were not majorly affected by oscillations in the LCP's sensor readings' oscillations but were affected by governor oscillations, which is a component oscillation.

6.2.4 EXFM Results Analysis

Analysis done on the results from the EXFM indicated that the FDI system isolated faults with a high confidence level for the detection of sensory faults. It was also noted that for Test 2, as indicated in Table 6.1, the oscillation in the EXFM signal caused heavy oscillations in the other sensor readings as well. This specific fault was found to be a negative wire fault where the EXFM sensor's negative wire was burned off. This then caused the signal to impact on all the shared negative signals in the system. This was found to be the worst case scenario, where if the oscillations were greater than this it would be unsafe to power notch the locomotive in the conventional manner and loadbox connection would need to be implemented.

Total component failures on this section of the excitation system will be discussed in section 6.3.

6.2.5 Normal Results Analysis

A number of tests were done on non-faulty locomotives to test the FDI system's performance on normal locomotives. As indicated in Table 6.1, it could be noted that the results indicated the FDI system displayed with high confidence levels that no faults occurred in the system. This ability to evaluate and store the performance of non-faulty locomotives could theoretically be used for preventative maintenance measures. This concept will be discussed in more detail in Chapter 7.

6.3 Total Component Failure Analysis

Table 6.1, test 5-7 for the EXACT indicated that there was a 100% probability of failure on three of the sections monitored, which had the effect that the FDI system was unable to isolate the fault. Analysis on the fault indicated that the problem occurred with the complete failure of a component in the system, where a complete failure can be described as a component for which there is an input but no output.

With a complete component failure the locomotive's control system adjusts the controlled variable upwards in an attempt to read an output on the measured output variable. This causes the input variables for the sectional fault isolation on the developed FDI system to be far beyond that of the training data used to train the neural network. The magnitude of the value is of such a size that it cannot be predicted by the neural network, due to the fact that the neural network can only successfully predict or generalize within the range it was trained and in this application the training data was gathered from a locomotive powering in a stationary position from notch 1 to 2. The training data was chosen as the function of the FDI system was to detect and isolate oscillatory faults.

It should also be noted that the control system of the locomotive does provide fault detection on total component failures and that the purpose of the project was not to isolate total component failures but rather oscillatory faults caused by faulty components or sensors; however, it was decided that due to the ".rec" file import capability of the developed system, which would enable FDI on the ".rec" file of the locomotive and not on the locomotive itself, to incorporate a fault detection and isolation for total component failures as well. This would enable the FDI to also detect these faults, which are indicated by the control system but are not included in the ".rec" file. To realize this, the following options were considered:

- Include the data from a total component failure in the training set
- Train neural network from locomotive power notch 1 to 8
- Use system analysis as done in section 4.2.1.1 to determine cause and effect of the interconnected components and develop a neural network to isolate the fault
- Use a knowledge base system which incorporates a fuzzy logic system.

In an effort not to alter the complete design of the FDI system's neural network design, the second option was selected to perform FDI for total component failures. Section 6.3.1 below provides a breakdown on the operation of the proposed system.

6.3.1 Total Component Failure FDI Application Design

As not to alter the design of the original specified FDI system for the detection of oscillatory faults, the analysis done in section 4.2.1.1 was used to construct a logical flow of the interconnected system to set up training data to train a neural network to isolate faults during total component failures. Figure 6.4 below illustrates the logical flow of the excitation system.

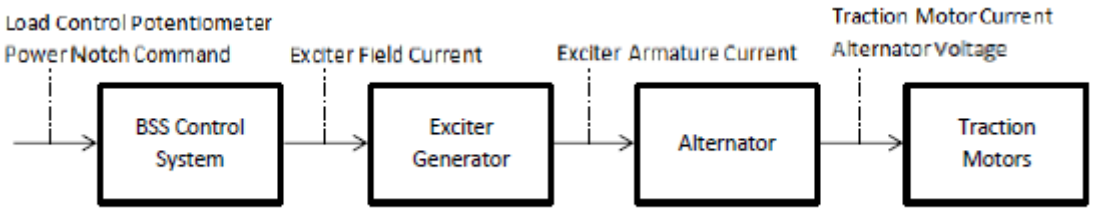


Figure 6.4: Power Flow of the Locomotive's Excitation System

Here it can be seen that the LCP and Power Notch Command have an impact on the exciter field current which is controlled by the locomotive's control system, which then has an effect on the exciter's output, thus affecting the alternator's rotor current and alternator's output going to the traction motors. Table 6.2 shows the theoretical fault classification based on the logical flow of the excitation system:

Table 6.2: Theoretical Fault Classification Based on a Two-Valued System

Inputs						***Faults***
ENC	*PNC*	*SCM8*	*EXACT*	*EXFM*	*LCP*	*Fault Description*
0	0	0	0	0	0	Power Notch Command Fault
0	0	100%	100%	100%	100%	Load Control Potentiometer Fault
0	0	100%	100%	100%	0	Exciter Field Open Circuit
0	0	100%	100%	0	0	Exciter Armature/Alternator Rotor No Output
0	0	100%	0	0	0	Alternator Field Open Circuit

The percentages represent the probability that a fault is active in that specified section. The probabilities will be received from the sectional fault isolation section of the FDI system.

The problem with this theory is that the sectional fault isolation, was not trained to notch 8; thus prediction of the compensated input variables is incorrect even if there is a reading which is not zero. To fully explain this, let's consider Figure 6.2 again; if there is no exciter armature current the control system increases the exciter's field current to 100%; this then gives a value of x amps. Now the sectional neural network predicts or estimates the exciter field current to its maximum value which would be close to the current obtained in notch 2, whereas the actual current flow would be notch 8(x amps). The residual generated from this will then also have a high probability close to or equal to 100%. This is due to the fact that the residuals are squared to remove all negative values from the residuals, thus scaling the data between 0 and 1 with the use of the tan-sig function. Owing to this the highlighted probabilities in Figure 6.2, will all be the same as the "Exciter Field Open Circuit" fault.

To counteract this, the residual from the sectional results were scaled into three groups with the use of the tan-sig function. The groups were as follows:

- '0' No Fault
- '1' Measured Signal > Neural Network Estimation
- '-1' Measured Signal < Neural Network Estimation.

From this a logical thought process was used to set up training data for a neural network, which would then be used to output a number which is coupled to a specific fault. Table 6.3 below shows the theoretical fault classification based on the three-valued system as discussed above.

Table 6.3: Theoretical Fault Classification Based on a Three-Valued System

Inputs						***Faults***	
ENC	*PNC*	*SCM8*	*EXACT*	*EXFM*	*LCP*	*Outputs*	*Faults Description*
0	0	-1	-1	-1	-1	1	LCP
0	0	-1	-1	-1	0	2	Exciter Field Fault
0	0	-1	-1	1	0	3	Exciter Fault
0	0	-1	1	1	0	4	SCM8
0	0	0	0	0	0	5	Power Notch Command Fault

Noise injection was performed on the data as specified in Table 6.3, to increase the number of training patterns in an effort to increase the neural networks generalization ability. A neural network was then trained to output a number which corresponds to a specified fault. See Appendix Q for the training results of the neural network. Figure 6.5 below shows the basic flow diagram of how the total component failure fault isolation is done. Appendix M provides a general flow of the integration of the FDI system for oscillatory faults and total component failure detection.

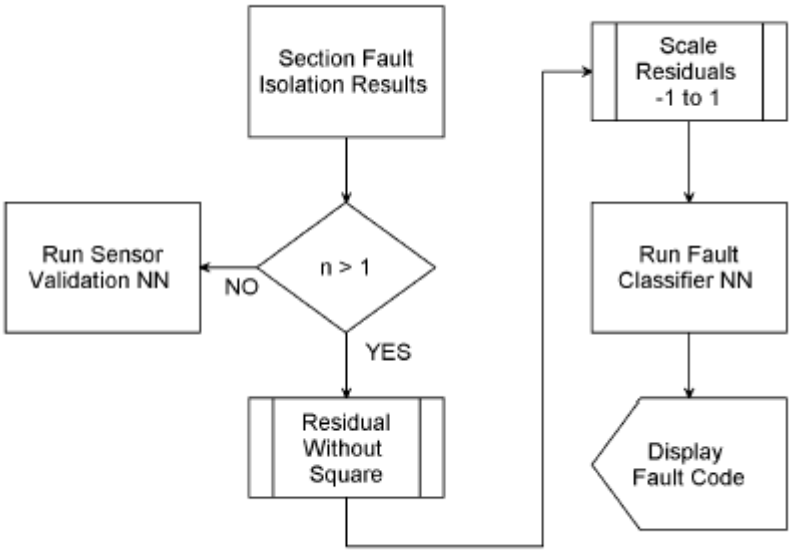


Figure 6.5: Total Component Failure Isolation Flow Diagram

From the figure above it could be noted that the output from the sectional fault isolation was monitored to see if more than one section had a high probability of being faulty. This was indicated by the n>1 and If this condition is true then the residuals from the sectional isolation are redone but not squared, thus providing negative values as well. These values are then scaled between -1 to 1 using a tan-sig function and ran through a neural network, which then outputs fault code or number as illustrated in Table 6.1.

6.4 Conclusion

The chapter showed that the FDI system was implemented with great success and that a small addition to the design made fault isolation of total component failures possible. The addition did not require any major research and was based on knowledge of the system

which was obtained from Chapter 4's analysis of the excitation control system. The isolation of faults causing oscillation in the excitation system performed with high accuracy.

A user friendly interface application was developed and proved to be easy to understand and effective for the purpose of fault isolation.

Chapter 7 – Conclusion

This chapter will discuss the success of the research done and look at possible improvements and future research. The chapter will thus be divided into two main sections; namely: Dissertation Conclusion and Future Research.

7.1 Dissertation Conclusion

The research done proved that a simple feedforward neural network trained with a gradient descent training algorithm can be used to model a complex closed loop control system. The model could be trained to function as a dedicated neural observer to detect and isolate components or sensors causing oscillatory faults. In order to detect sensor faults using a neural observer the neural network's input-output configuration needed to change; hence a bank of different neural networks was needed to detect sensor and component faults, where the number of neural networks was equal to the number of sensors being monitored.

The neural observer's accuracy was dependent on the amount of training data available, where it was beneficial to include all normal operational data, which would enable the observer to accurately estimate outputs from all different conditions of the plant. This was noted in this application where for oscillatory faults the neural observer was only trained to estimate outputs from the locomotive's power notch 1 and 2, due to the fact that excessive oscillation could be detected in the locomotive's power notch 1. When the FDI system's abilities to isolate total component failures were tested, it was found that it could not sufficiently isolate the faults with the design for the detection and isolation of oscillatory faults. Analysis of the input data to the different neural observers indicated that it was due to the control system increasing its reference input to try to compensate for the no output condition of the failing component. It was also noted that the reference input was increased to power notch 8 of the locomotive's power notches, but with the observer trained with data gathered from the locomotive's power notch 1 to 2 it could not sufficiently estimate outputs from the locomotive's power notch 8's reference input; thus it could not isolate the fault sufficiently. The detection of a total component failure was not the primary concern in this project as the locomotive's human machine interfacing module (HMI) could detect total component failures, but due to the fact that .rec files could be loaded into the GUI application the project included an extension which satisfactorily isolated total component faults.

As the literature in the document indicated, the residual evaluation technique was of utmost importance and it was found that the use of a normal fault count method which incremented if a residual exceeded a threshold was insufficient due to the fact that all signal feedbacks oscillate in the system if a sensor or component starts to oscillate. An alternative approach was used, which incorporated the use of a tan-sig and percentage above the threshold function. This function sufficiently isolated the faulty components and sensors in the case of oscillatory faults.

It was also noted that the oscillatory conditions were increased whenever a negative was removed from a sensing unit and even then the FDI system sufficiently isolated the faulty component, but with the decision margins being closer to each other than for normal oscillatory faults. *See Table 6.1 EXFM Test 2.*

In conclusion to the research done, it could be noted that the developed FDI system isolated oscillatory faults with high confidence and produced a 100% accuracy for the detection and isolation of the sensor or component causing the oscillation. The use of a neural observer model indicated that some knowledge of the system or plants architecture with regard to input-output configuration was necessary to develop an optimum FDI system. This knowledge of the system would enable the neural network to be trained on true cause and effect data, making training easier and errors smaller. The overall aim of the project, which was to develop a user friendly software based FDI system to isolate faults which cause oscillations, whilst decreasing the fault finding and isolation time from hours to minutes was successfully met. The project enabled technicians to isolate faults within 2 to 3 minutes and repair within 10 minutes compared to the approximately 3 hours previously. This had a major positive impact on the productivity, and enabled more work to be outputted with less time spend on one fault.

7.2 Future Research

More research is required in the use of the probability data to perform preventative maintenance. This would incorporate the use of data analysis of a normal performing locomotive's excitation control system during its service every 45 days. The analysis will entail the recording of the probability results from testing the locomotive gathered from the developed FDI system, as indicated in Figure 6.2, and using this data to predict a possible remaining life cycle for the sensing components.

Research into the use of different threshold limits to detect small oscillations due to interference or dirty components will also be done in an effort to increase the stability of the excitation system.

The ultimate result would be to design a system which could perform FDI onboard through the use of a network interfacing protocol. Fault isolation in terms of a redundant system which could isolate a faulty sensor reading and still have the locomotive operate normally to its service depot, would be the end result.

List of References

[1] R. Isermann, Fault Diagnosis Applications, Darmstadt: Springer, 2010.

[2] M. J. Bagajewicz and D. J. Chmielewski, Smart Process Plants Software and Hardware Solutions for Accurate Data and Profitable Operations, New York, Chicago, San Francisco, Lisbon, London, Madrid, Mexico City, Milan, New Delhi, San Juan, Seoul, Singapore, Sydney, Toronto: The McGraw-Hill Companies, Inc, 2010.

[3] P. M. Frank, E. Garcia and B. Koppen-Seliger, "Modelling for fault detection and isolation versus modelling for control," Elsevier Science B.V, Duisburg, Germany, 2000.

[4] R. Isermann, "Model-based fault detection and diagnosis - status and applications," Elsevier Ltd, Darmstadt, Germany, 2005.

[5] N. P. Srivastava, R. K. Srivastava and P. K. Vashishtha, "Fault Detection and Isolation (FDI) Via Neural Networks," *Journal of Engineering Research and Applications,* vol. IV, no. 1, pp. 81-86, 2014.

[6] K. Patan, Artificial Neural Networks for the Modeling and Fault Diagnosis of Technical Processes, Poland: Springer, 2008.

[7] S. Singh and T. V. R. Murthy, "Neural Network based Sensor Fault Detection for Flight Control Systems," *International Journal of Computer Applications,* vol. 59, no. 13, pp. 1-8, 2012.

[8] L. Mhamdi, H. Dhouibi, N. Liouane and Z. Simeu-Abazi, "Detection and localization method of Single and Simultaneous faults," *International Journal of Engineering and Science,* vol. 1, no. 7, pp. 25-35, 2012.

[9] Y. Kourd, D. Lefenvre and N. Guersi, "Fault Diagnosis Based on Neural Networks and Decision Trees: Application to DAMADICS," *International Journal of Innovative Computing, Information and Control,* vol. 9, no. 8, pp. 3185-3196, 2013.

[10] Q.-N. Xu, K.-M. Lee, H.-Z. Yang and H.-Y. Yang, "Model-Based Fault Detection and Isolation Scheme for a Rudder Servo System," *IEEE TRANSACTIONS ON INDUSTRIAL ELECTRONICS,* vol. 62, no. 4, pp. 2384-2396, 2015.

[11] M. Hashimoto, H. Kawashima and F. Oba, "A Multi-Model Based Fault Detection and Diagnosis of Internal Sensor for Mobile Robot," in *Conference on Intelligent Robots and Systems*, Las Vegas, Nevada, 2003.

[12] M. Addel-Geliel, S. Zakzouk and M. El Sengaby, "Application of Model based Fault Detection for an Industrial Boiler," in *Mediterranean Conference on Control & Automation (MED)*, Barcelona, Spain, 2012.

[13] R. Yusof, F. S. Ismail, R. Zafira and A. Rahman, “Model-based Fault Detection and Diagnosis Optimization for Process Control Rig,” *IEEE,* pp. 1-6, 2013.

[14] P. F. Odgaard, B. Lin and S. B. Jorgensen, “Observer and Data-Driven-Model-Based Fault Detection in Power Plant Coal Mills,” *IEEE TRANSACTIONS ON ENERGY CONVERSION,* vol. 23, no. 2, pp. 659-668, 2008.

[15] J. Chen, “Robust Residual Generation for Model-Based Fault Diagnosis of Dynamic Systems,” University of York, 1995.

[16] I. Samy and G. Da-Wei, Fault Detection and Flight Data Measurement: Demonstrated on Unmanned Air Vehicles using Neural Networks, Berlin: Springer, 2012.

[17] J. Korbicz, J. M. Koscielny, Z. Kowalczuk and W. Cholewa, Fault Diagnosis: Models, Artificial Intelligence, Applications, Berlin: Springer Science & Business Media, 2012.

[18] B. Koppen-Seliger and P. M. Frank, “Fault Detection and Isolation in Technical Processes with Neural Networks,” in *Conference on Decision & Control*, New Orleans, LA, 1995.

[19] E. Alcorta Garcia, M. Schubert and P. M. Frank, “Fault Isolation In a Winding-Machine Using RCE Networks,” in *European Control Conference*, Karlsruhe, Germany, 1999.

[20] A. P. Engelbrecht, Computational Intelligence: An Introduction Second Edition, Chichester, England: John Wiley and Sons Ltd, 2007.

[21] E. R. Jones, “An Introduction to Neural Networks,” Visual Numerics, Inc., San Ramon, USA, 2004.

[22] J. Gao, Digital Analysis of Remotely Sensed Imagery, New York, Chicago, San Francisco, Lisbon, London, Madrid, Mexico City, Milan, New Delhi, San Juan, Seoul, Singapore, Sydney, Toronto: The McGraw-Hill Companies, Inc, 2009.

[23] J. Guo, Y. Liu, X. Xu and Q. Chen, “Integrated distributed bond graph modeling and neural network for fault diagnosis system of hydro turbine governors,” *Kybernetes,* vol. 39, no. 6, pp. 925-934, 2010.

[24] H. Taplak, İ. Uzmay and Ş. Yıldırım, “An artificial neural network application to fault detection of a rotor bearing system,” *Industrial Lubrication and Tribology,* vol. 58, no. 1, pp. 32 - 44, 2006.

[25] S. Mousavi and K. Khorasani, “Fault detection of reaction wheels in attitude control subsystem of formation flying satellites: A dynamic neural network-based approach,” *International Journal of Intelligent Unmanned Systems,* vol. 2, no. 1, pp. 2 - 26, 2014.

[26] N. Saravanan, A. Duyar, T. Guo and W. Merrill, “Modeling of the Space Shuttle Main Engine Using Feed-forward Neural Networks,” in *American Control Conference*, San Francisco, California, 1993.

[27] M. R. Napolitano, G. Silvestri, D. A. Windon II, J. L. Casanova and M. Innocenti, “Sensor Validation Using Hardware-Based On-Line Learning Neural Networks,” *IEEE TRANSACTIONS ON AEROSPACE AND ELECTRONIC SYSTEMS,* vol. 34, no. 2, pp. 456-468, 1998.

[28] Electrical Energy Technology, “Academia.Edu,” [Online]. Available: http://www.academia.edu/6065222/48550_Electrical_Energy_Technology. [Accessed 16 July 2016].

[29] G. Chen and X. Shang, “Simulation used in education for a separately excited DC motor,” *World Transactions on Engineering and Technology Education,* vol. 12, no. 1, pp. 14-19, 2014.

[30] GE Transportation, Training Manual: BrightStar Sirius Locomotive Control, Nevada: GE Transportation, 2008.

[31] Transnet Freight Rail Chair in Railway Engineering, *Locomotive System Performance and Maintenance,* Pretoria, 2014.

[32] F. Golnaraghi and B. C. Kuo, Automatic Control Systems, Ninth edition, Illinois: John Wiley & Sons, Inc, 2009.

[33] X.-H. Yu, G.-A. Chen and S.-X. Cheng, “Dynamic Learning Rate Optimization of the Backpropagation Algorithm,” *IEEE TRANSACTIONS ON NEURAL NETWORKS,* vol. 6, no. 3, pp. 669-677, 1995.

[34] O. Ludwig and U. Nunes, “Novel Maximum-Margin Training Algorithms for Supervised Neural Networks,” *IEEE TRANSACTIONS ON NEURAL NETWORKS,,* vol. 21, no. 6, pp. 972-984, 2010.

[35] G. K. McMillan, Process/Industrial Instruments and Controls Handbook, Fifth Edition, New York, San Francisco, Washington, D.C., Auckland, Bogotá, Caracas, Lisbon, London, Madrid, Mexico City, Milan, Montreal, New Delhi, San Juan, Singapore, Sydney, Tokyo , Toronto: The McGraw-Hill Companies, Inc., 1999.

[36] K. Funahashi, “On the approximate realization of continuous mappings by neural networks,” *Neural Networks,* vol. 2, no. 3, pp. 183-192, 1989.

[37] A. Engelbrecht, Computational Intelligence, Chichester: John Wiley & Sons, Ltd, 2002.

[38] N. Cosgrove, “Digital Project Manager,” 16 April 2017. [Online]. Available: https://www.digital-project-manager.com/the-project-triangle/. [Accessed 15 June 2016].

Appendix A

Table A1: Experiment Results with the use of 8 Hidden Layer Neurons

Simulation	Training Set Error (MSE)	Validation Set Error (MSE)	Test Set Error(MSE)
1	5.090510908	4.259531145	6.983463273
2	4.947172123	5.700768941	6.712496066
3	5.264834926	4.557873941	5.911645794
4	4.886852427	6.693479352	5.848715852
5	5.603079136	4.758541021	4.635395013
6	5.158823557	4.464950476	5.944989395
7	5.421517248	5.975706796	4.712164564
8	5.036405685	5.638361279	5.520037824
9	5.129762225	6.068639414	4.851277591
10	5.18580781	6.185575724	5.414525735
11	4.946563491	6.179400989	5.572490846
12	5.140545687	6.255690048	4.468293669
13	5.31067794	4.600115608	6.129550385
14	5.293192986	5.417373528	5.032602481
15	5.185217826	4.403599089	6.328160686
16	5.402418087	5.578507239	4.297129512
17	5.018736495	5.591398958	5.867900382
18	5.210822077	5.796082571	4.934649578
19	5.157424741	5.994681392	4.9033612
20	4.808541227	6.095678166	6.168993299
21	5.31029426	4.36904281	6.348731654
22	5.505568481	3.599695479	5.959529589
23	5.461175851	5.334083706	4.400796773
24	5.541317936	4.067263838	5.106201826
25	5.385538104	4.503751653	5.272168568
26	5.556114567	4.996495888	4.181048062
27	5.144932098	5.837192726	5.331431299
28	5.473620113	4.36643784	5.394754514
29	5.261313712	5.258298072	5.779452151
30	5.237695708	5.652616885	4.700272375
Average Error	5.235882581	5.273361152	5.423740999

Table A2: Experiment Results with the use of 10 Hidden Layer Neurons

Simulation	Training Set Error (MSE)	Validation Set Error (MSE)	Test Set Error(MSE)
1	7.13513757	6.243367564	6.408967601
2	6.614764759	7.185734735	7.834411458
3	6.999417689	6.580587916	7.2712324
4	7.472850561	6.979292755	7.125984567
5	7.223198156	7.554414032	6.518684944
6	6.765265954	7.979145912	5.914255129
7	6.636864283	7.571433638	7.142409519
8	6.946766064	7.03583033	7.196978469
9	6.890212599	6.801672213	6.629602638
10	6.942303772	7.144529903	6.818721577
11	6.627625919	8.215745371	8.104436548
12	6.99611702	6.295543981	7.617274954
13	6.535668364	6.942778419	7.911566188
14	6.69124927	8.001679915	7.039079792
15	6.773964111	6.8138639	7.000092949
16	7.3948874	7.379583703	7.662407334
17	6.980704164	7.59019337	6.649124473
18	6.776704659	7.130022338	7.020446707
19	7.833692222	7.322152403	8.786507062
20	6.729297943	7.895906332	7.723821842
21	6.983963716	7.002703141	6.915247846
22	6.81947151	6.825227501	7.266004596
23	7.091536904	7.732001246	7.960833821
24	7.290288862	6.79360475	5.485859705
25	7.696144878	7.834515919	7.760804286
26	7.079569261	7.896731564	8.062195235
27	6.999937468	6.995264827	7.943375775
28	6.560836378	9.068201852	6.184948761
29	6.693450732	5.62414933	8.49696478
30	6.805055425	7.340266487	6.500985986
Average Error	6.966231587	7.259204845	7.231774231

Table A3: Experiment Results with the use of 12 Hidden Layer Neurons

Simulation	Training Set Error (MSE)	Validation Set Error (MSE)	Test Set Error(MSE)
1	5.434360699	4.374705067	4.774517055
2	5.126189072	5.692946829	5.72253091
3	5.260564908	3.662444934	6.893615715
4	5.04137397	4.676348623	6.26550731
5	4.955315694	5.212316943	5.846250068
6	5.397173437	4.976147424	5.098215675
7	5.341109445	4.575686764	5.459267097
8	5.138479355	5.327256748	5.6815396
9	5.125712222	4.928883142	5.677985146
10	5.185828076	5.296001362	5.999390114
11	5.346478346	4.502787078	5.014295169
12	5.120792157	5.022291725	6.240096828
13	5.289483389	5.966994061	4.409034437
14	5.314089539	4.35406318	4.930274693
15	5.017654888	5.254393883	5.683489032
16	5.099268993	5.44024179	5.300749956
17	4.936714527	5.294415925	6.489766312
18	4.950182207	7.428475078	4.604015769
19	4.959219758	6.560709578	4.74591713
20	5.29404219	5.163559626	4.180917607
21	5.090221013	6.079280493	4.71676264
22	4.678211012	6.737321609	6.039732248
23	5.427959064	4.068584213	5.26948119
24	4.642628975	7.234094705	6.315326506
25	5.485697446	3.567561653	4.913198234
26	5.102751524	5.315326287	5.867793643
27	4.803310154	5.211923103	6.302642445
28	5.342698752	4.625352473	5.013853978
29	5.45437141	4.984004433	5.001812929
30	5.24996237	4.922281733	4.927515528
Average Error	5.153728153	5.215213349	5.446183165

Table A4: Experiment Results with the use of 15 Hidden Layer Neurons

Simulation	Training Set Error (MSE)	Validation Set Error (MSE)	Test Set Error(MSE)
1	5.296996329	4.542477813	5.211958783
2	4.85116613	5.191864756	6.676569428
3	5.483907619	4.974437639	4.619719225
4	5.145299214	5.060735661	6.256549351
5	5.389632162	5.908173576	3.917809798
6	4.894362477	5.328530835	5.965893025
7	4.96078345	5.542360894	6.054485343
8	5.188943006	5.168674469	4.312485776
9	5.340240091	5.459292831	4.03578741
10	5.095229317	5.263783914	5.422178068
11	5.277119676	4.367789081	5.386647825
12	5.26331277	6.76351615	4.436919928
13	4.967211258	6.743006474	4.744580444
14	4.835958069	7.382490575	5.166465441
15	4.631539587	7.033344746	6.213142889
16	5.139658661	5.256899968	5.242881975
17	5.028227903	5.211801462	5.86802305
18	5.15688294	4.702573389	6.735438613
19	5.274222981	4.25584304	6.289108935
20	5.279700899	4.758150953	4.713673525
21	5.145719074	5.544759396	4.978902247
22	4.969867389	6.151150171	5.394635376
23	5.077225804	5.945698906	5.216265893
24	4.885035409	5.155373685	6.191632871
25	4.922911737	6.452316087	4.446481541
26	5.35174764	5.088510658	3.997676745
27	5.270203108	4.898251915	5.076055995
28	4.878127676	5.119945883	6.464125734
29	5.369650433	4.302214212	5.837963181
30	5.028694764	6.399287558	4.488105568
Average Error	5.113319252	5.465775223	5.312072133

Table A5: Experiment Results with the use of 18 Hidden Layer Neurons

Simulation	Training Set Error (MSE)	Validation Set Error (MSE)	Test Set Error(MSE)
1	5.243233285	5.245917195	4.530222415
2	5.320491122	4.796482433	5.242652337
3	5.822781383	5.720179014	6.847904459
4	5.199915865	4.368479418	5.375350041
5	5.261952201	5.056675672	4.661371291
6	4.995846977	4.893030244	6.746767748
7	5.327636241	4.494416684	4.923074126
8	5.277255639	5.485752737	4.213386689
9	4.984784994	4.614815025	6.426470362
10	4.537210107	7.2342359	5.969250554
11	5.057215284	4.831229255	5.469373841
12	5.366990484	5.01612358	4.854076292
13	4.75829124	6.872016724	4.925562
14	4.853579635	6.23895069	5.172453889
15	4.888106383	5.302182174	5.858947116
16	5.078095039	4.458054946	6.806764231
17	5.13557604	5.802976007	4.31502222
18	5.210936804	5.88591254	4.121707345
19	5.476215359	4.752228303	3.967136837
20	5.278184973	5.223056096	4.498383018
21	5.096189464	5.912930032	5.2093355
22	5.333683092	6.652522665	5.657675577
23	5.271425564	4.544820428	5.05766683
24	4.951583503	7.105712187	4.67850578
25	5.264219515	5.043499947	4.980287791
26	5.096998757	5.057809735	6.19411686
27	5.019489546	5.967047062	4.222374449
28	5.109103524	4.956334467	5.53220782
29	5.417284862	5.261700752	3.951224199
30	5.168462027	4.651864717	5.11078465

Table A6: Experiment Results with the use of 20 Hidden Layer Neurons

Simulation	Training Set Error (MSE)	Validation Set Error (MSE)	Test Set Error(MSE)
1	5.26576381	5.465415116	4.580874926
2	4.940113099	4.647396004	6.194101866
3	4.901906769	5.730248376	5.264514099
4	5.023455608	4.886398952	6.203375492
5	5.205175556	5.207611361	4.46084015
6	4.914486447	5.727828741	5.406518459
7	5.194301491	4.607342796	5.341905871
8	5.130659679	5.854037938	4.847002904
9	5.179113381	5.060721571	4.843222713
10	5.01037482	5.095026765	5.579895887
11	4.933751785	5.576129059	5.555158302
12	5.142624858	5.235787124	4.961007044
13	5.492231219	5.476237229	3.043934906
14	5.00703621	6.042829122	5.3088609
15	5.08517627	3.575771067	5.906266388
16	5.180363802	4.779014101	5.193068867
17	5.270640174	5.184469884	5.450143008
18	4.979304423	4.513143631	6.017144383
19	5.087415729	4.919405641	6.178110325
20	5.018122058	5.996446011	4.697993546
21	4.916359041	5.215874187	5.67233273
22	5.453863561	3.40513819	5.347494271
23	5.540453904	3.636638572	4.962769129
24	4.99357788	7.496589127	3.21942398
25	4.950274425	5.670412183	4.941032679
26	4.823449473	6.716745636	4.931209845
27	5.220331053	6.515183025	4.078343564
28	5.018083417	5.005235241	5.327376027
29	5.162116852	4.862452083	5.590329642
30	5.015937918	5.728064993	4.791769695
Average Error	5.101882157	5.261119791	5.129867387

Table A7: Experiment Results with the use of 24 Hidden Layer Neurons

Simulation	Training Set Error (MSE)	Validation Set Error (MSE)	Test Set Error(MSE)
1	5.322693008	4.072671127	5.680327525
2	5.03217556	5.595679089	4.628336895
3	5.32802557	4.240885286	5.073326697
4	4.807669975	5.822646248	5.258812651
5	5.017155749	5.375515516	4.957694419
6	5.109327125	4.828215258	5.96605122
7	5.144956238	5.082003898	5.035106952
8	4.949843875	6.271310487	4.564924972
9	5.232186508	4.81061029	4.200882696
10	4.860358717	6.004906689	5.83659572
11	4.978049976	4.973652226	5.957321114
12	4.819228528	6.603569439	5.473395162
13	5.336747508	5.333487771	3.908504653
14	4.853994352	5.669398631	5.588456442
15	4.935913196	5.148591102	5.279129231
16	5.073870271	5.83667815	5.022211941
17	4.962491195	4.990092367	5.465878493
18	5.176851537	4.372916876	4.62703784
19	5.062275347	4.702467572	6.20483242
20	5.28643744	4.052428122	5.704617471
21	5.027622728	5.168562934	5.373816972
22	5.122010712	5.482613485	5.568475467
23	5.140091078	5.181024254	5.194275206
24	5.091005896	5.964582323	4.385918365
25	5.326353677	3.566099609	5.193500228
26	5.049837319	5.8265518	4.513363253
27	5.078366209	5.143379384	5.37587651
28	5.354336099	4.329512152	4.995947786
29	4.976054716	4.930627681	5.669737353
30	5.368933826	4.570239467	4.888265472
Average Error	5.094162131	5.131697308	5.186420704

Appendix B

Table B1: Experiment Results with the use of 2 Hidden Layer Neurons

Simulation	Training Set Error (MSE)	Validation Set Error (MSE)	Test Set Error(MSE)
1	0.749650807	0.761656543	0.736602789
2	0.782621888	0.706938196	0.835427548
3	0.875685748	0.860163291	0.765452008
4	0.805008186	0.783877131	0.832654054
5	0.864373094	0.865044936	0.814355344
6	0.759015018	0.764429602	0.811791116
7	0.708397614	0.700593429	0.743617045
8	0.762386574	0.74516203	0.869651543
9	0.729145513	0.798774444	0.695702761
10	0.70930569	0.710111906	0.694540237
11	0.789656407	0.731669664	0.781088023
12	0.724928461	0.730277805	0.631744029
13	0.753652732	0.695864688	0.762702941
14	0.702497626	0.688610729	0.718916657
15	0.731455063	0.786146886	0.768595783
16	0.715612272	0.769372965	0.671763128
17	0.687292714	0.68082393	0.787431634
18	0.870141516	0.830156393	0.895001159
19	0.853419203	0.868684231	0.876475341
20	0.875189963	0.803810985	0.781893928
21	0.786619016	0.700839601	0.742028379
22	0.716460003	0.671950961	0.727557072
23	0.755883027	0.831891381	0.700376227
24	0.701127441	0.727230341	0.740570584
25	0.772747573	0.755987553	0.795641911
26	0.738052699	0.713620368	0.695513097
27	0.816895816	0.90768529	0.848944157
28	0.710124828	0.764269032	0.676001079
29	0.730776011	0.756392566	0.722030328
30	0.938834141	0.934026212	0.915087206
Average Error	0.770565221	0.768202103	0.767971904

Table B2: Experiment Results with the use of 4 Hidden Layer Neurons

Simulation	Training Set Error (MSE)	Validation Set Error (MSE)	Test Set Error(MSE)
1	0.707417702	0.708816752	0.650744015
2	0.702706265	0.762808087	0.707398513
3	0.684941981	0.718869952	0.715526768
4	0.705106861	0.673605729	0.768526346
5	0.693000921	0.628818065	0.717780429
6	0.698742865	0.627450706	0.672089231
7	0.708319359	0.639727409	0.726588994
8	0.710841673	0.738641348	0.774030199
9	0.655461507	0.740641237	0.696095582
10	0.728226681	0.616349046	0.692849027
11	0.783365944	0.942021664	0.790694337
12	0.698389866	0.66304652	0.70195991
13	0.7231463	0.738613924	0.65674968
14	0.71657477	0.689498734	0.71205901
15	0.675305902	0.647934438	0.686843968
16	0.705254645	0.687149163	0.670014656
17	0.670094257	0.622847357	0.707737762
18	0.718368972	0.722458665	0.679119665
19	0.697791791	0.706268655	0.690066688
20	0.717589987	0.672969523	0.713178339
21	0.672729294	0.750321676	0.666921408
22	0.743204701	0.753627258	0.790633398
23	0.695641594	0.726482039	0.570052704
24	0.761503003	0.756019557	0.809435684
25	0.716655974	0.717397886	0.720931307
26	0.730167709	0.700763892	0.748158496
27	0.707568135	0.664601508	0.729013034
28	0.697462282	0.693587792	0.685268032
29	0.719779044	0.656596313	0.704458744
30	0.71961824	0.617891257	0.748002909
Average Error	0.708832608	0.699527538	0.710097628

Table B3: Experiment Results with the use of 8 Hidden Layer Neurons

Simulation	Training Set Error (MSE)	Validation Set Error (MSE)	Test Set Error(MSE)
1	0.70708368	0.760108844	0.705503813
2	0.711495043	0.710287008	0.663759354
3	0.692000749	0.726532809	0.687600987
4	0.704520901	0.672150917	0.629646988
5	0.700927743	0.784320546	0.7584697
6	0.699892844	0.652697521	0.621975015
7	0.651281062	0.722033099	0.66184284
8	0.674724014	0.819684207	0.671912057
9	0.710633431	0.716699862	0.752497191
10	0.686499789	0.700183152	0.653242783
11	0.690934711	0.606658373	0.771347878
12	0.696177644	0.638929535	0.704146573
13	0.676972881	0.689884895	0.66346862
14	0.704798193	0.715156999	0.702024984
15	0.714101429	0.713215894	0.712807209
16	0.64695766	0.744721496	0.781854283
17	0.673563949	0.670691377	0.684651056
18	0.682990189	0.640631696	0.667361412
19	0.695751292	0.63777554	0.742657059
20	0.663440661	0.694981048	0.738440514
21	0.644633372	0.643585986	0.781746295
22	0.678544189	0.687859986	0.650575082
23	0.730521254	0.664981667	0.658664262
24	0.65522965	0.645293403	0.716169152
25	0.69437983	0.706944	0.730525693
26	0.680641463	0.683046786	0.611721215
27	0.660589205	0.655928076	0.633213787
28	0.691066276	0.575338318	0.659945112
29	0.663067071	0.727885889	0.622941891
30	0.72568669	0.728249534	0.761079434
Average Error	0.686970229	0.691215282	0.693393075

Table B4: Experiment Results with the use of 10 Hidden Layer Neurons

Simulation	Training Set Error (MSE)	Validation Set Error (MSE)	Test Set Error(MSE)
1	0.708314898	0.658814214	0.730291116
2	0.668583855	0.729388058	0.701850856
3	0.664717584	0.631192566	0.727750514
4	0.692927795	0.745891831	0.814125729
5	0.667611383	0.727473758	0.671314573
6	0.685407132	0.678129887	0.675553592
7	0.693778532	0.677402763	0.663964421
8	0.66824542	0.616463436	0.650390811
9	0.677365532	0.572263084	0.674561142
10	0.706144879	0.707500293	0.697598901
11	0.670579221	0.734965727	0.705254052
12	0.713478766	0.719901012	0.702086308
13	0.670829821	0.779855368	0.744228693
14	0.661874278	0.724789549	0.675125216
15	0.666342159	0.656890058	0.633310501
16	0.667926762	0.723047419	0.688555189
17	0.671688053	0.70828534	0.695714374
18	0.682804916	0.67009355	0.679390463
19	0.670804873	0.703318614	0.694960943
20	0.671690234	0.65629787	0.716474209
21	0.641790197	0.736285907	0.770731238
22	0.681886315	0.776981543	0.638625263
23	0.682036323	0.607339679	0.822090976
24	0.71068833	0.768330021	0.674432635
25	0.647814158	0.731880974	0.650991996
26	0.682528827	0.684476076	0.647944647
27	0.679790389	0.672152396	0.624317919
28	0.710905791	0.680229159	0.730771301
29	0.699912581	0.622070635	0.74555584
30	0.657979864	0.715917911	0.736277079
Average Error	0.679214962	0.693920957	0.699474683

Table B5: Experiment Results with the use of 12 Hidden Layer Neurons

Simulation	Training Set Error (MSE)	Validation Set Error (MSE)	Test Set Error(MSE)
1	0.683151237	0.677002358	0.644419905
2	0.691772186	0.698133096	0.737476583
3	0.679802869	0.583856915	0.698781935
4	0.669288422	0.604069389	0.715468959
5	0.72823056	0.669129866	0.652355512
6	0.683504293	0.754917733	0.78052712
7	0.662459603	0.68830394	0.717339969
8	0.679126099	0.678420606	0.600487724
9	0.693774231	0.780131519	0.740662653
10	0.67804251	0.646509904	0.778686539
11	0.673098743	0.670229966	0.817686025
12	0.668761436	0.677907363	0.689660524
13	0.790652165	0.732331676	0.829180647
14	0.674413075	0.613405016	0.732820239
15	0.681283299	0.715529489	0.74047214
16	0.675491913	0.704614935	0.759334325
17	0.686994831	0.680208713	0.679851364
18	0.665865936	0.723191897	0.689774877
19	0.676911226	0.738738971	0.742583152
20	0.66805417	0.733549323	0.666120154
21	0.66347673	0.688212836	0.705659708
22	0.65196817	0.637255293	0.636364022
23	0.676653095	0.599228846	0.731533342
24	0.663488612	0.703157845	0.696702076
25	0.701873811	0.714529875	0.71785327
26	0.696647799	0.67888871	0.617789821
27	0.700238094	0.707282998	0.689696306
28	0.694975713	0.691734244	0.747584007
29	0.64279014	0.666680246	0.654380242
30	0.699061752	0.621936754	0.641657596
Average Error	0.683395091	0.682636344	0.708430358

Table B6: Experiment Results with the use of 15 Hidden Layer Neurons

Simulation	Training Set Error (MSE)	Validation Set Error (MSE)	Test Set Error(MSE)
1	0.686501736	0.722974948	0.69758199
2	0.645565405	0.645383065	0.656578238
3	0.72707346	0.708021923	0.669540285
4	0.670298163	0.673257438	0.706205987
5	0.647077959	0.659514345	0.735177613
6	0.676192624	0.61294399	0.702215413
7	0.728794756	0.757005548	0.67717608
8	0.683725302	0.648570556	0.648196144
9	0.671216049	0.687271933	0.777405844
10	0.649322873	0.57073732	0.713141701
11	0.69325363	0.631303654	0.623412997
12	0.641997557	0.716825257	0.676782611
13	0.647529	0.684332507	0.68304243
14	0.700302129	0.729362361	0.657653738
15	0.671735221	0.626770841	0.712431368
16	0.711269907	0.725736339	0.692215032
17	0.718661797	0.752328624	0.669083786
18	0.686091967	0.652134335	0.66600756
19	0.679002332	0.599584543	0.675903598
20	0.671370873	0.696320859	0.777951276
21	0.6869675	0.693936882	0.811240424
22	0.674045437	0.677745386	0.688451348
23	0.684421504	0.647800277	0.780959928
24	0.661605712	0.630671918	0.683816384
25	0.660510305	0.58931051	0.745400442
26	0.709264608	0.596558081	0.628221943
27	0.729316463	0.772502758	0.681403973
28	0.681580292	0.643744418	0.693288222
29	0.646311795	0.639263183	0.773787593
30	0.662022085	0.68999538	0.714154712
Average Error	0.680100948	0.669396973	0.700614289

Table B7: Experiment Results with the use of 20 Hidden Layer Neurons

Simulation	Training Set Error (MSE)	Validation Set Error (MSE)	Test Set Error(MSE)
1	0.644075889	0.705327803	0.704872685
2	0.641150178	0.709764795	0.71424395
3	0.723854952	0.665804753	0.712938967
4	0.713086208	0.67747907	0.71824449
5	0.681715707	0.636802965	0.64267686
6	0.657433972	0.680953349	0.695600808
7	0.676677005	0.671692699	0.648985812
8	0.672688388	0.736518078	0.828842762
9	0.69706489	0.798467326	0.790974919
10	0.670819313	0.721493995	0.670108838
11	0.680235005	0.75575124	0.662205266
12	0.680296669	0.669260789	0.669482631
13	0.645240023	0.636321475	0.731856594
14	0.716340363	0.739655468	0.60792319
15	0.687765388	0.687272911	0.700934837
16	0.681805991	0.637707965	0.615850085
17	0.66587249	0.662268704	0.6328169
18	0.685339536	0.708155233	0.715842417
19	0.70781416	0.716227523	0.756283429
20	0.669843137	0.685911557	0.554292049
21	0.734992992	0.640016946	0.661320074
22	0.667435004	0.729490698	0.766741342
23	0.639974067	0.781478192	0.748748073
24	0.68265247	0.599905832	0.581512751
25	0.683536842	0.638749549	0.793600246
26	0.690733765	0.710729806	0.682542528
27	0.674542902	0.673270932	0.652238434
28	0.65165389	0.64095976	0.689332879
29	0.648426956	0.72256019	0.707214546
30	0.665294621	0.659614212	0.659732467
Average Error	0.677945426	0.689987127	0.690598694

Table B8: Experiment Results with the use of 24 Hidden Layer Neurons

Simulation	Training Set Error (MSE)	Validation Set Error (MSE)	Test Set Error(MSE)
1	0.656156899	0.728967576	0.617522688
2	0.687405713	0.636700886	0.67353764
3	0.681325981	0.65618274	0.603394871
4	0.666995336	0.779720872	0.651997893
5	0.671522863	0.665493637	0.718306138
6	0.660152427	0.615371364	0.735279243
7	0.667477993	0.643425064	0.637393423
8	0.658425321	0.612705822	0.748172455
9	0.669456618	0.646478148	0.650240335
10	0.668415627	0.583967185	0.688065192
11	0.667305105	0.649943783	0.670337688
12	0.672070765	0.657373085	0.691213648
13	0.64302068	0.688154147	0.637851548
14	0.670723414	0.690712888	0.664211481
15	0.666127647	0.759820982	0.571902749
16	0.663312627	0.695718247	0.691273258
17	0.631631481	0.723199909	0.717762748
18	0.660196248	0.722387065	0.651966893
19	0.651027823	0.684787731	0.659328858
20	0.663652818	0.640366649	0.689049436
21	0.669785711	0.722199062	0.663119271
22	0.678727387	0.615985216	0.696583632
23	0.643144911	0.697351368	0.667057858
24	0.688135033	0.621458639	0.674674428
25	0.714367824	0.706581979	0.751299745
26	0.672812877	0.675259579	0.693512512
27	0.657329459	0.69489572	0.715479596
28	0.634239149	0.664609674	0.704606511
29	0.667367774	0.756850545	0.65887969
30	0.669295168	0.65634166	0.632824557
Average Error	0.665720289	0.676433707	0.6742282

Table B9: Experiment Results with the use of 30 Hidden Layer Neurons

Simulation	Training Set Error (MSE)	Validation Set Error (MSE)	Test Set Error(MSE)
1	0.647720946	0.649228996	0.649459194
2	0.689200491	0.633690143	0.654902679
3	0.678829719	0.621571217	0.739682826
4	0.668268077	0.697960468	0.743521152
5	0.679567549	0.585312486	0.647981259
6	0.719248268	0.72054642	0.707596563
7	0.658667026	0.688925986	0.696023538
8	0.712431273	0.698993623	0.651199873
9	0.661978267	0.66888373	0.658058346
10	0.711425547	0.704969264	0.662246196
11	0.655662977	0.633454625	0.728510256
12	0.647791904	0.747085439	0.638121384
13	0.699653453	0.727371735	0.76600092
14	0.675133459	0.716593391	0.624154506
15	0.726500586	0.76765975	0.659003338
16	0.661680978	0.635575564	0.727761656
17	0.690013844	0.614571432	0.724847536
18	0.695803129	0.62752748	0.67344855
19	0.690321854	0.667772328	0.667910564
20	0.660214483	0.719636387	0.652706443
21	0.673944068	0.639892976	0.679328408
22	0.688157063	0.629945647	0.677718468
23	0.680044483	0.67892734	0.697806139
24	0.678894135	0.652731415	0.716353045
25	0.662651061	0.62982248	0.640898374
26	0.664266403	0.689535484	0.686488938
27	0.67608807	0.662266601	0.630315222
28	0.690723734	0.718265256	0.655960652
29	0.655369669	0.701250291	0.648717067
30	0.649319677	0.715389993	0.68219894
Average Error	0.678319073	0.674845265	0.679630734

Appendix C

Table C1: Experiment Results with the use of 2 Hidden Layer Neurons

Simulation	Training Set Error (MSE)	Validation Set Error (MSE)	Test Set Error(MSE)
1	1.656016052	1.713828815	1.692215008
2	1.668915921	1.908915038	1.876573492
3	1.900800911	1.865628795	1.915391726
4	1.743798777	1.696116632	1.819554482
5	1.678738028	1.623846758	1.511708519
6	1.696598326	1.938038004	1.644255247
7	1.70351775	1.742470877	1.527907319
8	1.664627037	1.789864332	1.691009659
9	1.81064654	1.64669392	1.741107389
10	1.649146799	1.805121822	1.678554745
11	1.666230302	1.69612932	1.832513037
12	1.576221948	1.916250907	1.760764125
13	1.767982528	1.75287325	1.756343293
14	1.664389222	1.841541143	1.438424469
15	1.806197021	1.78771782	1.800591159
16	1.688840476	1.625278101	1.532484897
17	1.664741963	1.604952267	1.553041984
18	1.682121065	1.643902359	1.539562915
19	1.796031477	1.602036947	1.81633221
20	1.691931674	1.576866158	1.745969928
21	1.740097503	1.644457021	1.942397581
22	1.721689364	1.747986437	1.815788805
23	1.861381698	1.906788931	1.883602409
24	2.117258581	2.239902454	1.933370281
25	1.677506398	1.720618956	1.695357891
26	1.688036303	1.668629841	1.718230086
27	1.816431832	2.056647296	1.881273933
28	1.605590512	1.53318513	1.751352968
29	1.749821177	1.685966242	1.850249713
30	1.655373775	1.856200073	1.752394714
Average Error	1.727022699	1.761281855	1.736610799

Table C2: Experiment Results with the use of 4 Hidden Layer Neurons

Simulation	Training Set Error (MSE)	Validation Set Error (MSE)	Test Set Error(MSE)
1	1.701423183	1.653554703	1.932118672
2	1.561519555	1.796592186	1.821799848
3	1.719521021	1.747240464	1.770380624
4	1.557481693	1.715520619	1.769140914
5	1.645333348	1.695757977	1.774607532
6	1.599486167	1.681201235	1.48033913
7	1.603506248	1.903030374	1.845073251
8	1.767411526	1.817759617	1.618872183
9	1.732734755	1.492214549	1.793141918
10	1.672791275	1.776028899	1.685028746
11	1.694648539	1.797936274	1.679418423
12	1.7715021	1.872627673	1.866409936
13	1.404433264	1.262785105	1.298258706
14	1.659324936	1.791293604	1.808519575
15	1.639962392	1.736768561	1.917403314
16	1.627446243	1.752151046	1.683392266
17	1.567390177	1.364541077	1.687532168
18	1.926392045	1.830555628	1.78028086
19	1.725984096	1.542365281	1.67100255
20	1.727425456	1.681362803	1.26193546
21	1.67077041	1.592457858	1.944408434
22	1.503312038	1.702733109	1.368520943
23	1.524415795	1.502357925	1.587801661
24	1.785867044	1.855755699	1.770699381
25	1.745810495	1.863267427	1.67529483
26	1.785566182	1.925908617	1.725584306
27	1.730355539	1.663813956	1.830762487
28	1.583765492	1.48851405	1.866552975
29	1.586604697	1.612037962	1.662796089
30	1.659378657	1.787165501	1.831563998
Average Error	1.662718812	1.696843326	1.713621373

Table C3: Experiment Results with the use of 8 Hidden Layer Neurons

Simulation	Training Set Error (MSE)	Validation Set Error (MSE)	Test Set Error(MSE)
1	1.808767782	1.502940512	1.765334493
2	1.608605599	1.540238259	1.726992194
3	1.683976789	1.5294674	1.85439071
4	1.557996485	1.431752378	1.506492513
5	1.521581937	1.395997955	1.585402443
6	1.618808625	1.518209935	1.836092874
7	1.589299698	1.728933616	1.443326761
8	1.503071672	1.504245989	1.614411853
9	1.885549189	1.670480196	1.848860401
10	1.513974299	1.72454472	1.524982692
11	1.399785442	1.636635437	1.503801565
12	1.49779055	1.723606709	1.395793921
13	1.520089168	1.515640773	1.672287795
14	1.766445576	1.745399376	1.58759051
15	1.745057016	1.750168958	1.587769937
16	1.653966271	1.74534385	1.585769155
17	1.369348606	1.490946333	1.451115057
18	1.651007642	1.56844775	1.778600013
19	1.640487237	1.687302993	1.881545131
20	1.380086472	1.427003956	1.473853939
21	1.877021388	1.932909212	1.740649607
22	1.503336414	1.580884074	1.735535922
23	1.681798259	1.803175833	1.754199448
24	1.652303531	1.770112623	1.701267366
25	1.630017156	1.867906015	1.694977536
26	1.648053205	1.384985553	1.688393233
27	1.762563361	1.577842624	1.621818888
28	1.730242851	1.570591433	1.827781948
29	1.54990044	1.628607593	1.524985803
30	1.737914555	1.525946329	1.7399942
Average Error	1.622961574	1.616008946	1.65513393

Table C4: Experiment Results with the use of 10 Hidden Layer Neurons

Simulation	Training Set Error (MSE)	Validation Set Error (MSE)	Test Set Error(MSE)
1	1.535899466	1.504338142	1.511504248
2	1.56824551	1.696698043	1.456044953
3	1.632996503	1.563649628	1.557622771
4	1.734256778	1.775137861	1.770269021
5	1.575175089	1.43822386	1.704750135
6	1.593495378	1.360798016	1.488580628
7	1.677125244	1.602929982	1.647326776
8	1.660140248	1.984745311	1.975871538
9	1.680150772	1.391449778	1.508943423
10	1.580116283	1.970779298	1.85091427
11	1.612293285	1.635808192	1.557621476
12	1.652950184	1.782130486	1.66644679
13	2.008293852	2.097974334	1.961789933
14	1.593331787	1.481565335	1.696224222
15	1.715848164	1.410932377	1.568257195
16	1.505348786	1.506970965	1.511411188
17	1.591167508	1.616942559	1.648125371
18	1.586734267	1.719364872	1.573302589
19	1.767698554	1.825184464	1.790531123
20	1.540157861	1.80891584	1.525668006
21	1.575772795	1.528811283	1.398037688
22	1.679918593	1.772844252	1.48000907
23	1.468056405	1.648251468	1.491385857
24	1.549159158	1.47573419	1.391413274
25	1.66593797	1.801603475	1.438402438
26	1.640174663	1.630063366	1.507041029
27	1.668113339	1.895279534	1.841149255
28	1.465283414	1.438228381	1.5408054
29	1.63981483	1.786732874	1.438943126
30	1.604467659	1.749939877	1.493055364
Average Error	1.625604145	1.663400935	1.599714939

Table C5: Experiment Results with the use of 12 Hidden Layer Neurons

Simulation	Training Set Error (MSE)	Validation Set Error (MSE)	Test Set Error(MSE)
1	1.571355443	1.683582707	1.772237624
2	1.477527538	1.59538205	1.65671493
3	1.617747058	1.533634654	1.718423816
4	1.663665436	1.808124454	1.769889477
5	1.603595487	1.526244044	1.608444335
6	1.818619568	2.005455092	1.927933346
7	1.677046477	1.973948891	1.539382469
8	1.535982724	1.545178459	1.944854999
9	1.755109625	1.853686797	1.726781289
10	1.728972884	1.639811373	1.623588046
11	1.500194526	1.325009663	1.690435373
12	1.489241072	1.590796303	1.684162599
13	1.678501251	1.711180369	1.460848417
14	1.643972671	1.625428389	1.397645667
15	1.578781781	1.85958933	1.654871869
16	1.456391358	1.402236702	1.428586936
17	1.644820348	1.719146702	1.849995805
18	1.743499643	1.850405429	1.908348085
19	1.657431718	1.508296232	1.581636338
20	1.695717136	1.778361669	1.739710339
21	1.486570008	1.571347256	1.628289808
22	1.643254816	1.853763881	1.720961446
23	1.740717563	1.501908782	1.808217439
24	1.713441849	1.86151524	1.974650653
25	1.644013716	1.696622965	1.830392673
26	1.513128046	1.516141411	1.341257271
27	1.681572395	1.667137163	1.760113187
28	1.620525412	1.751456117	1.693064301
29	1.550724273	1.70847087	1.716960705
30	1.657319408	1.783988543	1.699086655
Average Error	1.626314708	1.681595051	1.69524953

Table C6: Experiment Results with the use of 15 Hidden Layer Neurons

Simulation	Training Set Error (MSE)	Validation Set Error (MSE)	Test Set Error(MSE)
1	1.611682463	1.79386189	1.473028047
2	1.470843583	1.482249121	1.303380301
3	1.477297932	1.579117754	1.580789422
4	1.803208215	1.538107683	1.695114002
5	1.506823445	1.397021348	1.6552298
6	1.563893667	1.42441382	1.717353508
7	1.661044461	1.515957353	1.715247838
8	1.450559911	1.538878874	1.803137042
9	1.564357214	1.621853786	1.742228954
10	1.475518077	1.643220675	1.610282387
11	1.531780055	1.592642987	1.499446946
12	1.659846433	1.56576322	1.934301212
13	1.530873878	1.470411603	1.580876071
14	1.596676503	1.786766996	1.591614186
15	1.450671857	1.403847622	1.554580228
16	1.51796867	1.556795828	1.626498727
17	1.692697009	1.526244859	1.689430777
18	1.456471939	1.692005697	1.523747281
19	1.551660795	1.453574752	1.602837955
20	1.440858416	1.536754906	1.558484543
21	1.528699405	1.602062498	1.902113043
22	1.851107997	1.65267346	1.844777949
23	1.680563665	1.684343827	1.630052925
24	1.575001802	1.583036097	1.802023504
25	1.488277429	1.622772147	1.508368706
26	1.522001131	1.687277256	1.580537652
27	1.590362854	1.498022219	1.631871731
28	1.648138383	1.797800246	1.575540618
29	1.666644922	1.924912689	1.897906131
30	1.577825009	1.634213689	1.800008918
Average Error	1.571445237	1.593553497	1.654360347

Table C7: Experiment Results with the use of 20 Hidden Layer Neurons

Simulation	Training Set Error (MSE)	Validation Set Error (MSE)	Test Set Error(MSE)
1	1.443370691	1.487219244	1.566047027
2	1.382895541	1.337650027	1.462357501
3	1.57835011	1.429870051	1.552108653
4	1.611629099	1.468121131	1.591615342
5	1.720552645	1.6870914	1.438036844
6	1.582661221	1.492488951	1.438121348
7	1.517425119	1.301513339	1.725263877
8	1.786995375	2.061309576	1.872517742
9	1.757293916	1.939003033	1.804202346
10	1.586042126	1.642144801	1.777648819
11	1.636238082	1.801821064	1.699443727
12	1.623154773	1.621876806	1.723082229
13	1.576410012	1.667407872	1.571804185
14	1.617664355	1.673611494	1.801299799
15	1.511791108	1.531863938	1.449274399
16	1.599054131	1.530900994	1.591765135
17	1.617281102	1.580153768	1.532925748
18	1.730445967	1.6609215	1.635316581
19	1.511399319	1.271484997	1.566172021
20	1.720429651	1.632393084	1.9434375
21	1.597989059	1.592448131	1.619519758
22	1.709988502	1.69853364	1.582081302
23	1.589830704	1.864775801	1.588951804
24	1.524670944	1.651737847	1.545722504
25	1.571214664	1.473971339	1.723205843
26	1.605062887	1.684660469	1.459237537
27	1.526119193	1.476388961	1.749034624
28	1.471577841	1.480197851	1.507437319
29	1.60369175	1.73485129	1.440256524
30	1.536214063	1.696167835	1.426166232
Average Error	1.594914798	1.605752674	1.612801809

Table C8: Experiment Results with the use of 24 Hidden Layer Neurons

Simulation	Training Set Error (MSE)	Validation Set Error (MSE)	Test Set Error(MSE)
1	1.403621897	1.479315375	1.421510005
2	1.539524557	1.560761602	1.588567772
3	1.44985016	1.600100659	1.689733763
4	1.571096364	1.393682613	1.681674048
5	1.789774762	1.915357464	1.85411924
6	1.724253313	1.736066795	1.685901278
7	1.765196884	1.850034806	1.878766206
8	1.586155556	1.573582106	1.765640367
9	1.716848522	1.851896387	1.523727107
10	1.931399884	1.832143831	1.972136256
11	1.61675267	1.847178849	1.608209993
12	1.540582401	1.555132293	1.65250828
13	1.517082346	1.662221835	1.732844017
14	1.502180785	1.602503423	1.459984709
15	1.569430517	1.630157567	1.489956957
16	1.536218829	1.400016801	1.708134749
17	1.586419844	1.453757043	1.562019094
18	1.502308532	1.698435463	1.717616986
19	1.514742236	1.615633798	1.428006406
20	1.687319168	1.841374312	1.919780734
21	1.57993189	1.57459623	1.609359237
22	1.638795652	1.357260596	1.646986857
23	1.46800022	1.722906682	1.531767666
24	1.798677189	1.752090463	1.935664664
25	1.75146818	1.892565195	1.847622164
26	1.672578306	1.753301527	1.816421102
27	1.803601944	1.607044088	1.81386158
28	1.599938187	1.676429054	1.519367432
29	1.744912877	1.817784533	1.949543388
30	1.621910497	1.665900341	1.757317922
Average Error	1.624352472	1.663974391	1.692291666

Table C9: Experiment Results with the use of 30 Hidden Layer Neurons

Simulation	Training Set Error (MSE)	Validation Set Error (MSE)	Test Set Error(MSE)
1	1.875137868	1.984004747	1.847579867
2	1.882976822	1.752513353	1.998804148
3	1.735837266	1.901095078	1.626599117
4	1.558307866	1.53598618	1.471074125
5	1.483962626	1.484847086	1.388802272
6	1.547204546	1.848099539	1.567360311
7	1.725508038	1.847306869	1.994410112
8	1.790999314	1.681289817	1.870715667
9	1.678367264	1.716674362	1.4903873
10	1.595206448	1.703476287	1.50316256
11	1.778318323	1.678212263	1.908910572
12	1.685197741	1.822642356	1.870871585
13	1.766756502	1.644814994	1.770144119
14	1.605649108	1.665005516	1.669027237
15	1.681477736	1.515618113	1.693429278
16	1.773012434	1.967779651	1.832162279
17	1.667263584	1.885308776	1.580044199
18	1.57571175	1.552145494	1.336116101
19	1.497814427	1.813139576	1.713264062
20	1.757295994	1.775520674	1.859679943
21	1.494424173	1.457463095	1.451346398
22	1.554511912	1.612060325	1.469516528
23	1.674827954	1.785541603	1.731666144
24	1.630786377	1.629188867	1.787322069
25	1.839046536	1.701479747	1.910880354
26	1.661174389	1.530590445	1.567255659
27	1.866224207	1.887325357	1.815727215
28	1.684611403	1.670819683	1.634472693
29	1.499721934	1.645947362	1.550150228
30	1.663058551	1.706602714	1.734756743
Average Error	1.674346436	1.713416664	1.688187963

Appendix D

Table D1: Experiment Results with the use of 2 Hidden Layer Neurons

Simulation	Training Set Error (MSE)	Validation Set Error (MSE)	Test Set Error(MSE)
1	0.236378419	0.239034363	0.232368859
2	0.250854696	0.241758296	0.240624851
3	0.239663992	0.245115792	0.241122379
4	0.240290834	0.242917905	0.234343598
5	0.227136261	0.228759534	0.232388164
6	0.23687898	0.241008921	0.249247472
7	0.252614882	0.253102892	0.247777478
8	0.22724621	0.245069142	0.243021777
9	0.243399623	0.243833361	0.242205234
10	0.236427083	0.241148401	0.238905315
11	0.255049676	0.260772535	0.260891672
12	0.258489775	0.24467522	0.250156664
13	0.239552732	0.227699815	0.2320814
14	0.239547069	0.234554504	0.242621696
15	0.239174462	0.239039062	0.232335918
16	0.238876921	0.243293423	0.231047645
17	0.235267808	0.235464828	0.234621977
18	0.247808182	0.246677547	0.239248708
19	0.244431888	0.248678447	0.229891678
20	0.249771299	0.249587885	0.25262257
21	0.227863439	0.230740656	0.227908951
22	0.235549022	0.24686033	0.250076963
23	0.259974683	0.258270826	0.245430795
24	0.237486576	0.230154145	0.223233299
25	0.25079062	0.237830599	0.248903683
26	0.242314385	0.237140807	0.258070871
27	0.236986043	0.231626197	0.233303401
28	0.229330193	0.239566516	0.218797564
29	0.25322721	0.239696894	0.258581079
30	0.235752676	0.240062774	0.23279533
Average Error	0.241604521	0.241471387	0.240154233

Table D2: Experiment Results with the use of 4 Hidden Layer Neurons

Simulation	Training Set Error (MSE)	Validation Set Error (MSE)	Test Set Error(MSE)
1	0.236417588	0.233698076	0.233135731
2	0.234096411	0.234697101	0.241508211
3	0.224619125	0.223726463	0.225639337
4	0.236960451	0.240374141	0.242127841
5	0.229372549	0.227688772	0.230229953
6	0.224677843	0.22510748	0.221246688
7	0.236290726	0.235417404	0.221570142
8	0.225359725	0.22382915	0.220643035
9	0.219187862	0.230637459	0.223466516
10	0.221016304	0.226938417	0.227752567
11	0.240019405	0.251099467	0.245636669
12	0.236769692	0.23363363	0.234325631
13	0.225848178	0.233403013	0.219214629
14	0.227690477	0.227885398	0.235096088
15	0.226908199	0.23576755	0.239278819
16	0.238077439	0.23054	0.246408256
17	0.226368665	0.220770161	0.221701219
18	0.237901812	0.236618484	0.237525683
19	0.239719308	0.233996706	0.242167407
20	0.237898901	0.237896665	0.23534388
21	0.240919791	0.244652447	0.237711852
22	0.23132919	0.22691614	0.222836588
23	0.238972743	0.23227613	0.237762864
24	0.224631297	0.232905839	0.230294258
25	0.23738082	0.242740866	0.234709967
26	0.226207109	0.220495753	0.234713993
27	0.238576094	0.23744368	0.242542486
28	0.238273974	0.232062408	0.235872016
29	0.239419935	0.248206077	0.227032147
30	0.231650777	0.243653124	0.247825645
Average Error	0.232418746	0.2335026	0.233177337

Table D3: Experiment Results with the use of 8 Hidden Layer Neurons

Simulation	Training Set Error (MSE)	Validation Set Error (MSE)	Test Set Error(MSE)
1	0.227880525	0.231271935	0.23582548
2	0.226384968	0.214527622	0.22805228
3	0.234335379	0.233901307	0.238803795
4	0.236932183	0.227836089	0.23272224
5	0.236997337	0.233900158	0.225755857
6	0.220917614	0.237144218	0.215369046
7	0.239303515	0.23489239	0.239546616
8	0.222582915	0.230331396	0.231071003
9	0.235228118	0.244060439	0.24178465
10	0.23223875	0.230510763	0.234318556
11	0.233696223	0.223278523	0.243947326
12	0.216707141	0.222369258	0.220848791
13	0.234834853	0.238918937	0.233127262
14	0.222588869	0.233232697	0.221650083
15	0.241342491	0.237346882	0.246950568
16	0.231443078	0.218634072	0.229807014
17	0.234109069	0.234040555	0.242532444
18	0.235060255	0.238412171	0.25224906
19	0.228506631	0.235484002	0.219583167
20	0.234786074	0.234088894	0.234593146
21	0.214634681	0.219551619	0.223939518
22	0.22909119	0.233142723	0.236609209
23	0.234023478	0.231660307	0.249053786
24	0.226346999	0.231256124	0.228389931
25	0.226114813	0.23878307	0.236689641
26	0.219535739	0.232264626	0.223896865
27	0.239785619	0.238546862	0.229130405
28	0.229101668	0.235617059	0.238416979
29	0.228189058	0.226209538	0.216470321
30	0.237201543	0.228673991	0.240098575
Average Error	0.230330026	0.231662941	0.23304112

Table D4: Experiment Results with the use of 10 Hidden Layer Neurons

Simulation	Training Set Error (MSE)	Validation Set Error (MSE)	Test Set Error(MSE)
1	0.240496176	0.231517058	0.234578922
2	0.221376968	0.237625471	0.220698447
3	0.238182633	0.240303953	0.229198895
4	0.226924942	0.234464386	0.221572044
5	0.234990798	0.222757592	0.224327439
6	0.235760924	0.232667375	0.235885242
7	0.227637323	0.229363726	0.226986595
8	0.219638268	0.228865322	0.22363059
9	0.233171805	0.238903169	0.242429424
10	0.240331561	0.227924638	0.228639541
11	0.232192111	0.231638295	0.242087131
12	0.240923644	0.253603328	0.251229372
13	0.232673723	0.237783057	0.240364688
14	0.226528418	0.229875728	0.220600453
15	0.234466422	0.235044539	0.240919189
16	0.235913195	0.237758734	0.23073223
17	0.243844392	0.24056142	0.253368158
18	0.234072411	0.233406742	0.227571415
19	0.238232702	0.224472201	0.235107974
20	0.22719346	0.22794899	0.241670585
21	0.236520093	0.243472503	0.2490897
22	0.23159895	0.222216161	0.244011649
23	0.219250727	0.221985758	0.228350881
24	0.230766464	0.230753269	0.238258197
25	0.23102376	0.220724941	0.230844193
26	0.225167408	0.226212707	0.224213582
27	0.228013166	0.232394912	0.232675081
28	0.239686592	0.239654847	0.233386427
29	0.235925784	0.23242086	0.236070812
30	0.237474223	0.246601188	0.245826838
Average Error	0.232665968	0.233097429	0.234477523

Table D5: Experiment Results with the use of 12 Hidden Layer Neurons

Simulation	Training Set Error (MSE)	Validation Set Error (MSE)	Test Set Error(MSE)
1	0.23295818	0.211684221	0.22738149
2	0.226980976	0.211264741	0.228073261
3	0.234770767	0.23440925	0.238960463
4	0.231415474	0.235760029	0.237542998
5	0.234674065	0.246149501	0.244907889
6	0.241995903	0.228804956	0.238924287
7	0.230579619	0.237842716	0.238965268
8	0.237319338	0.22753074	0.249517287
9	0.234906994	0.22977016	0.222722603
10	0.233390691	0.240492856	0.227292485
11	0.229199247	0.220702018	0.235735567
12	0.223693409	0.214569901	0.215208526
13	0.225403181	0.219873736	0.224449649
14	0.229533929	0.217195024	0.234754814
15	0.224494918	0.229170056	0.219562158
16	0.228072237	0.225810071	0.241406416
17	0.232365201	0.240603878	0.237362106
18	0.224052841	0.222188486	0.225712253
19	0.22487817	0.222279945	0.21769476
20	0.233583278	0.224080225	0.233595643
21	0.237590334	0.226366374	0.236349179
22	0.231223291	0.231271257	0.230685528
23	0.234614198	0.231415725	0.229387613
24	0.23442272	0.241324814	0.233717447
25	0.238238379	0.23282502	0.236987142
26	0.2195346	0.227109885	0.21948211
27	0.228633959	0.232399621	0.226142556
28	0.230237094	0.235538935	0.23367058
29	0.233521247	0.218261729	0.237062753
30	0.234564646	0.229717095	0.231626909
Average Error	0.231228296	0.228213766	0.231829391

Table D6: Experiment Results with the use of 15 Hidden Layer Neurons

Simulation	Training Set Error (MSE)	Validation Set Error (MSE)	Test Set Error(MSE)
1	0.23849353	0.23760681	0.241190393
2	0.238024054	0.228841728	0.238904583
3	0.221080844	0.21558861	0.208413889
4	0.227068284	0.243969705	0.230108291
5	0.232416699	0.247296145	0.240362779
6	0.238834448	0.235105989	0.225411462
7	0.236990271	0.237735571	0.235639484
8	0.234646268	0.234526162	0.229692081
9	0.226211631	0.23654241	0.236132998
10	0.225346829	0.226614733	0.225763503
11	0.234504471	0.232992315	0.228860827
12	0.230560041	0.232236214	0.220081816
13	0.219868756	0.240010925	0.222321348
14	0.222920723	0.227954339	0.225574605
15	0.234943969	0.233029144	0.229829562
16	0.235825653	0.244574032	0.224820069
17	0.22645886	0.225144576	0.22804375
18	0.222206642	0.226013829	0.23192505
19	0.234928701	0.240348496	0.229945486
20	0.222591066	0.220660979	0.238300233
21	0.233748021	0.232953521	0.221659717
22	0.231107948	0.238541586	0.236145713
23	0.230732588	0.234510985	0.232951121
24	0.230459052	0.236668422	0.237977459
25	0.235085134	0.238304037	0.238484264
26	0.235088652	0.231807699	0.239625025
27	0.233560281	0.241225537	0.227229817
28	0.230906068	0.23758427	0.228382734
29	0.230142112	0.224966335	0.232903117
30	0.235099983	0.237433754	0.237989787
Average Error	0.230995053	0.234026295	0.230822365

Table D7: Experiment Results with the use of 20 Hidden Layer Neurons

Simulation	Training Set Error (MSE)	Validation Set Error (MSE)	Test Set Error(MSE)
1	0.233230026	0.215366457	0.240906803
2	0.227444322	0.231827719	0.230258276
3	0.223133206	0.237987699	0.217078525
4	0.22008439	0.220885522	0.222934146
5	0.224895194	0.230758169	0.228057918
6	0.232323242	0.243004424	0.236746619
7	0.240363695	0.236524962	0.243170738
8	0.239609924	0.238532448	0.236991795
9	0.219923316	0.224907869	0.214767113
10	0.228762163	0.239000408	0.23392547
11	0.238791428	0.244258407	0.229834997
12	0.237112994	0.235684701	0.244573434
13	0.242435565	0.24104179	0.24490712
14	0.224925558	0.238653387	0.220185525
15	0.24075235	0.224507063	0.246618068
16	0.226435057	0.211649043	0.221360906
17	0.220476628	0.232503127	0.226297552
18	0.220996057	0.216404797	0.218602632
19	0.231890541	0.227177409	0.244520897
20	0.22837518	0.230751032	0.232773128
21	0.221754407	0.226836236	0.222572416
22	0.222167698	0.223452551	0.21747153
23	0.219440896	0.222836501	0.220478001
24	0.240558384	0.242388414	0.249858869
25	0.231288663	0.232771231	0.225777332
26	0.236704765	0.256317378	0.227442627
27	0.234289414	0.228003904	0.235921826
28	0.234061839	0.236665452	0.231852017
29	0.241234958	0.239920312	0.248918751
30	0.230323846	0.249830333	0.237889457
Average Error	0.230459524	0.232681625	0.231756483

Table D8: Experiment Results with the use of 24 Hidden Layer Neurons

Simulation	Training Set Error (MSE)	Validation Set Error (MSE)	Test Set Error(MSE)
1	0.234405966	0.242632628	0.233575366
2	0.230270606	0.242174003	0.227849429
3	0.225808106	0.233041531	0.231798466
4	0.227708798	0.231088713	0.229678916
5	0.22190394	0.222395601	0.224832237
6	0.22351717	0.235280525	0.217666319
7	0.223241761	0.228204532	0.229376063
8	0.235809818	0.239272529	0.234404813
9	0.234376402	0.227486949	0.237454673
10	0.220282032	0.226399893	0.239550023
11	0.239536383	0.236744363	0.234135976
12	0.230172309	0.226184019	0.23683513
13	0.232332379	0.235170587	0.251113306
14	0.23339577	0.237914459	0.230933619
15	0.221591946	0.216006457	0.226952355
16	0.238901057	0.237238584	0.238883034
17	0.228435422	0.221458159	0.223868467
18	0.236539827	0.254006852	0.238071877
19	0.229833493	0.237184097	0.224221465
20	0.237194594	0.2283653	0.234477085
21	0.235301317	0.222166926	0.235523335
22	0.232401809	0.225172258	0.230195665
23	0.232934434	0.231361392	0.239185326
24	0.23795321	0.231687412	0.243898014
25	0.242352966	0.237656216	0.227636935
26	0.227464709	0.231834056	0.230464017
27	0.228467859	0.226654621	0.228164379
28	0.233429427	0.228765169	0.229328795
29	0.229308205	0.234391545	0.232754065
30	0.233056784	0.254387632	0.238423945
Average Error	0.231264283	0.232744234	0.232708437

Table D9: Experiment Results with the use of 30 Hidden Layer Neurons

Simulation	Training Set Error (MSE)	Validation Set Error (MSE)	Test Set Error(MSE)
1	0.23777317	0.24986245	0.245640766
2	0.222733198	0.224663492	0.233569644
3	0.234565727	0.235430619	0.235894588
4	0.233176689	0.235806125	0.230814764
5	0.232295814	0.222817536	0.233083732
6	0.231064293	0.22747038	0.234087231
7	0.248731588	0.257566632	0.235859727
8	0.229167553	0.226552058	0.236236257
9	0.229059287	0.22676396	0.234953409
10	0.221270395	0.226385561	0.221297177
11	0.227195242	0.228084647	0.224386528
12	0.220058044	0.220251232	0.223216745
13	0.221169314	0.222822772	0.221765953
14	0.236602794	0.229056374	0.240477218
15	0.235008826	0.229574766	0.241960181
16	0.222066506	0.220889686	0.233381843
17	0.222649412	0.213651621	0.229110833
18	0.233271714	0.252655767	0.235663401
19	0.22446603	0.227634571	0.238663685
20	0.229499645	0.237143414	0.220079688
21	0.226099052	0.230538769	0.239074591
22	0.234735485	0.23397857	0.232039037
23	0.222099853	0.227811079	0.224763976
24	0.220668576	0.226260812	0.235469882
25	0.237949503	0.23616712	0.232238605
26	0.231667199	0.217542292	0.222497161
27	0.233763505	0.225193013	0.236371518
28	0.225108451	0.226538554	0.231720889
29	0.235349477	0.234929734	0.240906292
30	0.226916061	0.215328853	0.224876006
Average Error	0.229539413	0.229645749	0.232336711

Appendix E

Table E1: Experiment Results with the use of 2 Hidden Layer Neurons

Simulation	Training Set Error (MSE)	Validation Set Error (MSE)	Test Set Error(MSE)
1	0.667713292	0.691425073	0.639642315
2	0.642514074	0.872872272	0.502283436
3	0.667013499	0.526764885	0.71429327
4	0.715390164	0.52208352	0.501038876
5	0.674102494	0.751217593	0.734375284
6	0.708219964	0.563734777	0.512974104
7	0.670066858	0.797450632	0.519053001
8	0.651708604	0.476837777	0.861928428
9	0.666535834	0.619837735	0.725047516
10	0.658007146	0.65620412	0.633493815
11	0.687029421	0.6107431	0.504077885
12	0.612865734	0.774273378	0.739504998
13	0.706356208	0.636615192	0.834371425
14	0.645392726	0.764717567	0.641209012
15	0.668336078	0.568585089	0.673898146
16	0.645092753	0.622816744	0.757306014
17	0.637245132	0.653861343	0.741510481
18	0.652503812	0.606135272	0.683262838
19	0.625826986	0.742181483	0.713644116
20	0.638455328	0.668324313	0.72231102
21	0.638034123	0.83007129	0.592178878
22	0.655788154	0.717363302	0.591148456
23	0.637634377	0.618720328	0.863709142
24	0.673078866	0.648952422	0.59835557
25	0.638515326	0.648327342	0.733056728
26	0.677252473	0.571440342	0.615383978
27	0.653461401	0.649396769	0.658673293
28	0.622575247	0.731583932	0.728387322
29	0.667587989	0.659724021	0.649085514
30	0.656764431	0.531282969	0.757475528
Average Error	0.658702283	0.657784819	0.67142268

Table E2: Experiment Results with the use of 4 Hidden Layer Neurons

Simulation	Training Set Error (MSE)	Validation Set Error (MSE)	Test Set Error(MSE)
1	0.609529601	0.758076547	0.904022926
2	0.644000875	0.579506318	0.721989652
3	0.632297997	0.671074637	0.743476332
4	0.594482158	0.904837865	0.650412522
5	0.663666618	0.747583344	0.533230308
6	0.663031949	0.848924088	0.427772847
7	0.644588837	0.692863627	0.630671662
8	0.695105708	0.580638931	0.490101379
9	0.649030912	0.581857745	0.758402036
10	0.610528497	0.768795226	0.745249496
11	0.585877923	0.813377991	0.794285579
12	0.670054964	0.709061668	0.51008259
13	0.672848379	0.615795274	0.582041458
14	0.646931461	0.674003167	0.623457694
15	0.627825507	0.645622009	0.478505151
16	0.697958401	0.498527276	0.618191063
17	0.665602645	0.688711236	0.548547113
18	0.696482913	0.611545952	0.51883404
19	0.626451753	0.710221999	0.689818161
20	0.661902292	0.532735321	0.788364994
21	0.689190125	0.421465973	0.719166312
22	0.615088997	0.681346575	0.779554674
23	0.683016667	0.525937829	0.624763268
24	0.659622352	0.390265121	0.598754191
25	0.692293493	0.493118399	0.629950996
26	0.59914735	0.84823205	0.713236556
27	0.67062869	0.540201684	0.685956451
28	0.667553929	0.563995534	0.69635546
29	0.614017532	0.786040481	0.670603881
30	0.690756383	0.536364149	0.603732594
Average Error	0.651317164	0.647357601	0.649317713

Table E3: Experiment Results with the use of 8 Hidden Layer Neurons

Simulation	Training Set Error (MSE)	Validation Set Error (MSE)	Test Set Error(MSE)
1	0.681622954	0.481240651	0.670200251
2	0.653309829	0.500647257	0.707144462
3	0.655534653	0.504658572	0.781397179
4	0.639819988	0.500733949	0.855382667
5	0.643209806	0.675617958	0.617503932
6	0.613392626	0.803704695	0.70025559
7	0.645225984	0.565512579	0.776837583
8	0.670292594	0.707726038	0.491819816
9	0.589994033	0.716107621	0.801486012
10	0.672457066	0.435033262	0.691249644
11	0.607369323	0.819004319	0.586712418
12	0.65543877	0.487946822	0.840813191
13	0.668845825	0.617369867	0.58769666
14	0.664854949	0.579705562	0.657032203
15	0.62410267	0.752808003	0.632159911
16	0.607241654	0.781131193	0.695674241
17	0.667511539	0.516901949	0.699681944
18	0.677971564	0.480809321	0.645735919
19	0.598806379	0.618413434	0.549494639
20	0.653936569	0.48893482	0.758853841
21	0.574938447	0.854004843	0.599294354
22	0.6539592	0.686307073	0.623875485
23	0.669560461	0.736838585	0.498241779
24	0.648102746	0.693771466	0.589592724
25	0.651431617	0.726328029	0.515063593
26	0.615064946	0.693503326	0.727486731
27	0.62156789	0.716308919	0.771073955
28	0.686791041	0.657813324	0.433890055
29	0.642027388	0.676429489	0.709798022
30	0.673272554	0.49173504	0.643189317
Average Error	0.644255169	0.632234932	0.661954604

Table E4: Experiment Results with the use of 10 Hidden Layer Neurons

Simulation	Training Set Error (MSE)	Validation Set Error (MSE)	Test Set Error(MSE)
1	0.64681649	0.719985374	0.577857861
2	0.667305588	0.699572792	0.626340582
3	0.548017647	0.595329987	0.864646542
4	0.644830235	0.689933358	0.596406972
5	0.707880304	0.527926923	0.61689104
6	0.677831597	0.821873482	0.57630001
7	0.6520426	0.736303518	0.647418645
8	0.645381996	0.739976105	0.563995295
9	0.656954395	0.586400428	0.739369361
10	0.636433835	0.582472165	0.565814113
11	0.678987001	0.505520069	0.592593243
12	0.657170411	0.547958912	0.646161139
13	0.647911616	0.643823031	0.729641106
14	0.711789029	0.390748533	0.67436575
15	0.705002937	0.544075677	0.559718199
16	0.634683814	0.751076485	0.612682237
17	0.634247605	0.517382431	0.895893964
18	0.650425211	0.584606012	0.511711717
19	0.651029892	0.646515788	0.60109127
20	0.628056811	0.716291154	0.526086434
21	0.583246093	0.828090476	0.813674477
22	0.598952078	0.804402481	0.673848703
23	0.698970421	0.40907674	0.618192164
24	0.669711083	0.629202445	0.64172781
25	0.667168605	0.593883672	0.953561695
26	0.622545124	0.583772092	0.823475834
27	0.681256828	0.600878428	0.589822262
28	0.64976612	0.736715359	0.576497971
29	0.650821252	0.692464712	0.498810391
30	0.633463115	0.580755465	0.579712706
Average Error	0.651289991	0.633567136	0.649810316

Table E5: Experiment Results with the use of 12 Hidden Layer Neurons

Simulation	Training Set Error (MSE)	Validation Set Error (MSE)	Test Set Error(MSE)
1	0.624596793	0.642224588	0.772254634
2	0.616597043	0.499549352	0.700746233
3	0.678257263	0.503124233	0.684792724
4	0.642770401	0.876299078	0.39450962
5	0.629052243	0.711993073	0.76420939
6	0.61256436	0.630918657	0.611567227
7	0.611913684	0.692429726	0.65487001
8	0.676038953	0.631986539	0.563710573
9	0.68820909	0.601468477	0.595651283
10	0.659232147	0.530573714	0.667375794
11	0.613060739	0.696579205	0.780473792
12	0.679000604	0.478494246	0.619250715
13	0.681397015	0.57310387	0.63600688
14	0.655877517	0.495861586	0.828485652
15	0.635695989	0.42754227	0.902231649
16	0.601607642	0.632617259	0.648454505
17	0.650836777	0.59884212	0.604991985
18	0.667745815	0.689742391	0.647924435
19	0.615278698	0.762704537	0.763874837
20	0.736714381	0.56214995	0.868774571
21	0.617064554	0.730844423	0.664122658
22	0.682756644	0.867620222	0.669548139
23	0.685271543	0.653357664	0.581635931
24	0.718069657	0.56373867	0.40585666
25	0.620772501	0.683586869	0.657967426
26	0.700801648	0.492311508	0.616878901
27	0.639670344	0.69940324	0.695418712
28	0.614517958	0.701929586	0.735268546
29	0.693214503	0.600262022	0.619710601
30	0.653671478	0.551250788	0.684238359
Average Error	0.653408599	0.626083662	0.668026748

Table E6: Experiment Results with the use of 15 Hidden Layer Neurons

Simulation	Training Set Error (MSE)	Validation Set Error (MSE)	Test Set Error(MSE)
1	0.717503906	0.665363883	0.446204754
2	0.63136588	0.810739319	0.69891084
3	0.630520177	0.77842037	0.504515474
4	0.615777461	0.767976027	0.645379912
5	0.63651573	0.573582944	0.582265114
6	0.653834649	0.549382651	0.567963527
7	0.625039128	0.791016705	0.710685387
8	0.692099655	0.64751755	0.603328944
9	0.686717014	0.803476607	0.503954958
10	0.630578432	0.78562071	0.697952131
11	0.60252887	0.619860896	0.561369923
12	0.712245414	0.507622785	0.471413146
13	0.632690117	0.589063025	0.740847142
14	0.663093252	0.653175071	0.585816398
15	0.667649293	0.470321332	0.716681213
16	0.727626135	0.522037133	0.537058144
17	0.658793455	0.607006569	0.578545768
18	0.652646396	0.678727157	0.433681071
19	0.715728153	0.643350329	0.636376615
20	0.578252429	0.725483797	0.550032011
21	0.648456921	0.614941881	0.713482064
22	0.629489509	0.764042085	0.570897753
23	0.602946093	0.781404099	0.555949199
24	0.669407894	0.637110611	0.625989003
25	0.644722049	0.623189193	0.679942557
26	0.590750991	0.689594932	0.857711624
27	0.636086651	0.754190052	0.605771421
28	0.715011338	0.537887099	0.363586277
29	0.660107888	0.67056929	0.713811988
30	0.635957131	0.587190794	0.603649799
Average Error	0.652138067	0.661662163	0.602125805

Table E7: Experiment Results with the use of 20 Hidden Layer Neurons

Simulation	Training Set Error (MSE)	Validation Set Error (MSE)	Test Set Error(MSE)
1	0.612053936	0.669157534	0.620829738
2	0.580105495	0.802128601	0.677205146
3	0.653786113	0.556375552	0.635661049
4	0.650134001	0.571664557	0.781470668
5	0.614819035	0.822795363	0.612998746
6	0.681901013	0.602214438	0.616751094
7	0.647625183	0.555469108	0.706364679
8	0.599928448	0.61130774	0.952648124
9	0.60614755	0.666447903	0.826399125
10	0.997800421	1.210894175	1.200166603
11	0.605442423	0.892189714	0.42681525
12	0.672709177	0.628597677	0.572349785
13	0.621724014	0.644648657	0.671523219
14	0.625297087	0.638629839	0.750872284
15	0.570292431	0.644491038	0.631212696
16	0.616876516	0.577401705	0.657368127
17	0.610083278	0.805499661	0.685050589
18	0.719839097	0.52208492	0.701989052
19	0.670685605	0.55475011	0.582140589
20	0.584313128	0.706329912	0.716845669
21	0.729117492	0.474399808	0.556055881
22	0.605715064	0.703319066	0.738969011
23	0.586559201	0.58483264	0.87771036
24	0.634163646	0.715323939	0.5905825
25	0.67952308	0.630924339	0.557871558
26	0.647256893	0.702241587	0.475773124
27	0.647071358	0.515211629	0.684201071
28	0.629545723	0.622568707	0.551237453
29	0.641074957	0.81199136	0.539765037
30	0.643461183	0.787619074	0.452520623
Average Error	0.646168418	0.674383678	0.668378295

Table E8: Experiment Results with the use of 24 Hidden Layer Neurons

Simulation	Training Set Error (MSE)	Validation Set Error (MSE)	Test Set Error(MSE)
1	0.654072428	0.681572758	0.587593853
2	0.627713578	0.687633583	0.882082657
3	0.630165149	0.603879608	0.656818374
4	0.666635272	0.735333533	0.670854629
5	0.653773317	0.529038253	0.624308878
6	0.654020716	0.755306194	0.498773042
7	0.632399636	0.782950236	0.676294464
8	0.664838404	0.656678749	0.580309455
9	0.602585829	0.76555648	0.62645708
10	0.592999804	0.899205327	0.501915835
11	0.646832086	0.542168097	0.490774744
12	0.612365504	0.713434275	0.522143349
13	0.605769086	0.836932585	0.675981986
14	0.632938431	0.69853933	0.459580133
15	0.609913775	0.775529705	0.774798716
16	0.621565198	0.517230522	0.736811915
17	0.692619444	0.528715261	0.815168629
18	0.620082571	0.749641937	0.690641565
19	0.659246588	0.542920012	0.586859686
20	0.65670955	0.455616668	0.723896645
21	0.642287694	0.603120582	0.681001168
22	0.574668326	0.714765989	0.638706522
23	0.718262695	0.548022989	0.437565144
24	0.696649159	0.636128691	0.671902013
25	0.639747112	0.615719134	0.612648505
26	0.608713237	0.751697757	0.65821042
27	0.602926069	0.64103157	0.546009822
28	0.685273658	0.583255026	0.618842652
29	0.623833448	0.77045839	0.668959674
30	0.640957529	0.533771765	0.770397956
Average Error	0.639018843	0.661861834	0.636210317

Table E9: Experiment Results with the use of 30 Hidden Layer Neurons

Simulation	Training Set Error (MSE)	Validation Set Error (MSE)	Test Set Error(MSE)
1	0.60385803	1.004198886	0.674169626
2	0.690748137	0.588149285	0.698824372
3	0.638661059	0.621910605	0.608617551
4	0.686148262	0.641902846	0.605795964
5	0.664262937	0.600337866	0.705009589
6	0.611321443	0.700601665	0.62236577
7	0.591993474	0.714468901	0.564247704
8	0.57768389	0.826389096	0.683972808
9	0.736645277	0.52377495	0.51068211
10	0.597210611	0.506334446	0.703350317
11	0.688924881	0.643592116	0.634825292
12	0.641799867	0.666845223	0.547607421
13	0.586426661	0.744074785	0.749070679
14	0.657530671	0.624933122	0.659492677
15	0.566247514	0.753648389	0.570302698
16	0.64548582	0.497883173	0.726421727
17	0.737606392	0.668689875	0.718262106
18	0.634147799	0.499256332	0.541317721
19	0.614831349	0.89264729	0.63831838
20	0.645176379	0.529180211	0.774723314
21	0.653488816	0.776417534	0.381999877
22	0.591090322	0.735849885	0.673538777
23	0.649513069	0.582640229	0.659217601
24	0.677411262	0.568949243	0.688176091
25	0.656611459	0.431591929	0.686908301
26	0.687312125	0.729629169	0.554454049
27	0.640416137	0.761565572	0.350044801
28	0.67354587	0.606219963	0.748174324
29	0.621079252	0.639022074	0.709861724
30	0.683271727	0.428279701	0.803439979
Average Error	0.645015016	0.650299479	0.639773112

Appendix F

Table F1: Experiment Results with the use of 2 Hidden Layer Neurons

Simulation	Training Set Error (MSE)	Validation Set Error (MSE)	Test Set Error(MSE)
1	0.114845936	0.116040766	0.126949535
2	0.013937963	0.016865107	0.010908025
3	0.01696825	0.013043887	0.016779125
4	0.122753802	0.100076222	0.130280107
5	0.01484339	0.018954629	0.014462688
6	0.013710807	0.011719951	0.015984542
7	0.012939286	0.01405091	0.011949086
8	0.122270045	0.12742026	0.119211673
9	0.116554319	0.123580172	0.113709477
10	0.013236681	0.016146956	0.010255974
11	0.014045768	0.018563488	0.011058649
12	0.018990854	0.019364979	0.01767949
13	0.013630994	0.014645002	0.010483787
14	0.122121409	0.115172645	0.11818469
15	0.115559016	0.134674518	0.130576611
16	0.014097596	0.014878977	0.013296507
17	0.082810365	0.093229761	0.078343091
18	0.122276469	0.114893745	0.111528655
19	0.012831026	0.014521079	0.016787226
20	0.011555967	0.019214983	0.015146477
21	0.123329529	0.123142293	0.103972059
22	0.122206816	0.115277252	0.118280892
23	0.013610601	0.013206121	0.012374045
24	0.013542761	0.010187384	0.021503097
25	0.014264311	0.012493049	0.01145576
26	0.016713389	0.014303432	0.016630634
27	0.117221754	0.126163484	0.129309145
28	0.015146903	0.01209564	0.00403826
29	0.11448473	0.126198793	0.12349105
30	0.012817331	0.01504111	0.013425844
Average Error	0.055110602	0.05617222	0.054935207

Table F2: Experiment Results with the use of 4 Hidden Layer Neurons

Simulation	Training Set Error (MSE)	Validation Set Error (MSE)	Test Set Error(MSE)
1	0.015115828	0.018218932	0.019101247
2	0.015631577	0.018232824	0.016773405
3	0.015869369	0.010702945	0.010397716
4	0.015007248	0.019063783	0.017158618
5	0.014022313	0.017305239	0.02322116
6	0.0132023	0.015881132	0.014887189
7	0.019932646	0.022515398	0.019721838
8	0.013038051	0.016499984	0.019438773
9	0.01629779	0.015115574	0.010711554
10	0.011946172	0.02125933	0.011140371
11	0.01545279	0.018352154	0.013807938
12	0.015998779	0.013163104	0.010642433
13	0.014809195	0.015629922	0.017006965
14	0.014475772	0.010602666	0.012861635
15	0.016102004	0.009896368	0.015971292
16	0.019966022	0.015346268	0.014800246
17	0.014176757	0.017217688	0.011904339
18	0.014802031	0.01418719	0.012300117
19	0.015024604	0.015087395	0.013622795
20	0.015735129	0.012534169	0.016129122
21	0.012933934	0.015533271	0.01913282
22	0.015560815	0.009819841	0.009866731
23	0.016956517	0.013944501	0.0106282
24	0.015642442	0.018295373	0.016672667
25	0.017316322	0.017975969	0.018674864
26	0.018149317	0.010397402	0.021386514
27	0.014812078	0.015767831	0.022861057
28	0.01617801	0.01386089	0.016725837
29	0.019521318	0.018611529	0.021405063
30	0.019046291	0.018912047	0.010523112
Average Error	0.015757447	0.015664357	0.015649187

Table F3: Experiment Results with the use of 8 Hidden Layer Neurons

Simulation	Training Set Error (MSE)	Validation Set Error (MSE)	Test Set Error(MSE)
1	0.015514516	0.013304821	0.025174979
2	0.014576764	0.016628186	0.016857828
3	0.013563793	0.016817833	0.012678149
4	0.017055843	0.015664196	0.019090476
5	0.018514942	0.014613967	0.02129795
6	0.01626642	0.012789124	0.012168348
7	0.015886323	0.018977425	0.016722845
8	0.017241706	0.011286062	0.01624925
9	0.016742953	0.01248488	0.017531922
10	0.015232839	0.01556589	0.015053018
11	0.017715122	0.015177777	0.020700919
12	0.013758777	0.012908518	0.022064971
13	0.013898923	0.015985817	0.016721807
14	0.016905668	0.01518528	0.014440182
15	0.016421864	0.018037546	0.012818958
16	0.017226776	0.018863489	0.014612888
17	0.016907799	0.019208004	0.016229079
18	0.020566607	0.022013991	0.017497217
19	0.017185043	0.016240747	0.017418758
20	0.016395208	0.015943401	0.019297852
21	0.018404061	0.014623424	0.023452737
22	0.01673129	0.014706572	0.010049485
23	0.016902596	0.013317978	0.017760189
24	0.015787516	0.019157877	0.015155997
25	0.01542527	0.016870794	0.017716684
26	0.016580665	0.015053596	0.027202375
27	0.016139203	0.01652126	0.020093157
28	0.018291239	0.016006447	0.015385356
29	0.01872671	0.015767061	0.020367753
30	0.014115002	0.017227814	0.01704051
Average Error	0.016489381	0.015898326	0.017628388

Table F4: Experiment Results with the use of 10 Hidden Layer Neurons

Simulation	Training Set Error (MSE)	Validation Set Error (MSE)	Test Set Error(MSE)
1	0.025443131	0.028679524	0.023638157
2	0.017188332	0.015366743	0.016622522
3	0.01813184	0.022423104	0.019818414
4	0.015688326	0.013868225	0.016999417
5	0.015834834	0.014016587	0.014980548
6	0.018604681	0.019181157	0.018540244
7	0.019108774	0.019326161	0.016901194
8	0.016025266	0.014487583	0.013664733
9	0.01951408	0.012706056	0.015478966
10	0.017151726	0.016531622	0.015816954
11	0.017840688	0.016860469	0.020792861
12	0.021928476	0.022930172	0.018982192
13	0.01915612	0.021698217	0.01594559
14	0.021504452	0.020456077	0.014531285
15	0.019500518	0.025739888	0.016638959
16	0.017812059	0.019134115	0.015108307
17	0.015495167	0.016019268	0.022638726
18	0.019322976	0.018630128	0.022790015
19	0.017284668	0.020171545	0.019610053
20	0.016888454	0.017737122	0.013964506
21	0.015533702	0.019813359	0.019564639
22	0.018748379	0.013304339	0.012909621
23	0.014860008	0.021151419	0.016985094
24	0.016921163	0.023424557	0.012920029
25	0.018608863	0.016065833	0.022753763
26	0.017886218	0.014696263	0.012264595
27	0.019111102	0.022201448	0.024360256
28	0.022404231	0.01765961	0.020020704
29	0.022102375	0.024200653	0.023014961
30	0.022781418	0.026930351	0.020074463
Average Error	0.018612734	0.019180387	0.017944392

Table F5: Experiment Results with the use of 12 Hidden Layer Neurons

Simulation	Training Set Error (MSE)	Validation Set Error (MSE)	Test Set Error(MSE)
1	0.015756669	0.021403713	0.01871602
2	0.01620617	0.018303225	0.020075393
3	0.016701861	0.018985885	0.010023039
4	0.016666441	0.016747626	0.019494693
5	0.015541649	0.019497495	0.015976876
6	0.015835639	0.022270776	0.020966082
7	0.016120691	0.01609243	0.017928378
8	0.017630691	0.011480376	0.018870719
9	0.015256053	0.016371554	0.015349694
10	0.018274123	0.013795462	0.023616617
11	0.020924658	0.020501112	0.016306048
12	0.01511559	0.014956624	0.018717599
13	0.015414543	0.010982498	0.01680195
14	0.017911077	0.016563779	0.02069509
15	0.019009826	0.012401617	0.014025592
16	0.015904608	0.015659666	0.018460964
17	0.018993623	0.017902374	0.017575063
18	0.018531468	0.020043861	0.01808011
19	0.018427091	0.016040163	0.014566574
20	0.020335293	0.016165574	0.015389235
21	0.015906799	0.015692236	0.017146341
22	0.018618196	0.019595201	0.015915437
23	0.017973153	0.018695183	0.021001884
24	0.021699149	0.024679924	0.024305969
25	0.018153482	0.017598051	0.020918751
26	0.017622821	0.017841133	0.014976036
27	0.016797814	0.015073757	0.014428622
28	0.017215965	0.01978079	0.021583975
29	0.016875941	0.01534	0.01114678
30	0.021656822	0.021292584	0.034258366
Average Error	0.017569264	0.017391822	0.01824393

Table F6: Experiment Results with the use of 15 Hidden Layer Neurons

Simulation	Training Set Error (MSE)	Validation Set Error (MSE)	Test Set Error(MSE)
1	0.016161235	0.014026028	0.020309079
2	0.017754333	0.012493261	0.016725932
3	0.016569109	0.025228367	0.017871372
4	0.016044391	0.012107916	0.016151993
5	0.01669761	0.011169471	0.02153247
6	0.021580037	0.019245521	0.021318239
7	0.017918196	0.016498691	0.020511717
8	0.019619934	0.017040264	0.017350234
9	0.022145102	0.021705899	0.019795238
10	0.017926523	0.016774614	0.020786178
11	0.019920903	0.022908478	0.020730247
12	0.019483244	0.021439176	0.015270377
13	0.016527634	0.02045347	0.017250197
14	0.014652527	0.016959853	0.019869305
15	0.020382253	0.016277783	0.018161664
16	0.022311043	0.028519875	0.026481125
17	0.0253355	0.026848361	0.02503446
18	0.018182628	0.015925235	0.019067254
19	0.017251902	0.023592588	0.015974328
20	0.016215791	0.016775344	0.017876171
21	0.017125227	0.017357298	0.015071691
22	0.018798685	0.011995278	0.015297154
23	0.017259378	0.01583904	0.01197171
24	0.018705211	0.019316954	0.015407256
25	0.018930799	0.020363068	0.016825089
26	0.020440476	0.017456789	0.021002213
27	0.01920116	0.01926816	0.016572215
28	0.019119663	0.018139954	0.012467373
29	0.019356228	0.015786221	0.017766969
30	0.016599148	0.0213553	0.017241372
Average Error	0.018607196	0.018428942	0.018256354

Table F7: Experiment Results with the use of 20 Hidden Layer Neurons

Simulation	Training Set Error (MSE)	Validation Set Error (MSE)	Test Set Error(MSE)
1	0.01937282	0.018291966	0.012166577
2	0.018889538	0.015936114	0.017341873
3	0.020143162	0.015675289	0.022720357
4	0.019364844	0.016137906	0.022645952
5	0.019155674	0.01581808	0.016786442
6	0.017399133	0.026908491	0.015001928
7	0.014978839	0.018370469	0.017320631
8	0.019146099	0.017466195	0.016952214
9	0.018017656	0.015384463	0.013434255
10	0.017612284	0.01477512	0.023880715
11	0.018754557	0.020054527	0.021886374
12	0.019228019	0.02011105	0.02075904
13	0.019016037	0.017072954	0.017924779
14	0.01823724	0.015059522	0.017763641
15	0.018121693	0.023030565	0.018820591
16	0.019578489	0.024043839	0.025294103
17	0.015048036	0.017787899	0.018093215
18	0.018659626	0.01583757	0.018945299
19	0.014276049	0.018927476	0.019991156
20	0.018448155	0.023328144	0.027369701
21	0.019106399	0.022774597	0.024549656
22	0.020001555	0.024424515	0.024035675
23	0.023487443	0.01908468	0.01639416
24	0.01726014	0.015692395	0.022414346
25	0.016325711	0.014157574	0.025674187
26	0.01851629	0.019434452	0.018125726
27	0.020310063	0.014954756	0.021738923
28	0.01731609	0.020595033	0.018989202
29	0.019029328	0.020678635	0.016108453
30	0.020755098	0.023327872	0.016311681
Average Error	0.018518536	0.018838072	0.019648028

Table F8: Experiment Results with the use of 24 Hidden Layer Neurons

Simulation	Training Set Error (MSE)	Validation Set Error (MSE)	Test Set Error(MSE)
1	0.022077938	0.017688649	0.020087145
2	0.021426066	0.019394398	0.023418337
3	0.019616308	0.018261375	0.021010527
4	0.017285456	0.022391012	0.016003216
5	0.018296467	0.016684067	0.023904087
6	0.017475798	0.017049303	0.020554082
7	0.015586737	0.021196627	0.01928545
8	0.018403672	0.021066782	0.017123576
9	0.023145228	0.023775852	0.026348147
10	0.020647943	0.025279341	0.024789109
11	0.020977598	0.02127528	0.019337629
12	0.018113755	0.017270068	0.015971635
13	0.020597745	0.01926635	0.026993276
14	0.015740491	0.0216707	0.020330236
15	0.018500498	0.017264158	0.014907976
16	0.020283919	0.023101027	0.017307358
17	0.016175267	0.01852986	0.019569569
18	0.022564343	0.020815379	0.014721321
19	0.018289441	0.022811718	0.022375351
20	0.016984406	0.017868009	0.016193051
21	0.018462214	0.018639279	0.021360201
22	0.017466125	0.017686164	0.020759361
23	0.020620968	0.017409795	0.022545497
24	0.018673551	0.023102096	0.019520507
25	0.019658704	0.019988826	0.023409966
26	0.015943418	0.020274716	0.023379358
27	0.017727805	0.020767755	0.018889979
28	0.018019965	0.023608019	0.012651199
29	0.019290644	0.02103875	0.020644501
30	0.019638499	0.020613028	0.019313526
Average Error	0.018923032	0.020192946	0.020090172

Table F9: Experiment Results with the use of 30 Hidden Layer Neurons

Simulation	Training Set Error (MSE)	Validation Set Error (MSE)	Test Set Error(MSE)
1	0.021826665	0.02352665	0.022374986
2	0.020619442	0.017857479	0.024895334
3	0.016309333	0.018212973	0.019344155
4	0.018906839	0.018543006	0.023420731
5	0.022545457	0.028151942	0.029520882
6	0.020708317	0.020280308	0.020020989
7	0.018573138	0.021354054	0.020254409
8	0.021583848	0.020644621	0.025608276
9	0.019338751	0.022760238	0.018103636
10	0.017010955	0.022023278	0.010140958
11	0.019426397	0.015490423	0.016808223
12	0.017410772	0.020289728	0.021145902
13	0.018965593	0.016055712	0.020835027
14	0.018782712	0.019697642	0.02371784
15	0.020465464	0.0224925	0.029614379
16	0.019950584	0.022289383	0.021736807
17	0.020515357	0.019748547	0.014666111
18	0.017708202	0.022009027	0.024390611
19	0.017522075	0.018534942	0.02616724
20	0.018831101	0.026492335	0.023716075
21	0.022327193	0.023139977	0.023449257
22	0.023034036	0.024327531	0.02009028
23	0.020158656	0.023000573	0.019604229
24	0.018111547	0.020620675	0.026518825
25	0.017982845	0.020324042	0.023113171
26	0.01952029	0.025984317	0.017813277
27	0.016110817	0.024471954	0.016988395
28	0.019539424	0.019852422	0.026782198
29	0.018207076	0.016046457	0.016806877
30	0.017724334	0.017481296	0.017581483
Average Error	0.019323907	0.021056801	0.021507685

Appendix G

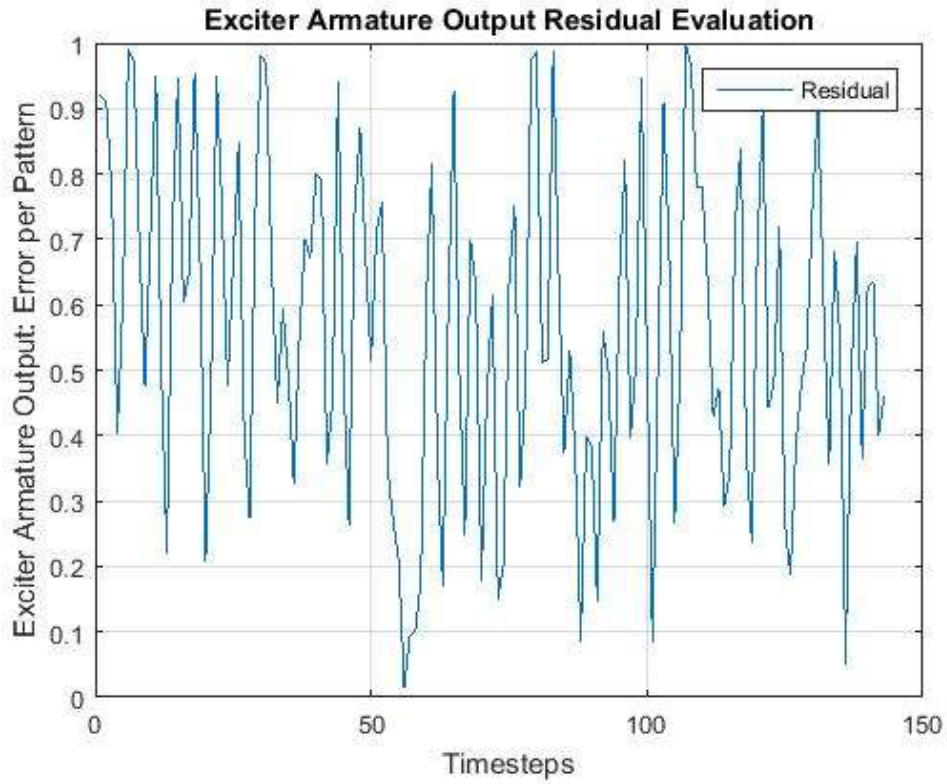


Figure G1: Sampling Number M = 2

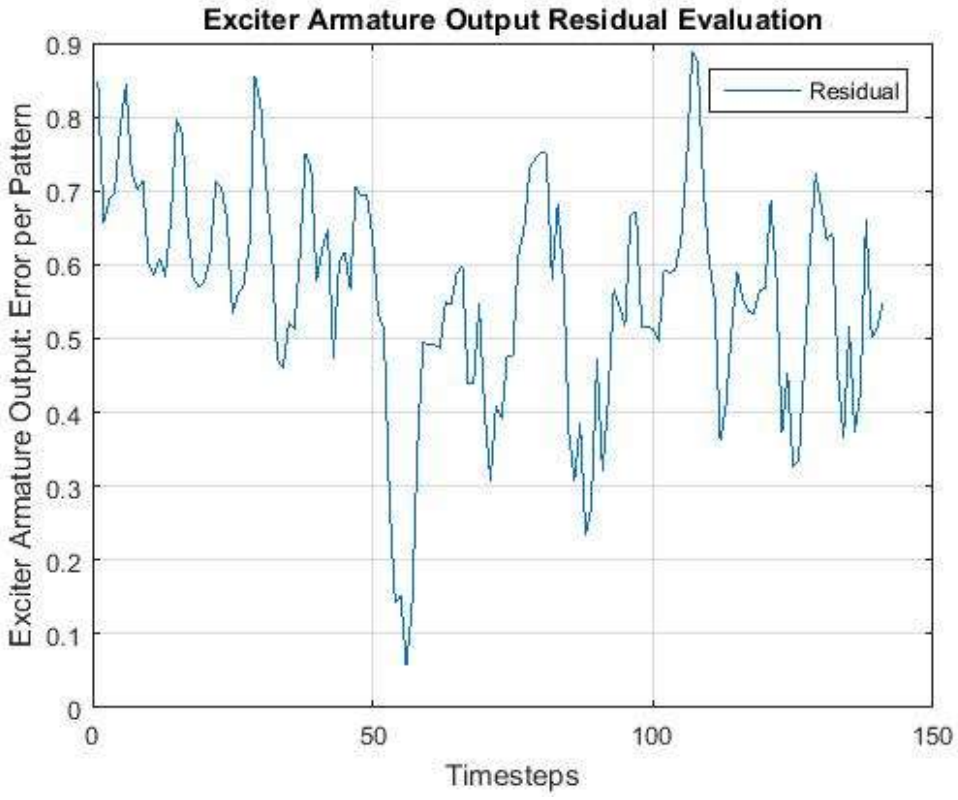


Figure G2: Sampling Number M = 4

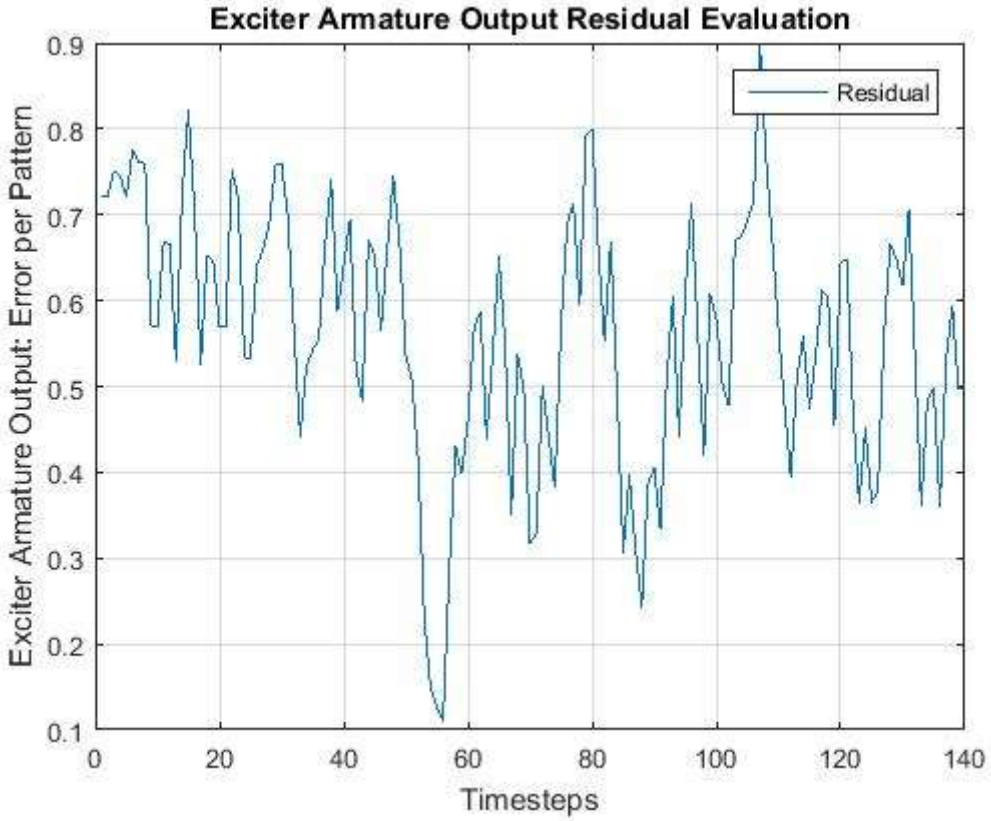


Figure G3: Sampling Number M = 5

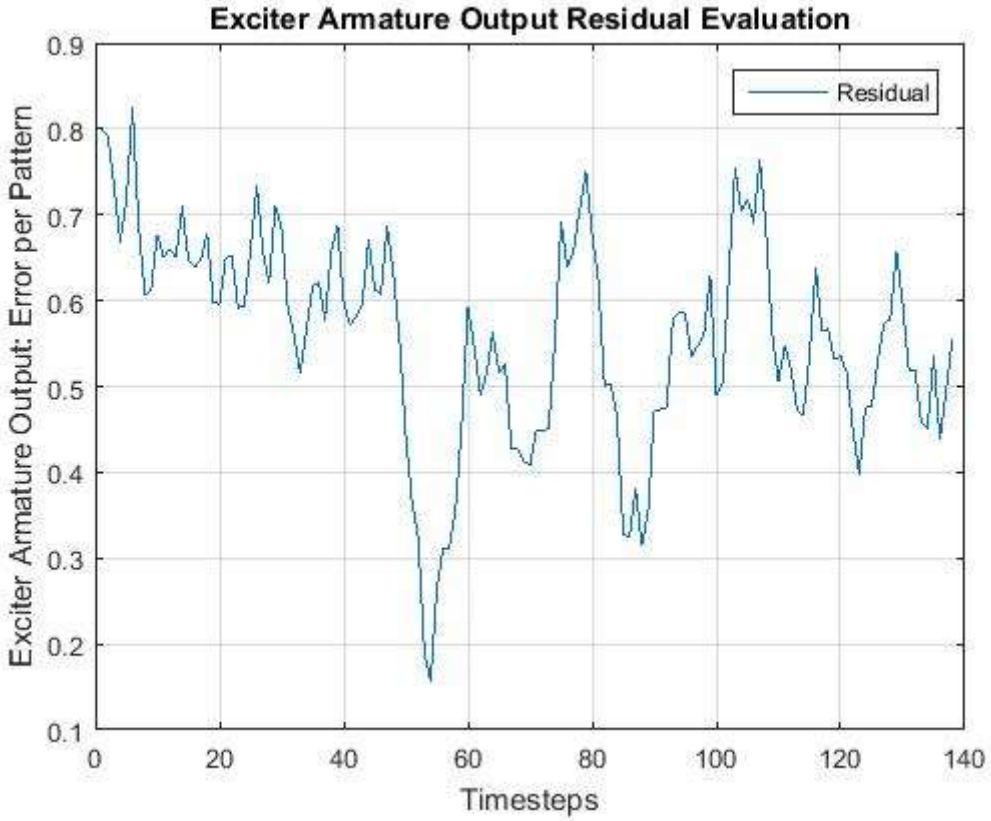


Figure G4: Sampling Number M = 7

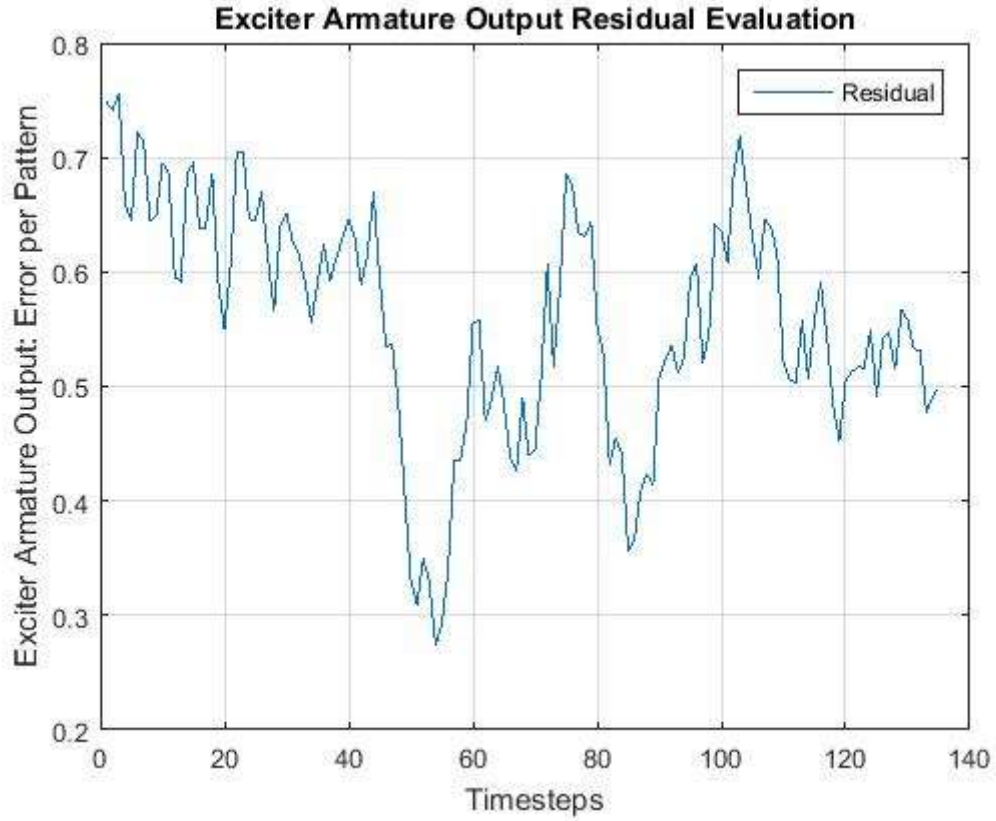


Figure G5: Sampling Number M = 10

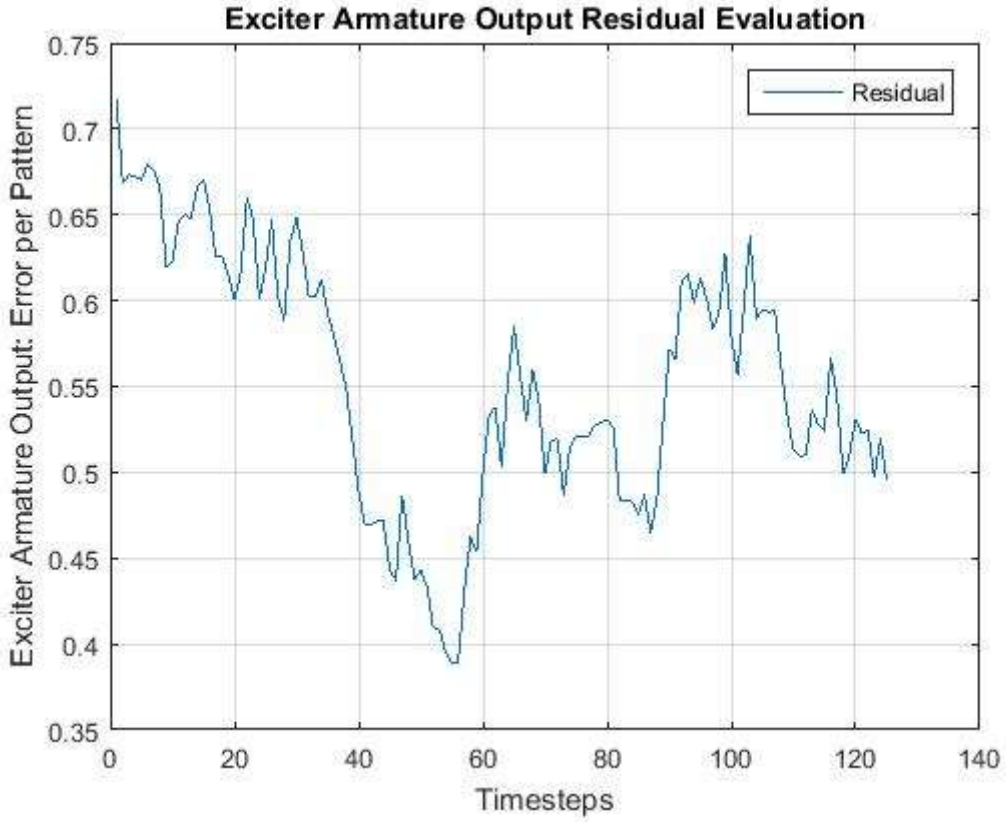


Figure G6: Sampling Number M = 20

Appendix H

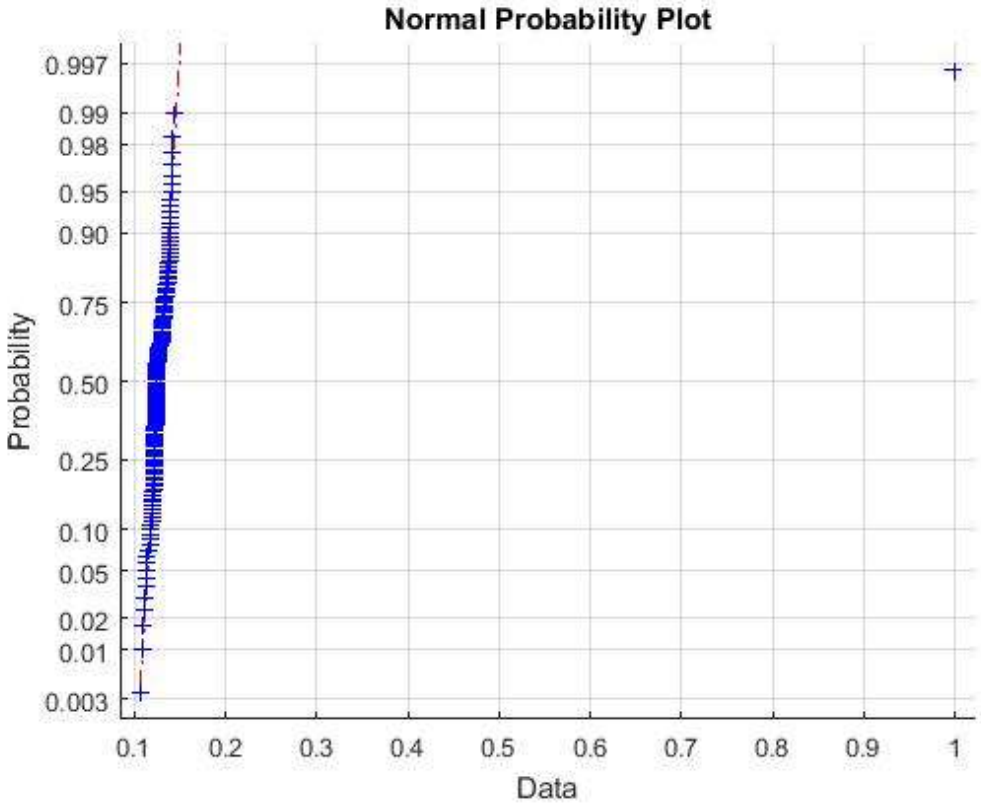


Figure H1: Exciter field current Probability Plot

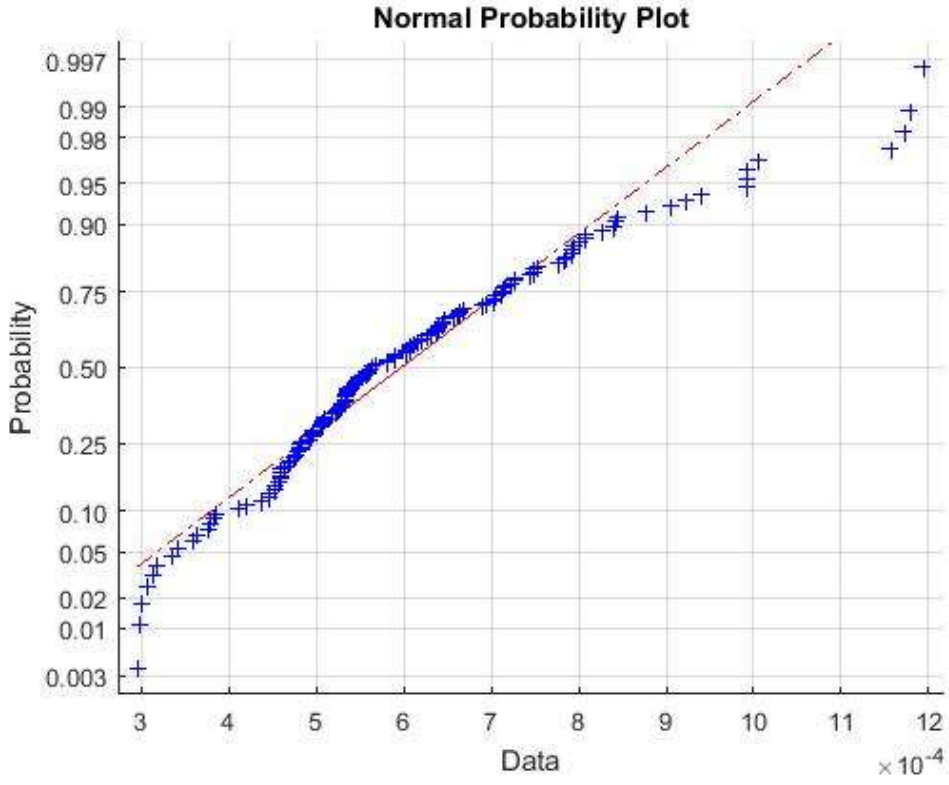


Figure H2: Engine Notch Command Probability Plot

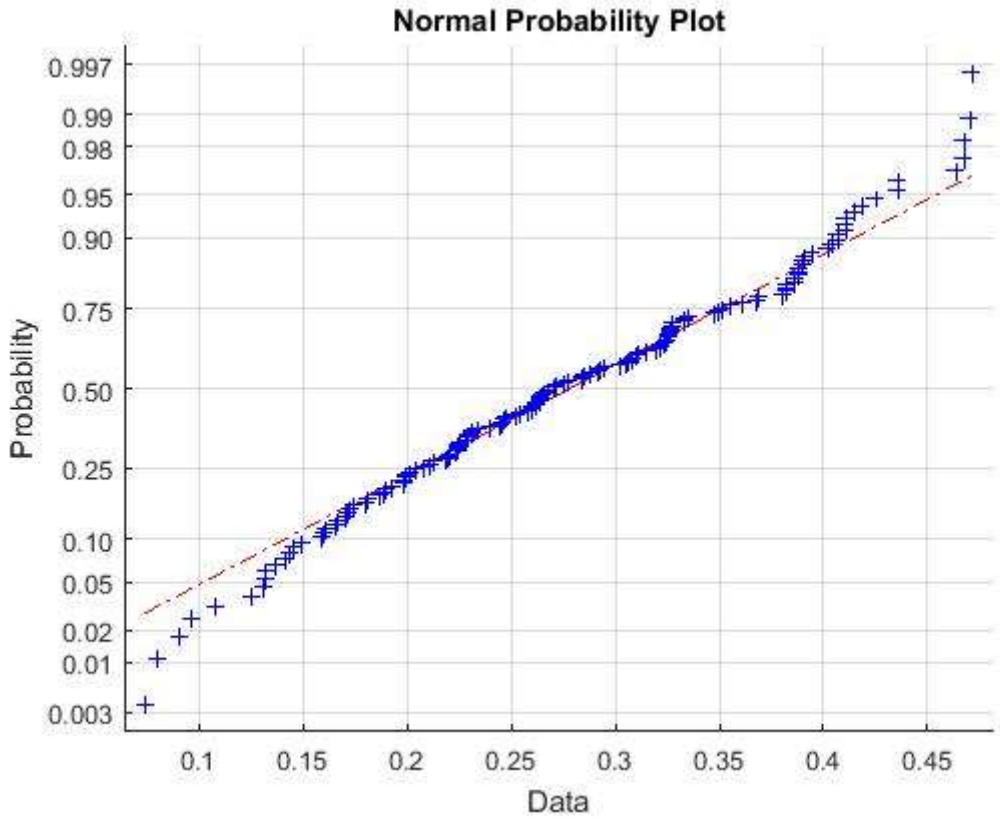


Figure H3: EXACT Probability Plot

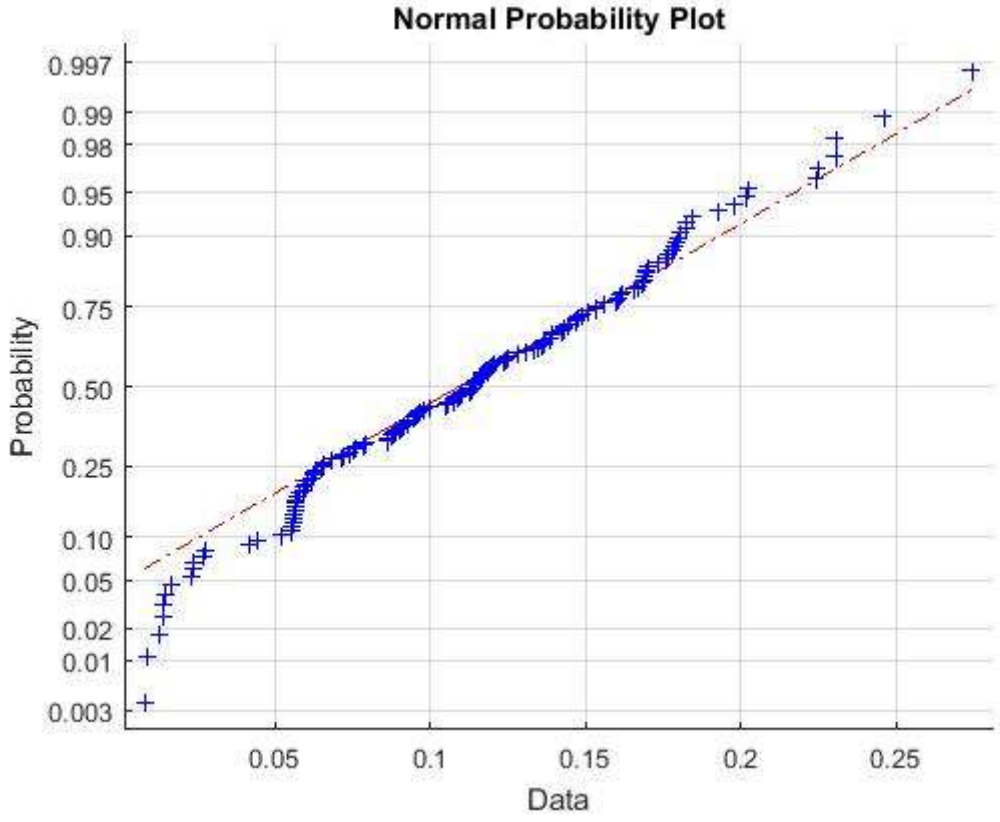


Figure H4: EXFM Probability Plot

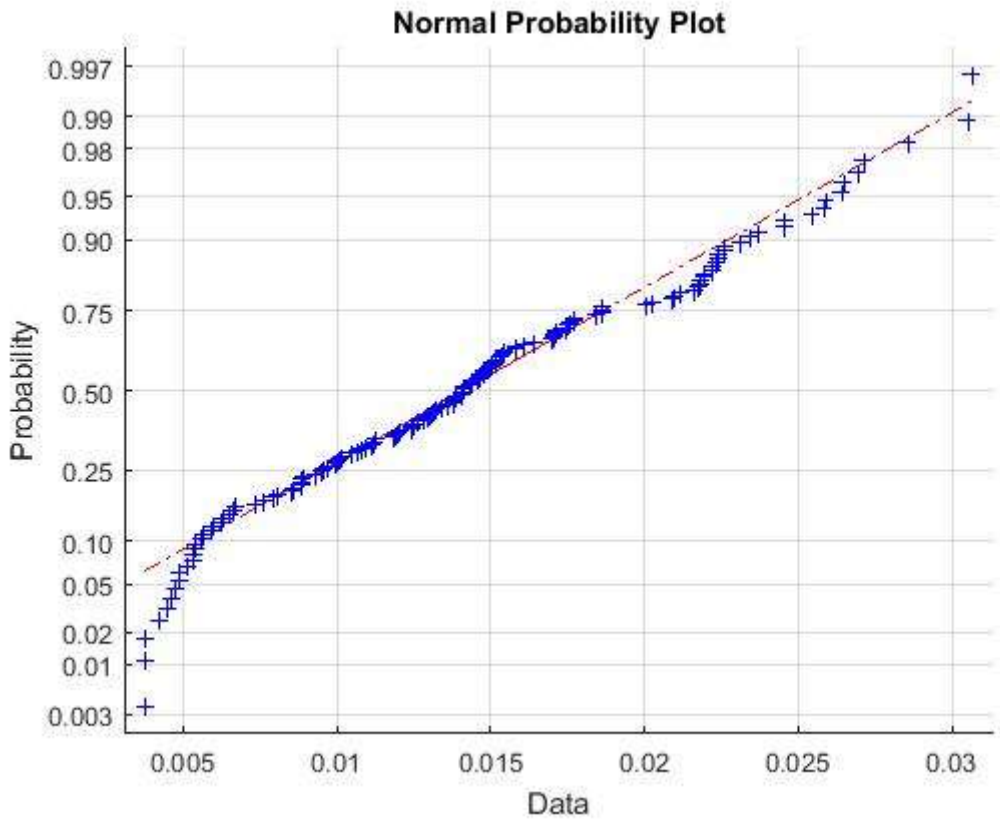


Figure H5: LCP Probability Plot

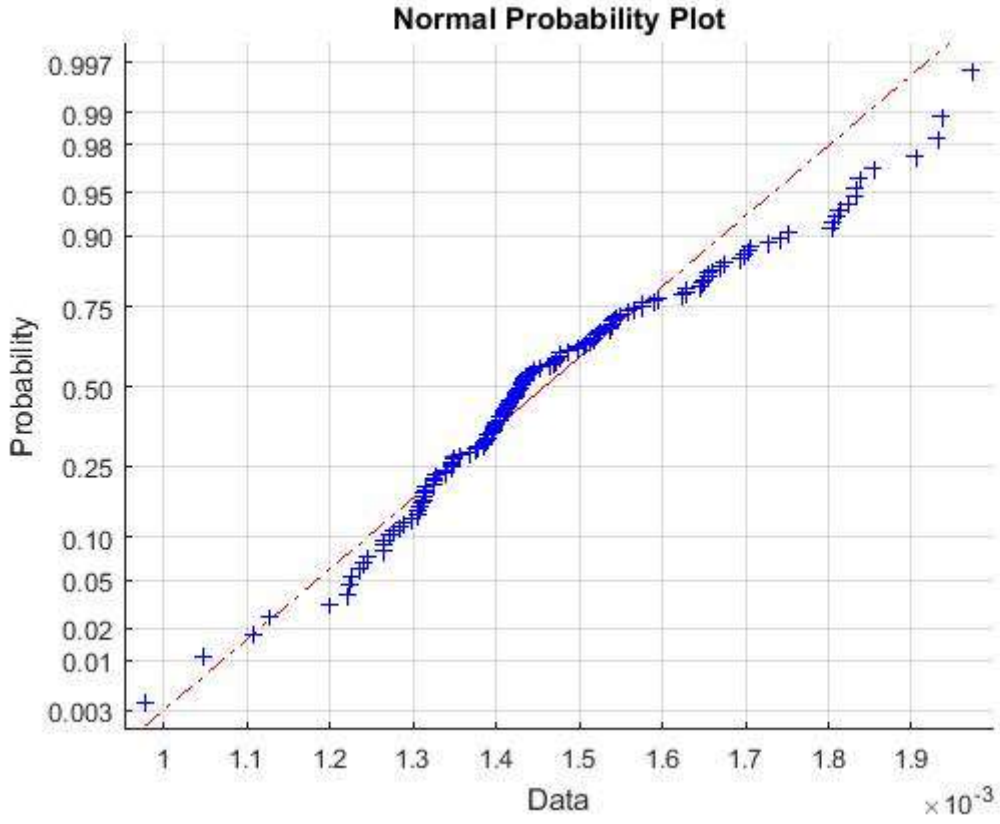


Figure H6: Power Notch Command Probability Plot

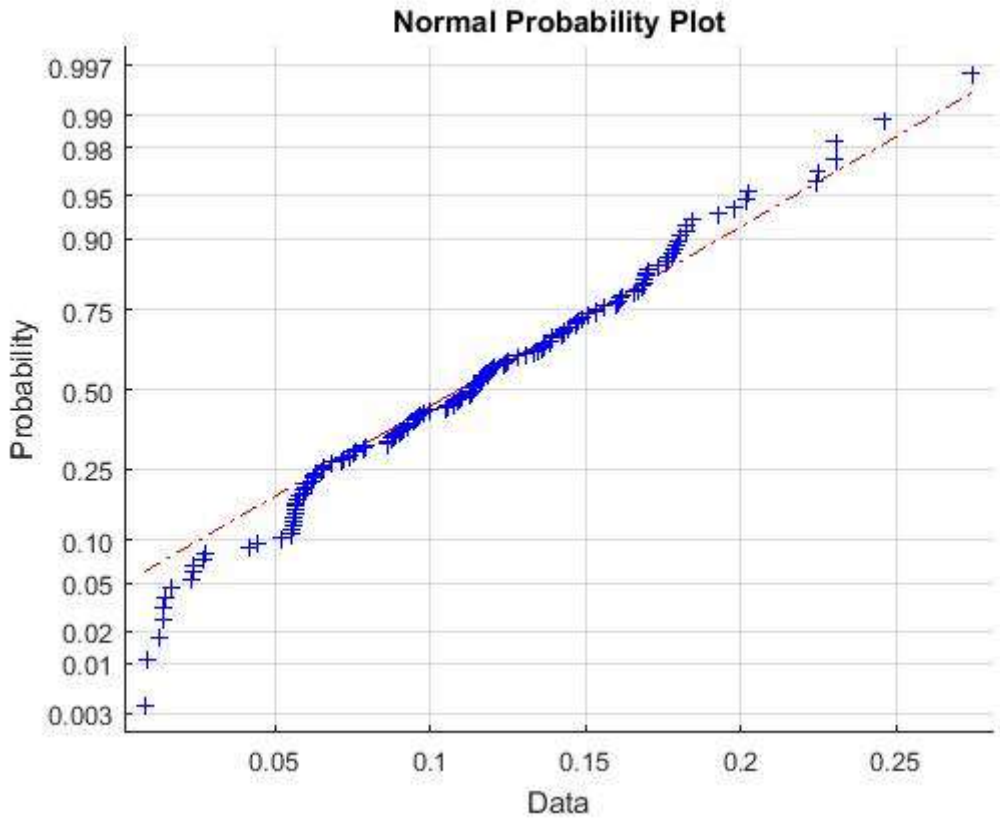


Figure H7: SCM8 Probability Plot

Appendix I

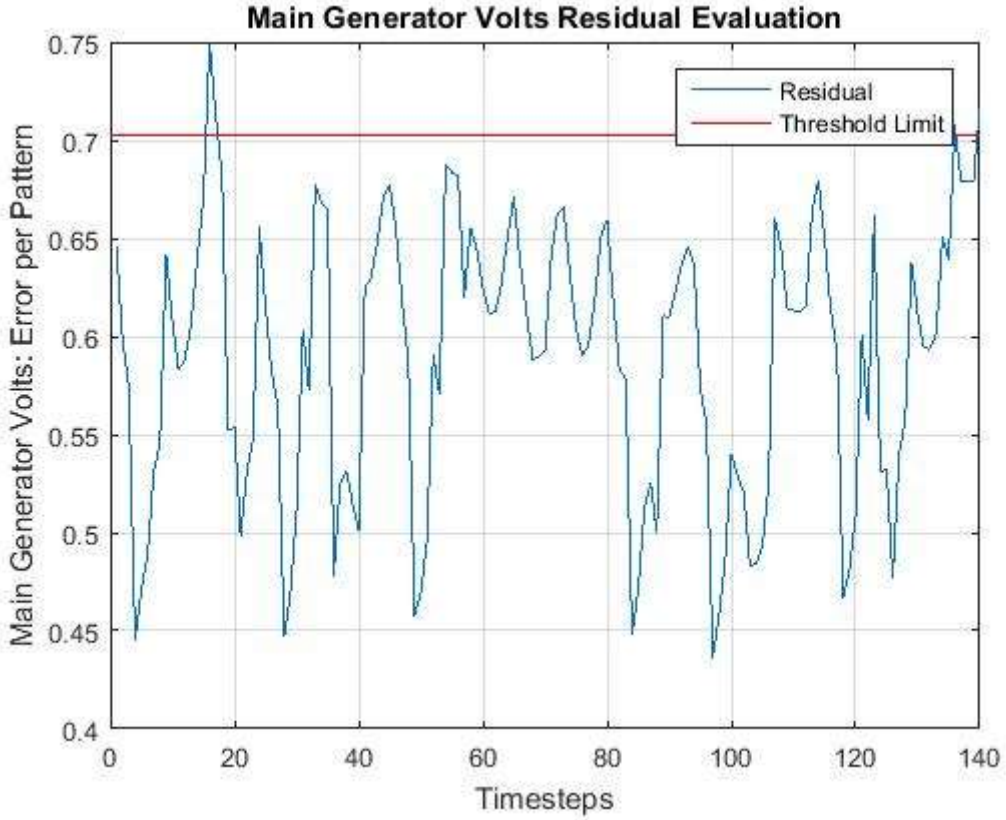


Figure I1: SCM8 Threshold and Residual Plot

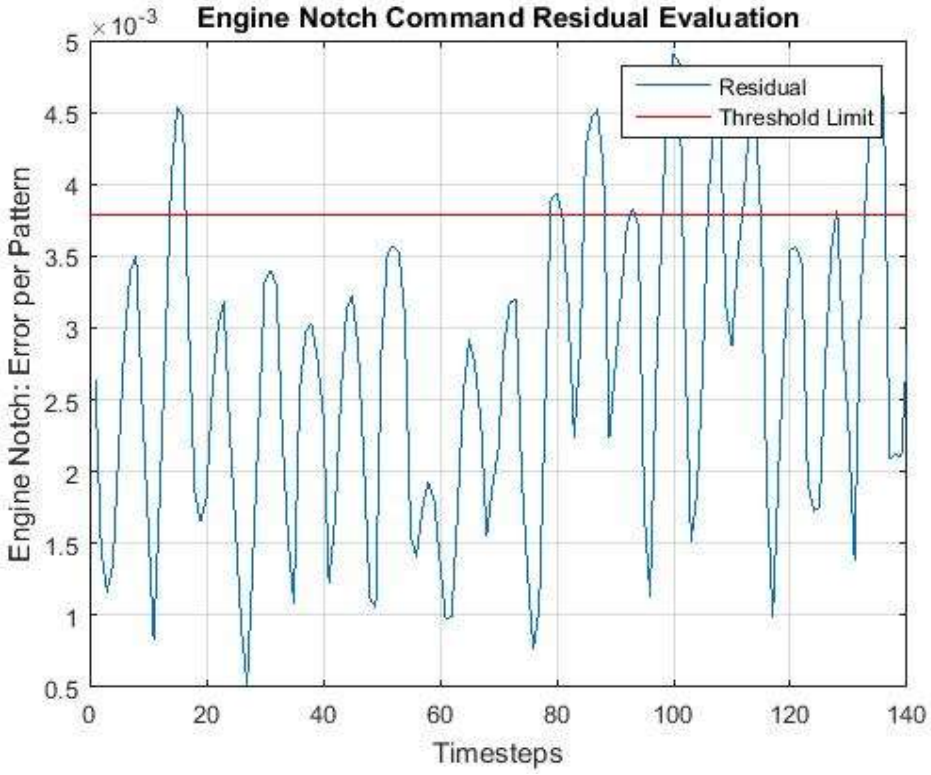


Figure I2: Engine Notch Command Threshold and Residual Plot

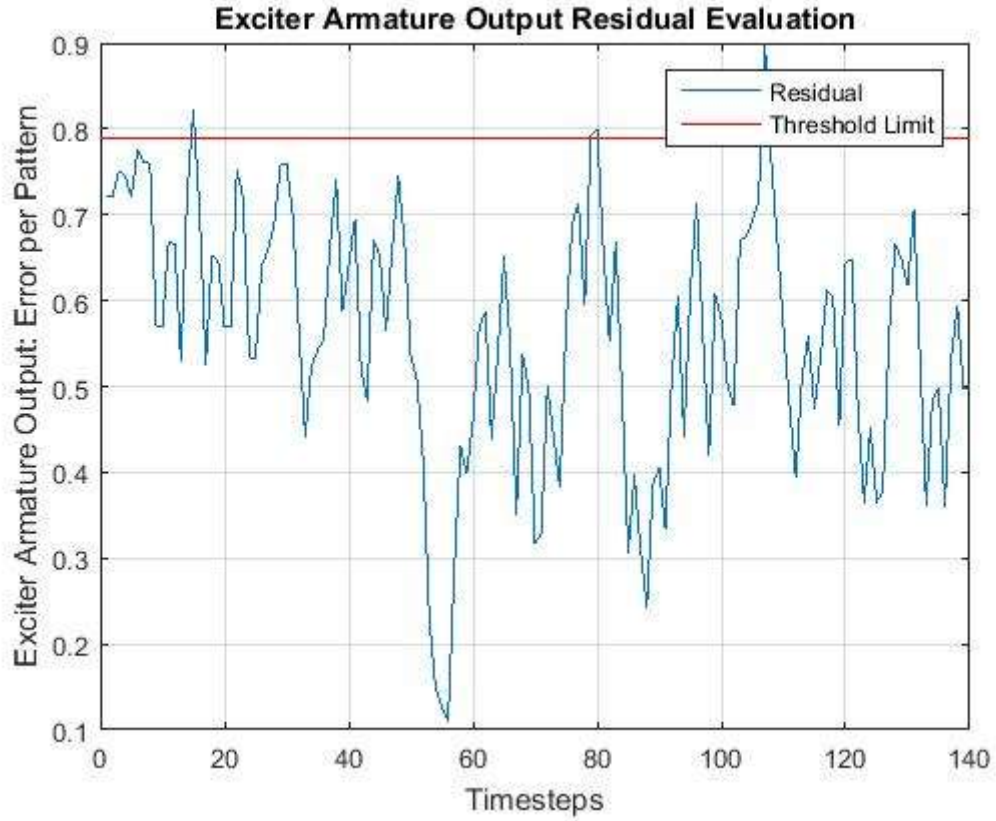


Figure I3: EXACT Threshold and Residual Plot

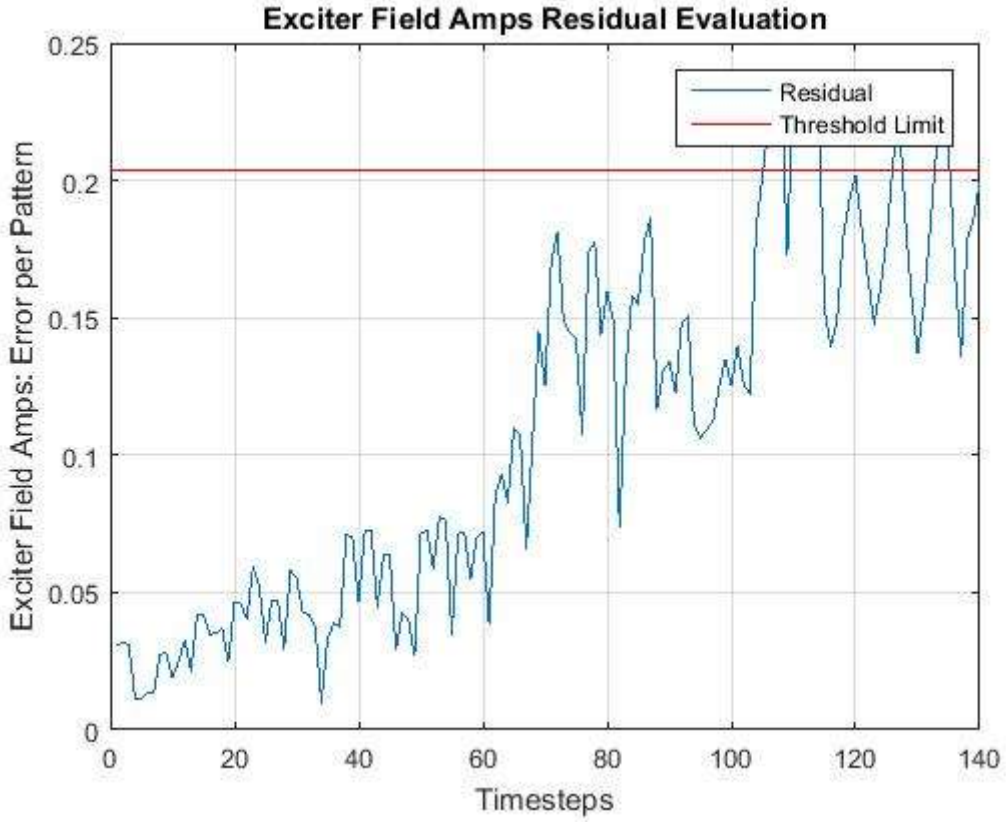


Figure I4: EXFM Threshold and Residual Plot

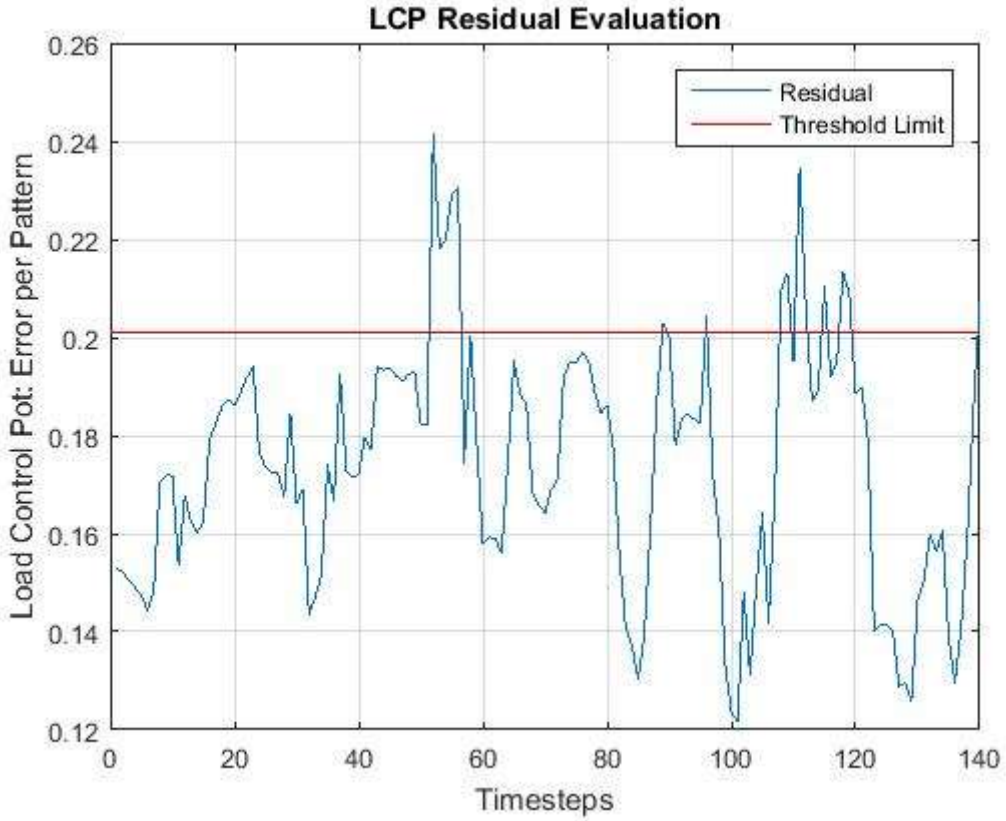


Figure I5: LCP Threshold and Residual Plot

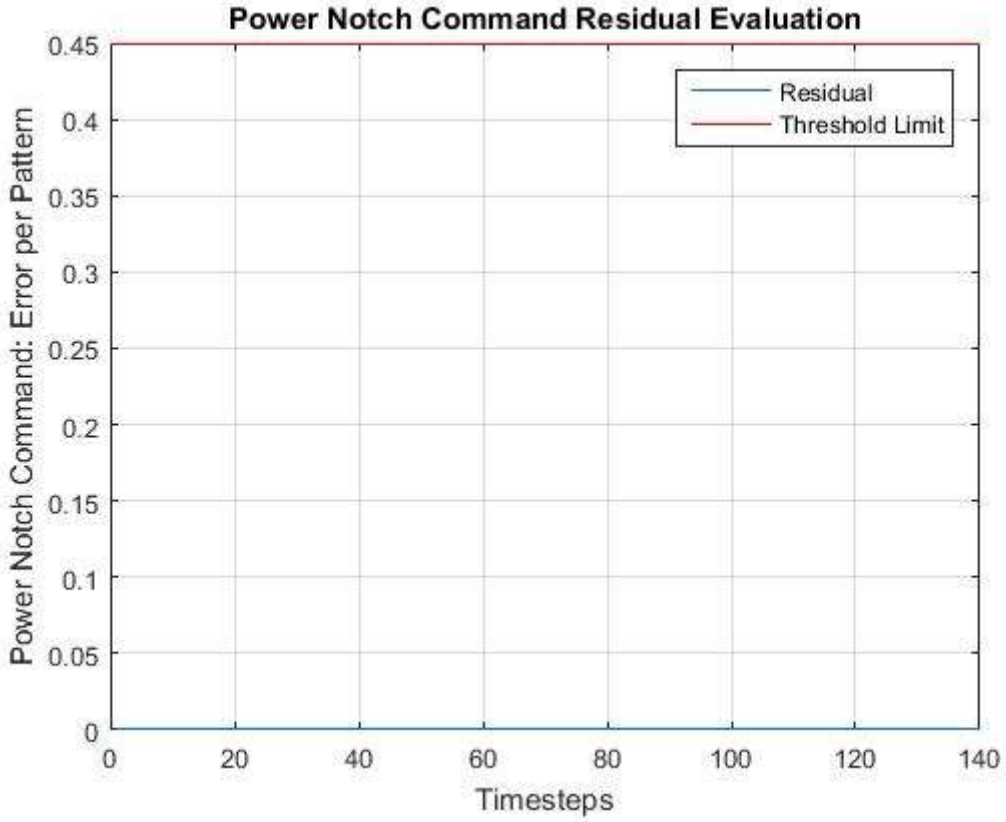


Figure I6: Power Notch Command Threshold and Residual Plot

Appendix J

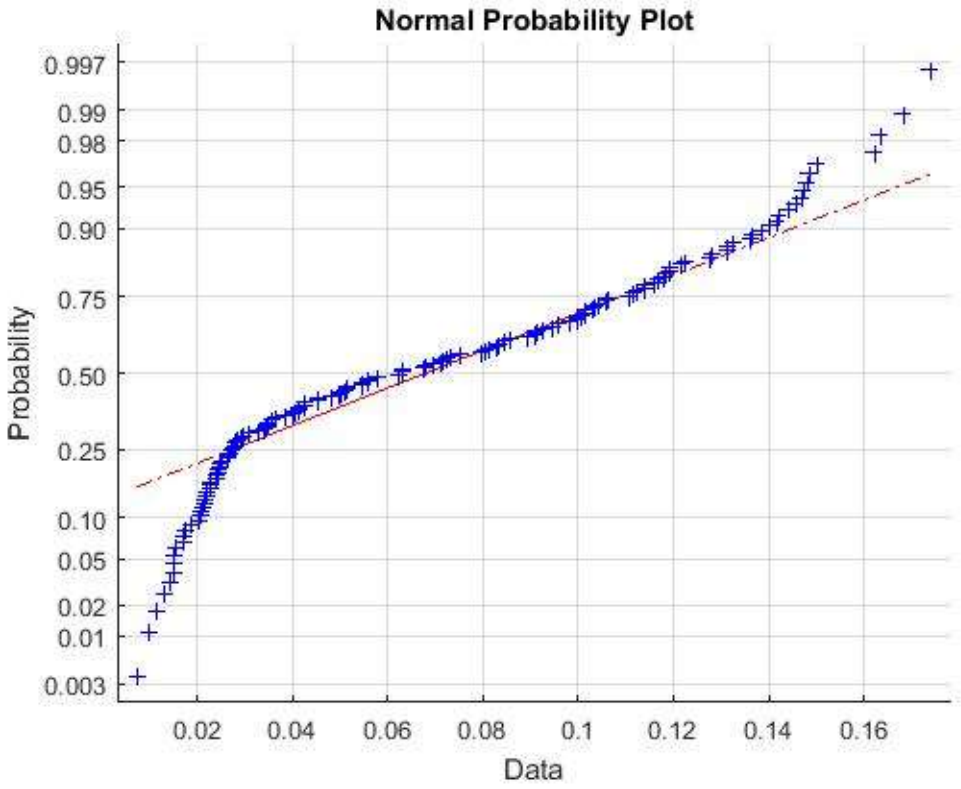


Figure J1: EXFM Probability Plot

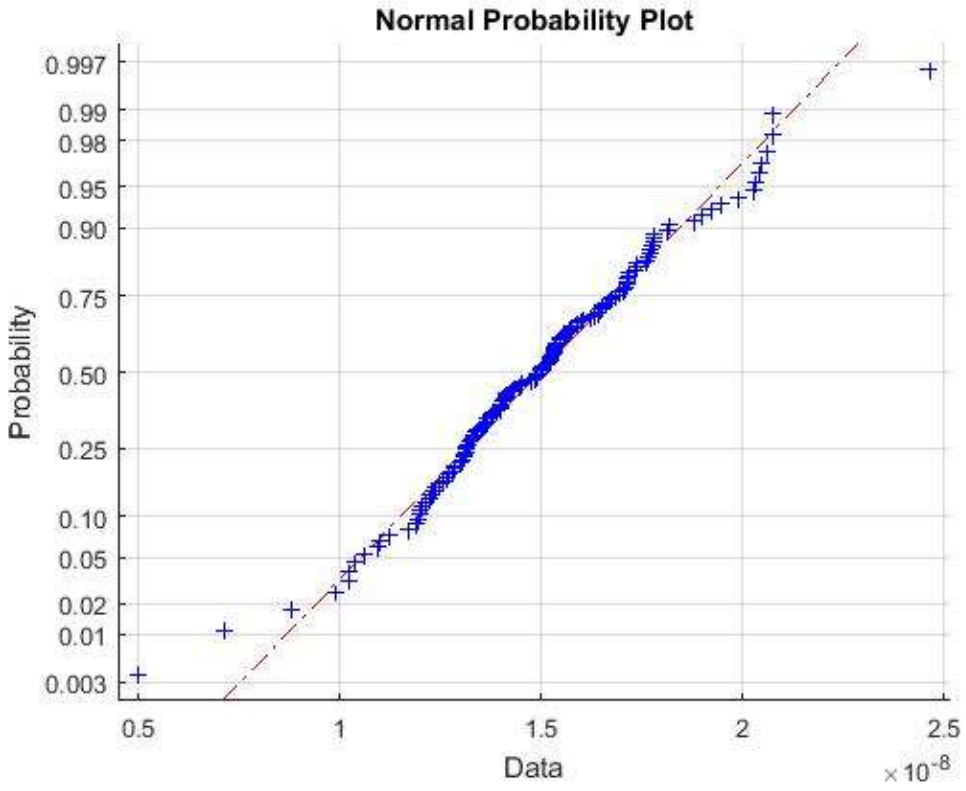


Figure J2: Engine Notch Command Probability Plot

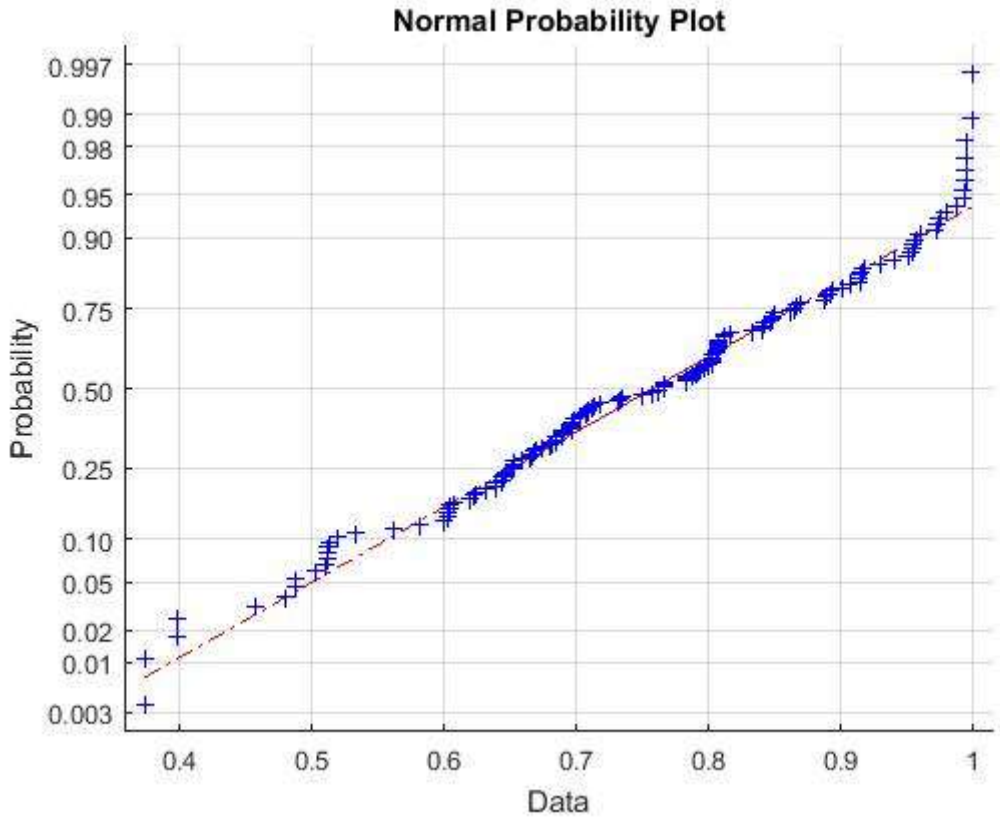


Figure J3: EXACT Probability Plot

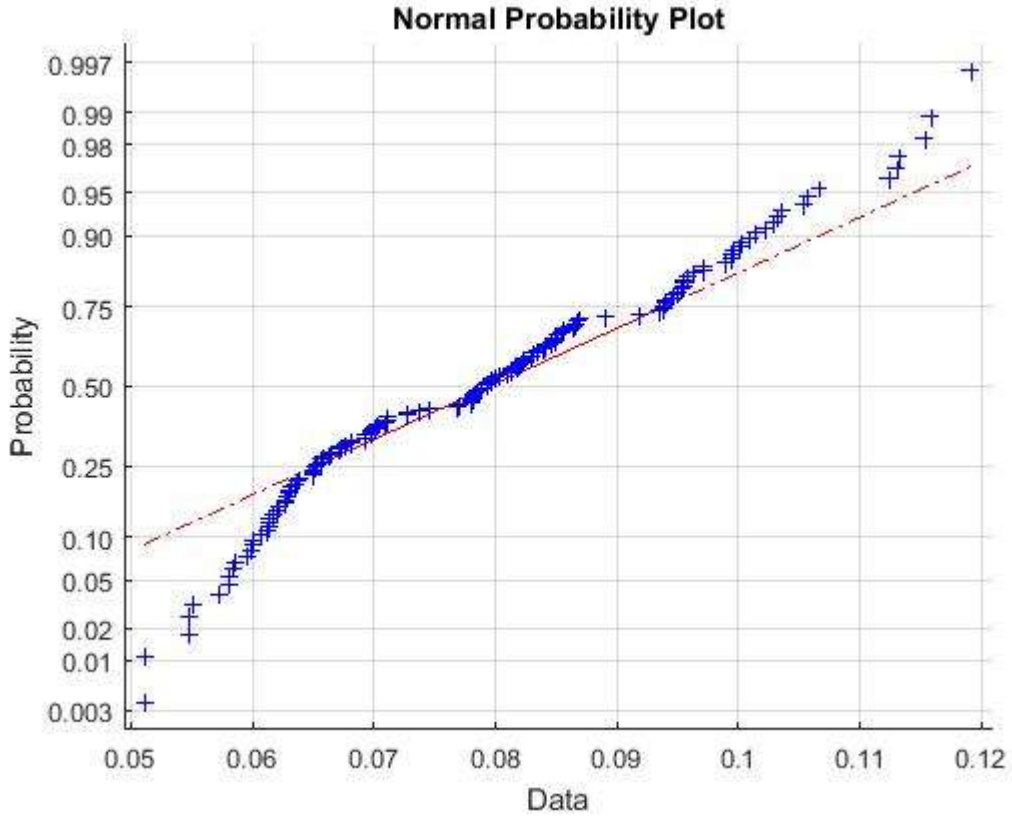


Figure J4: LCP Probability Plot

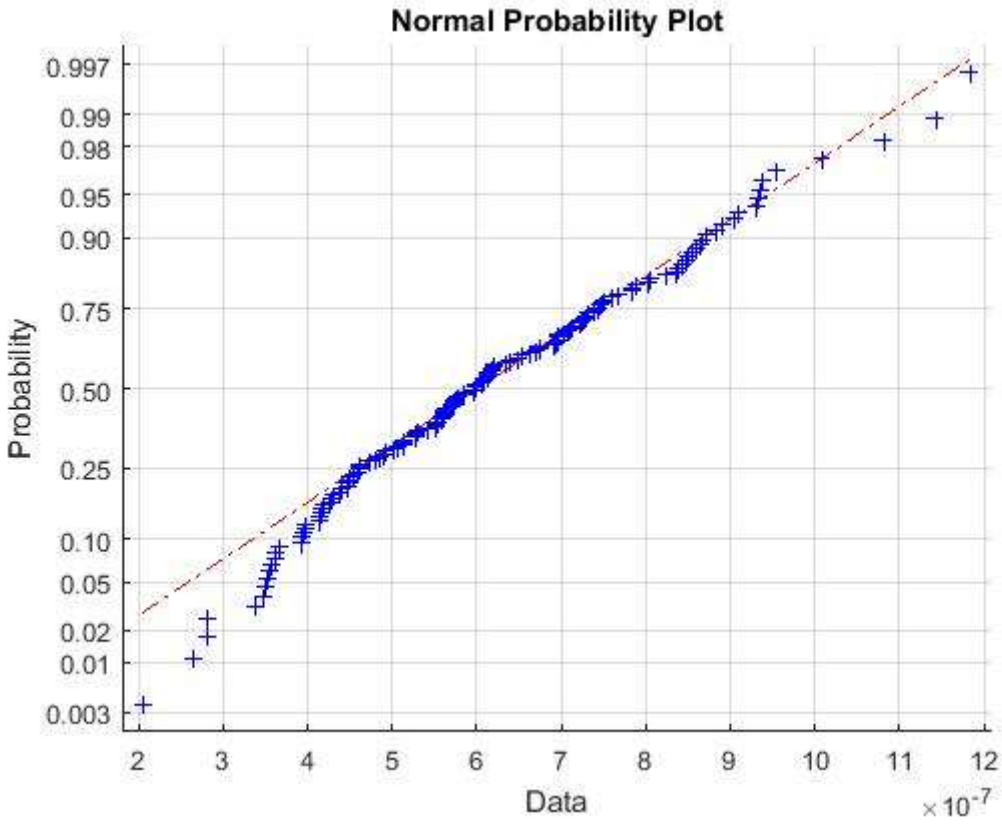


Figure J5: Power Notch Command Probability Plot

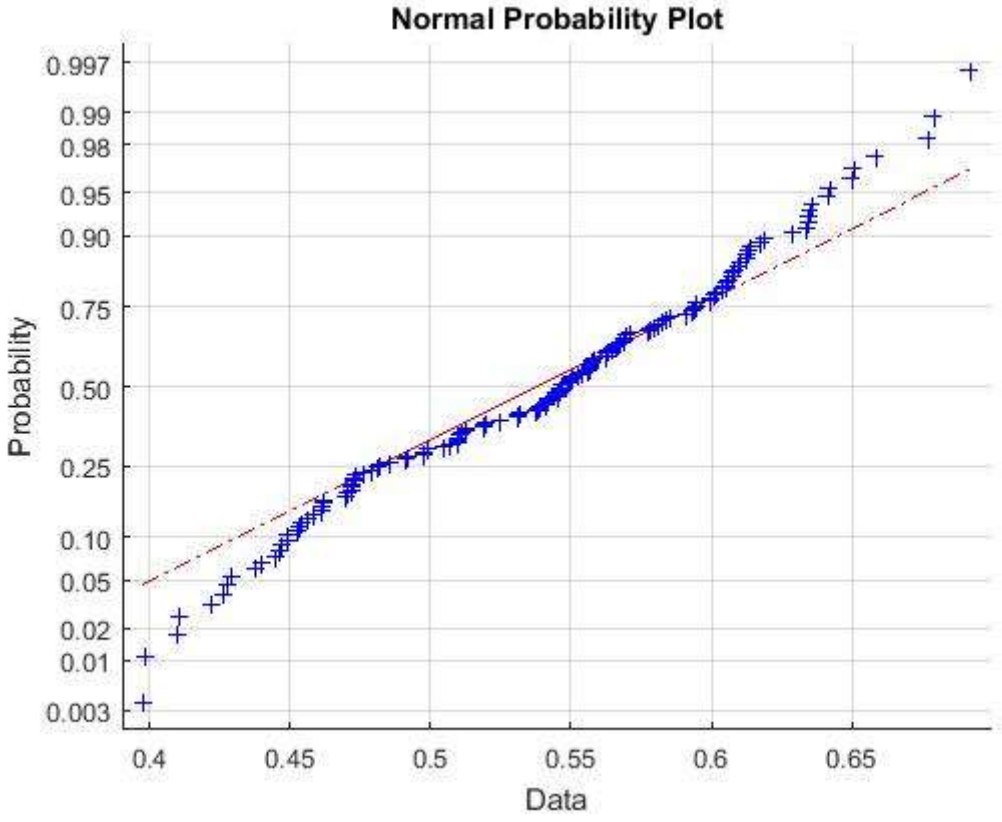


Figure J6: SCM8 Probability Plot

Appendix K

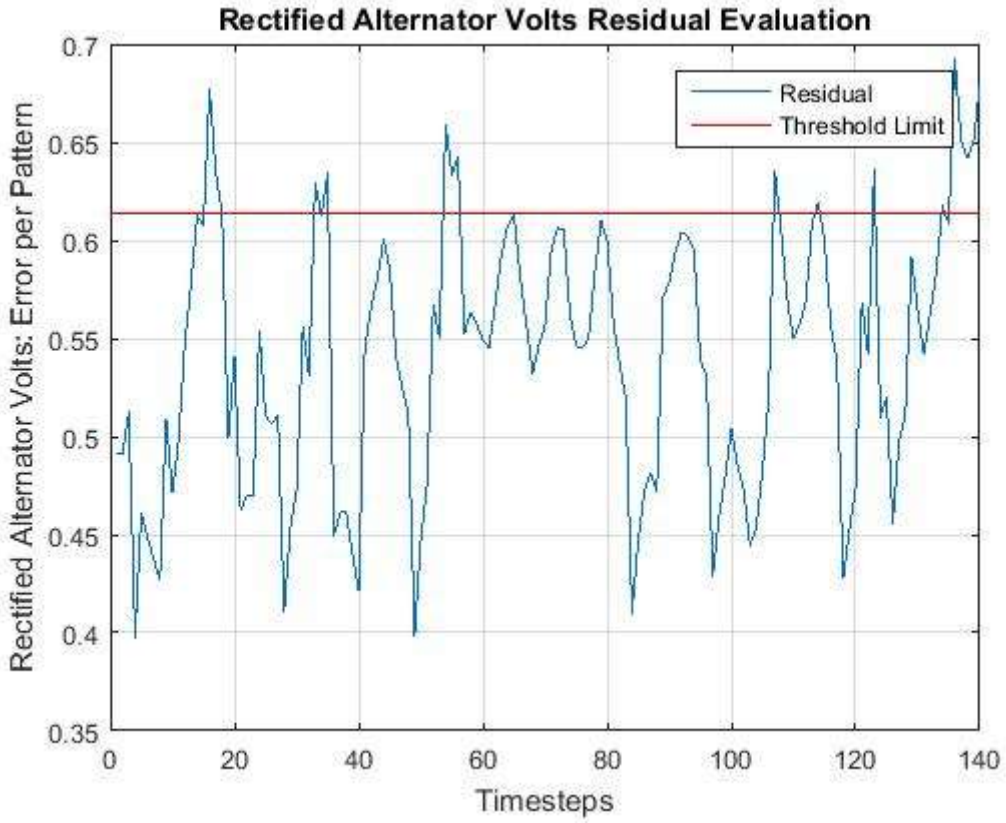


Figure K1: SCM8 Threshold and Residual Plot

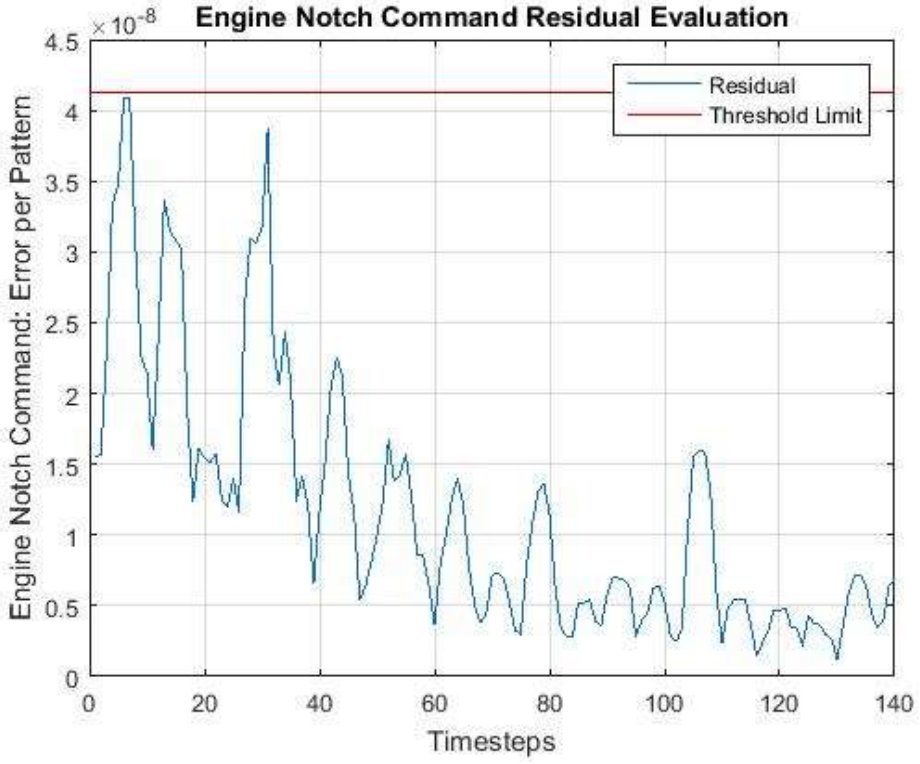


Figure K2: Engine Notch Command Threshold and Residual Plot

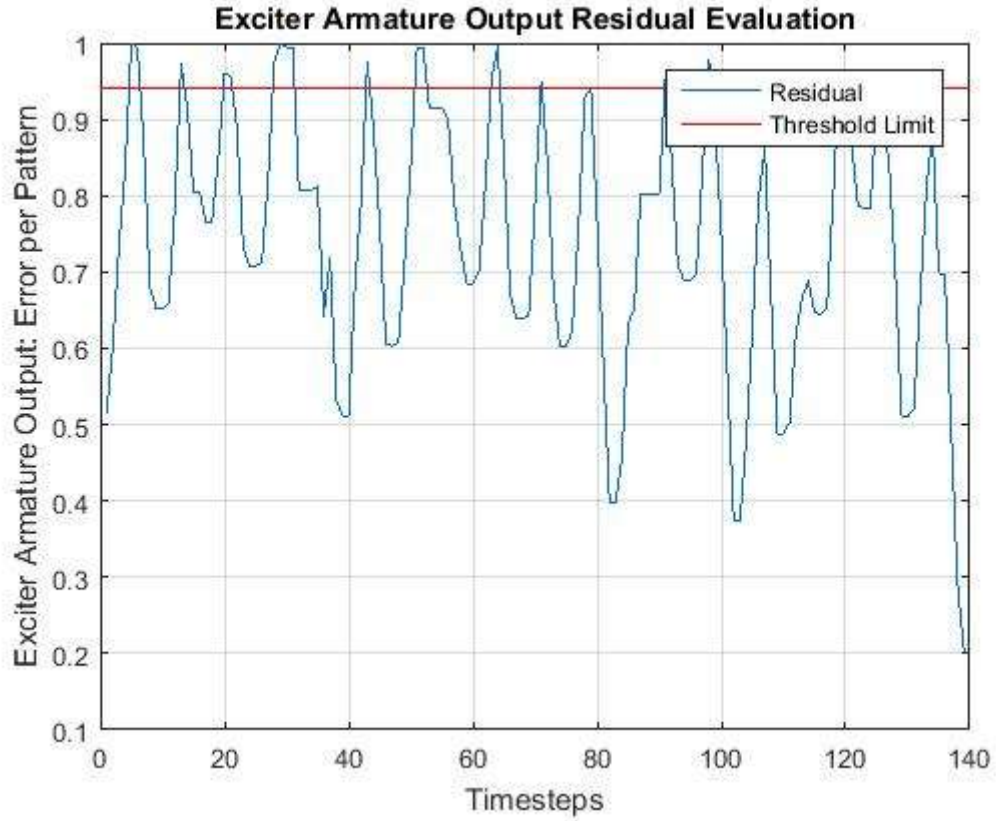


Figure K3: EXACT Threshold and Residual Plot

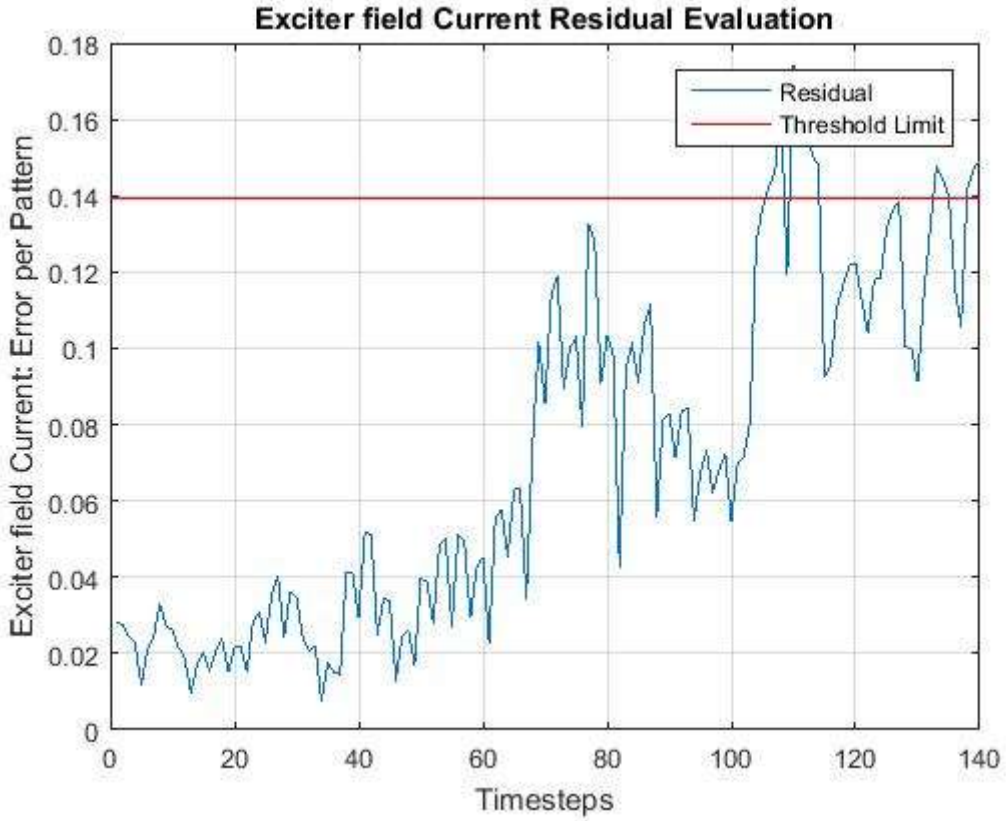


Figure K4: EXFM Threshold and Residual Plot

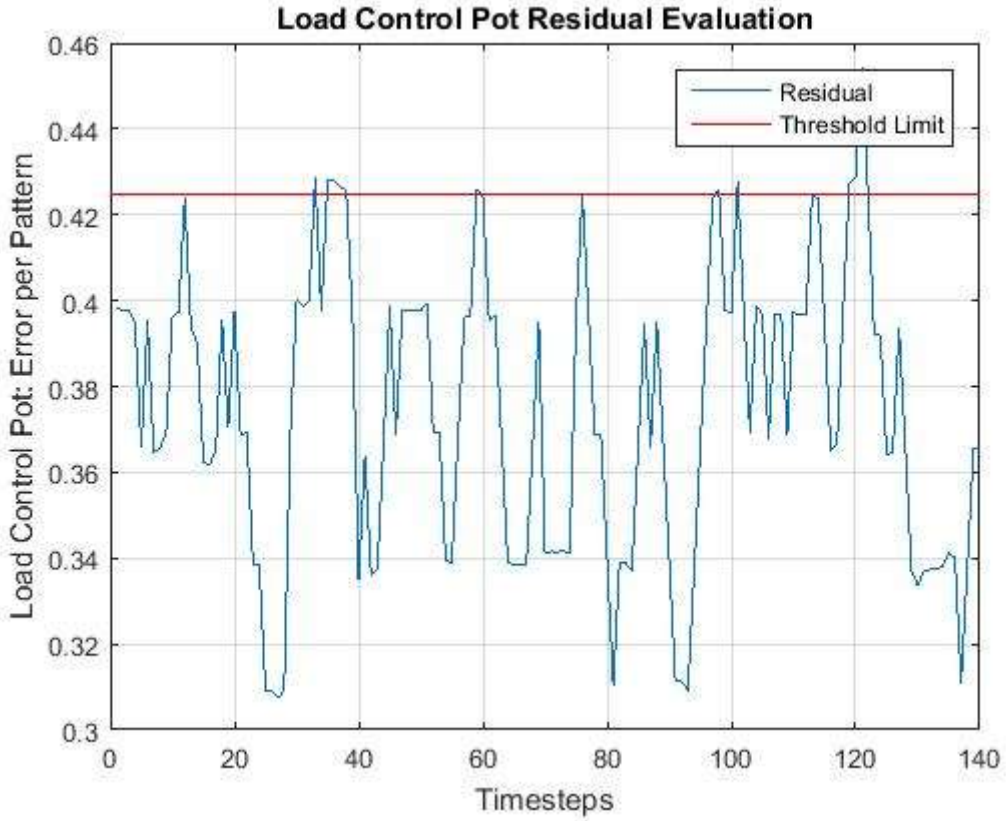


Figure K5: LCP Threshold and Residual Plot

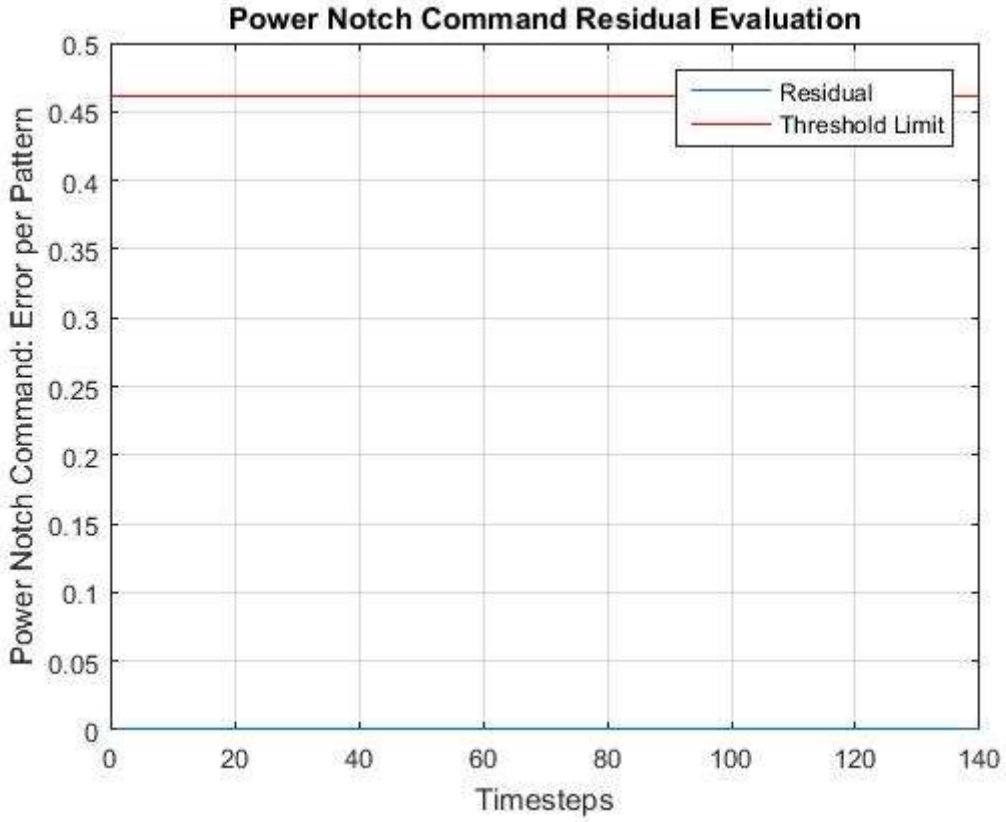


Figure K6: Power Notch Command Threshold and Residual Plot

Appendix L

L1.1 SCM8 Test 1 Results

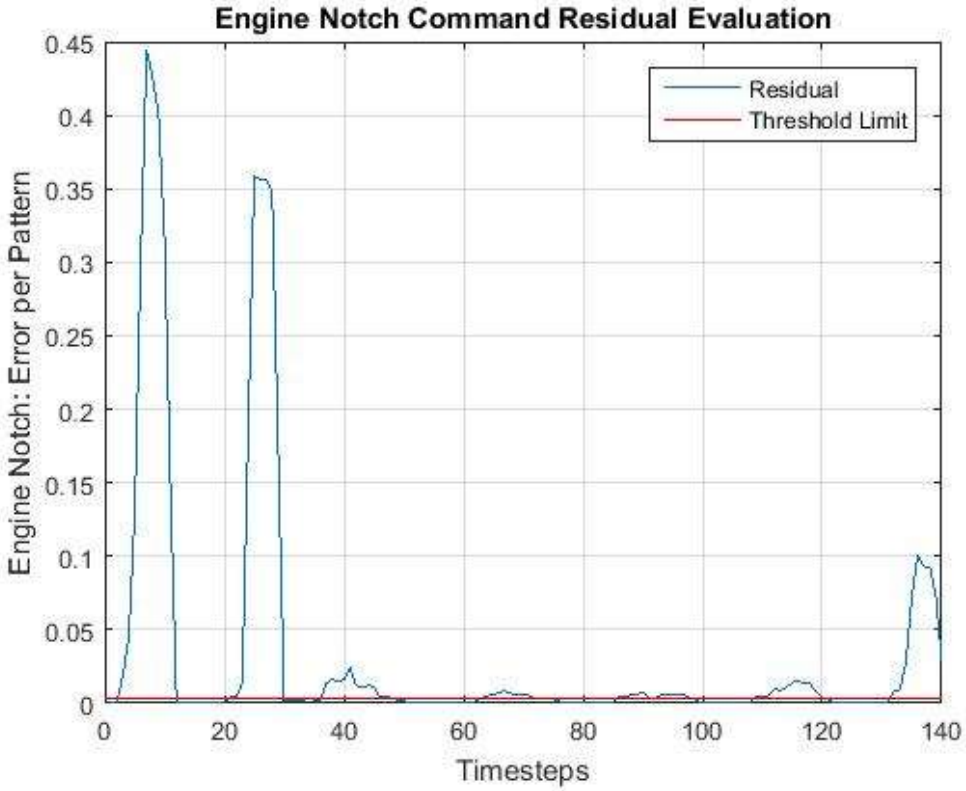


Figure L1.1: SCM8 Test 1: Engine Notch Command Test Result

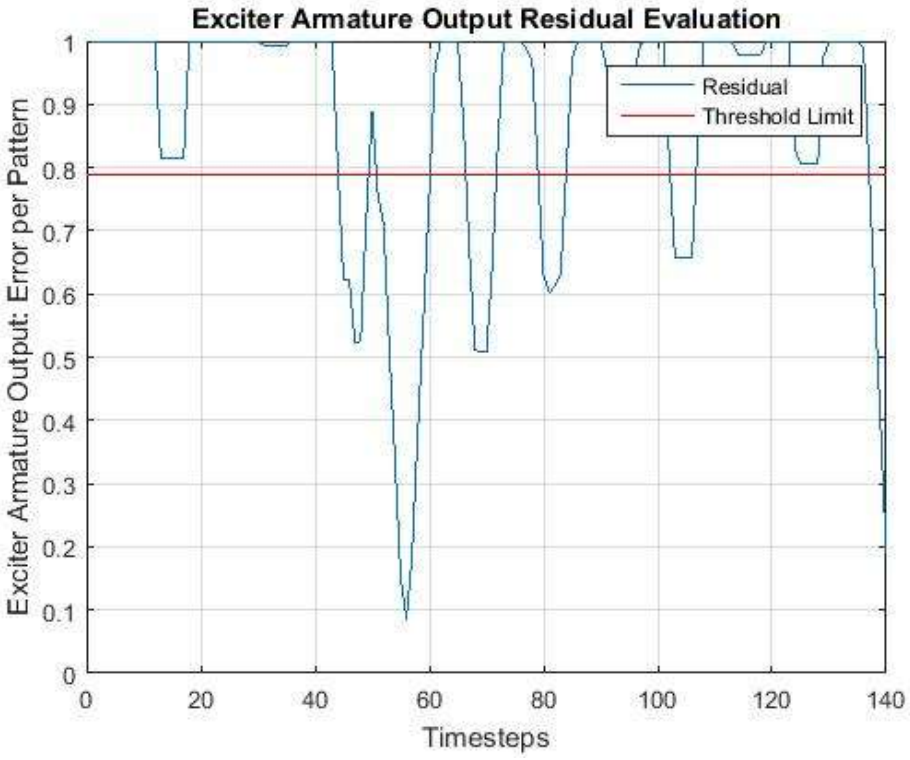


Figure L1.1.2: SCM8 Test 1: EXACT Test Result

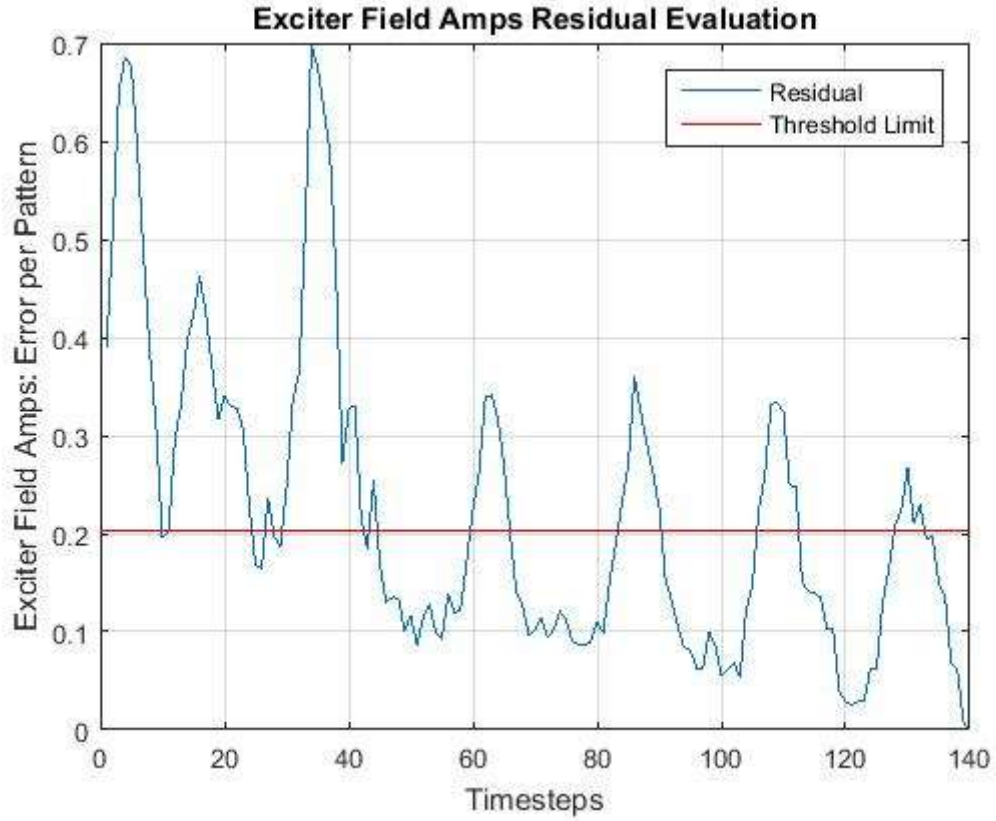


Figure L1.1.3: SCM8 Test 1: EXFM Test Result

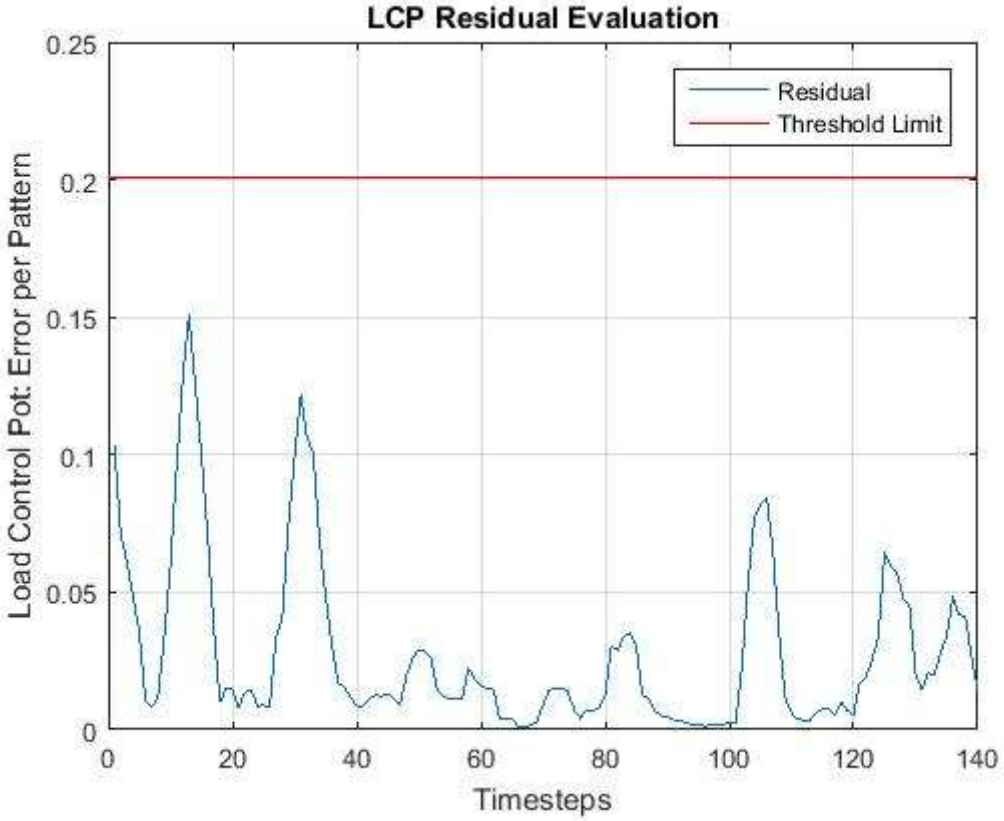


Figure L1.1.4: SCM8 Test 1: LCP Test Result

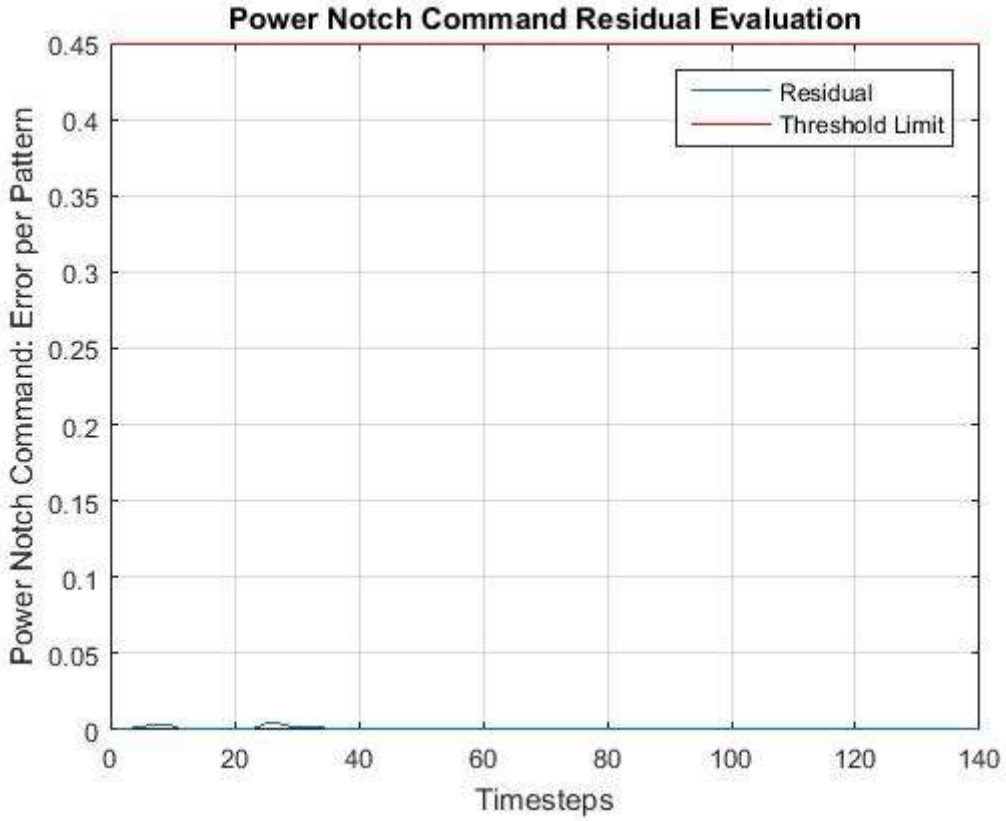


Figure L1.1.5: SCM8 Test 1: Power Notch Command Test Result

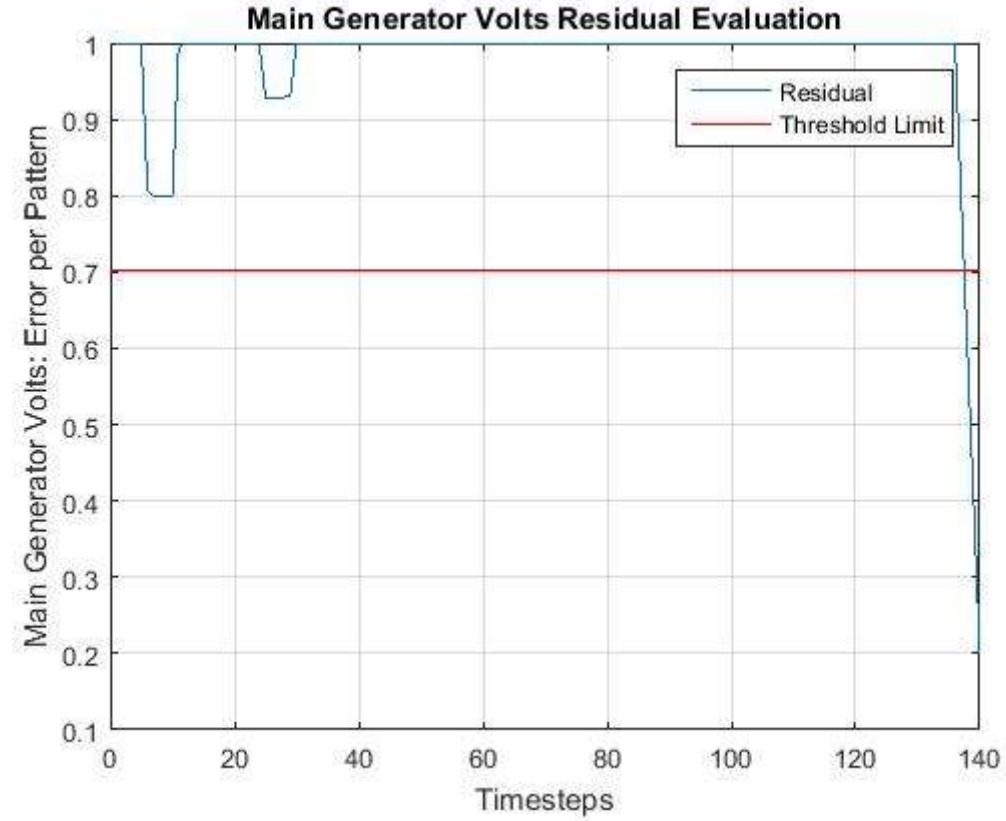


Figure L1.1.6: SCM8 Test 1: SCM8 Test Result

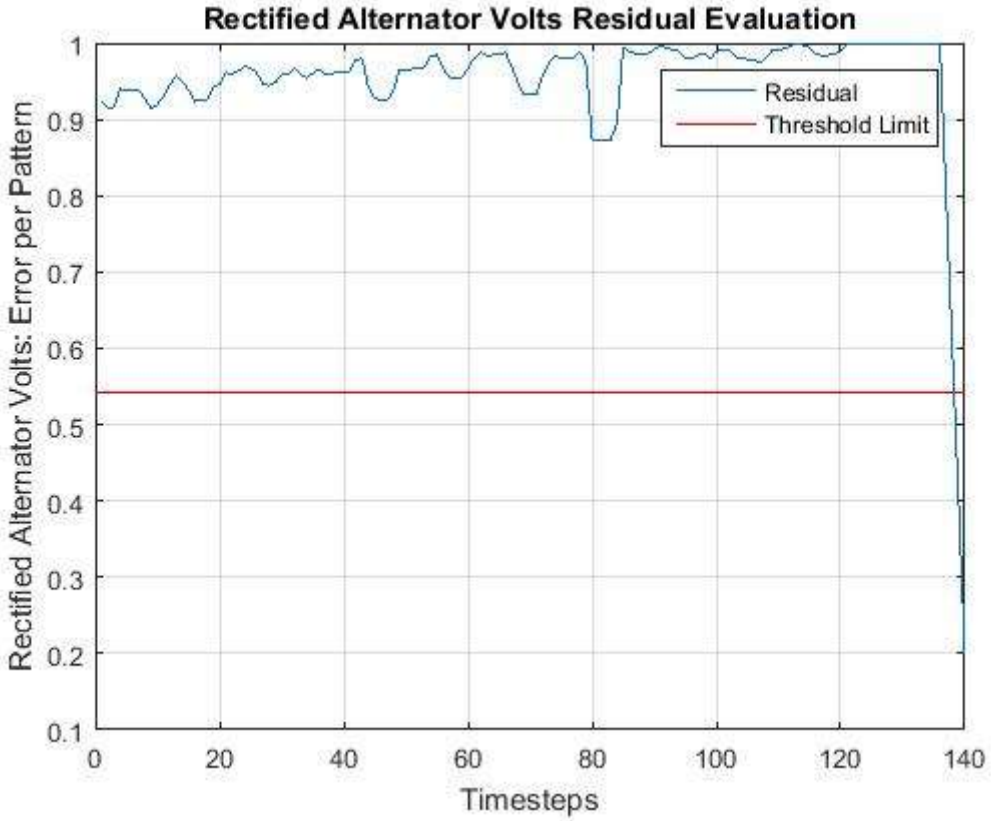


Figure L1.1.7: SCM8 Test 1: SCM8 Sensor Validation Test Result

L1.2 SCM8 Test 2 Results

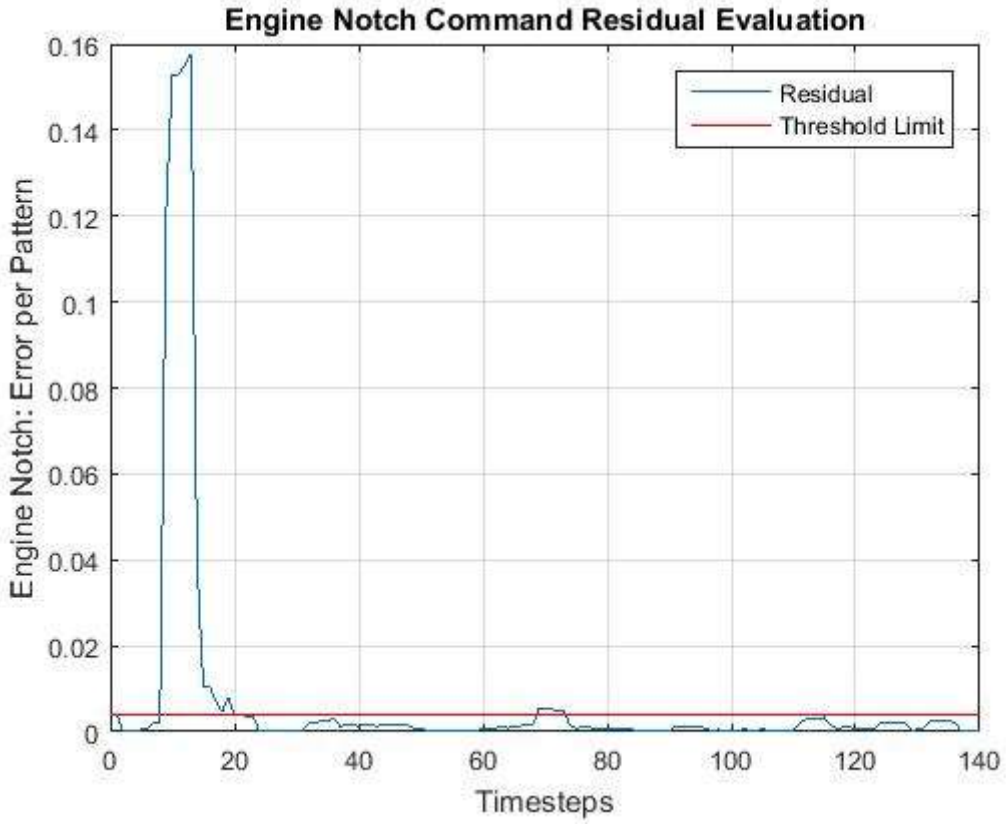


Figure L1.2.1: SCM8 Test 2: Engine Notch Command Test Result

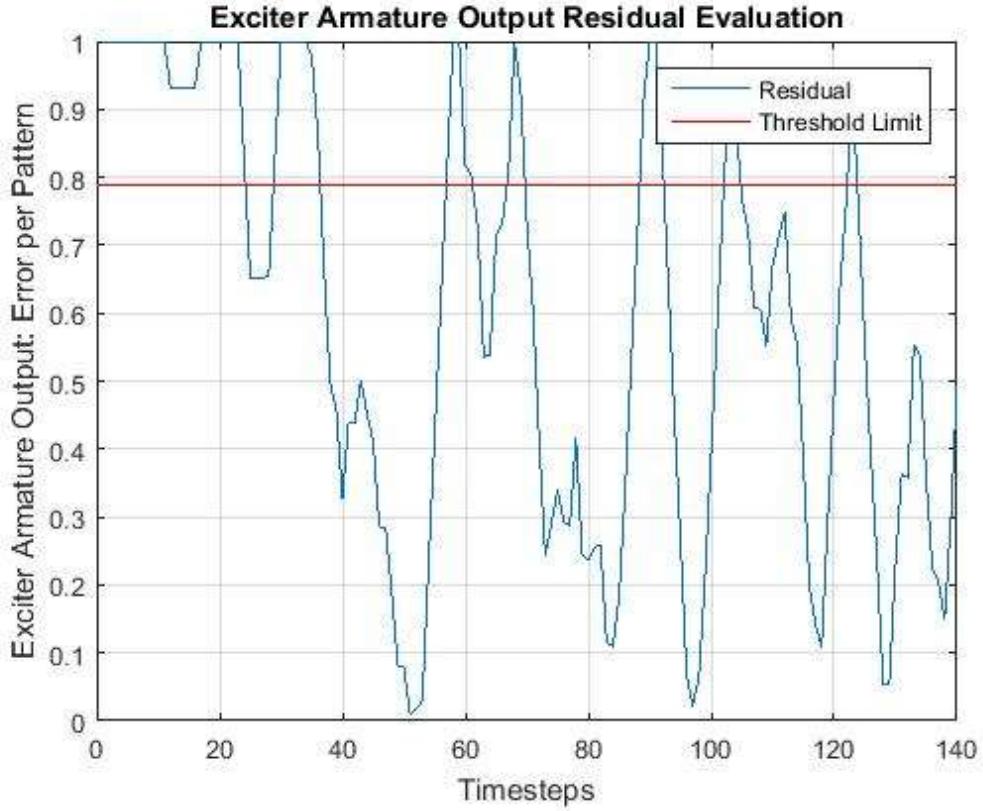


Figure L1.2.2: SCM8 Test 2: EXACT Test Result

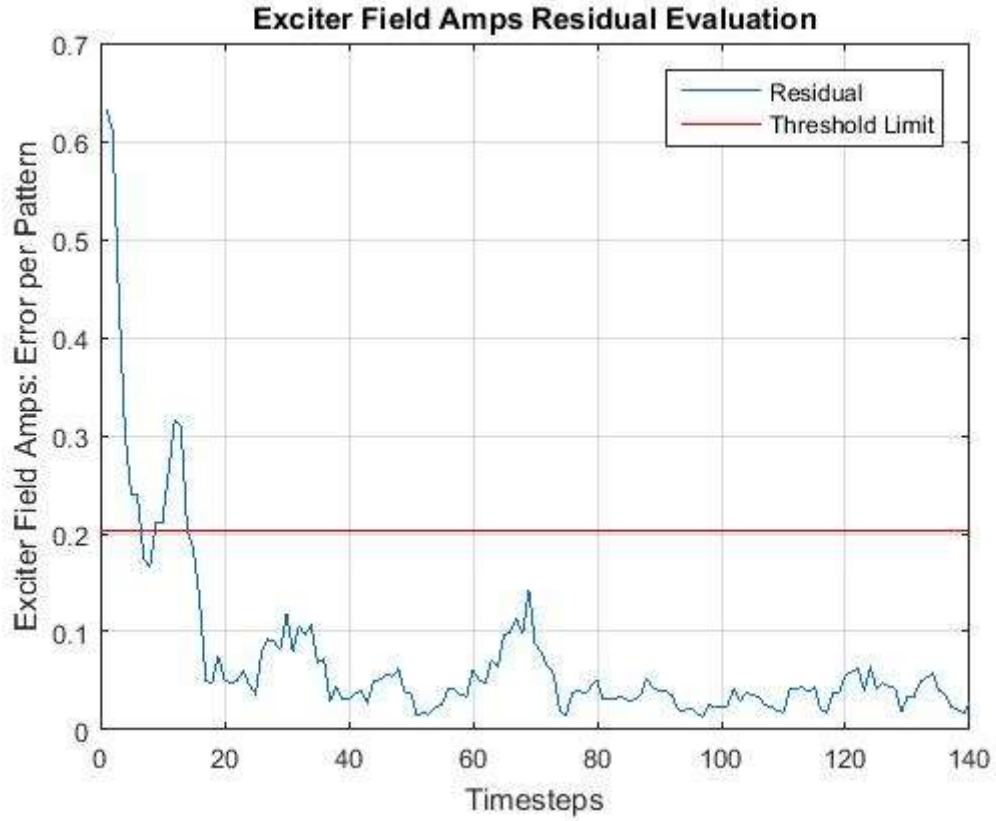


Figure L1.2.3: SCM8 Test 2: EXFM Test Result

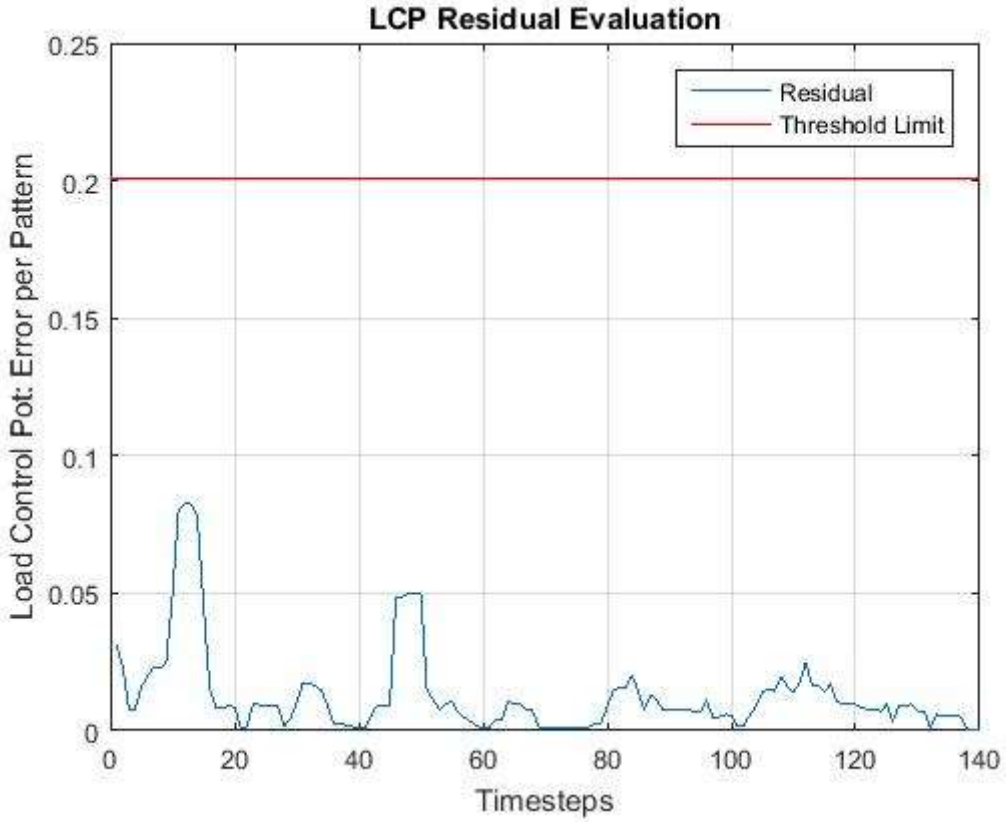


Figure L1.2.4: SCM8 Test 2: LCP Test Result

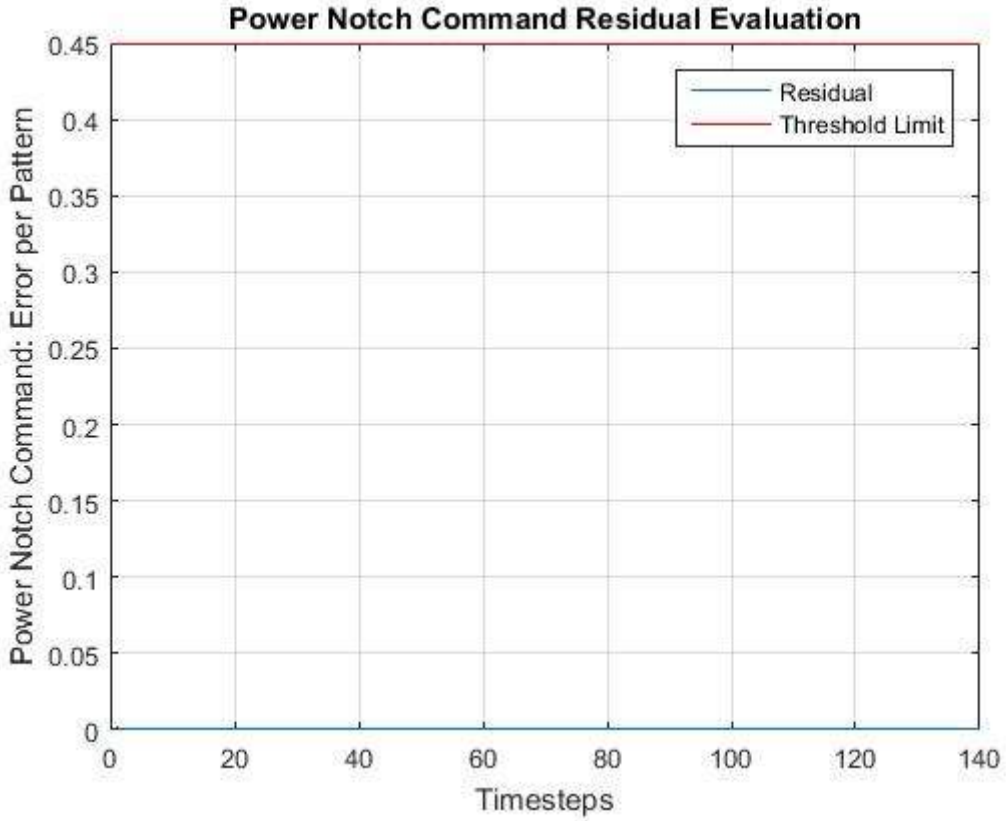


Figure L1.2.5: SCM8 Test 2: Power Notch Command Test Result

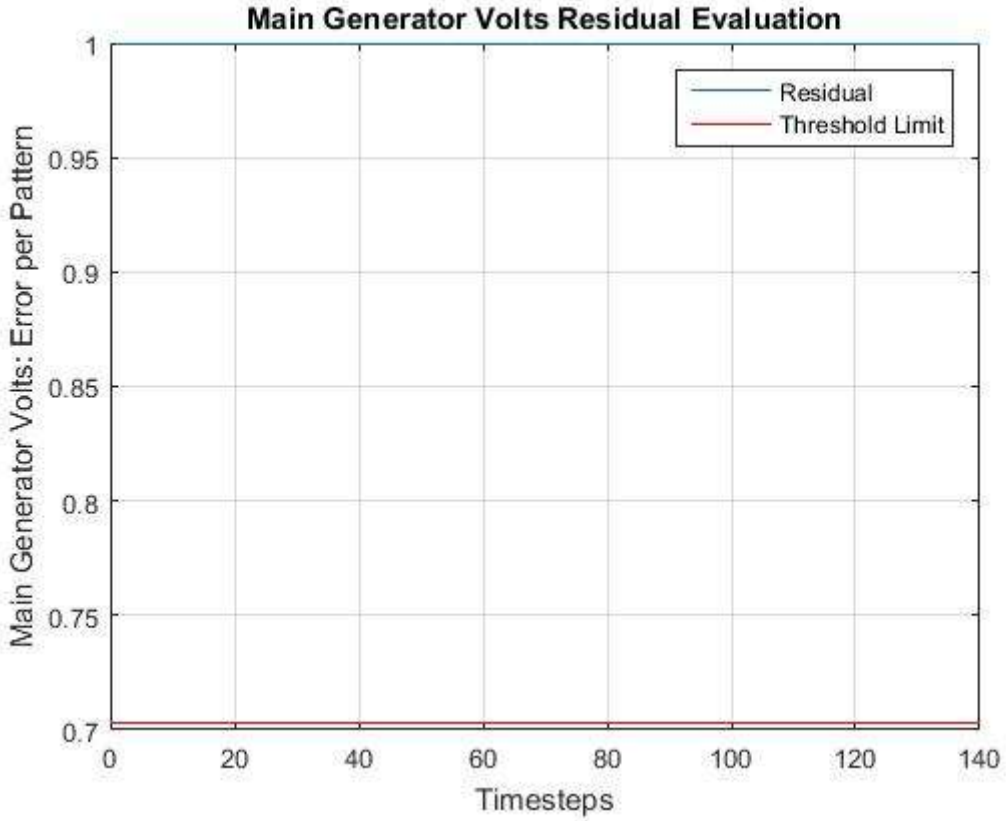


Figure L1.2.6: SCM8 Test 2: SCM8 Test Result

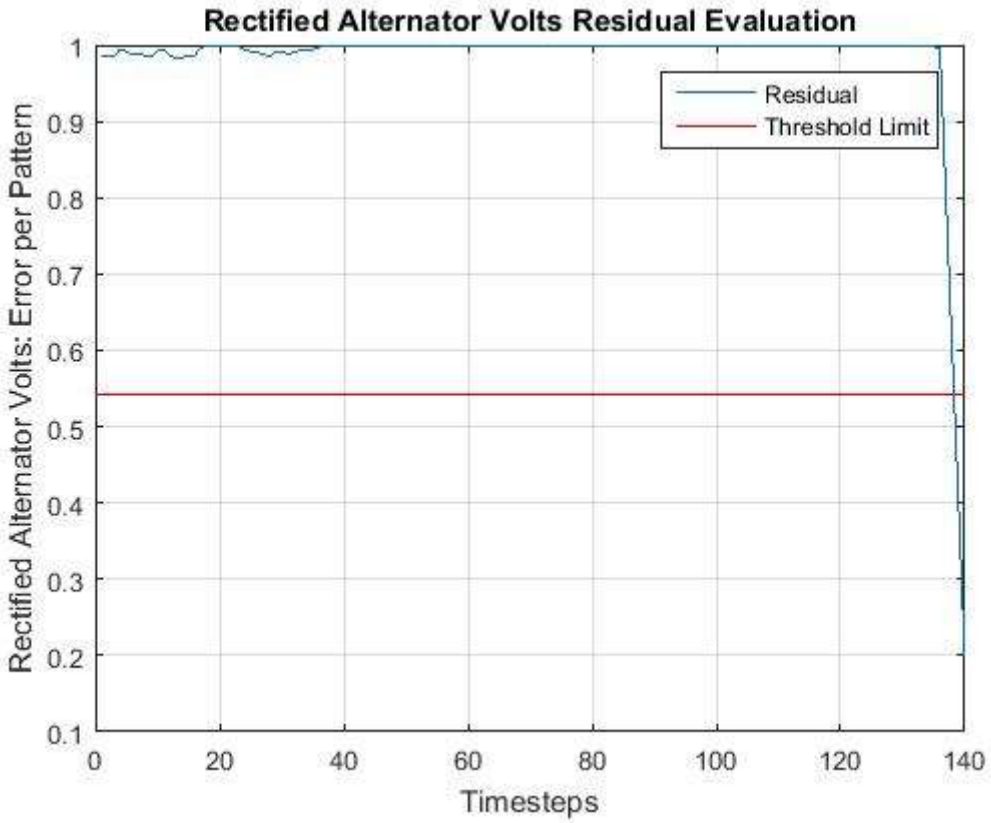


Figure L1.2.7: SCM8 Test 2: SCM8 Sensor Validation Test Result

L1.3 SCM8 Test 3 Results

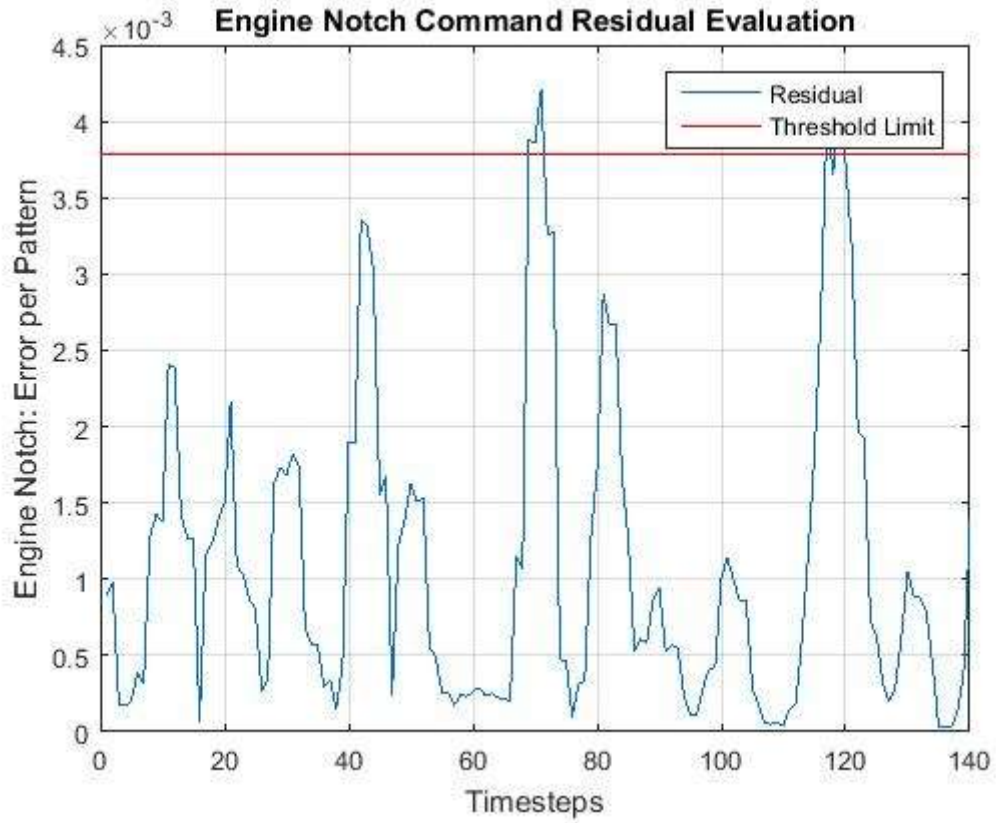


Figure L1.3.1: SCM8 Test 3: Engine Notch Command Test Result

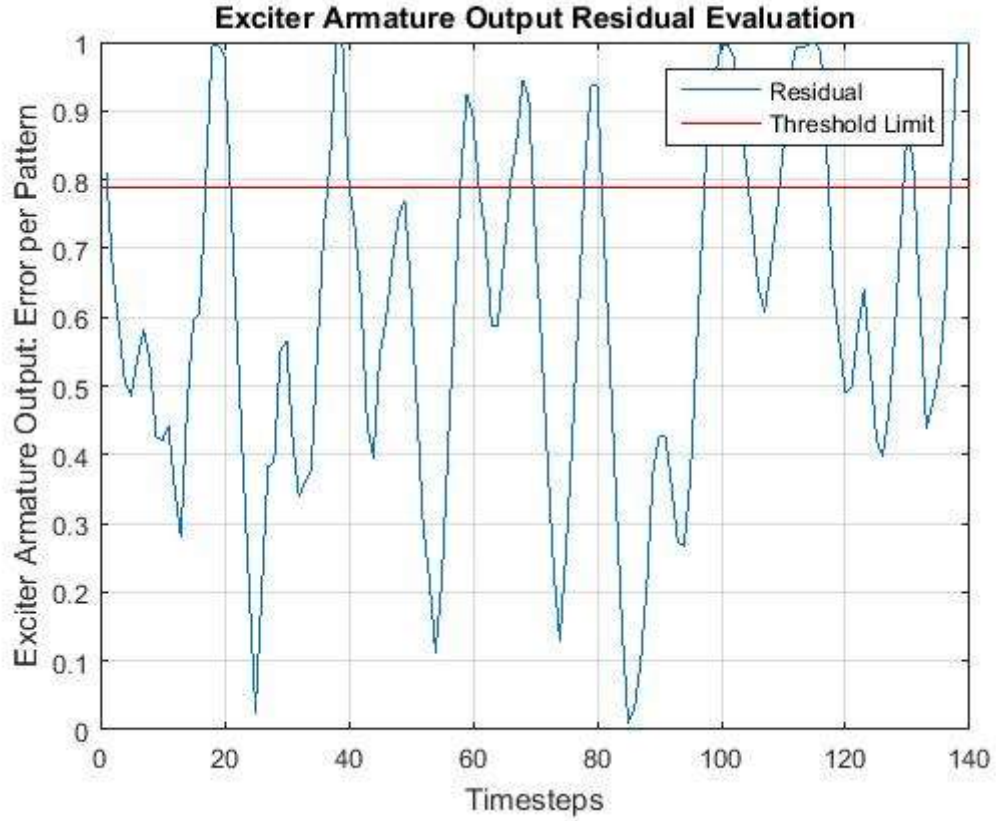


Figure L1.3.2: SCM8 Test 3: EXACT Test Result

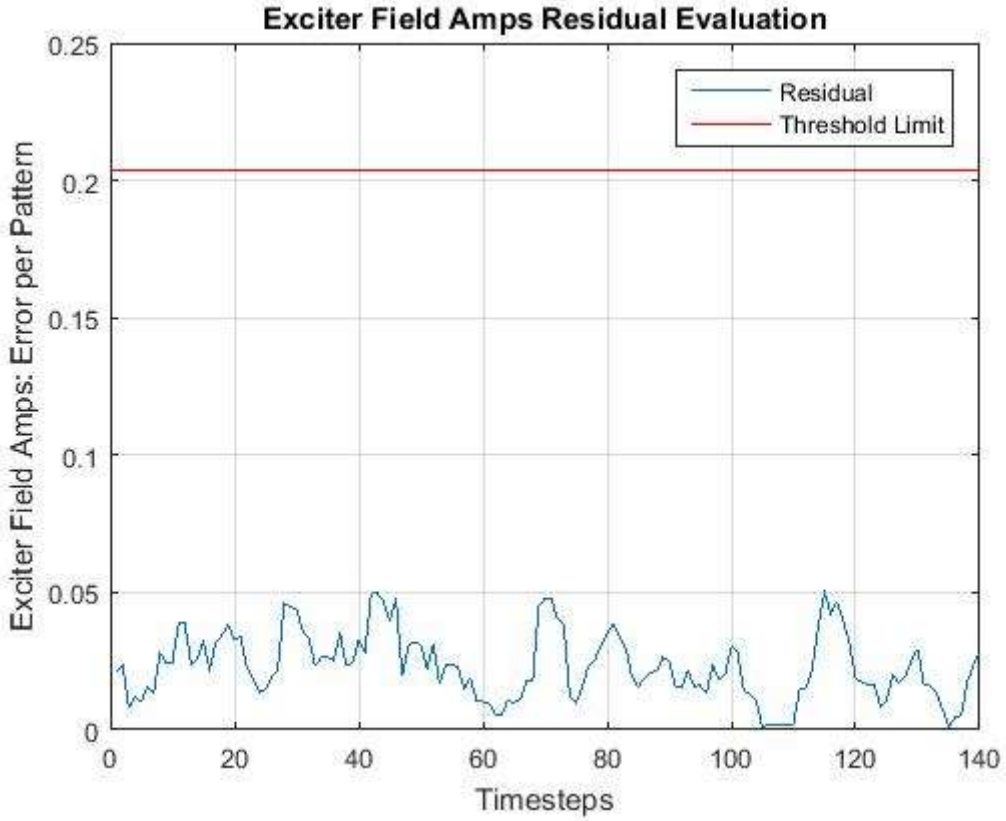


Figure L1.3.3: SCM8 Test 3: EXFM Test Result

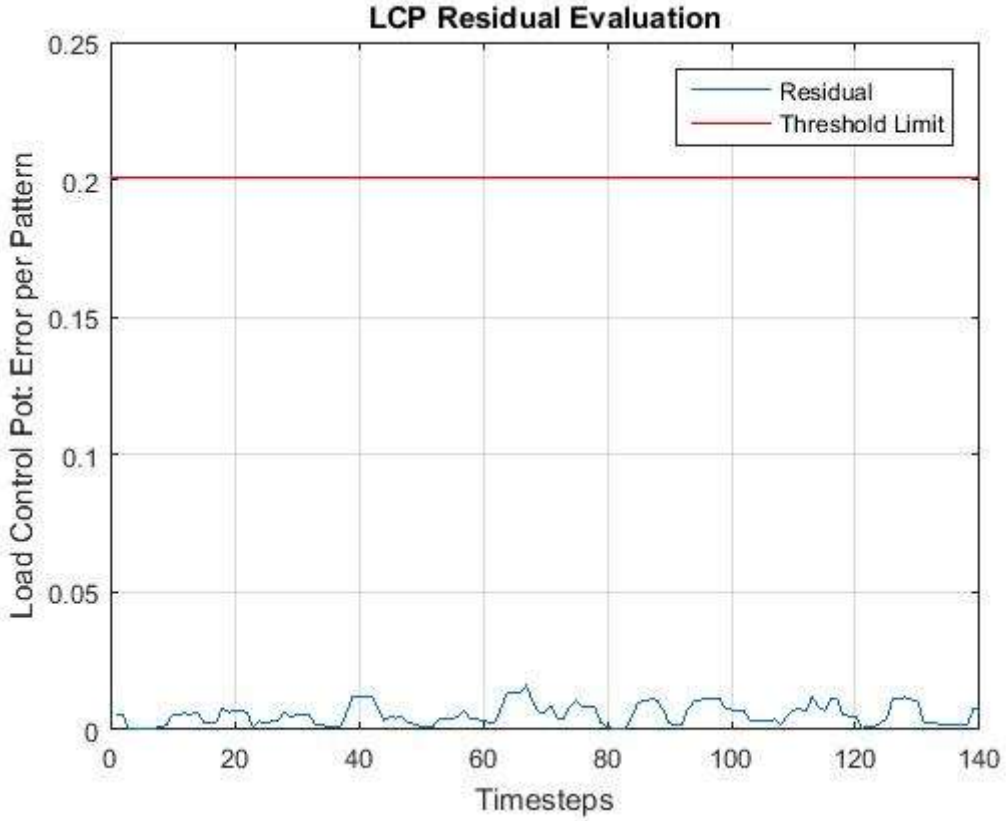


Figure L1.3.4: SCM8 Test 3: LCP Test Result

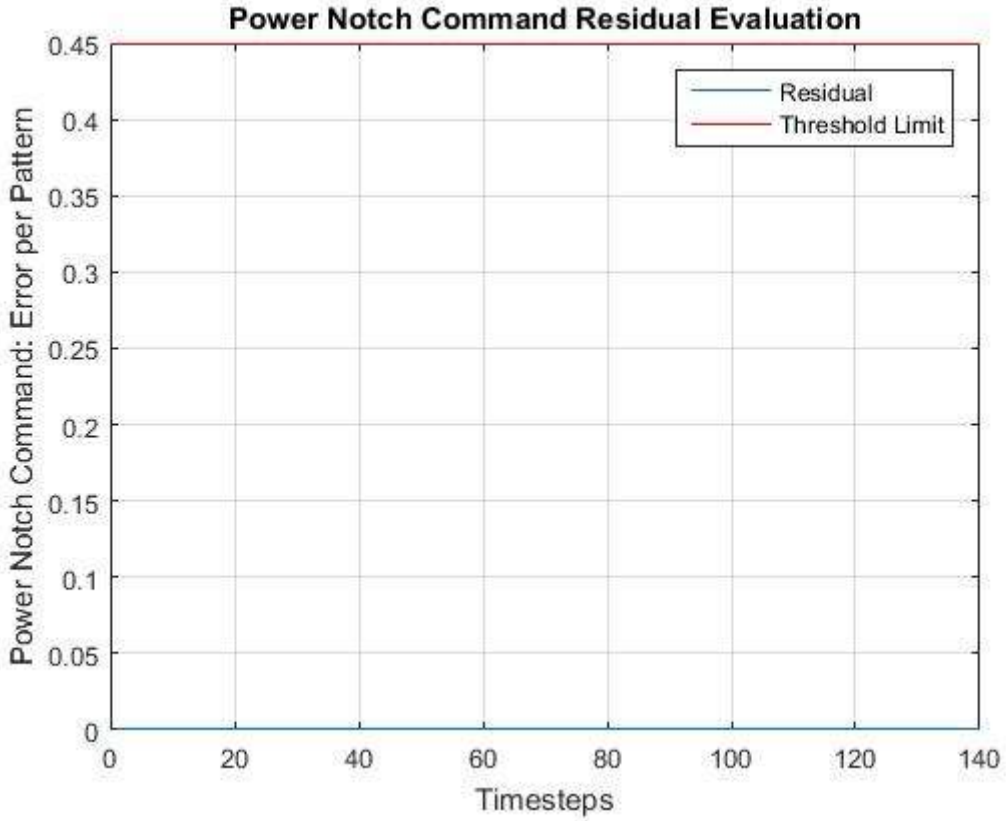


Figure L1.3.5: SCM8 Test 3: Power Notch Command Test Result

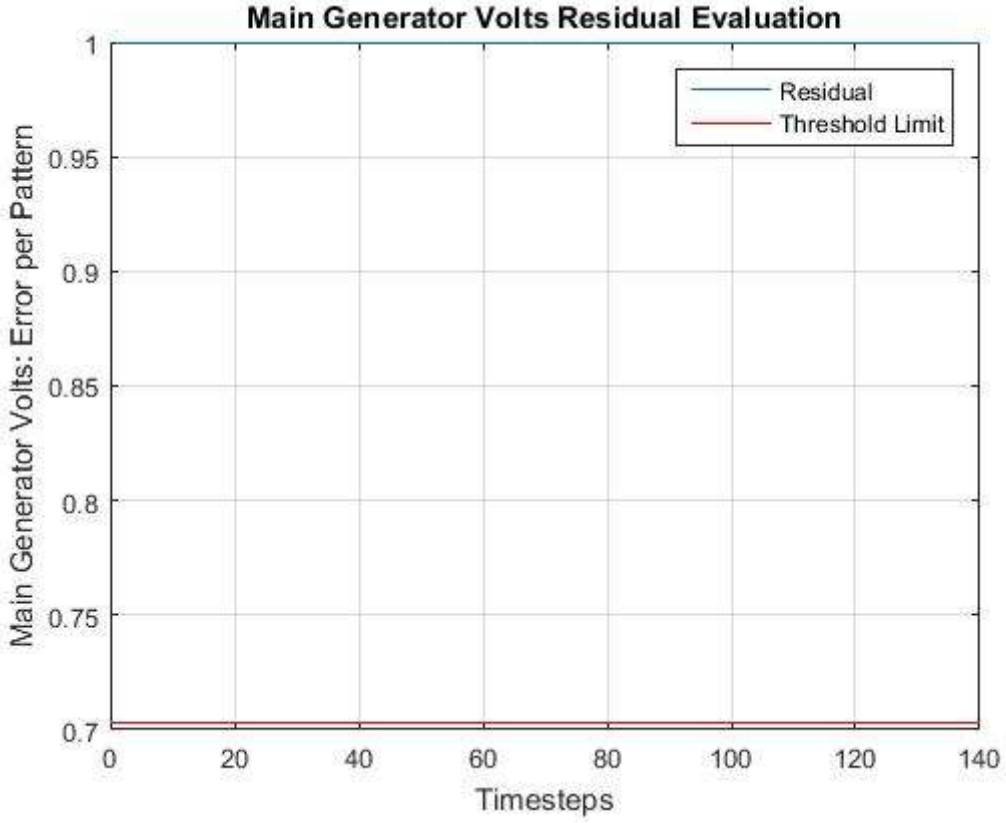


Figure L1.3.6: SCM8 Test 3: SCM8 Test Result

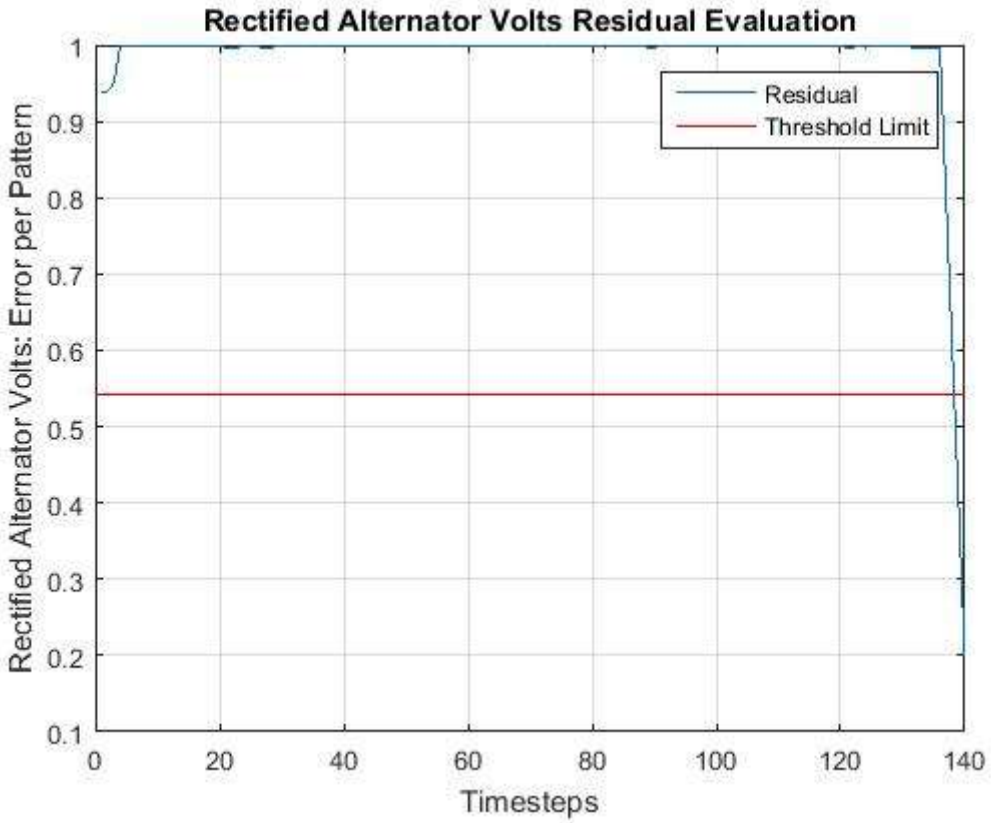


Figure L1.3.7: SCM8 Test 3: SCM8 Sensor Validation Test Result

L1.4 SCM8 Test 4 Results

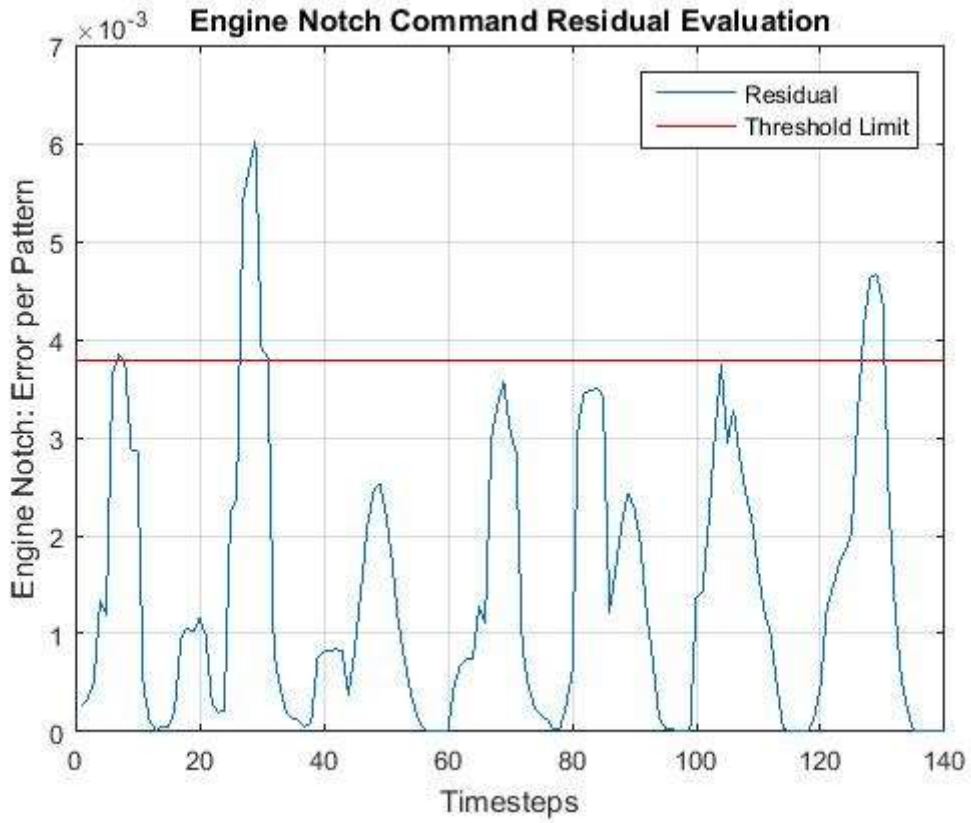


Figure L1.4.1: SCM8 Test 4: Engine Notch Command Test Result

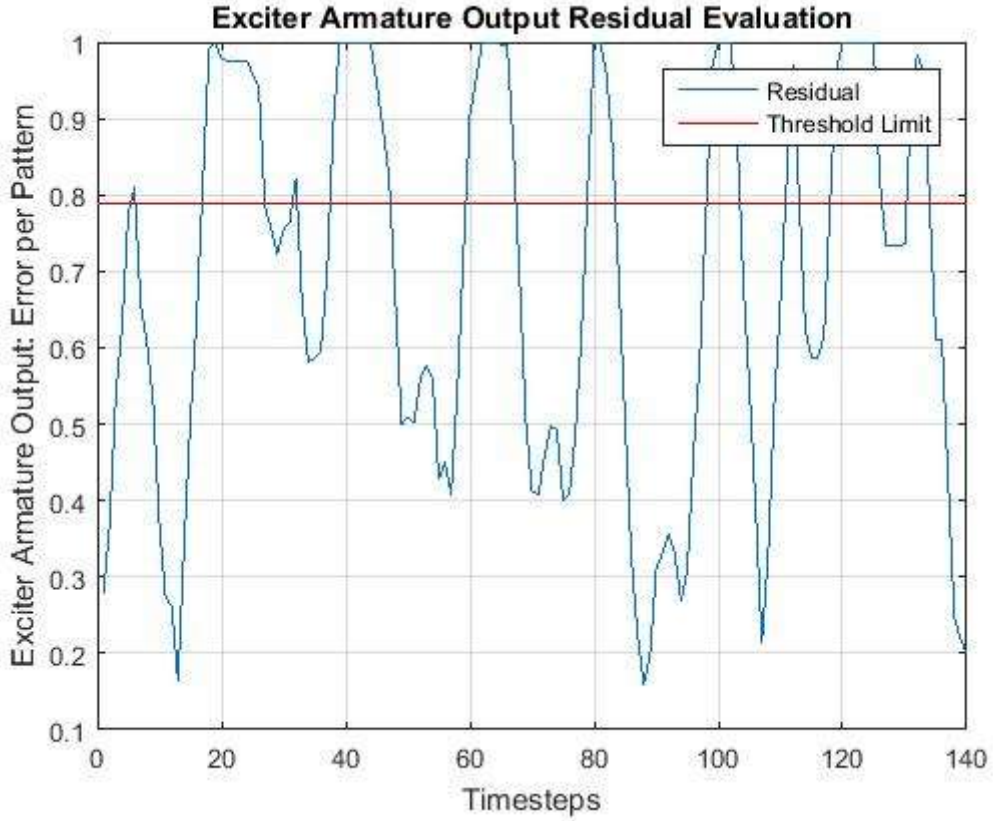


Figure L1.4.2: SCM8 Test 4: EXACT Test Result

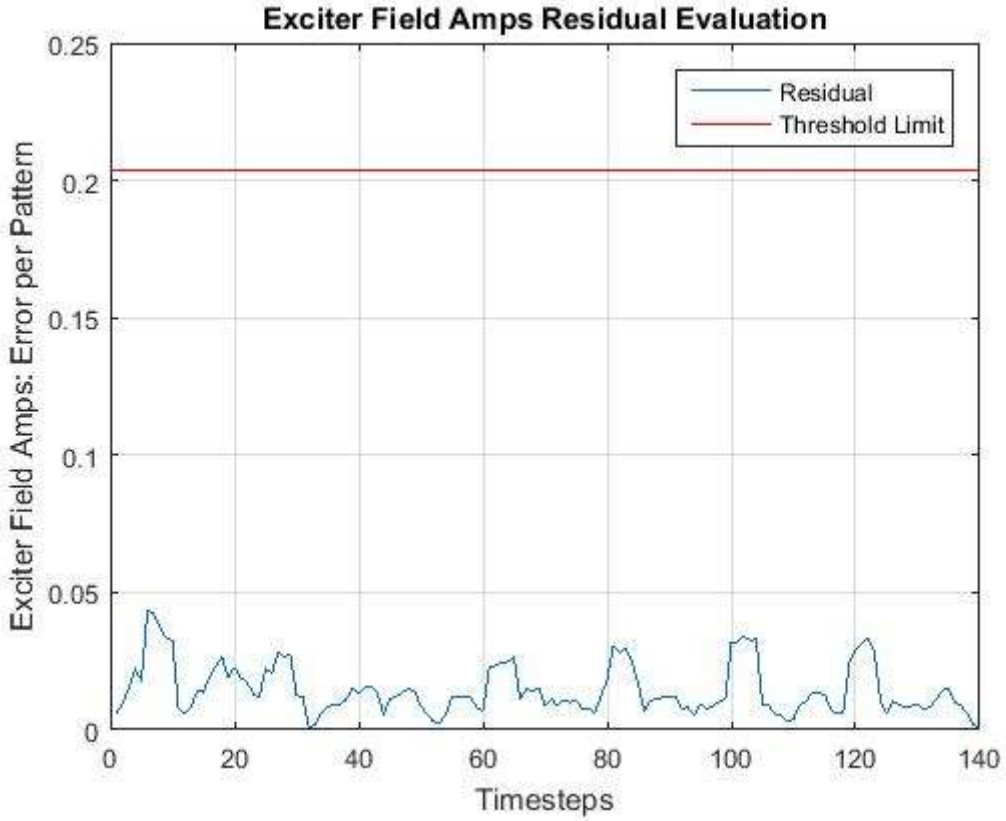


Figure L1.4.3: SCM8 Test 4: EXFM Test Result

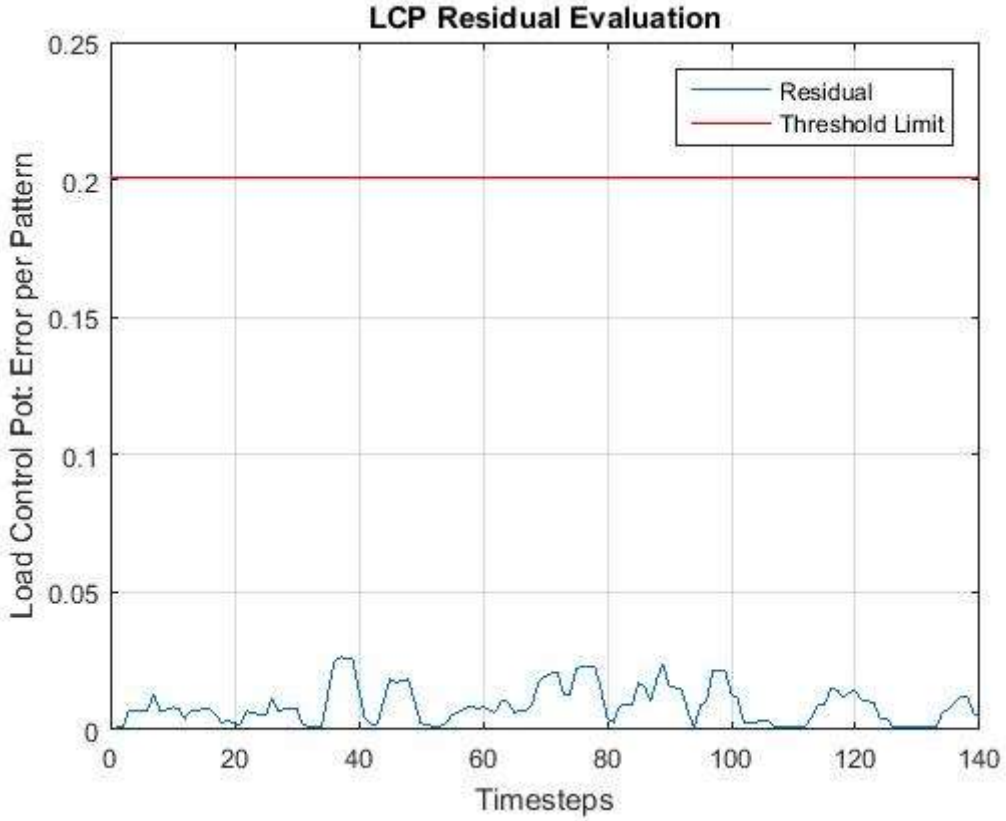


Figure L1.4.4: SCM8 Test 4: LCP Test Result

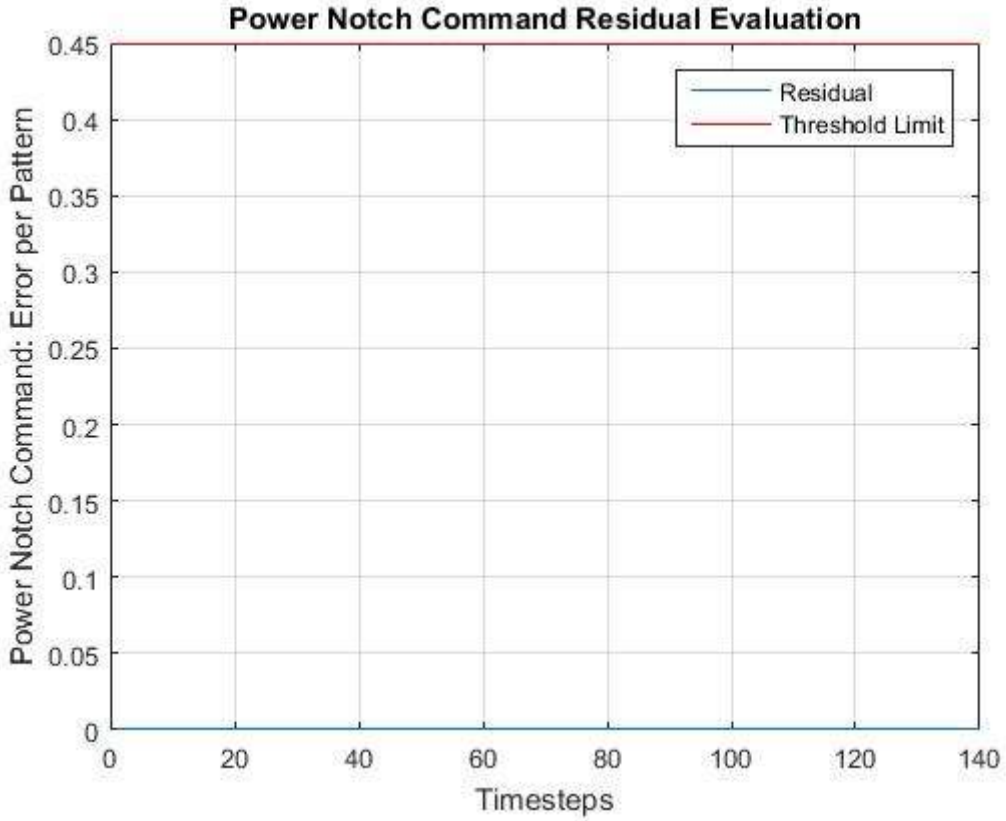


Figure L1.4.5: SCM8 Test 4: Power Notch Command Test Result

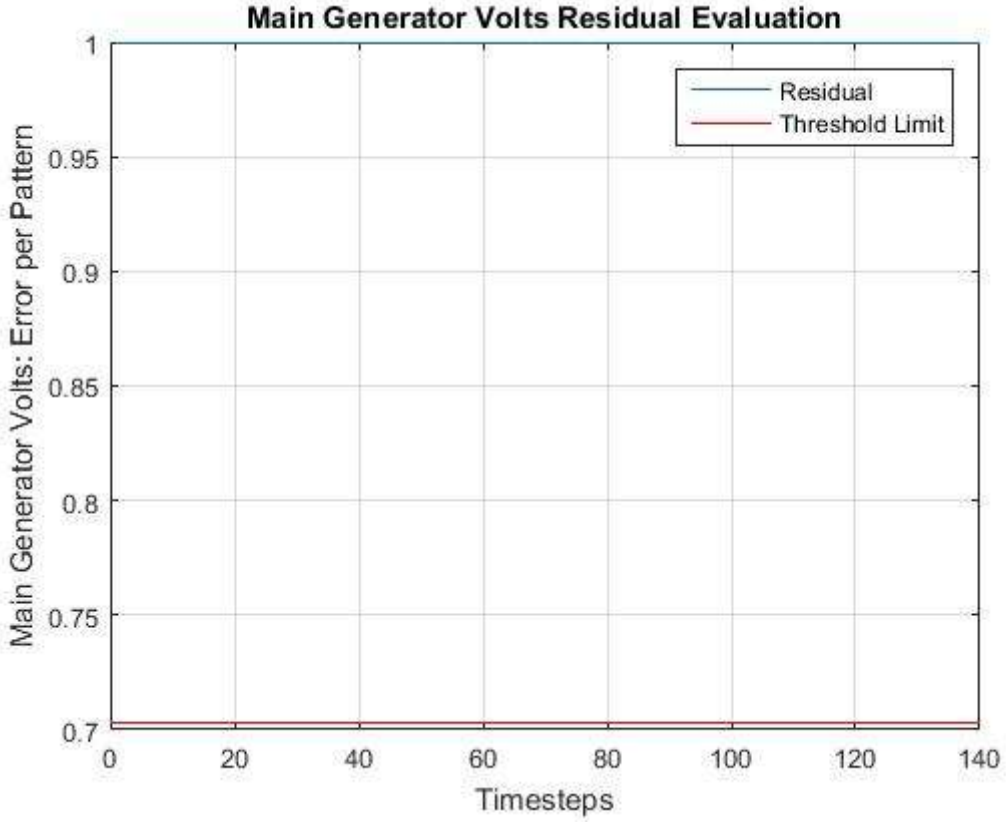


Figure L1.4.6: SCM8 Test 4: SCM8 Test Result

L1.5 SCM8 Test 5 Results

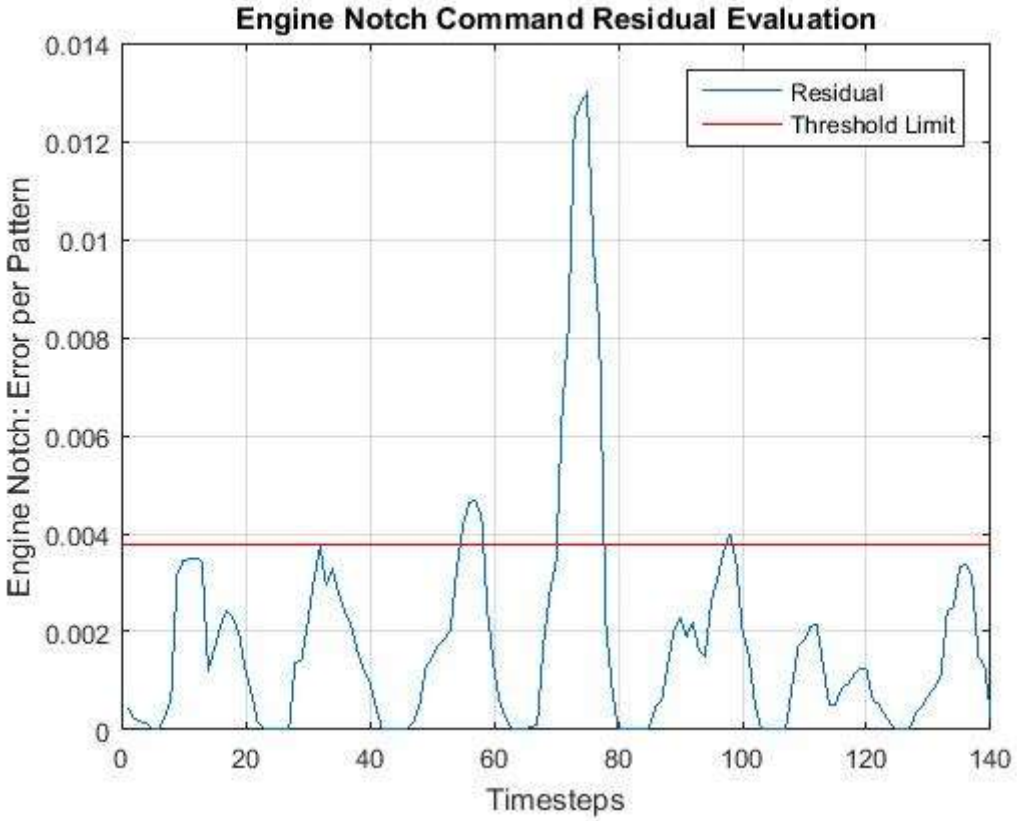


Figure L1.5.1: SCM8 Test 5: Engine Notch Command Test Result

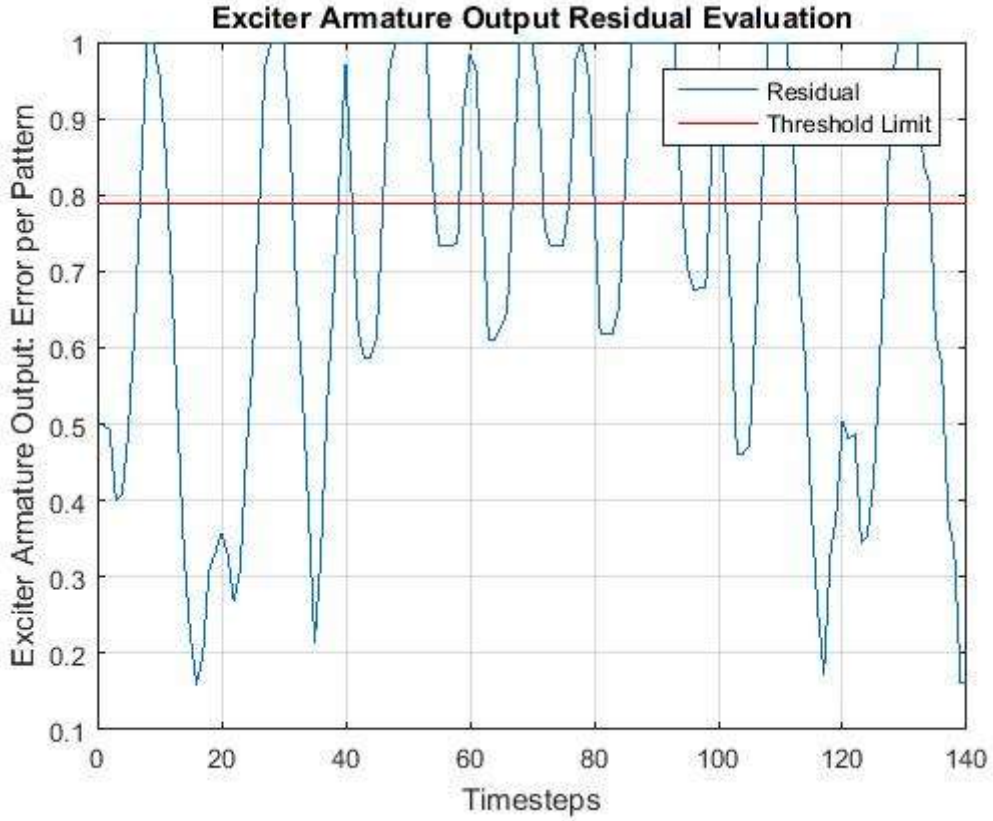


Figure L1.5.2: SCM8 Test 5: EXACT Test Result

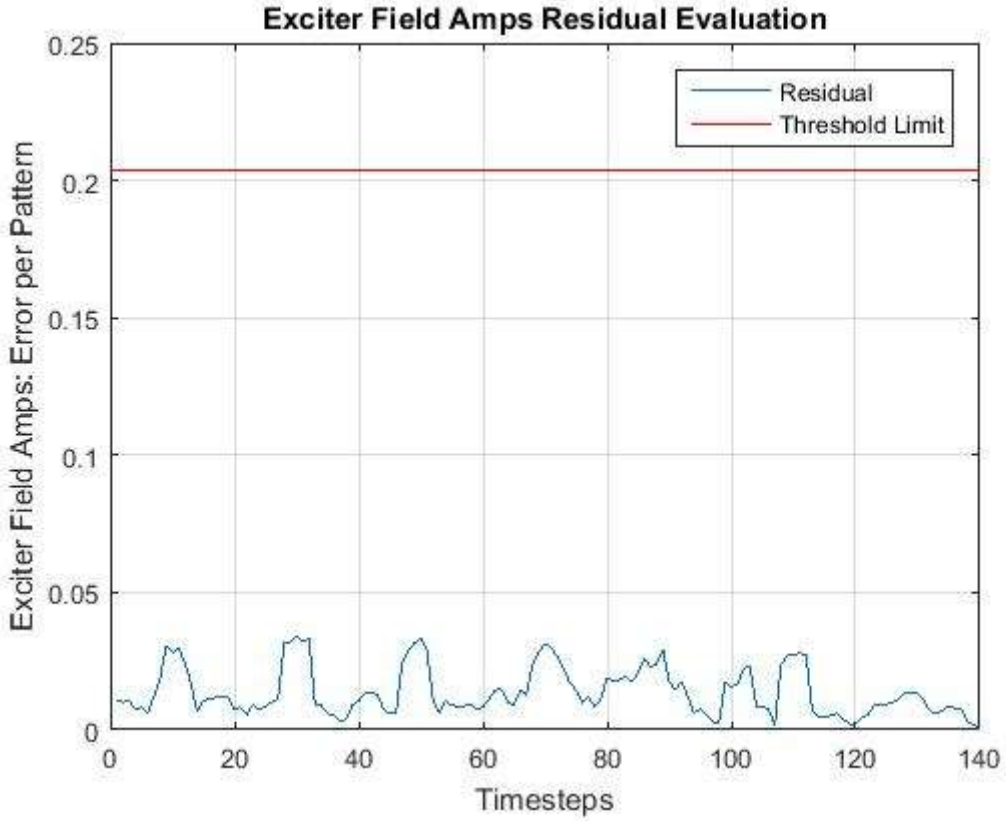


Figure L1.5.3: SCM8 Test 5: EXFM Test Result

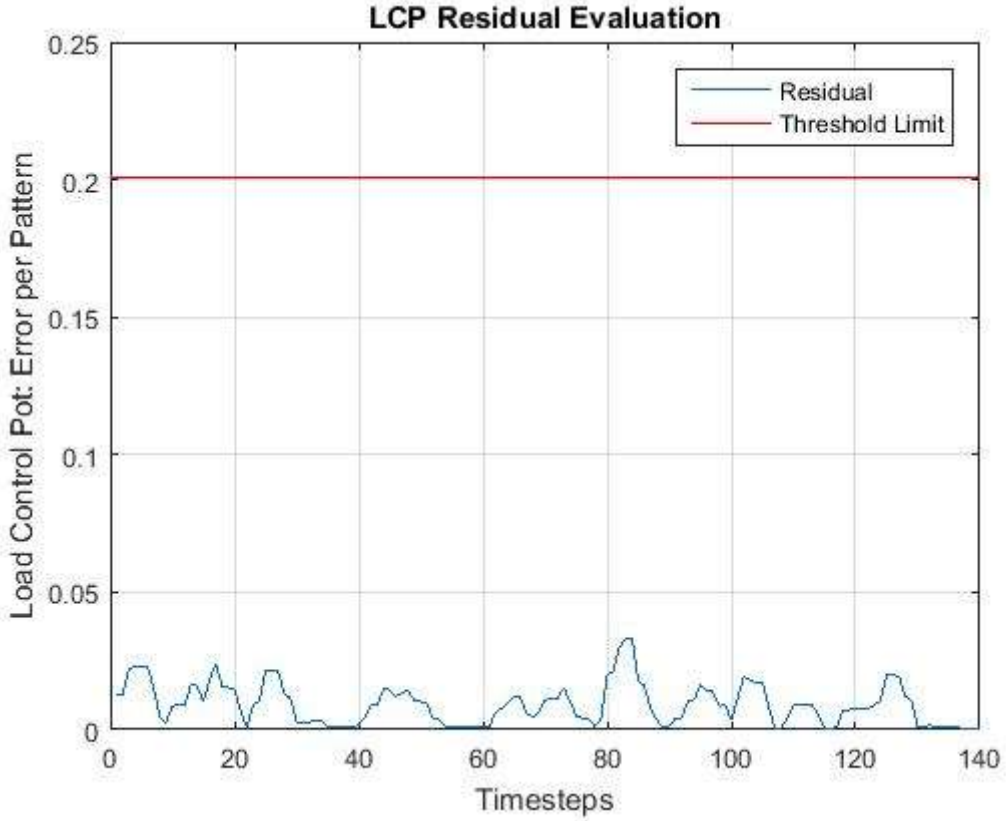


Figure L1.5.4: SCM8 Test 5: LCP Test Result

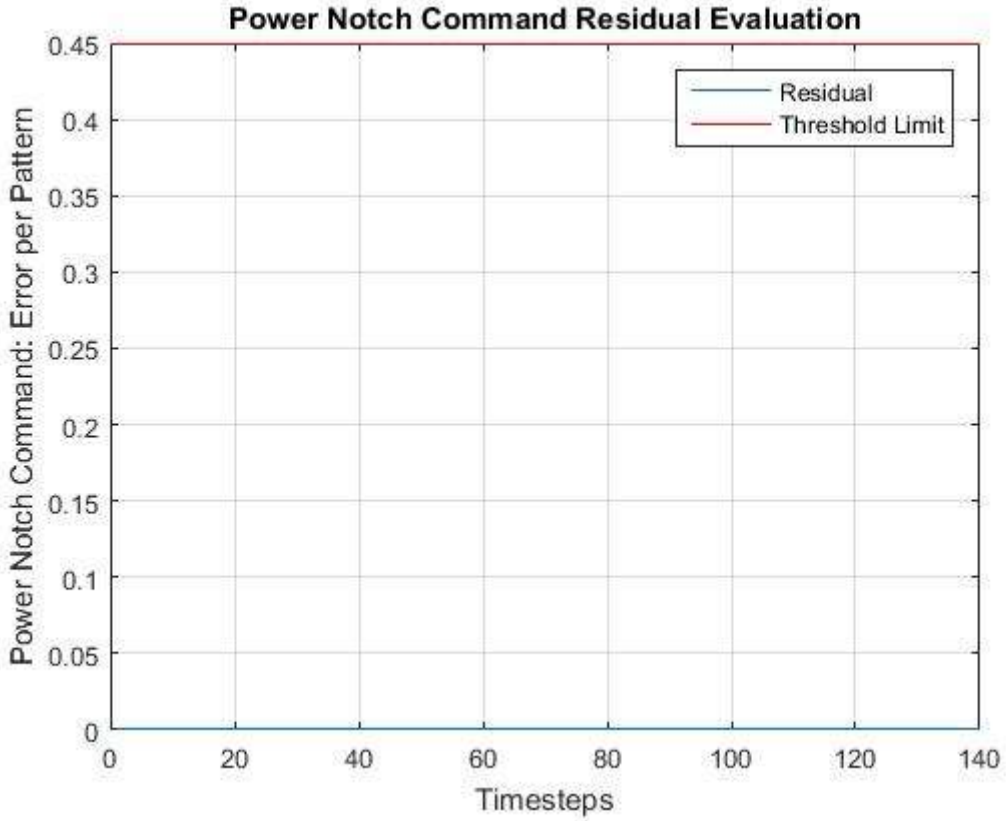


Figure L1.5.5: SCM8 Test 5: Power Notch Command Test Result

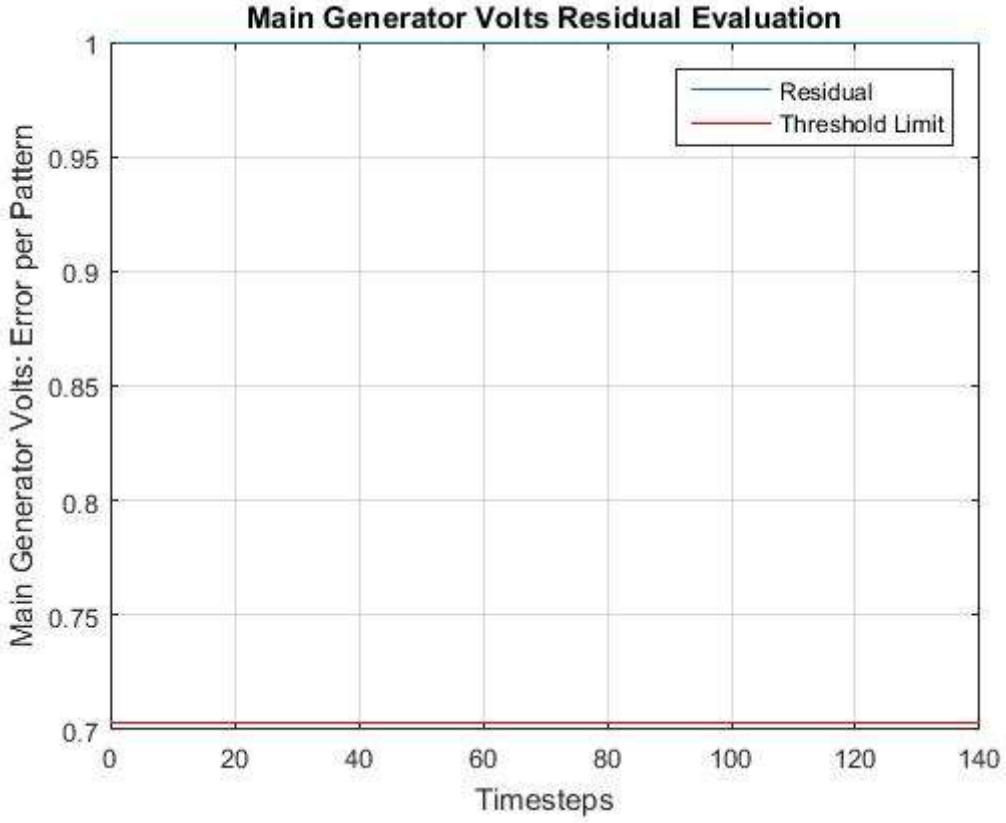


Figure L1.5.6: SCM8 Test 5: SCM8 Test Result

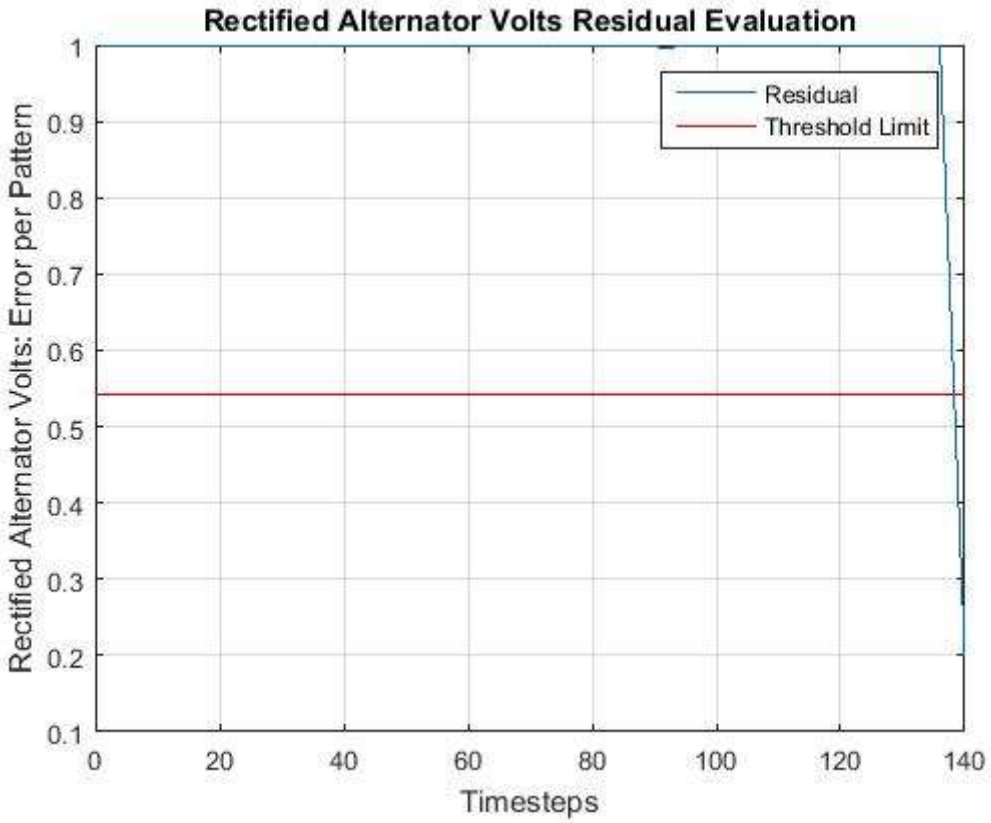


Figure L1.5.7: SCM8 Test 5: SCM8 Sensor Validation Test Result

L1.6 SCM8 Test 6 Results

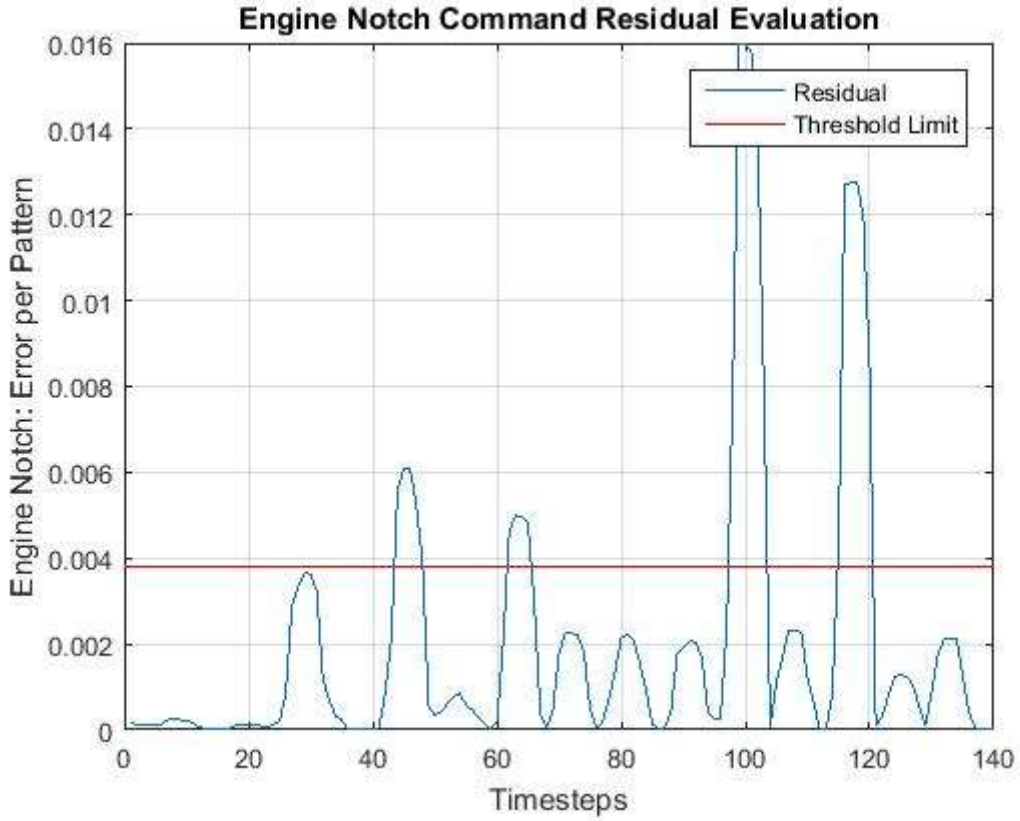


Figure L1.6.1: SCM8 Test 6: Engine Notch Command Test Result

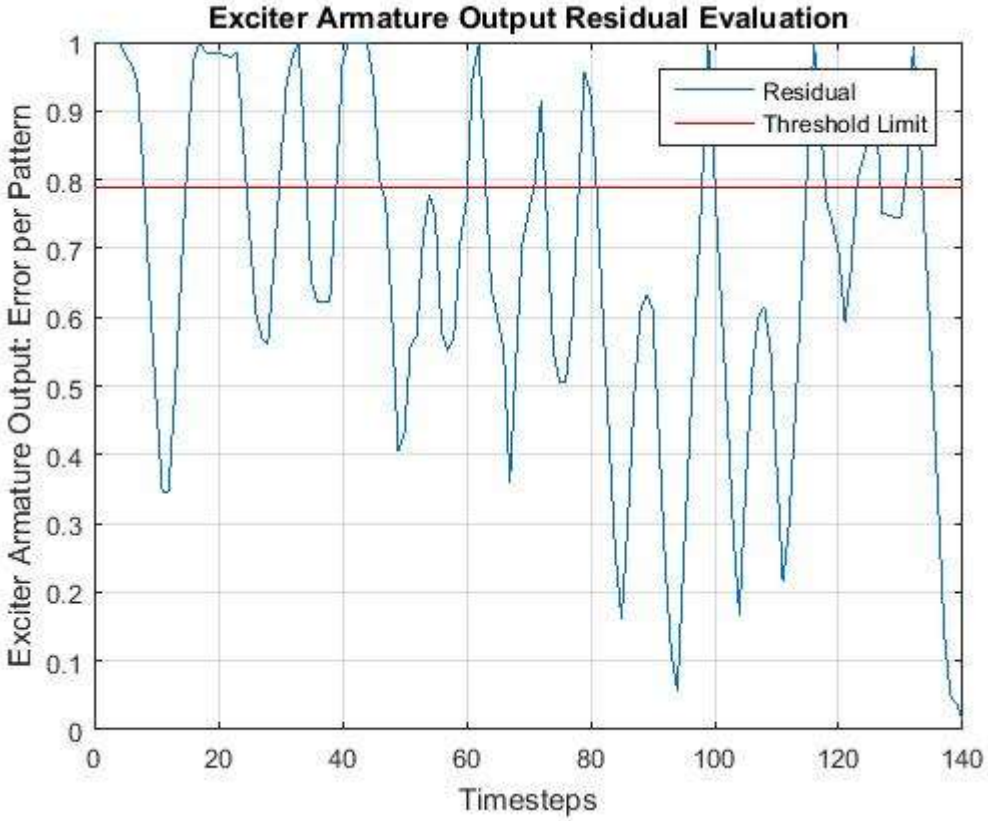


Figure L1.6.2: SCM8 Test 6: EXACT Test Result

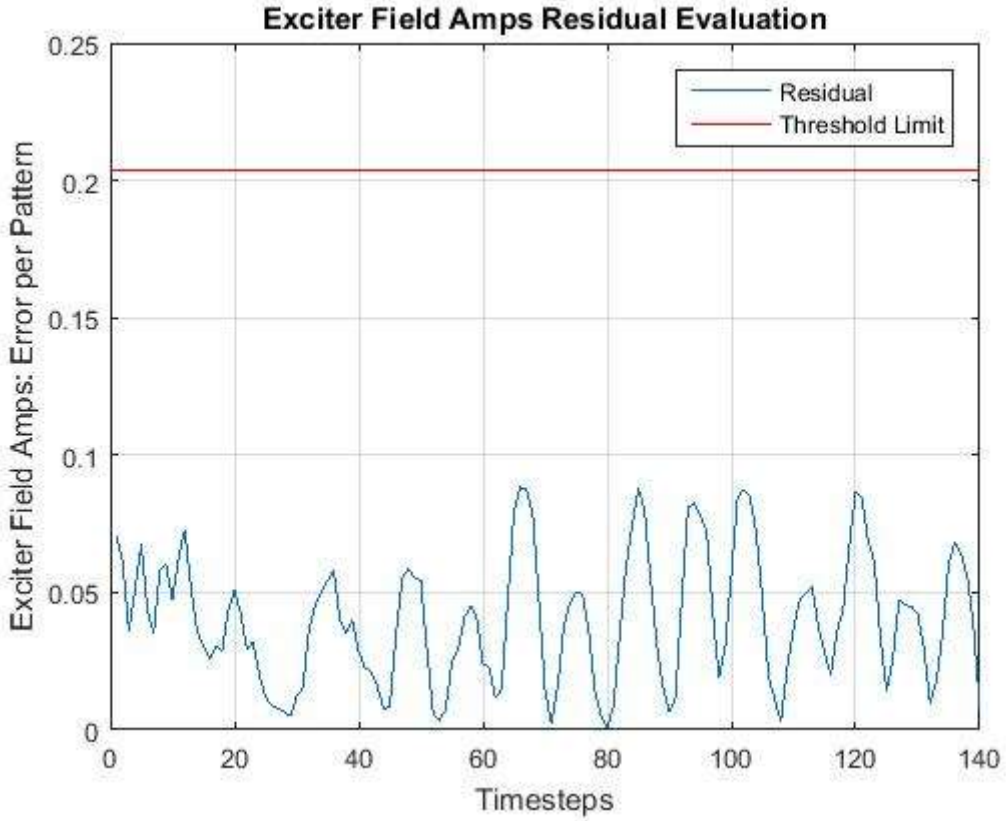


Figure L1.6.3: SCM8 Test 6: EXFM Test Result

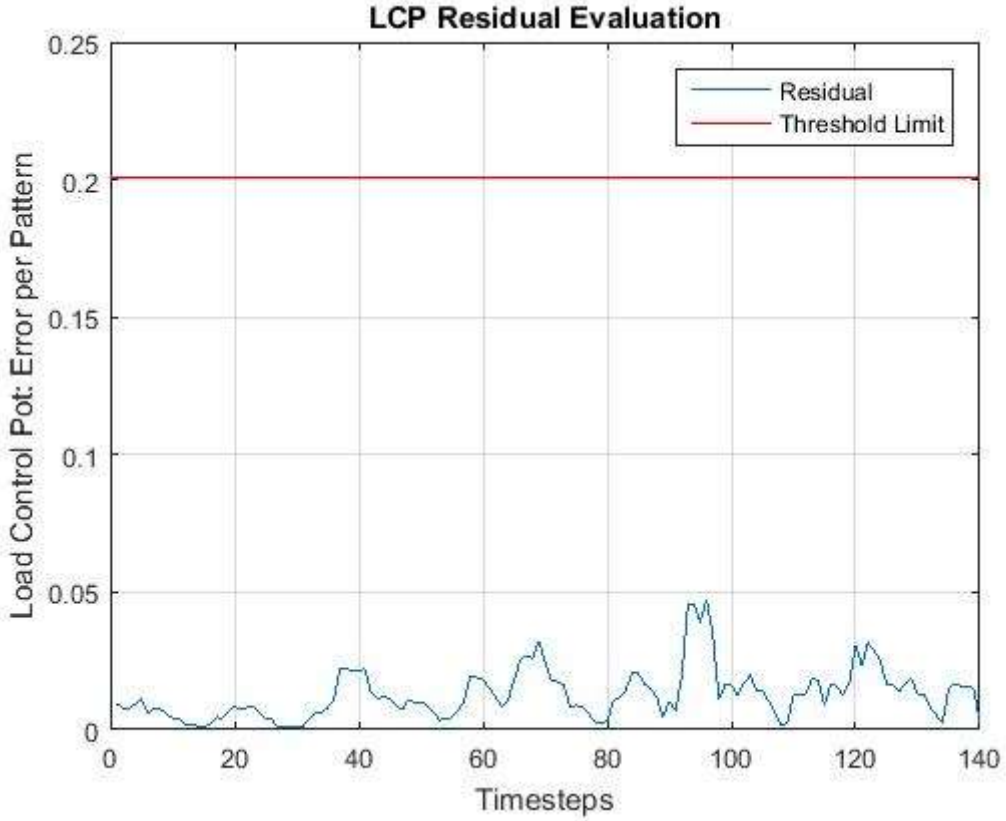


Figure L1.6.4: SCM8 Test 6: LCP Test Result

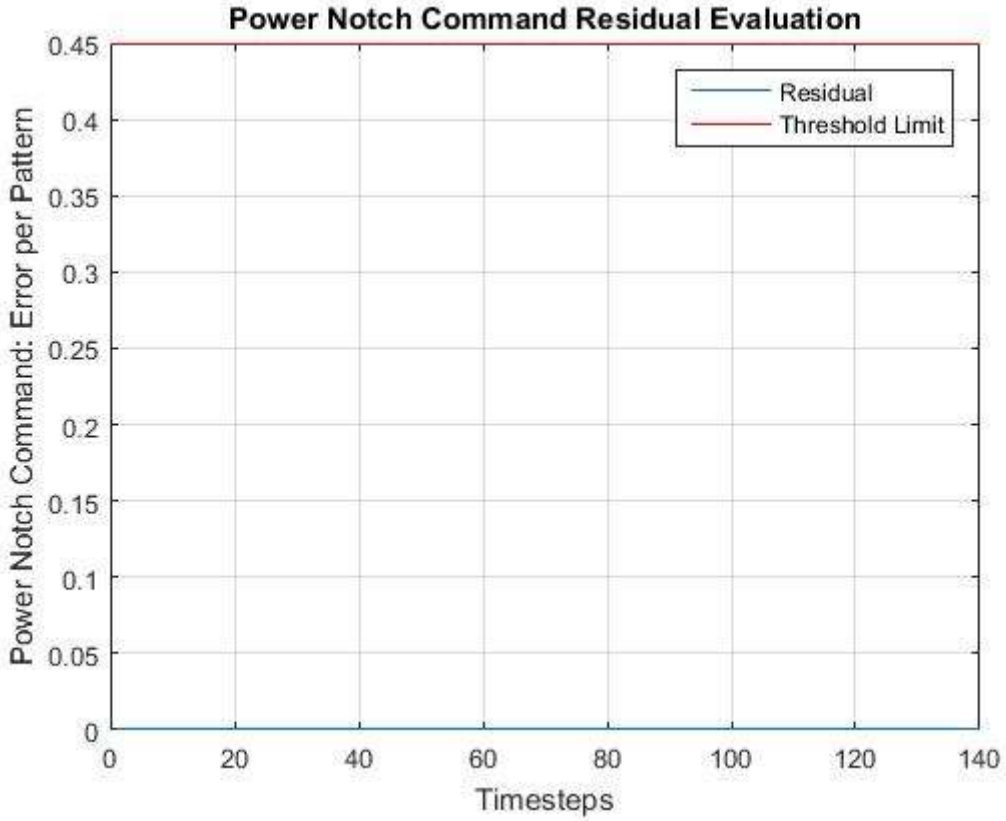


Figure L1.6.5: SCM8 Test 6: Power Notch Command Test Result

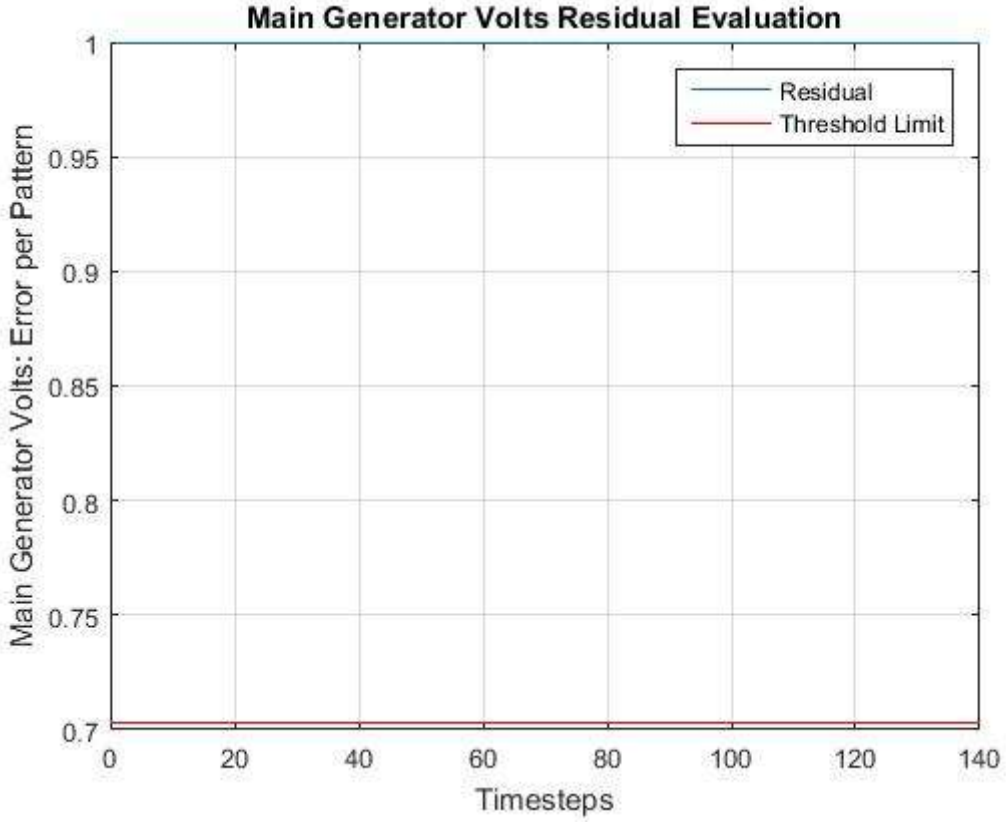


Figure L1.6.6: SCM8 Test 6: SCM8 Test Result

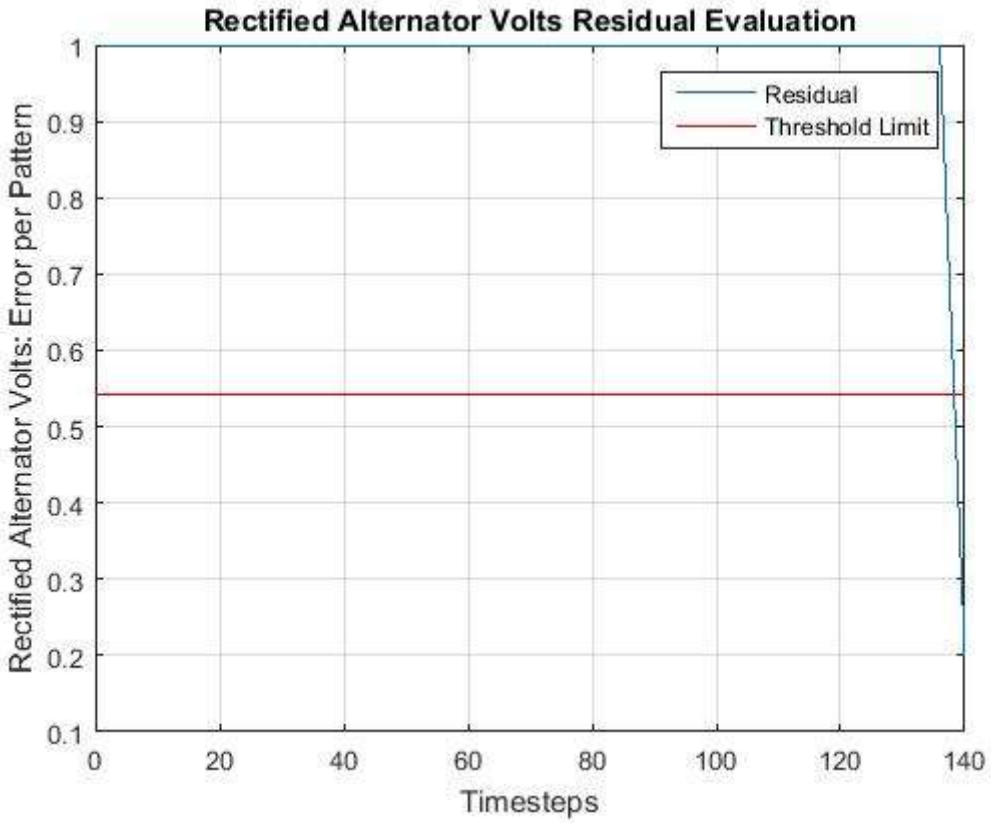


Figure L1.6.7: SCM8 Test 6: SCM8 Sensor Validation Test Result

L1.7 SCM8 Test 7 Results

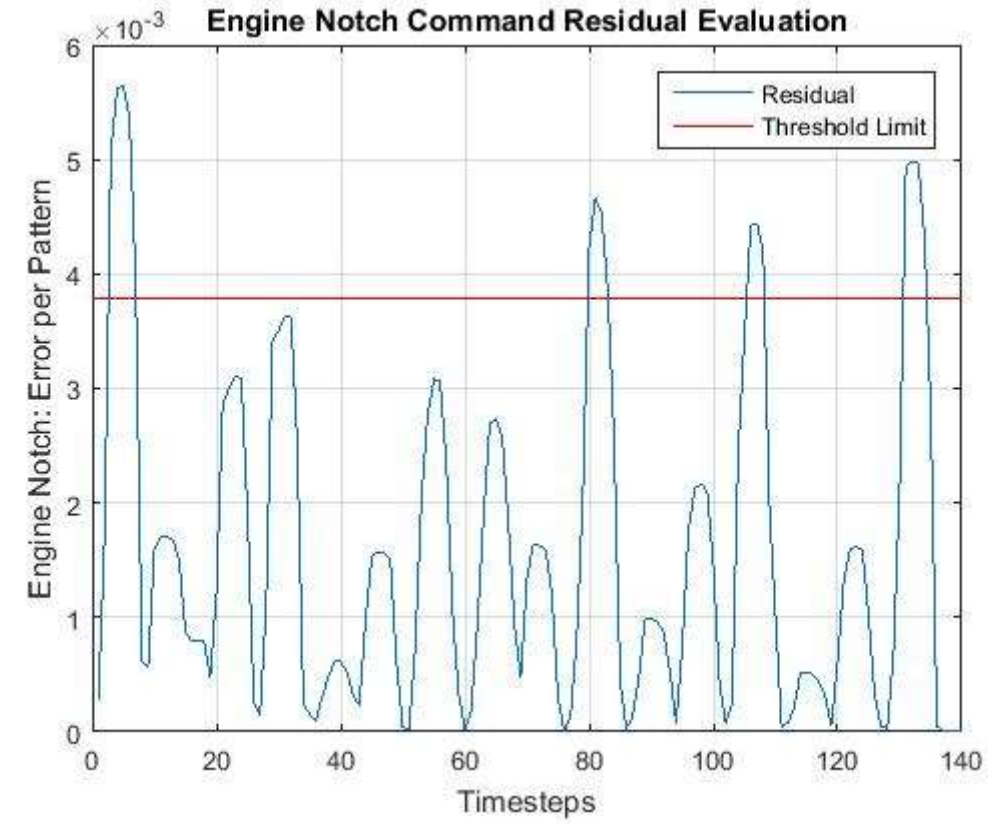


Figure L1.7.1: SCM8 Test 7: Engine Notch Command Test Result

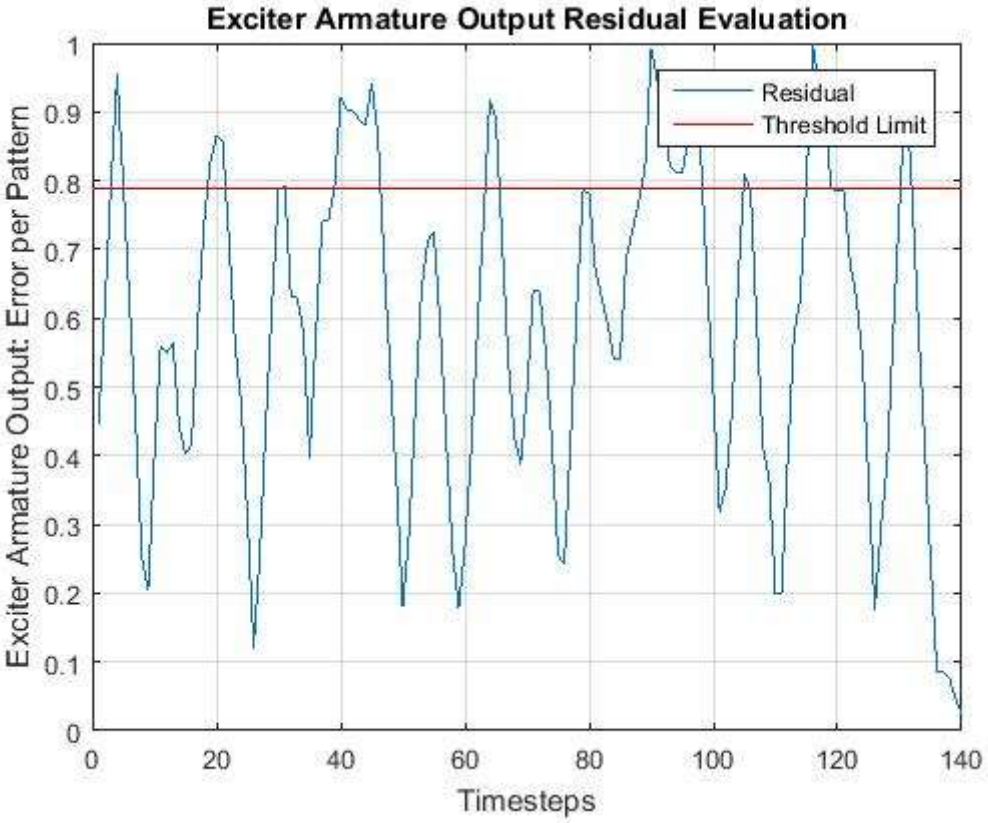


Figure L1.7.2: SCM8 Test 7: EXACT Test Result

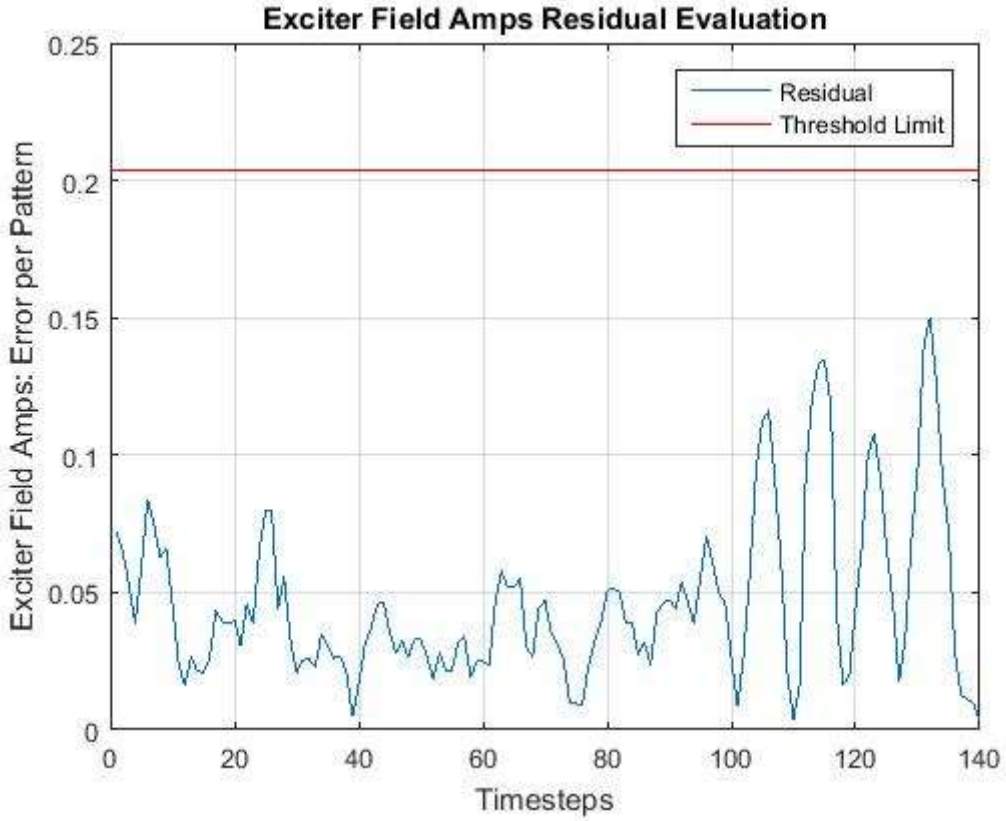


Figure L1.7.3: SCM8 Test 7: EXFM Test Result

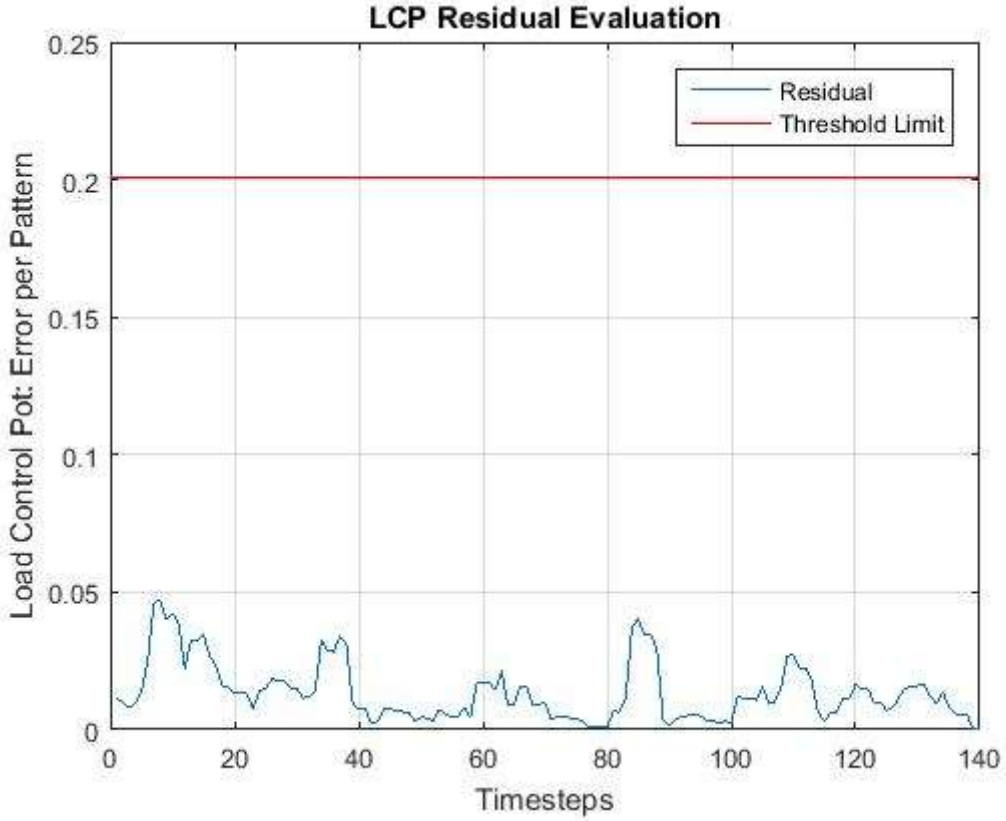


Figure L1.7.4: SCM8 Test 7: LCP Test Result

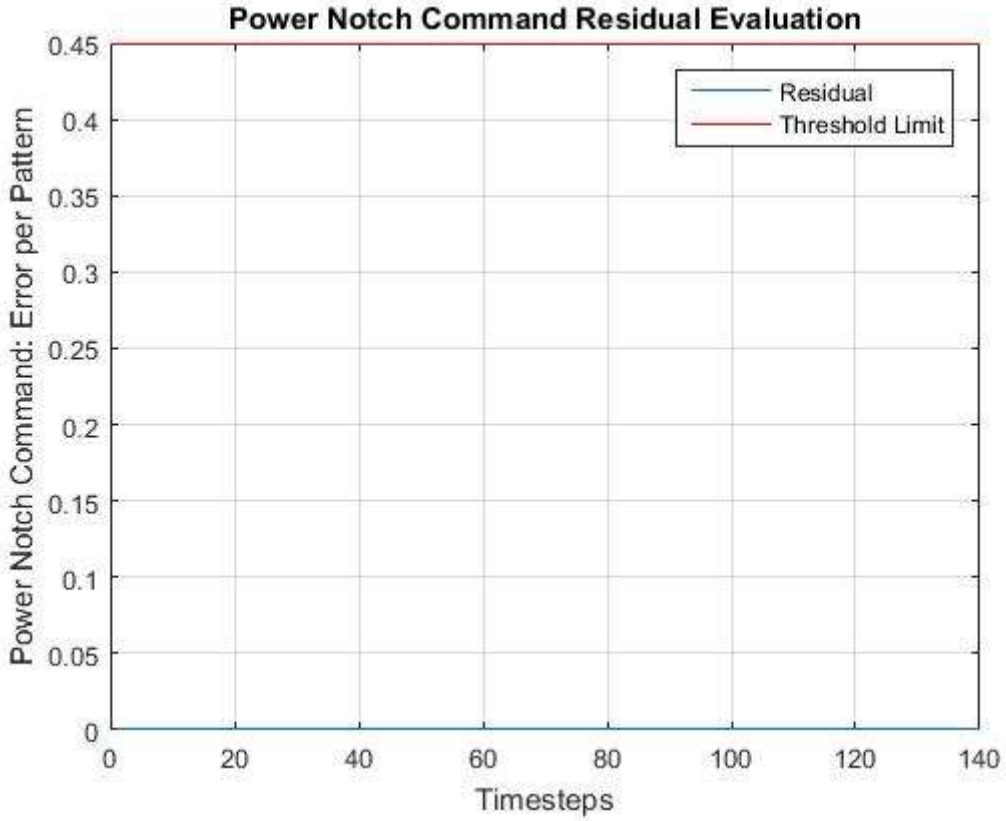


Figure L1.7.5: SCM8 Test 7: Power Notch Command Test Result

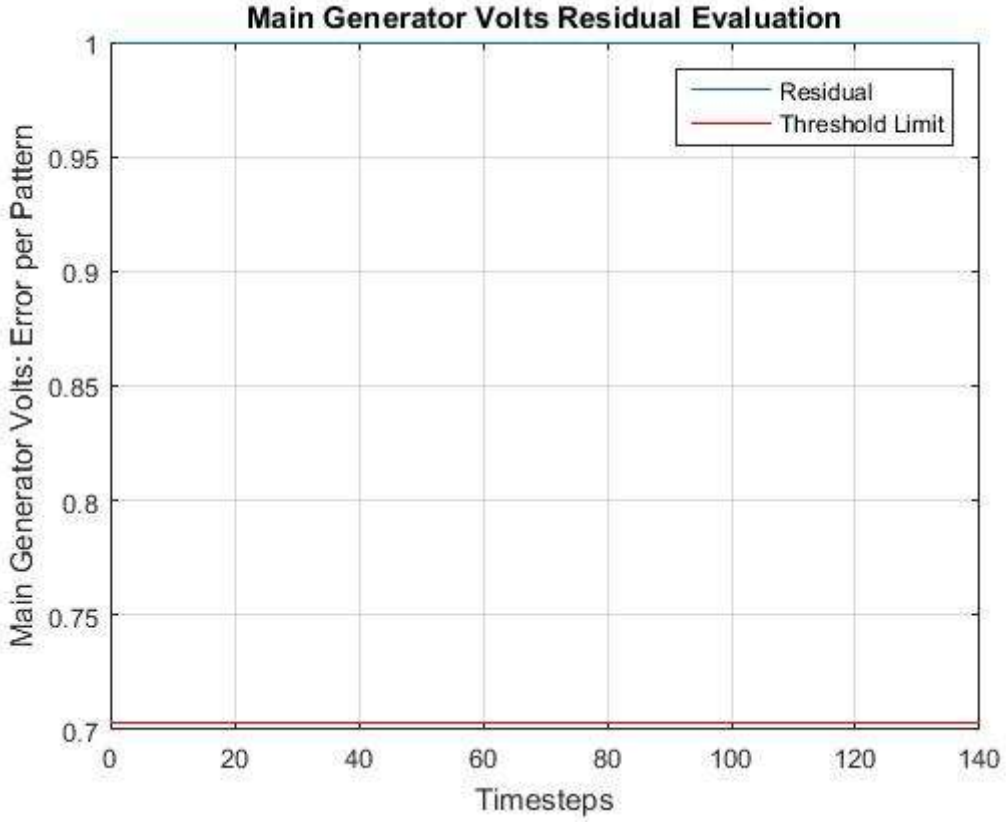


Figure L1.7.6: SCM8 Test 7: SCM8 Test Result

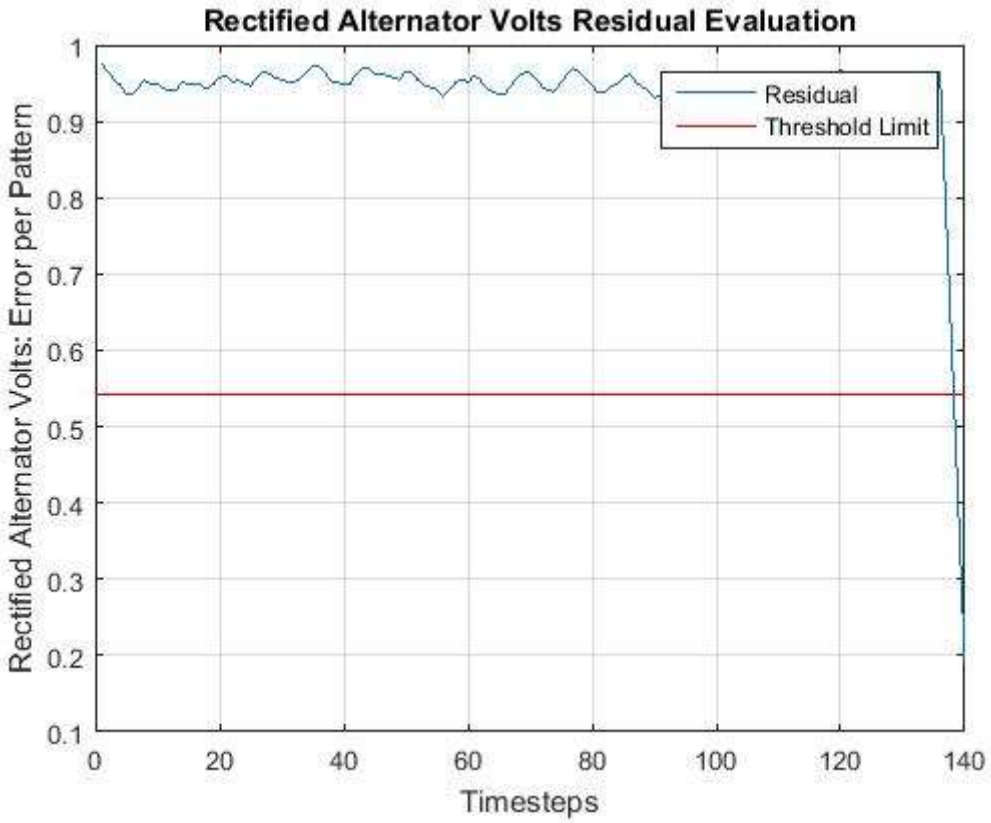


Figure L1.7.7: SCM8 Test 7: SCM8 Sensor Validation Test Result

L1.8 SCM8 Test 8 Results

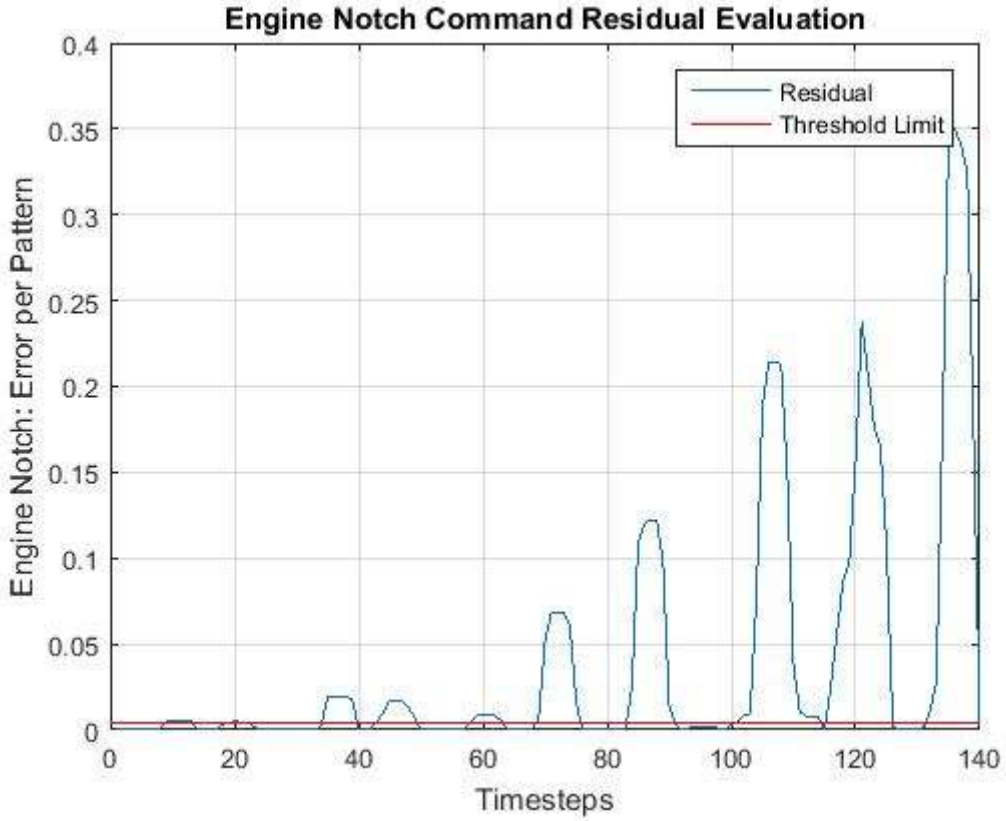


Figure L1.8.1: SCM8 Test 8: Engine Notch Command Test Result

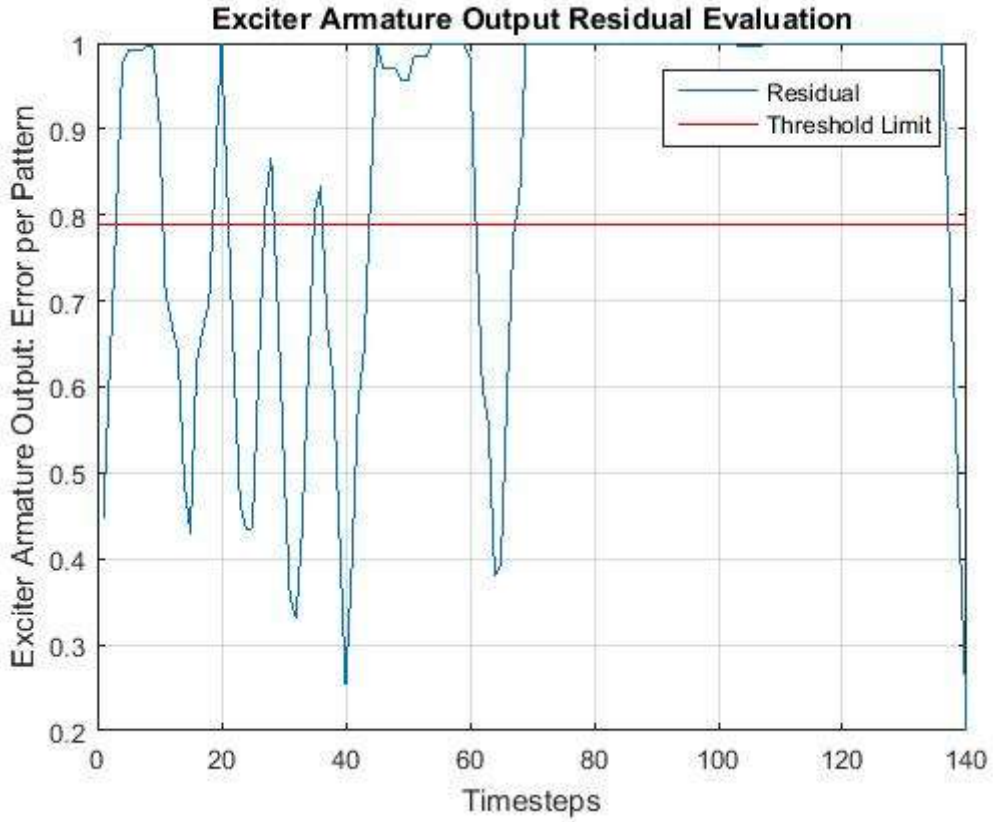


Figure L1.8.2: SCM8 Test 8: EXACT Test Result

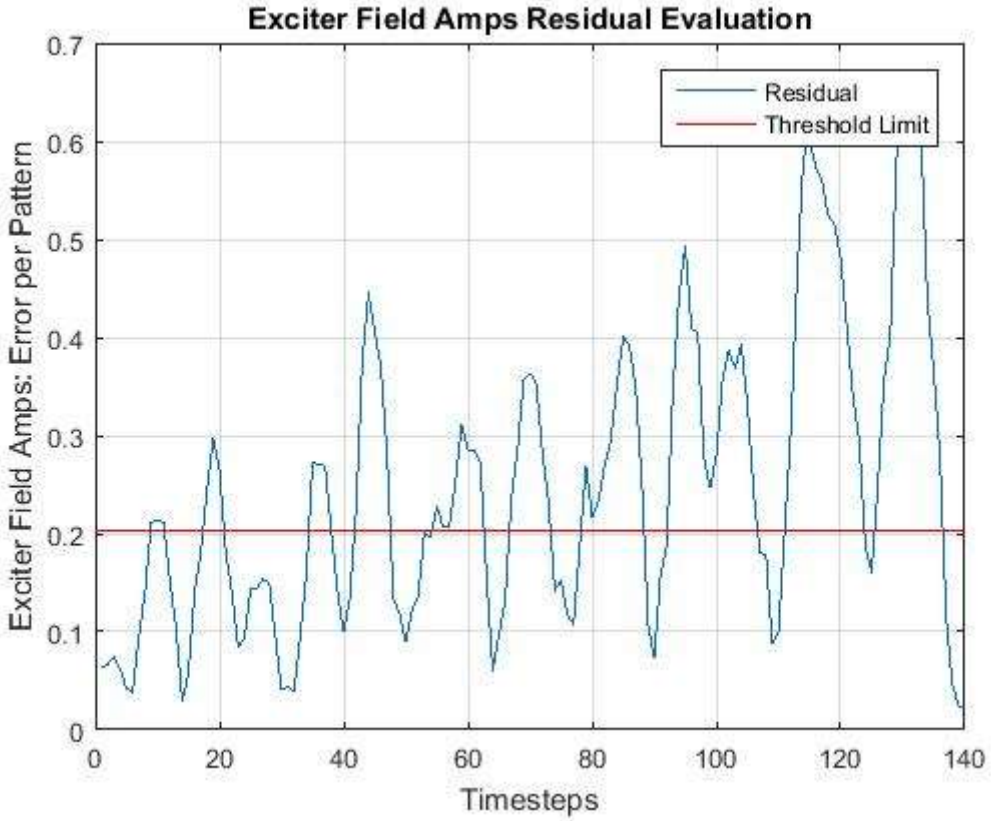


Figure L1.8.3: SCM8 Test 8: EXFM Test Result

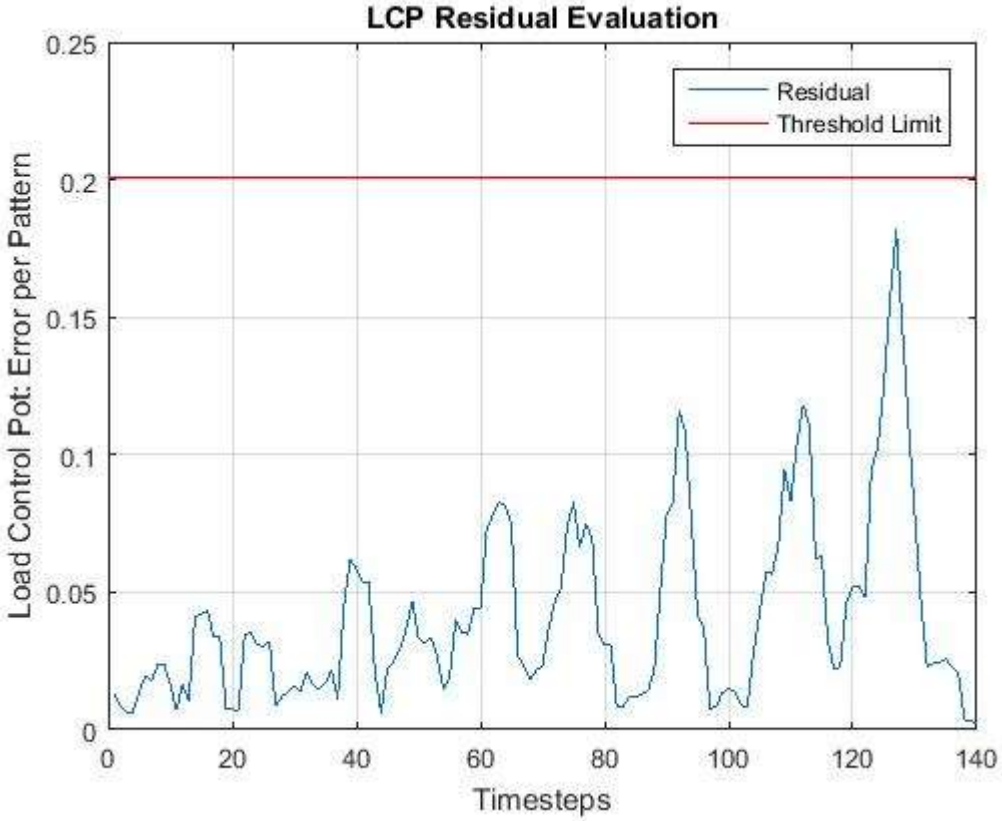


Figure L1.8.4: SCM8 Test 8: LCP Test Result

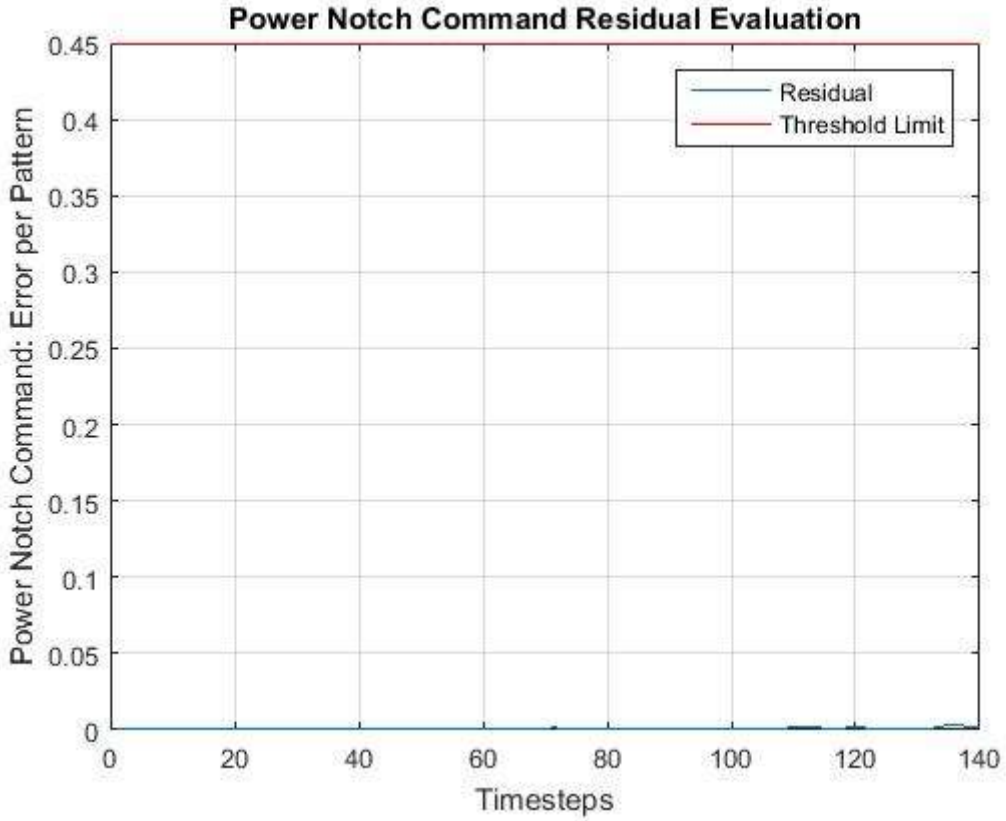


Figure L1.8.5: SCM8 Test 8: Power Notch Command Test Result

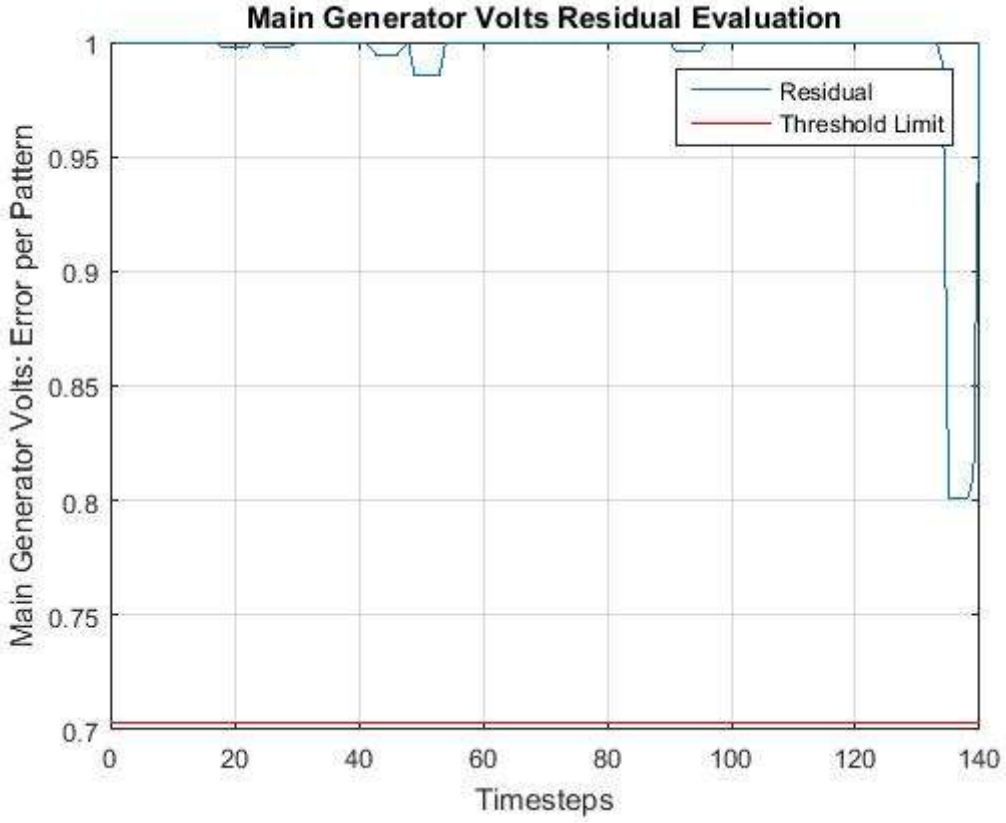


Figure L1.8.6: SCM8 Test 8: SCM8 Test Result

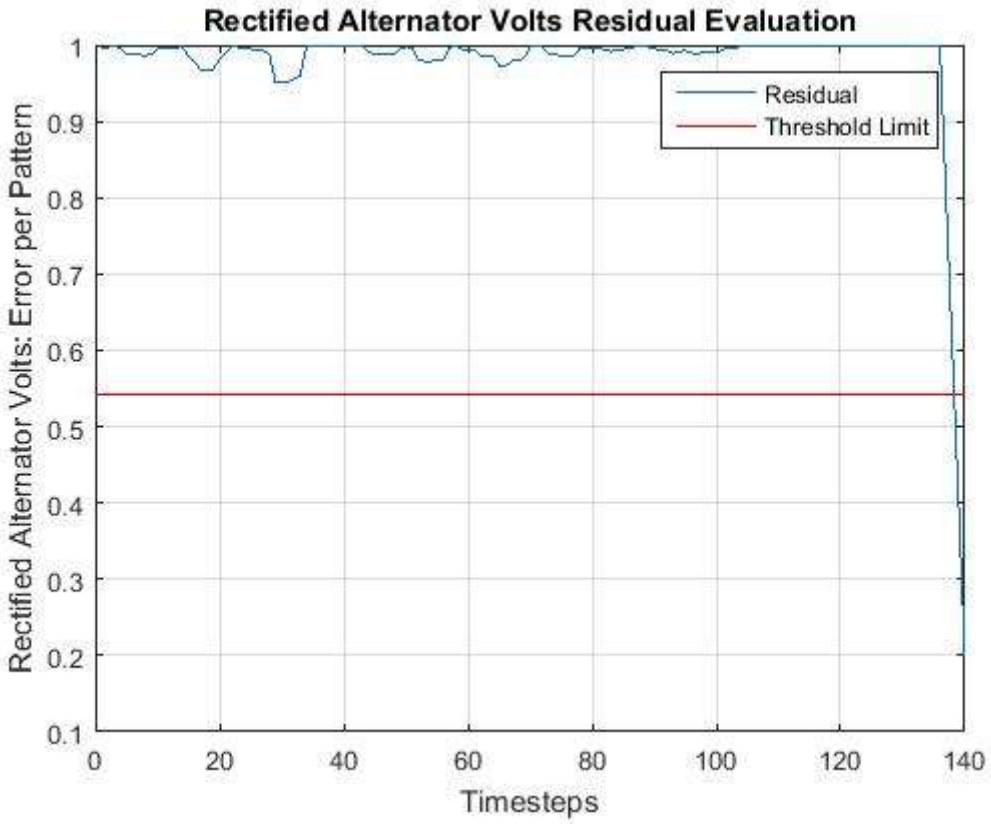


Figure L1.8.7: SCM8 Test 8: SCM8 Sensor Validation Test Result

L2.1 EXACT Test 1 Results

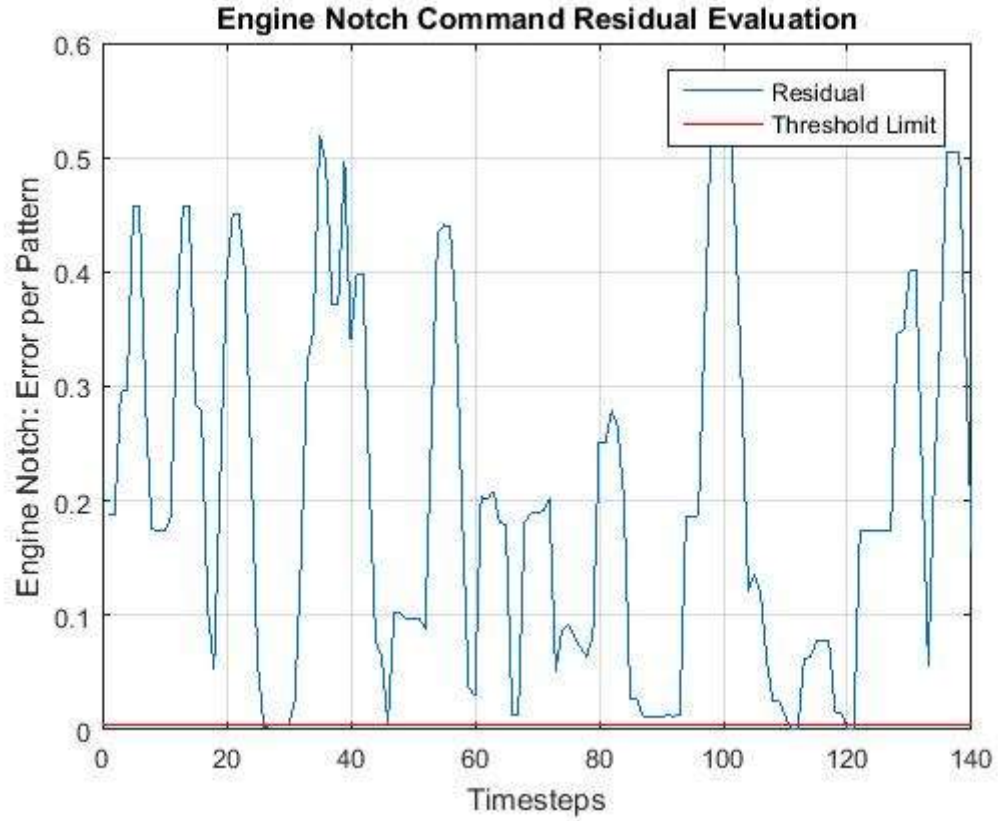


Figure L2.1.1: EXACT Test 1: Engine Notch Command Test Result

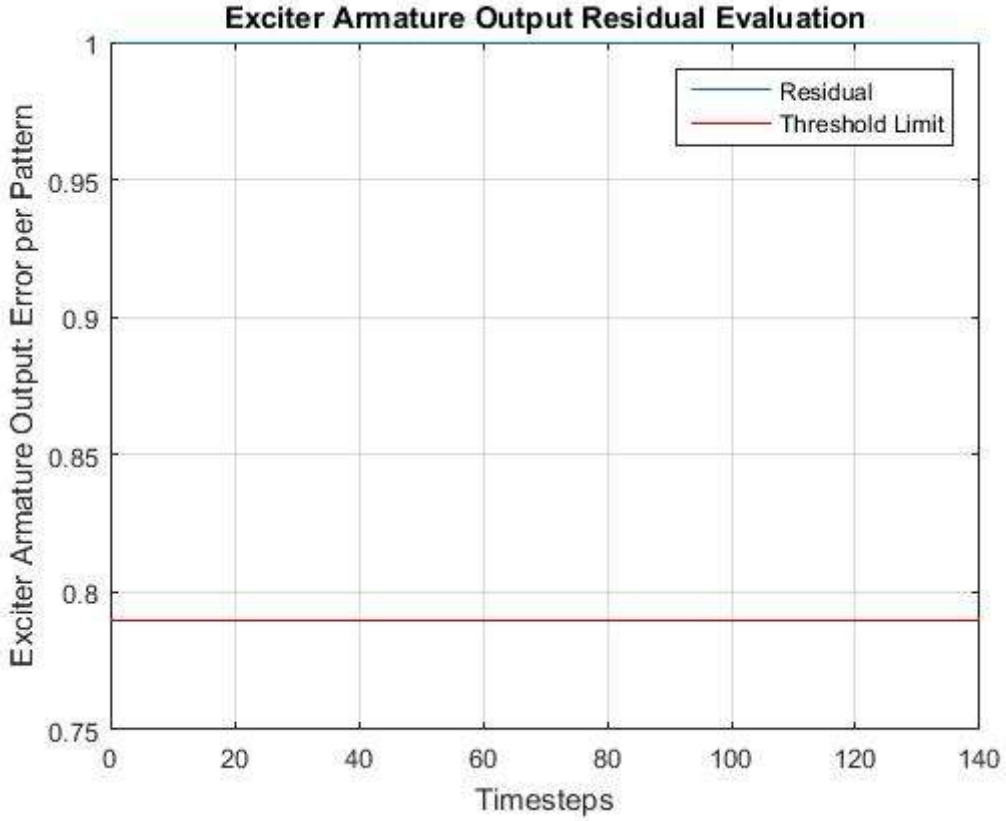


Figure L2.1.2: EXACT Test 1: EXACT Test Result

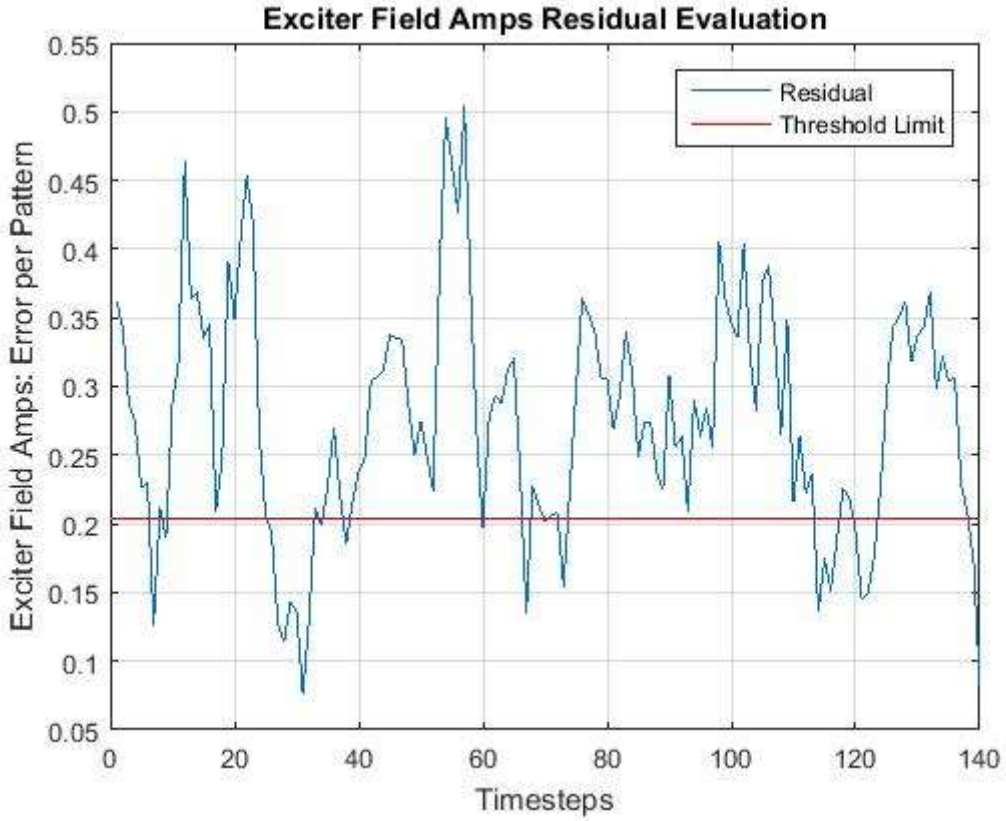


Figure L2.1.3: EXACT Test 1: EXFM Test Result

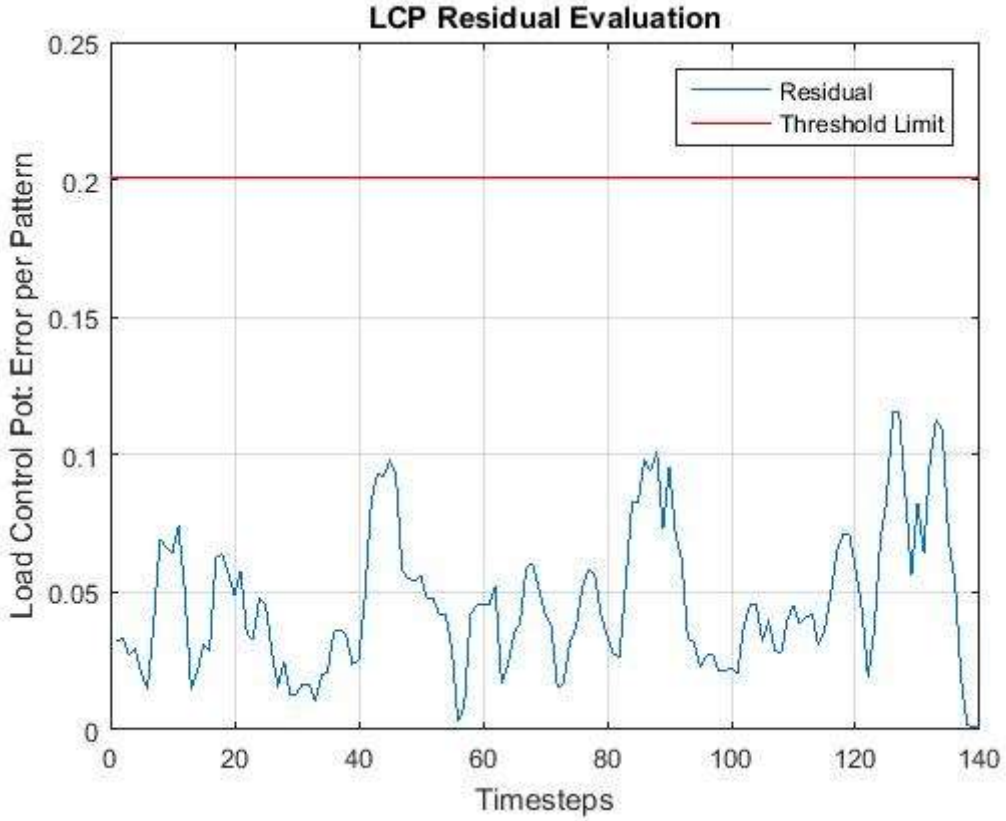


Figure L2.1.4: EXACT Test 1: LCP Test Result

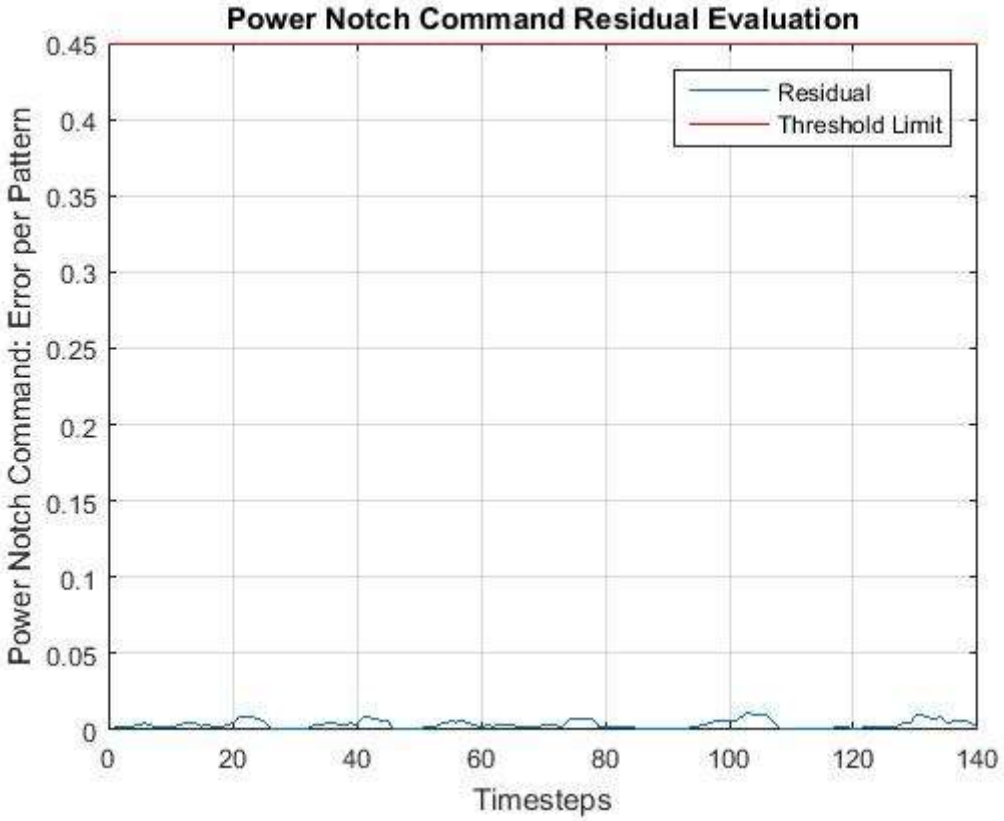


Figure L2.1.5: EXACT Test 1: Power Notch Command Test Result

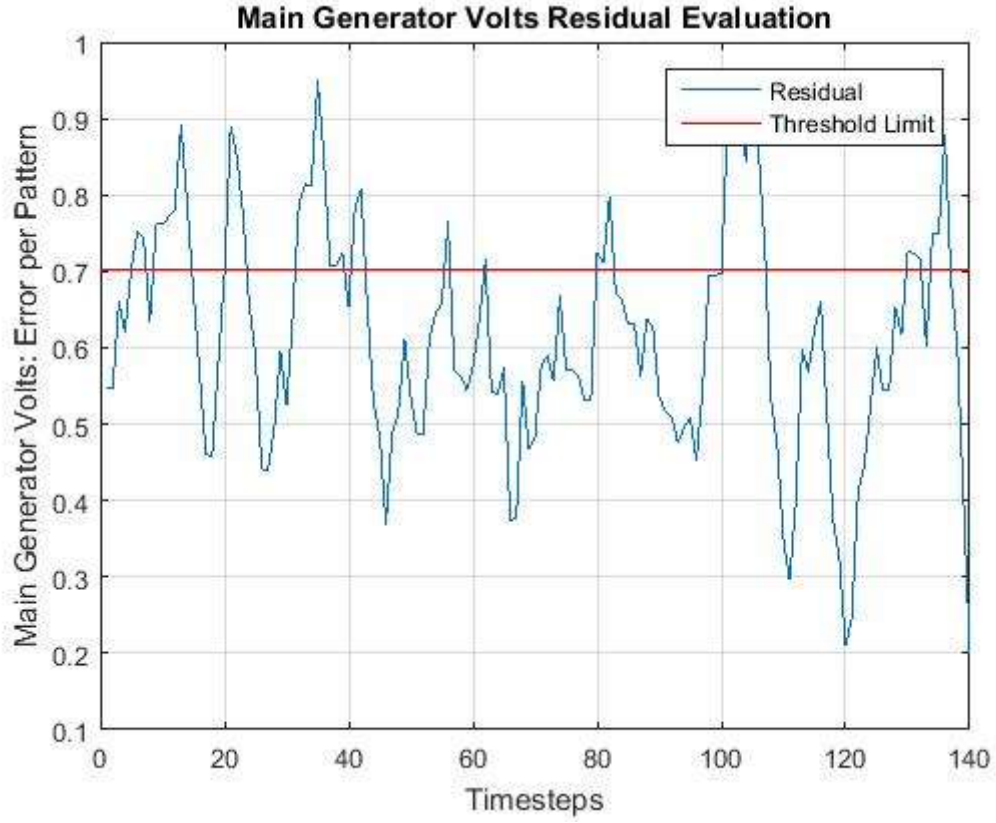


Figure L2.1.6: EXACT Test 1: SCM8 Test Result

L2.2 EXACT Test 2 Results

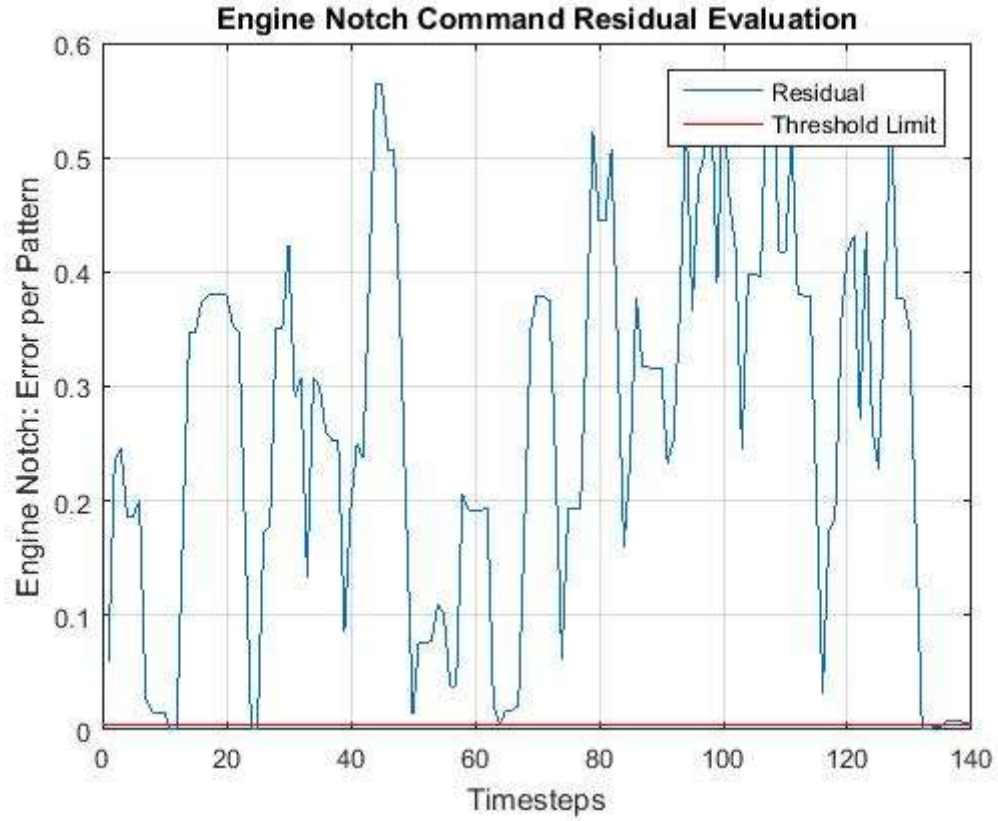


Figure L2.2.1: EXACT Test 2: Engine Notch Command Test Result

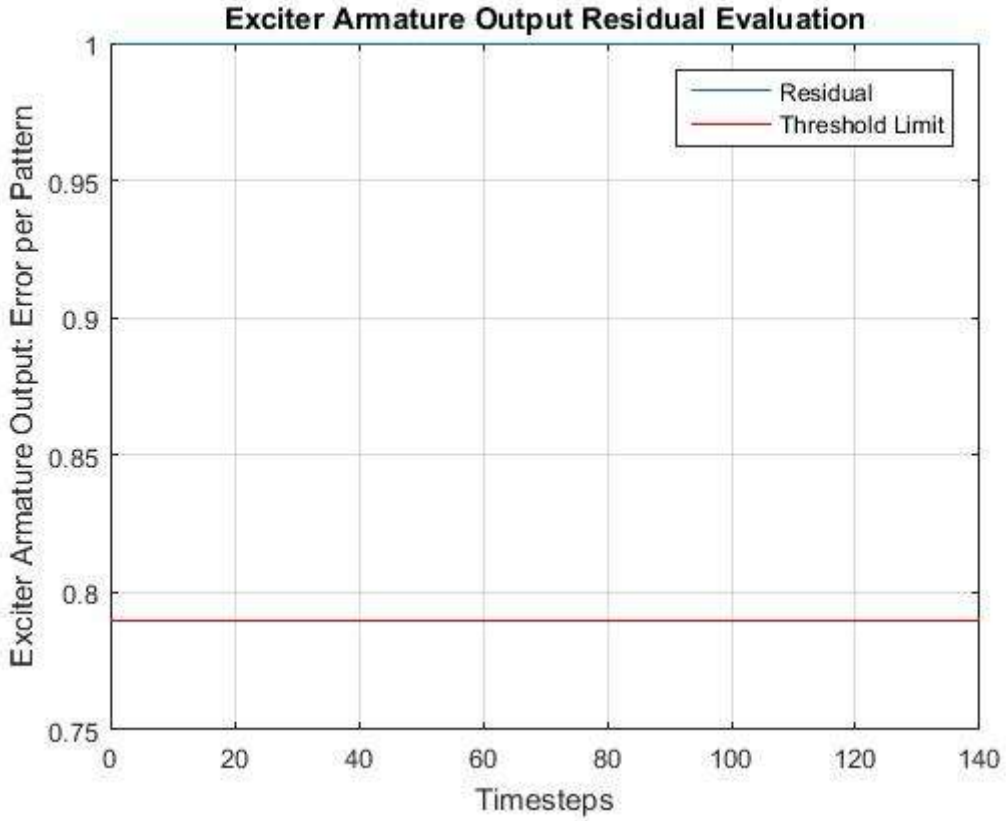


Figure L2.2.2: EXACT Test 2: EXACT Test Result

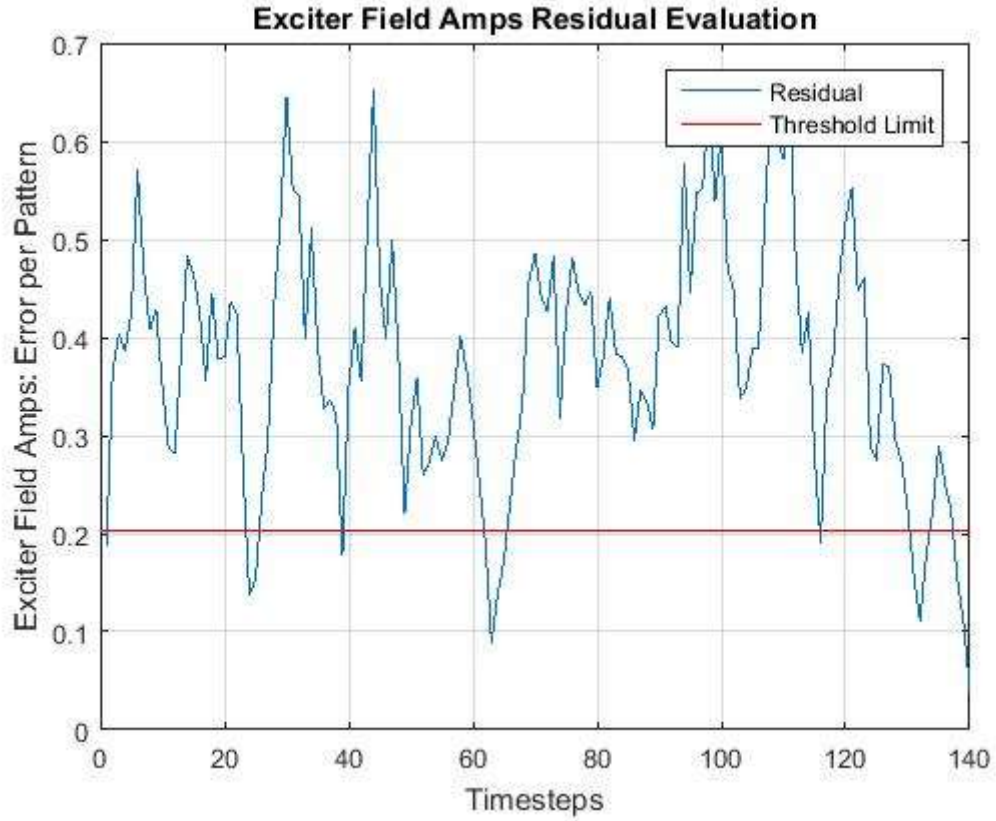


Figure L2.2.3: EXACT Test 2: EXFM Test Result

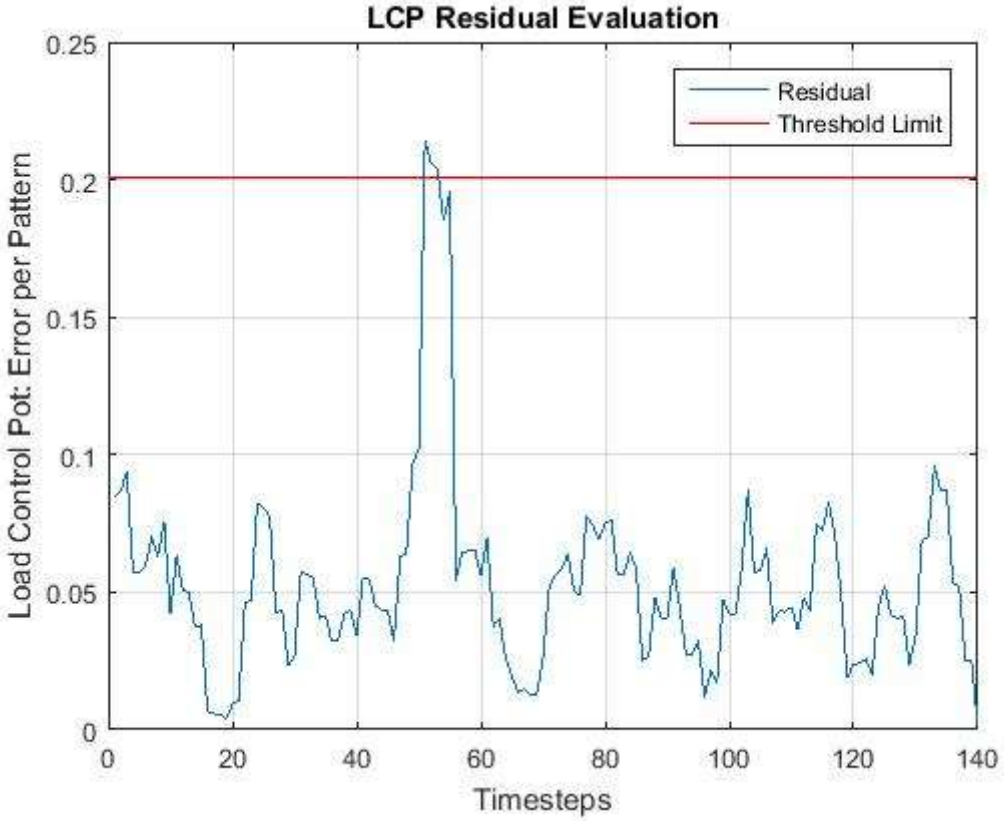


Figure L2.2.4: EXACT Test 2: LCP Test Result

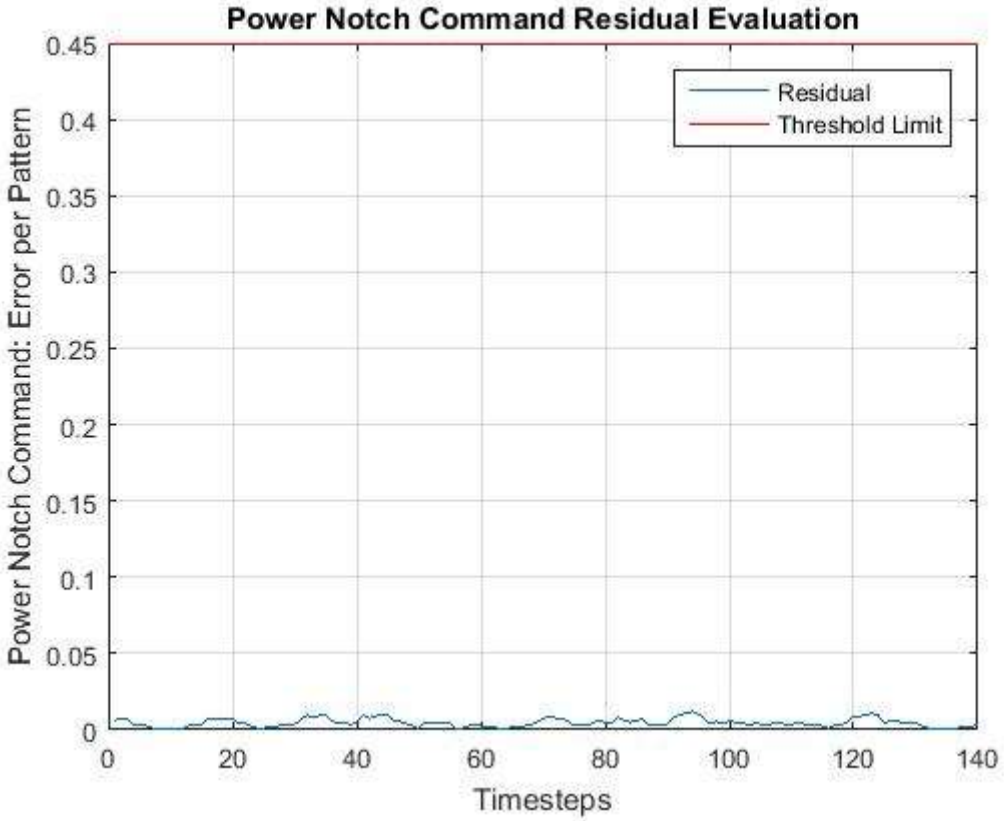


Figure L2.2.5: EXACT Test 2: Power Notch Command Test Result

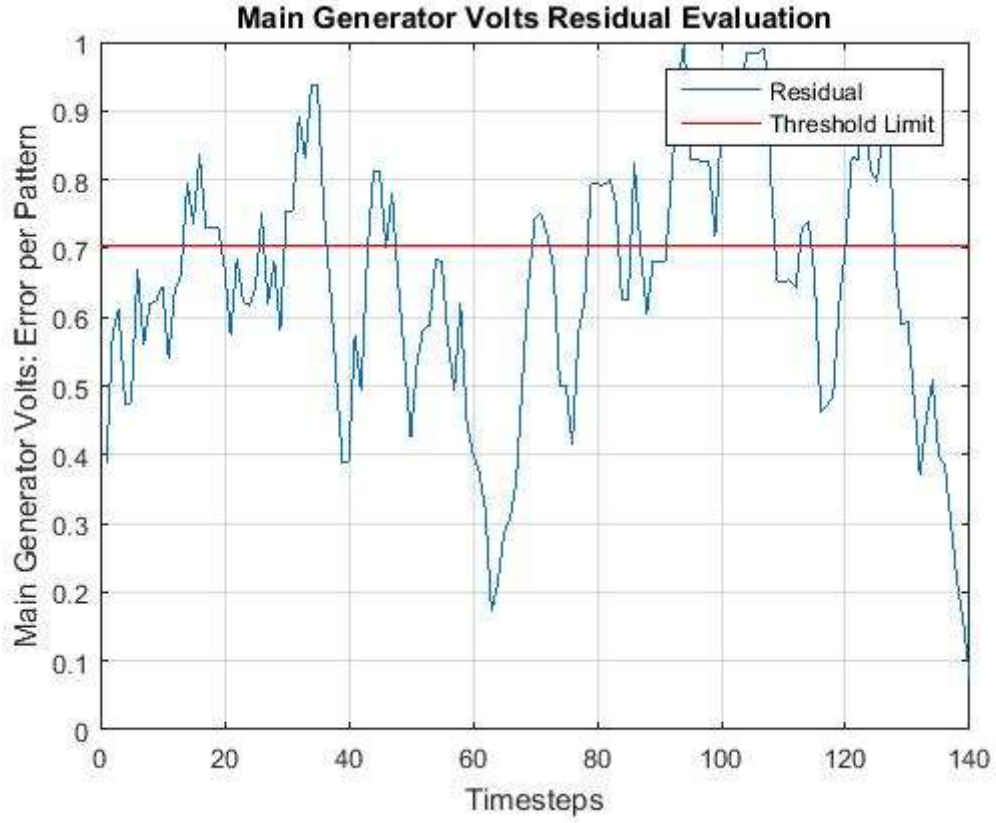


Figure L2.2.6: EXACT Test 2: SCM8 Test Result

L2.3 EXACT Test 3 Results

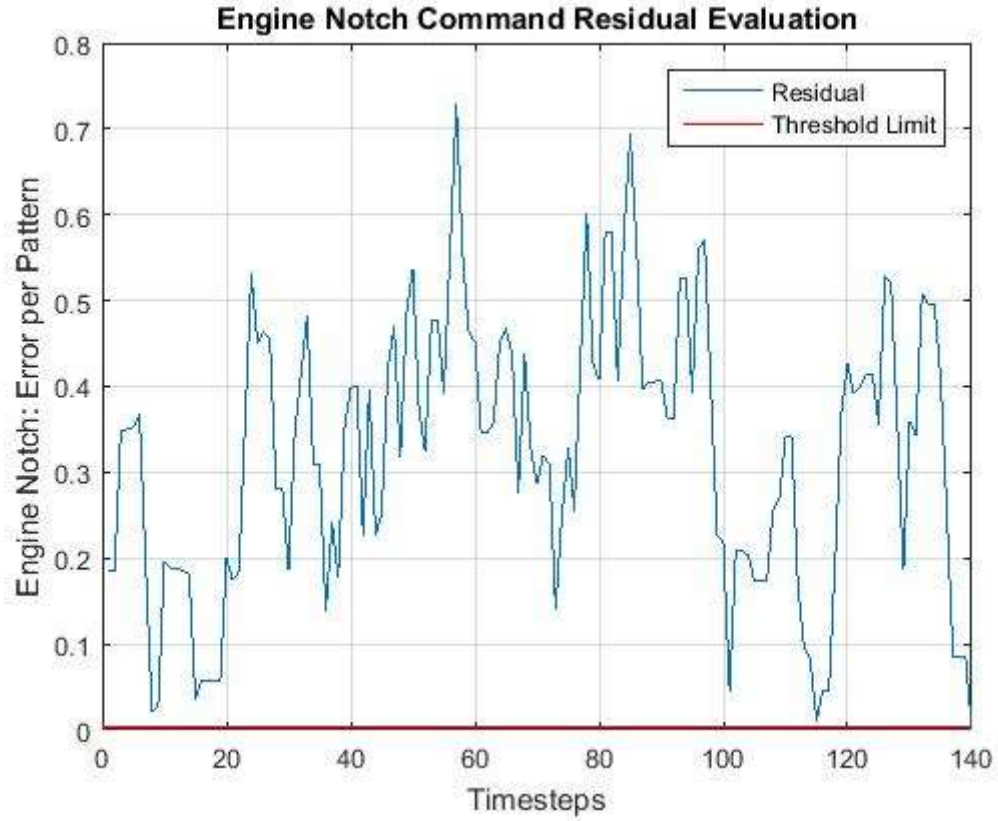


Figure L2.3.1: EXACT Test 3: Engine Notch Command Test Result

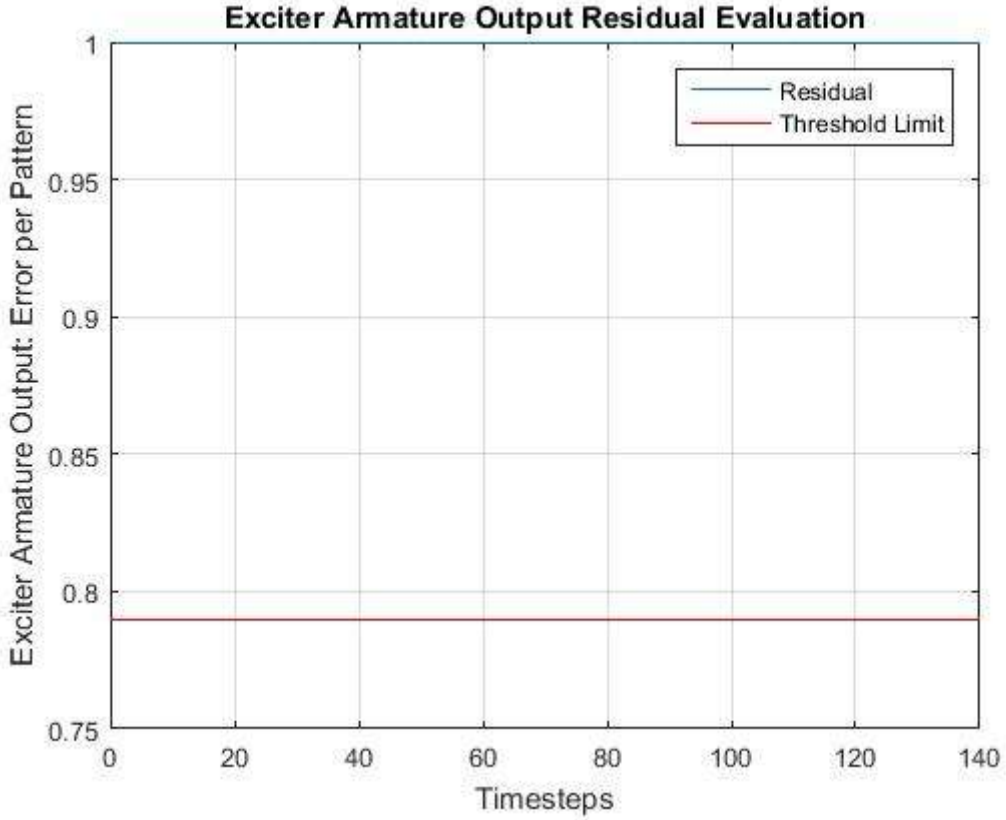


Figure L2.3.2: EXACT Test 3: EXACT Test Result

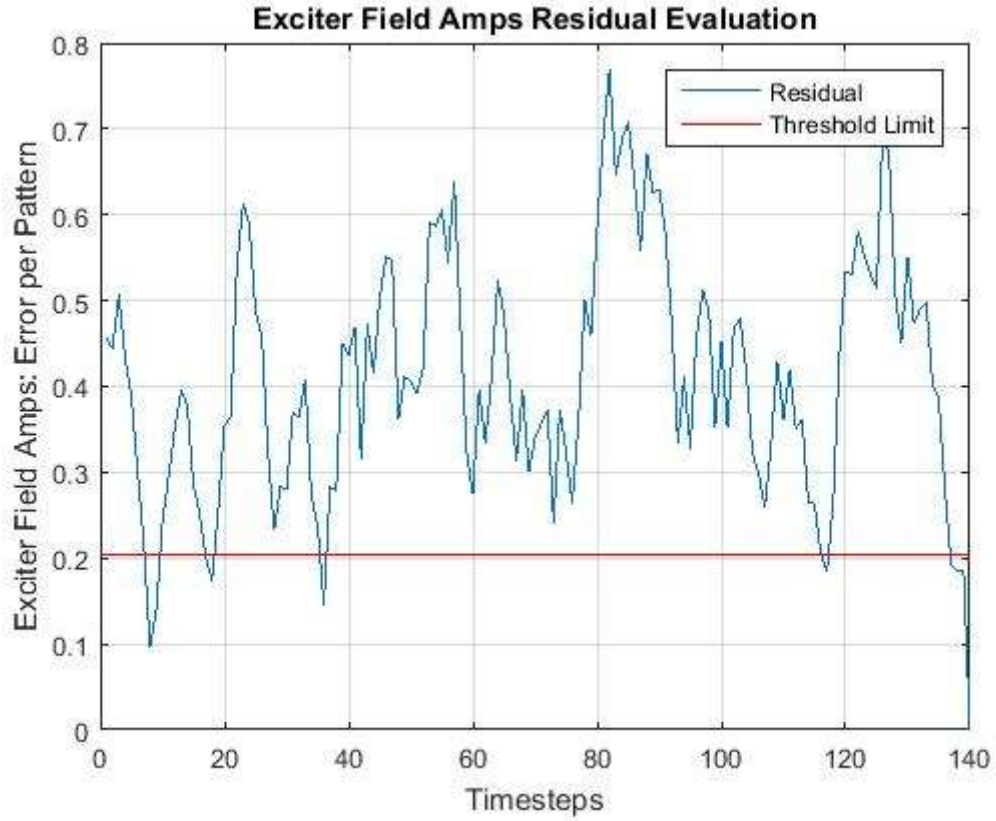


Figure L2.3.3: EXACT Test 3: EXFM Test Result

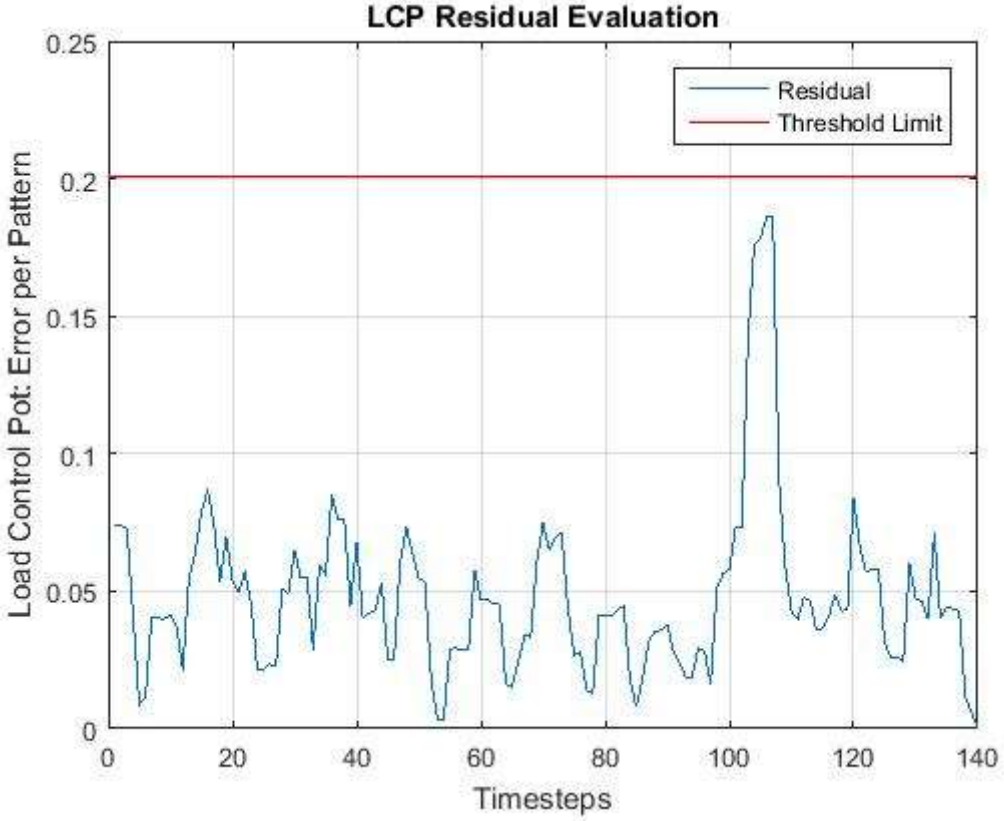


Figure L2.3.4: EXACT Test 3: LCP Test Result

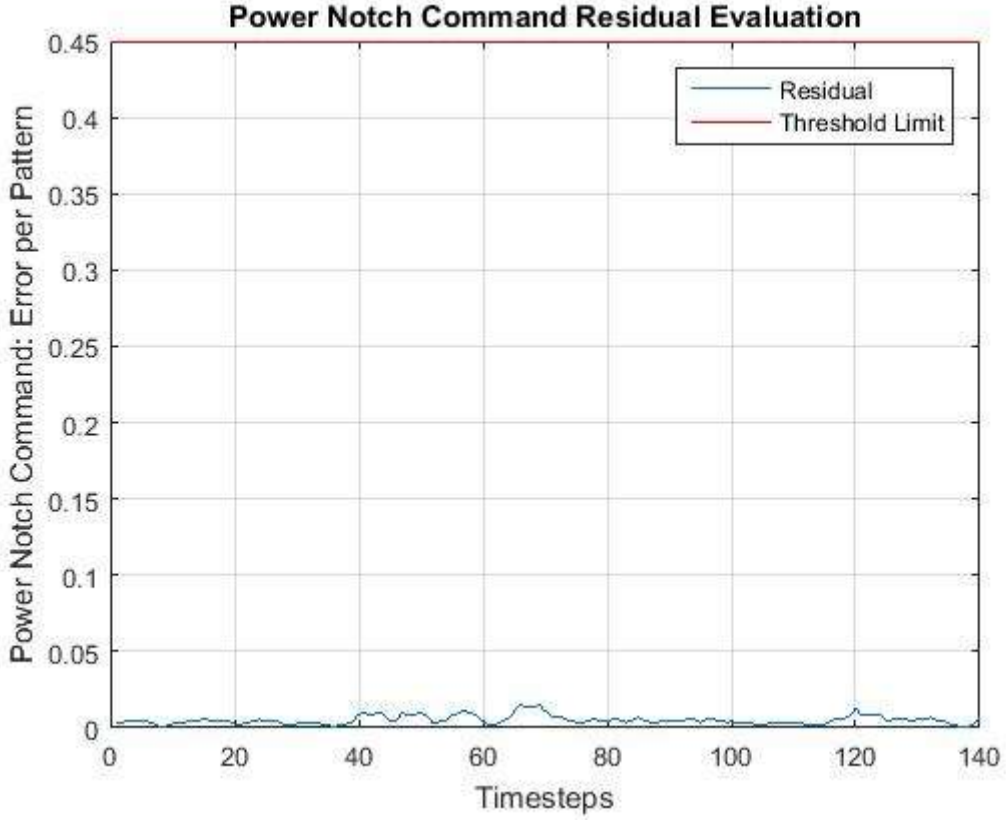


Figure L2.3.5: EXACT Test 3: Power Notch Command Test Result

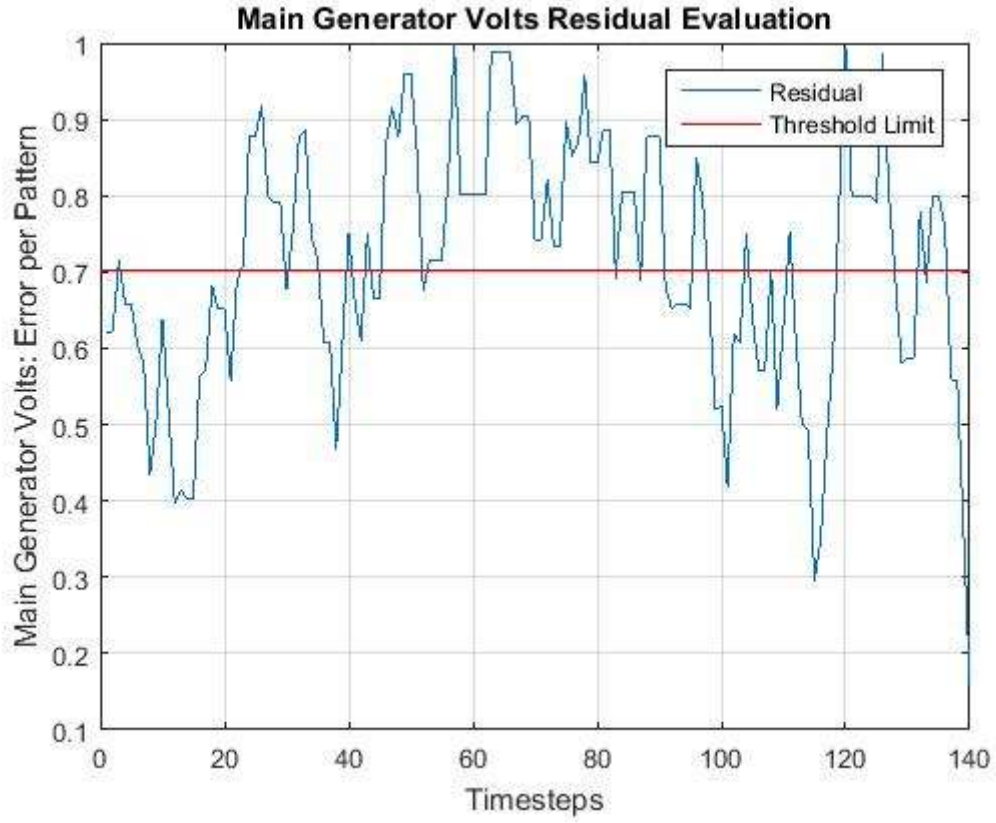


Figure L2.3.6: EXACT Test 3: SCM8 Test Result

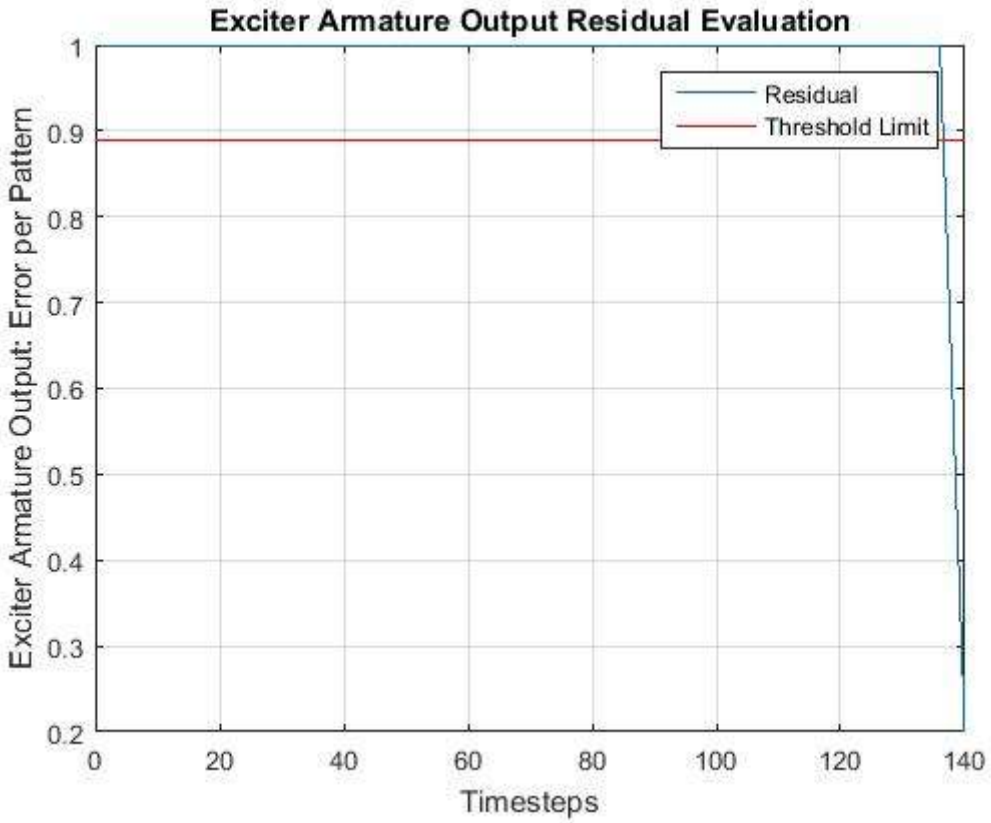


Figure L2.3.7: EXACT Test 3: EXACT Sensor Validation Test Result

L2.4 EXACT Test 4 Results

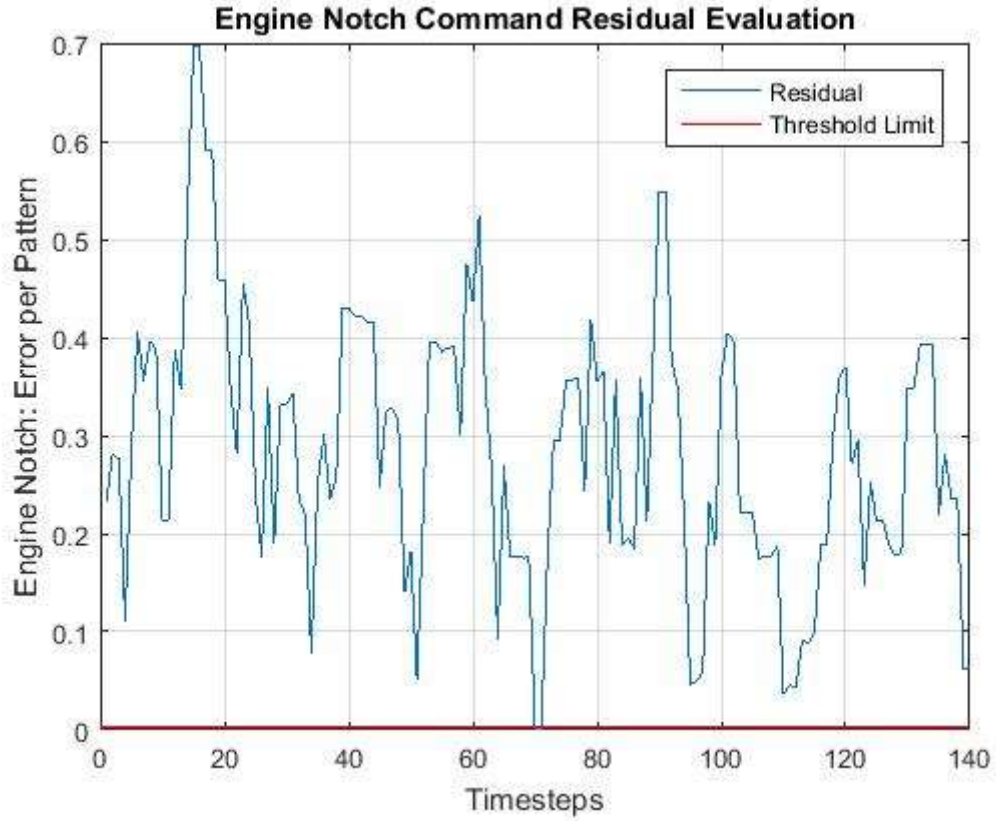


Figure L2.4.1: EXACT Test 4: Engine Notch Command Test Result

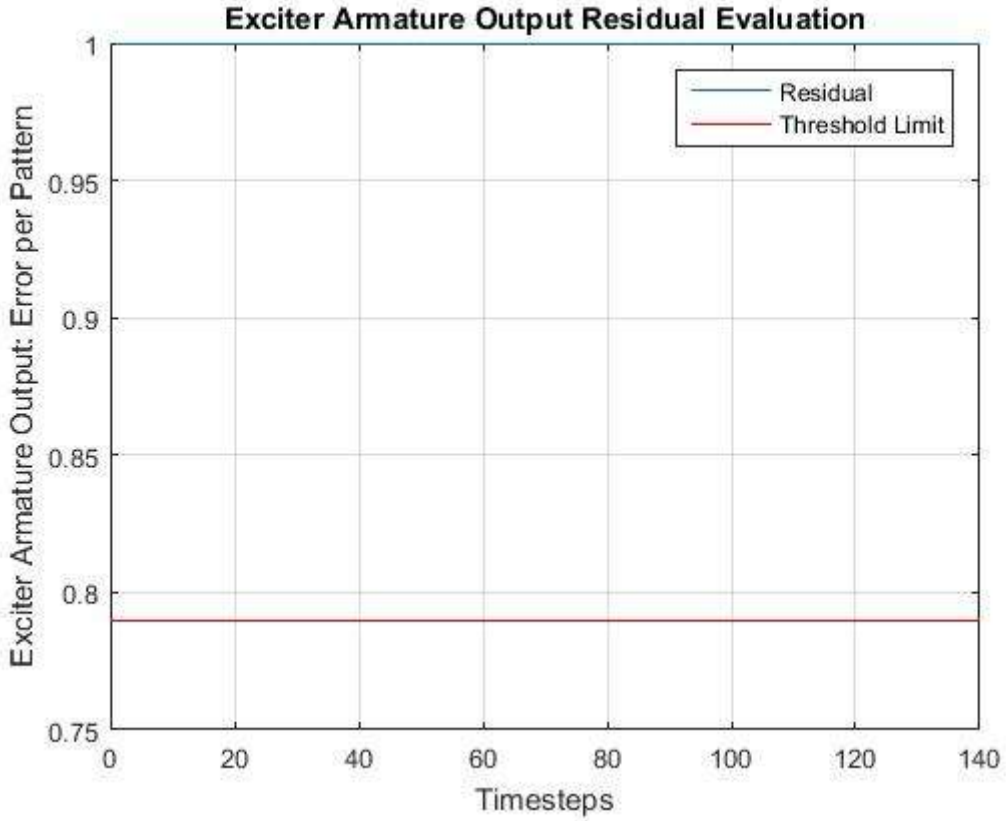


Figure L2.4.2: EXACT Test 4: EXACT Test Result

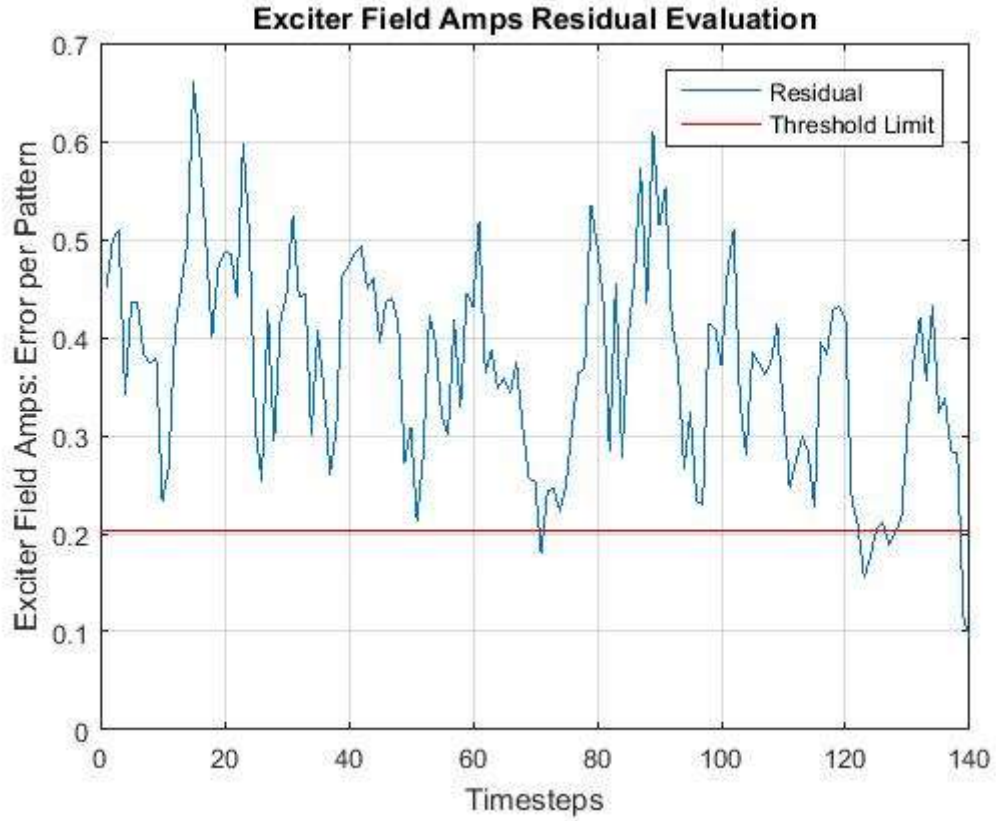


Figure L2.4.3: EXACT Test 4: EXFM Test Result

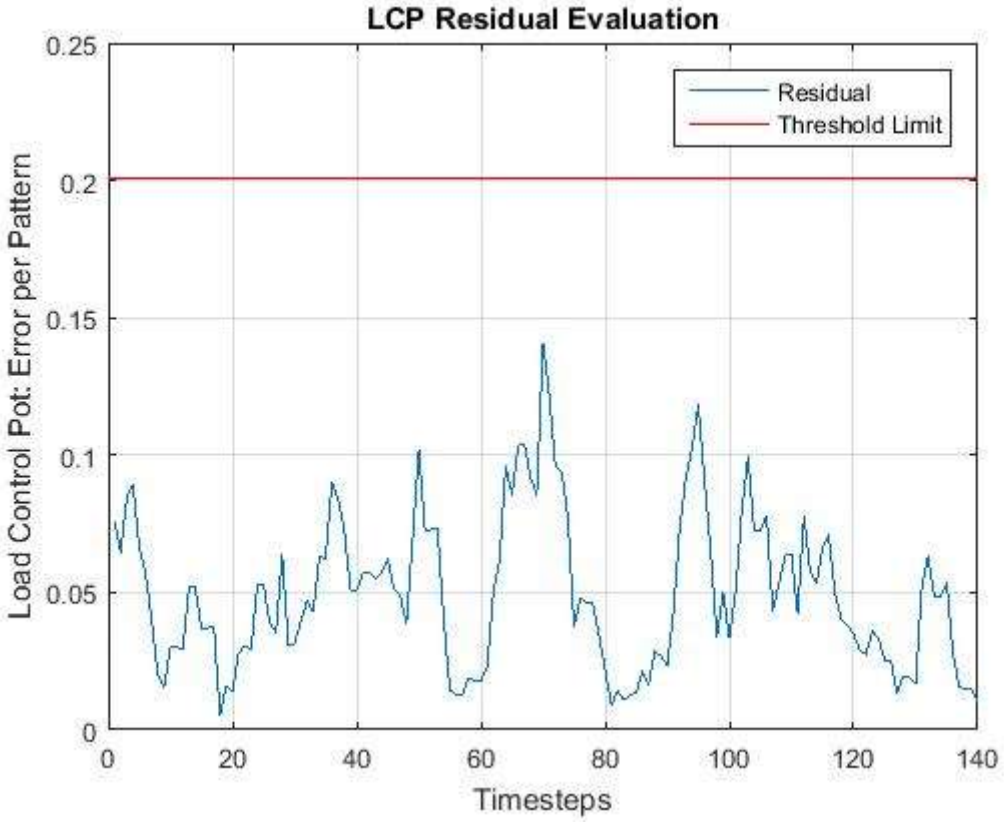


Figure L2.4.4: EXACT Test 4: LCP Test Result

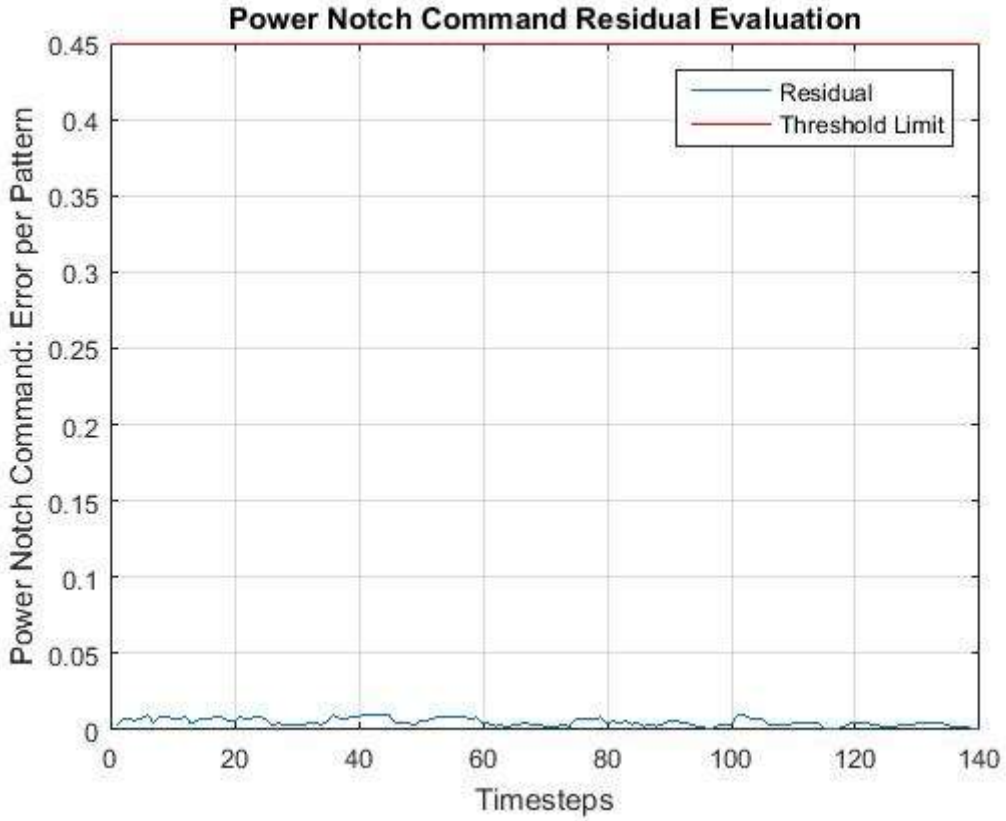


Figure L2.4.5: EXACT Test 4: Power Notch Command Test Result

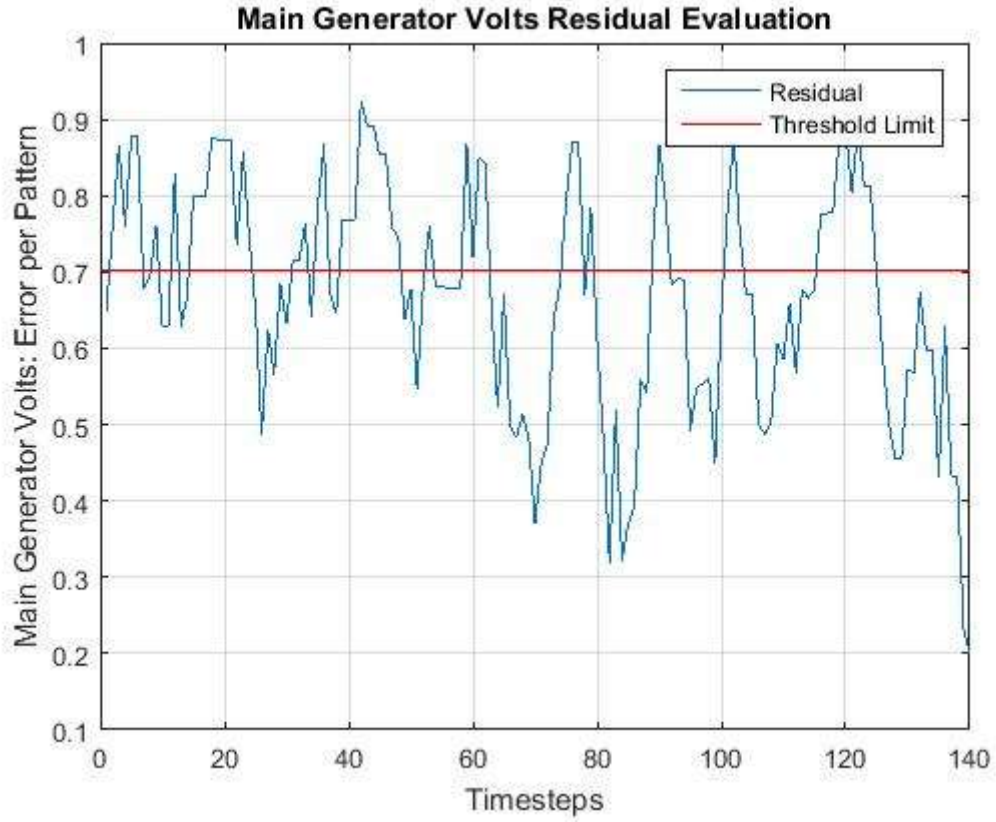


Figure L2.4.6: EXACT Test 4: SCM8 Test Result

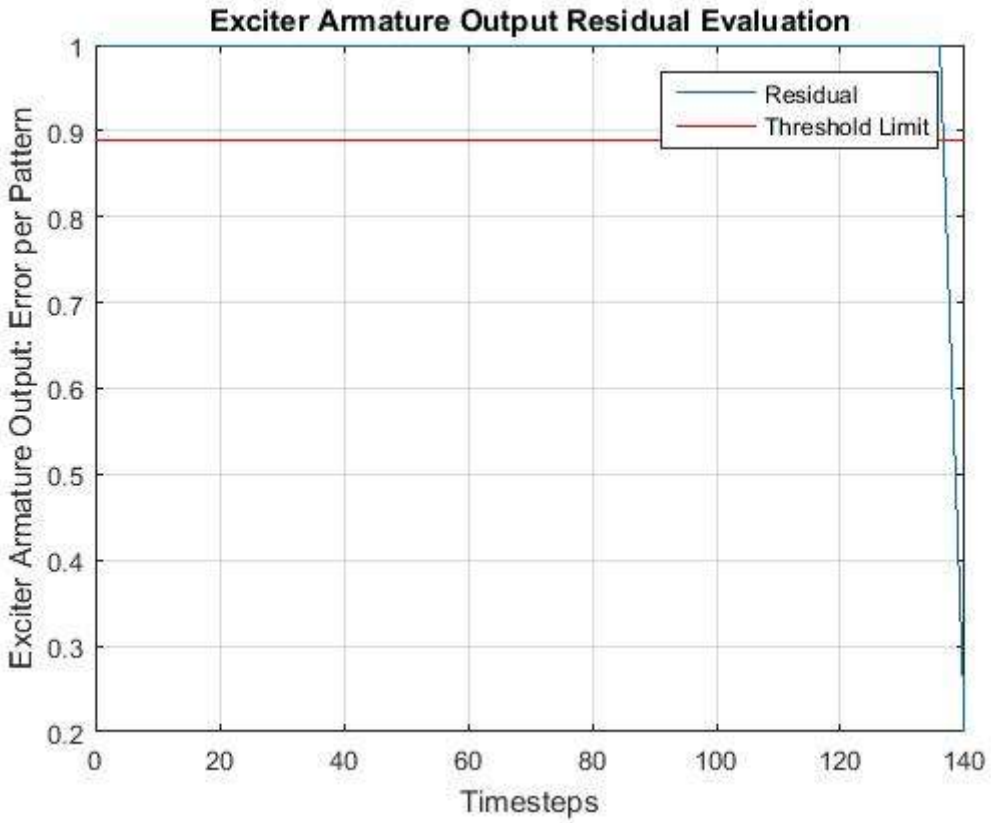


Figure L2.4.7: EXACT Test 4: EXACT Sensor Validation Test Result

L2.5 EXACT Test 5 Results

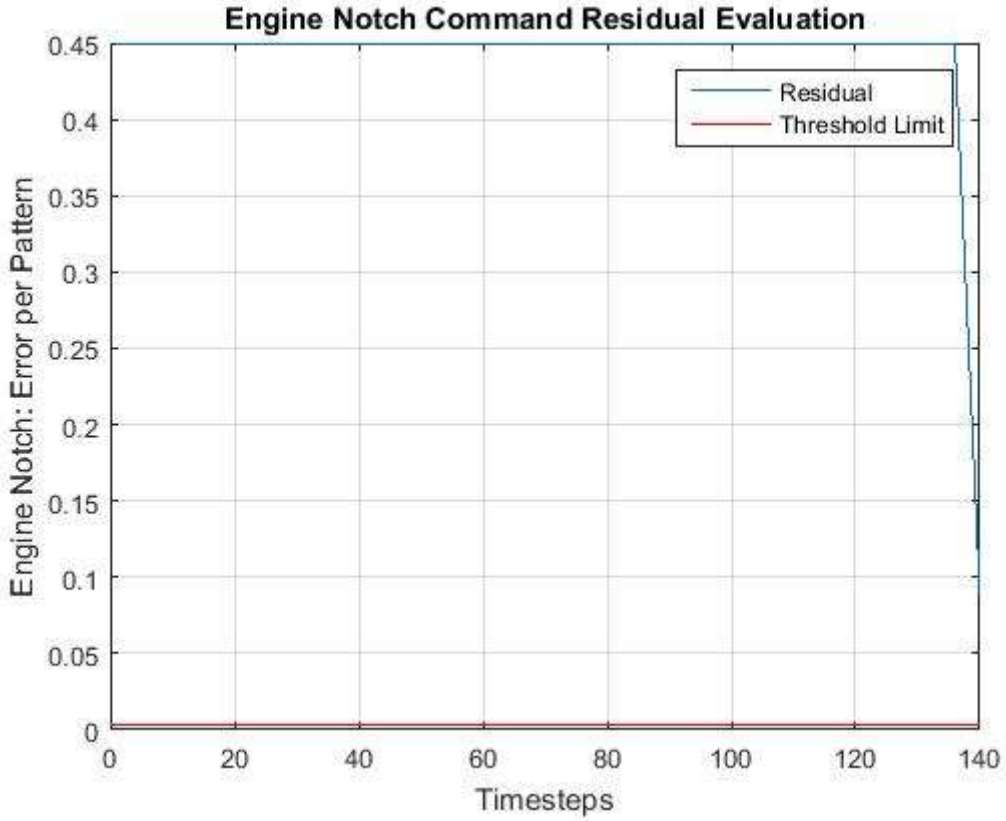


Figure L2.5.1: EXACT Test 5: Engine Notch Command Test Result

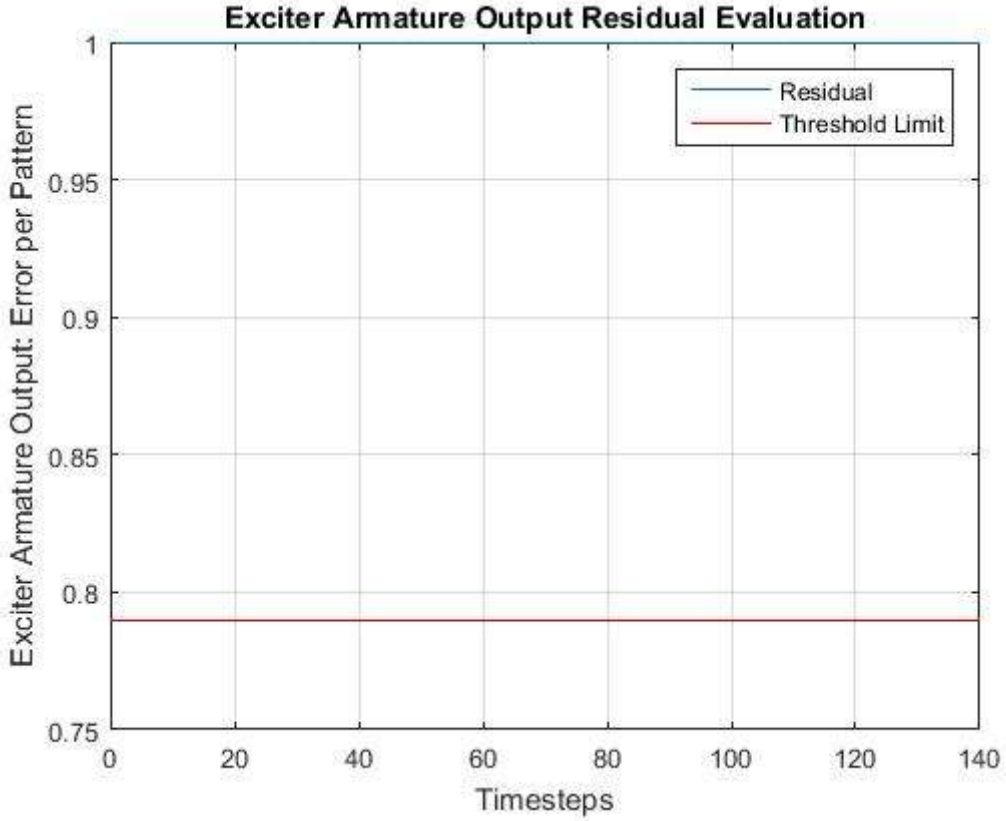


Figure L2.5.2: EXACT Test 5: EXACT Test Result

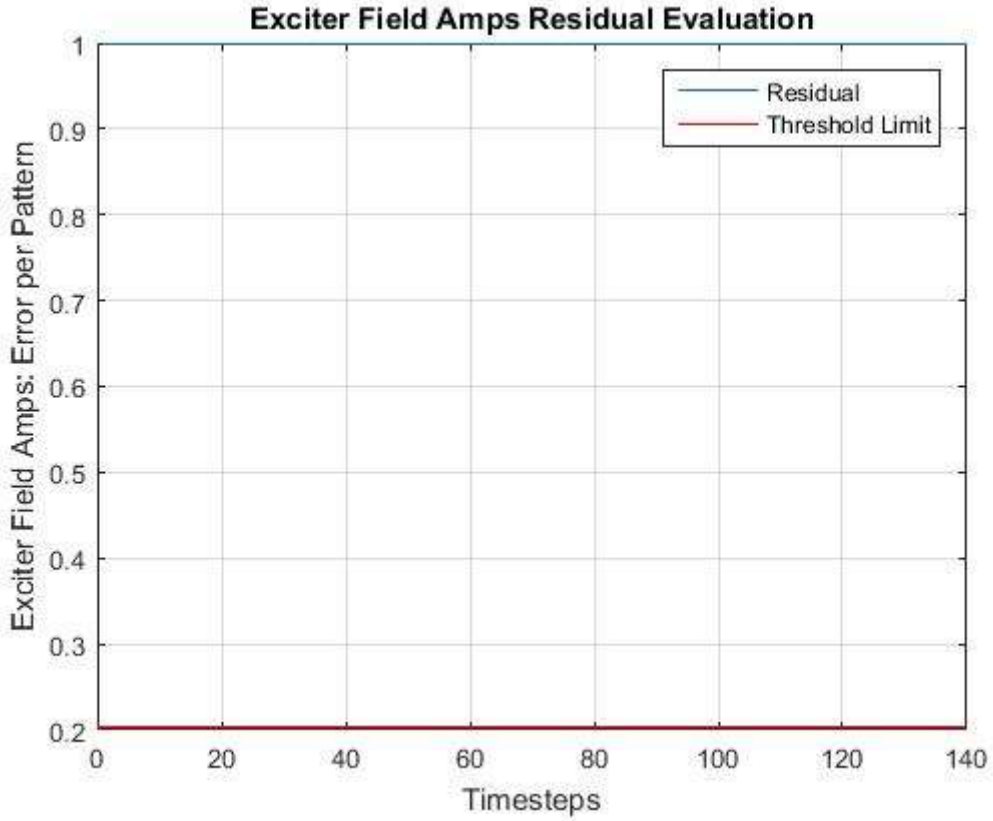


Figure L2.5.3: EXACT Test 5: EXFM Test Result

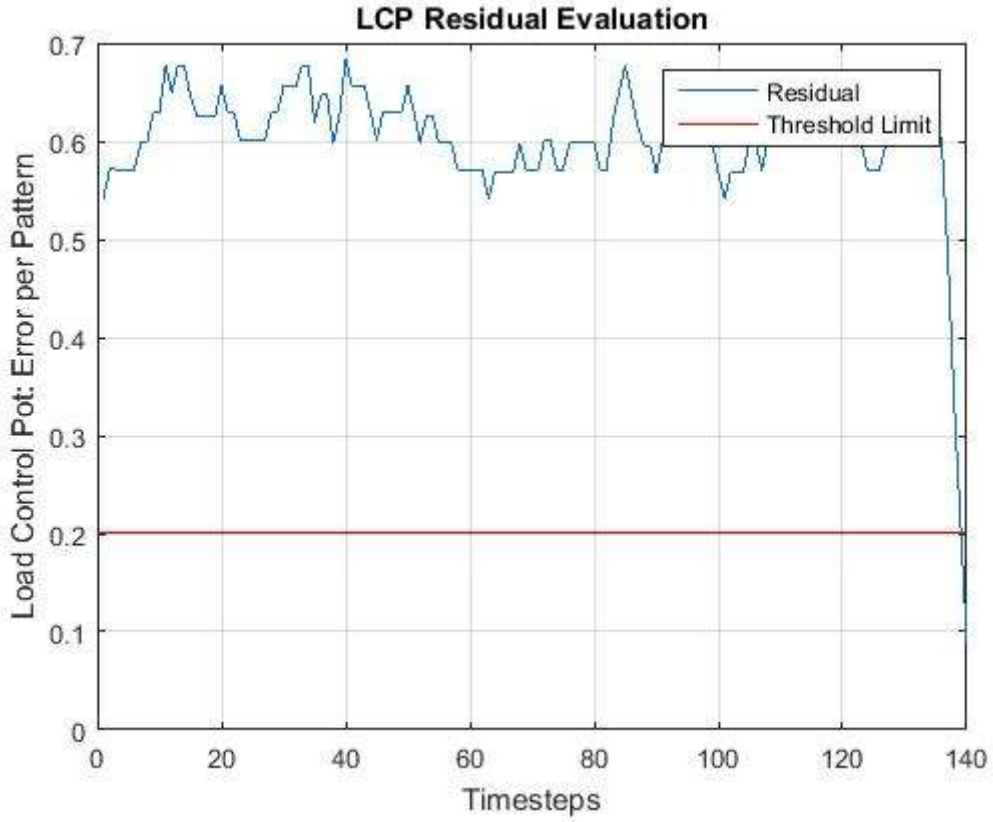


Figure L2.5.4: EXACT Test 5: LCP Test Result

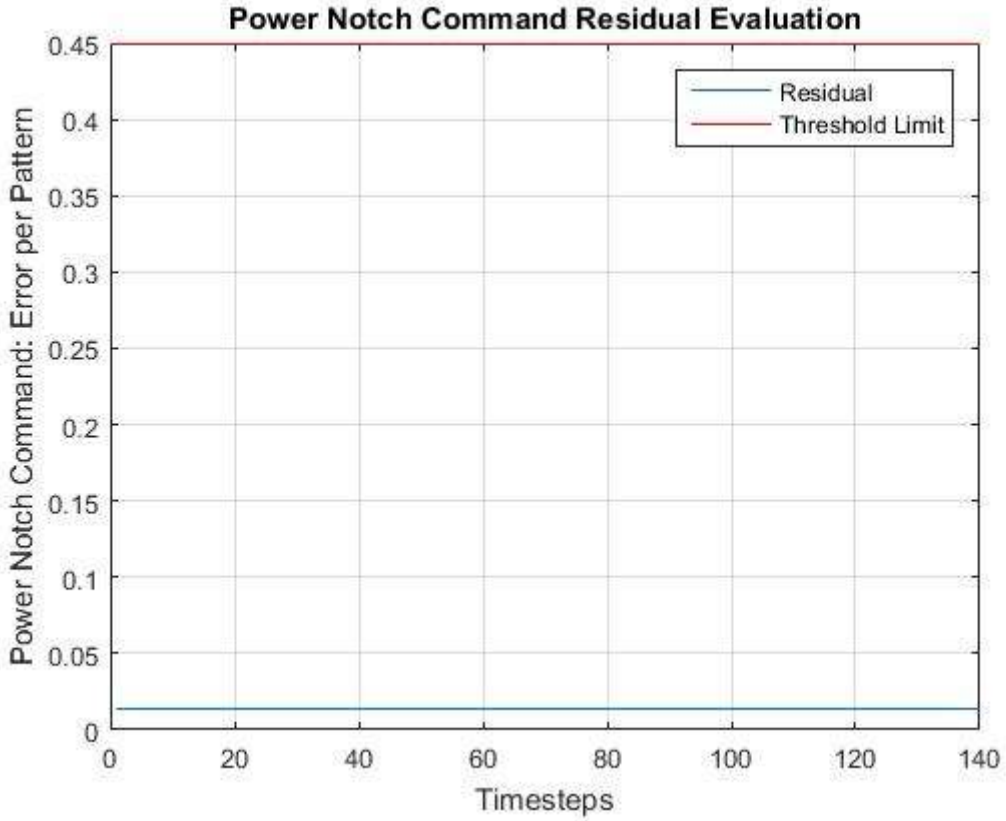


Figure L2.5.5: EXACT Test 5: Power Notch Command Test Result

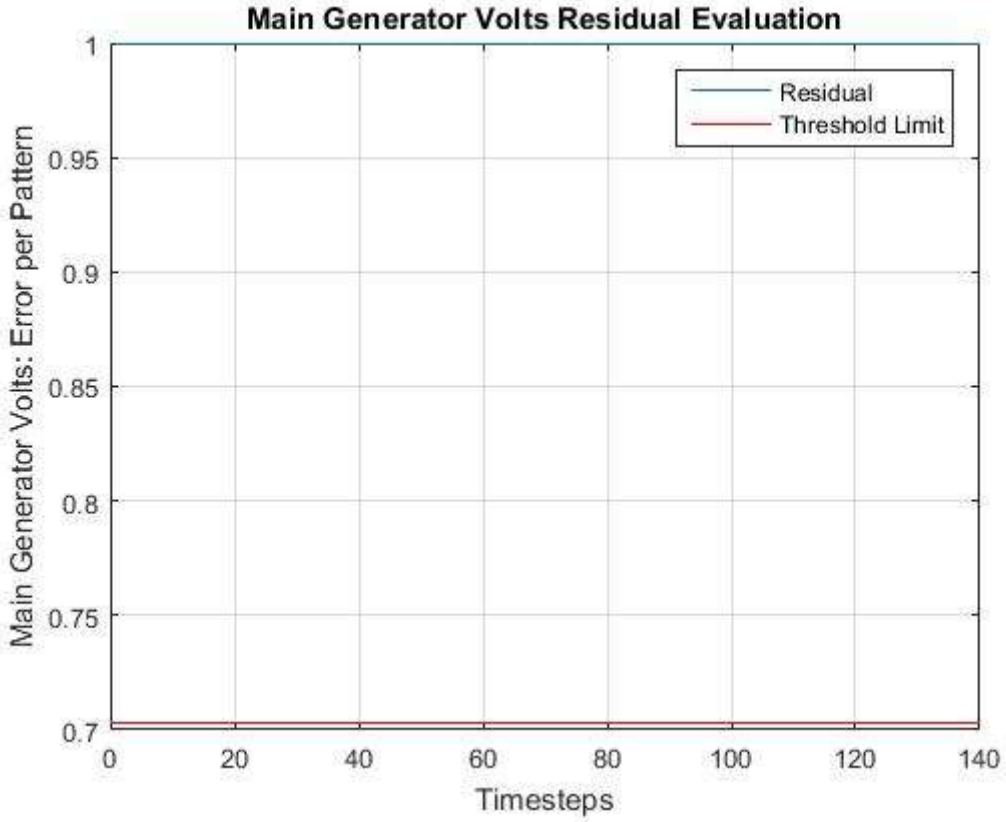


Figure L2.5.6: EXACT Test 5: SCM8 Test Result

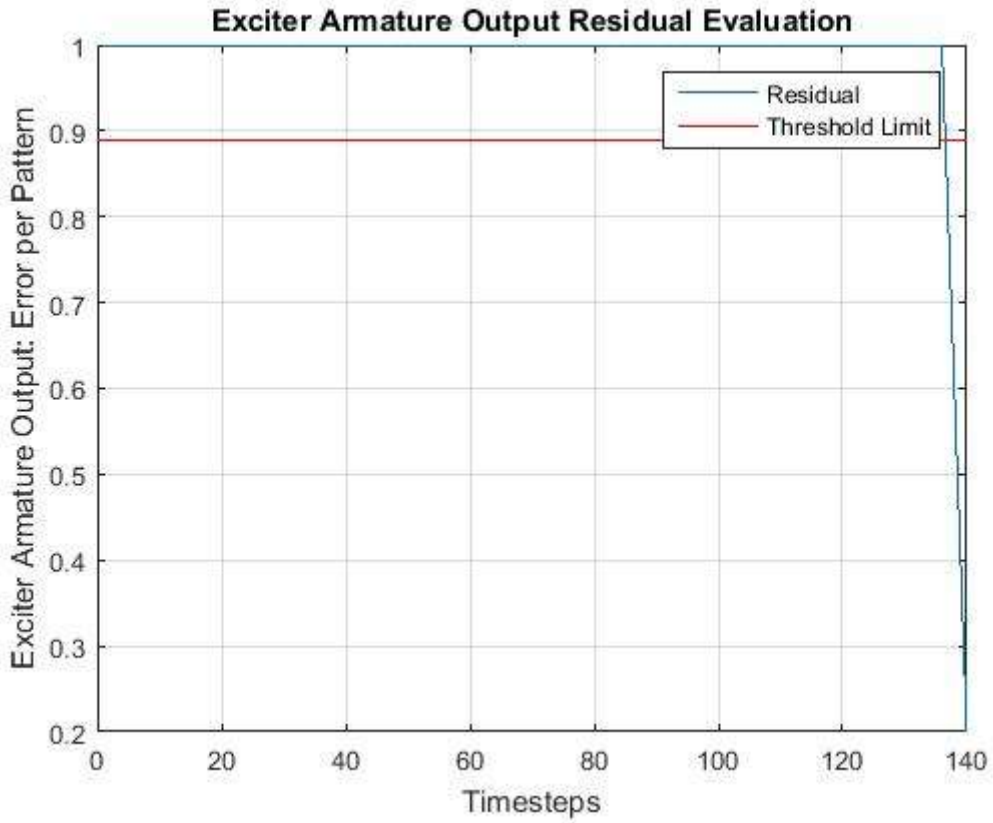


Figure L2.5.7: EXACT Test 5: EXACT Sensor Validation Test Result

L2.6 EXACT Test 6 Results

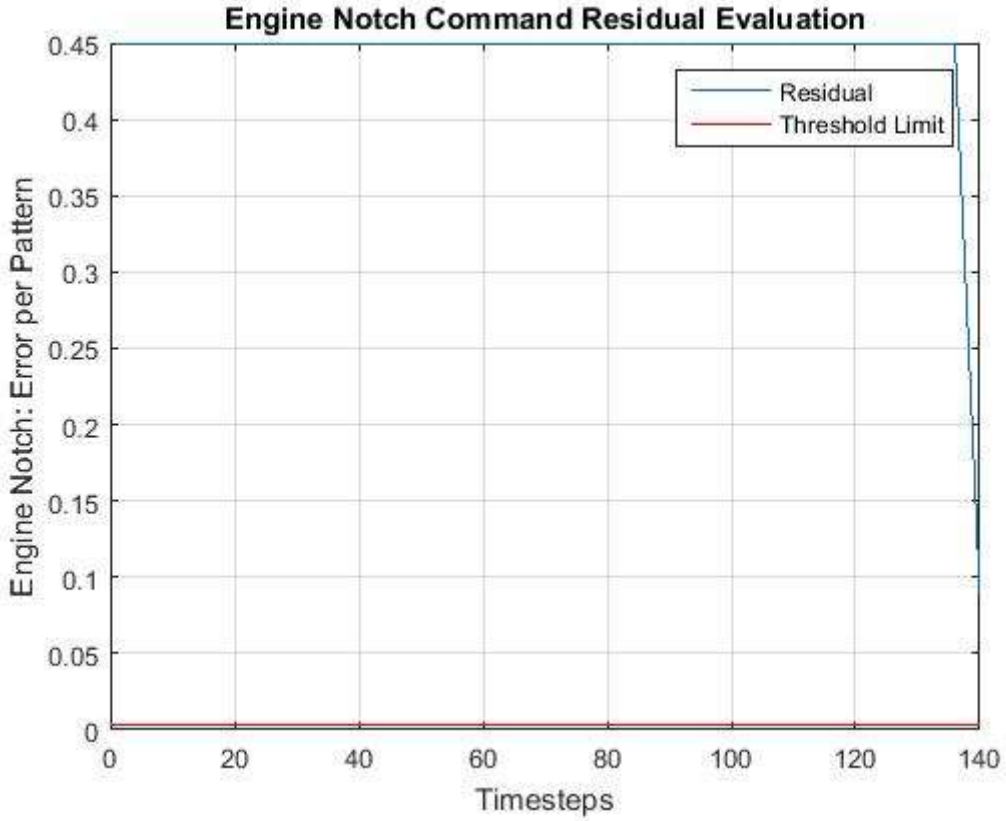


Figure L2.6.1: EXACT Test 6: Engine Notch Command Test Result

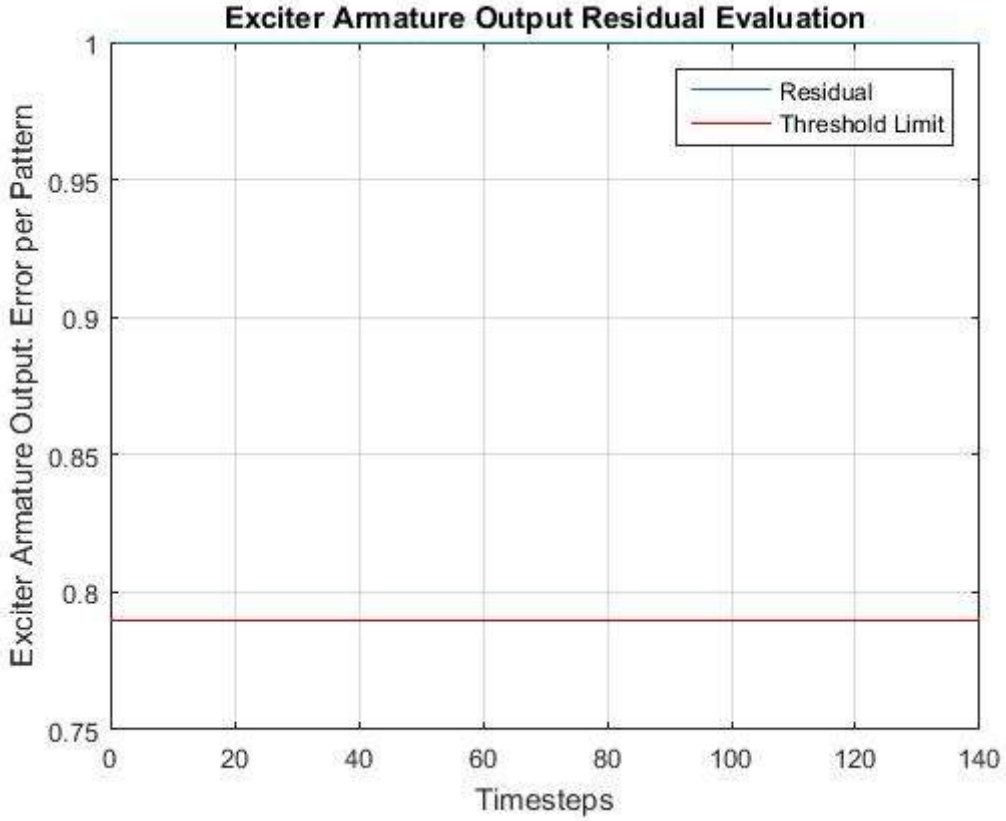


Figure L2.6.2: EXACT Test 6: EXACT Test Result

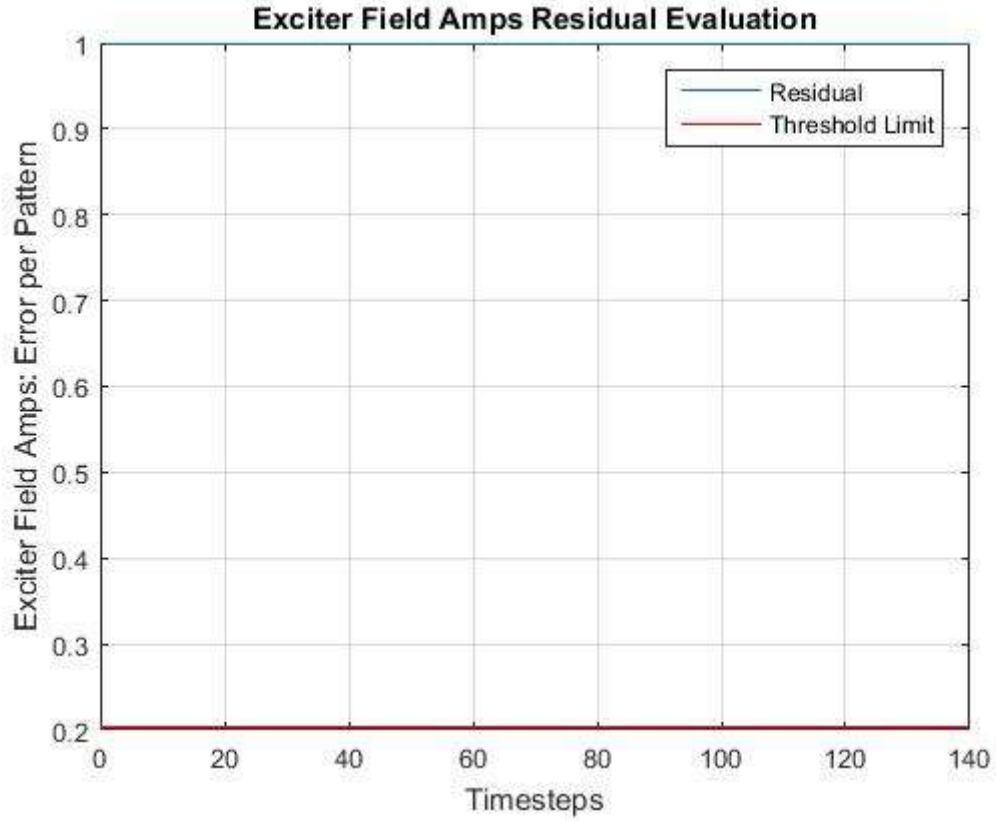


Figure L2.6.3: EXACT Test 6: EXFM Test Result

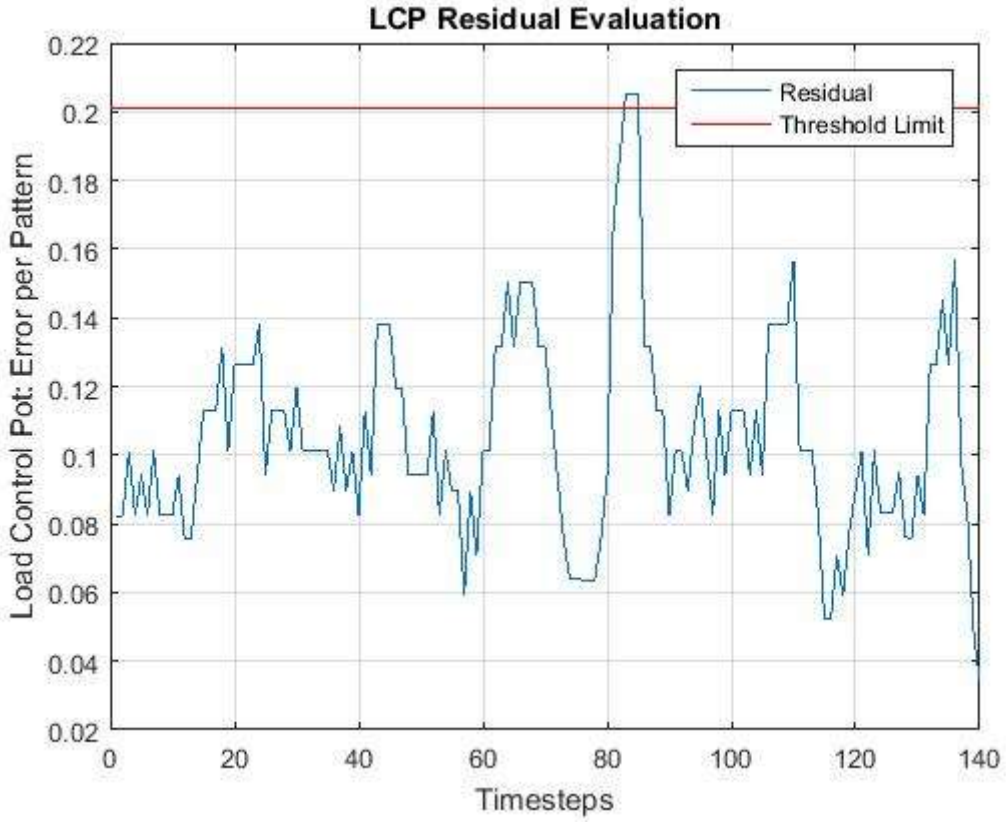


Figure L2.6.4: EXACT Test 6: LCP Test Result

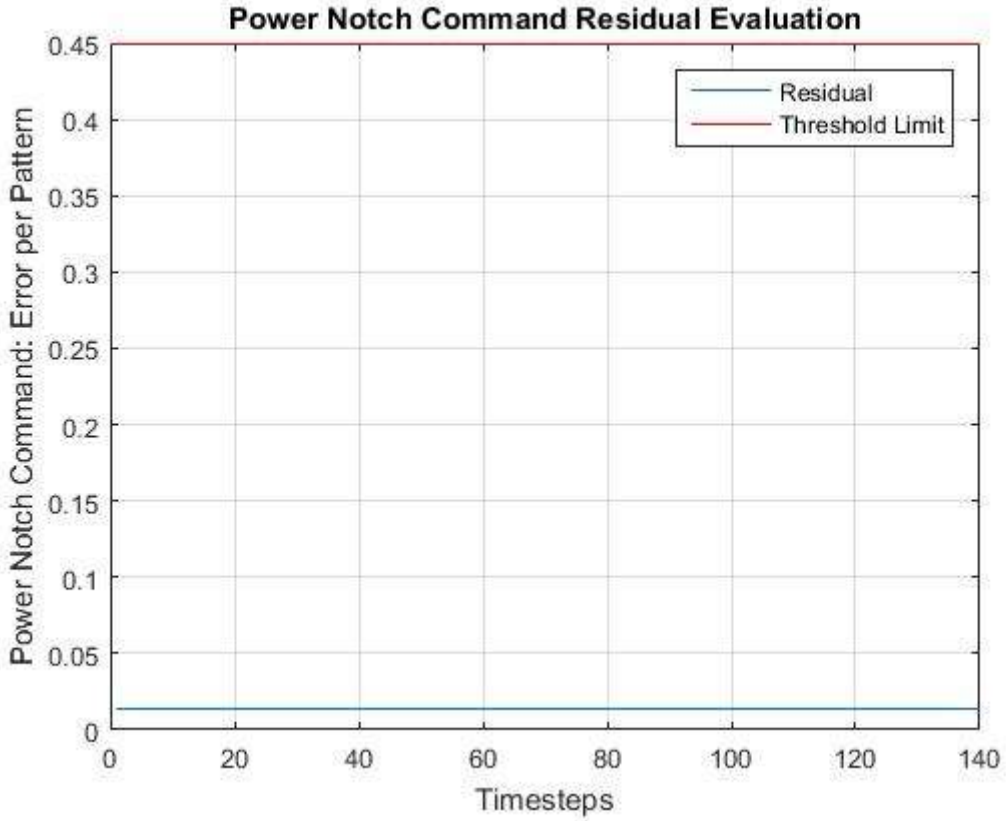


Figure L2.6.5: EXACT Test 6: Power Notch Command Test Result

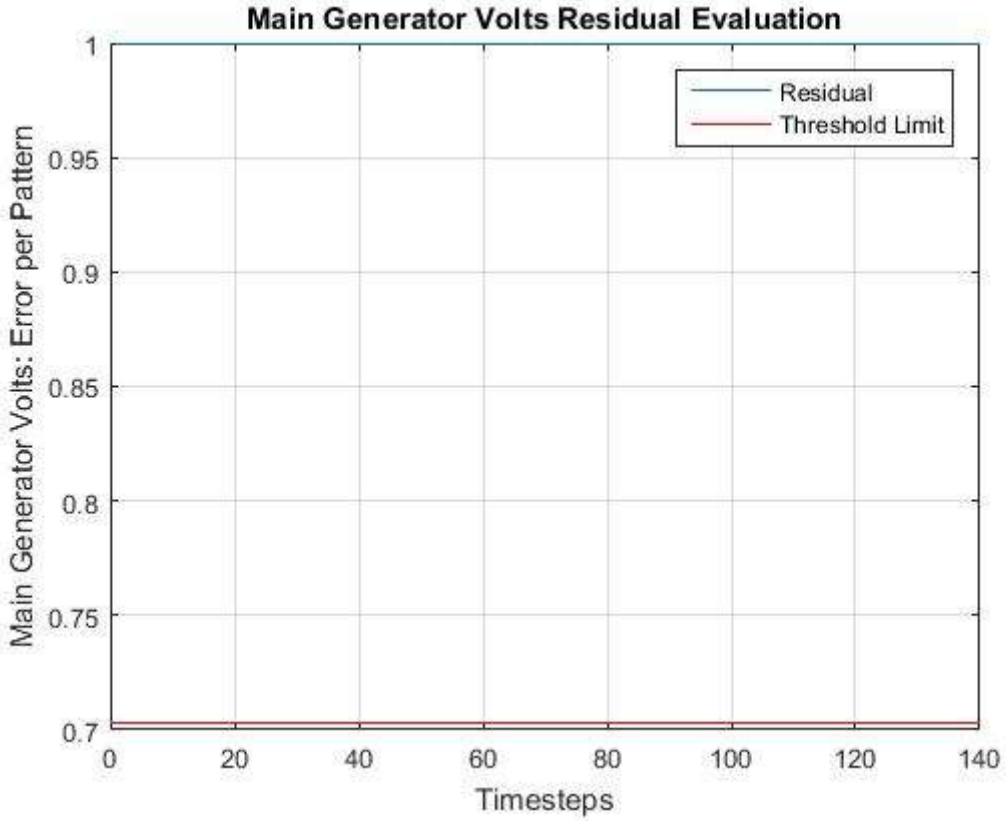


Figure L2.6.6: EXACT Test 6: SCM8 Test Result

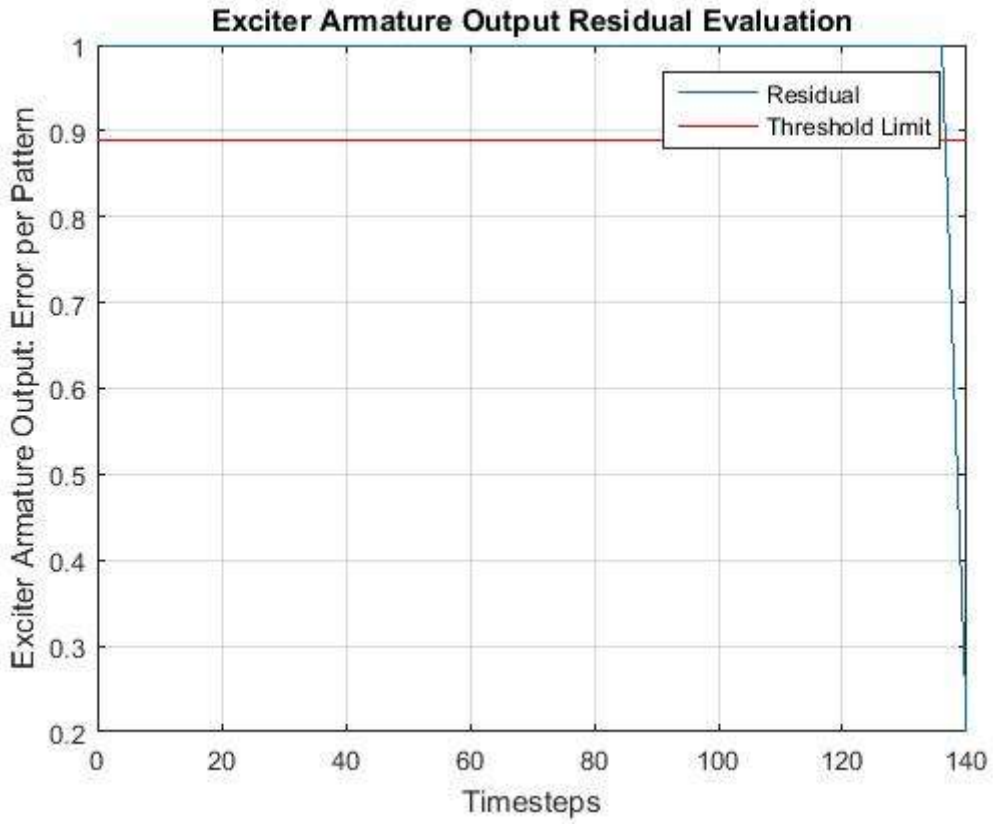


Figure L2.6.7: EXACT Test 6: EXACT Sensor Validation Test Result

L2.7 EXACT Test 7 Results

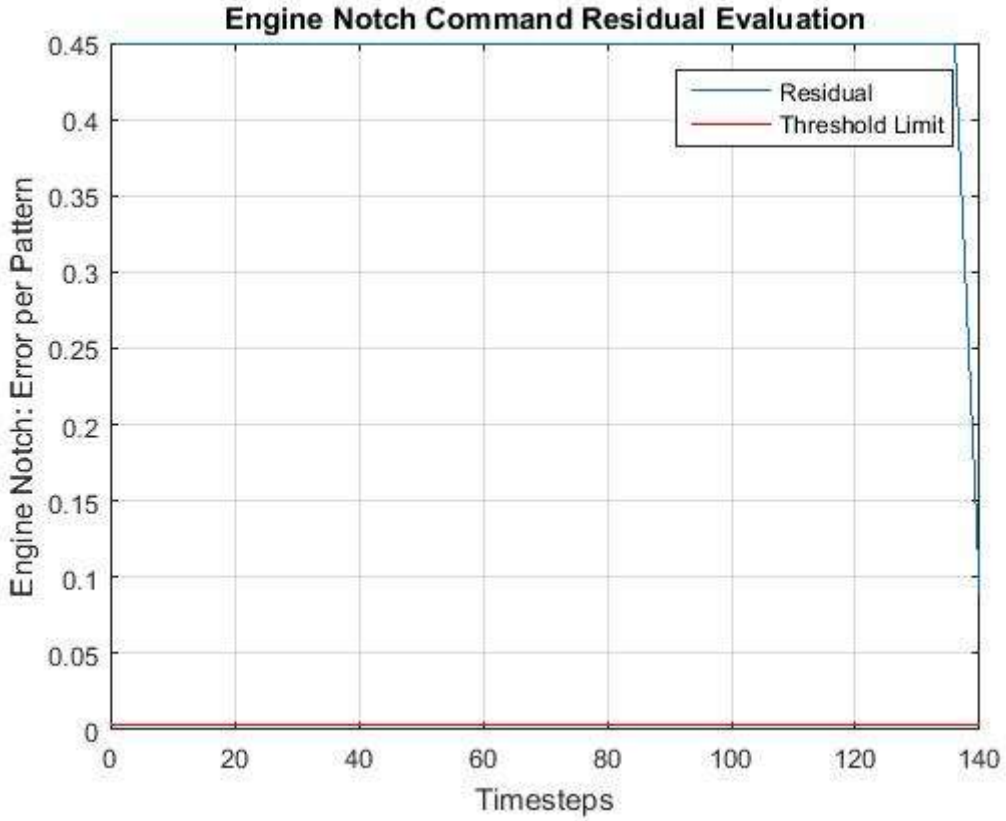


Figure L2.7.1: EXACT Test 7: Engine Notch Command Test Result

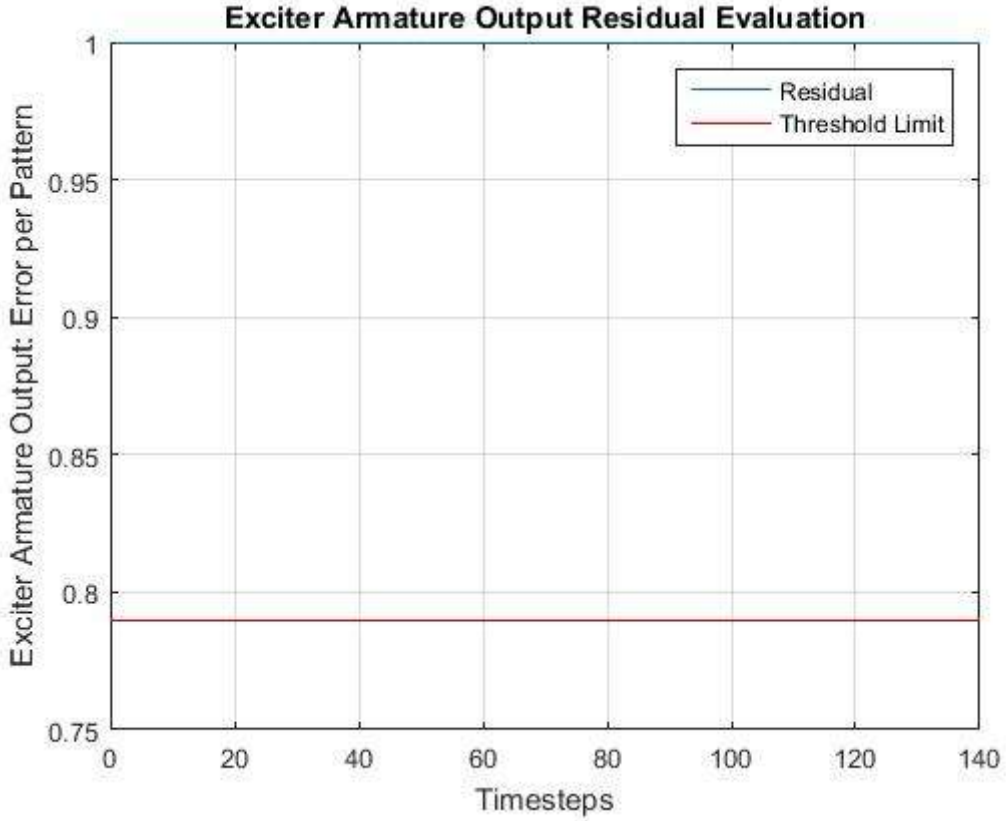


Figure L2.7.2: EXACT Test 7: EXACT Test Result

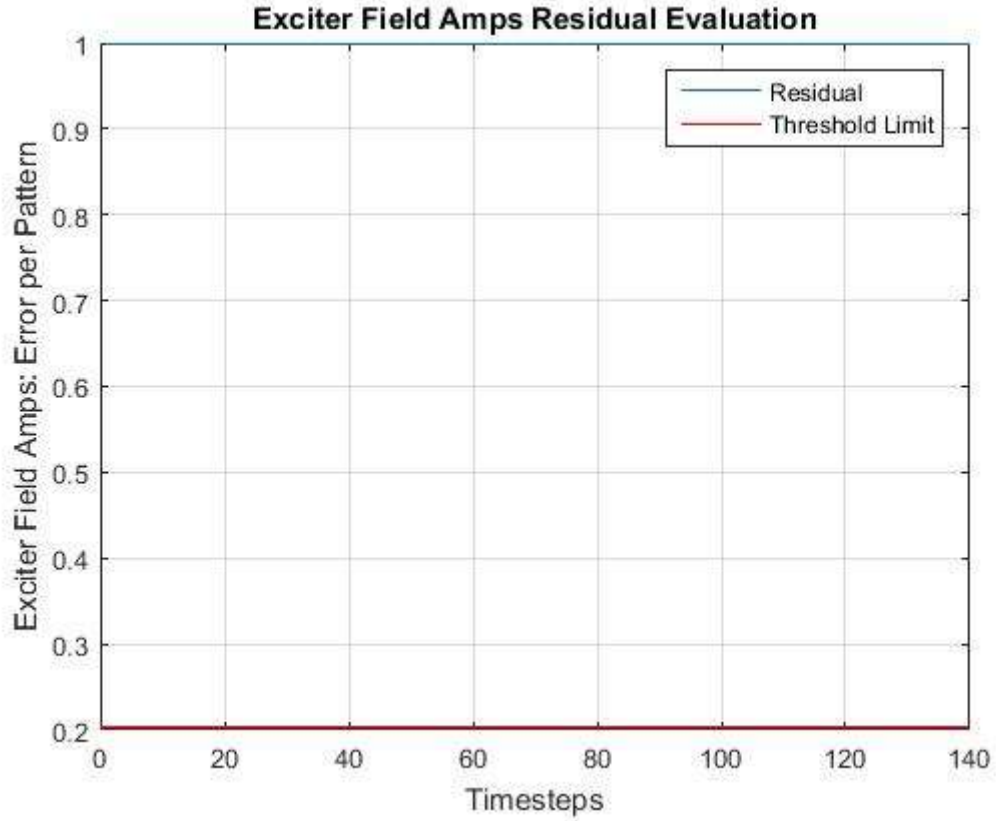


Figure L2.7.3: EXACT Test 7: EXFM Test Result

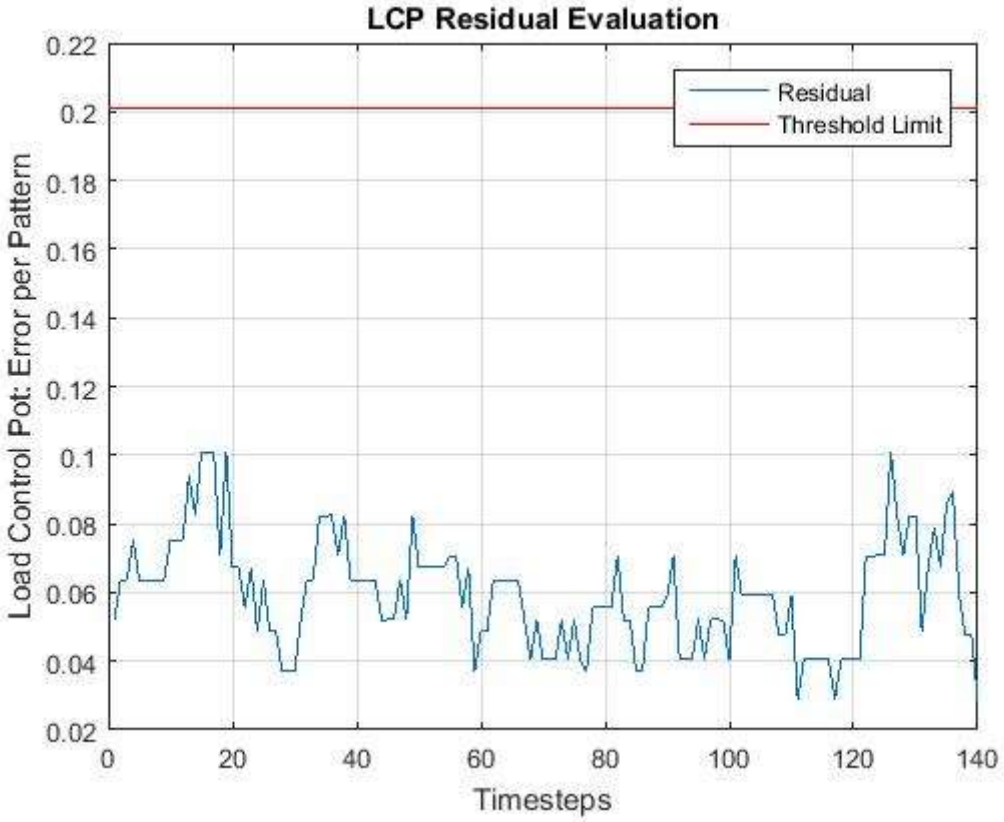


Figure L2.7.4: EXACT Test 7: LCP Test Result

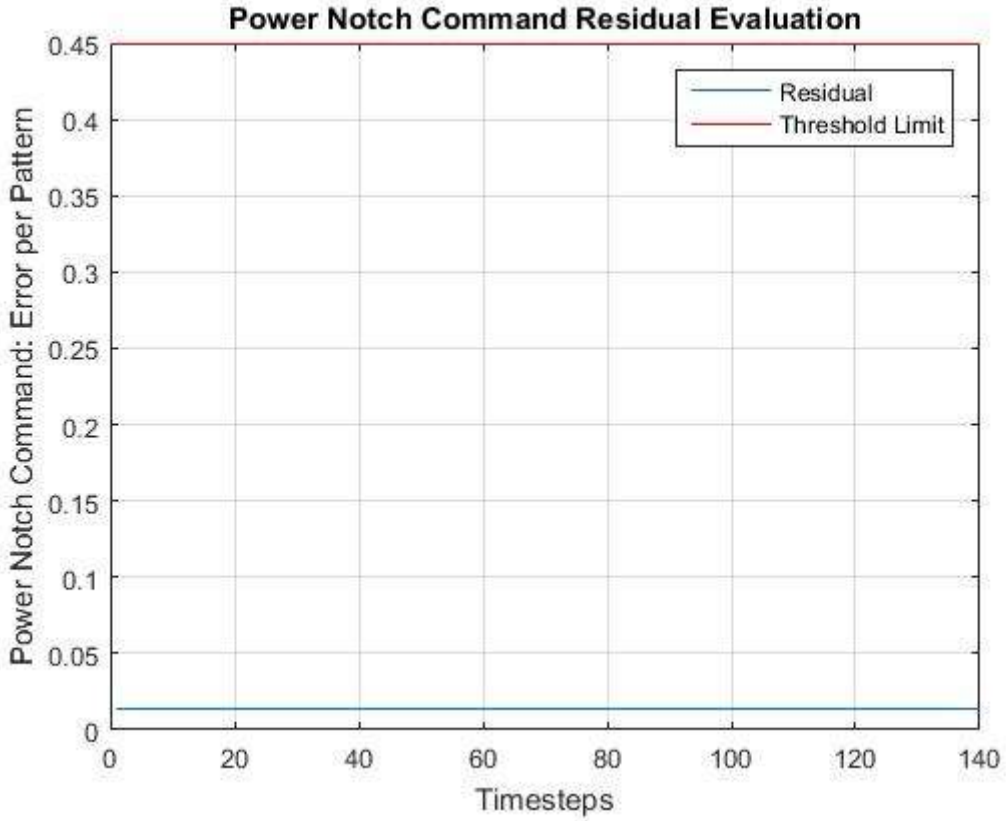


Figure L2.7.5: EXACT Test 7: Power Notch Command Test Result

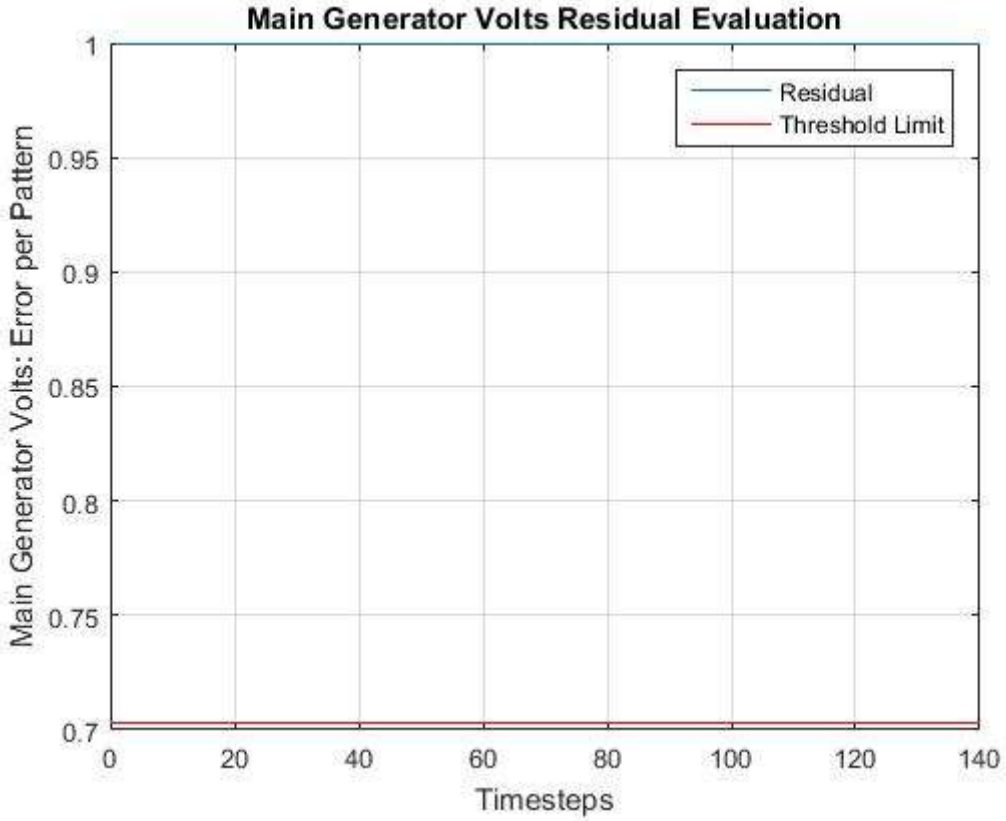


Figure L2.7.6: EXACT Test 7: SCM8 Test Result

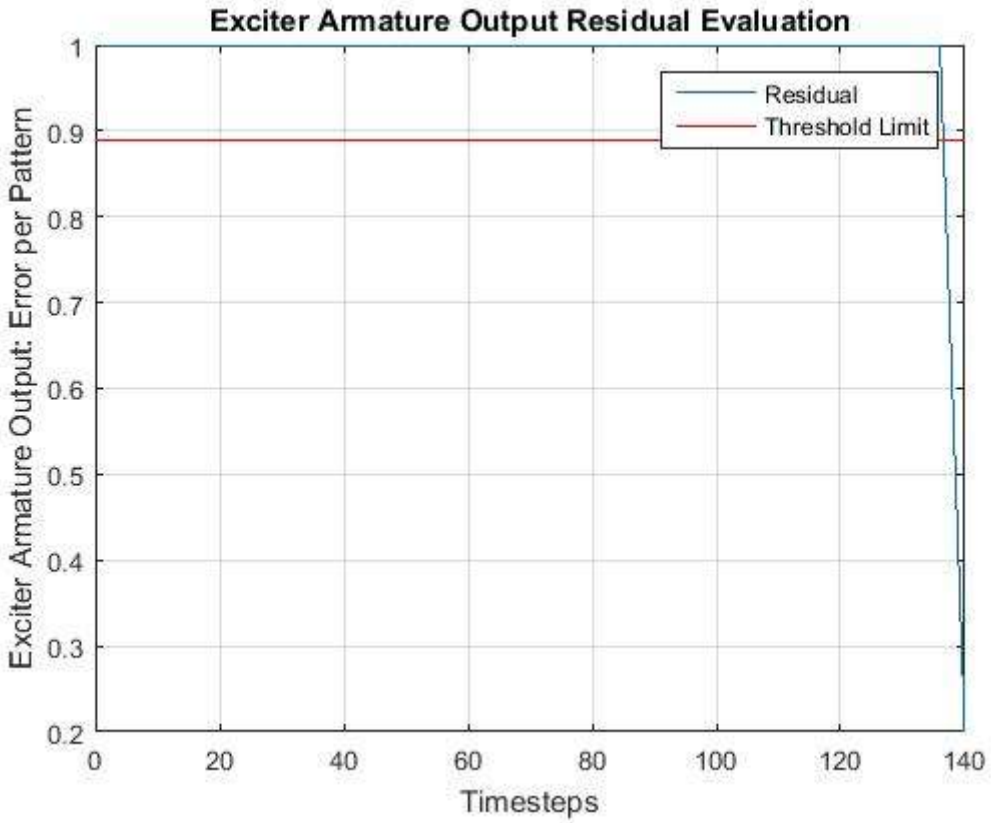


Figure L2.7.7: EXACT Test 7: EXACT Sensor Validation Test Result

L3.1 LCP Test 1 Results

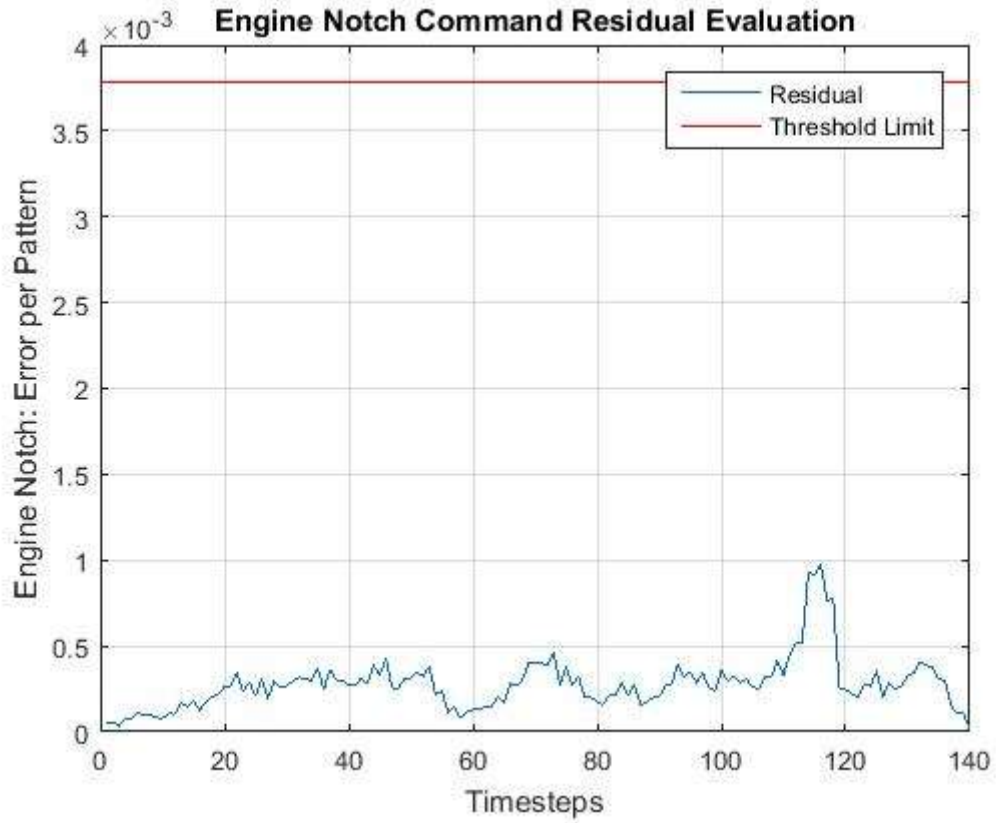


Figure L3.1.1: LCP Test 1: Engine Notch Command Test Result

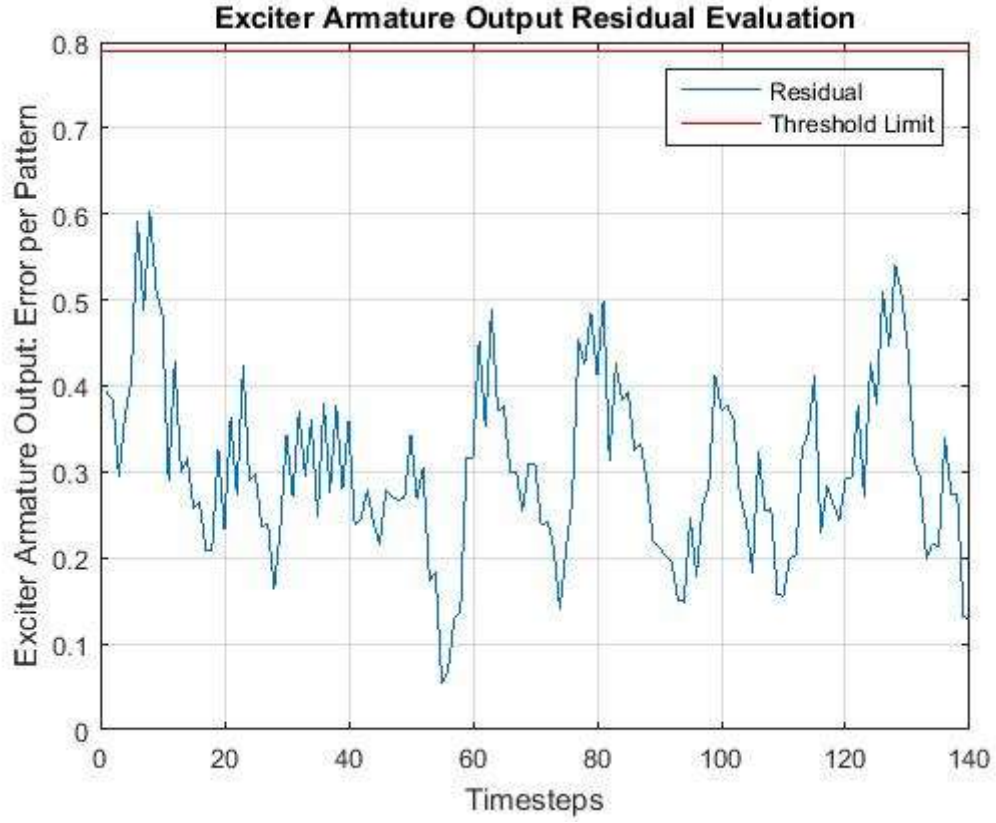


Figure L3.1.2: LCP Test 1: EXACT Test Result

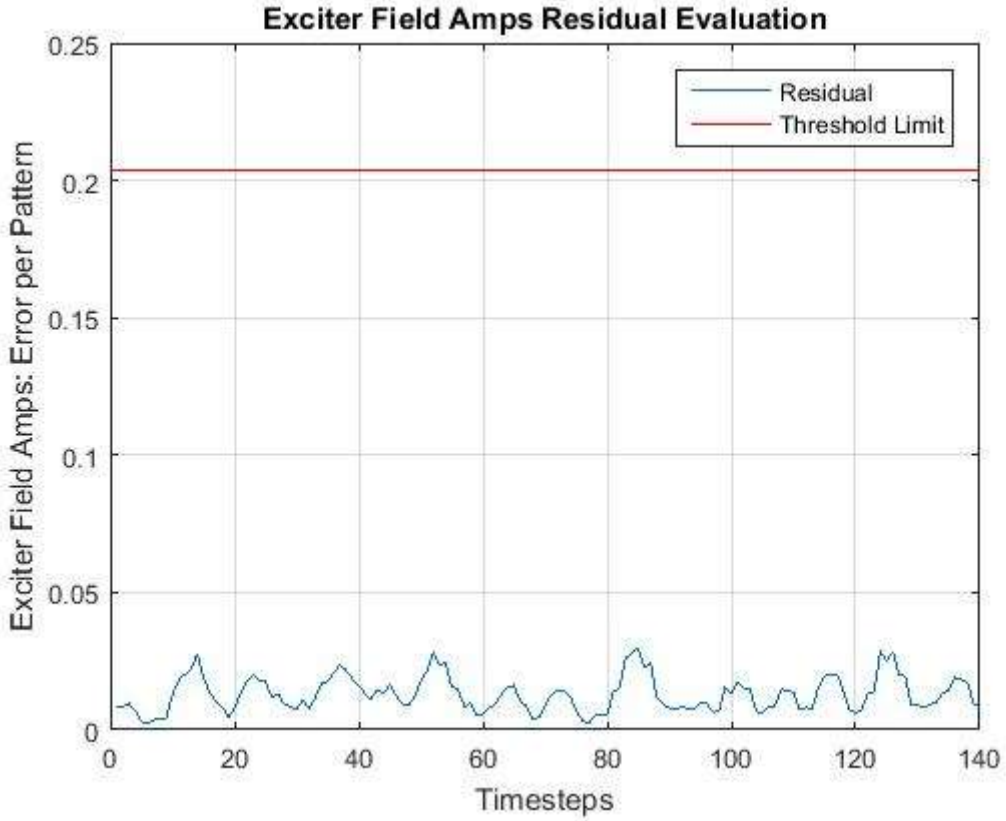


Figure L3.1.3: LCP Test 1: EXFM Test Result

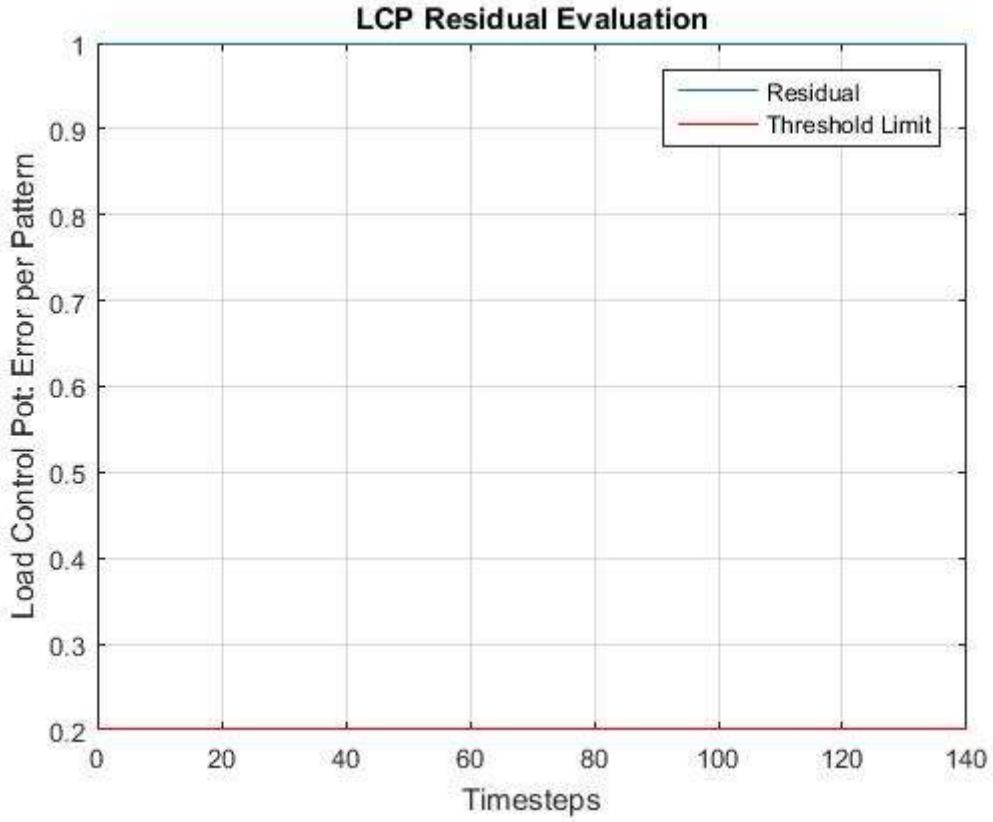


Figure L3.1.4: LCP Test 1: LCP Test Result

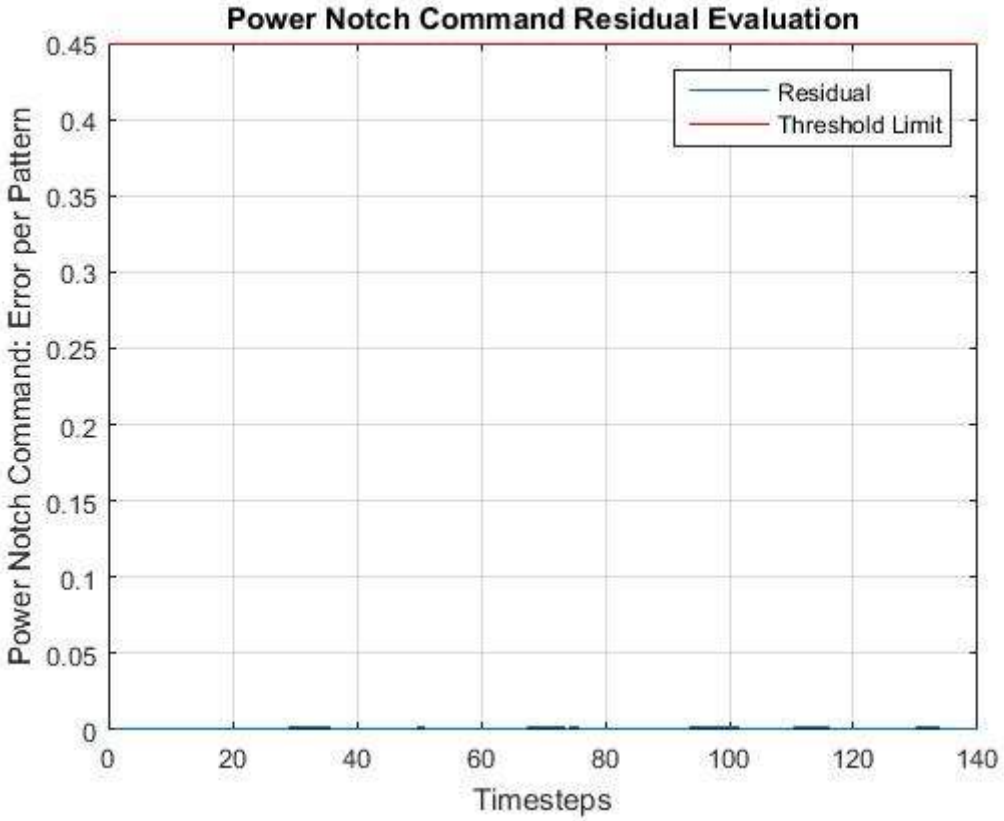


Figure L3.1.5: LCP Test 1: Power Notch Command Test Result

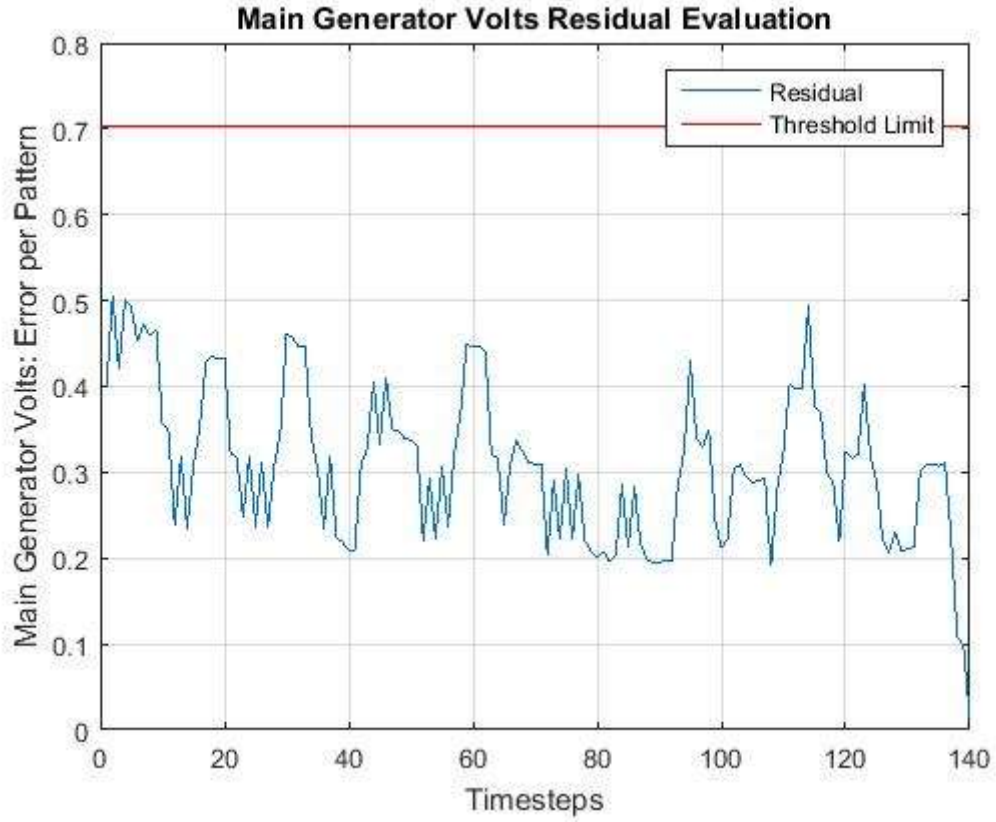


Figure L3.1.6: LCP Test 1: SCM8 Test Result

L3.2 LCP Test 2 Results

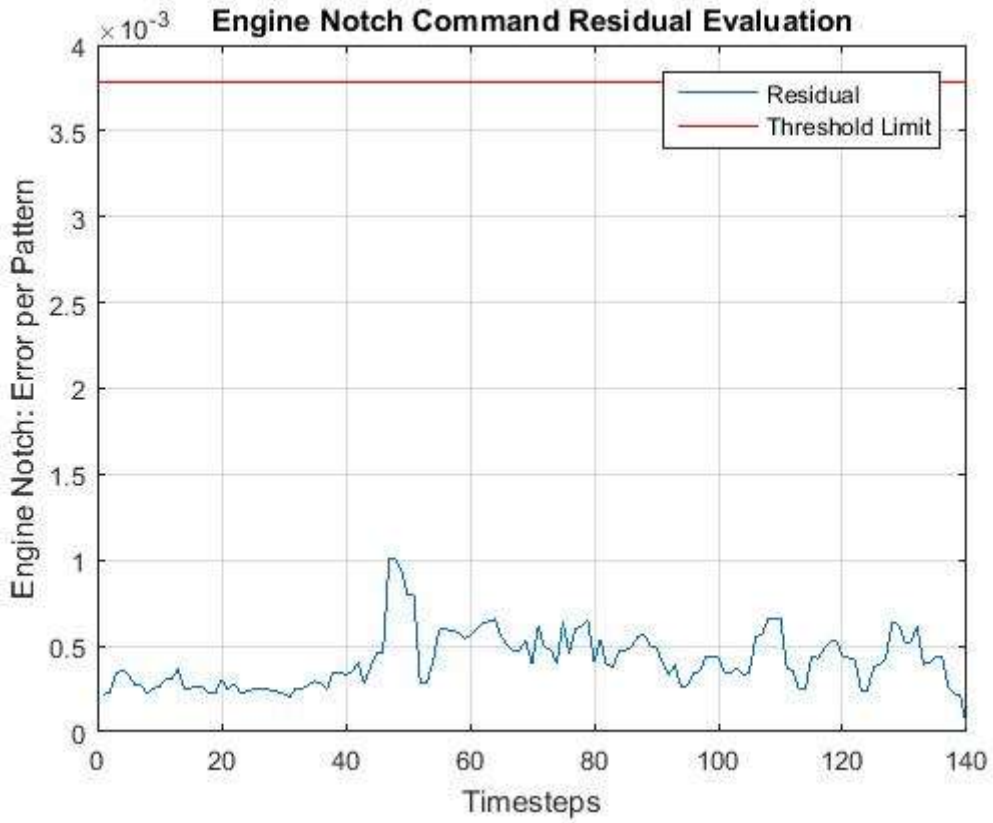


Figure L3.2.1: LCP Test 2: Engine Notch Command Test Result

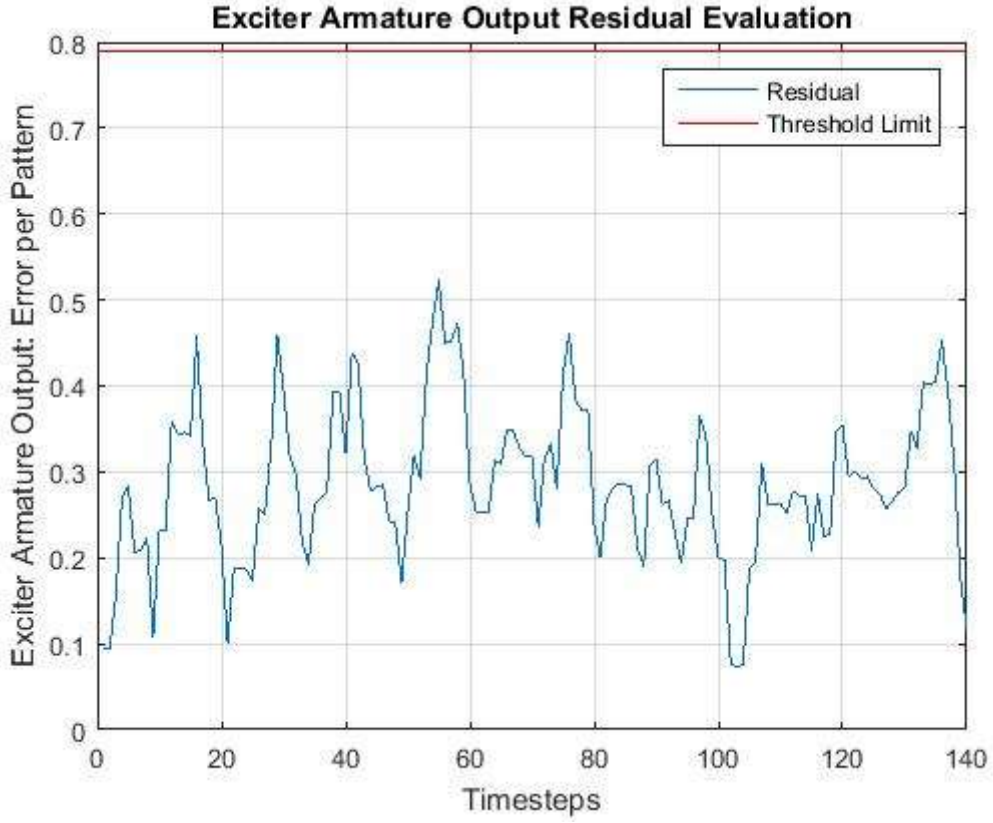


Figure L3.2.2: LCP Test 2: EXACT Test Result

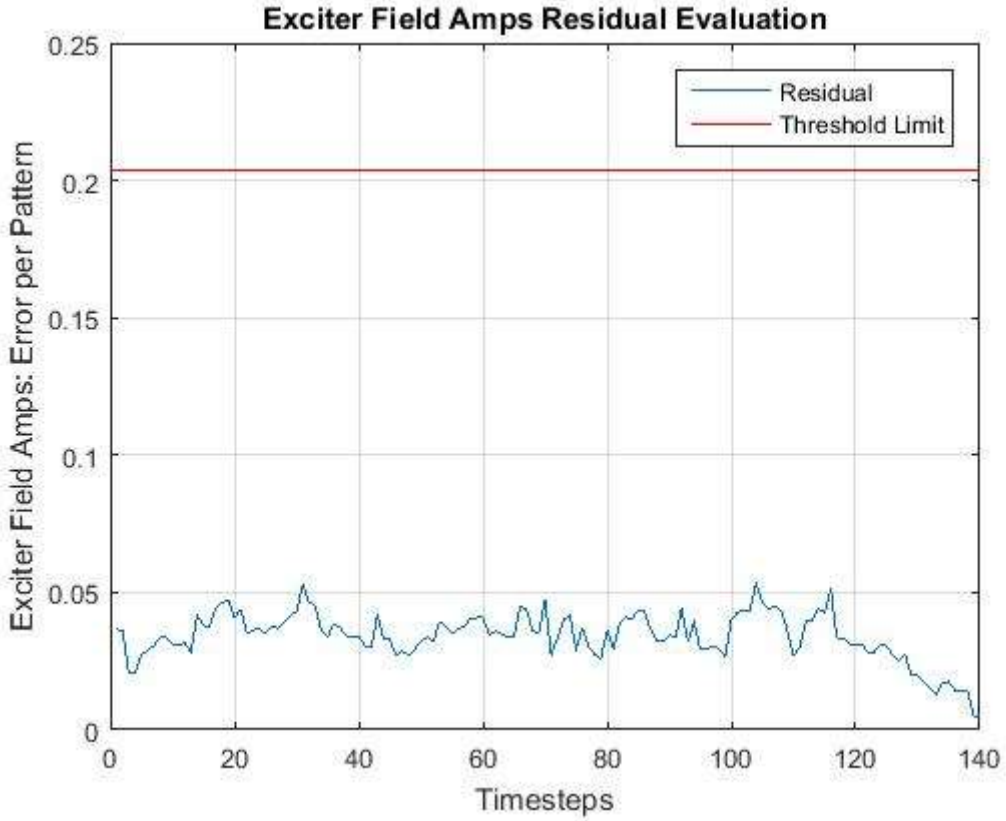


Figure L3.2.3: LCP Test 2: EXFM Test Result

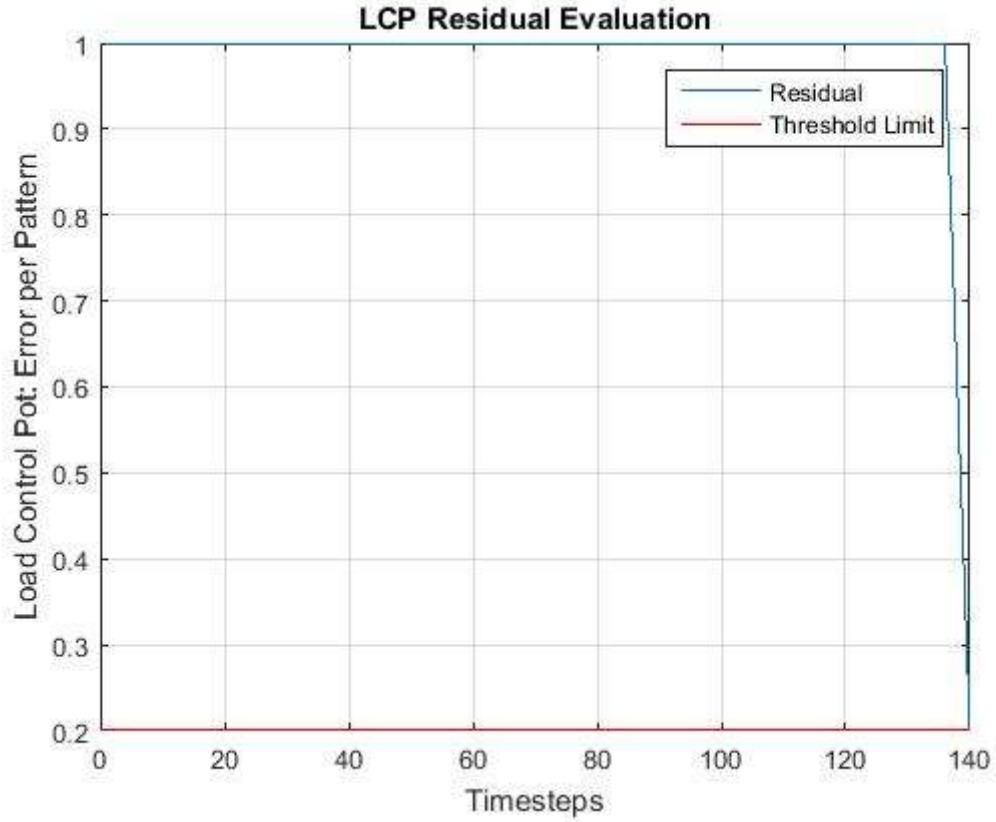


Figure L3.2.4: LCP Test 2: LCP Test Result

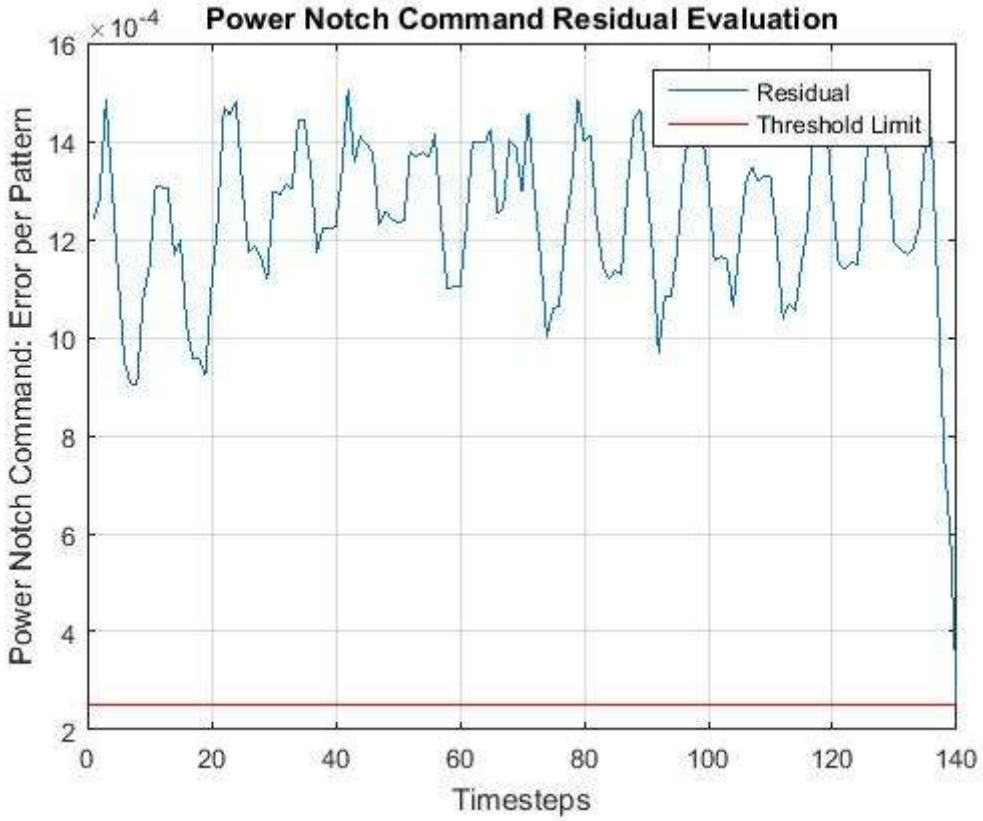


Figure L3.2.5: LCP Test 2: Power Notch Command Test Result

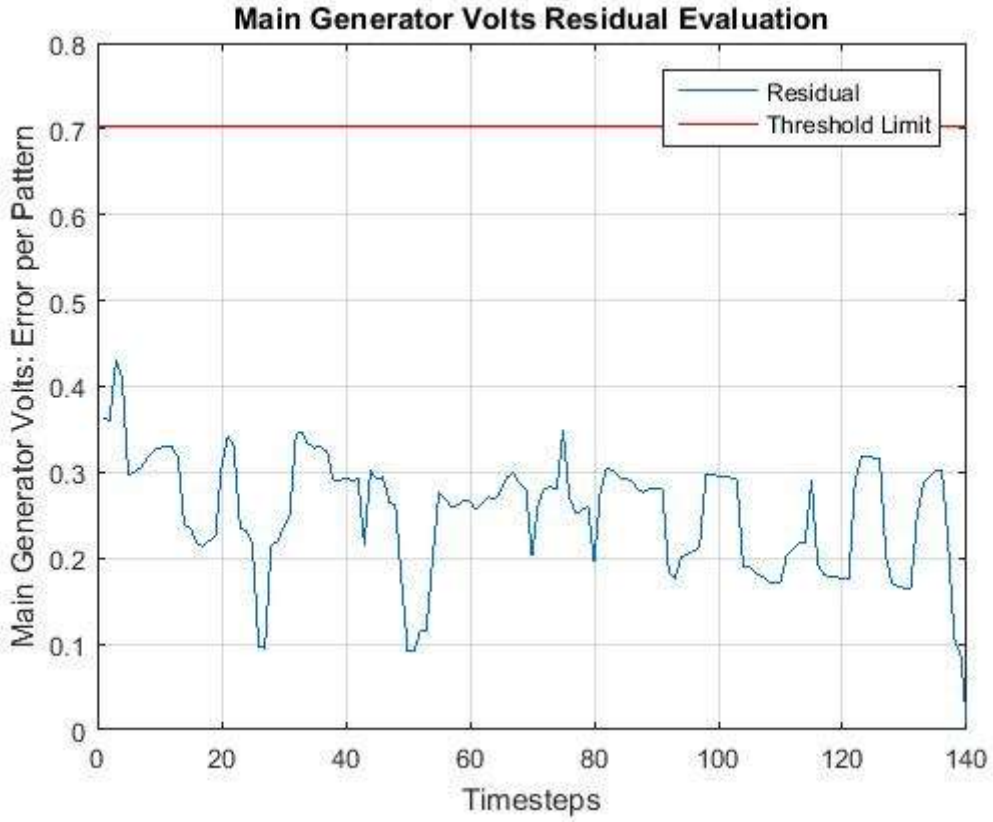


Figure L3.2.6: LCP Test 2: SCM8 Test Result

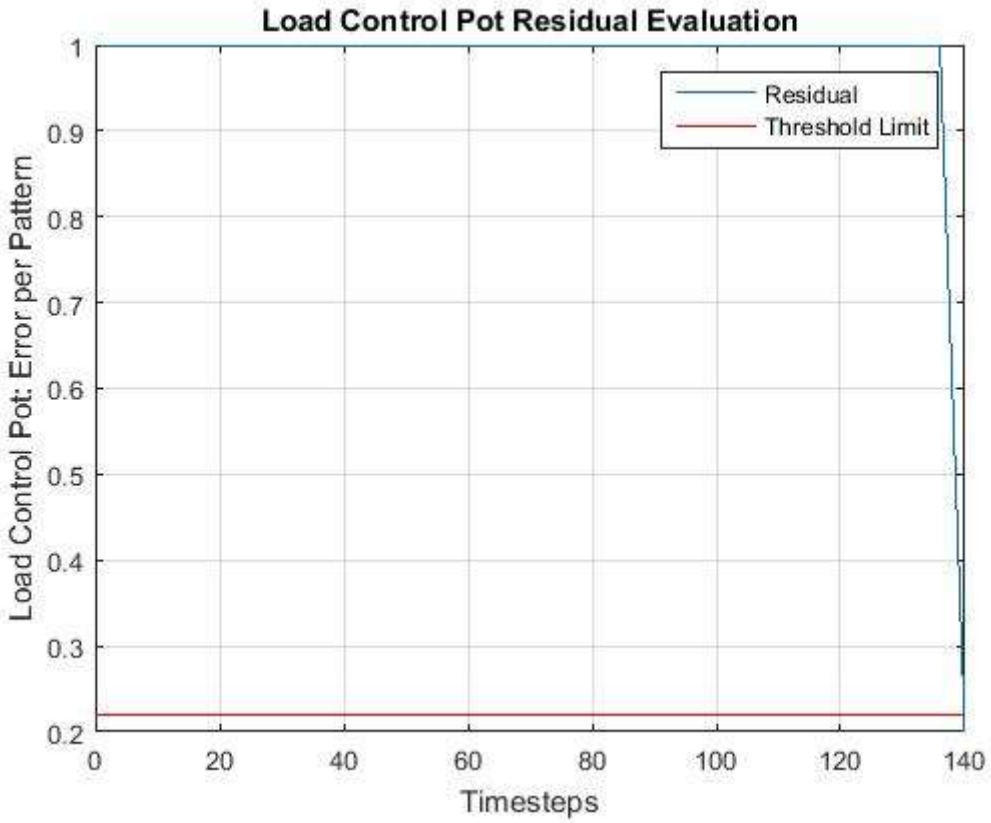


Figure L3.2.7: LCP Test 2: LCP Sensor Validation Test Result

L3.3 LCP Test 3 Results

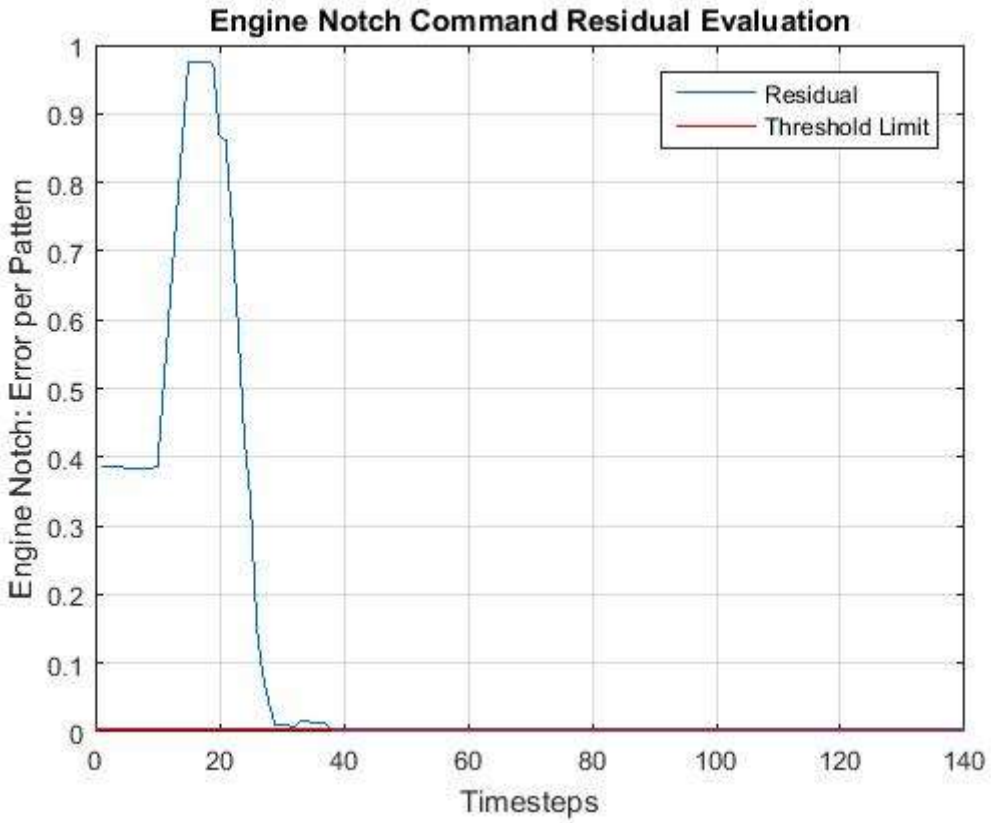


Figure L3.3.1: LCP Test 3: Engine Notch Command Test Result

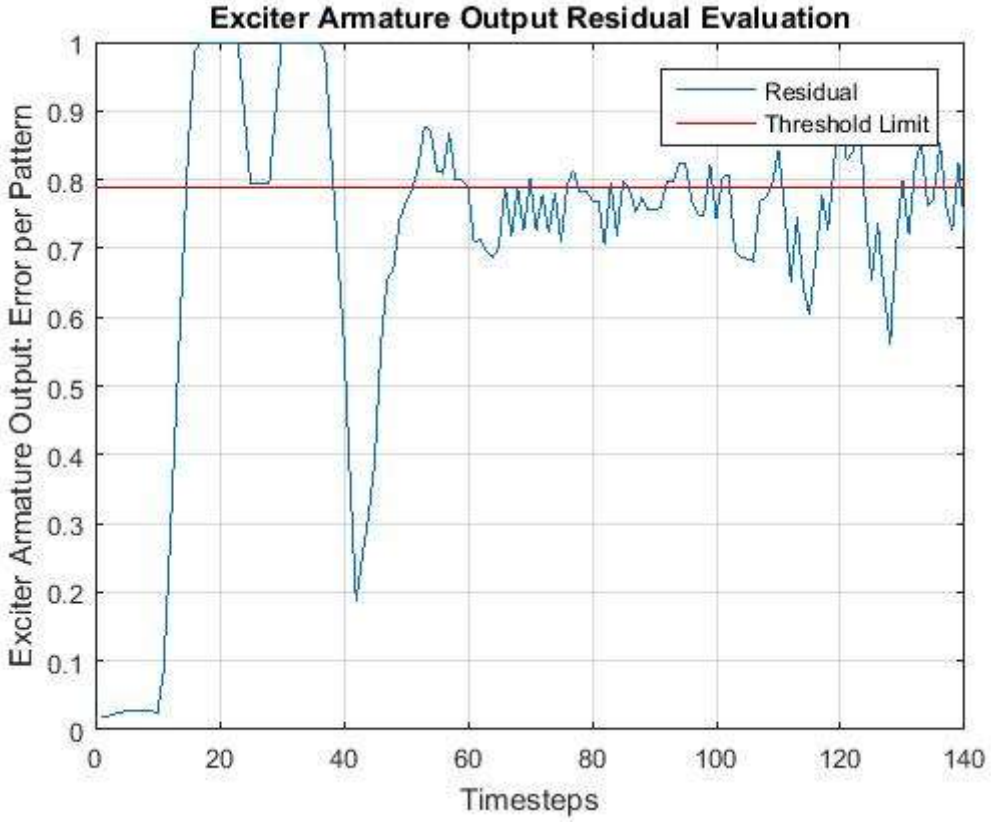


Figure L3.3.2: LCP Test 3: EXACT Test Result

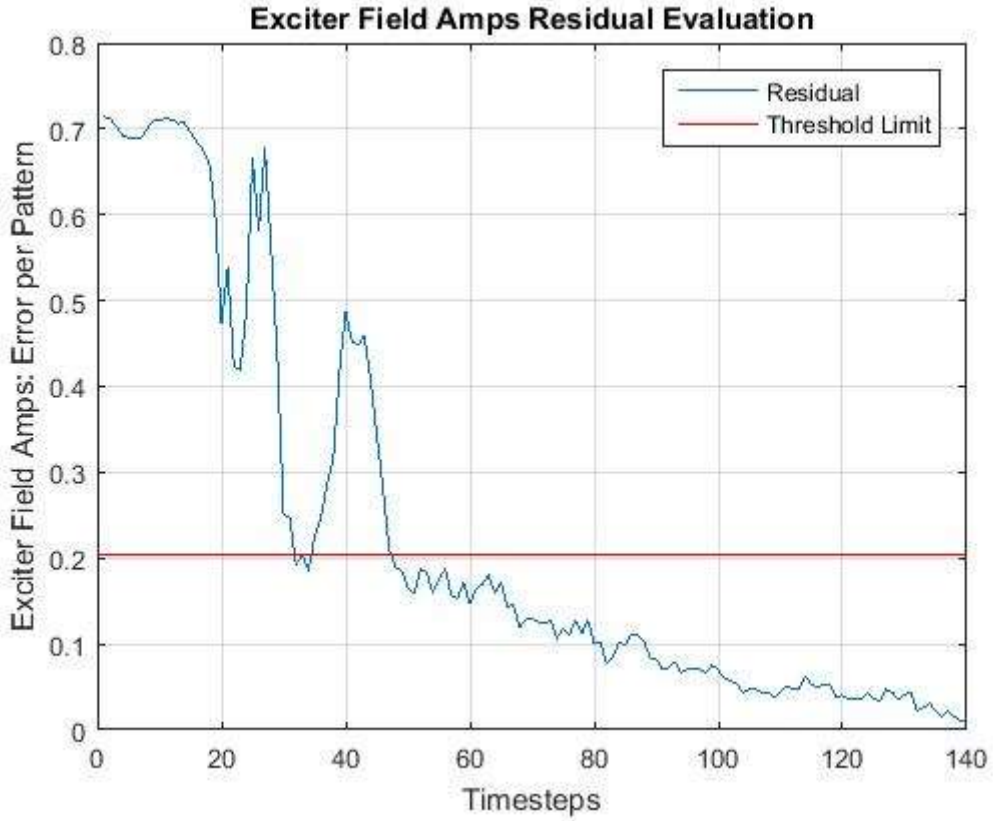


Figure L3.3.3: LCP Test 3: EXFM Test Result

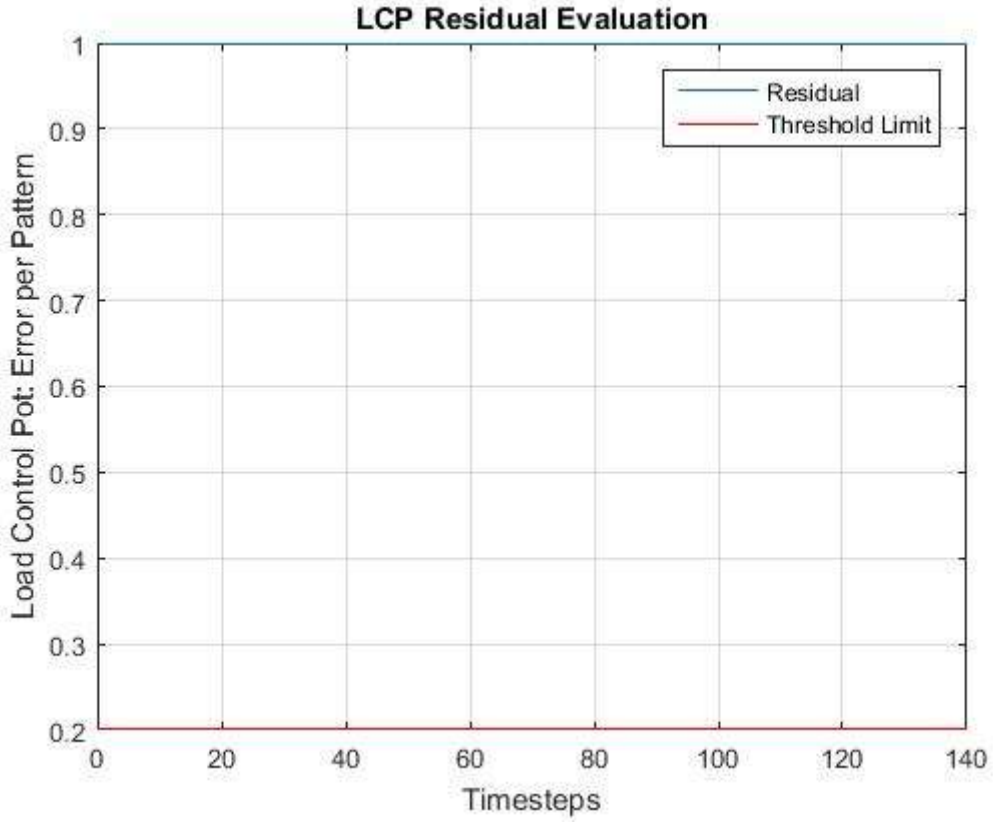


Figure L3.3.4: LCP Test 3: LCP Test Result

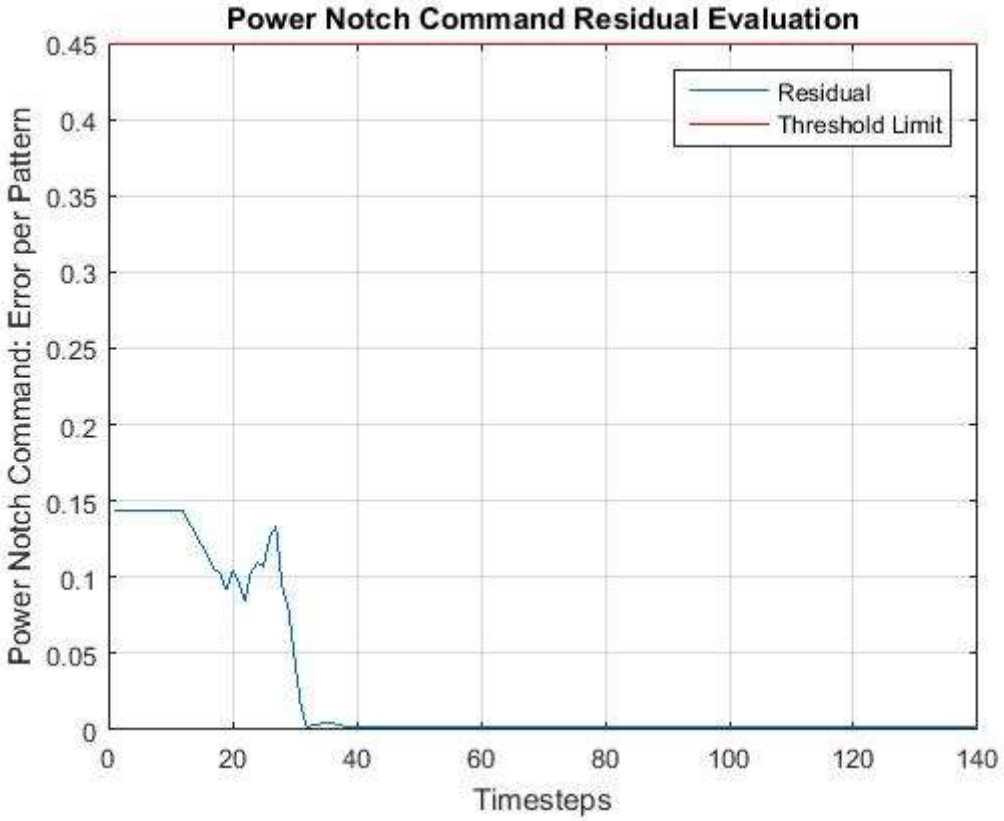


Figure L3.3.5: LCP Test 3: Power Notch Command Test Result

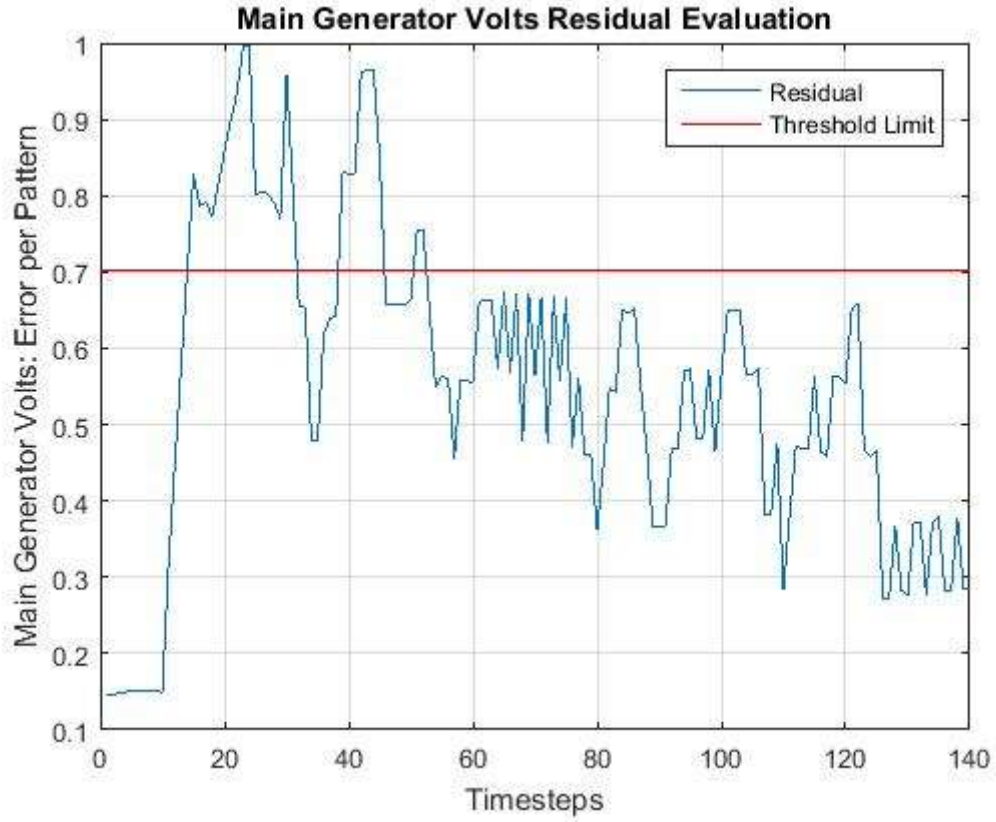


Figure L3.3.6: LCP Test 3: SCM8 Test Result

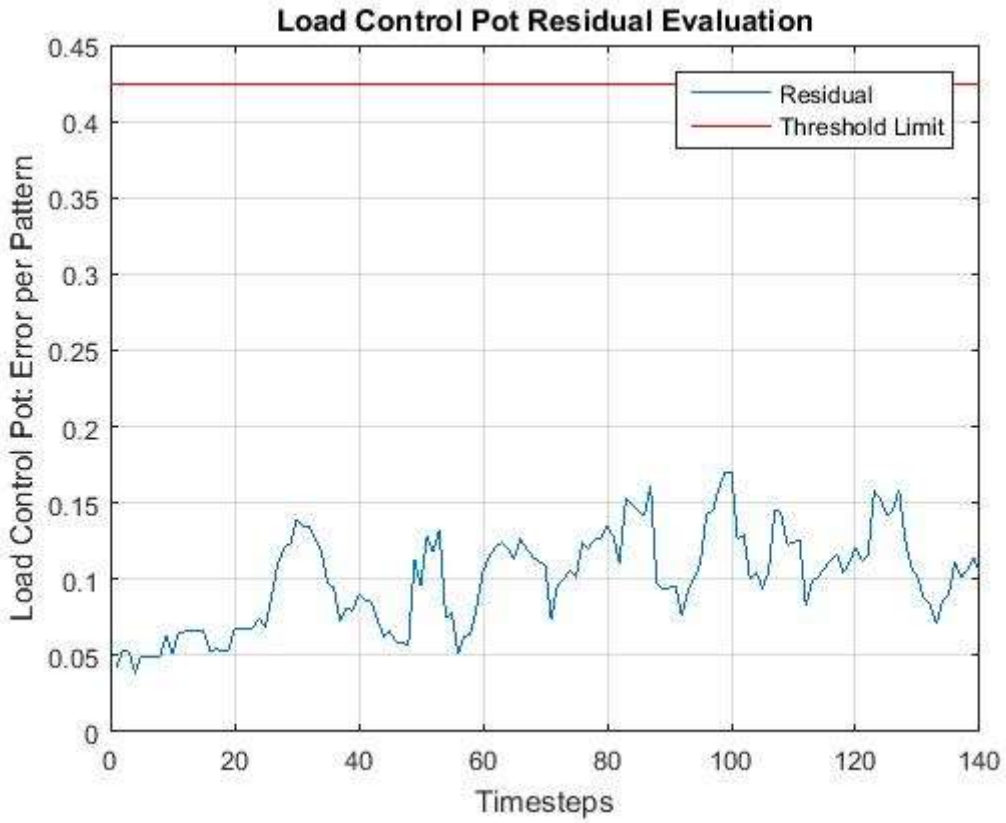


Figure L3.3.7: LCP Test 3: LCP Sensor Validation Test Result

L4.1 EXFM Test 1 Results

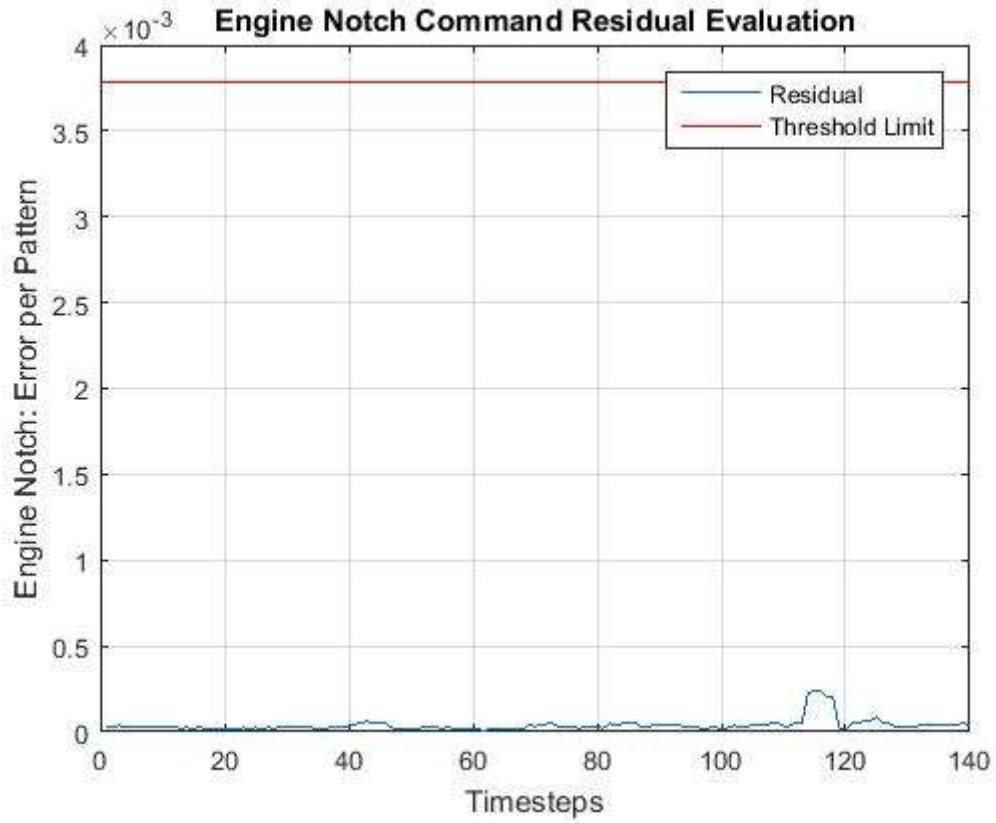


Figure L4.1.1: EXFM Test 1: Engine Notch Command Test Result

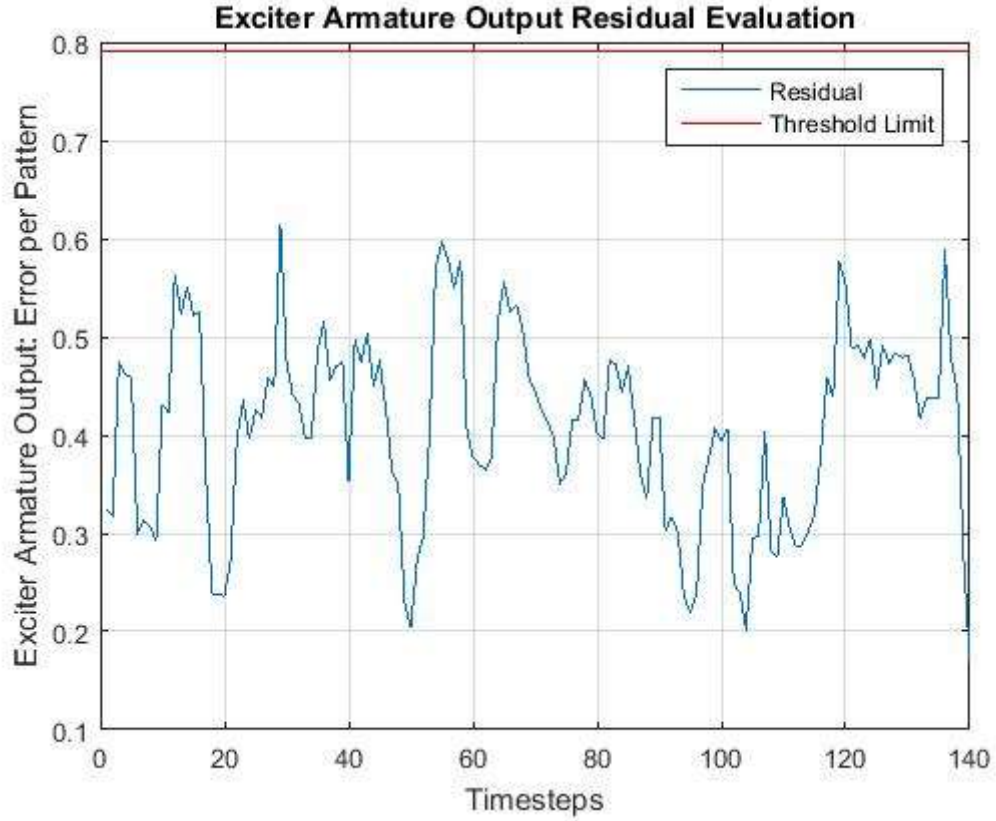


Figure L4.1.2: EXFM Test 1: EXACT Test Result

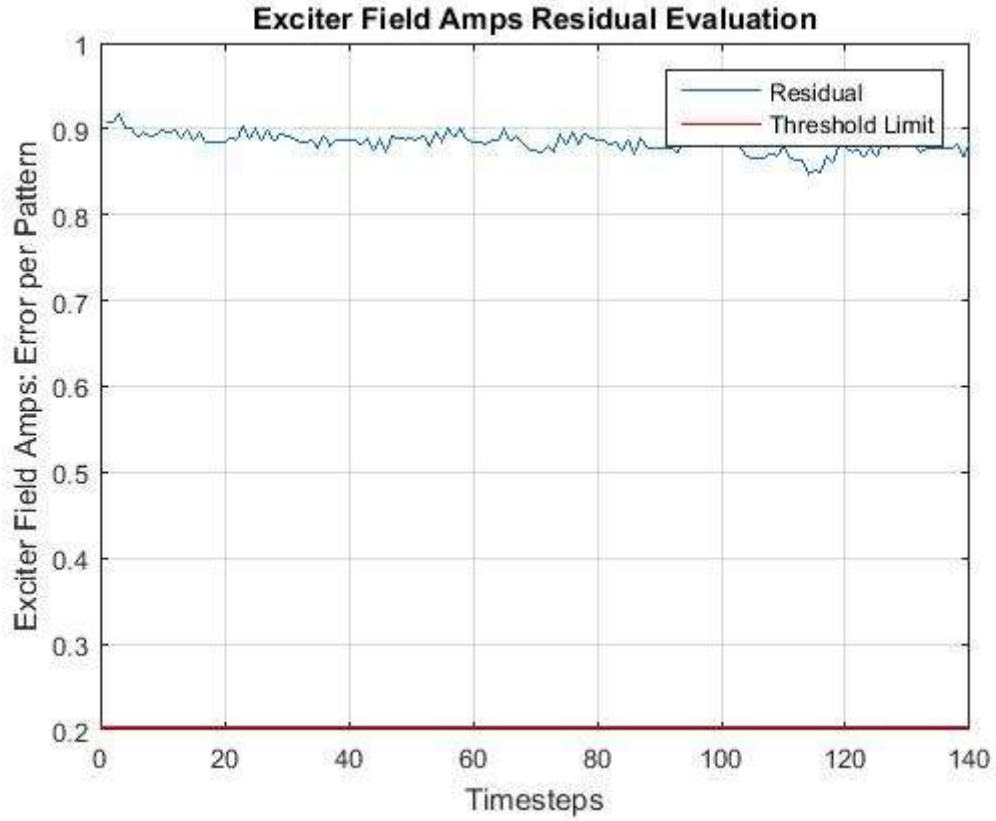


Figure L4.1.3: EXFM Test 1: EXFM Test Result

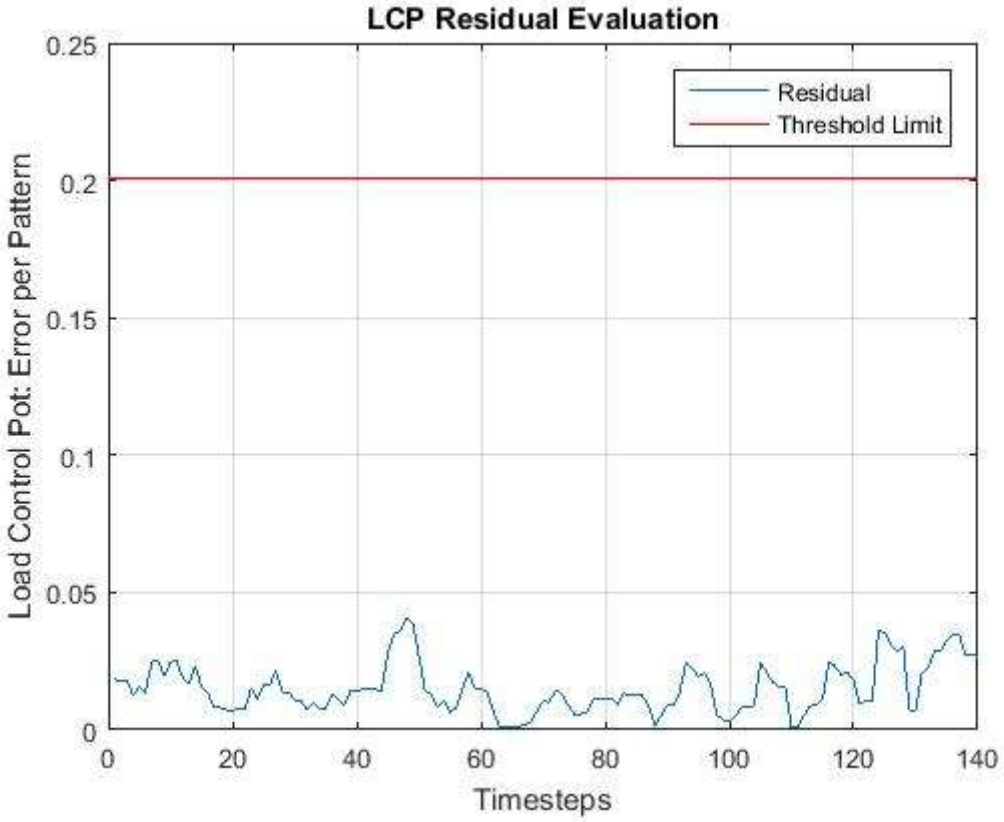


Figure L4.1.4: EXFM Test 1: LCP Test Result

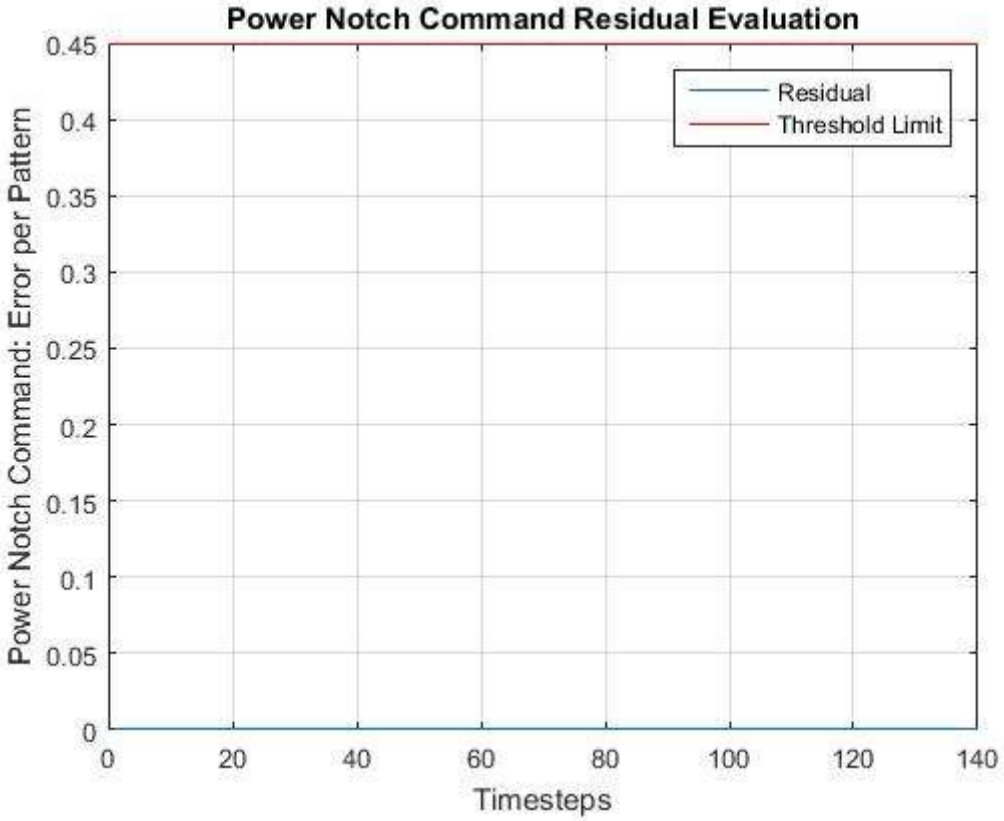


Figure L4.1.5: EXFM Test 1: Power Notch Command Test Result

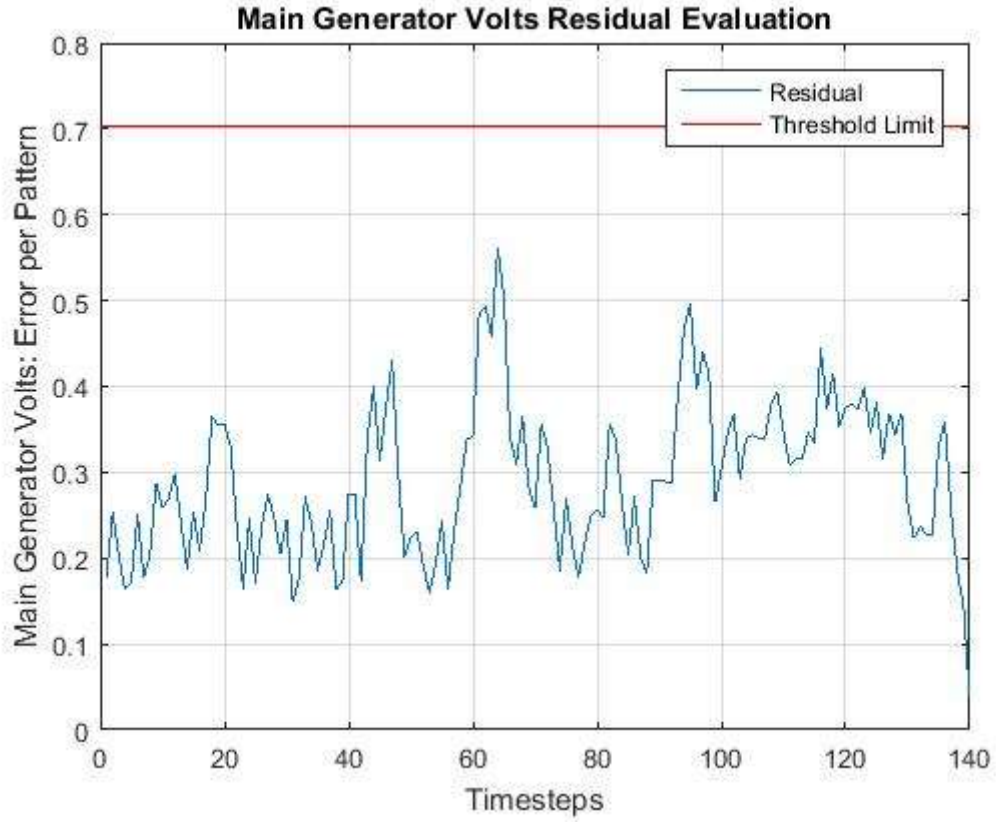


Figure L4.1.6: EXFM Test 1: SCM8 Test Result

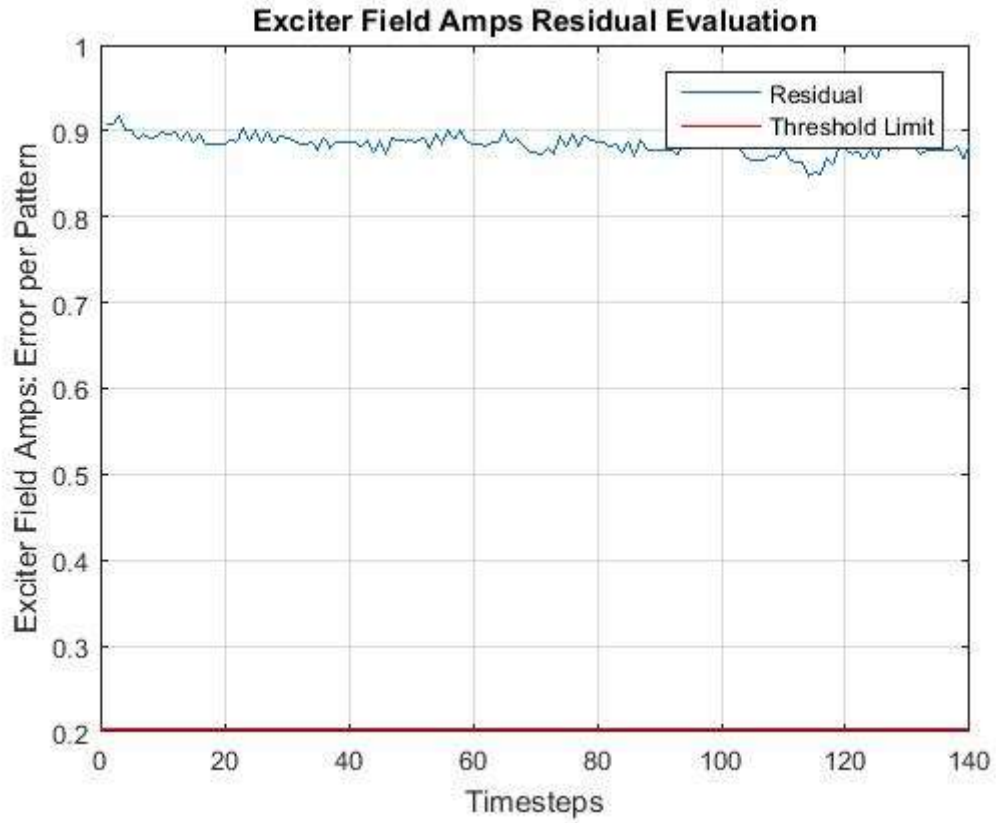


Figure L4.1.7: EXFM Test 1: EXFM Sensor Validation Test Result

L4.2 EXFM Test 2 Results

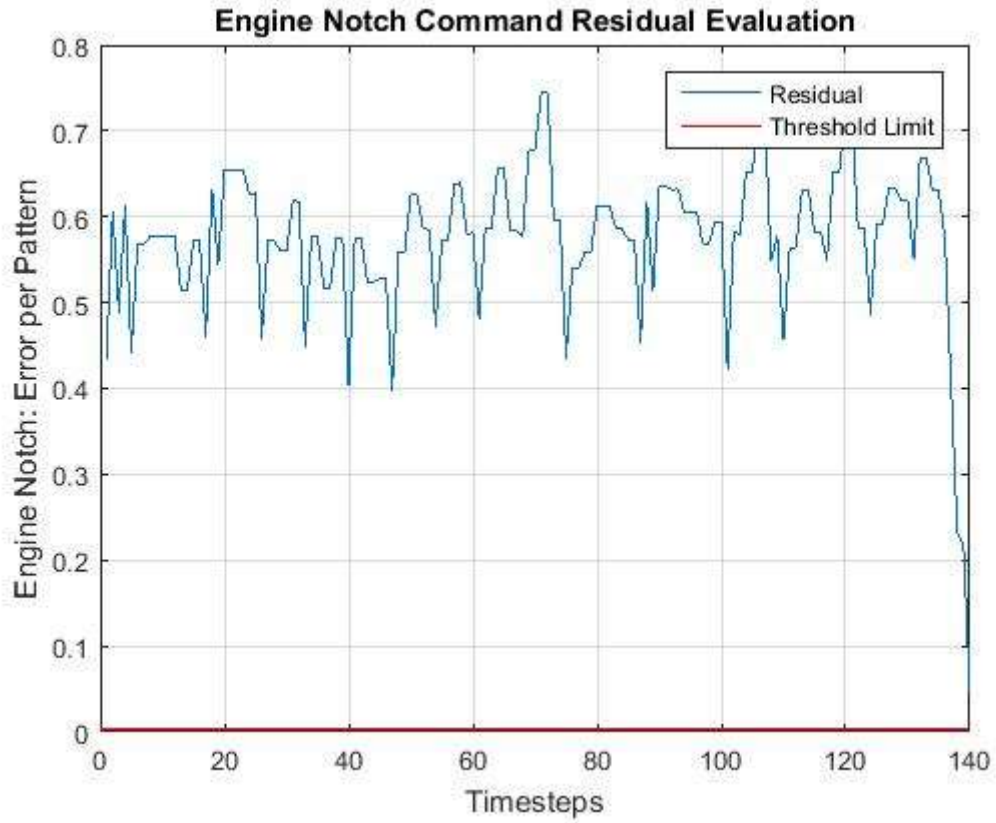


Figure L4.2.1: EXFM Test 2: Engine Notch Command Test Result

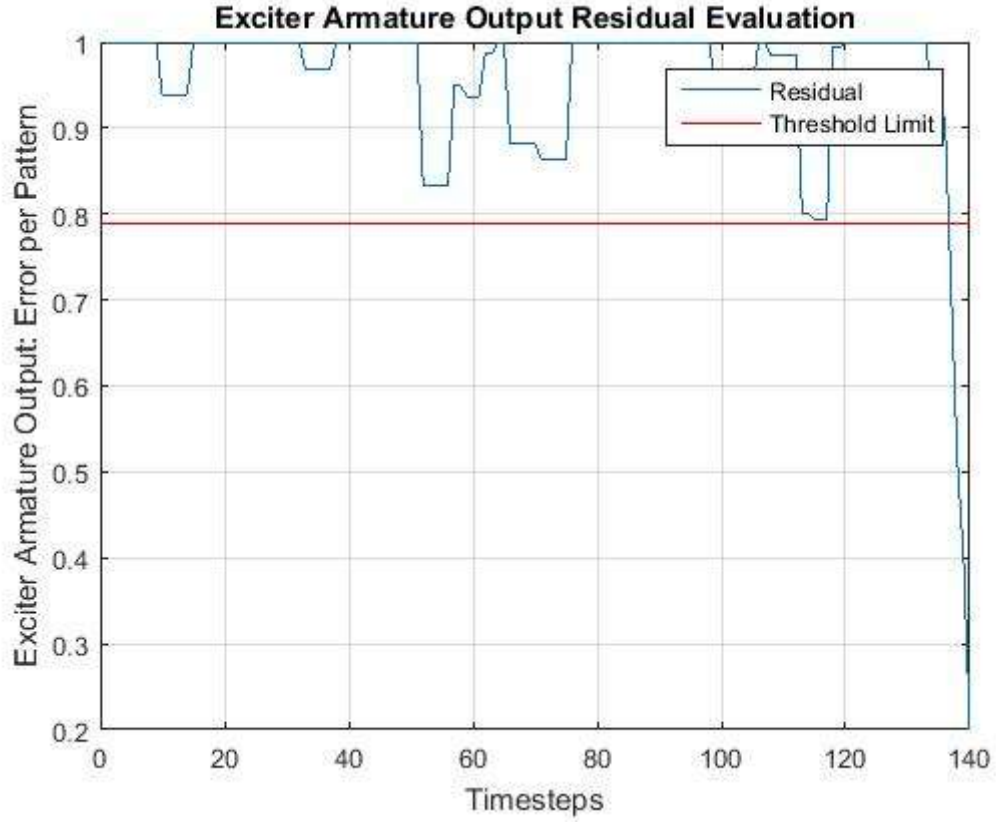


Figure L4.2.2: EXFM Test 2: EXACT Test Result

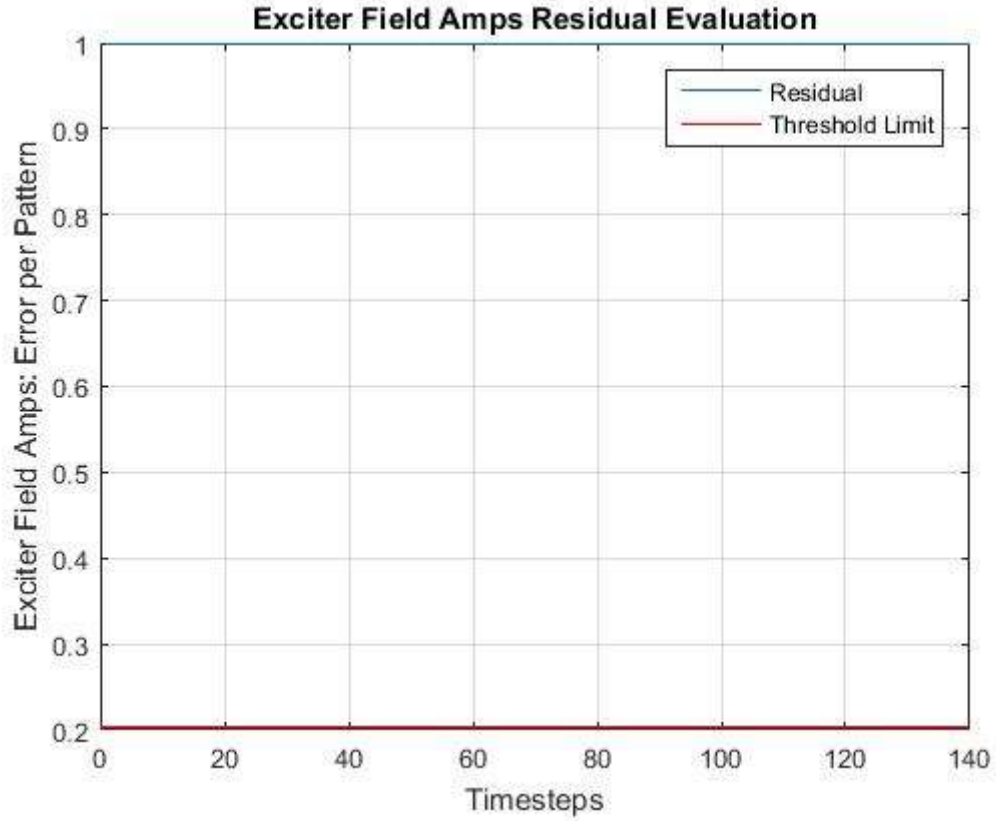


Figure L4.2.3: EXFM Test 2: EXFM Test Result

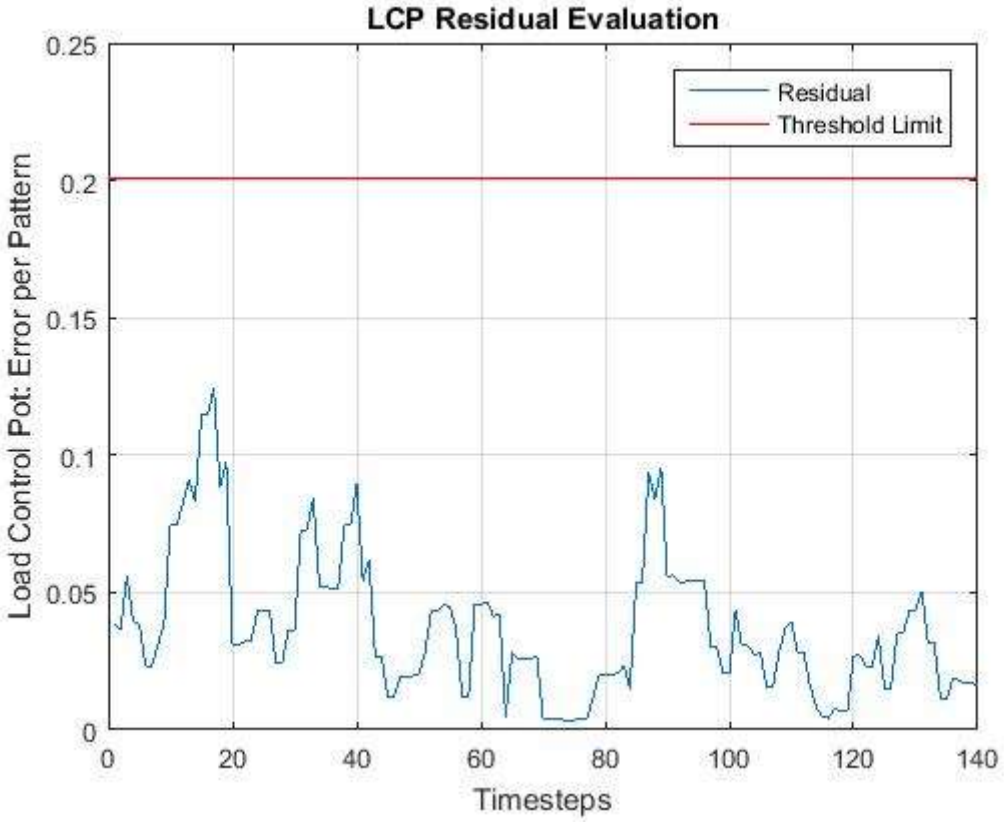


Figure L4.2.4: EXFM Test 2: LCP Test Result

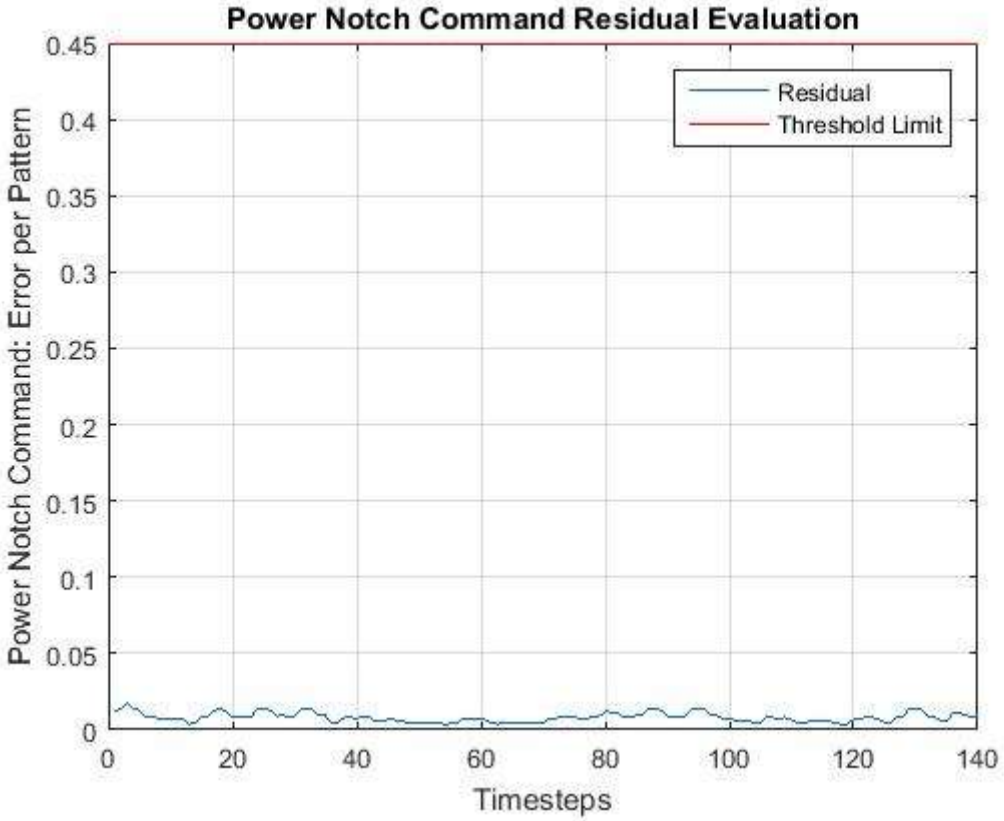


Figure L4.2.5: EXFM Test 2: Power Notch Command Test Result

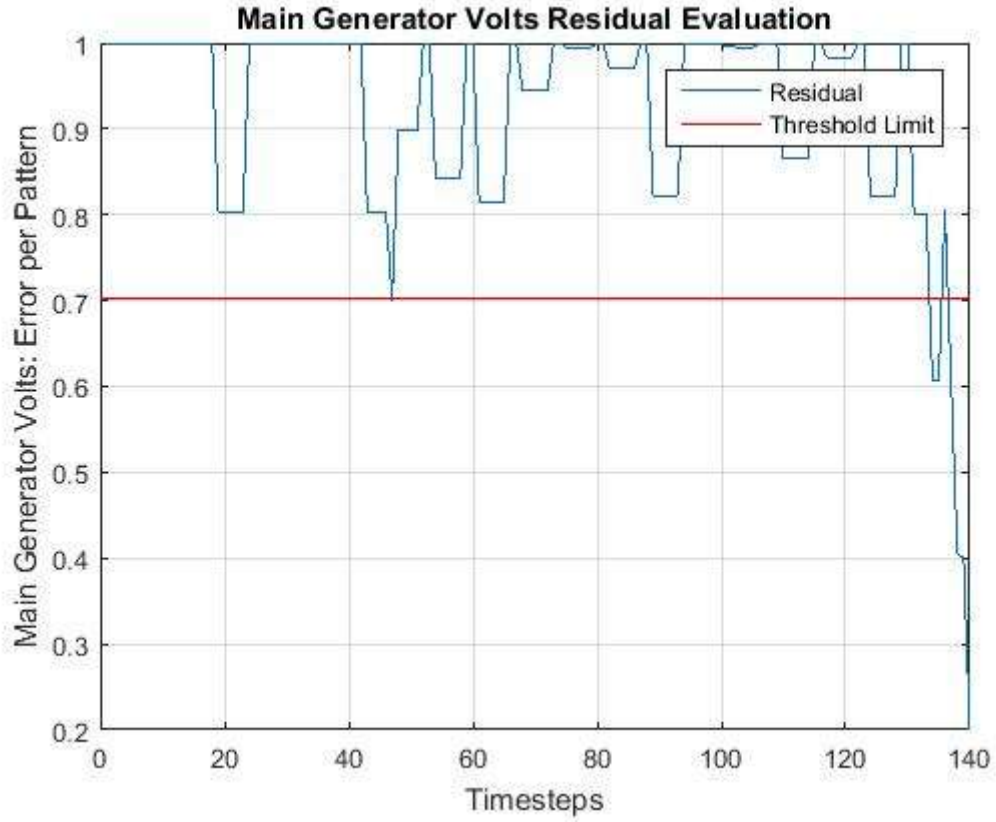


Figure L4.2.6: EXFM Test 2: SCM8 Test Result

L4.3 EXFM Test 3 Results

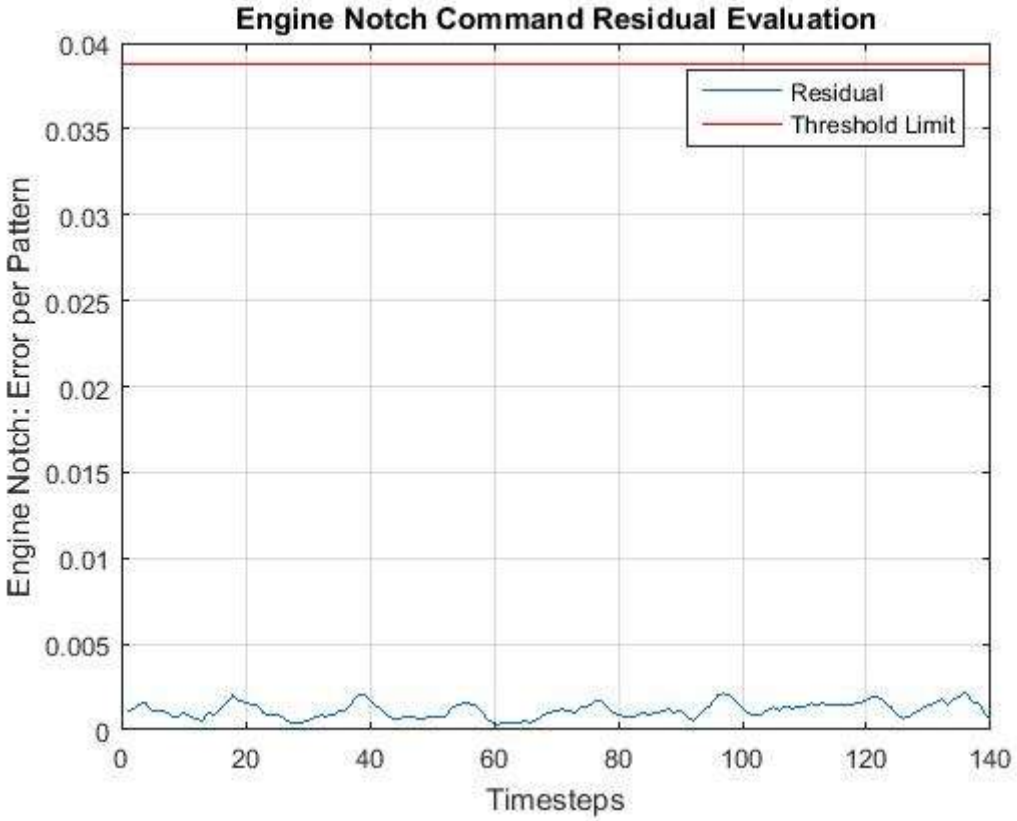


Figure L4.3.1: EXFM Test 3: Engine Notch Command Test Result

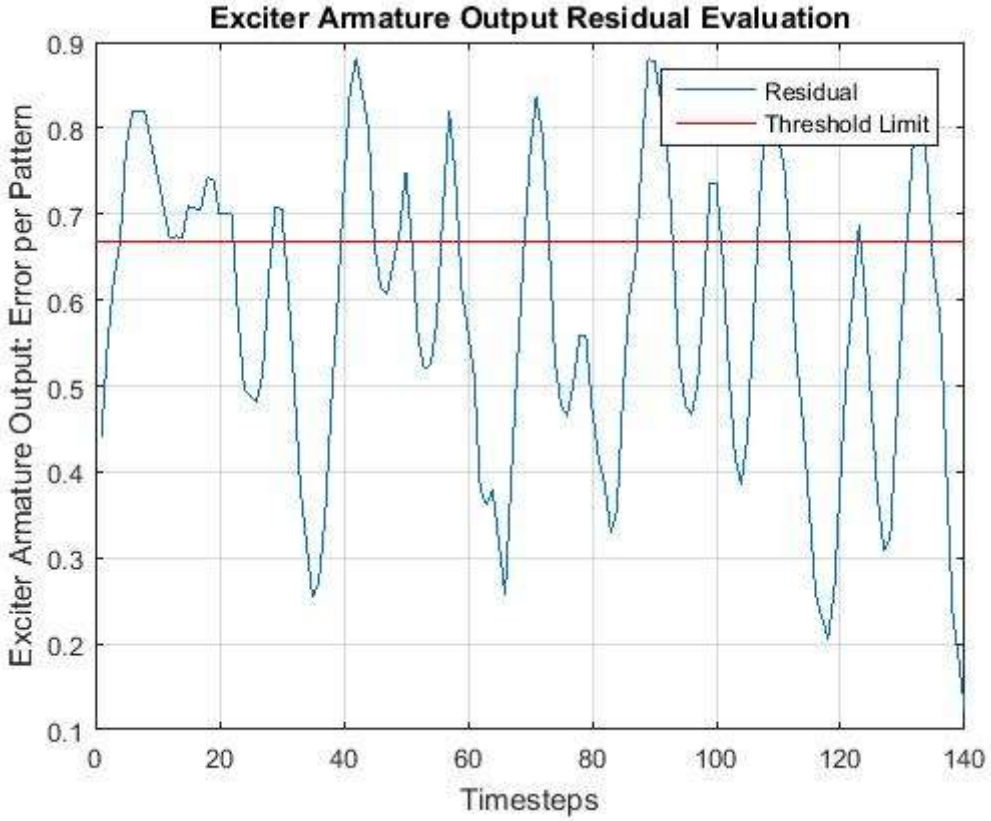


Figure L4.3.2: EXFM Test 3: EXACT Test Result

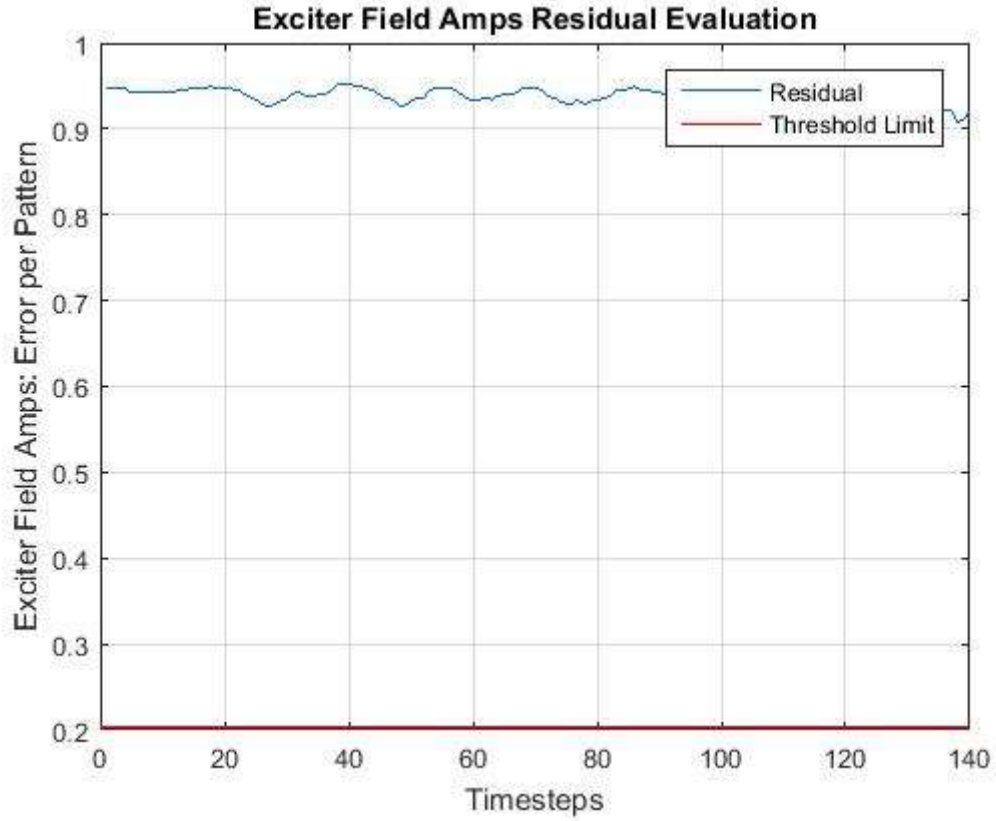


Figure L4.3.3: EXFM Test 3: EXFM Test Result

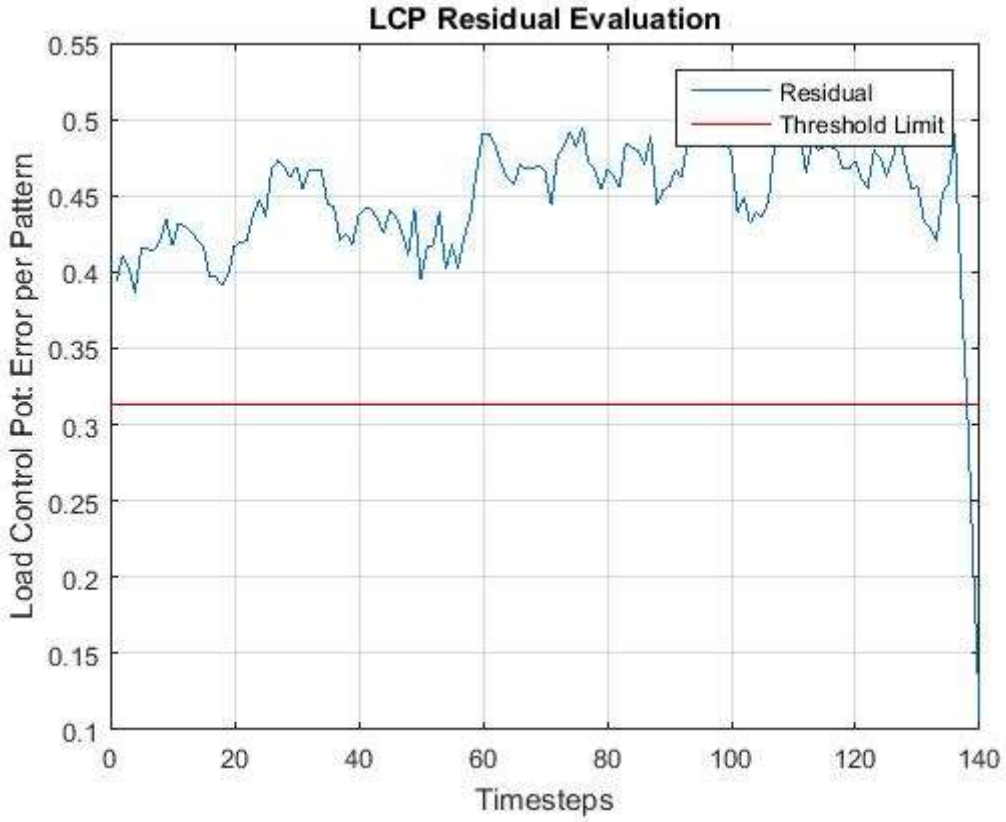


Figure L4.3.4: EXFM Test 3: LCP Test Result

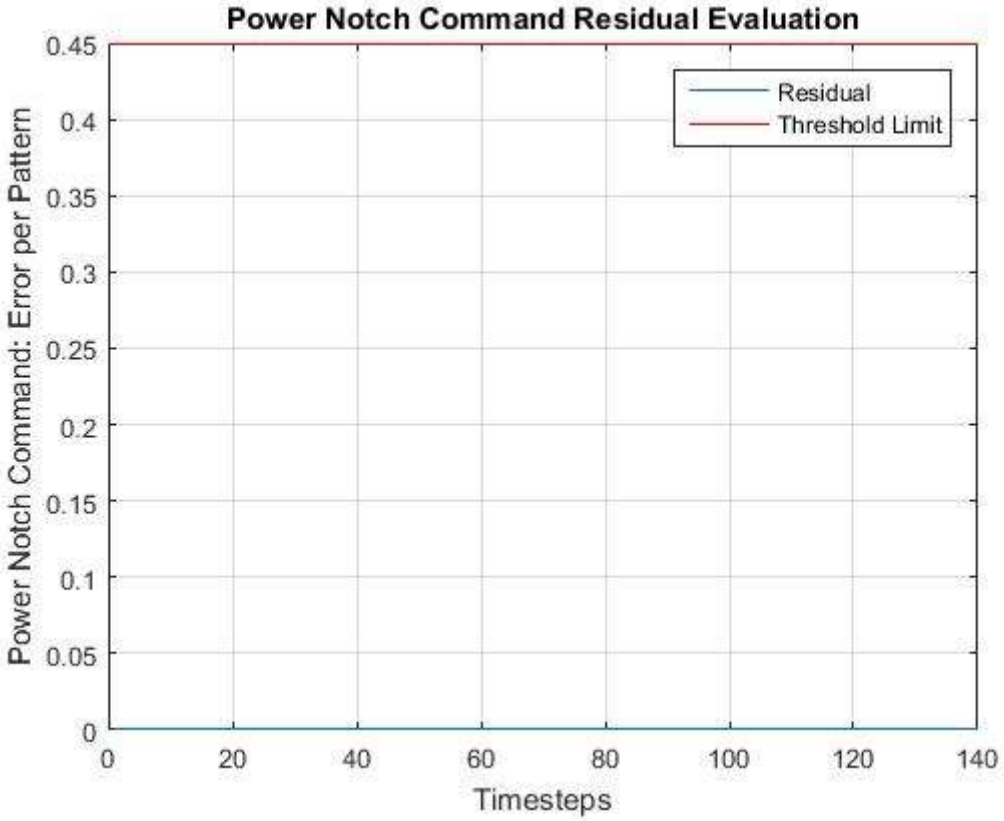


Figure L4.3.5: EXFM Test 3: Power Notch Command Test Result

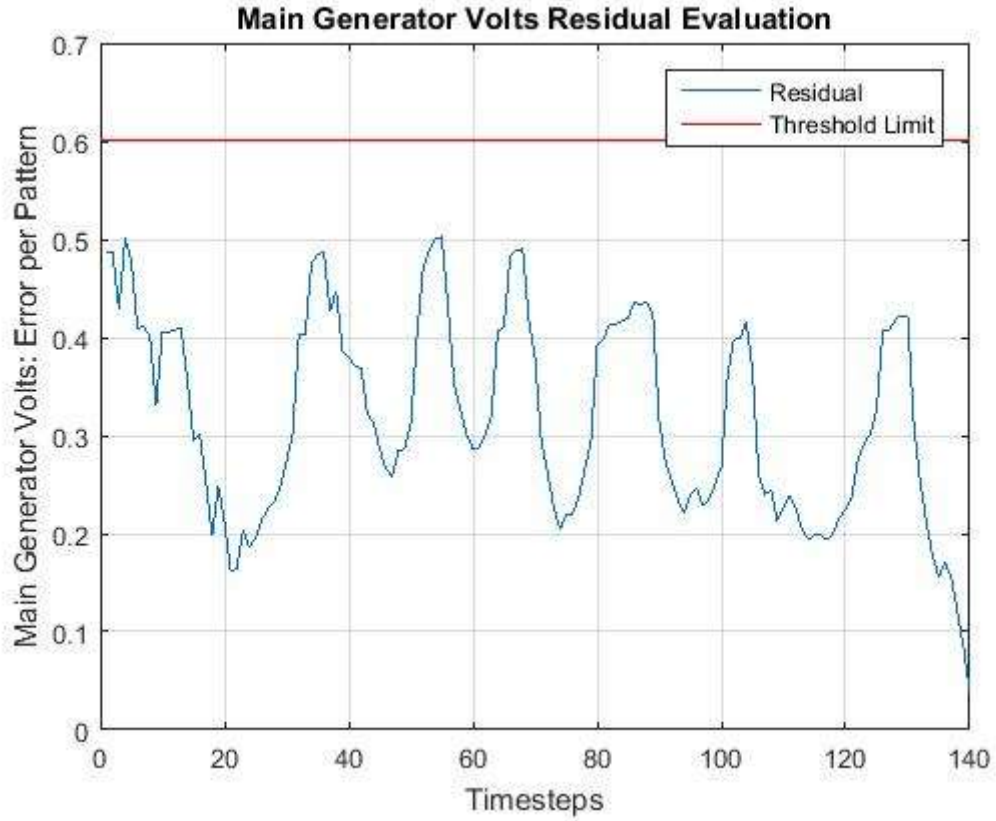


Figure L4.3.6: EXFM Test 3: SCM8 Test Result

L5.1 Normal Test 1 Results

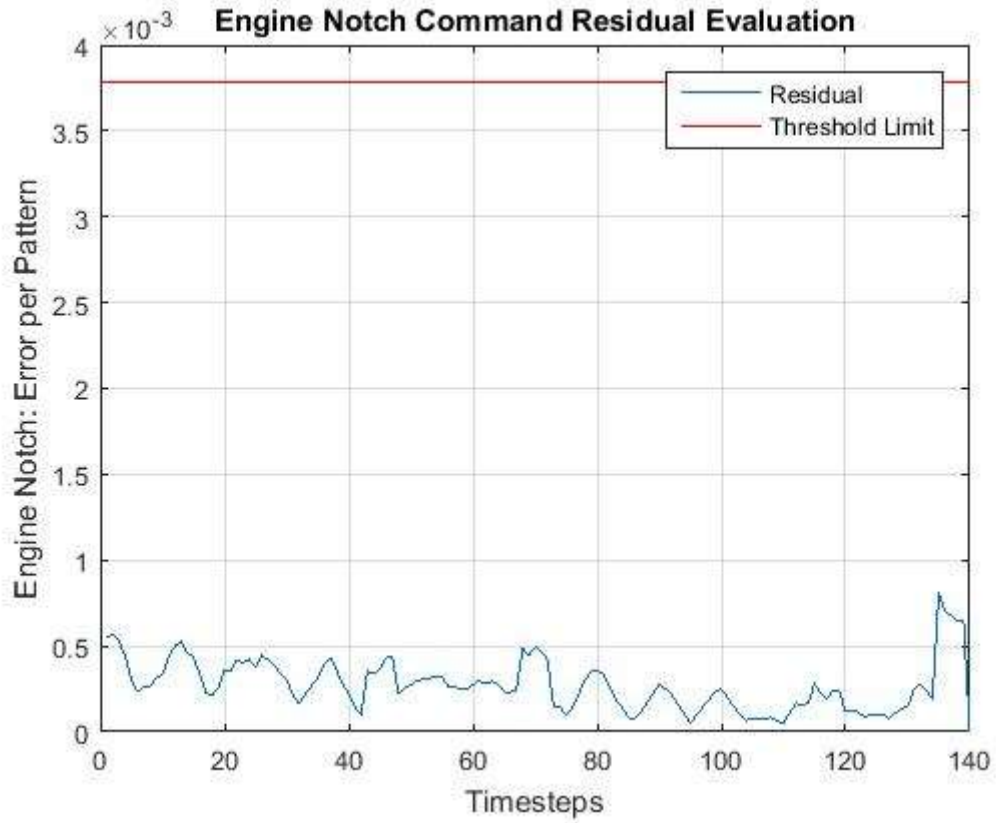


Figure L5.1.1: EXFM Test 1: Engine Notch Command Test Result

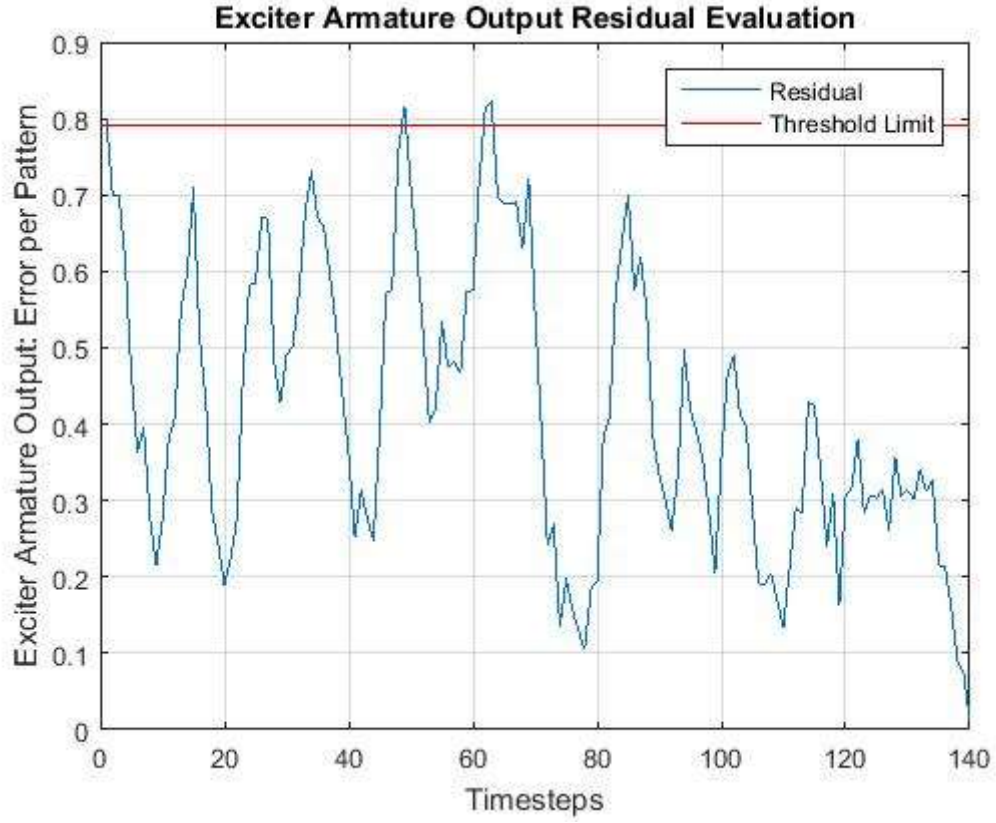


Figure L5.1.2: Normal Test 1: EXACT Test Result

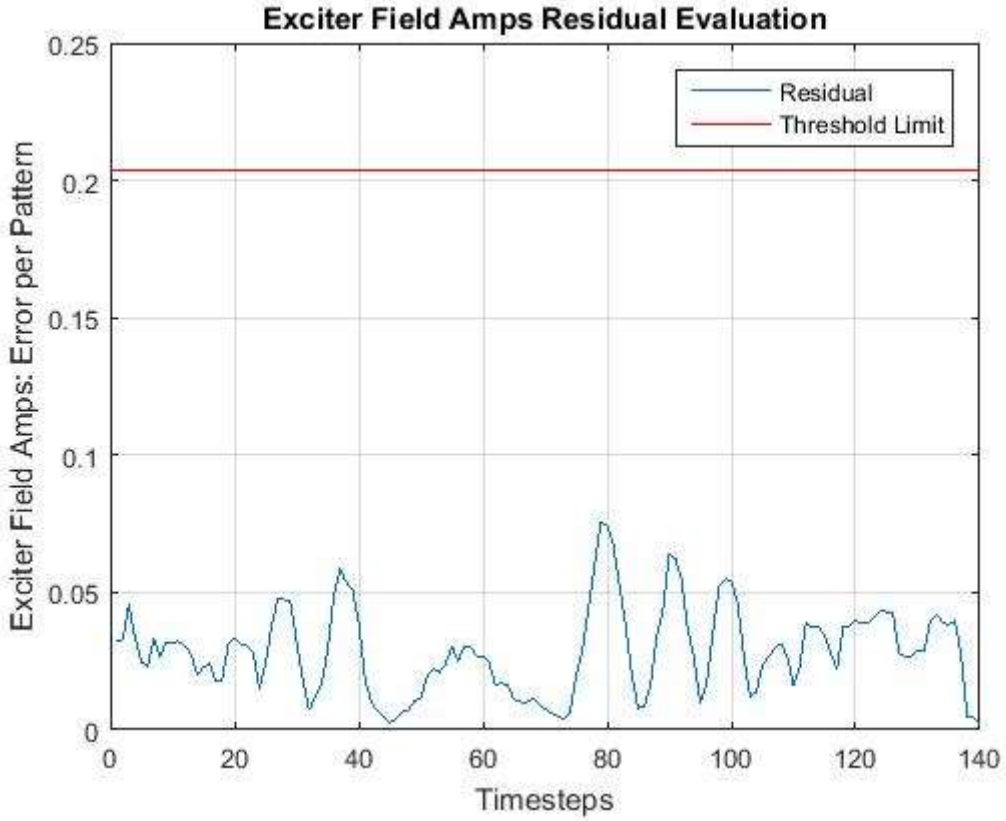


Figure L5.1.3: Normal Test 1: EXFM Test Result

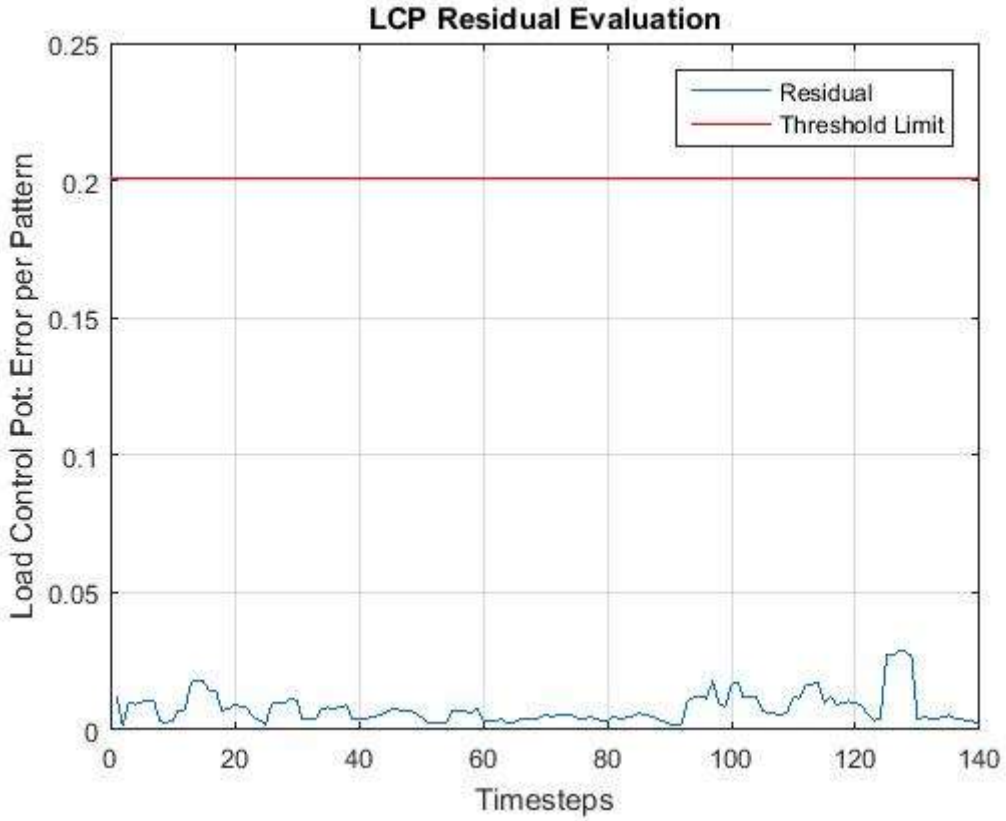


Figure L5.1.4: Normal Test 1: LCP Test Result

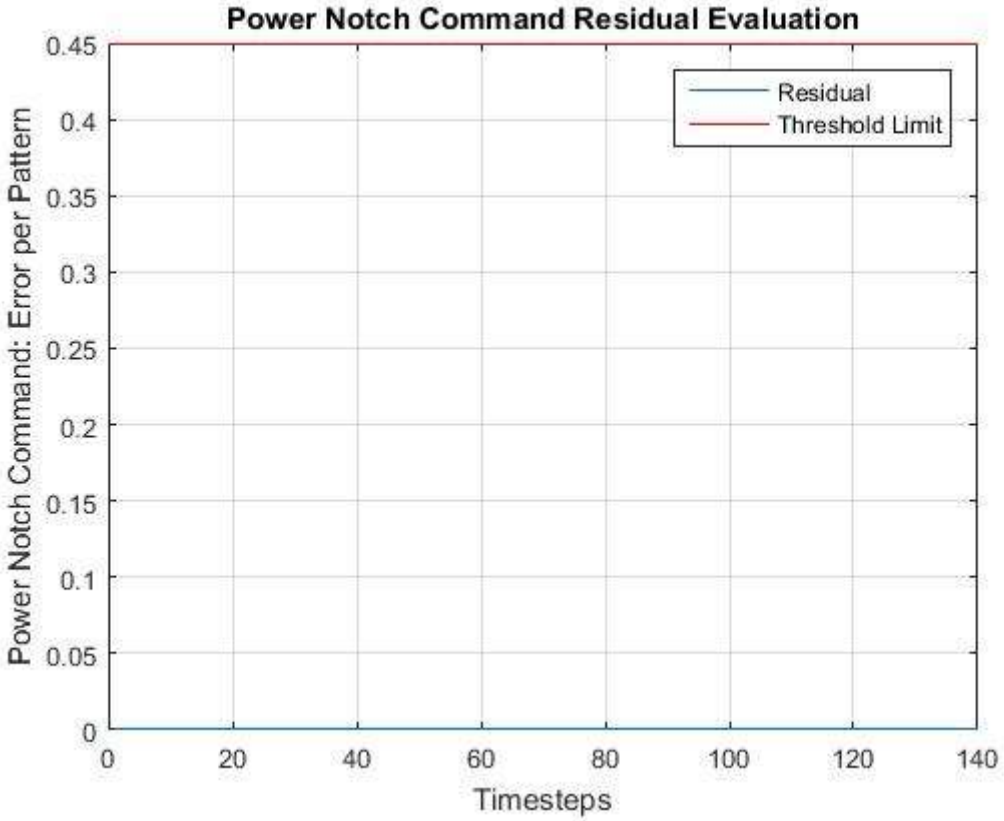


Figure L5.1.5: Normal Test 1: Power Notch Command Test Result

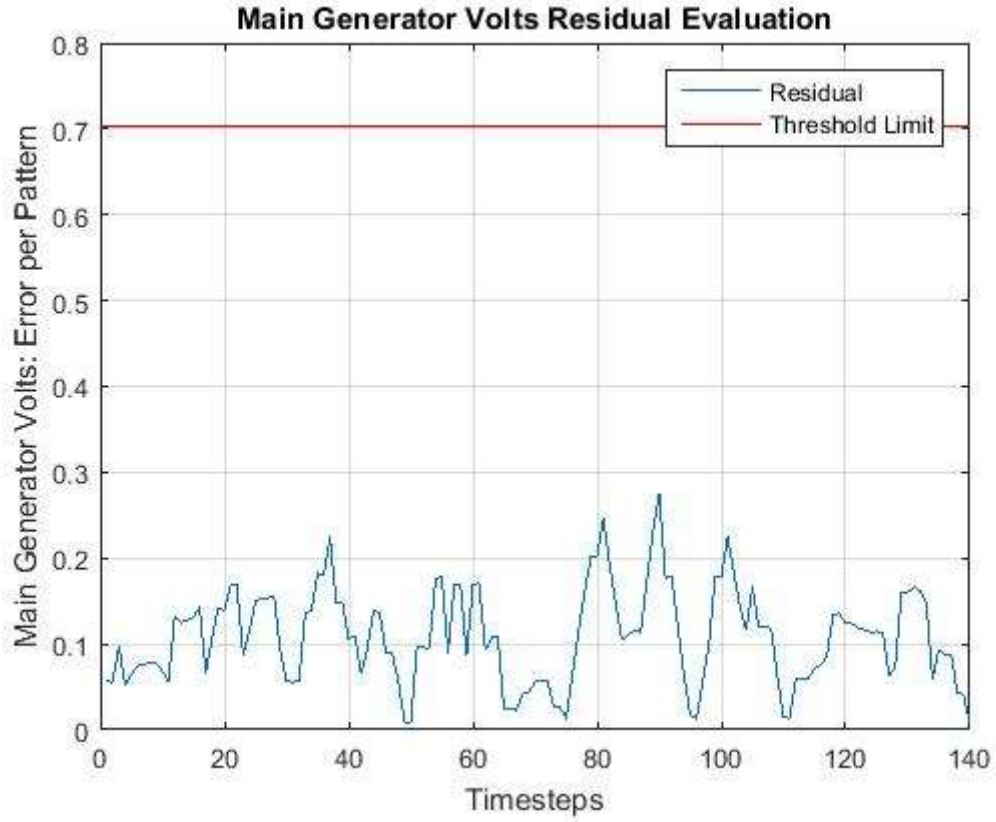


Figure L5.1.6: Normal Test 1: SCM8 Test Result

L5.2 Normal Test 2 Results

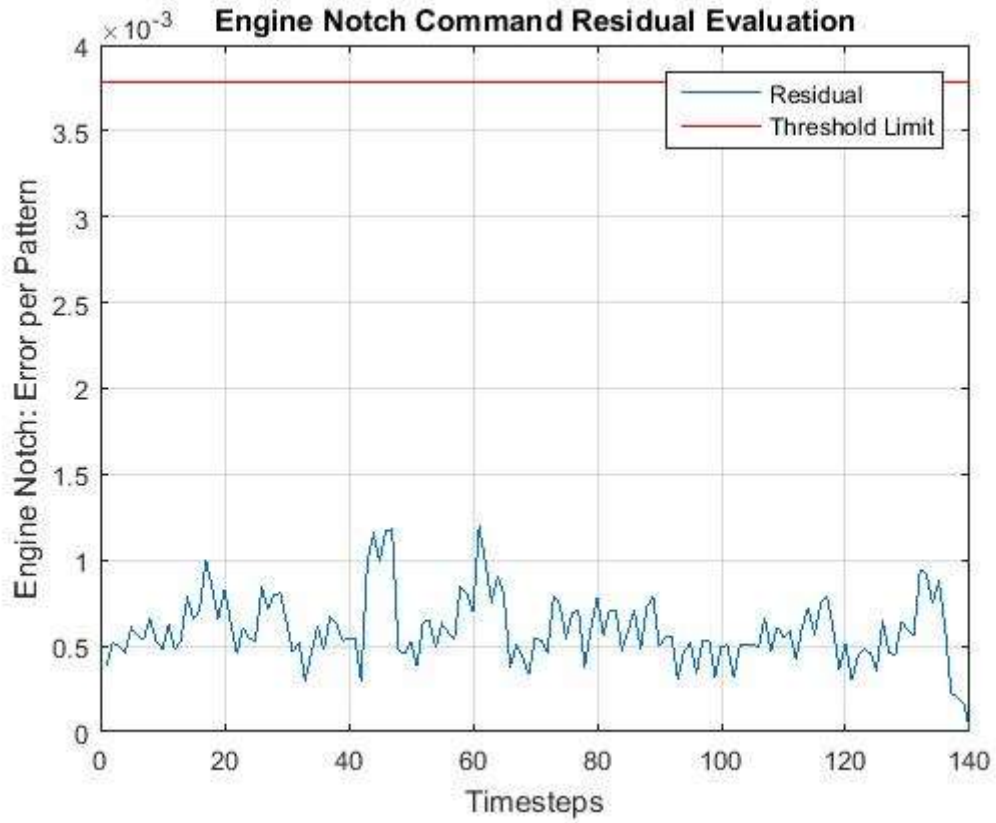


Figure L5.2.1: EXFM Normal 2: Engine Notch Command Test Result

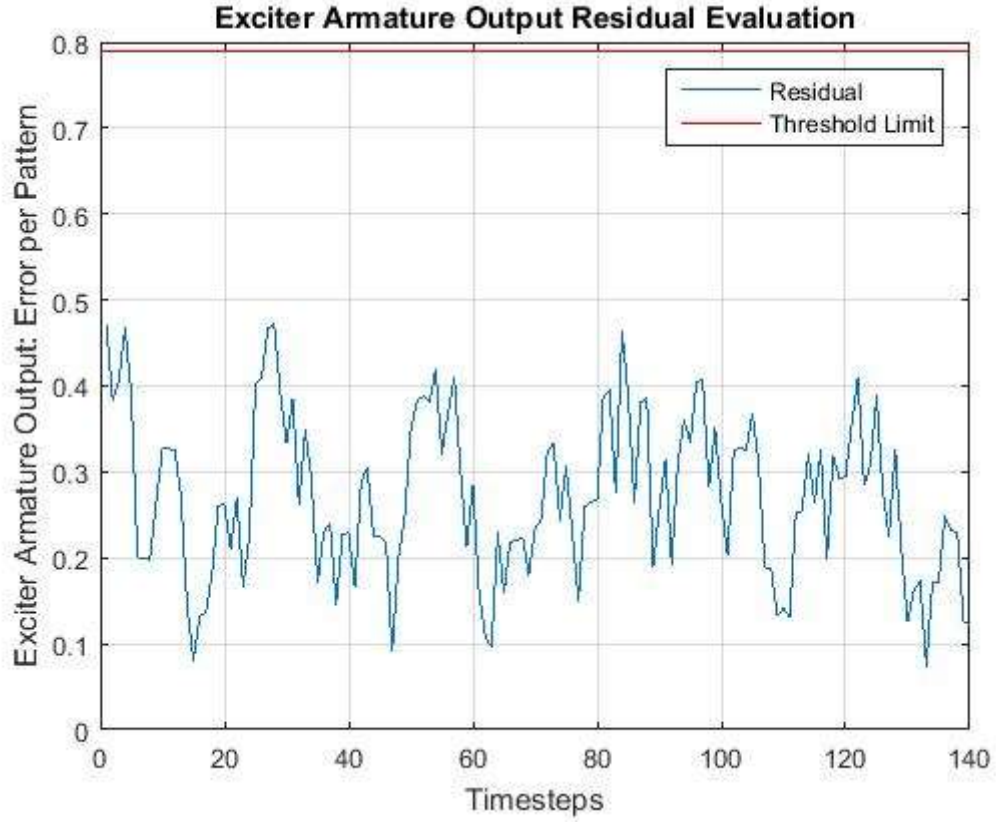


Figure L5.2.2: Normal Test 2: EXACT Test Result

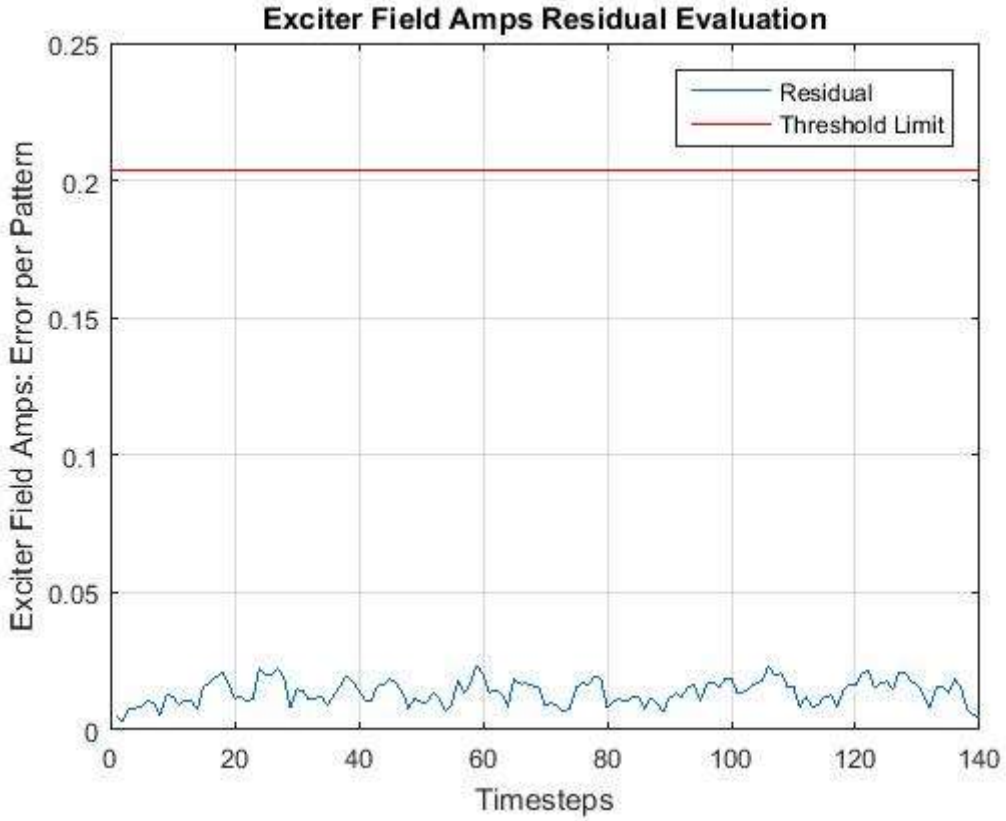


Figure L5.2.3: Normal Test 2: EXFM Test Result

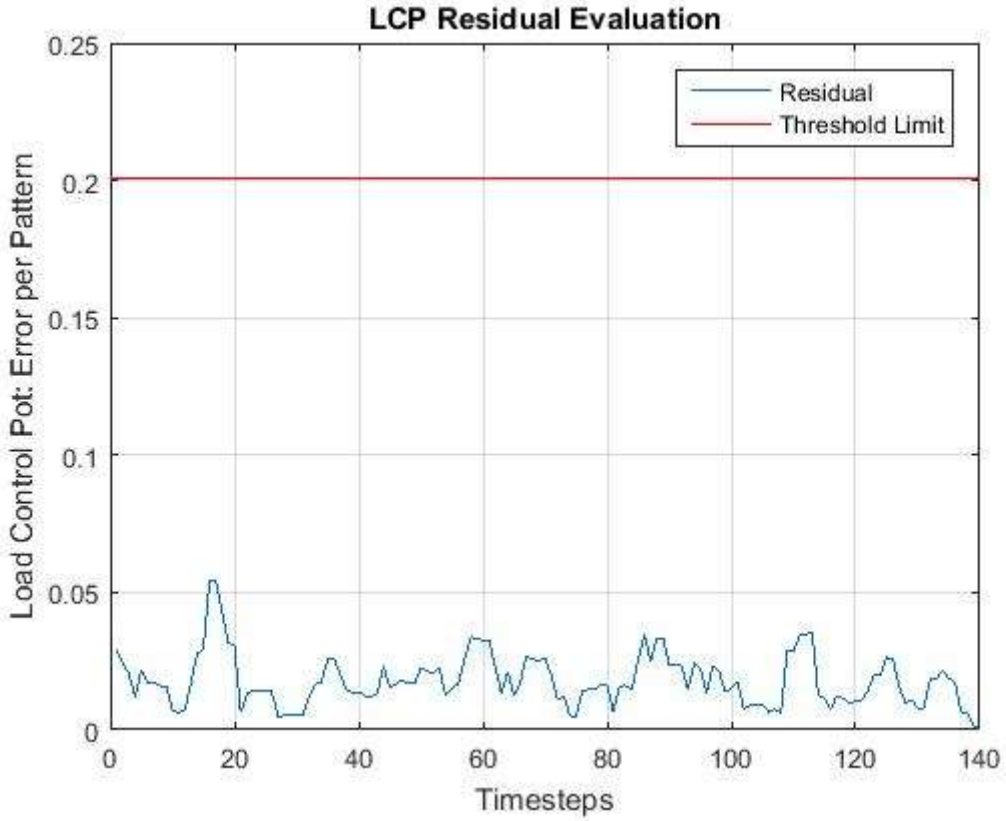


Figure L5.2.4: Normal Test 2: LCP Test Result

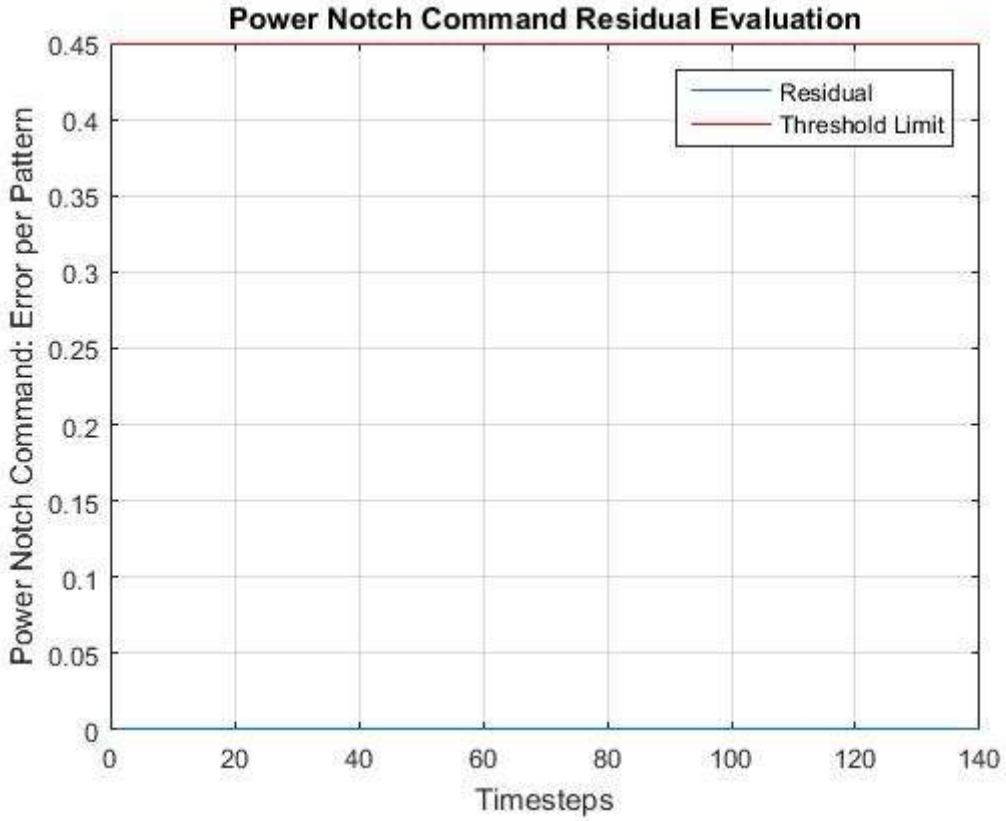


Figure L5.2.5: Normal Test 2: Power Notch Command Test Result

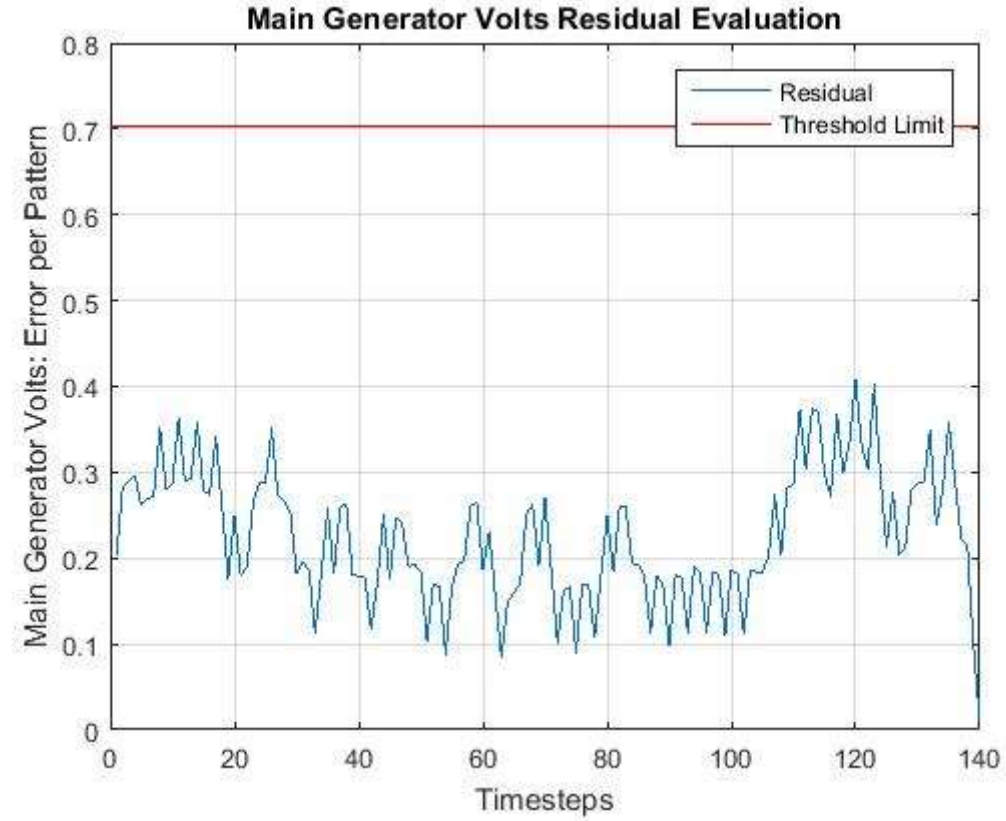


Figure L5.2.6: Normal Test 2: SCM8 Test Result

L5.3 Normal Test 3 Results

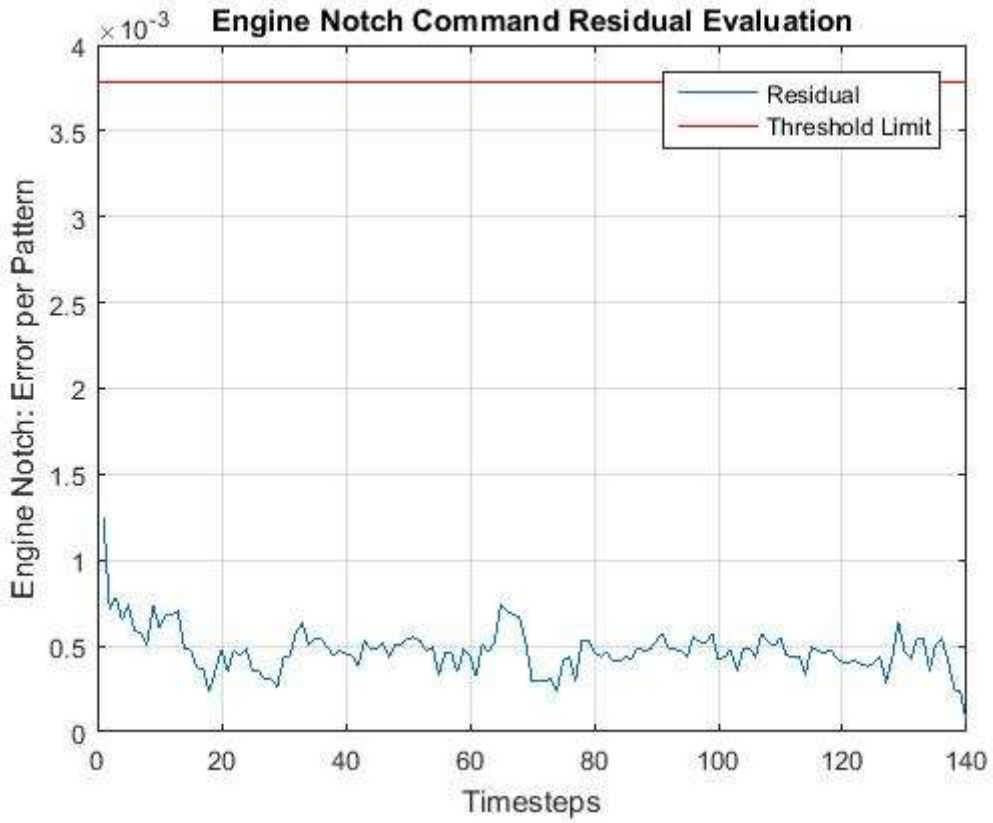


Figure L5.3.1: Normal Test 3: Engine Notch Command Test Result

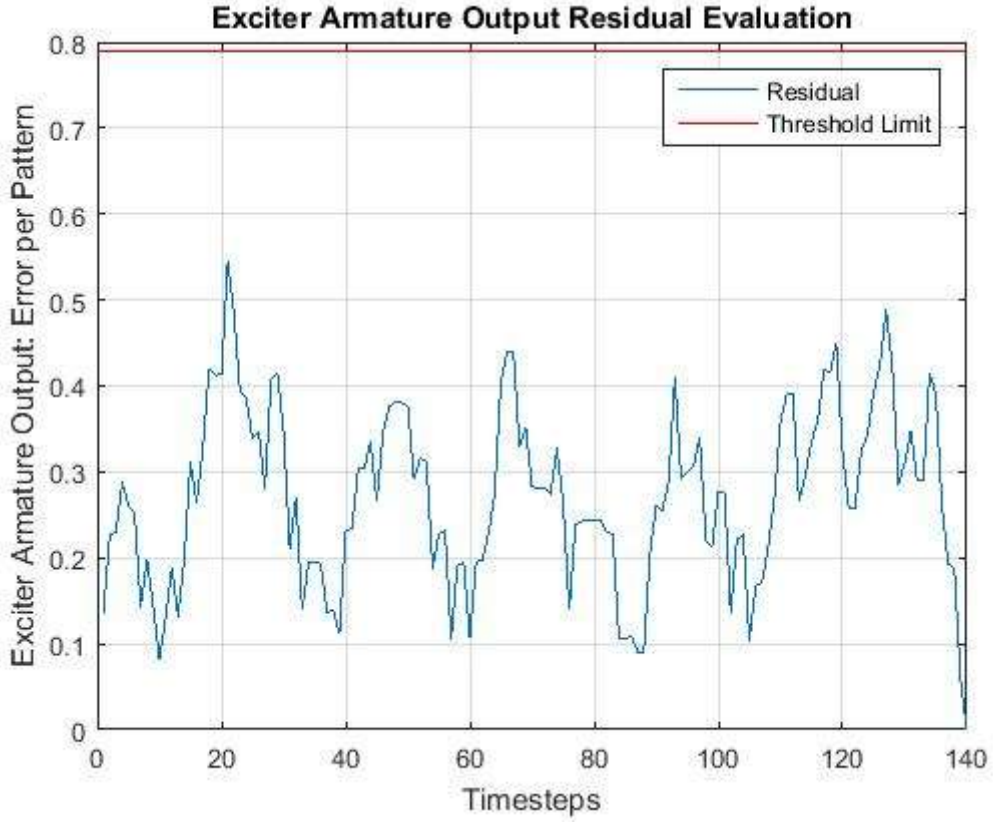


Figure L5.3.2: Normal Test 3: EXACT Test Result

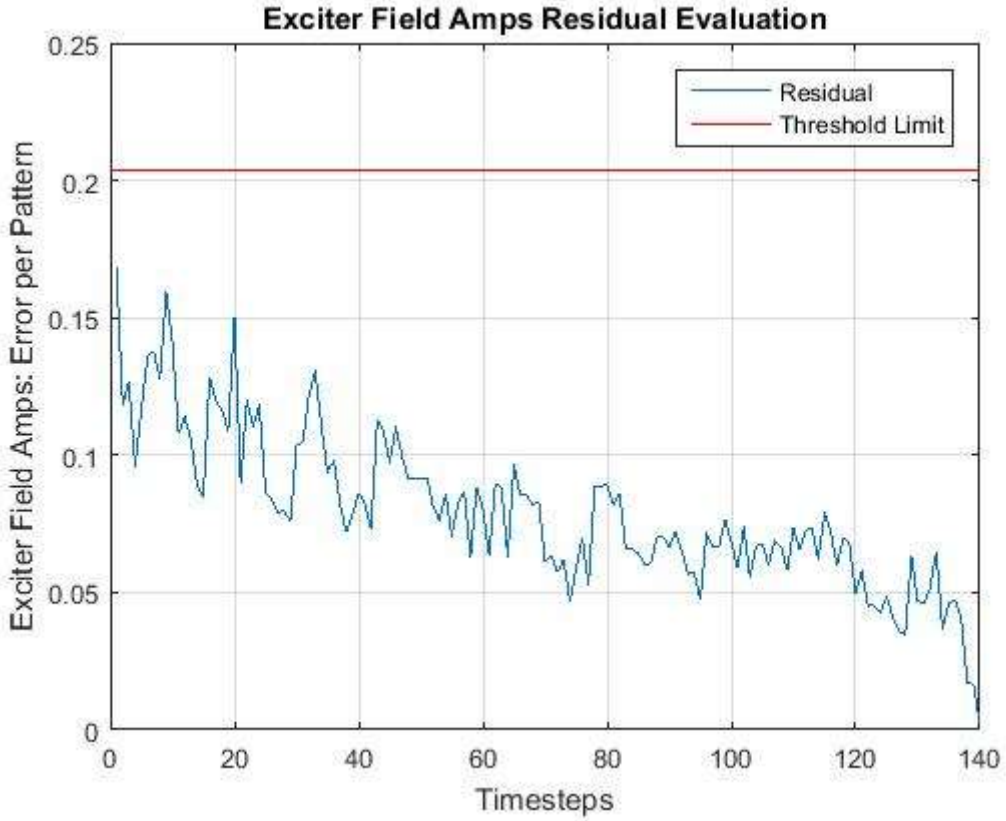


Figure L5.3.3: Normal Test 3: EXFM Test Result

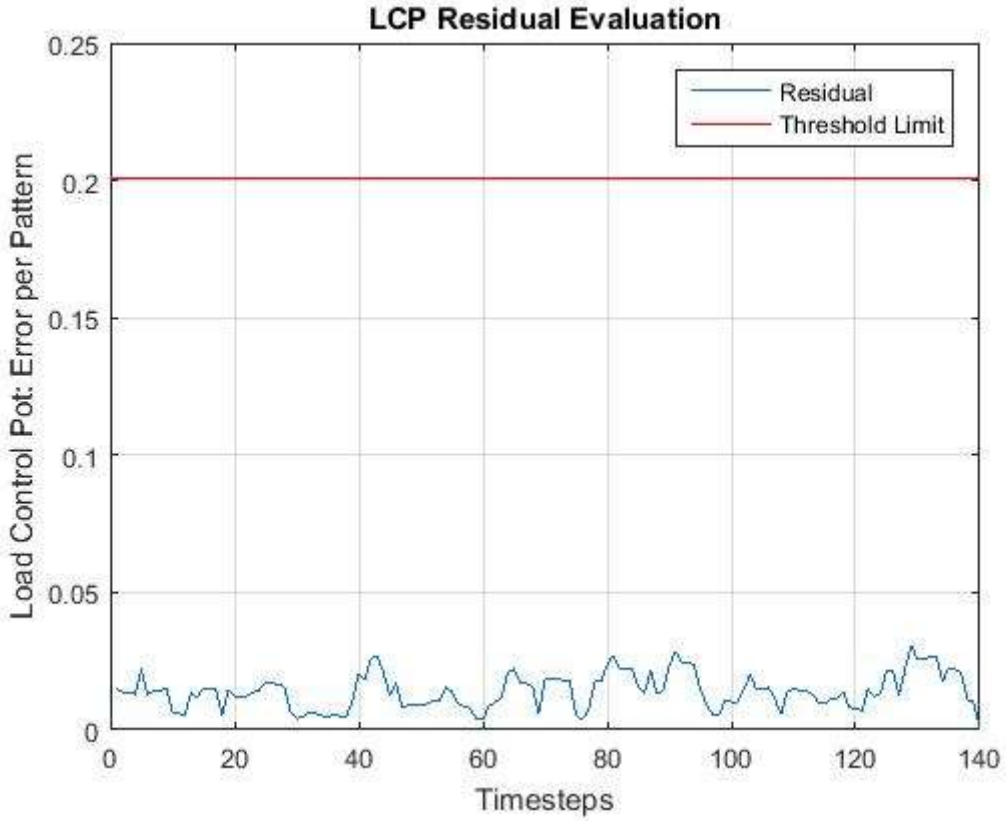


Figure L5.3.4: Normal Test 3: LCP Test Result

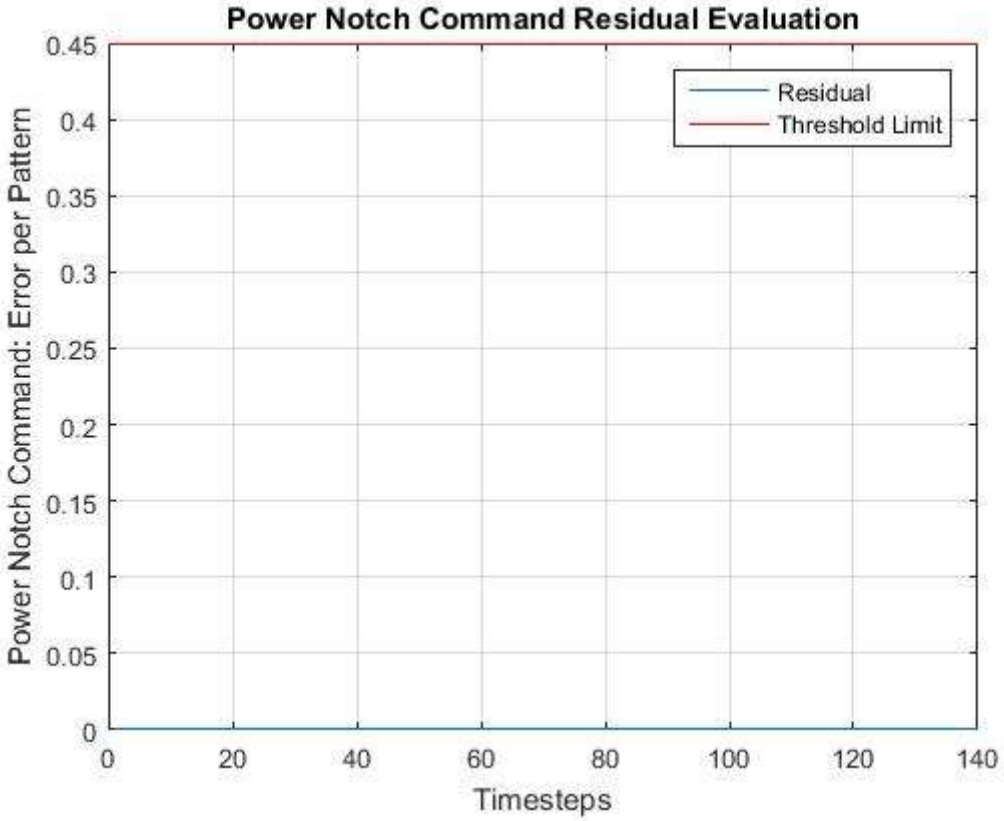


Figure L5.3.5: Normal Test 3: Power Notch Command Test Result

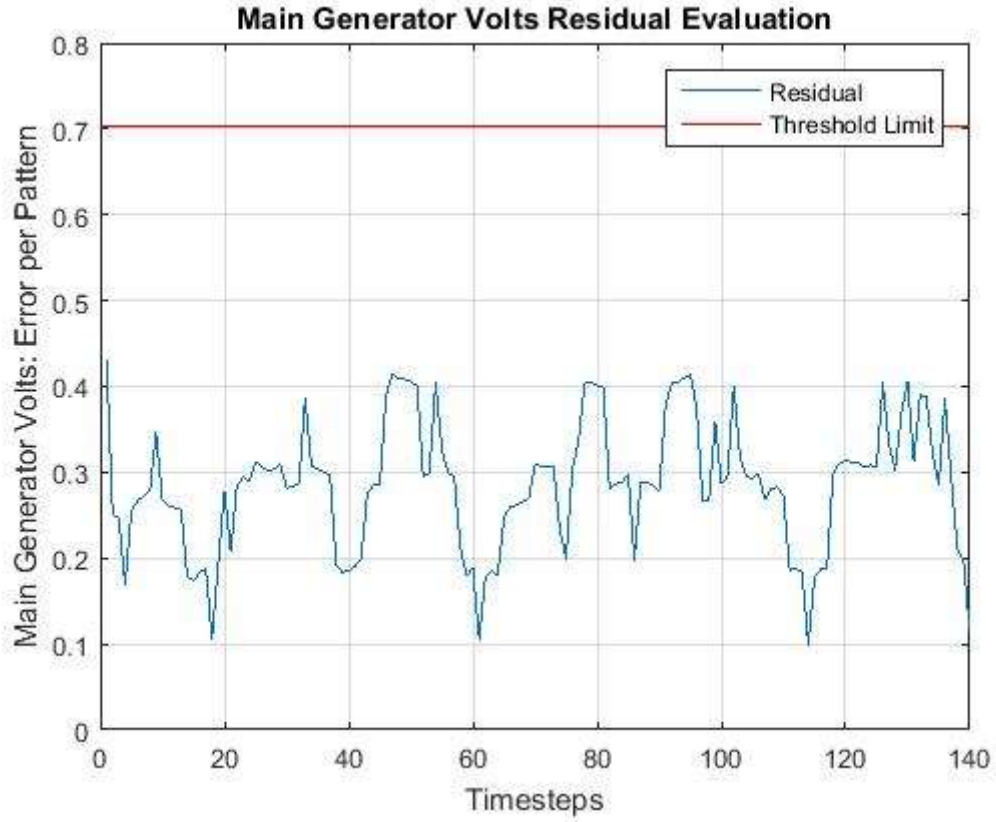


Figure L5.3.6: Normal Test 3: SCM8 Test Result

L5.4 Normal Test 4 Results

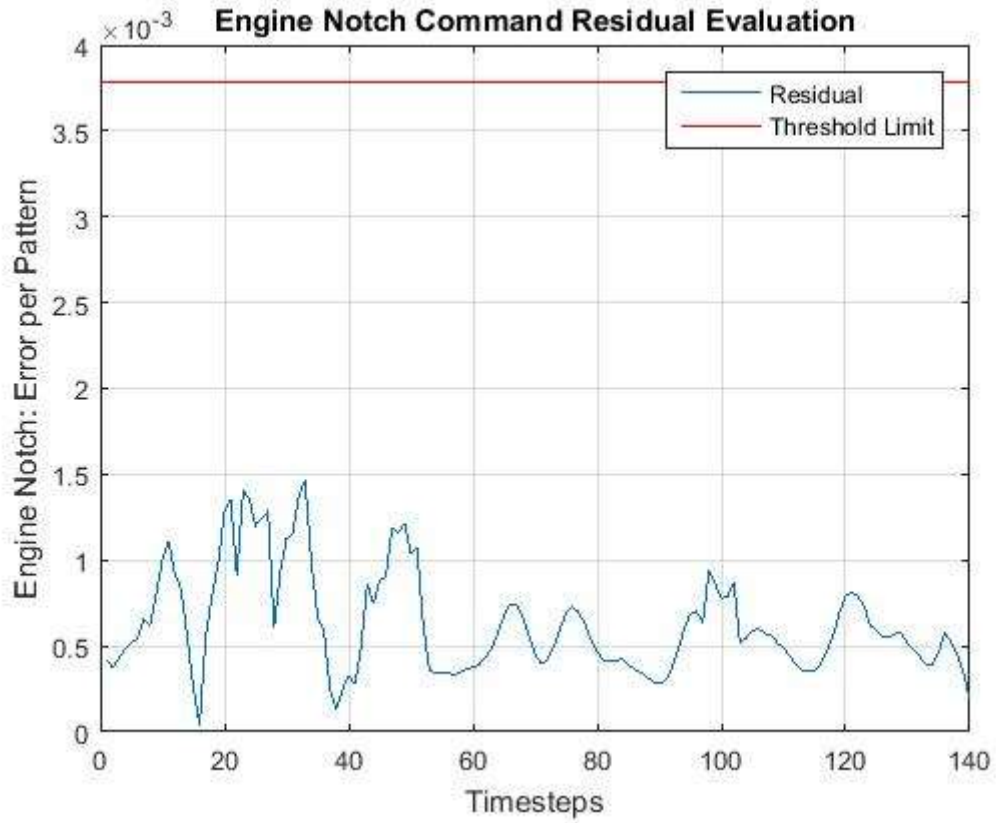


Figure L5.4.1: Normal Test 4: Engine Notch Command Test Result

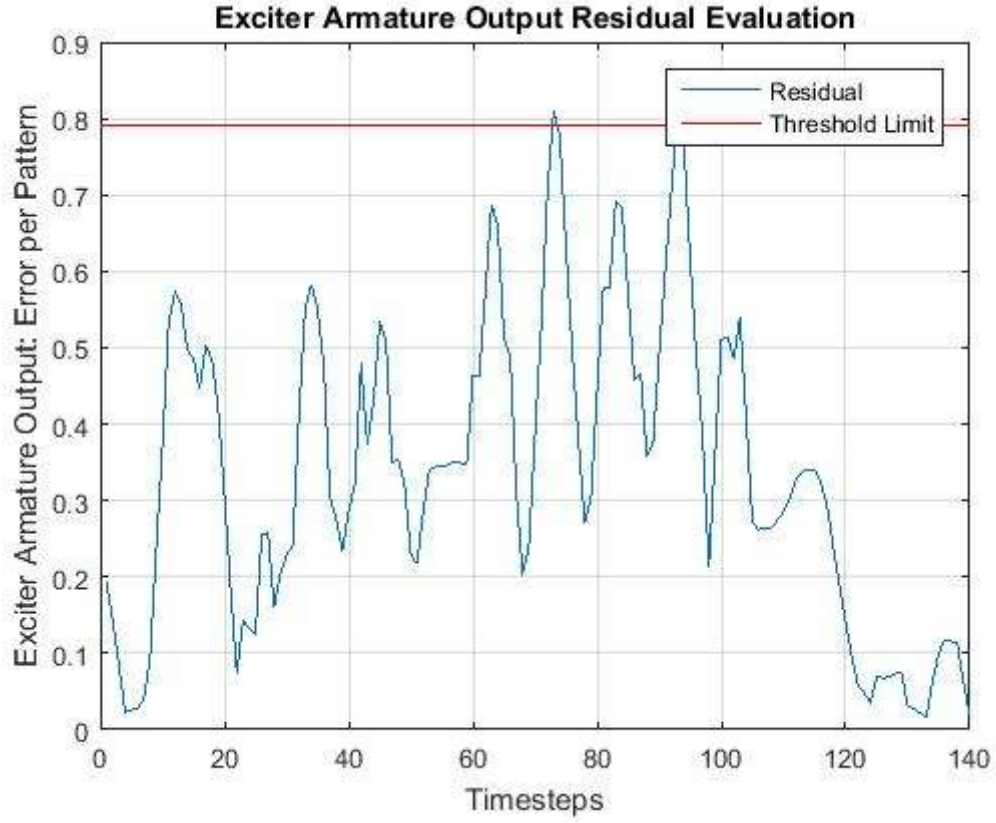


Figure L5.4.2: Normal Test 4: EXACT Test Result

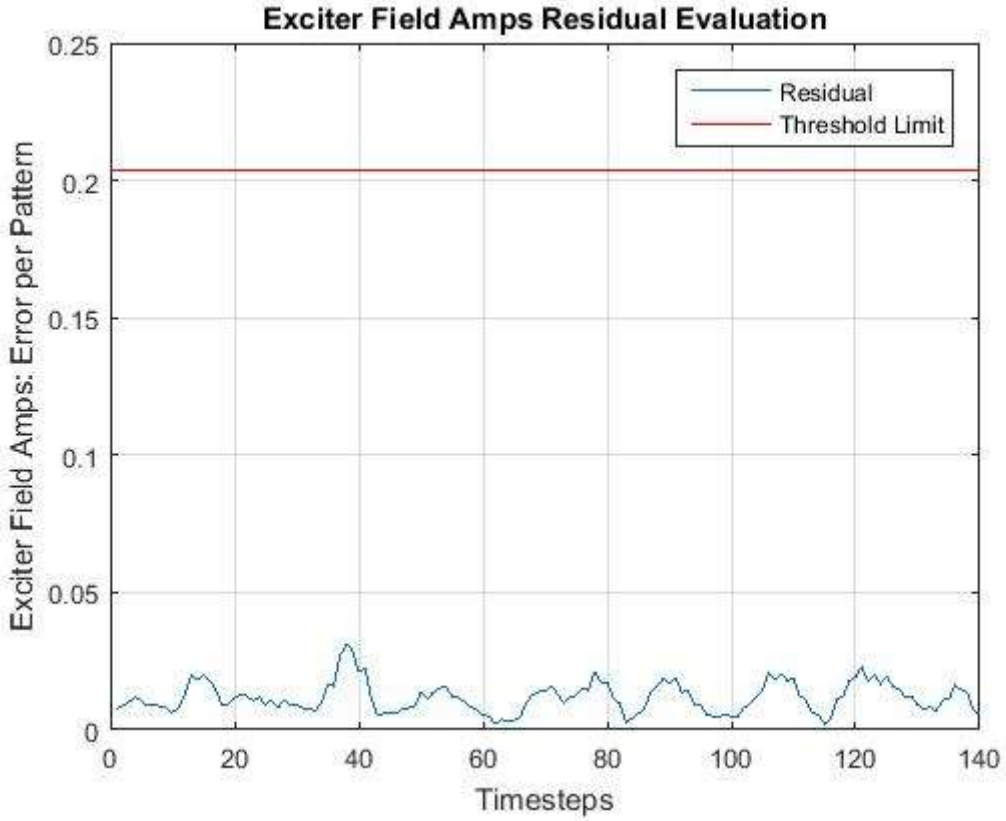


Figure L5.4.3: Normal Test 4: EXFM Test Result

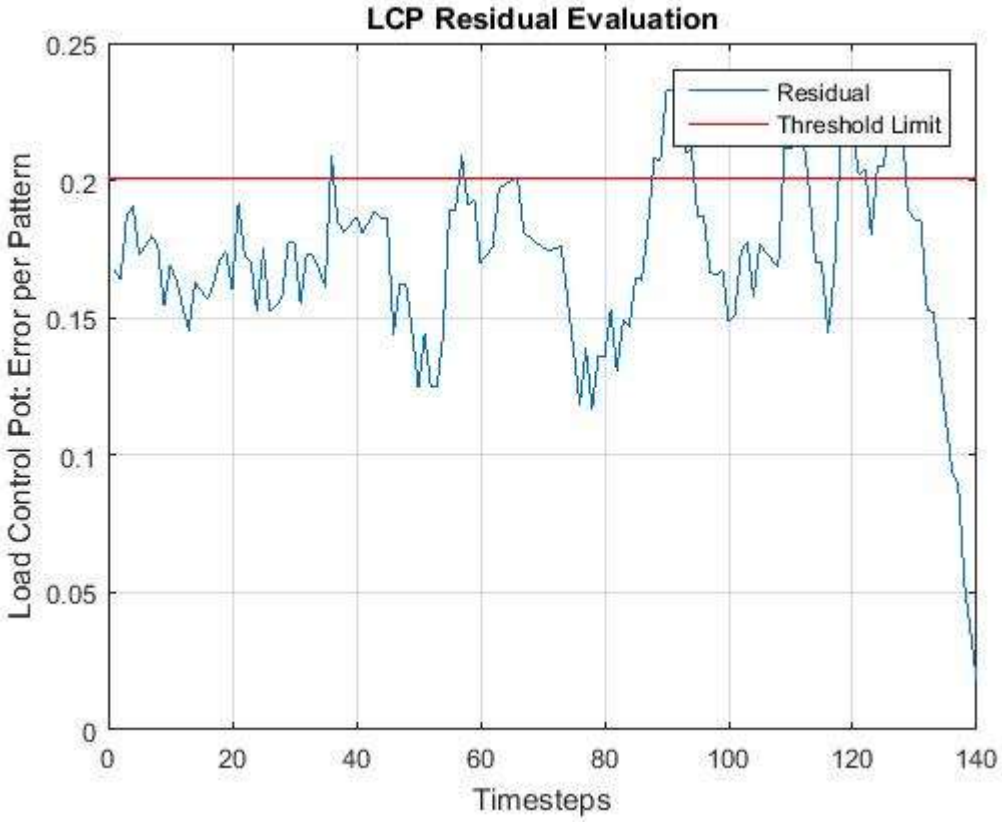


Figure L5.4.4: Normal Test 4: LCP Test Result

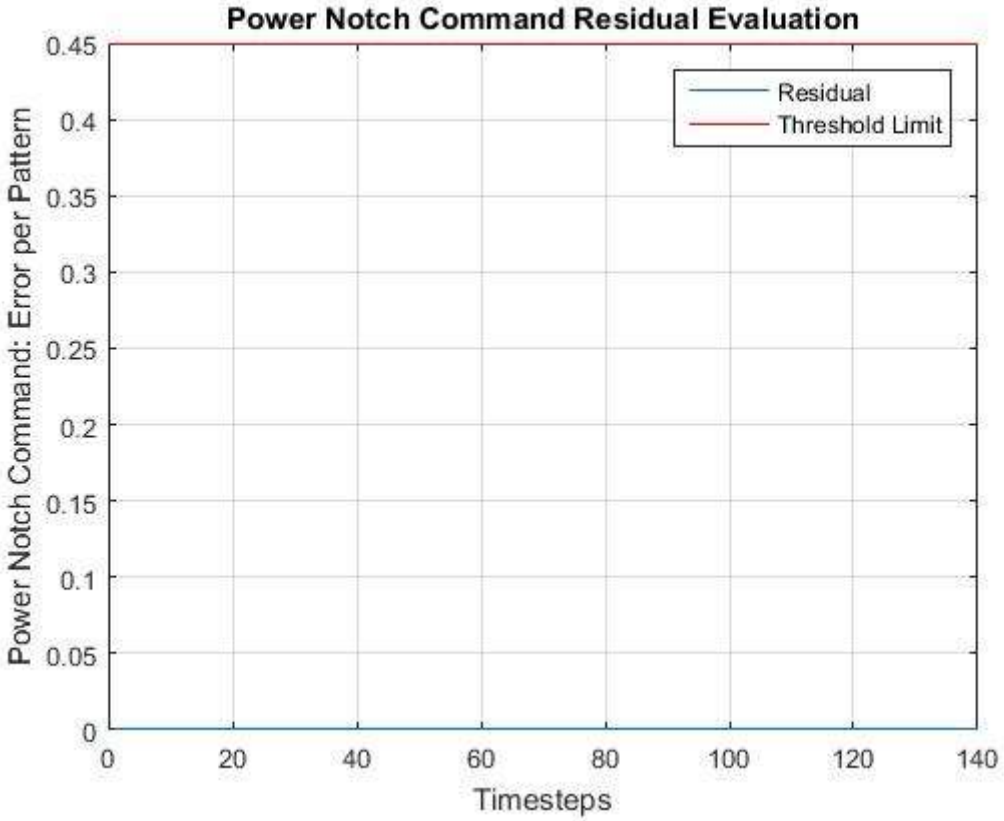


Figure L5.4.5: Normal Test 4: Power Notch Command Test Result

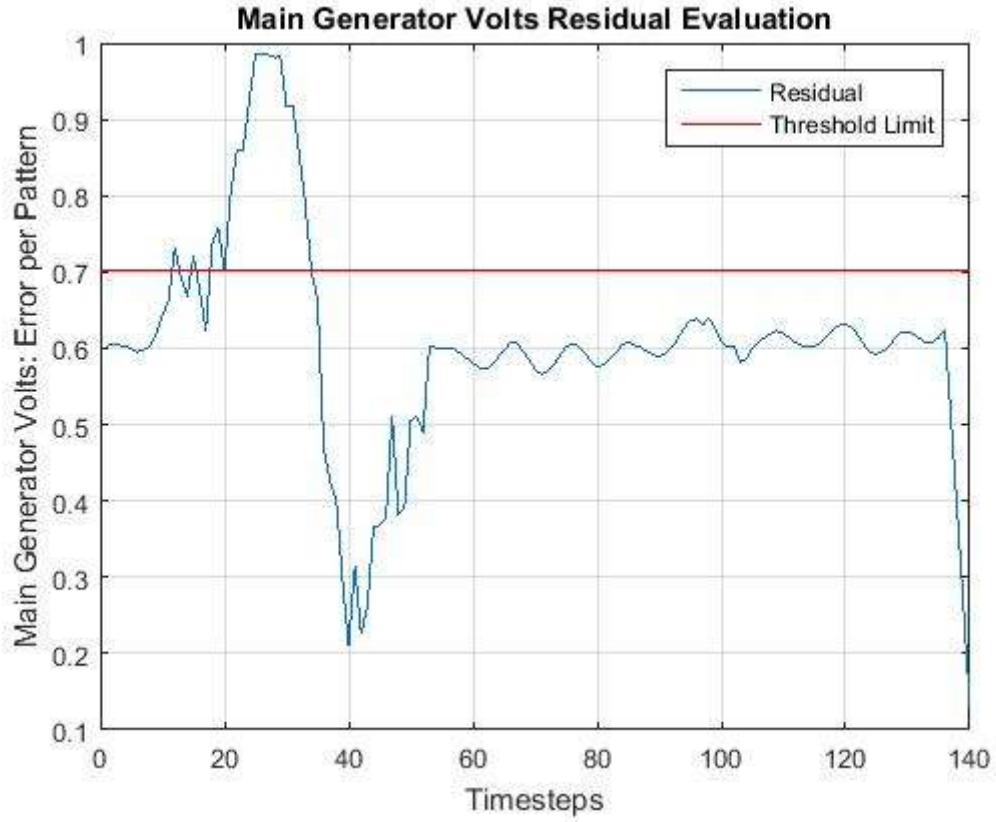


Figure L5.4.6: Normal Test 4: SCM8 Test Result

Appendix M

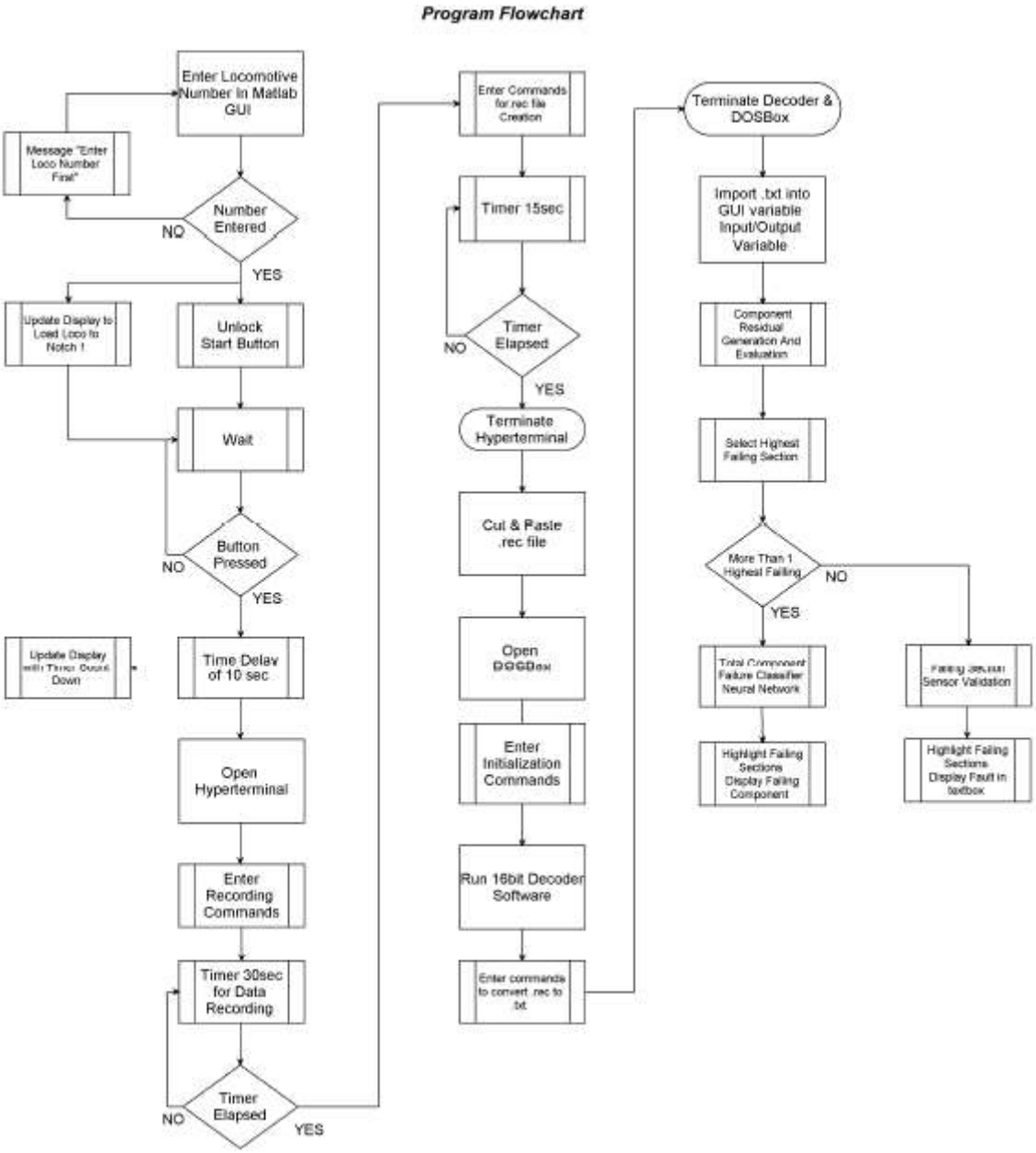


Figure M: GUI Program Flow Chart

Appendix N

N1 Component Neural Network Model Training Source Code

```
% Neural Network Training for Component or sectional Residual Generator
result = zeros(30,3);
x = input';
t = target';
while (n<31)
trainFcn = 'traingdx'; % Gradient Descent with adaptive learning rate and
% momentum backpropagation.
% Create a Feedforward Network
hiddenLayerSize = 10;
net = fitnet(hiddenLayerSize,trainFcn);
% Choose Input and Output Pre/Post-Processing Functions
% For a list of all processing functions type: help nnprocess
net.input.processFcns = {'mapstd'};
net.output.processFcns = {'mapstd'};
% Setup Division of Data for Training, Validation, Testing
% For a list of all data division functions type: help nndivide
net.divideFcn = 'dividerand'; % Divide data randomly
net.divideMode = 'sample'; % Divide up every sample
net.trainParam.epochs = 500000;
net.trainParam.lr = 0.0001;
net.trainParam.lr_inc = 1.0005;
net.trainParam.lr_dec = 0.007;
net.trainParam.mc = 0.9;
net.divideParam.trainRatio = 70/100;
net.divideParam.valRatio = 15/100;
net.divideParam.testRatio = 15/100;
% Performance Function
net.performFcn = 'mae'; % Mean Squared Error
%Plot Functions
net.plotFcns = {'plotperform','plottrainstate','ploterrhist', ...
'plotregression', 'plotfit'};
% Train the Network
[net,tr] = train(net,x,t);
% Test the Network
y = net(x);
e = gsubtract(t,y);
performance = perform(net,t,y);
% Recalculate Training, Validation and Test Performance
trainTargets = t .* tr.trainMask{1};
valTargets = t .* tr.valMask{1};
testTargets = t .* tr.testMask{1};
trainPerformance = perform(net,trainTargets,y);
valPerformance = perform(net,valTargets,y);
testPerformance = perform(net,testTargets,y);
result(n,1) = [trainPerformance];
result(n,2) = [valPerformance];
result(n,3) = [testPerformance];
%Function to check for the best generalization results
if n>1
for e = 1:n-1
if (result(n,3) < result(e,3))
c = c +1;
end
end
end
%Create Function + Plot for best generalization results
```

```
if (c == (n-1))
genFunction(net,'myNeuralNetworkFunction','MatrixOnly','yes');
y = myNeuralNetworkFunction(x);
figure, plotperform(tr), grid on, title('Best Generalization Result')
end
c=0;
n=n+1
end
```

N2 SCM8 Neural Network Model Training Source Code

```
%SCM8 Training Algorithm
x = input';
t = target';
while (n<31)
trainFcn = 'traingdx'; % Gradient Descent with adaptive learning rate and
% momentum backpropagation.
% Create a Feedforward Network
hiddenLayerSize = 24;
net = fitnet(hiddenLayerSize,trainFcn);
% Input and Output Pre/Post-Processing Functions
net.input.processFcns = {'mapstd'};
net.output.processFcns = {'mapstd'};
% Setup Division of Data for Training, Validation, Testing
net.divideFcn = 'dividerand'; % Divide data randomly
net.divideMode = 'sample'; % Divide up every sample
net.trainParam.epochs = 500000;
net.trainParam.lr = 0.0001;
net.trainParam.lr_inc = 1.0005;
net.trainParam.lr_dec = 0.007;
net.trainParam.mc = 0.9;
net.divideParam.trainRatio = 70/100;
net.divideParam.valRatio = 15/100;
net.divideParam.testRatio = 15/100;
%Performance Function
net.performFcn = 'mae'; % Mean Squared Error
% Plot Functions
net.plotFcns = {'plotperform','plottrainstate','ploterrhist', ...
'plotregression', 'plotfit'};
% Train the Network
[net,tr] = train(net,x,t);
% Test the Network
y = net(x);
e = gsubtract(t,y);
performance = perform(net,t,y);
% Recalculate Training, Validation and Test Performance
trainTargets = t .* tr.trainMask{1};
valTargets = t .* tr.valMask{1};
testTargets = t .* tr.testMask{1};
trainPerformance = perform(net,trainTargets,y);
valPerformance = perform(net,valTargets,y);
testPerformance = perform(net,testTargets,y);
result(n,1) = [trainPerformance];
result(n,2) = [valPerformance];
result(n,3) = [testPerformance];
if n>1
for e = 1:n-1
if (result(n,3) < result(e,3))
c = c +1;
end
end
end
if (c == (n-1))
genFunction(net,'myNeuralNetworkFunction','MatrixOnly','yes');
y = myNeuralNetworkFunction(x);
%figure, plotperform(tr), grid on, title('Best Generalization Result')
end
c=0;
end
```

N3 Power Notch Command Neural Network Model Training Source Code

```
%Power Notch Command Training Algorithm
x = input';
t = target';
while (n<31)
trainFcn = 'traingdx'; % Gradient Descent with adaptive learning rate and
% momentum backpropagation.
% Create a Feedforward Network
hiddenLayerSize = 4;
net = fitnet(hiddenLayerSize,trainFcn);
% Input and Output Pre/Post-Processing Functions
net.input.processFcns = {'mapstd'};
net.output.processFcns = {'mapstd'};
% Setup Division of Data for Training, Validation, Testing
net.divideFcn = 'dividerand'; % Divide data randomly
net.divideMode = 'sample'; % Divide up every sample
net.trainParam.epochs = 500000;
net.trainParam.lr = 0.0001;
net.trainParam.lr_inc = 1.0005;
net.trainParam.lr_dec = 0.007;
net.trainParam.mc = 0.9;
net.divideParam.trainRatio = 70/100;
net.divideParam.valRatio = 15/100;
net.divideParam.testRatio = 15/100;
% Performance Function
net.performFcn = 'mae'; % Mean Squared Error
% Plot Functions
net.plotFcns = {'plotperform','plottrainstate','ploterrhist', ...
'plotregression', 'plotfit'};
% Train the Network
[net,tr] = train(net,x,t);
% Test the Network
y = net(x);
e = gsubtract(t,y);
performance = perform(net,t,y);
% Recalculate Training, Validation and Test Performance
trainTargets = t .* tr.trainMask{1};
valTargets = t .* tr.valMask{1};
testTargets = t .* tr.testMask{1};
trainPerformance = perform(net,trainTargets,y);
valPerformance = perform(net,valTargets,y);
testPerformance = perform(net,testTargets,y);
result(n,1) = [trainPerformance];
result(n,2) = [valPerformance];
result(n,3) = [testPerformance];
if n>1
for e = 1:n-1
if (result(n,3) < result(e,3))
% Generate a matrix-only MATLAB function for neural network code
% generation with MATLAB Coder tools.
c = c +1;
end
end
end
if (c == (n-1))
genFunction(net,'myNeuralNetworkFunction','MatrixOnly','yes');
y = myNeuralNetworkFunction(x);
%figure, plotperform(tr), grid on, title('Best Generalization Result')
end
c=0;
end
```

N4 LCP Neural Network Model Training Source Code

```
%LCP Training Algorithm
x = input';
t = target';
while (n<31)
trainFcn = 'traingdx'; % Gradient Descent with adaptive learning rate and
% momentum backpropagation.
% Create a Feedforward Network
hiddenLayerSize = 15;
net = fitnet(hiddenLayerSize,trainFcn);
% Input and Output Pre/Post-Processing Functions
net.input.processFcns = {'mapstd'};
net.output.processFcns = {'mapstd'};
% Setup Division of Data for Training, Validation, Testing
net.divideFcn = 'dividerand'; % Divide data randomly
net.divideMode = 'sample'; % Divide up every sample
net.trainParam.epochs = 500000;
net.trainParam.lr = 0.0001;
net.trainParam.lr_inc = 1.0005;
net.trainParam.lr_dec = 0.007;
net.trainParam.mc = 0.9;
net.divideParam.trainRatio = 70/100;
net.divideParam.valRatio = 15/100;
net.divideParam.testRatio = 15/100;
% Performance Function
net.performFcn = 'mae'; % Mean Squared Error
% Plot Functions
net.plotFcns = {'plotperform','plottrainstate','ploterrhist', ...
'plotregression', 'plotfit'};
% Train the Network
[net,tr] = train(net,x,t);
% Test the Network
y = net(x);
e = gsubtract(t,y);
performance = perform(net,t,y);
% Recalculate Training, Validation and Test Performance
trainTargets = t .* tr.trainMask{1};
valTargets = t .* tr.valMask{1};
testTargets = t .* tr.testMask{1};
trainPerformance = perform(net,trainTargets,y);
valPerformance = perform(net,valTargets,y);
testPerformance = perform(net,testTargets,y);
result(n,1) = [trainPerformance];
result(n,2) = [valPerformance];
result(n,3) = [testPerformance];
if n>1
for e = 1:n-1
if (result(n,3) < result(e,3))
c = c + 1;
end
end
end
if (c == (n-1))
genFunction(net,'myNeuralNetworkFunction','MatrixOnly','yes');
y = myNeuralNetworkFunction(x);
%figure, plotperform(tr), grid on, title('Best Generalization Result')
end
c=0;
end
```

N5 EXFM Neural Network Model Training Source Code

```
%EXFM Training Algorithm
x = input';
t = target';
while (n<31)
trainFcn = 'traingdx'; % Gradient Descent with adaptive learning rate and
% momentum backpropagation.
% Create a Feedforward Network
hiddenLayerSize = 15;
net = fitnet(hiddenLayerSize,trainFcn);
% Input and Output Pre/Post-Processing Functions
net.input.processFcns = {'mapstd'};
net.output.processFcns = {'mapstd'};
% Setup Division of Data for Training, Validation, Testing
net.divideFcn = 'dividerand'; % Divide data randomly
net.divideMode = 'sample'; % Divide up every sample
net.trainParam.epochs = 500000;
net.trainParam.lr = 0.0001;
net.trainParam.lr_inc = 1.0005;
net.trainParam.lr_dec = 0.007;
net.trainParam.mc = 0.9;
net.divideParam.trainRatio = 70/100;
net.divideParam.valRatio = 15/100;
net.divideParam.testRatio = 15/100;
% Performance Function
net.performFcn = 'mae'; % Mean Squared Error
% Plot Functions
net.plotFcns = {'plotperform','plottrainstate','ploterrhist', ...
'plotregression', 'plotfit'};
% Train the Network
[net,tr] = train(net,x,t);
% Test the Network
y = net(x);
e = gsubtract(t,y);
performance = perform(net,t,y);
% Recalculate Training, Validation and Test Performance
trainTargets = t .* tr.trainMask{1};
valTargets = t .* tr.valMask{1};
testTargets = t .* tr.testMask{1};
trainPerformance = perform(net,trainTargets,y);
valPerformance = perform(net,valTargets,y);
testPerformance = perform(net,testTargets,y);
result(n,1) = [trainPerformance];
result(n,2) = [valPerformance];
result(n,3) = [testPerformance];
if n>1
for e = 1:n-1
if (result(n,3) < result(e,3))
c = c +1;
end
end
end
if (c == (n-1))
genFunction(net,'myNeuralNetworkFunction','MatrixOnly','yes');
y = myNeuralNetworkFunction(x);
%figure, plotperform(tr), grid on, title('Best Generalization Result')
end
c=0;
end
```

N6 EXACT Neural Network Model Training Source Code

```
%EXACT Training Algorithm
x = input';
t = target';
while (n<31)
trainFcn = 'traingdx'; % Gradient Descent with adaptive learning rate and
% momentum backpropagation.
% Create a Feedforward Network
hiddenLayerSize = 10;
net = fitnet(hiddenLayerSize,trainFcn);
%Input and Output Pre/Post-Processing Functions
net.input.processFcns = {'mapstd'};
net.output.processFcns = {'mapstd'};
% Setup Division of Data for Training, Validation, Testing
net.divideFcn = 'dividerand'; % Divide data randomly
net.divideMode = 'sample'; % Divide up every sample
net.trainParam.epochs = 500000;
net.trainParam.lr = 0.0001;
net.trainParam.lr_inc = 1.0005;
net.trainParam.lr_dec = 0.007;
net.trainParam.mc = 0.9;
net.divideParam.trainRatio = 70/100;
net.divideParam.valRatio = 15/100;
net.divideParam.testRatio = 15/100;
% Performance Function
net.performFcn = 'mae'; % Mean Squared Error
% Plot Functions
net.plotFcns = {'plotperform','plottrainstate','ploterrhist', ...
'plotregression', 'plotfit'};
% Train the Network
[net,tr] = train(net,x,t);
% Test the Network
y = net(x);
e = gsubtract(t,y);
performance = perform(net,t,y);
% Recalculate Training, Validation and Test Performance
trainTargets = t .* tr.trainMask{1};
valTargets = t .* tr.valMask{1};
testTargets = t .* tr.testMask{1};
trainPerformance = perform(net,trainTargets,y);
valPerformance = perform(net,valTargets,y);
testPerformance = perform(net,testTargets,y);
result(n,1) = [trainPerformance];
result(n,2) = [valPerformance];
result(n,3) = [testPerformance];
if n>1
for e = 1:n-1
if (result(n,3) < result(e,3))
c = c +1;
end
end
end
if (c == (n-1))
genFunction(net,'myNeuralNetworkFunction','MatrixOnly','yes');
y = myNeuralNetworkFunction(x);
%figure, plotperform(tr), grid on, title('Best Generalization Result')
end
c=0;
end
end
```

N7 Engine Notch Command Neural Network Model Training Source Code

```
%Engine Notch Command Training Algorithm
x = input';
t = target';
while (n<31)
trainFcn = 'traingdx'; % Gradient Descent with adaptive learning rate and
% momentum backpropagation.
% Create a Feedforward Network
hiddenLayerSize = 2;
net = fitnet(hiddenLayerSize,trainFcn);
% Input and Output Pre/Post-Processing Functions
net.input.processFcns = {'mapstd'};
net.output.processFcns = {'mapstd'};
% Setup Division of Data for Training, Validation, Testing
net.divideFcn = 'dividerand'; % Divide data randomly
net.divideMode = 'sample'; % Divide up every sample
net.trainParam.epochs = 500000;
net.trainParam.lr = 0.0001;
net.trainParam.lr_inc = 1.0005;
net.trainParam.lr_dec = 0.007;
net.trainParam.mc = 0.9;
net.divideParam.trainRatio = 70/100;
net.divideParam.valRatio = 15/100;
net.divideParam.testRatio = 15/100;
% Performance Function
net.performFcn = 'mae'; % Mean Squared Error
% Plot Functions
net.plotFcns = {'plotperform','plottrainstate','ploterrhist', ...
'plotregression', 'plotfit'};
% Train the Network
[net,tr] = train(net,x,t);
% Test the Network
y = net(x);
e = gsubtract(t,y);
performance = perform(net,t,y);
% Recalculate Training, Validation and Test Performance
trainTargets = t .* tr.trainMask{1};
valTargets = t .* tr.valMask{1};
testTargets = t .* tr.testMask{1};
trainPerformance = perform(net,trainTargets,y);
valPerformance = perform(net,valTargets,y);
testPerformance = perform(net,testTargets,y);
result(n,1) = [trainPerformance];
result(n,2) = [valPerformance];
result(n,3) = [testPerformance];
if n>1
for e = 1:n-1
if (result(n,3) < result(e,3))
c = c +1;
end
end
end
if (c == (n-1))
genFunction(net,'myNeuralNetworkFunction','MatrixOnly','yes');
y = myNeuralNetworkFunction(x);
%figure, plotperform(tr), grid on, title('Best Generalization Result')
end
c=0;
```

Appendix O

O1 Component Residual Generation and Evaluation Source Code

```
%Component Residual Generation and Evaluation Source Code
r=140;
count =zeros(6,3);
%Read In Locomotives recorded Data from Data.txt file
filename = 'C:\Decoder\DATA.txt';
M = dlmread(filename,',',6,2);
%Deleting Unused Inputs
M(:,5:9) = [];
M(:,6:12) = [];
M(:,8:16) = [];
M(:,9:71) = [];
x = M(:,3:5);
M(:,3:4) = [];
%Sorted .txt file for target arrangement for the NN function
a = M;
a = abs(a); %Target
%sorting x for input configuration for the NN function
x(:,3) = x(:,2);
x(:,2) = x(:,1);
x(:,1) = x(:,3);
x(:,3)=[];
x = abs(x); %Input
y1=0;
stdd = zeros(6,1);
meann = zeros(6,1);
error2 = zeros(r,6);
error4 = zeros(r,6);
error5 = zeros(6,1);
per = zeros(6,1);
%T = zeros(6,2);
T =
[0.00378275971487120;0.45;0.702682576988365;0.789902292644835;0.20386973857
8888;
0.201130210593137];
error1 = zeros(r+5,6);
y = NNFunc(x'); %Neural Network Model
l = [1;1;1.5;1.5;1.5;1];
error =(((a'-y))').^2; %Unfiltered Residual
%Scale data between 0 and 1 using Tan-Sig Function
for o=1:6
for j = 1:r+5
error1(j,o) = (2/(1+exp(-error(j,o))))-1;
end
end
%Residual Filter************************************************************
for o=1:6
for j = 1:r
for g = 0:4
error2(j,o) = error2(j,o)+(1/5)*(error1(j+g,o));
end
end
end
%***************************************************************************
% Threshold Calculation
for t=1:6
meann(t,1) = mean(error2(:,t)); %Mean
```

```
stdd(t,1) = std(error2(:,t)); %Standard Deviation
%T(t,1) = meann(t,1) + l(t,1)*stdd(t,1); %Upper Threshold Calculation where
l is
lamda
%T(t,2) = meann(t,1) - l(t,1)*stdd(t,1);
end
d = 1:r;
% Count Number of Faulty Conditions
for e = 1:6
for q = 1:r
if error2(q,e) > T(e,1)
per(e,1) = (error2(q,e) - T(e,1))/(1-T(e,1));
count(e,1) = count(e,1) + 1*per(e,1);
end
end
count(e,1) = (count(e,1)/140).*100;
end
% Check for Multiple Faults
for j = 1:6
count(j,2) = count(j,1)/100;
count(j,2) = 0.1*round(count(j,2)/0.1);
if count(j,2) == 1
count(1,3) = count(1,3) + 1;
end
end
% Save count values in count.txt
fileID = fopen('count.txt','w');
fprintf(fileID,'%f,%f,%f,%f,%f,%f',count(:,1));
fclose(fileID);
fileID = fopen('count1.txt','w');
fprintf(fileID,'%f',count(:,3));
fclose(fileID);
if count(1,3) > 1
error3 =(((a'-y))'); %Unfiltered Residual
%Scale data between -1 and 1
for o=1:6
for j = 1:r
error4(j,o) = (2/(1+exp(-error3(j,o))))-1;
end
for j=1:r
error5(o,1) = error5(o,1) + (error4(j,o))/140;
end
error5(o,1) = 0.1*round(error5(o,1)/0.1);
end
y1 = Classifier(error5); %Neural Network Model

y1 = round(y1,1);
fileID = fopen('class.txt','w');
fprintf(fileID,'%f',y1);
fclose(fileID);
else
end
figure
plot(d,error2(d,1), [0 r], [T(1,1) T(1,1)], 'r');
xlabel('Timesteps');
ylabel('Engine Notch: Error per Pattern');
legend('Residual', 'Threshold Limit');
title('Engine Notch Command Residual Evaluation')
%ylim([-0.1 0.1]);
grid on;
figure
plot(d,error2(d,2), [0 r], [T(2,1) T(2,1)], 'r');
```

```
xlabel('Timesteps');
ylabel('Power Notch Command: Error per Pattern');
legend('Residual', 'Threshold Limit');
title('Power Notch Command Residual Evaluation')
%ylim([-0.1 0.1]);
grid on;
figure
plot(d,error2(d,3), [0 r], [T(3,1) T(3,1)], 'r');
xlabel('Timesteps');
ylabel('Main Generator Volts: Error per Pattern');
legend('Residual', 'Threshold Limit');
title('Main Generator Volts Residual Evaluation')
%ylim([-0.1 0.1]);
grid on;
figure
plot(d,error2(d,4),[0 r], [T(4,1) T(4,1)], 'r');
xlabel('Timesteps');
ylabel('Exciter Armature Output: Error per Pattern');
legend('Residual', 'Threshold Limit');
title('Exciter Armature Output Residual Evaluation')
%ylim([0 0.005]);
grid on;
figure
plot(d,error2(d,5), [0 r], [T(5,1) T(5,1)], 'r');
xlabel('Timesteps');

ylabel('Exciter Field Amps: Error per Pattern');
legend('Residual', 'Threshold Limit');
title('Exciter Field Amps Residual Evaluation')
%ylim([-0.1 0.1]);
grid on;
figure
plot(d,error2(d,6), [0 r], [T(6,1) T(6,1)], 'r');
xlabel('Timesteps');
ylabel('Load Control Pot: Error per Pattern');
legend('Residual', 'Threshold Limit');
title('LCP Residual Evaluation')
%ylim([-0.1 0.1]);
grid on;
```

O2 SCM8 Residual Generation and Evaluation Source Code

```
%SCM8 Sensor Residual Generation and Evaluation Source Code
r=140;
count =zeros(6,2);
%Read In Locomotives recorded Data from Data.txt file
filename = 'C:\Decoder\DATA.txt';
M = dlmread(filename,',',6,2);
%Deleting Unused Inputs
M(:,5:9) = [];
M(:,6:12) = [];
M(:,8:16) = [];
M(:,9:71) = [];
x = M;
%sorting x for input configuration for the NN function
x(:,1) = M(:,7);
x(:,2) = M(:,3);
x(:,3) = M(:,2);
x(:,4) = M(:,1);
x(:,5) = M(:,8);
x(:,6) = M(:,6);
x(:,7) = M(:,4);
x(:,8) = [];
%Sorted .txt file for target arrangement for the NN function
a = M(:,5);
% Get absolute value of the input and target values
a = abs(a); %Target
x = abs(x); %Input
stdd = zeros(1,1);
meann = zeros(1,1);
error2 = zeros(r,1);
per = zeros(6,1);
T = [0.614165657480314];
%T = zeros(1,1);
error1 = zeros(r+5,6);
y = SCM8Func(x'); %Neural Network Model
l = [1.1;1;1;1;1;1];
error =(((a'-y))').^2; %Unfiltered Residual
%Scale data between 0 and 1 using Sigmoid Function
for o=1:1
for j = 1:r+5
error1(j,o) = (2/(1+exp(-error(j,o))))-1;
end
end
%Residual Filter************************************************************
for o=1:1
for j = 1:r
for g = 0:4
error2(j,o) = error2(j,o)+(1/5)*(error1(j+g,o));
end
end
end
%***************************************************************************
% Threshold Calculation
for t=1:1
meann(t,1) = mean(error2(:,t)); %Mean
stdd(t,1) = std(error2(:,t)); %Standard Deviation
%T(t,1) = meann(t,1) + l(t,1)*stdd(t,1); %Upper Threshold Calculation where
l is
lamda
%T(t,2) = meann(t,1) - l(t,1)*stdd(t,1);
end
```

```
d = 1:r;
% Count Number of Faulty Conditions
for e = 1:1
for q = 1:r
if error2(q,e) > T(e,1)
per(e,1) = (error2(q,e) - T(e,1))/(1-T(e,1));
count(e,1) = count(e,1) + 1*per(e,1);
end
end
count(e)=(count(e)/140)*100;
end
% Save count values in count.txt
fileID = fopen('temp.txt','w');
fprintf(fileID,'%f',count(1,1));
fclose(fileID);
figure
plot(d,error2(d,1),[0 r], [T(1,1) T(1,1)], 'r');
xlabel('Timesteps');
ylabel('Rectified Alternator Volts: Error per Pattern');
legend('Residual', 'Threshold Limit');
title('Rectified Alternator Volts Residual Evaluation')
%ylim([0 0.005]);
grid on;
```

O3 Power Notch Command Residual Generation and Evaluation Source Code

```
%Power Notch Command Sensor Residual Generation and Evaluation Source Code
r=140;
count =zeros(6,2);
%Read In Locomotives recorded Data from Data.txt file
filename = 'C:\Decoder\DATA.txt';
M = dlmread(filename,',',6,2);
%Deleting Unused Inputs
M(:,5:9) = [];
M(:,6:12) = [];
M(:,8:16) = [];
M(:,9:71) = [];
x = M;
%sorting x for input configuration for the NN function
x(:,1) = M(:,6);
x(:,2) = M(:,8);
x(:,3) = M(:,4);
x(:,4) = M(:,1);
x(:,5) = M(:,7);
x(:,6) = M(:,5);
x(:,7) = M(:,3);
x(:,8) = [];
%Sorted .txt file for target arrangement for the NN function
a = M(:,2);
% Get absolute value of the input and target values
a = abs(a); %Target
x = abs(x); %Input
stdd = zeros(1,1);
meann = zeros(1,1);
error2 = zeros(r,1);
per = zeros(6,1);
%T = [0.506891303177012];
T = [9.97187996335642e-07];
error1 = zeros(r+5,6);
y = PNCFunc(x'); %Neural Network Model
l = [2;1;1;1;1;1];
error =(((a'-y))').^2; %Unfiltered Residual
%Scale data between 0 and 1 using Sigmoid Function
for o=1:1
for j = 1:r+5
error1(j,o) = (2/(1+exp(-error(j,o))))-1;
end
end
%Residual Filter************************************************************
for o=1:1
for j = 1:r
for g = 0:4
error2(j,o) = error2(j,o)+(1/5)*(error1(j+g,o));
end
end
end
%***************************************************************************
% Threshold Calculation
for t=1:1
meann(t,1) = mean(error2(:,t)); %Mean
stdd(t,1) = std(error2(:,t)); %Standard Deviation
%T(t,1) = meann(t,1) + l(t,1)*stdd(t,1); %Upper Threshold Calculation where
l is
lamda
%T(t,2) = meann(t,1) - l(t,1)*stdd(t,1);
end
```

```
d = 1:r;
% Count Number of Faulty Conditions
for e = 1:1
for q = 1:r
if error2(q,e) > T(e,1)
per(e,1) = (error2(q,e) - T(e,1))/(1-T(e,1));
count(e,1) = count(e,1) + 1*per(e,1);
end
end
count(e) = (count(e)/140)*100;
end
% Save count values in count.txt
fileID = fopen('temp.txt','w');
fprintf(fileID,'%f',count(1,1));
fclose(fileID);
figure
plot(d,error2(d,1),[0 r], [T(1,1) T(1,1)], 'r');
xlabel('Timesteps');
ylabel('Power Notch Command: Error per Pattern');
legend('Residual', 'Threshold Limit');
title('Power Notch Command Residual Evaluation')
%ylim([0 0.005]);
grid on;
```

O4 LCP Residual Generation and Evaluation Source Code

```
%LCP Sensor Residual Generation and Evaluation Source Code
r=140;
count =zeros(6,2);
%Read In Locomotives recorded Data from Data.txt file
filename = 'C:\Decoder\DATA.txt';
M = dlmread(filename,',',6,2);
%Deleting Unused Inputs
M(:,5:9) = [];
M(:,6:12) = [];
M(:,8:16) = [];
M(:,9:71) = [];
x = M;
%sorting x for input configuration for the NN function
x(:,8) = x(:,1);
x(:,1) = x(:,6);
x(:,6) = x(:,2);
x(:,2) = x(:,3);
x(:,3) = x(:,6);
x(:,6) = x(:,4);
x(:,4) = x(:,8);
x(:,8) = x(:,5);
x(:,5) = x(:,7);
x(:,7) = x(:,6);
x(:,6) = x(:,8);
x(:,8) = [];
%Sorted .txt file for target arrangement for the NN function
a = M(:,8);
% Get absolute value of the input and target values
a = abs(a); %Target
x = abs(x); %Input
stdd = zeros(1,1);
meann = zeros(1,1);
error2 = zeros(r,1);
per = zeros(6,1);
T = [0.424699204643921];
error1 = zeros(r+5,6);
y = LCPFunc(x'); %Neural Network Model
l = [1.5;1;1;1;1;1];
error =(((a'-y))').^2; %Unfiltered Residual
%Scale data between 0 and 1
for o=1:1
for j = 1:r+5
error1(j,o) = (2/(1+exp(-error(j,o))))-1;
end
end
%Residual Filter************************************************************
for o=1:1
for j = 1:r
for g = 0:4
error2(j,o) = error2(j,o)+(1/5)*(error1(j+g,o));
end
end
end
%***************************************************************************
% Threshold Calculation
for t=1:1
meann(t,1) = mean(error2(:,t)); %Mean
stdd(t,1) = std(error2(:,t)); %Standard Deviation
%T(t,1) = meann(t,1) + l(t,1)*stdd(t,1); %Upper Threshold Calculation where
l is
```

```
lamda
%T(t,2) = meann(t,1) - l(t,1)*stdd(t,1);
end
d = 1:r;
% Count Number of Faulty Conditions
for e = 1:1
for q = 1:r
if error2(q,e) > T(e,1)
per(e,1) = (error2(q,e) - T(e,1))/(1-T(e,1));
count(e,1) = count(e,1) + 1*per(e,1);
end
end
count(e) = (count(e)/140)*100;
end
% Save count values in count.txt
fileID = fopen('temp.txt','w');
fprintf(fileID,'%f',count(1,1));
fclose(fileID);
figure
plot(d,error2(d,1),[0 r], [T(1,1) T(1,1)], 'r');
xlabel('Timesteps');
ylabel('Load Control Pot: Error per Pattern');
legend('Residual', 'Threshold Limit');
title('Load Control Pot Residual Evaluation')
%ylim([0 0.005]);
grid on;
```

O5 EXFM Residual Generation and Evaluation Source Code

```
%EXFM Sensor Residual Generation and Evaluation Source Code
r=140;
count =zeros(6,2);
%Read In Locomotives recorded Data from Data.txt file
filename = 'C:\Decoder\DATA.txt';
M = dlmread(filename,',',6,2);
%Deleting Unused Inputs
M(:,5:9) = [];
M(:,6:12) = [];
M(:,8:16) = [];
M(:,9:71) = [];
x = M;
%sorting x for input configuration for the NN function
x(:,1) = M(:,6);
x(:,2) = M(:,3);
x(:,3) = M(:,2);
x(:,4) = M(:,1);
x(:,5) = M(:,8);
x(:,6) = M(:,5);
x(:,7) = M(:,4);
x(:,8) = [];
%Sorted .txt file for target arrangement for the NN function
a = M(:,7);
% Get absolute value of the input and target values
a = abs(a); %Target
x = abs(x); %Input
stdd = zeros(1,1);
meann = zeros(1,1);
error2 = zeros(r,1);
per = zeros(6,1);
%T = 0;
T = 0.139353725477192;
error1 = zeros(r+5,6);
y = EXFMFunc(x'); %Neural Network Model
l = [1.5;1;1;1;1;1];
error =(((a'-y))').^2; %Unfiltered Residual
%Scale data between 0 and 1 using Sigmoid Function
for o=1:1
for j = 1:r+5
error1(j,o) = (2/(1+exp(-error(j,o))))-1;
end
end
%Residual Filter************************************************************
for o=1:1
for j = 1:r
for g = 0:4
error2(j,o) = error2(j,o)+(1/5)*(error1(j+g,o));
end
end
end
%***************************************************************************
% Threshold Calculation
for t=1:1
meann(t,1) = mean(error2(:,t)); %Mean
stdd(t,1) = std(error2(:,t)); %Standard Deviation
%T(t,1) = meann(t,1) + l(t,1)*stdd(t,1); %Upper Threshold Calculation where
l is
lamda
%T(t,2) = meann(t,1) - l(t,1)*stdd(t,1);
end
```

```
d = 1:r;
% Count Number of Faulty Conditions
for e = 1:1
for q = 1:r
if error2(q,e) > T(e,1)
per(e,1) = (error2(q,e) - T(e,1))/(1-T(e,1));
count(e,1) = count(e,1) + 1*per(e,1);
end
end
count(e)=(count(e)/140)*100;
end
% Save count values in count.txt
fileID = fopen('temp.txt','w');
fprintf(fileID,'%f',count(1,1));
fclose(fileID);
figure
plot(d,error2(d,1),[0 r], [T(1,1) T(1,1)], 'r');
xlabel('Timesteps');
ylabel('Exciter field Current: Error per Pattern');
legend('Residual', 'Threshold Limit');
title('Exciter field Current Residual Evaluation')
%ylim([0 0.005]);
grid on;
```

O6 EXACT Residual Generation and Evaluation Source Code

```
%EXACT Sensor Residual Generation and Evaluation Source Code
r=140;
count =zeros(6,2);
%Read In Locomotives recorded Data from Data.txt file
filename = 'C:\Decoder\DATA.txt';
M = dlmread(filename,',',6,2);
%Deleting Unused Inputs
M(:,5:9) = [];
M(:,6:12) = [];
M(:,8:16) = [];
M(:,9:71) = [];
x = M;
%sorting x for input configuration for the NN function
x(:,1) = M(:,7);
x(:,2) = M(:,3);
x(:,3) = M(:,2);
x(:,4) = M(:,1);
x(:,5) = M(:,8);
x(:,6) = M(:,5);
x(:,7) = M(:,4);
x(:,8) = [];
%Sorted .txt file for target arrangement for the NN function
a = M(:,6);
% Get absolute value of the input and target values
a = abs(a); %Target
x = abs(x); %Input
stdd = zeros(1,1);
meann = zeros(1,1);
error2 = zeros(r,1);
per = zeros(6,1);
T = 0.940231031675765;
error1 = zeros(r+5,6);
y = EXACTFunc(x'); %Neural Network Model
l = [1.2;1;1;1;1;1];
error =(((a'-y))').^2; %Unfiltered Residual
%Scale data between 0 and 1
for o=1:1
for j = 1:r+5
error1(j,o) = (2/(1+exp(-error(j,o))))-1;
end
end
%Residual Filter************************************************************
for o=1:1
for j = 1:r
for g = 0:4
error2(j,o) = error2(j,o)+(1/5)*(error1(j+g,o));
end
end
end
%***************************************************************************
% Threshold Calculation
for t=1:1
meann(t,1) = mean(error2(:,t)); %Mean
stdd(t,1) = std(error2(:,t)); %Standard Deviation
%T(t,1) = meann(t,1) + l(t,1)*stdd(t,1); %Upper Threshold Calculation where
l is
lamda
%T(t,2) = meann(t,1) - l(t,1)*stdd(t,1);
end
d = 1:r;
```

```
% Count Number of Faulty Conditions
for e = 1:1
for q = 1:r
if error2(q,e) > T(e,1)
per(e,1) = (error2(q,e) - T(e,1))/(1-T(e,1));
count(e,1) = count(e,1) + 1*per(e,1);
end
end
count(e) = (count(e)/140)*100;
end
% Save count values in count.txt
fileID = fopen('temp.txt','w');
fprintf(fileID,'%f',count(1,1));
fclose(fileID);
figure
plot(d,error2(d,1),[0 r], [T(1,1) T(1,1)], 'r');
xlabel('Timesteps');
ylabel('Exciter Armature Output: Error per Pattern');
legend('Residual', 'Threshold Limit');
title('Exciter Armature Output Residual Evaluation')
%ylim([0 0.005]);
grid on;
```

O7 Engine Notch Command Residual Generation and Evaluation Source Code

```
%Engine Notch Command Sensor Residual Generation and Evaluation Source Code
r=140;
count =zeros(6,2);
%Read In Locomotives recorded Data from Data.txt file
filename = 'C:\Decoder\DATA.txt';
M = dlmread(filename,',',6,2);
%Deleting Unused Inputs
M(:,5:9) = [];
M(:,6:12) = [];
M(:,8:16) = [];
M(:,9:71) = [];
x = M;
%sorting x for input configuration for the NN function
x(:,1) = M(:,6);
x(:,2) = M(:,8);
x(:,3) = M(:,4);
x(:,4) = M(:,7);
x(:,5) = M(:,5);
x(:,6) = M(:,3);
x(:,7) = M(:,2);
x(:,8) = [];
%Sorted .txt file for target arrangement for the NN function
a = M(:,1);
% Get absolute value of the input and target values
a = abs(a); %Target
x = abs(x); %Input
stdd = zeros(1,1);
meann = zeros(1,1);
error2 = zeros(r,1);
per = zeros(6,1);
T = [4.12642051435862e-08];
%T = [0.940231031675765];
error1 = zeros(r+5,6);
y = ENCFunc(x'); %Neural Network Model
l = [3.2;1;1;1;1;1];
error =(((a'-y))').^2; %Unfiltered Residual
%Scale data between 0 and 1 using Sigmoid Function
for o=1:1
for j = 1:r+5
error1(j,o) = (2/(1+exp(-error(j,o))))-1;
end
end
%Residual Filter************************************************************
for o=1:1
for j = 1:r
for g = 0:4
error2(j,o) = error2(j,o)+(1/5)*(error1(j+g,o));
end
end
end
%***************************************************************************
% Threshold Calculation
for t=1:1
meann(t,1) = mean(error2(:,t)); %Mean
stdd(t,1) = std(error2(:,t)); %Standard Deviation
%T(t,1) = meann(t,1) + l(t,1)*stdd(t,1); %Upper Threshold Calculation where
l is
lamda
%T(t,2) = meann(t,1) - l(t,1)*stdd(t,1);
end
```

```
d = 1:r;
% Count Number of Faulty Conditions
for e = 1:1
for q = 1:r
if error2(q,e) > T(e,1)
per(e,1) = (error2(q,e) - T(e,1))/(1-T(e,1));
count(e,1) = count(e,1) + 1*per(e,1);
end
end
count(e) = ((count(e))/140)*100;
end
% Save count values in count.txt
fileID = fopen('temp.txt','w');
fprintf(fileID,'%f',count(1,1));
fclose(fileID);
figure
plot(d,error2(d,1),[0 r], [T(1,1) T(1,1)], 'r');
xlabel('Timesteps');
ylabel('Engine Notch Command: Error per Pattern');
legend('Residual', 'Threshold Limit');
title('Engine Notch Command Residual Evaluation')
%ylim([0 0.005]);
grid on;
```

Appendix P: GUI Application Source Code

```
function varargout = untitled(varargin)
gui_Singleton = 1;
gui_State = struct('gui_Name', mfilename, ...
'gui_Singleton', gui_Singleton, ...
'gui_OpeningFcn', @untitled_OpeningFcn, ...
'gui_OutputFcn', @untitled_OutputFcn, ...
'gui_LayoutFcn', [] , ...
'gui_Callback', []);
if nargin && ischar(varargin{1})
gui_State.gui_Callback = str2func(varargin{1});
end
if nargout
[varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
gui_mainfcn(gui_State, varargin{:});
end
end
% --- Executes just before untitled is made visible.
function untitled_OpeningFcn(hObject, eventdata, handles, varargin)
% Choose default command line output for untitled
handles.output = hObject;
% Update handles structure
guidata(hObject, handles);
end
% --- Outputs from this function are returned to the command line.
function varargout = untitled_OutputFcn(hObject, eventdata, handles)
varargout{1} = handles.output;
end
% --- Executes on button press in pushbutton1.
function pushbutton1_Callback(hObject, eventdata, handles)
%Check if locomotive number is inserted into the textbox if not Display
%windows pop-up
loco = get(handles.edit1,'string');
if isempty(loco)
msgbox('Enter Locomotive Number First', 'Error','error');
else
% Display Count Down to start with the FDI Process
set(handles.text33,'string','Test will start in 10 sec')
pause(1)
set(handles.text33,'string','Test will start in 9 sec')
pause(1)
set(handles.text33,'string','Test will start in 8 sec')
pause(1)
set(handles.text33,'string','Test will start in 7 sec')
pause(1)
set(handles.text33,'string','Test will start in 6 sec')
pause(1)
set(handles.text33,'string','Test will start in 5 sec')
pause(1)
set(handles.text33,'string','Test will start in 4 sec')
pause(1)
set(handles.text33,'string','Test will start in 3 sec')
pause(1)
set(handles.text33,'string','Test will start in 2 sec')
pause(1)
set(handles.text33,'string','Test will start in 1 sec')
pause(1)
set(handles.text33,'string','Test Running')
```

```
NET.addAssembly('System.Windows.Forms')
%Open Locomotive Interfacing Program
system('C:\Users\Chantel\Desktop\Werk\BSS\Hyperterminal\Hyperterminal\hyper
trm.exe
C:\Users\Chantel\Desktop\Werk\BSS\Hyperterminal\Hyperterminal\Morne.ht &')
pause(1);
System.Windows.Forms.SendKeys.SendWait('{ENTER}');
pause(1);
%Username and Password Login to enable Recording
System.Windows.Forms.SendKeys.SendWait('root{Enter}');
pause(1);
System.Windows.Forms.SendKeys.SendWait('d7erie{Enter}');
pause(2);
%Call Record Function
System.Windows.Forms.SendKeys.SendWait('tbdis{Enter}{d}y{Enter}200{Enter}n{
Enter}
exc.ds{Enter}');
pause(30);
%Create DATA.REC File
System.Windows.Forms.SendKeys.SendWait('{e}{e}cd work{Enter}sz
data.rec{Enter}');
pause(20);
%Exit Recording Mode
System.Windows.Forms.SendKeys.SendWait('exit{Enter}');
pause(1);
%Close Hyperterminal
System.Windows.Forms.SendKeys.SendWait('%{F4}{Enter}');
%Cut And Paste .rec File
if exist('C:\Users\Chantel\Desktop\Werk\BSS\Hyperterminal\DATA.rec',
'file')
movefile('C:\Users\Chantel\Desktop\Werk\BSS\Hyperterminal\DATA.rec', 'C:
\Decoder\DATA.rec');
%Open DOS Box to decode the .rec file to a .txt file
system('C:\Program Files (x86)\DOSBox-0.74\DOSBox.exe &')
%Using DOSBox to run the Decoder program
pause(4);
System.Windows.Forms.SendKeys.SendWait(' mount c c:\');
pause(1);
%Code for mounting the Decoder directory ************************
System.Windows.Forms.SendKeys.SendWait('Decoder {Enter}');
pause(1);
System.Windows.Forms.SendKeys.SendWait(' c: {Enter}');
pause(1);
System.Windows.Forms.SendKeys.SendWait(' Program {Enter}');
pause(1);
%'Running the Coding software and procdure************************
System.Windows.Forms.SendKeys.SendWait(' Data.rec {Enter}');
pause(1);
System.Windows.Forms.SendKeys.SendWait(' Data.txt {Enter}');
pause(10);
System.Windows.Forms.SendKeys.SendWait(' Exit {Enter}');
%Delete .rec file
if exist('C:\Decoder\DATA.rec', 'file')
delete 'C:\Decoder\DATA.rec';
end
% Cut and Paste decoded .txt file
if exist('C:\Decoder\DATA.txt', 'file')
movefile('C:\Decoder\DATA.txt', 'C:\Decoder\txt Data files\DATA.txt')
end
%Remove Unwanted Text from .txt file
if exist('C:\Decoder\DATA.txt', 'file')
Filter;
```

```
end
end
%Read In Locomotives recorded Data from Data.txt file
filename = 'count.txt';
count = dlmread(filename,',',0,0);
set(handles.edit3,'string',num2str(count(1,1)));
set(handles.edit4,'string',num2str(count(1,2)));
set(handles.edit5,'string',num2str(count(1,3)));
set(handles.edit6,'string',num2str(count(1,4)));
set(handles.edit7,'string',num2str(count(1,5)));
set(handles.edit8,'string',num2str(count(1,6)));
filename2 = 'count1.txt';
count1 = dlmread(filename2,',',0,0);
n=0;
%Highlight the Highest Probability for the failing Section
for i = 1:6
if max(count(:)) == (count(1,i))
n=i;
switch n
case 1
set(handles.edit3,'BackgroundColor','red');
case 2
set(handles.edit4,'BackgroundColor','red');
case 3
set(handles.edit5,'BackgroundColor','red');
case 4
set(handles.edit6,'BackgroundColor','red');
case 5
set(handles.edit7,'BackgroundColor','red');
case 6
set(handles.edit8,'BackgroundColor','red');
end
end
end
%Check to see if there are more than one highest failing sections
if count1 < 2
n=0;
%Find the highest Probability of failing Component and Perform Sensor
%Validation
for i = 1:6
if max(count(:)) == (count(1,i))
n = i;
end
end
%Set GUI to indicate faulty sections and display faut in Textbox
switch n
case 1
ENC; %Engine Notch Command Function
filename1 = 'temp.txt';
ENC1 = dlmread(filename1,',',0,0);
set(handles.edit10,'string','N/A');
set(handles.edit11,'string','N/A');
set(handles.edit12,'string','N/A');
set(handles.edit13,'string','N/A');
set(handles.edit14,'string','N/A');
set(handles.edit9,'string',num2str(ENC1));
set(handles.edit21,'string','N/A');
set(handles.edit22,'string','N/A');
set(handles.edit23,'string','N/A');
set(handles.edit24,'string','N/A');
set(handles.edit25,'string','N/A');
set(handles.edit26,'string','N/A');
```

```
if ENC1 > 90
set(handles.edit9,'BackgroundColor','red');
set(handles.text33,'string','Engine Notch Command Card Faulty')
else
set(handles.text33,'string','Master Controller Contact Tips Faulty')
end
case 2
PNC; %Power Notch Command Function
filename1 = 'temp.txt';
PNC1 = dlmread(filename1,',',0,0);
set(handles.edit9,'string','N/A');
set(handles.edit11,'string','N/A');
set(handles.edit12,'string','N/A');
set(handles.edit13,'string','N/A');
set(handles.edit14,'string','N/A');
set(handles.edit10,'string',num2str(PNC1));
set(handles.edit21,'string','N/A');
set(handles.edit22,'string','N/A');
set(handles.edit23,'string','N/A');
set(handles.edit24,'string','N/A');
set(handles.edit25,'string','N/A');
set(handles.edit26,'string','N/A');
if PNC1 > 90
set(handles.edit10,'BackgroundColor','red');
set(handles.text33,'string','Power Notch Command Card Faulty')
else
set(handles.text33,'string','Master Controller Contact Tips Faulty')
end
case 3
SCM8; %Main Alternator Generator Function
filename1 = 'temp.txt';
SCM81 = dlmread(filename1,',',0,0);
set(handles.edit9,'string','N/A');
set(handles.edit10,'string','N/A');
set(handles.edit12,'string','N/A');
set(handles.edit13,'string','N/A');
set(handles.edit14,'string','N/A');
set(handles.edit11,'string',num2str(SCM81));
set(handles.edit21,'string','N/A');
set(handles.edit22,'string','N/A');
set(handles.edit23,'string','N/A');
set(handles.edit24,'string','N/A');
set(handles.edit25,'string','N/A');
set(handles.edit26,'string','N/A');
if SCM81 > 90
set(handles.edit11,'BackgroundColor','red');
set(handles.text33,'string','SCM8 Sensor Faulty')
else
set(handles.text33,'string','Alternator Causing Oscilation')
end
case 4
EXACT; %Exciter Armature Current Function
filename1 = 'temp.txt';
EXACT1 = dlmread(filename1,',',0,0);
set(handles.edit9,'string','N/A');
set(handles.edit10,'string','N/A');
set(handles.edit11,'string','N/A');
set(handles.edit13,'string','N/A');
set(handles.edit14,'string','N/A');
set(handles.edit12,'string',num2str(EXACT1));
set(handles.edit21,'string','N/A');
set(handles.edit22,'string','N/A');
```

```
set(handles.edit23,'string','N/A');
set(handles.edit24,'string','N/A');
set(handles.edit25,'string','N/A');
set(handles.edit26,'string','N/A');
if EXACT1 > 90
set(handles.edit12,'BackgroundColor','red');
set(handles.text33,'string','EXACT Sensor Faulty')
else
set(handles.text33,'string','Exciter Armature causing oscillation')
end
case 5
EXFM; %Exciter Field Current Function
filename1 = 'temp.txt';
EXFM1 = dlmread(filename1,',',0,0);
set(handles.edit9,'string','N/A');
set(handles.edit10,'string','N/A');
set(handles.edit11,'string','N/A');
set(handles.edit12,'string','N/A');
set(handles.edit14,'string','N/A');
set(handles.edit13,'string',num2str(EXFM1));
set(handles.edit21,'string','N/A');
set(handles.edit22,'string','N/A');
set(handles.edit23,'string','N/A');
set(handles.edit24,'string','N/A');
set(handles.edit25,'string','N/A');
set(handles.edit26,'string','N/A');
if EXFM1 > 90
set(handles.edit13,'BackgroundColor','red');
set(handles.text33,'string','EXFM Module Faulty')
else
set(handles.text33,'string','BSS Box/Loose wires on circuit')
end
case 6
LCP; %Load Control Potensiometer Function
filename1 = 'temp.txt';
LCP1 = dlmread(filename1,',',0,0);
set(handles.edit9,'string','N/A');
set(handles.edit10,'string','N/A');
set(handles.edit11,'string','N/A');
set(handles.edit12,'string','N/A');
set(handles.edit13,'string','N/A');
set(handles.edit14,'string',num2str(LCP1));
set(handles.edit21,'string','N/A');
set(handles.edit22,'string','N/A');
set(handles.edit23,'string','N/A');
set(handles.edit24,'string','N/A');
set(handles.edit25,'string','N/A');
set(handles.edit26,'string','N/A');
if LCP1 > 90
set(handles.edit14,'BackgroundColor','red');
set(handles.text33,'string','LCP Sensing Device Faulty')
else
set(handles.text33,'string','Governor Defective')
end
end
%If more than one section is faulty perform Total failure Component FDI
elseif count1 > 1
set(handles.edit9,'string','N/A');
set(handles.edit10,'string','N/A');
set(handles.edit11,'string','N/A');
set(handles.edit12,'string','N/A');
set(handles.edit13,'string','N/A');
```

```
set(handles.edit14,'string','N/A');
filename1 = 'class.txt';
c = dlmread(filename1,',',0,0);
switch c
case 1 %Governor Faulty
set(handles.edit21,'string','OK');
set(handles.edit22,'string','OK');
set(handles.edit23,'string','OK');
set(handles.edit24,'string','OK');
set(handles.edit25,'string','OK');
set(handles.edit26,'string','Fault');
set(handles.edit26,'BackgroundColor','red');
set(handles.text33,'string','Governor Faulty')
case 2 %Exciter Field Winding or Circuit Faulty
set(handles.edit21,'string','OK');
set(handles.edit22,'string','OK');
set(handles.edit23,'string','OK');
set(handles.edit24,'string','OK');
set(handles.edit25,'string','Fault');
set(handles.edit25,'BackgroundColor','red');
set(handles.text33,'string','Exciter Field Winding or Circuit Faulty')
set(handles.edit26,'string','OK');
case 3 %Exciter Armature/Alternator Rotor Open
set(handles.edit21,'string','OK');
set(handles.edit22,'string','OK');
set(handles.edit23,'string','OK');
set(handles.edit24,'string','Fault');
set(handles.edit24,'BackgroundColor','red');
set(handles.text33,'string','Exciter Armature/Alternator Rotor Open')
set(handles.edit25,'string','OK');
set(handles.edit26,'string','OK');
case 4 %Alternator Stator Winding Open
set(handles.edit21,'string','OK');
set(handles.edit22,'string','OK');
set(handles.edit23,'string','Fault');
set(handles.edit23,'BackgroundColor','red');
set(handles.text33,'Alternator Stator Winding Open')
set(handles.edit24,'string','OK');
set(handles.edit25,'string','OK');
set(handles.edit26,'string','OK');
end
end
end
end
function edit1_Callback(hObject, eventdata, handles)
set(handles.edit2,'string',date)
set(handles.text33,'string','Notch Locomotive to Notch 1 and Press Start
Button')
end
% --- Executes during object creation, after setting all properties.
function edit1_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'), get
(0,'defaultUicontrolBackgroundColor'))
set(hObject,'BackgroundColor','white');
end
end
% --- Executes during object creation, after setting all properties.
function edit2_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'), get
(0,'defaultUicontrolBackgroundColor'))
set(hObject,'BackgroundColor','white');
end
```

```
end
% --- Executes during object creation, after setting all properties.
function edit7_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'), get
(0,'defaultUicontrolBackgroundColor'))
set(hObject,'BackgroundColor','white');
end
end
% --- Executes during object creation, after setting all properties.
function edit8_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'), get
(0,'defaultUicontrolBackgroundColor'))
set(hObject,'BackgroundColor','white');
end
end
% --- Executes during object creation, after setting all properties.
function edit3_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'), get
(0,'defaultUicontrolBackgroundColor'))
set(hObject,'BackgroundColor','white');
end
end
% --- Executes during object creation, after setting all properties.
function edit4_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'), get
(0,'defaultUicontrolBackgroundColor'))
set(hObject,'BackgroundColor','white');
end
end
% --- Executes during object creation, after setting all properties.
function edit5_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'), get
(0,'defaultUicontrolBackgroundColor'))
set(hObject,'BackgroundColor','white');
end
end
function edit6_Callback(hObject, eventdata, handles)
set(hObject,'BackgroundColor','red');
end
% --- Executes during object creation, after setting all properties.
function edit6_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'), get
(0,'defaultUicontrolBackgroundColor'))
set(hObject,'BackgroundColor','white');
end
end
% --- Executes during object creation, after setting all properties.
function edit9_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'), get
(0,'defaultUicontrolBackgroundColor'))
set(hObject,'BackgroundColor','white');
end
end
% --- Executes during object creation, after setting all properties.
function edit10_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'), get
(0,'defaultUicontrolBackgroundColor'))
set(hObject,'BackgroundColor','white');
end
end
% --- Executes during object creation, after setting all properties.
function edit11_CreateFcn(hObject, eventdata, handles)
```

```
if ispc && isequal(get(hObject,'BackgroundColor'), get
(0,'defaultUicontrolBackgroundColor'))
set(hObject,'BackgroundColor','white');
end
end
% --- Executes during object creation, after setting all properties.
function edit12_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'), get
(0,'defaultUicontrolBackgroundColor'))
set(hObject,'BackgroundColor','white');
end
end
% --- Executes during object creation, after setting all properties.
function edit13_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'), get
(0,'defaultUicontrolBackgroundColor'))
set(hObject,'BackgroundColor','white');
end
end
% --- Executes during object creation, after setting all properties.
function edit14_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'), get
(0,'defaultUicontrolBackgroundColor'))
set(hObject,'BackgroundColor','white');
end
end
% --- Executes during object creation, after setting all properties.
function edit21_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'), get
(0,'defaultUicontrolBackgroundColor'))
set(hObject,'BackgroundColor','white');
end
end
% --- Executes during object creation, after setting all properties.
function edit22_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'), get
(0,'defaultUicontrolBackgroundColor'))
set(hObject,'BackgroundColor','white');
end
end
% --- Executes during object creation, after setting all properties.
function edit23_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'), get
(0,'defaultUicontrolBackgroundColor'))
set(hObject,'BackgroundColor','white');
end
end
% --- Executes during object creation, after setting all properties.
function edit24_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'), get
(0,'defaultUicontrolBackgroundColor'))
set(hObject,'BackgroundColor','white');
end
end
% --- Executes during object creation, after setting all properties.
function edit25_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'), get
(0,'defaultUicontrolBackgroundColor'))
set(hObject,'BackgroundColor','white');
end
end
% --- Executes during object creation, after setting all properties.
```

```
function edit26_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'), get
(0,'defaultUicontrolBackgroundColor'))
set(hObject,'BackgroundColor','white');
end
end
```

Appendix Q: Total Component Failure Neural Network Training

Q1 Training Data

Table Q1: Training Data Setup Configuration

Inputs						***Outputs***	***Description***
ENC	*PNC*	*SCM8*	*EXACT*	*EXFM*	*LCP*		
0	0	-1	-1	1	0	3	Exciter Fault
0.5	0.5	-1	-1	1	0.4	3	Exciter Fault
-0.5	0.4	-1	-1	1	0.3	3	Exciter Fault
0.4	0.5	-1	1	1	0.1	4	SCM8
0.2	-0.5	-1	1	1	0.5	4	SCM8
-0.2	0.3	-1	1	1	0.2	4	SCM8
0	0	-1	-1	-1	-1	1	LCP
0.5	0.3	-1	-1	-1	-1	1	LCP
-0.2	-0.4	-1	-1	-1	-1	1	LCP
0	0	-1	-1	-1	0	2	Exciter Field Fault
0.1	0.5	-1	-1	-1	0.4	2	Exciter Field Fault
-0.2	0.4	-1	-1	-1	0.2	2	Exciter Field Fault
0.2	0	-1	-1	1	0.4	3	Exciter Fault
0.1	0.4	-1	-1	1	0.6	3	Exciter Fault
-0.6	0.1	-1	-1	1	0.1	3	Exciter Fault
-0.1	-0.1	-1	1	1	0.3	4	SCM8
0.3	0.4	-1	1	1	0.2	4	SCM8
-0.4	-0.3	-1	1	1	0.1	4	SCM8
0.1	0.2	-1	-1	-1	-1	1	LCP
-0.2	-0.3	-1	-1	-1	-1	1	LCP
0.4	0.1	-1	-1	-1	-1	1	LCP
0.1	0.4	-1	-1	-1	0.2	2	Exciter Field Fault
-0.2	0.3	-1	-1	-1	-0.1	2	Exciter Field Fault
0.3	-0.5	-1	-1	-1	0.3	2	Exciter Field Fault
-0.5	0.1	-1	-1	1	-0.2	3	Exciter Fault

Q2 Neural Network Training Results

Table Q2: Training Results

Simulation	Training Set Error (MAE)	Validation Set Error (MAE)	Test Set Error (MAE)
1	0.00068589	1.058377625	0.042328098
2	2.98E-05	0.00185801	0.000799082
3	4.25E-15	0.003090358	0.008383483
4	0.001585	0.028975578	0.1222421
5	2.35E-05	0.012849337	0.024484411
6	1.40E-08	0.040633866	0.007681244
7	0.000177991	0.000813129	0.002730875
8	3.24E-05	0.04595278	0.262538907
9	0.000214427	0.144206087	0.071278713
10	0.1143902	0.020502828	0.245571781
11	0.017006559	0.196096484	0.155852689
12	5.30E-19	0.059741526	0.123838874
13	1.03E-05	0.001615204	0.003031861
14	0.001070175	0.004503877	0.001888122
15	5.48E-05	0.035887614	0.008609193
16	1.29E-06	0.011660207	0.0926219
17	1.25E-06	0.054353636	0.009202409
18	0.009424607	0.019996632	0.025580056
19	1.99E-05	0.007391007	0.068345232
20	5.32E-09	0.121071965	0.539757006
21	3.99E-20	0.00611468	0.024412182
22	2.08E-12	0.211557765	0.122602661
23	0.02478263	0.052287692	0.067237094
24	0.000185002	0.002960259	0.003659416
25	0.001070704	0.024869734	0.114168337
26	1.11E-07	0.000844669	0.000310895
27	9.48E-07	0.019574906	0.077848734
28	9.61E-25	0.009218039	0.104062425
29	0.003246482	0.241741671	0.023790243
30	4.08E-17	0.040471839	0.097242118

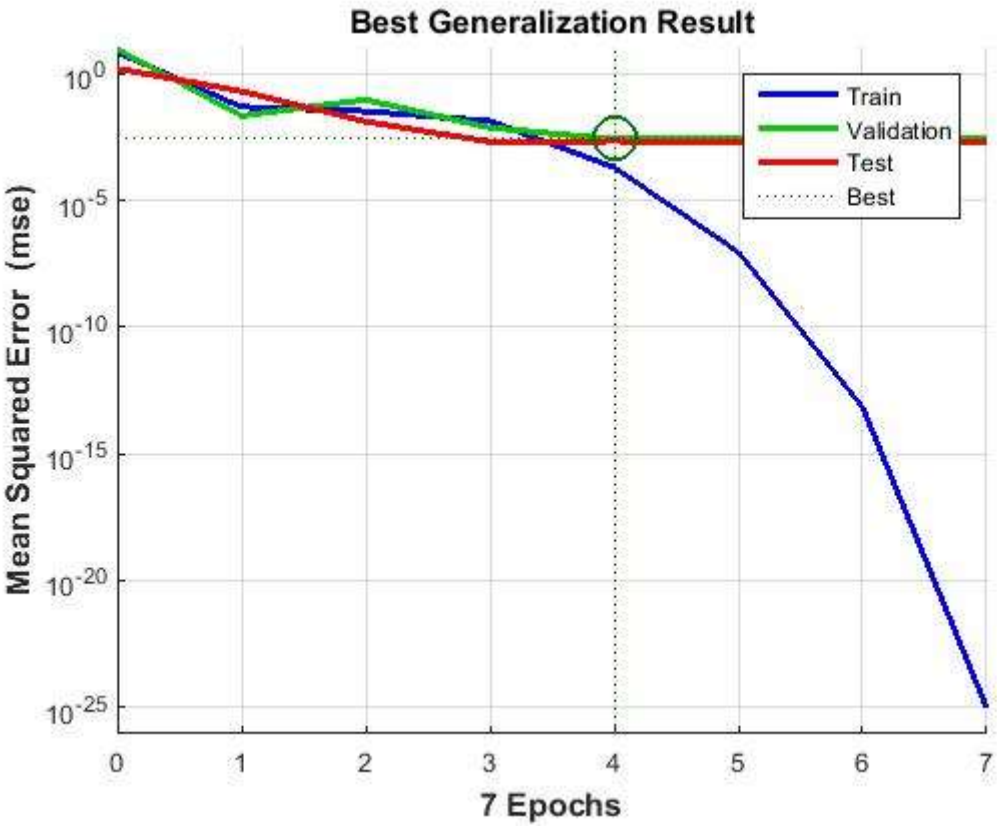


Figure Q1: Best Training Performance

Q3 Total Component Failure Neural Classifier Training Source Code

```
result = zeros(30,3);
n=1;
while (n<31)
x = a';
t = b';

% Training Function

trainFcn = 'trainlm';  % Levenberg-Marquardt backpropagation.

% Create a Fitting Network
hiddenLayerSize = 10;
net = fitnet(hiddenLayerSize,trainFcn);

%Input and Output Pre/Post-Processing Functions

net.input.processFcns = {'mapstd'};
net.output.processFcns = {'mapstd'};

% Setup Division of Data for Training, Validation, Testing
net.divideFcn = 'dividerand';  % Divide data randomly
net.divideMode = 'sample';  % Divide up every sample
net.divideParam.trainRatio = 70/100;
net.divideParam.valRatio = 15/100;
net.divideParam.testRatio = 15/100;

% Performance Function
net.performFcn = 'mae';  % Mean Squared Error

% Plot Functions
net.plotFcns = {'plotperform','plottrainstate','ploterrhist', ...
    'plotregression', 'plotfit'};

% Train the Network
[net,tr] = train(net,x,t);

% Test the Network
y = net(x);
e = gsubtract(t,y);
performance = perform(net,t,y)

% Recalculate Training, Validation and Test Performance
trainTargets = t .* tr.trainMask{1};
valTargets = t .* tr.valMask{1};
testTargets = t .* tr.testMask{1};
trainPerformance = perform(net,trainTargets,y)
valPerformance = perform(net,valTargets,y)
testPerformance = perform(net,testTargets,y)


result(n,1) = [trainPerformance];
result(n,2) = [valPerformance];
result(n,3) = [testPerformance];

if n>1
    for e = 1:n-1
if (result(n,3) < result(e,3))
```

```
    c = c +1;
end
    end
end

%Create Function + Plot for best generalization results
    if (c == (n-1))
    genFunction(net,'myNeuralNetworkFunction','MatrixOnly','yes');
    y = myNeuralNetworkFunction(x);
    figure, plotperform(tr), grid on, title('Best Generalization Result')
    end
c=0;
n=n+1

end
```