To the Graduate Council:

I am submitting herewith a dissertation written by Matthew Humberstone entitled “An Adaptive Nonparametric Modeling Technique for Expanded Condition Monitoring of Processes.” I have examined the final electronic copy of this dissertation for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Doctor of Philosophy, with a major in Nuclear Engineering.

J. Wesley Hines
Major Professor

We have read this dissertation
and recommend its acceptance:

Belle R. Upadhyaya

Haitao Liao

Russell L. Zaretzki

Accepted for the Council:

Carolyn R. Hodges
Vice Provost and Dean of the Graduate School

An Adaptive Nonparametric Modeling Technique for Expanded Condition Monitoring of Processes

A Dissertation Presented for the
Doctor of Philosophy
Degree
The University of Tennessee, Knoxville

Matthew John Humberstone
May 2010

ACKNOWLEDGMENTS

I would like to thank the members of my dissertation committee, Dr. Belle R. Upadhyaya, Dr. Haitao Liao, Dr. Russell Zaretzki, for their contributions and guidance in my research and my educational development. I would like to especially thank Dr. J. Wes Hines, my advising professor, for his guidance and management throughout my PhD. studies. Dr. Hines has been an excellent advisor, professor, and mentor and has made my studies a truly rewarding experience.

I would like to thank the faculty and staff of the Nuclear Engineering department at the University of Tennessee for their continued effort in creating a strong educational environment. Their efforts have been very beneficial in my educational growth and pushing me to strive for excellence in all areas of life. I would like to thank the head of the Nuclear Engineering department, Dr. H. Lee Dodds, specifically for his assistance and guidance throughout my graduate studies.

I would like to thank the research grant that funded me throughout my research. This research has been funded by a NERI-C research contract, advanced instrumentation and control methods for small to medium reactors with IRIS demonstration. This contract was awarded to The University of Tennessee with award number DE-FG07-07ID14895. I would like to thank Sergio Perillo, Brian Wood, James Henkel, Jamie Coble, and Michael Sharp for their contibutions with development of SIMULINK codes, the creation of the flow loop, and continued support.

I would like to thank all my friends who have supported and encouraged me in my education and other areas of life. And finally I would like to thank my family for being there for

me through both my undergraduate and graduate studies. They supported me through the ups and downs of life and were extremely encouraging.

ABSTRACT

New reactor designs and the license extensions of the current reactors has created new condition monitoring challenges. A major challenge is the creation of a data-based model for a reactor that has never been built or operated and has no historical data. This is the motivation behind the creation of a hybrid modeling technique based on first principle models that adapts to include operating reactor data as it becomes available.

An Adaptive Non-Parametric Model (ANPM) was developed for adaptive monitoring of small to medium size reactors (SMR) but would be applicable to all designs. Ideally, an adaptive model should have the ability to adapt to new operational conditions while maintaining the ability to differentiate faults from nominal conditions. This has been achieved by focusing on two main abilities. The first ability is to adjust the model to adapt from simulated conditions to actual operating conditions, and the second ability is to adapt to expanded operating conditions. In each case the system will not learn new conditions which represent faulted or degraded operations. The ANPM architecture is used to adapt the model's memory matrix from data from a First Principle Model (FPM) to data from actual system operation. This produces a more accurate model with the capability to adjust to system fluctuations.

This newly developed adaptive modeling technique was tested with two pilot applications. The first application was a heat exchanger model that was simulated in both a low and high fidelity method in SIMULINK. The ANPM was applied to the heat exchanger and improved the monitoring performance over a first principle model by increasing the model accuracy from an average MSE of 0.1451 to 0.0028 over the range of operation. The second pilot application was a flow loop built at the University of Tennessee and simulated in SIMULINK.

An improvement in monitoring system performance was observed with the accuracy of the model improving from an average MSE of 0.302 to an MSE of 0.013 over the adaptation range of operation. This research focused on the theory, development, and testing of the ANPM and the corresponding elements in the surveillance system.

EXECUTIVE SUMMARY

New condition monitoring challenges have surfaced through new reactor designs and current reactor power upgrades. The challenges is that of creating a data-based model for a reactor that has never been built or operated and has no historical data. This is the motivation behind the creation of a hybrid modeling technique based on first principle models that adapts to include operating reactor data as it becomes available while still maintaining the ability to monitor the condition of the reactor accurately.

Adaptive models used for system monitoring must strike a balance between stability and elasticity. Ideally, an adaptive model should have the ability to adapt to new operational conditions while maintaining the ability to differentiate faults from nominal conditions. There are two main abilities an adaptive model should have. The first ability is to adjust the model to adapt from simulated conditions to actual operating conditions, and the second ability is to adapt to expanded operating conditions. In each case, the system will not learn new conditions which represent faulted or degraded operations. Both abilities are important for empirical modeling in an adaptive environment. To this end, an Adaptive Non-Parametric Model (ANPM) has been developed for integrated monitoring, diagnostic, and prognostic use on small to medium size reactors (SMR). Despite the purpose of the ANPM's development there are other applications where an adaptive model will have promise. This research has focused on the theory, development, and testing of the ANPM.

The first ability of the ANPM is to gradually change the data basis of the model. The ANPM's original intent is to adapt a nonparametric model's memory matrix from data created using a first principle model (FPM) to the system's actual un-faulted data. This is useful for

monitoring new system designs from first construction and operation when the only available data is from FPMs. The FPM's data are used to build the best possible models initially, but during the system's operation new data can be collected that are more accurate for future predictions. The ANPM is used to replace the model's basis from the original FPM data to ultimately only the system's un-faulted data.

The second ability of the ANPM is to perform expanded condition monitoring (ECM). This ability can monitor for expanded conditions and determine if this condition is due to a fault or a new range of operation which should be added to the model. Non-linear modeling techniques cannot be extrapolated for predictive purposes. A new principal component analysis (PCA) ECM technique is proposed for use within the ANPM. This ability is useful in uncertain systems where the range of the model could be exceeded by the system when it is under non-fault conditions. Differentiating between expanded and fault conditions is essential to the ANPM. A method for differentiating between expanded and a fault condition for use in an adaptive model is described. Statistical process monitoring methods using principal component analysis (PCA) have been extensively studied and heavily applied in the chemical industry. A thorough literature review is given on the past applications of such methods, highlighting the applications strengths and weaknesses. A comparison between traditional fault detection and monitoring techniques and the newly proposed expanded process differentiation technique is discussed. Adaptive modeling requirements in a dynamic environment are shown to extend beyond the traditional modeling requirements. The basic approach to PCA is described, with a detailed description of the differentiation method. The Hotelling's T^2-statistic and the squared prediction error (SPE), also known as the Q-statistic, are used as indicators of the condition of the system. Dependence on the internal linear relationships of the data is discussed. This

multivariate differentiation technique is applied to a simulated data set and data from a flow control loop and is shown to correctly differentiate between expanded operating conditions and faulted conditions in an adaptive environment.

Two versions of the ANPM have been created and tested. The use of the ANPM is demonstrated with two pilot applications. The first application was a heat exchanger model that is modeled in SIMULINK with both a low fidelity and a high fidelity simulation. The second pilot application is a flow loop that was developed at the University of Tennessee. The flow control loop has the actual system and a SIMULINK model that is used to produce the first principle model representation of the flow loop. Both versions of the ANPM have been tested on the heat exchanger and have improved the monitoring performance. ANPM1 has improved the monitoring performance over the first principle model by increasing the model accuracy from an average MSE of 0.1451 to 0.0028 over the range of operation. While ANPM1_5 increased the model accuracy from an average MSE of 0.1451 to 0.02447 over the range of operation. ANPM1 outperformed the ANPM1_5 which was expected due to the vector addition techniques that were used. The ANPM was also appied to the flow loop with similar results. An improvement in monitoring system performance was observed with the accuracy of the model improving from an average MSE of 0.302 to an MSE of 0.013 over the adaptation range of operation.

The ANPM was designed for integration into a complete process surveillance system. This system contains a monitoring, detection, identification, and prognostic module which give a complete process management capability. Some generated data sets were used to test the principal component analysis (PCA) expanded condition monitoring (ECM) technique and the fault detection and identification (FDI) techniques.

The ANPM fault detection techniques have also been tested. Faults were created in the flow loop during operation to generate faulty data sets. These data sets were used to show the fault detection capabilities of the ANPM. This highlights the ability of the model to adapt without losing the needed fault detection capability.

This dissertation for the degree of Doctor of Philosophy highlights the areas of research that have been explored in the adaptive modeling realm. The theory behind the ANPM is discussed and the uses of this type of modeling techniques are shown. This adaptive modeling technique is shown to be beneficial in a number of difficult modeling environments, and shown to be an innovative solution to a developing problem.

Table of Contents

List of Figures

List of Tables

Notation and Symbol Definitions

AAKR	Autoassociative Kernel Regression
ANPM	Adaptive Nonparametric Model
CLT	Central Limit Theorem
ECM	Expanded Condition Monitoring
EULM	Error Uncertainty Limit Monitoring
FPM	First Principle Model
GNEP	Global Nuclear Energy Partnership
IPSR	Integral Primary System Reactor
IRIS	International Reactor Innovative and Secure
KICA	Kernel Independent Component Analysis
KPCA	Kernel Principal Component Analysis
KR	Kernel Regression
NFIS	Nonparametric Fuzzy Inference System
NLMSPCA	Nonlinear Multiscale Principal Component Analysis
OLS	Ordinary Least Squares
PACE	Path Classification Estimation
PCA	Principal Component Analysis
PEM	Process and Equipment Monitoring
SPE	Squared Prediction Error
SPRT	Sequential Probability Ratio Test
SSE	Sum of Squared Error
SVD	Singular Value Decomposition

1 INTRODUCTION

A number of different condition monitoring techniques have been demonstrated and shown to work to a degree necessary for on-line applications on complex systems. Applying such techniques to the current light water reactor (LWR) fleet would increase the reliability, improve performance, and reduce the maintence costs. These condition monitoring techniques have been and are currently being investigated for use on LWRs and other capital intensive systems (Hines et al. 2006a; Hines et al. 2007a; Hines et al. 2007b). These techniques use data-based models to ascertain the condition of the system while the system is operational.

This approach can be tied into a complete surveillance system which contains monitoring, diagnostic, and prognostic capabilities. Such a surveillance system would provide an accurate framework to determine the state of a system and predict future operational needs. This is beneficial in many areas of complex system management. These techniques are used to monitor systems for process degradation, fault detection, and abnormal operation. Early detection can be useful in optimizing system maintenance, safety, and in increasing reliability.

New condition monitoring challenges have been created through new reactor designs and the license extensions of current reactors. As reactors age new components will be added that can challenge a data-based monitoring system to perform accurately. The International Reactor Innovative and Secure (IRIS) design for general deployment, also creates challenges for condition monitoring. The IRIS reactor was chosen as the Integral Primary System Reactor (IPSR) for this research (Storrick et al. 2005). The challenge is that of creating a data-based model for a reactor that has never been built or operated and has no historical data. This is the

motivation behind the creation of a hybrid modeling technique that adapts to include new reactor data while still maintaining the ability to monitor the condition of the reactor accurately. The adaptive nonparametric model (ANPM) is a hybrid model that is initially based on first principle models (FPMs) and has been designed with the ability to adapt to new reactor data. New reactors that have never been operated can take advantage of this adaptive modeling capability. These techniques would give the desired dynamic capability to the monitoring of the reactor. As the reactor or system operates under non-fault conditions it generates data that can be used for more accurate modeling, this is one of the motivations behind adaptive modeling. The performance of the model will increase as more operational data becomes available and is implemented into the model.

FPMs are used for design and development of new systems and processes; however, these models rarely allow for the sensitivity necessary for early detection. Empirical techniques are more recent approaches that have the sensitivity needed for early detection (Hines et al. 2007a). Empirical models are applicable for systems with available operating data recorded over the span of the operating range. However, empirical models cannot be used to extrapolate outside the training data range (Hines et al. 2007b). FPMs have the advantage for systems that are new and have no prior operational data. The ANPM is a hybrid model that capitalizes on the strengths of the two model types and attempts to mitigate their shortcomings.

1.1 BACKGROUND

Adaptive models used for system monitoring must strike a balance between stability and elasticity. An adaptive model should have the ability to adapt to new data without losing the

ability to differentiate faults from nominal conditions. Ideally, an adaptive model should produce accurate results while having the ability to change the data basis of the model and expand beyond the training limits.

New reactors that have never been operated can take advantage of this adaptive modeling capability. These techniques would give the desired dynamic capability to the monitoring of the reactor. As the reactor or system operates under non-fault conditions it generates data that can be used for more accurate modeling; this is one of the motivations behind adaptive modeling. The performance of the model will increase as more operational data become available and are incorporated into the model.

Another motivation behind an adaptive model is to accurately model new system operations. This new operation could be because of slightly different plant designs, assumptions made within the FPM, environments that have not previously been seen, or other causes. The model's adaptive ability provides more accurate predictions while generating an improved model. Using a nonparametric modeling structure provides an ideal base which makes adaptation immediate and effortless. Parametric modeling structures are not suitable for easy adaptation, instead at every adaptation step parameters need to be recalculated, making them problematic.

The two main abilities an adaptive model should have are the changing of the data basis, and the expanded condition monitoring. The first ability is used to adjust the data basis of the model, and the second ability is for expanded condition monitoring. Both abilities are important for empirical modeling in an adaptive environment. The Adaptive Non-Parametric Model (ANPM) has been developed for this purpose and for integration into a monitoring, diagnostic, and prognostic system for use on small and medium size reactors (SMR). There are multiple

applications for such a model and despite the purpose of the ANPM's development there are other areas of promise. This research has focused on the theory, development, and testing of the ANPM.

The first ability of the ANPM is to gradually change the data basis of the model. The ANPM's original intent is to adapt a nonparametric model's memory matrix from data created using a first principle model (FPM) to the system's actual un-faulted data. This could also be done using less accurate data or a low fidelity model and adapt to the system's more accurate values. This is useful for monitoring new system designs from first construction and operation when the only available data is from FPMs. The FPM's data is used to build the best possible models initially, but during the system's operation new data can be collected that are more accurate for future predictions.

The second ability of the ANPM is expanded condition monitoring (ECM). This ability can monitor for expanded conditions and determine if this condition is due to a fault or a new range of operation which should be added to the model. This gives the model the ability to expand beyond the original training data's range which is different from non-linear modeling techniques which cannot extrapolate for predictive purposes. A new principal component analysis (PCA) ECM technique is proposed for use within the ANPM. This ability is useful in uncertain systems where the range of the model could be exceeded by the system when under non-fault conditions. A method for differentiating between expanded and a fault condition for use in an adaptive model is described. Statistical process monitoring methods using principal component analysis (PCA) have been extensively studied and heavily applied in the chemical industry. A thorough literature review is given on the past applications of such methods,

highlighting the applications strengths and weaknesses. Some traditional fault detection techniques are discussed and evaluated for the dynamic environment of adaptation. A comparison between traditional fault detection and monitoring techniques and the newly proposed expanded process differentiation technique is discussed. Adaptive modeling requirements in a dynamic environment are shown to extend beyond the traditional modeling requirements. The basic approach to PCA is described, with a detailed description of the differentiation method. The Hotelling's T^2-statistic and the squared prediction error (SPE), also known as the Q-statistic, are used as indicators of the condition of the system. Dependence on the internal linear relationships of the data is discussed. This multivariate differentiation technique is applied to a simulated data set and is shown to correctly differentiate between expanded operating conditions and faulted conditions in an adaptive environment.

This dissertation highlights the areas of research that have been explored and the corresponding results. The theory behind the ANPM is discussed and the uses of this type of modeling techniques are shown.

1.2 ORIGINAL CONTRIBUTIONS

Extensive research has been performed on the creation of new monitoring, diagnostic, and prognostic techniques. This research has been focused on improving the prediction capabilities of an equipment surveillance system. Within the monitoring techniques a plethora of new models have been developed and used on a variety of problems. These prediction algorithms range from artificial neural networks to traditional linear regression techniques. In this study, a new type of monitoring technique will be introduced. This is the ANPM, which is a hybrid model that takes

advantage of the strengths of the FPMs and empirical models. Rather than have a static base, the ANPM uses a dynamic modeling structure. This gives the model the ability to evolve over time into a more accurate model. An outline of the original contributions is presented below.

1. Development of an adaptive approach that can be used in a nonparametric modeling environment. This approach allows the model to evolve over time. This increases the predictive capability of the model. Included in the ANPM is a sequence of optimum vector addition techniques, FPM vector deletion techniques, evolving vector selection, and a combination of fault detection and expanded condition monitoring techniques.

2. Development of a principal component analysis (PCA) based expanded condition monitoring (ECM) technique. This ECM technique differentiates between an expanded nominal condition and an expanded fault condition. This technique is one of the major contributors to the ANPM's ability to adapt.

3. Integration of the ANPM into a complete monitoring, detection, identification system. This allows a complete equipment surveillance system to have a base dynamic modeling structure that will increase the performance of the complete system.

4. Development and analysis of kernel principal component analysis (KPCA) technique for ECM applications. The KPCA ECM is shown to be not effective as a differentiation method between expanded nominal condition and an expanded fault condition, for use with non-linear sensor relationships.

5. Development of MATLAB based software that implements the ANPM and ECM techniques.

This dissertation covers the development of the ANPM and the applications of this technique. A literature review is given that discusses the details of condition based monitoring, diagnostics, and prognostics and their usage for adaptive environments. This is used in the development of the current ANPM. Details of the PCA ECM technique are discussed with use of data from heat exchanger models and an experimental flow loop to show the adaptive ability of the ANPM. Generated data was also used to show the PCA ECM ability to perform accurately. The threshold fault detection technique is used in conjunction with the PCA ECM to give the ANPM an overall fault detection and expanded condition monitoring capability. The overall methods used in the development of the monitoring and fault detection techniques are discussed.

1.3 DISSERTATION ORGANIZATION

The dissertation is organized within five chapters followed by the references, appendices, and vita. Chapter 1, the introduction, includes background, original contributions and document organization. This is followed by chapter 2, literature review, which gives a detailed survey of prediction, fault detection, diagnostics, and prognostic techniques with a focus on giving a complete survey with relevance to the contributions. This is followed by methodology in chapter 3. Methodology gives an in depth description of the techniques that were created and the algorithms that were used to accomplish the research goals. Next, the results for the given methods are shown in chapter 4. Finally, the conclusion in chapter 5, which summarizes the

research that was undertaken, discusses the original contributions and the research that was done to accomplish the given tasks, and gives recommendations for future work.

The introduction is concentrated into three sections, the first section is the background. The background focuses on the motivation and need for this research with a brief overview of the desired abilities. The second section of the introduction is the original contributions, this highlights the five original contributions that are credited to this research. And the third and final section of the introduction is the document organization which discusses the complete dissertation and a description of each chapter and the corresponding content. The next chapter is the literature review which gives an in depth analysis of past research that is related. The literature review has four major sections, prediction, detection, diagnostics, and prognostics.

2 LITERATURE REVIEW

An equipment surveillance system can be constructed for monitoring, fault detection, diagnosis, and prognosis of complex systems. Fig. 1 shows the basic connection among monitoring, diagnostics, and prognostics. The research presented in this dissertation focuses on the monitoring, fault detection, and diagnostics portions of an equipment surveillance system. There is a great deal of research that focuses on these areas. It would be unreasonable to do a complete literature review on these areas; however, an overview of the past research that is relevant to the research presented is given. A literature review on prediction techniques is given in section 2.1, on fault detection in section 2.2, on diagnostics in section 2.3, and on prognostics in section 2.4. This provides a literature review that covers the individual pieces of a complete equipement surveillance system.

An equipment surveillance system could be used to supervise a process according to definition put forth by Isermann (1984). A popular approach uses a monitoring module or system model to predict the state of the system. The predictions are compared to the systems nominal values to produce residuals, which can be used to estimate the state of the system, then to diagnose the faults, and ultimately used in the prognosis of the system. A plethora of research has been conducted on equipment surveillance systems and the individual contributing pieces of such a system. Some of this research can be seen in the work of Isermann (1984, 1995, 2004), which looks at model based fault detection and diagnosis of complex systems.

Research by Garvey (2007) developed a complete monitoring, diagnostic, and prognostic framework that used a nonparametric fuzzy inference system (NFIS). The prognostic technique

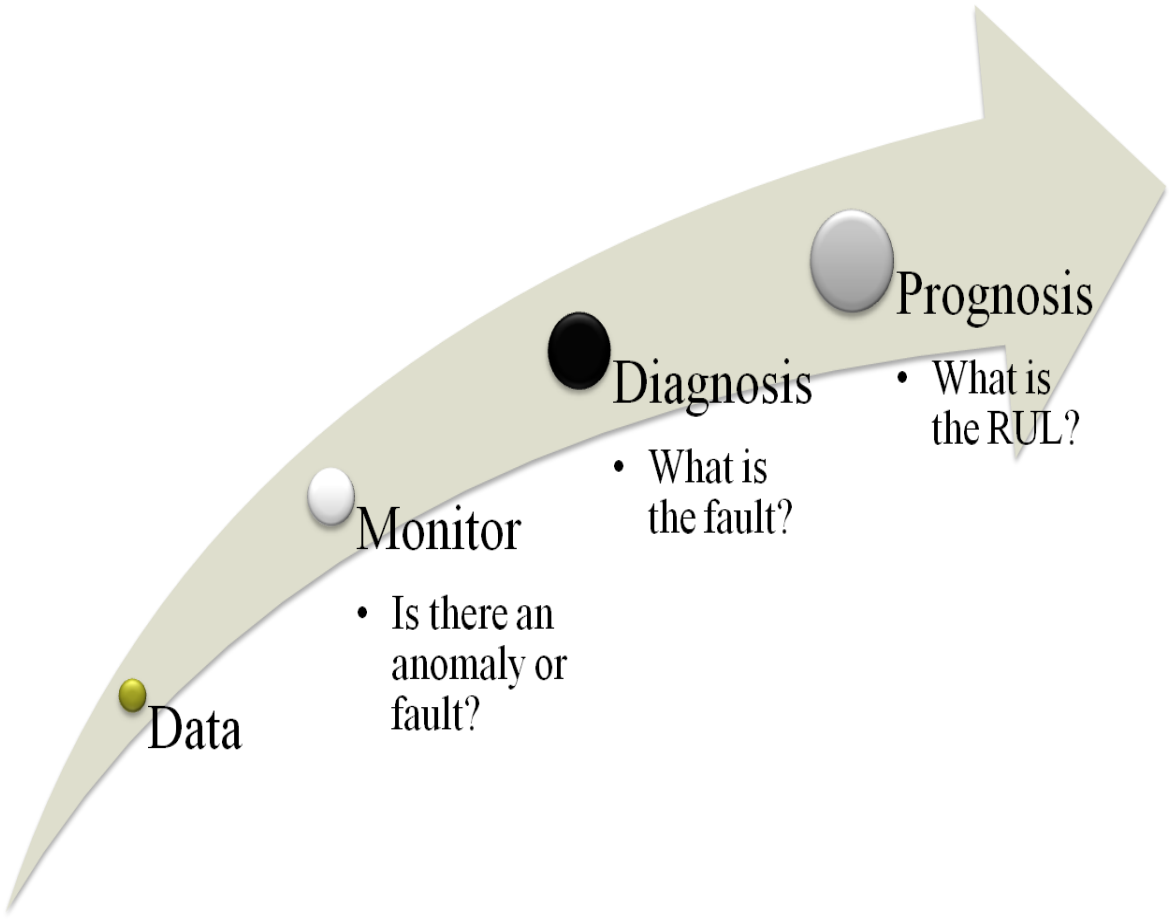


Figure 1: Equipment Surveillance System (Hines 2006)

that was used was the path classification estimation (PACE), which looked at a comparison of previous exemplar paths to the current degradation.

There is a huge selection of differing techniques for designing a complete equipment surveillance system. The proposed research focuses on the monitoring or prediction and fault detection portion of an equipment surveillance system. A literature review that covers prediction and fault detection techniques and relates them to the current research is given below in the following sections.

2.1 PREDICTION

There are a number of different monitoring techniques that have been proposed within the literature. These techniques range from first principle models (FPM), that use physics of the system to predict the nominal conditions of the process, to empirical models that use actual system data to build prediction models. An example of using a FPM is shown in the research on the Tokamak energy confinement by Kotschenreuther et al. (1995). These models can be built during the design phase of the system. There are a number of benefits and drawbacks to using a FPM. One of the benefits of a FPM is the physical equation base which gives the model a direct physics based foundation. Experimental data from the system can be used to verify the FPM and test the accuracy of such a model. Despite the benefits of a FPM there are a number of drawbacks. These include the engineering time and understanding that is required to build a good FPM and the lack of sensitivity needed to detect slight changes in the system. Due to the complex nature of building a FPM it is rare to have a FPM that incorporates the complete system; it is common for a FPM to include portions of the system.

Empirical or data based models have a number of benefits and drawbacks as well. These types of models are built using process parameter measurements that have been collected over the range of the system or process of interest. The relationships between these measurements are used within the model architecture to produce accurate predictions. One of the drawbacks of empirical models is the limitation on accuracy of the models. Since the relationships are usually non-linear the models are only accurate when applied to similar operating conditions as the data used to create the model. When the model predictions are extrapolated outside the training range the results are unreliable. This is different from linear models where predictions can be extrapolated outside the training range.

Hybrid models are the combination of FPM and empirical models. These models provide the sensitivity of empirical models while providing the robustness of FPM. The ANPM is a hybrid model that takes advantage of the benefits supplied by both the modeling techniques. When the system progresses outside the training range the FPM can be used to give robust results, however, when the system is operating within the training range the empirical model base can be used. In the next section, a survey of parametric models is given.

2.1.1 Parametric Models

There are two commonly used types of empirical model architectures, parametric and non-parametric. In a parametric model a set of parameters are used to define the functional relationship of the system. An example of a common parametric model is linear regression with p predictor variables and n observations.

$$y_i = \beta_1 x_{i1} + \ldots + \beta_p x_{ip} + \varepsilon_i \quad \textbf{(2.1)}$$

This can be described in matrix form as:

$$Y = X\beta + \varepsilon \tag{2.2}$$

where

$$y = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix}, \quad X = \begin{pmatrix} x'_1 \\ x'_2 \\ \vdots \\ x'_n \end{pmatrix} = \begin{pmatrix} x_{11} & x_{12} & \cdots & x_{1p} \\ x_{21} & x_{22} & \cdots & x_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n1} & x_{n2} & \cdots & x_{np} \end{pmatrix}, \quad \beta = \begin{pmatrix} \beta_1 \\ \vdots \\ \beta_p \end{pmatrix}, \quad \varepsilon = \begin{pmatrix} \varepsilon_1 \\ \varepsilon_2 \\ \vdots \\ \varepsilon_n \end{pmatrix} \tag{2.3}$$

Parametric architectures use the training data to determine the parameters of the model that fit a predefined mathematical model. This is normally done by minimizing some objective function, such as the sum of squared error (SSE) or some other error metric. A number of parameter estimation techniques have been developed. One of the most popular methods is ordinary least squares (OLS); this method minimizes the sum of squared residuals, and leads to a closed form expression.

$$\hat{\beta} = (X'X)^{-1}X'y \tag{2.4}$$

Another parameter estimation technique is the generalized least squares (GLS), which is an extension from OLS where a weighted sum of squared residuals is minimized. This is used when heteroscedasticity or correlation is present within the error terms. Maximum likelihood estimation (MLE) is a popular technique that is used to fit a statistical model with data and a parameter estimation technique. MLE can be performed when the distribution of the error terms are known; when this is a normal distribution with mean zero the results are the same as OLS.

The parameters for parametric models are defined in finite-dimensional parameter space. There has been a lot of research done on parametric models, too much to include a complete literature review of the topic. However, interested readers are referred to Bickel et al. (2001), Davidson (2003), Freedman (2009), and Seber et al. (2003). For more specific modeling needs, the reader is referred to Haykin (1994) for neural networks and Draper et al. (1966) for ordinary least squares (OLS) regression.

Unlike parametric models that specify the structure before the analysis, a non-parametric model does not specify a structure. Non-parametric models may have parameters; however, these parameters have flexibility and are not defined beforehand. Past data exemplars are stored in the memory and used as the actual model. When a query is made the non-parametric model performs a weighted regression of the training exemplars within the vicinity of the query. The proximity of the training exemplars to the query designated the corresponding weights. There are benefits and drawbacks of both empirical modeling techniques. Atkeson et al. (1997b) describes some of the benefits and drawbacks of non-parametric models, Hines et al. (2006b) addresses some of the drawbacks by proposing a robust vector selection method that improves the quality of the locally weighted regression models. For a complete overview of non-parametric techniques the reader is referred to Wasserman (2007) and Gibbons (2003).

The ANPM system is designed for use with non-parametric models because of the retraining inherent in the adaptive model architecture. Non-parametric models have distinct advantage over parametric models when retraining for new operating conditions. Retraining a parametric model requires recalculating the parameters at each retraining interval. Conversely,

retraining a non-parametric model involves simply appending the memory matrix with new observations and possibly removing old, unnecessary observations.

Given in this section was a review over parametric models. In the next section, a survey of kernel regression prediction methods is discussed.

2.1.2 Kernel Regression

Kernel Regression (KR) is defined by Atkeson et al. (1997a) as the process of estimation by using a weighted average of historical exemplar observations. The outputs of an AAKR model are the corrected values of the inputs (Fig 2).

The base empirical model behind the ANPM is the AAKR model. AAKR is a non-parametric model that uses past normal operational data to correct faulty observations which may be due to system degradation, sensor faults, data acquisition problems, etc. The Nadaraya-Watson estimator shown below is the general KR estimator. Nadaraya (1964) and Watson (1964) proposed a method using a kernel as a weighting function for estimation.

$$\hat{m}_h(x) = \frac{\sum_{i=1}^{n} K_h(x - X_i) Y_i}{\sum_{i=1}^{n} K_h(x - X_i)} \qquad \textbf{(2.5)}$$

An important step in developing an AAKR model is variable selection. Variable selection is used in determining which predictors should be included within the model. There are a number of variable selection techniques (Hines 2008b); however, for this research the correlation analysis approach is used. For an autoassociative model the predictors are the measured process variables, it is desired to have correlated variables within the model. If the whole system is to be

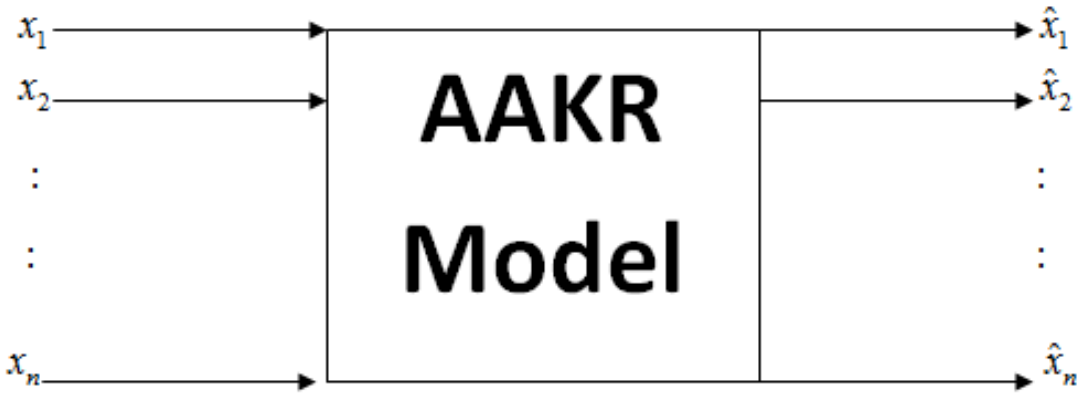


Figure 2: AAKR model

modeled then a variable grouping technique can be used. The variable grouping technique optimizes the groups to include variables that are highly correlated with each other. This results in a number of models that together describe the system. Variables that are included in a model but are not beneficial to a model add variance to the predictions. And variables that are beneficial but are not included into the model cause a bias in the predictions. Uncertainty can be defined as the square root of the sum of bias squared and variance.

$$U = \sqrt{bias^2 + \sigma^2} \tag{2.6}$$

Vector selection also plays an important part in the overall performance of an AAKR model. A subset of the exemplar observations can be used to characterize the training data set. Using vector selection can increase the performance of the model over using the complete training data set (Hines 2008b). There are a number of different vector selection techniques that can be used in determining which vectors best represent the training space. One of the common traditional approaches is the Min-Max technique. The Min-Max technique separates the training data into bands and then uses the vectors that include the minima and maxima for each sensor

within each band (Fig. 3). Another common technique is vector ordering, which orders the vectors based on their vector norms, and then periodically samples through the training data set (Fig. 4).

Clustering is another technique that is used in vector selection. Adeli-Hung and Fuzzy C-means clustering is used to cluster the training data and then observations that are close to the cluster centers are used (Fig. 5). The clustering techniques takes more computational resources however, it selects higher quality observations.

The AAKR model uses a distance metric to compare the input query data to the exemplar vectors, which make up the model's memory matrix. There are a number of distance metrics that can be used. The Euclidean distance, which is known as the L^2-norm, is used as the distant metric within the ANPM. The distant metric is then used as an input into the kernel similarity function. The kernel function is used to calculate the weights for each observation. There are a number of kernel functions that could be used such as the exponential, inverse distance, biquadratic, uniform weighted, and tricube kernel (Atkeson et al. 1997a). There are more advanced kernel functions that have been proposed such as the Hermite kernel (Zavaljevski et al. 2000a, 2000b) and the asymmetric Gaussian kernel (Mackenzie et al. 2004). A kernel function should have a large weight for small distances and small weight for large distances. There are advantages and disadvantages specific kernels for each situation, it has been shown that the kernel function does not have a large impact on the performance of the locally weighted models (Scott 1992; Cleveland et al. 1994a; Cleveland et al. 1994b). A common kernel is the Gaussian kernel (Fan et al. 1996a); this kernel is a function of the Euclidean distance and the kernel

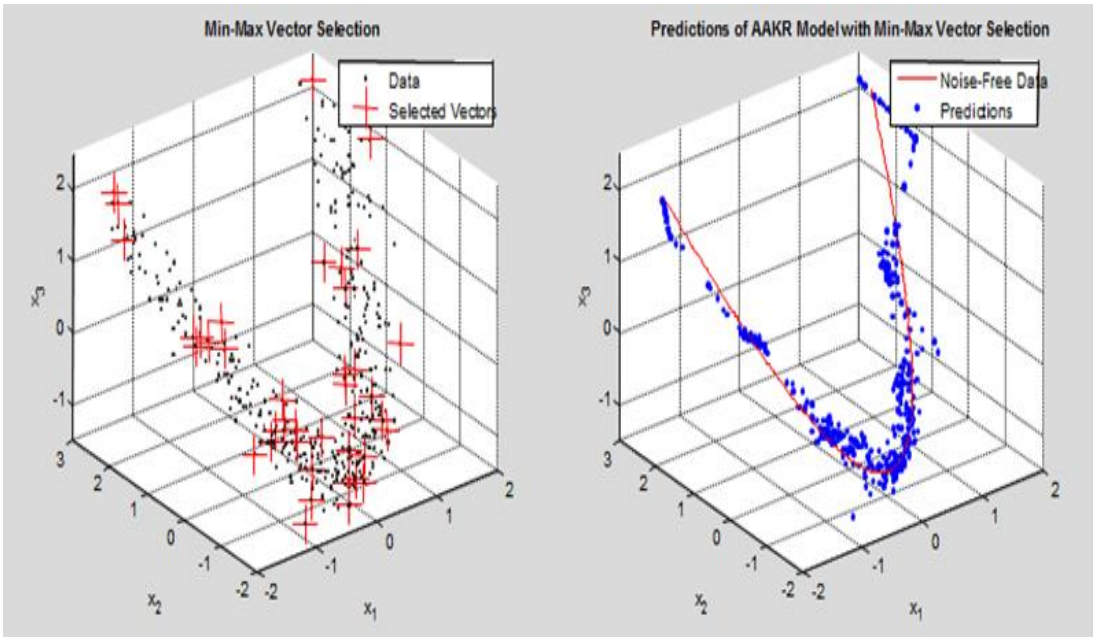


Figure 3: Min-Max Vector Selection Results (Hines 2008b)

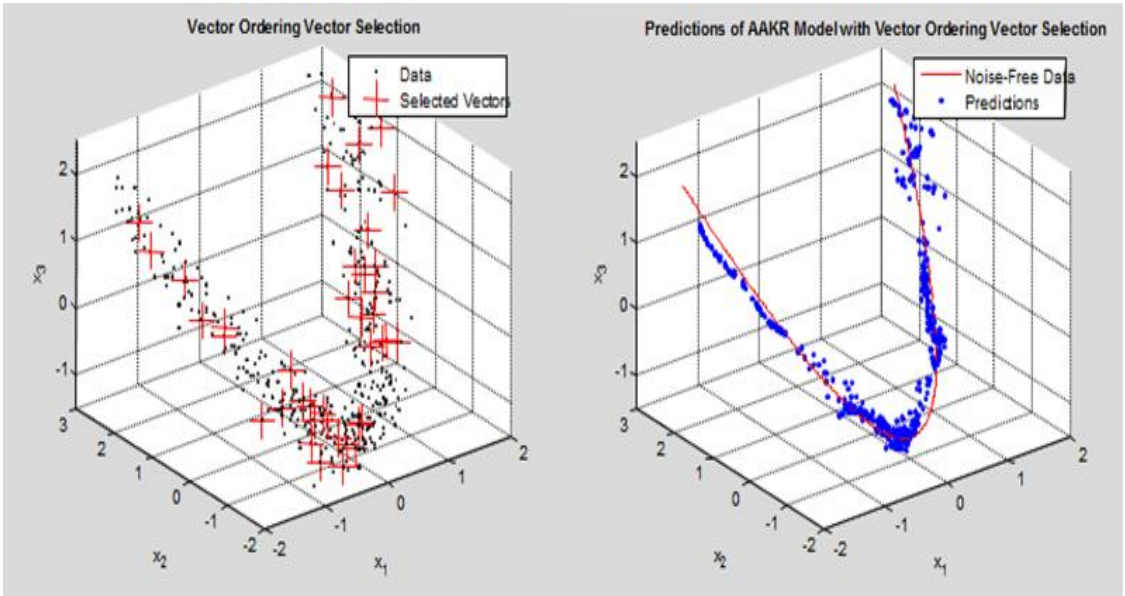


Figure 4:Vector Ordering Vector Selection Results (Hines 2008b)

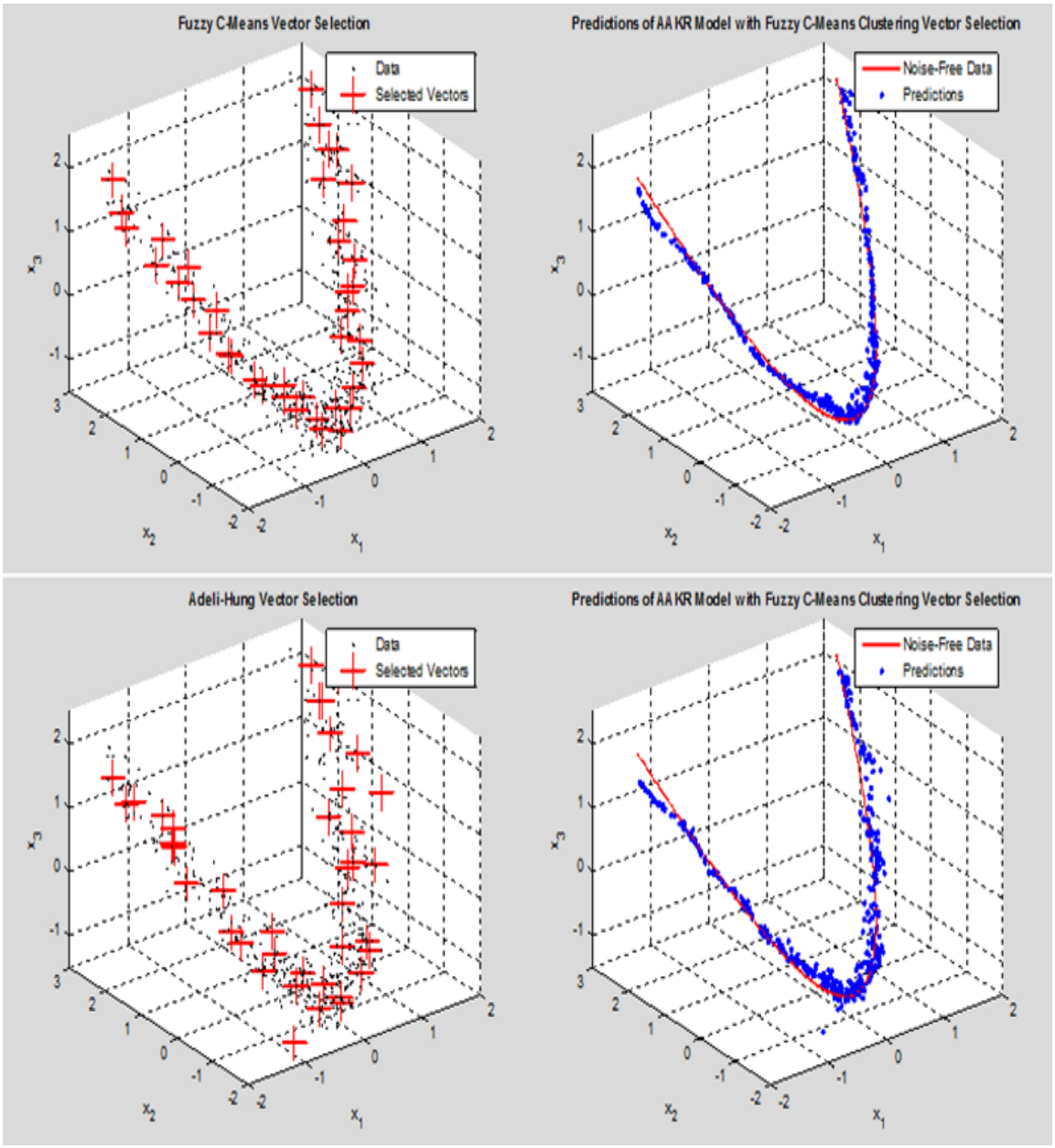


Figure 5: Fuzzy C-mean and Adeli-Hung Vector Selection Results (Hines 2008b)

bandwidth *h*. The ANPM uses a Gaussian kernel function with the Euclidean distant metric. In general the bandwidth is optimized by minimizing the error or uncertainty for a test data set. Automatic procedures for bandwidth selection have been developed along with variable bandwidth (Fan et al. 1995; Fan et al. 1996b). This is done initially, before applying the model for system monitoring, to find the optimal bandwidth. Uncertainty or mean integrated squared error (MISE) can be defined as the sum of the bias squared and the variance. By minimizing the error the model's performance is optimized by the choice of the proper bandwidth. An example of this can be seen in Fig. 6.

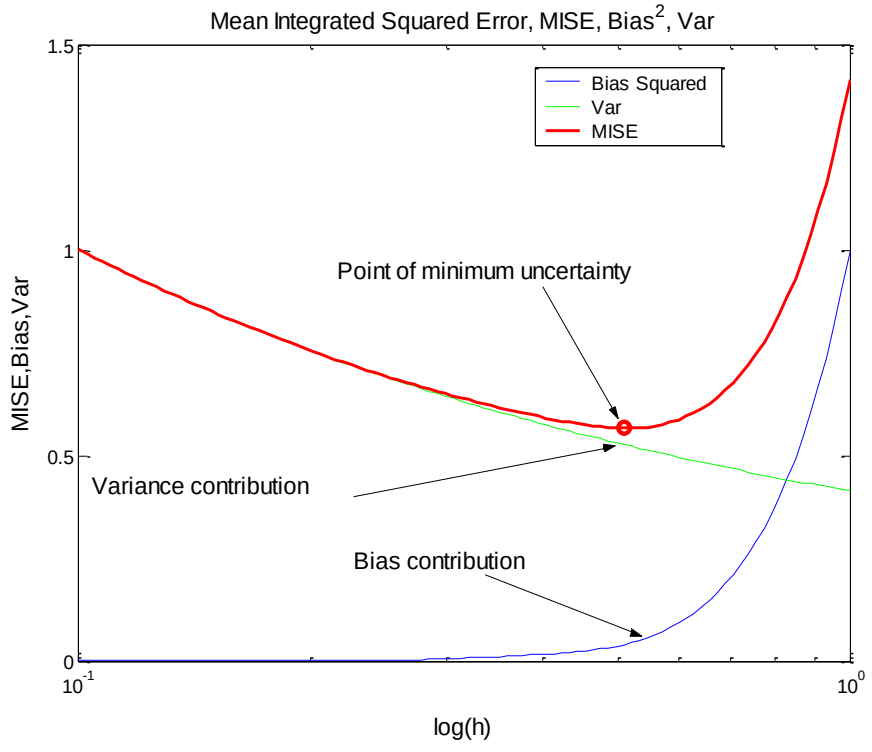


Figure 6: Optimum Bandwidth (Hines 2008b)

A large bandwidth results in a bias and a small bandwidth results in an increase in variance. The optimum bandwidth minimizes the affects of the bias and variance together. The affect of a large bandwidth is shown in Fig. 7, a small bandwidth is shown in Fig. 8, and with the optimum bandwidth in Fig. 9.This process of finding the optimum bandwidth is done before applying the model for system monitoring.

The final AAKR model consists of, the variables that were chosen using the variable selection techniques, the vectors that were chosen to represent the training data using the vector selections techniques, and the optimum bandwidth. After the development of the model the final prediction of an AAKR model is a weighted sum of the exemplar vectors.

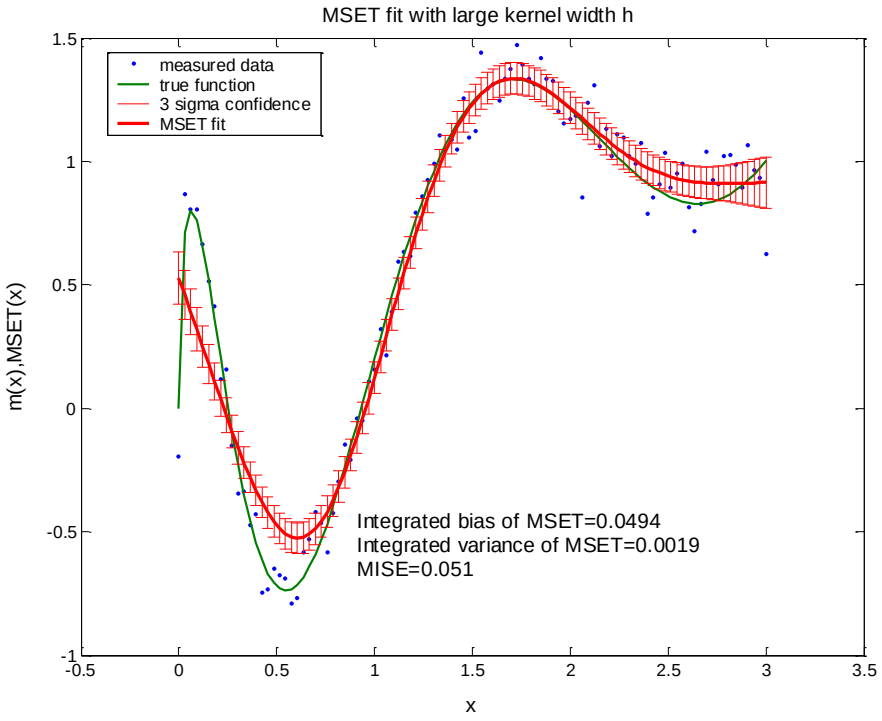


Figure 7: Large Bandwidth (Hines 2008b)

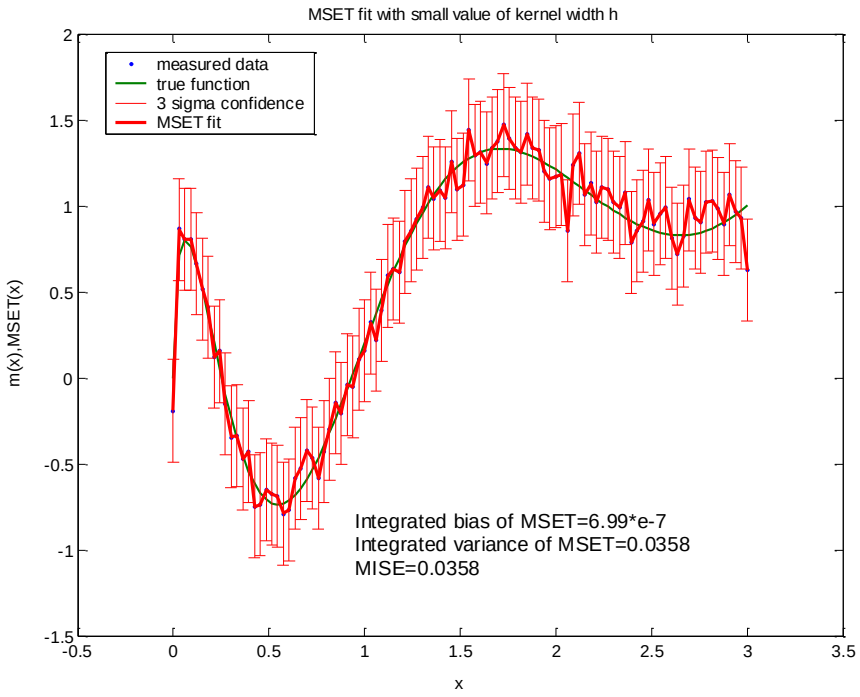


Figure 8: Small Bandwidth (Hines 2008b)

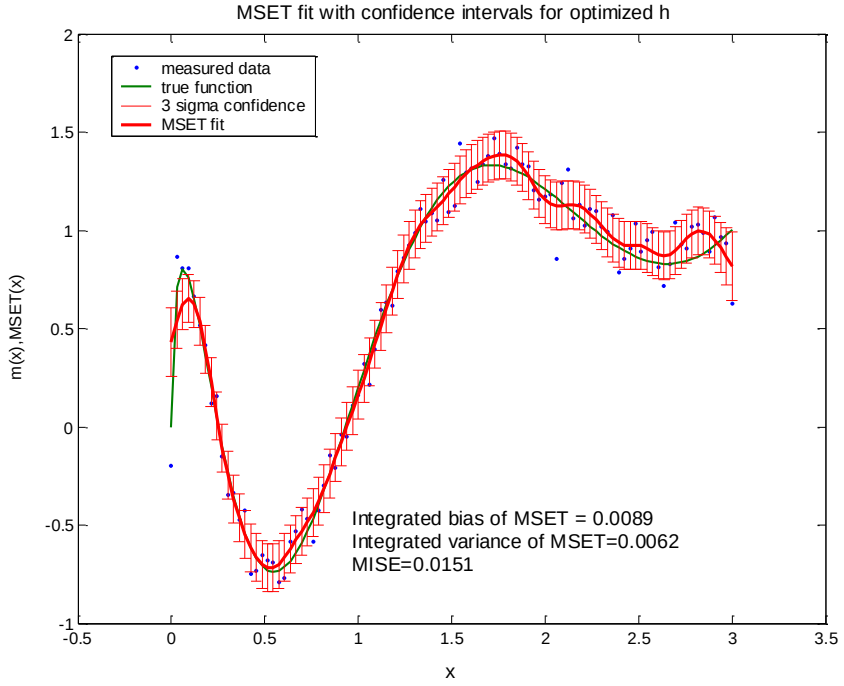


Figure 9: Optimum Bandwidth (Hines 2008b)

In this section a survey on AAKR prediction methods was given. In the next section, a survey of PCA prediction methods is discussed.

2.1.3 Principal Component Analysis

Principal Component Analysis (PCA) has been widely used throughout process monitoring. The chemical industry has applied PCA heavily as a multivariate monitoring method (Wang et al, 2001a; Wang et al, 2002; Lin et al 2000; Kresta et al, 1991; Piovoso et al, 1992). PCA has also been used as a viable way for fault identification and reconstruction, this is important for corrective actions (Dunia et al. 1998). The use of PCA models have been highlighted in the use of monitoring temperature sensors in a nuclear research reactor (Penha et al. 2001).

In general PCA is a decomposition method which is used to reduce the dimensionality of the data set. This reduction in dimensionality is used to create two subspaces: the principal component (PC) subspace and the residual subspace. The PC subspace includes the orthogonal components of the data; the PC subspace contains the majority of the variation within the data set (Jolliffe 2002). The residual subspace is the orthogonal components of the data that are not included in the PC subspace; the residual subspace contains a small amount of the variation, normally attributed to the noise within the data set. The PC subspace is the dimensions that the PCA model represents. This decomposition is done by transforming the data into the PCs, which are uncorrelated (Jackson 1991). The PCs are ordered by the amount of variation described within, and then divided into the PC and residual subspaces.

Two commonly used statistical indices are the Hotteling's T^2-statistic and the squared prediction error (SPE), or the Q-statistic. These statistics can be used to indicate where a data

point is with respect to the population. The T^2-statistic describes the variation within the model, and the Q-statistic describes the variation outside the model. A good way to visualize the T^2-statistic and the Q-statistic is shown in a three dimensional example. Two principal components are used and one residual component. Fig. 10 shows this example with the T^2-statistic and the Q-statistic highlighted for a particular data point.

This results in one of four possible outcomes (Wang et al. 2002):

1) Both the T^2-statistic and Q-statistic are within their limits.
2) T^2-statistic is outside its limit and Q-statistic is within its limit.
3) Q-statistic is outside its limit and T^2-statistic is within its limit.
4) Both the T^2-statistic and Q-statistic are beyond their limits.

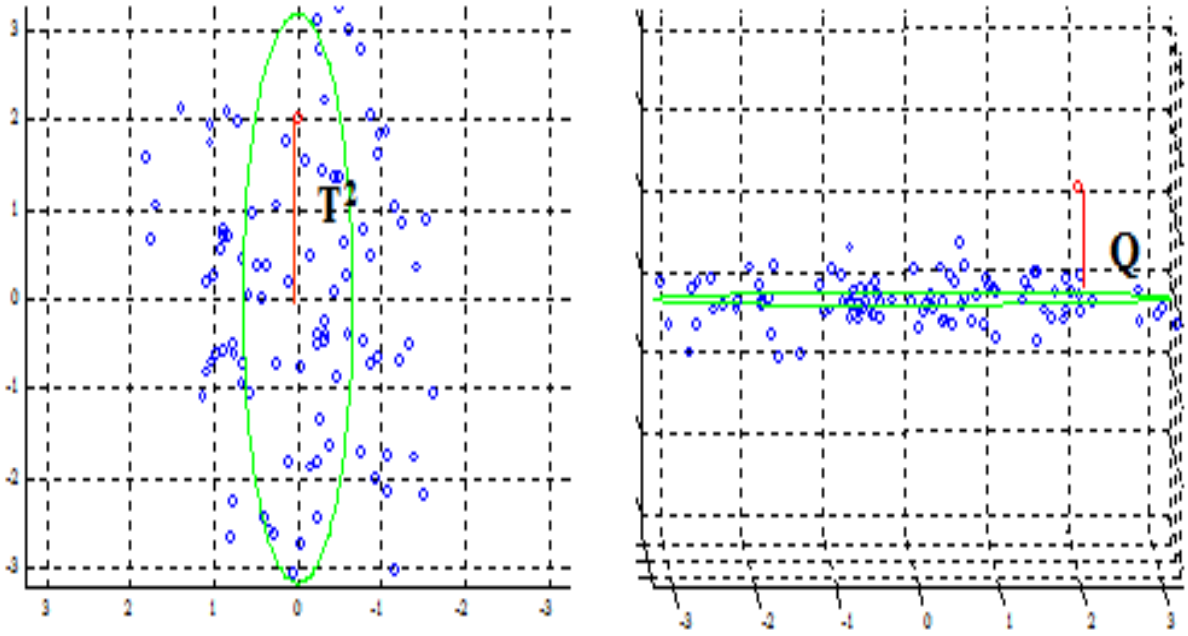


Figure 10: Example T2 and Q statistics

It has been shown that the T^2-statistic may also be an indicator of a fault condition (Wang et al. 2002). When and increase in the T^2-statistic alone indicates a change has occurred that is consistent with the developed model (Dunia et al. 1998). This signifies that outcomes 2 and 4 can be indicators of a fault within the system. The Q-statistic is used in general to monitor the correlations between the variables. An increase in the Q-statistic would signify a change within the correlations or the variables which would result in a fault being detected (Bakshi 1998; Dunia et al. 1998). There are a number of fault detection and identification approaches that use a PCA base. One identification approach is the analysis of the contribution of each variable to the Q-statistic (MacGregor et al. 1994). Other approaches such as a sensor validity index (SVI) which is used to identify sensor faults have been proposed (Dunia et al. 1996). The purpose of the SVI is to indicate the validity of each sensor and look at the use of this technique when looking at four types of sensor faults.

1) Bias
2) Complete Failure
3) Drift
4) Precision Degradation

This monitoring process can be successful when the linearity assumption is held.

When data contains nonlinear relationships, PCA monitoring can result in poor performance (Dong et al. 1996). Many non-linear techniques have been developed to deal with these types of relationships. A method that uses principal curves and neural network is proposed by Dong et al. (1996). Another method proposed by Fourie et al. (2000) is nonlinear multiscale

principal component analysis (NLMSPCA). This method utilizes multilevel wavelet decomposition and neural networks to apply a nonlinear PCA to process monitoring and fault detection. One of the most popular non-linear PCA techniques is kernel principal component analysis (KPCA) which was first proposed by Scholkopf et al. (1998). KPCA has many advantages over other non-linear techniques; this advantage is mostly due to the simplicity of application (Cho et al. 2005). Another attractive attribute of KPCA is the flexibility in component designation; the components do not need to be designated prior to modeling (Choi et al. 2004; Lee et al. 2004). The superiority of KPCA over PCA for non-linear relationships has been investigated thoroughly (Choi et al. 2004; Choi et al. 2005; Cui et al. 2008). There has been a plethora of research on advancements of KPCA for process monitoring and fault detection. One such example is kernel independent component analysis (KICA) which works in conjunction with KPCA and has been proposed for multivariate statistical process monitoring (Zhang 2009).

There are many interesting publications on these techniques and their applications. This research, however, focuses on the applications of PCA to solve the problem of differentiating between faulted conditions and expanded nominal conditions. There could be potential use of more advanced PCA techniques and extensions of them for future applications within adaptive modeling. An example is given of the use of PCA in differentiating between expanded and fault conditions for linear systems.

A brief overview of PCA is given; the interested reader is referred to Jackson (1991) and Jolliffe (2002) for a more complete description. Consider a normalized data matrix X (mxn)

which contains process data, where m is the number of observations and n is the number of measured variables. This matrix can be decomposed as:

$$X = \hat{X} + E \tag{2.7}$$

This decomposition represents X as a predicted value $\hat{X}$ and the error value E, where $\hat{X}$ is the modeled portion of X and E is the portion excluded from the model. The principal component subspace is covered by $\hat{X}$ and is orthogonal to the residual subspace which is covered by E.

$$\hat{X} = TP^T = \sum_{i=1}^{l} t_i p_i^T \tag{2.8}$$

$$E = T_e P_e^T = \sum_{i=l+1}^{m} t_i p_i^T \tag{2.9}$$

Where P is the loading matrix and T is the score matrix. The principal component loadings are set as the first $P(n \times l)$ and the corresponding scores are the first $T(m \times l)$. The residual loadings T_e are set as the last $n-l$ column vectors, and the residual scores, P_e are the last $n-l$ row vectors. A choice of $l < n$ is made to reduce the model dimensions and set the cutoff of $\hat{X}$ from E. Singular value decomposition (SVD) of the correlation matrix R of X can be used as shown:

$$R = U\Lambda^{\frac{1}{2}}U^T \tag{2.10}$$

where

$$R = \frac{(X^T X)}{(n-1)} = \frac{1}{n-1}\left[\hat{X}^T\hat{X} + E^T E\right] \tag{2.11}$$

and

$$U = [P^T \quad P_e^T] \tag{2.12}$$

The scores are defined as:

$$[T \quad \hat{T}] = U\Lambda^{\frac{1}{2}} \tag{2.13}$$

The diagonal matrix Λ is the eigenvalues of the correlation matrix R.

Two commonly used statistical indices are the Hotteling's T^2-statistic and the squared prediction error (SPE), or the Q-statistic. These statistics can be used to indicate where a data point is with respect to the population. The T^2-statistic describes the variation within the model, and the Q-statistic describes the variation outside the model.

$$T^2 = T\Lambda_k^{-1}T^T \leq \delta_T^2 \tag{2.14}$$

$$Q = ee^T = X(I - P_e P_e^T)X^T \leq \delta_Q^2 \tag{2.15}$$

Hard limits such as δ_T^2 and δ_Q^2 for these statistics can be set. This can be done by determining the sampling distributions and the limits based on the confidence limits that are desired (Jackson et al. 1979; Jackson 1991). The fault detection process that is followed is the examination of the Q-statistic and its limit. So the process is within normal operation when (Dunia et al. 1998):

$$Q \leq \delta_Q^2 \tag{2.16}$$

The Q-statistic is used in general to monitor the correlations between the variables. An increase in the Q-statistic would signify a change within the correlations or the variables which would indicate a fault (Bakshi, 1998). It has been shown that the T^2-statistic may also be an indicator of a fault condition when the Q-statistic has not exceeded its limit (Wang et al. 2002). This signifies that outcomes 3 and 4 can be indicators of a fault within the system. This

monitoring process can be successful when the linearity assumption is held. When nonlinear relationships are within the data, PCA monitoring can result in poor performance (Dong et al. 1996). Many non-linear techniques have been developed to deal with these types of relationships.

In this section a survey on PCA based prediction methods was given. In the next section, a survey of hybrid modeling methods is given.

2.1.4 Hybrid Modeling

Hybird modeling uses the strengths of data-based models and first principle models to take advantage of the strengths of each of the techniques while reducing the weaknesses (Wilson and Zorettso 1997; Thompson and Kramer 1994; te Braake and van Can 1998). Hybrid models are also referred to as semi-mechanistic models or "grey box" models. There are a number of different techniques that have been applied to this type of modeling. First principle models (FPM) can require long developing times and can be difficult to obtain the desired sensitivity for early fault detection. Data-based models have the sensitivity to detect plant degradation for early fault detection (Gribok et al. 2001); however, data-based models are only accurate when applied to the same operating conditions. There are two main hybrid modeling approaches, the serial approach and the parallel approach.

The proposed ANPM is a hybrid model that capatalizes on the strengths of data-based modeling and first principle modeling which minimizes the weaknesses of the two modeling types. The hybrid modeling structure used within the ANPM uses a first principle model to first populate the original memory matrix and during adaptation system data is appended onto the

memory matrix while FPM data is slowly deleted (Fig. 11). This adaptation only occurs during normal system operation and does not happen when faults and abnormalities are detected. This is quite different from the two main hybrid approaches, the serial (Fig. 12) and parallel approach (Fig 13).

The serial hybrid modeling approach was first proposed by Psichogios and Ungar (1992) and applied to a simulated fed-batch bioreactor. Alessandri and Parisini (1997) applied a similar approach to a 320MW power reactor. The serial hybrid modeling approach uses a data-driven model to predict the parameters for use in the first principle models, the first principle models are then used to model the system.

The parallel hybrid modeling approach was first proposed by Kramer et al. (1992) and used to correct the FPM predictions in a Continuous Stirred Tank Reactor (CSTR).

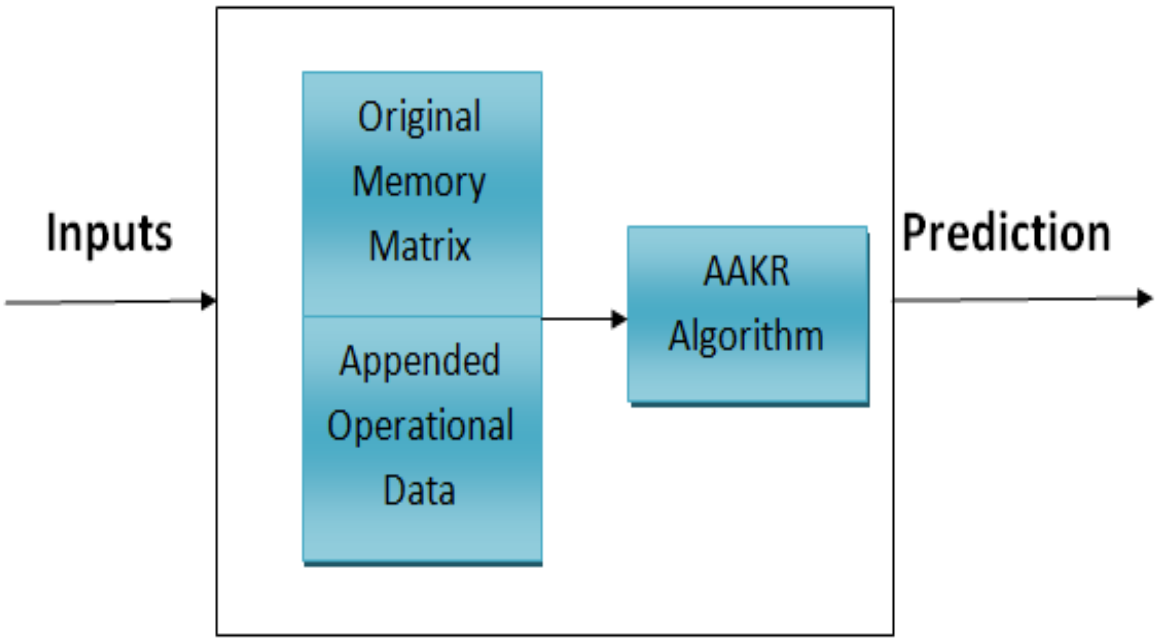


Figure 11: Proposed Adaptive Non Parametric Model hybrid structure

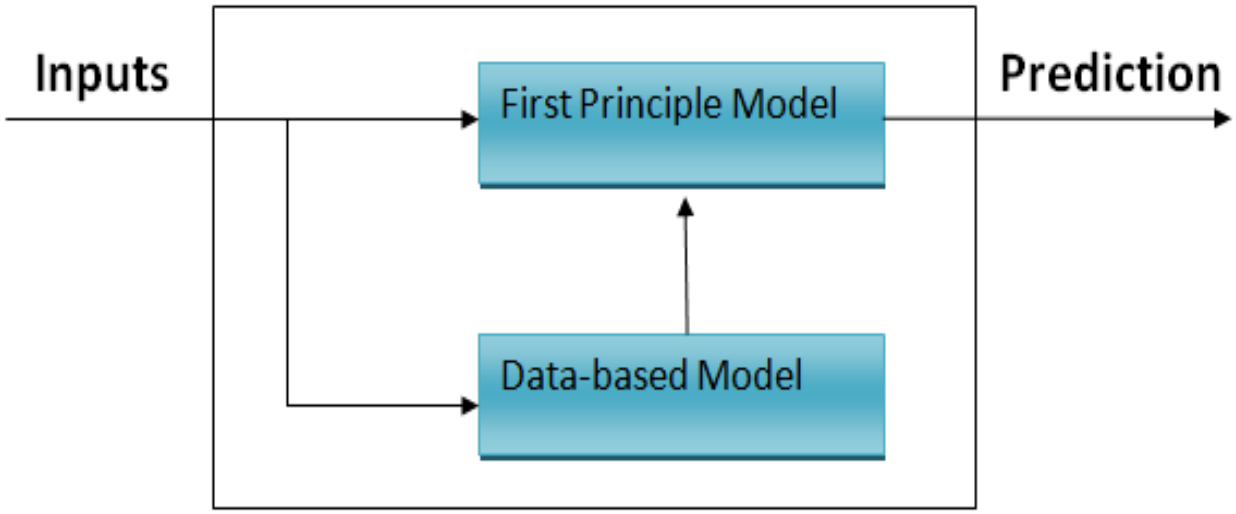


Figure 12: Serial hybrid modeling structure

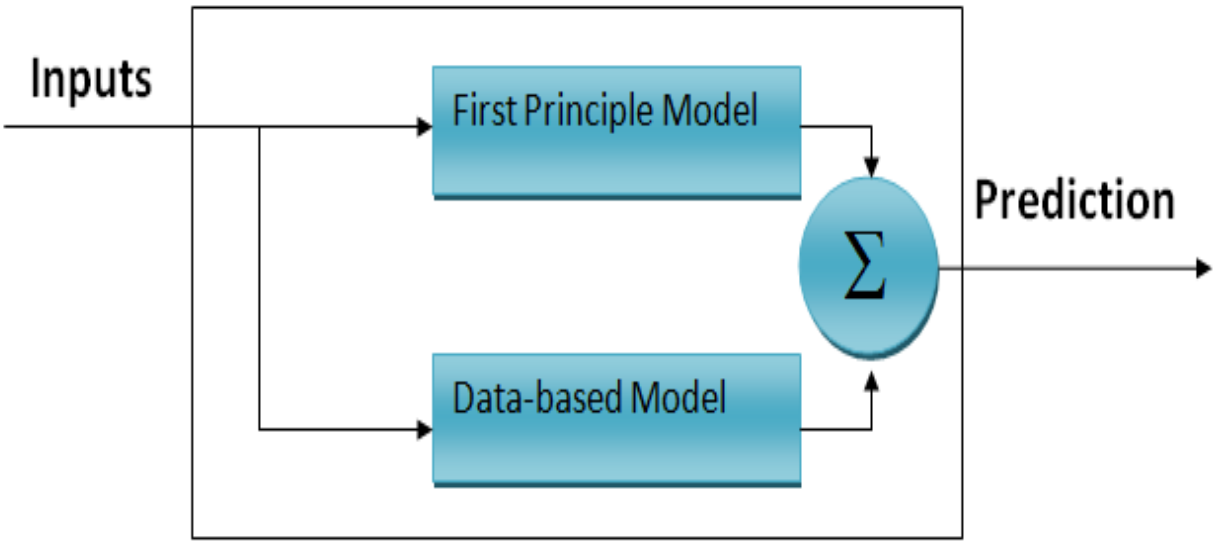


Figure 13: Parallel hybrid modeling structure

In a parallel hybrid model a data-driven model is used to predict the residuals from the FPM, these residual predictions are added to the FPM predictions to give the total prediction which is more accurate. The data-driven models are used to predict the complexities of the system that the FPM does not predict.

In this section a survey of prediction methods was given. In the next section, a review on detection methods is covered.

2.1.5 Kalman Filters

The Kalman filter uses FPM predictions and the corresponding uncertainty to produce more accurate predictions. Kalman filters are popular prediction techniques or data processing algorithms that have been used in a number of different applications such as in electronic communications, autonomous or assisted navigation, navigation of cruise missiles, the international space station, macroeconomics (Strid et al. 2009), and many more areas. The Kalman filter was first introduced by Kalman (1960) as a recursive solution to a discrete-data linear filtering problem. It is also referred to as the Stratonovich-Kalman-Bucy filter because of a general non-linear filter developed by the mathematician Stratonovich and published before Kalman's paper.

The Kalman filter uses FPMs of the system and knowledge of the system's noise to produce a more accurate prediction. The Kalman filter uses the system's dynamics model to form a predictor corrector estimator which uses a set of mathematical equations to minimize the estimated error covariance making it the optimal estimator, when certain conditions are met (Maybeck 1979). These conditions for optimality rarely exist but the filter is very useful despite

the lack of optimality. The process followed in producing the corrected estimate starts with predicting a value by using a FPM (physical laws), measurements and estimating the uncertainty of the predicted value. These values are used to compute a weighted average of the measured value with the predicted value, the lower the uncertainty the higher the weight given. This produces an estimate closer to the true values because the estimated uncertainty of the weighted average is less than either of the values that were used (Welch et al. 2001).

The basic process of the Kalman filter produces an estimate of the desired variable by combining the FPM, prior knowledge of the system, and sensors with the available measurement data. This is done by using the corrector estimator which uses a set of mathematical equations to minimize the estimated error covariance making it the optimal estimator.

2.2 DETECTION

The purpose of the empirical model is to determine when the system is deviating from normal operating conditions, to diagnose the particular fault which is affecting system operation, and to generate prognostic parameters that characterize the degradation. The model must be built using non-faulty data so that deviations from nominal conditions are reflected in the residuals. A fault detection technique is used to determine if the new data is from nominal conditions or faulted conditions. The results of the fault detection routine are used to determine if the new data should be appended onto the model. Because of the particular needs of the ANPM method, traditional fault detection methods are not generally able to differentiate between faulted conditions and expanded, but un-faulted, operating conditions.

The supervision of technical processes is in high demand due to the increased demands on reliability and safety of systems. Isermann (1984) describes the need for early fault detection techniques and the use of mathematical process models and signal models in early detection. He gives an overview of process supervision and fault detection methods. For process supervision a complete description of the monitoring process is described, from the fault detection through the fault evaluation. This description includes the definition of a fault, which is the system deviation from nominal conditions that results in the inability to fulfill the desired purpose. He discusses a number of fault detection methods, based on four quantities:

1) Measurable signals
2) Nonmeasurable state variables
3) Nonmeasurable process parameters
4) Nonmeasurable characteristic quantities

2.2.3 Absolute Value Checks

For measurable signals, a common fault detection technique is the limit and trend checking technique. A limit check for signal Y(t) is set so if the signal value exceeds the Y_{max} or Y_{min} limits a fault is detected.

$$Y_{\min} < Y(t) < Y_{\max} \quad \textbf{(2.17)}$$

This is an absolute value check where the limits are set at a distance from the normal conditions that give the desired false alarm to missed alarm probability. The limit distance is a tradeoff of the false alarm to missed alarm probabilities. The narrower the limit is set the higher false alarm probability and the lower the missed alarm probability. A large limit band gives a low false

alarm probability and a large missed alarm probability. An example of this can be seen in Fig. 14. This is one of the detriments of using such a fault detection technique.

Limit check can also be used on the trend of signal Y(t), this can give an earlier fault alarm in some cases.

$$\dot{Y}_{\min} < \dot{Y}(t) < \dot{Y}_{\max} \tag{2.18}$$

These techniques can be applied to the actual system signals or for better results can be applied to the prediction of the system signals.

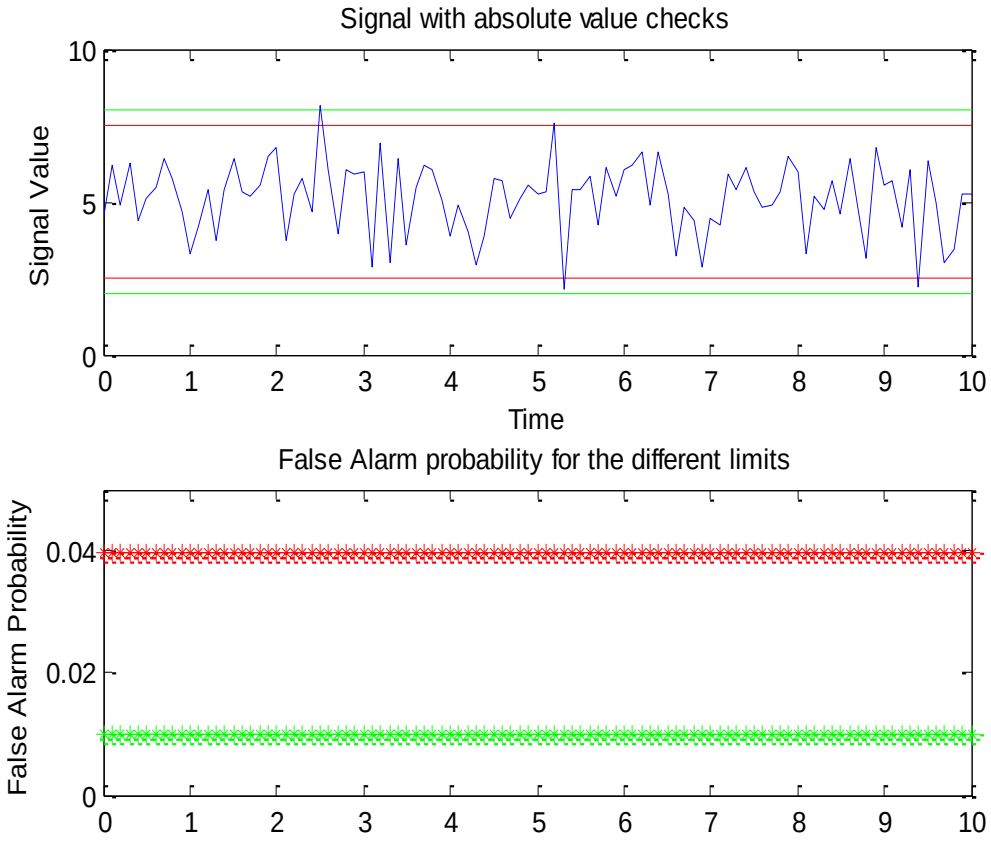


Figure 14: Absolute Value Check

$$Y_{\min} < \hat{Y}(t) < Y_{\max} \quad \textbf{(2.19)}$$

For nonmeasurable state variables a process model can be used to estimate the state variables from measurable signals. Wilsky (1976) gives a survey of statistical techniques for the detection of failures in dynamic systems and Wilsky et al. (1974) looks at the use of generalized likelihood ratios. A method of using a chi-square test and whiteness of the residuals of the normal Kalman-filter is described in Mehra et al. (1971). Many of these techniques require a precise knowledge of the system parameters and the relationships to the signals.

For nonmeasurable process parameters a process model can be used to predict the relationships within the process. A fault can be detected by looking at the changes in the coefficients and how they map to previous faults. Most of this modeling has a strong basis in the theory and physics of the system, so a strong understanding is needed.

The fault detection methods based on nonmeasurable characteristic quantities use process signals to predict the characteristic quantities of the system, such as efficiency. The effects of the faulty process on the predictions can be used to obtain fault signatures. Fault signatures are then used for fault decision or identification which corresponds to specific failures. The use of such fault detection techniques are then shown on a direct current driven centrifugal pump and a pipeline (Isermann 1984).

2.2.4 Sequential Probability Ratio Test

The use of residuals in a standard hypothesis testing procedure can be used to determine if the system is in a degraded state. The sequential probability ratio test (SPRT) is a sequential hypothesis test used as a fault detection technique that was developed by Wald (1947). This

technique was developed from the Neyman-Pearson lemma (Neyman 1933), which states: When performing a hypothesis test between two point hypotheses, the likelihood-ratio test is the most powerful test of size α for a threshold η . So when $H_0 : \theta = \theta_0$ and $H_1 : \theta = \theta_1$ then the likelihood ratio test rejects H_0 for H_1 when,

$$\Lambda(x) = \frac{L(\theta_0 \mid x)}{L(\theta_1 \mid x)} \leq \eta \quad \text{where} \quad P(\Lambda(X) \leq \eta \mid H_0) = \alpha \qquad \textbf{(2.20)}$$

The SPRT technique is a modification of the Neyman-Pearson lemma by transforming it into a sequential problem. The objective of the SPRT fault detection technique is to detect the anomaly as soon as possible while minimizing the probability of making the wrong conclusion. The SPRT can be used to detect a change in the mean or variance of the system parameters. Given the likelihood equation P with residuals s_k at time k and the mean m_i and variance σ_i for hypothesis I, the likelihood ratio is:

$$\lambda_k = \frac{P_1(s_k, m_1, \sigma_1)}{P_0(s_k, m_0, \sigma_0)} \qquad \textbf{(2.21)}$$

The log likelihood ratio becomes:

$$\lambda_k = \ln(\lambda_k) = \ln\left[\frac{P_1(s_k, m_1, \sigma_1)}{P_0(s_k, m_0, \sigma_0)}\right] = \sum_{i=1}^{k} \ln\left[\frac{P_1(s_i, m_1, \sigma_1)}{P_0(s_i, m_0, \sigma_0)}\right] \qquad \textbf{(2.22)}$$

This can be written in the recurrent form as:

$$\lambda_k = \lambda_{k-1} + \ln\left[\frac{P_1(s_k, m_1, \sigma_1)}{P_0(s_k, m_0, \sigma_0)}\right] \qquad \textbf{(2.23)}$$

The residual distributions can be assumed to be normally distributed (CLT) (Rice 1995).

Plugging this into the recurrent form gives:

$$\lambda_k = \lambda_{k-1} + \ln\left[\frac{\frac{1}{\sqrt{2\pi\sigma_1^2}} e^{\left[-\frac{(s_k - m_1)^2}{2\sigma_1^2}\right]}}{\frac{1}{\sqrt{2\pi\sigma_0^2}} e^{\left[-\frac{(s_k - m_0)^2}{2\sigma_0^2}\right]}}\right] \tag{2.24}$$

$$\lambda_k = \lambda_{k-1} + \ln\left(\frac{\sigma_0}{\sigma_1}\right) + \frac{(s_k - m_0)^2}{2\sigma_0^2} - \frac{(s_k - m_1)^2}{2\sigma_1^2} \tag{2.25}$$

For a nominal condition the mean of the residuals can be estimated at zero, this simplifies the log likelihood ratio to the following.

$$\lambda_k = \lambda_{k-1} + \frac{m_1}{\sigma^2}(s_k - \frac{m_1}{2}) \tag{2.26}$$

Using the Wald's two sided test A and B are defined as:

$$A = \ln\left(\frac{\beta}{1-\alpha}\right) \quad \text{and} \quad B = \ln\left(\frac{1-\beta}{\alpha}\right) \tag{2.27}$$

where

α =is the probability of a false alarm which should be kept small, Type I error

β = is the probability of missing an alarm, Type II error

The fault verdict is determined by a comparison of A and B with the log likelihood ratio.

$$\lambda_m < A \quad \textbf{Sensor is OK}$$

$$\lambda_m > B \quad \textbf{Sensor is DEGRADED}$$

Despite the benefits of a fault detection technique like the SPRT there are major difficulties in applying such a technique in an adaptive environment. Most traditional techniques can create a scenario where the adaptive model slowly adapts to the fault, which causes it to never detect a fault. This is a problem with drift faults in particular. While initially not detecting a small fault, the new model can adapt to a stage where the large faults are undetectable. This would cause the adaptive model to be useless. To avoid these sort of complications some new fault detection and expanded condition monitoring techniques have been developed and investigated. One technique is the principal component analysis (PCA) expanded condition monitoring (ECM) technique (Humberstone et al. 2009).

In this section a literature review on detection methods was presented. In the next section, a review on diagnostic methods is given.

2.3 DIAGNOSTICS

This section of the literature review gives a brief overview of the diagnostic methods currently available in the literature. Several diagnostic techniques are described with examples of applications given. Diagnostic is also referred to as identification and isolation; these terms will be used interchangeably throughout this paper. Diagnostics is a key portion of a complete equipement surveillance system, much of the literature focuses on the complete system not specifically the diagnostics. Mehra et al. (1971) discusses a general approach using system

theory and statistical decision theory for fault detection, and diagnosis. A two step diagnostic approach was proposed by Simani et al. (2003). The first step of this approach was the generation of residuals or symptoms of the system. Where residuals are the discrepancy between the the predictions and the actual or expected system values. A number of papers use these residuals as the indicator of incipient faults (Fenu et al. 1998). The second step is the use of a technique to relate these residuals to the specific fault conditions. This is the basic overview approach that is taken in much of the diagnostic research. There are a number of papers that give a good overview of the challenges and techniques used in diagnostics (Riedesel 1989; Gertler 1988; Frank 1990; Natke 1997; Patton et al. 2000; Milne 1987).

Kreutzer et al. (1987) used predefined tests on the individual systems of a computer network and compared the results to determine which systems were failing. Milne (1987) discusses the evolution of diagnostics from a basic rule-based system to expert systems that can use more complex reasoning techniques. A similar approach was taken by Gertler (1988), where a survey of diagnostic techniques for complex systems was given. He highlights the main components of these techniques as residual generation, signature generation, and signature analysis. Analytic redundancy is used within a number of detection and diagnostic techniques to create the needed residuals (Gertler 1988; Frank 1990). Frank (1990) discusses model based residual generation using parameter identification. Riedesel (1989) uses a rule based systems for fault diagnosis and control decisions.

These diagnostic techniques have been developed for a varaiety of applications. Many of the techniques are focused on the specific systems or situations that are being addressed and not generalized. There are a number of approaches that have been used in diagnostics. One common approach is the mutlivariate statistical method, this is a diagnostic method that uses PCA

(Russell et al. 2000; Johnson et al. 1992; Jolliffe 2002). Another common approach is knowledge based systems also known as expert systems, these are historically used in fault diagnosis (Germond et al. 1992; Venkatasubramanian et al. 2003). This technique has commonly been coupled with fuzzy logic or fuzzy inference systems (FIS) for added benefits (Tarifa et al. 1997; Ben-Abdennour et al. 1996; Del Amo et al. 2005; Kavuri et al. 1993). Tarifa et al. (1997) developed a new model based fault diagnosis approach for chemical processes. They used a signed directed graph (SDG) to determine possible outcomes and then used an expert system and fuzzy logic to evaluate the information. Ben-Abdennour et al. (1996) investigates the use of a coordinated intelligent control scheme on power plants.

Another common approach is the nearest neighbor or k-nearest neighbor (kNN) methods. Nearest neighbor classification is a decision rule used to find the nearest set of previously classified points to the observed sample point (Cover et al. 1967). The kNN classification technique was described as a three step process used to diagnose faults in transformers by Dong et al. [2004]. This technique has been used in various research, such as the geo-location of mobile phones (Wong 2001), and for sign language recognition used for robot guidance (Pook et al. 1994). Other work has paired the kNN classifier with other techniques such as neural networks (Koutroumbas et al. 1994; Murphy 1990), wavelet analysis (Creusere et al. 1994), learning vector quantization (LVQ) in supervised learning (Geva et al. 1991).

An area where diagnostics has been heavely researched and patented is the automotive industry (Breed 2002; Gayme et al. 2003; Breed 2005; Basir 2004). There is a wide degree of diagnostic techniques that have been shown to integrate such methods as PCA, fuzzy logic, kNN, expert systems, and many others into diagnostic algorithms.

In this section a survey on diagnostic techniques and applications was presented. In the next section, a review on prognostics is given.

2.4 PROGNOSTICS

Prognostics is an important part of a complete equipment surveillance system or a condition monitoring system. Prognostics is defined as the prediction of time when a component fails or no longer meets a particular function. Component failure is often defined as the point at which the component no longer performs. Prognostics is commonly tethered to prognostics and health management (PHM) or system health management (SHM). The desired prediction is commonly the remaining useful life (RUL) or probability of failure (POF) of the system or component. There are a number of prognostic models available within the literature, these approaches can often be catagorized as data-driven, model-based, and hybrid approaches.

Different types of information can be used as inputs into a prognostic model, the type of information can be used to separate the models into three types. Type I prognostics is reliability based and uses the components historical failure time, it looks at the average component under average operation. Type II prognostics uses information that contains the enviromental conditions of the components, so the stress that the system undergoes during operation is considered. Type II looks at the average component under specific operation. Finally, type III prognostics, analyzes the individual component under specific conditions. In type III, the degradation of the specific component is measured and used to predict the RUL. Fig 15 below illustrates the informational contributions to the three types of prognostic models.

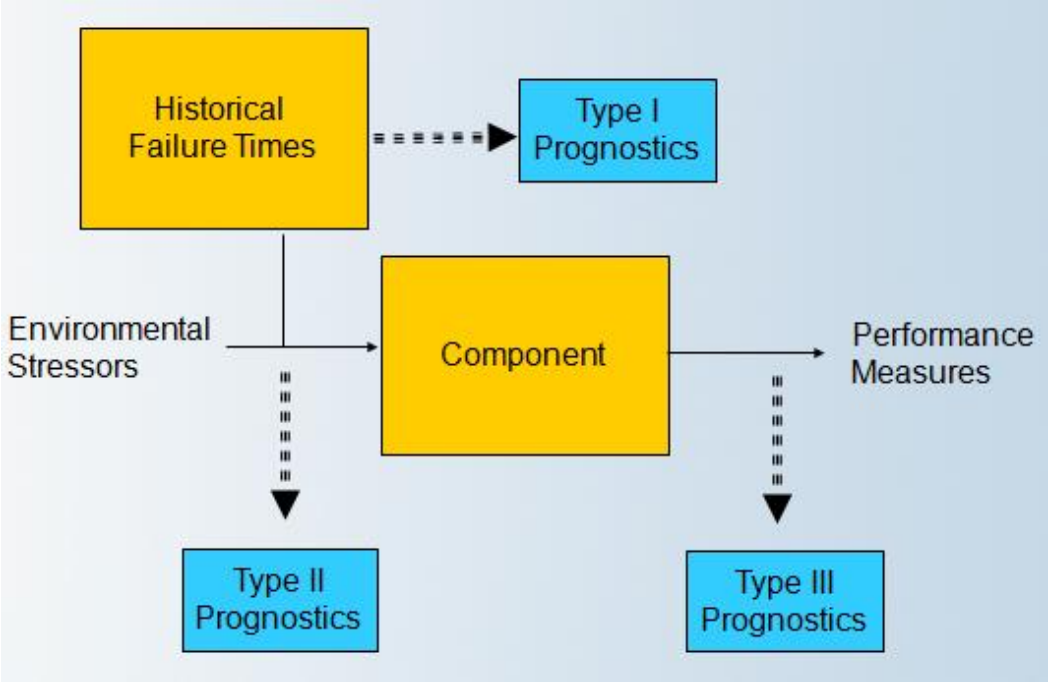


Figure 15: Prognostic types

Much of the literature focuses on the applications of a given prognostic method to a specific system and not a general approach. Prognostic research has focused on specific problems and rarely generalized, examples of these areas of focus are applications within electronics (Vichare et al. 2006; Mishra et al. 2002), applications on the joint strike fighter (JSF) (Ferrell 1999; Line et al. 2005; Hess et al. 2002; Ferrell 2000; Roemer et al. 2005). Other areas of focus have included prognostics for helicopter gearbox (Heng et al. 2009; Engel et al. 2000; Orchard et al. 2007; Wang et al. 2001b; Kacprzynski et al. 2004; Vachtsevanos et al. 1997) and vibration analysis (Carden et al. 2004; Catbs et al. 2002). There are many examples of specific prognostic models created for specific systems; however, the following sections will focus on the three types of prognostics and some common methods.

2.4.3 Type I Prognostics

Type I prognostics is commonly referred to as time to failure analysis and is closely related to tradiational reliability methods. Traditional reliability methods for cumulative time to failure (TTF) distributions is where many of the prognostic methods have evolved from (Pecht et al. 1995a). These technique estimate the cumulative TTF distribution by observing a population of devices or using accelerated life testing (Meeker et al. 1998; Ebeling 2005). These algorithms estimate the RUL based on the average system, operating under average conditions. Since these algorithms do not use any information from the specific system they are referred to as population based prognostics. A common metric used in Type I prognostics is the mean time to failure (MTTF) which is defined as the expected failure time of a device from a given population. Equation 2.28 gives the MTTF for a continuous TTF distribution with $f(t)$ being the probability density function (PDF).

$$\text{MTTF} = \int_0^\infty t\, f(t)\, dt \tag{2.28}$$

Another popular metric is the failure rate or hazard function $\lambda(t)$. The failure rate function is defined in equation 2.29.

$$\lambda(t) = \frac{f(t)}{R(t)} \tag{2.29}$$

A common model used in reliability analysis is the Weibull analysis. This is a parametric model that has the flexibility needed to be benefial in a number of situations. Two parameters, the shape

parameter β and the characteristic life θ , are used to describe the failure rate $\lambda(t)$ (Eqn. 2.30).

$$\lambda(t) = \frac{\beta}{\theta}\left(\frac{t}{\theta}\right)^{\beta-1} \tag{2.30}$$

The Weibull distribution can be used to model the exponential, normal, or Rayleigh distributions, dependent on the shape parameter. For a more information on Weibull modeling the reader is referred to Abernethy (1996). It has been shown that Type I or population based prognostics do not provide accurate results (Pecht et al. 2002; Lall et al. 1997). This is no surprise since no specific system information is used within the predictions. This leads into the next section which is Type II prognostics.

2.4.4 Type II Prognostics

Type II prognostics is a funtion of the stress a system undergoes.This type of prognostics uses information that contains the enviromental conditions of the components, so the stress that the system undergoes during operation is considered. Type II looks at the average component under specific operation. By using the conditions that the system is and was subjected to a better prediction of RUL can be obtained (Vichare 2004; Azzam 1997). There are a number of methods used in Type II prognostics, such as Proportional Hazards Models (Liao et al. 2006; Dale 1985; Kumar et al. 1994), Markov Chain Models (Bogdanoff et al. 1985; Kharoufeh et al. 2005), Life Consumption Models (Ramakrishnan et al. 2003; Mishra 2004), physics-of failure models (Kacprzynski et al. 2004; Pecht et al. 1995b; Kacprzynski et al. 2002), and more.

Proportional Hazards (PH) Model uses both failure time data and stress data to estimate the RUL of the system (Cox et al. 1984). This is done by modifying the hazard rate to correpond to the components specific enviromental stresses. The PH model transforms the hazard function from a function of time to a function of time and other varaibles that indicate the stress of the system (Nelson 1990). PH models have been used in a number of applications including the estimation of RUL (Liao et al. 2006), product reliability (Dale et al.1985) , etc. For a thorough approach to developing a PH model the interested reader is referred to Kumar et al. (1994).

Markov Chain Models are used to simulate the transition from one state to the next for the component of interest. This is based on the idea that future state of the component is dependent only on the current state. This is done by creating a transition probability matrix that defines the transition probabilities between the components states. As the future state predictions are simulated they are mapped to a degradation of the component. A threshold is given that defines failure; therefore, the simulated degradation paths are used to estimate the distribution of the time to failure for the component. The interested reader is referred to Bogdanoff et al. (1985) who describes uses and development of Markov Chain Models.

Life Consumption Models (LCM) are used as a way to predict the fraction of system life lost due to the conditions experienced. Ramakrishnan et al. (2003) first developed this technique to predict the RUL of electronic systems. These models rely on the accuracy of the physics of failure models used to estimate the damage of the system.

Another Type II prognostic approach is the physics of failure models; these models use first principle models to determine the degradation of the specific component. These models can be time intensive and expensive to build, and underlying assumptions and component unknowns

can cause inaccuracies within the predictions. However, physics of failure models can offer a better understanding of the system operation, and root cause of failure. These types of models have been used in a number of applications, such as electronic prognostics (Valentin et al. 2003), helicopter gearboxes (Kacprzynski et al. 2004).

Several Type II prognostic techniques were breifly discussed in this section. The following section will discuss Type III prognostics and give some example techniques.

2.4.5 Type III Prognostics

Type III prognostics is the prediction of RUL for an individual component under specific conditions. In type III, the degradation of the specific component is measured and used to predict the RUL. The main difference between Type III and Type II prognostics is Type III analyzes the specific component, where Type II looks at the average component under specific enviromental effects. There are several Type III techniques that have been proposed, such as the General Path Model (GPM) (Lu and Meeker 1993; Upadhyaya et al. 1994), Markov Chain Models (Hines et al. 2008), and Shock Models (Esary et al. 1973; Gut 1990; Mallor et al. 2003).

One of the major methods used in Type III prognostics is the General Path Model (GPM) or degradation modeling. Developed by Lu and Meeker (1993) to estimate the TTF distribution for crack growth. Later, this technique has been evolved into the tracking of a degradation parameter and extrapolating this path to failure. The degradation path is assumed to increase or decrease monotonically towards the failure threshold. An underlying functional form is predefined to to fit the degradation path, this form can be from historical data, physics of failure models, or from expert opinions. These parameters are found by fitting the model to the degradation paths, a

number of techniques have been developed for parameter estimation and functional form estimation (Chinnam 1999; Kutner et al. 2004). A critical level of degradation or threshold is defined as the degradation level where failure occurs. The GPM is the most popular Type III prognostic technique and has been applied in a multitude of areas such as electronic for GPS applications (Brown et al. 2007), nuclear applications (Heo 2008), flight control actuators (Byington et al. 2004), etc.

Another Type III prognostic technique is the Markov Chain Model, this method was described earlier as a Type II prognostic method but can also be applied as a Type III method. The difference between these two Markov Chain Model applications is for Type III prognostics the systems degradation levels are used to generate the random shock arrival times. Where Type II Markov Chain Model generates possible future paths for the system based on the state of the system. By incorporating the degradation of the actual system the Markov Chain Model becomes a Type III prognostic method.

There are a number of prognostic technique that are available within the literature, a brief overview of Type I, Type II, and Type III prognostics was given with some examples.

The literature review gave an in depth analysis of past research that is related to the research presented within this dissertation. The literature review had four major sections, prediction, detection, diagnostics, and prognostics. These sections covered the traditional techniques with an emphasis on highlighting the similarities and differences between these techniques and the techniques presented in this research. The next chapter is methodology which describes in detail the approaches taken to accomplish the research tasks while accomplishing the original contributions. The methodology focuses on the design of the ANPM, PCA ECM,

and the diagnostic technique, with a discussion on prognostics and the integration of these techniques into a complete surveillance system.

3 METHODOLOGY

3.1 ADAPTIVE NONPARAMETRIC MODEL

Traditional non-parametric models can only make reliable predictions within the range of the training data. The need for an adaptive modeling technique arises when a monitored system's operation shifts from the original training data relationships. Such a change in data relationships can occur for several reasons, including seasonal changes, load changes, and unforeseen operating profiles. Training data sets may be inadequate because the future operating conditions cannot be accurately predicted, development of an inclusive training data set may be complex and uncertain, or no actual operating data is available for new systems and processes. In any of these cases, FPM data can be used in a temporary memory matrix to account for these shortcomings for prediction. However, the actual data will have superior predictive performance to the FPM data. The FPM may include assumptions that don't hold for the process of interest, as well as some modeling error or uncertainty that was not considered in the FPM development. A model with the highest predictive accuracy is desired. As actual operating data is collected, the model should incorporate it in predictions, eventually replacing the FPM data all together. One concern in this process is ensuring data is characteristic of non-fault condition to maintain the fault detection ability of the monitoring system.

The base model used in the ANPM is the auto-associative kernel regression (AAKR), a non-linear, non-parametric modeling technique. The ease of data addition to the original model is one of the major benefits of this type of model. When appropriate, a new observation can be added to

the model by simply appending it to the model's memory matrix. As nominal operation data becomes available, the first principle data can slowly be removed from the model.

The ANPM was developed for the primary use on the International Reactor Innovative and Secure (IRIS) reactor used in the Global Nuclear Energy Partnership (GNEP) program. The IRIS reactor was chosen as the Integral Primary System Reactor (IPSR) for this research (Storrick et al. 2005). As will be shown through the examples, the ANPM can be applied to a variety of process monitoring systems.

This dissertation presents the development of the first versions of the ANPM and shows its application to a simple heat exchanger model. Different fault detection techniques have been applied to the ANPM; however, due to the dynamic nature of the model, traditional fault detection is difficult to implement. The traditional methods such as Sequential Probability Ratio Test (SPRT) do not accurately differentiate faults from expanded operating conditions. A number of fault detection techniques were investigated for implementation with the ANPM.

3.1.3 Basic Method

3.1.3.1 Base Model

The base empirical model uses an AAKR (Hines et al 2007a), model. AAKR is a non-parametric model that uses past normal operational data to correct faulty observations which may be due to system degradation, sensor faults, data acquisition problems, etc. The outputs of an AAKR model are the predicted or corrected values of the inputs.

The AAKR model uses a Euclidean distance, which is known as the L^2-norm, to compare the input query data to the exemplar vectors which make up the model's memory matrix. This is for n inputs.

$$u_j = \sqrt{\sum_{i=1}^{n} \left(x_q^i - m_j^i \right)^2} \tag{3.1}$$

The Euclidean distance is then used in a Gaussian kernel similarity function to define weights for the population exemplars. A kernel function should have a large weight for small distances and small weight for large distances. There are advantages and disadvantages specific kernels for each situation; it has been shown that the kernel function does not have a large impact on the performance of the locally weighted models (Scott 1992; Cleveland et al. 1994a; Cleveland et al. 1994b). A common kernel is the Gaussian kernel (Fan et al. 1996); this kernel is a function of the Euclidean distance and the kernel bandwidth *h* and is used in the ANPM.

$$w = K(u,h) = \frac{1}{\sqrt{2\pi \cdot h^2}} e^{-u^2/h^2} \tag{3.2}$$

In general the bandwidth is optimized by minimizing the error for a test data set. This is done initially, before applying the model for system monitoring, to find the optimal bandwidth; this bandwidth is used throughout the adaptation period or phase 1. The final prediction is a weighted sum of the exemplar vectors, m_i.

$$\hat{x}_q = \frac{\sum_{i=1}^{n_m} w_i \cdot m_i}{\sum_{i=1}^{n_m} w_i} \tag{3.3}$$

3.1.3.2 Overview of the Approach

A three phase approach is proposed for the application of the ANPM. A diagram of this approach is shown in Fig. 16. This approach starts with phase 1, which uses data from the developed first principle model. This first principle model is based on the underlying physics of the system of interest. During phase 1, the ANPM is modified using the actual data collected from the system. The time frame for this phase is subjective, and should be determined on a case by case basis. The time frame needs to allow for data collection sufficient to cover the operating conditions of the system. This is necessary to build a model that will make accurate predictions for the entire span of the operating conditions.

The second phase of the three phase approach is the validation phase. The updated model is used for monitoring and fault detection without adapting during this phase. In addition, a second model, which is built in the traditional way from the nominal operation data collected in phase 1, is also used during this phase. This model will be referred to as the non-adaptive model. The new data collected from the system during this phase is used to validate the models, and their performance is compared to determine which model is better suited for monitoring and fault detection. It is believed that the non adaptive model should be able to perform at least as well as the ANPM developed model; however, further research is needed to validate this claim. This assumption is made because the ANPM may include only a subset of the data used to build the non-adaptive model. During the adaptive phase, the ANPM decides which new observations are important for improving model performance. The ANPM predictive performance depends on how well these decisions are made.

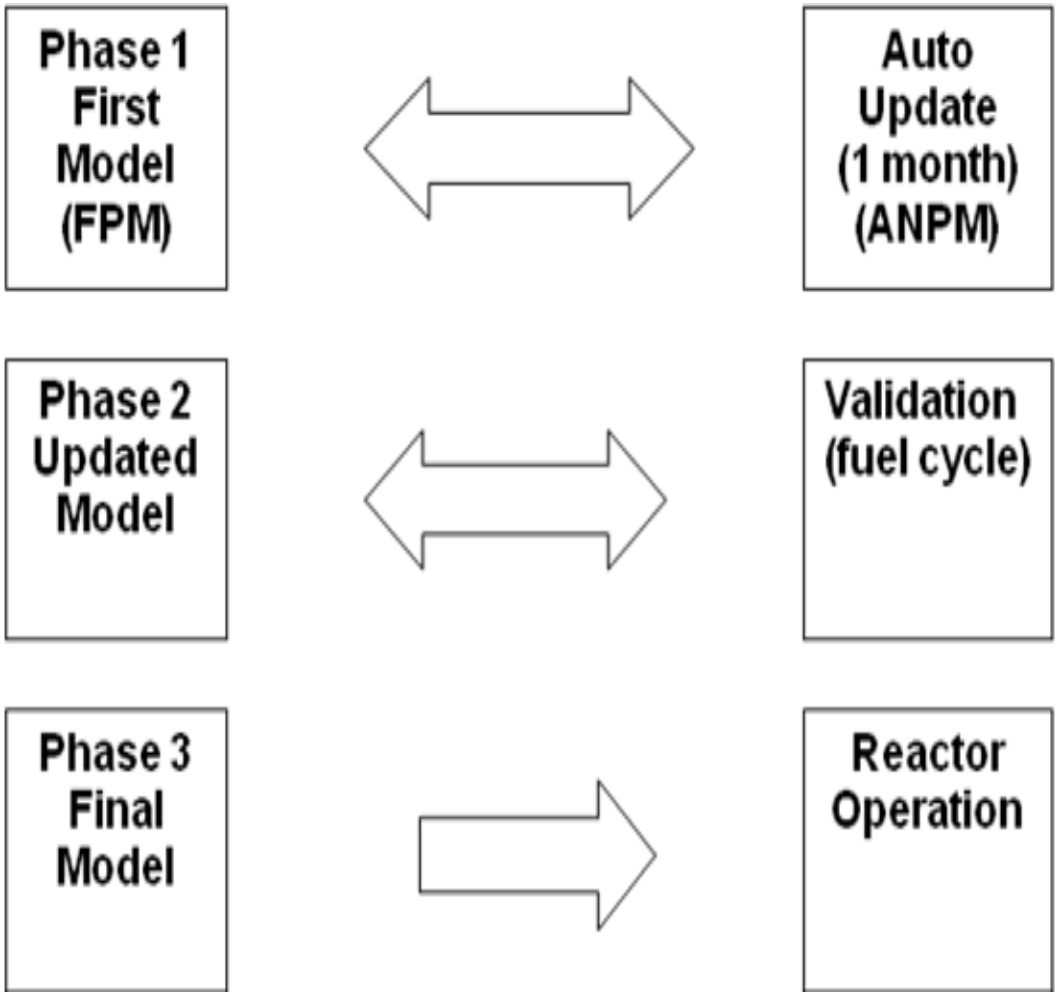


Figure 16: ANPM Three phase approach

In the third, and final, phase the final model is chosen from the ANPM and the non-adaptive model and applied to the system. Some model maintenance may be needed over time, but the adaptive phase of model development which is the changing of the data basis is complete. The expanded condition monitoring applications can be applied during the adaptive phase and during the normal operation of the model.

3.1.4 Phase 1 Overview

The adaptive portion of ANPM development is completed in phase 1 of the system operation. Fig. 17 gives a basic overview of phase 1 compared to a traditional hybrid model. Fig. 17 shows how the first principle model is used to generate data to populate the original memory matrix in the ANPM. The simulated data is used to make AAKR predictions during initial system operation.

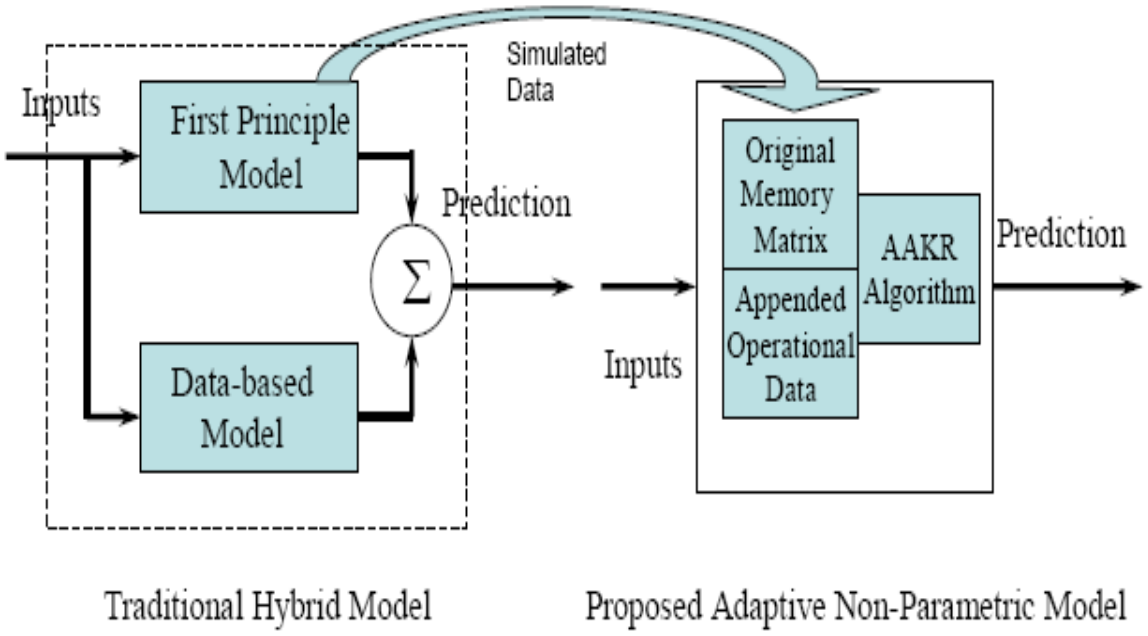
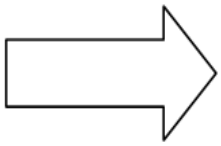


Figure 17: Phase 1 Overview Diagram

These predictions are used to find the residual between the actual data the expected system operation. Analysis is done to determine if the actual system data represents faulted or non-faulted conditions. Then, if the data vector is determined to be from non-faulted data, this new observation can be appended onto the original memory matrix of simulated data. The decision to add the data to the model is based on the fault detection results for the current observation, previous observations, and future observations. In this way, a new model evolves which is based on a combination of the first principle model data and the data collected during system operation.

As actual system data is appended to the ANPM memory matrix, the data simulated by the first principle model decays from the model. At the end of phase 1, a model is desired that is based entirely on actual data from the system. This is done by deleting the observations contained in the original memory matrix as actual system observations which characterize the same operating conditions are collected. The method used for vector deletion from the model is a constant rate vector deletion technique. This consists of deleting vectors from the original memory matrix in an evenly spaced time sequence. This is done over the operating period of the ANPM or during phase 1. The vector with the greatest contributions to past predictions is deleted as is shown in Fig. 18. This eliminates the vectors that are the most similar to the new vectors that are being appended. This is an important feature of adaptation because it directs the evolution of the model so that the models range is complete. This process is continued, until only the system's data is in the model's memory matrix.

Memory Matrix	Contribution
V1	0
V2	0
V3	0.2
V4	0.6
V5	0.2
V6	0
V7	0



New Memory Matrix
V1
V2
V3
New Vector
V5
V6
V7

Figure 18: Memory matrix decay

3.2 FAULT DETECTION TECHNIQUES

Empirical modeling techniques are used to assess the system performance and can be used for condition based maintenance. Autoassociative architectures produce process variable predictions which are used with the process variable measurements to create residuals. These residuals are used within a fault detection algorithm such as the sequential probability ratio test (SPRT) to evaluate the state of the system. This is traditionally a fault detection technique and does not have the ability to detect expanded conditions. The complete equipment surveillance system consists of the monitoring, diagnostic and prognostic capabilities. The ANPM increases the accuracy of the monitoring portion which affects the whole equipment surveillance system by increasing the performance of the diagnostics and prognostics. The fault detection ability

governs the accuracy of the ANPM and is important for the overall equipment surveillance system.

The main difference between an adaptive model and a non-adaptive model, beyond the capability to adapt, is the ability to differentiate between an expanded condition and a fault. Non-adaptive models classify expanded conditions as faults. Adaptive models, however, need the capability to distinguish the two conditions (Fig. 19).

Adaptive models are dynamic models that rely heavily on the internal fault detection techniques. The fault detection within the model determines the evolution of the model. The correct classification of expanded conditions gives the adaptive model the ability to adjust beyond the initial range of the model. This ability is a desirable attribute that increases the usefulness of the adaptive model.

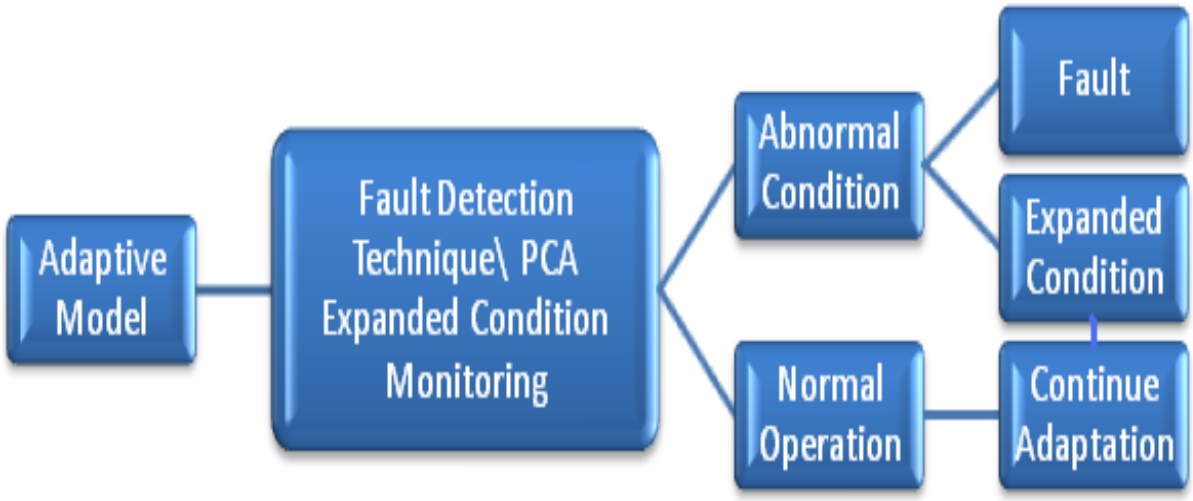


Figure 19: Adaptive Modeling Fault Procedure

An adaptive model that does not have this ability could be developed without using an expanded condition monitoring technique, but this would affect the potential evolution of the model and limits its true adaptability.

The need for an expanded condition monitoring approach is realized in adaptive modeling environments. Traditional fault detection techniques do not offer the capability to differentiate between faulted conditions and expanded conditions. Both the sequential probability ratio test (SPRT) (Wald 1945) and the error uncertainty limit monitoring (EULM) (Garvey et al. 2006) techniques do not have the ability to differentiate between expanded and fault conditions. The proposed PCA expanded condition monitoring (ECM) technique, however, is shown to have this ability.

3.2.3 Principal Component Analysis Expanded Condition Monitoring

Expanded process monitoring is one of the major challenges within adaptive modeling. Adaptive models have the challenge to balance stability and elasticity. The model should be able to adapt to new operating conditions without losing the capability to differentiate faults from nominal conditions (Humberstone et al. 2009). Where non-adaptive models may declare any extension beyond the range of the model as a fault, an adaptive model needs the capability to determine if it is a fault or an extended nominal condition. A similar approach to the principal component analysis (PCA) monitoring technique can be applied to the adaptive modeling procedure, which distinguishes the condition of the system.

Two commonly used statistical indices are the Hotteling's T^2-statistic and the squared prediction error (SPE), or the Q-statistic. These statistics can be used to indicate where a data point is with respect to the population. The T^2-statistic describes the variation within the model, and the Q-statistic describes the variation outside the model.

$$T^2 = T\Lambda_k^{-1}T^T \leq \delta_T^2 \quad \textbf{(3.4)}$$

$$Q = ee^T = X(I - P_e P_e^T)X^T \leq \delta_Q^2 \quad \textbf{(3.5)}$$

This results in one of four possible outcomes which are shown in Table 1 (Wang et al. 2002), where within limits is designated as WL and outside the limits is designated as OL.

The need for an expanded condition monitoring approach was shown for an adaptive modeling environment. Traditional fault detection techniques do not offer the capability to differentiate between faulted conditions and expanded conditions. However, such capability is crucial for truly adaptive modeling. In the proposed PCA ECM technique, the Q-statistic is used as the fault indicator, and the T^2-statistic is used as the expanded condition indicator. This is the process that is followed in this research.

Table 1: Possible Scenarios

Scenarios	Q-statistic	T^2-statistic	Conclusion
1	WL	WL	Normal Condition
2	WL	OL	Expanded Condition
3	OL	WL	Fault
4	OL	OL	Fault

It is shown that this ECM technique can correctly differentiate between expanded and fault conditions.

3.2.4 Kernel Principal Component Analysis Expanded Condition Monitoring

Another approach to the fault detection and expanded condition monitoring challenge was investigated. Similar to the PCA ECM technique, the uses of kernel principal component analysis (KPCA) is used instead of the previously discussed PCA. The purpose of investigating the use of KPCA is to use a nonlinear approach that would give the ability to map the nonlinear relationships within the data. KPCA is a nonlinear PCA technique that maps the input space into a feature space using a nonlinear kernel function (Scholkopf et al. 1998; Mika et al. 1999). The feature space is defined as a higher dimensional mapping, which transforms the nonlinear relationships into linear relationships. Then a traditional PCA technique is used to calulate the PCs of the higher dimension feature space. Effectively by performing linear PCA on the feature space it is equivalent to performing a nonlinear PCA on the initial input space (Romdhani et al. 1999). KPCA is a simple nonlinear PCA approach that has the ability to use a number of different kernels, making it capable of handing a number of nonlinear relationships (Lee et al. 2004). This is beneficial and detrimental, because there are an infinite number of kernels that can be used, and determining the appropriate kernel can be difficult (Christianini et al. 2000). The kernel function choice determines the final feature space and contributes to the performance of using KPCA. There are an infinite number of kernel functions available, some popular kernels are the polynomial (homogenous) kernel (Equation 3.6), polynomial (inhomogenous) kernel

(Equation 3.7), radial basis kernel (Equation 3.8), Guassian radial basis function (Equation 3.9) ; where d and c are user defined.

$$k(x_i, x_j) = \left(x_i \bullet x_j\right)^d \tag{3.6}$$

$$k(x_i, x_j) = \left(x_i \bullet x_j + 1\right)^d \tag{3.7}$$

$$k(x_i, x_j) = \exp\left(-\frac{\left(x_i - x_j\right)^2}{c}\right) \tag{3.8}$$

$$k(x_i, x_j) = \exp\left(-\frac{\left(x_i - x_j\right)^2}{2\sigma^2}\right) \tag{3.9}$$

The steps taken to perform the KPCA ECM is as follows:

Initial Steps:

1) Obtain normal operational data and mean center.
2) Apply the Kernel function to transform the data from the input space to the feature space.
3) Mean center the feature space data.
4) Apply PCA to this data and follow the discussed PCA ECM approach

ECM Technique Steps:

1) The data is first mean centered using the mean from step 1 in the initial steps.

2) The Kernel function is applied to transform the data from the input space to the feature space.

3) The feature space data is then mean centered using the means from step 3 in the initial steps.

4) PCA is applied to the feature space.

5) The same procedure as discussed for PCA ECM is then followed.

The method behind the KPCA ECM technique was discussed as a way to extend the PCA ECM technique to variables with nonlinear relationships.

The next section discusses the methods followed in the elimination and similarity diagnostic technique.

3.3 ELIMINATION AND SIMILARITY DIAGNOSTICS

A complete equipment surveillance system consists of a monitoring, fault detection, diagnostic, and prognostic capabilities. A number of these techniques have been studied in depth and applied to varying systems of interest. A fault diagnostic module has been created that can be implemented into a complete equipment surveillance system, which will be capable of determining the most likely fault given the faulty residuals. A two step process which consists of fault signature elimination and a weighted path similarity are used to give the best identification. This technique is referred to as the elimination and similarity (ES) diagnostic technique. A

detailed description of both steps is provided. Starting with faulty residuals and ending with a most likely fault.

A fault detection and identification (FDI) technique has been developed for application with the adaptive nonparametric model (ANPM). Diagnosis is also referred to as identification and isolation; these terms will be used interchangeably throughout this paper. The approach used in this FDI system can be applied to numerous equipment surveillance systems, an adaptive modeling technique is not required. An overview of the basic implementation of this elimination and similarity (ES) diagnostic technique into a complete equipment surveillance system is discussed.

One of the main requirements of the monitoring and fault detection techniques are, the prediction ability of the model which results in residuals, and the fault detection ability that produces a set of faulty residuals corresponding to known sensors, respectively. The fault detection portion of an equipment surveillance system can be a number of different techniques such as the sequential probability ratio test (SPRT), error uncertainty limit monitoring (EULM), principal component analysis (PCA) expanded condition monitoring (PCA), or many other techniques, and still be compatible with the ES diagnostic module presented. Depending on the system of interest's monitoring needs, different detection methods have different strengths. The SPRT is a sequential hypothesis testing techniques that has good detection results for a number of systems; however, both the sequential probability ratio test (SPRT) (Wald 1945) and the error uncertainty limit monitoring (EULM) (Garvey et al. 2006) techniques do not have the ability to differentiate between expanded and fault conditions. The PCA ECM technique (Humberstone et

al. 2009), however, is shown to have this ability to detect faults and differentiate expanded conditions in an adaptive environment.

The residuals are the inputs into the ES diagnostic module. These faulty residuals are classified as either a positive drift or bias which is given the signature designation +1, a negative drift or bias which has a signature designation -1, or a noise increase which has a signature designation of 3. The non-fault residuals have a fault signature designation of 0. This process of classification of fault signatures can be seen in Fig. 20. The end result is a complete fault signature vector.

The complete fault signature vector is compared to a failure bank of fault signature vectors that correspond to all possible failures. After the possible failures are determined there may be more than one possible fault corresponding to the given fault signature vector. This elimination of possible faults to only include failures with similar fault signature vectors is the first step in the ES diagnostics. A historical residual memory matrix of failures is used to

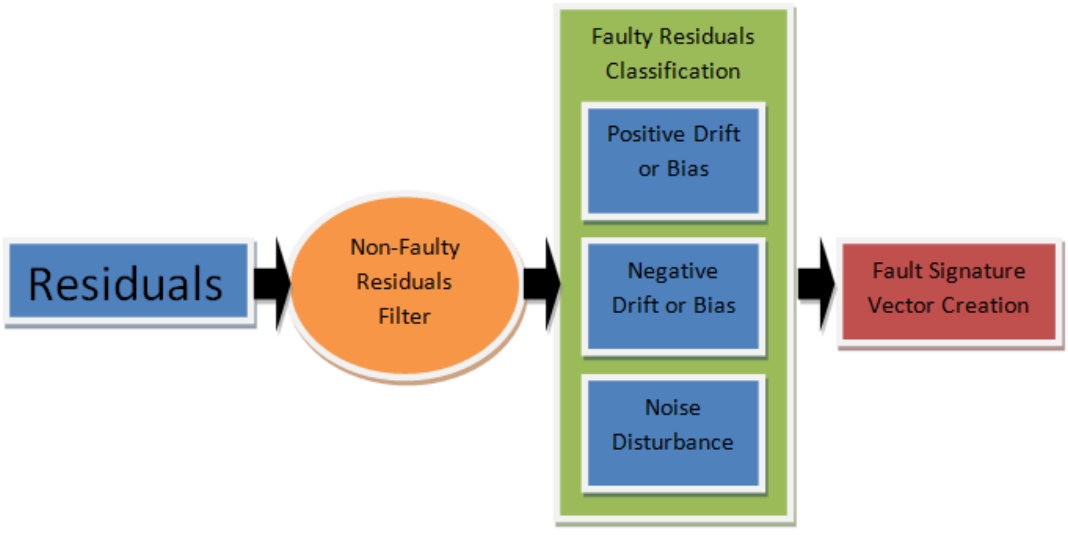


Figure 20: Classification of fault signatures

designate which fault is most similar to the current failure. The similarity method uses the Euclidean distance from the past failure residuals and the current faulted residuals.

$$S = \sum_{i=1}^{n} \left[\sum_{j=1}^{m} \left(PFR_{ij} - CFR_{ij} \right)^2 \right] \quad \textbf{(3.6)}$$

Where n is the number of faulty residuals and m is the number of observations from the time a fault was initially detected. First the historical residual memory matrix of failures is trimmed to include only the faults that match the given fault signature vector. And then this matrix is trimmed to include only the residuals that have failures. Fig. 21 shows the process of comparing the current failure to the past failures that have the same fault signature vector. This example shows that system failures 1, 4, and 6 have the same fault signature vectors. The Euclidean distances between the actual failure residuals and the historic residuals of the designated failures measure the similarity of the failures.

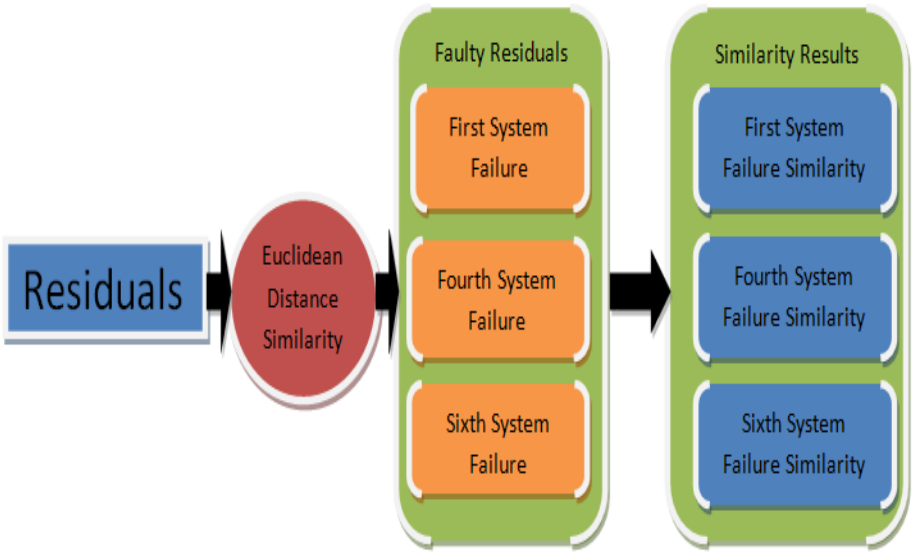


Figure 21: Failure Similarity Process

3.4 ADAPTIVE NONPARAMETRIC MODEL FLOW DIAGRAM

The ANPM was developed in MATLAB using the Process and Equipment Monitoring (PEM) toolbox developed by Dr. Dustin Garvey (Hines et al. 2005). Fig. 22 gives the ANPM flow diagram. This diagram continually changes as the ANPM is being updated. This diagram continually changes as the ANPM is being updated.

The ANPM starts with the original first principle model data simulated for the system of interest. The query data collected during system operation is compared to observations in a data matrix which stores the actual data for the non-adaptive model created at the end of phase 1 and also used for reference throughout the model adaptation for previous vector relationships (Hines et al. 2006). The query is input to the model to obtain a prediction and this prediction is used to find the residual which is used to determine if the query has a faulted sensor or if the data is from non-faulted data. Several fault detection techniques that have been tested, and are continually being improved.

Methodology described in detail the approaches taken to accomplish the research tasks while accomplishing the original contributions. The methodology focused on the design of the ANPM, PCA ECM, and the diagnostic technique, with a discussion on prognostics and the integration of these techniques into a complete surveillance system.

The next chapter, results, shows the research that has taken place and the results of the application of the methodologies chapter. Results are shown to highlight the use of these methods on actual data sets and to prove the proposed concepts.

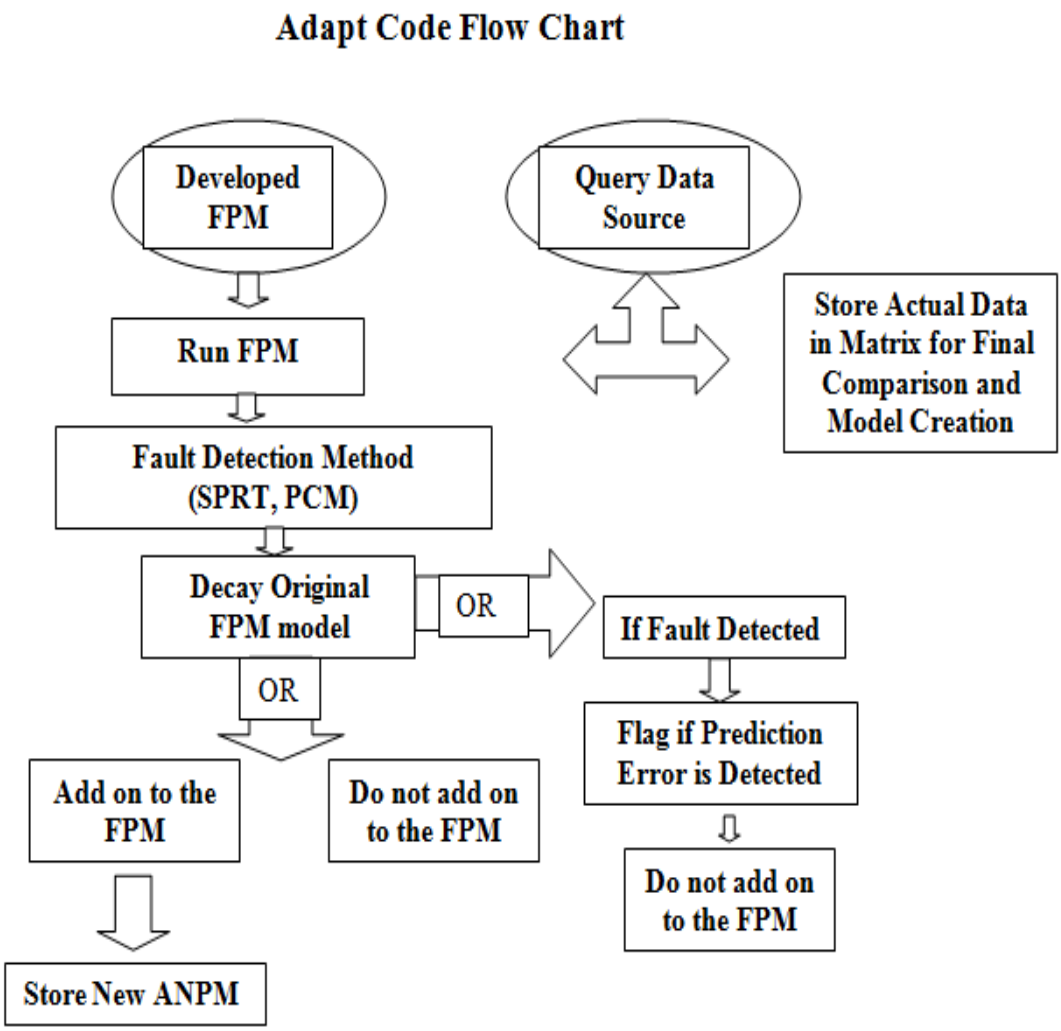


Figure 22: ANPM flow diagram

4 RESULTS

4.1 PRINCIPAL COMPONENT ANALYSIS EXPANDED CONDITION MONITORING RESULTS

A process to test the PCA ECM technique was designed to show the potential use of such a technique in an adaptive environment. This process consists of testing this technique on faulty and expanded condition data. This was done using two different test data sets. The results of these tests are shown in the following examples.

4.1.3 Principal Component Analysis Expanded Condition Monitoring test on generated data

A data set was generated to test the proposed PCA ECM technique. A highly correlated multivariate data set with noise was desired. This was generated with normally distributed noise and five variables which represents five sensors from a process of interest (Fig. 23). The correlation matrix for this data can be seen in Fig. 24. This shows that the data is highly correlated and will perform well in a non-parametric auto-associative model.

Faults were added to the generated data to simulate possible fault scenarios. Faults were only added to sensor 2. These consisted of six drift faults, six bias faults, and a stuck sensor fault, for a total of 13 faults. The drift faults consisted of positive and negative drifts of 5%, 10%, and 20%. The bias faults consisted of positive and negative bias of 5%, 10%, and 20%. The stuck sensor fault is constant from a given time forward. All of the faults were initiated at one third of the data simulation period, just after the 300^{th} observation. Fig. 25 shows all of the faults and their relations to the original data.

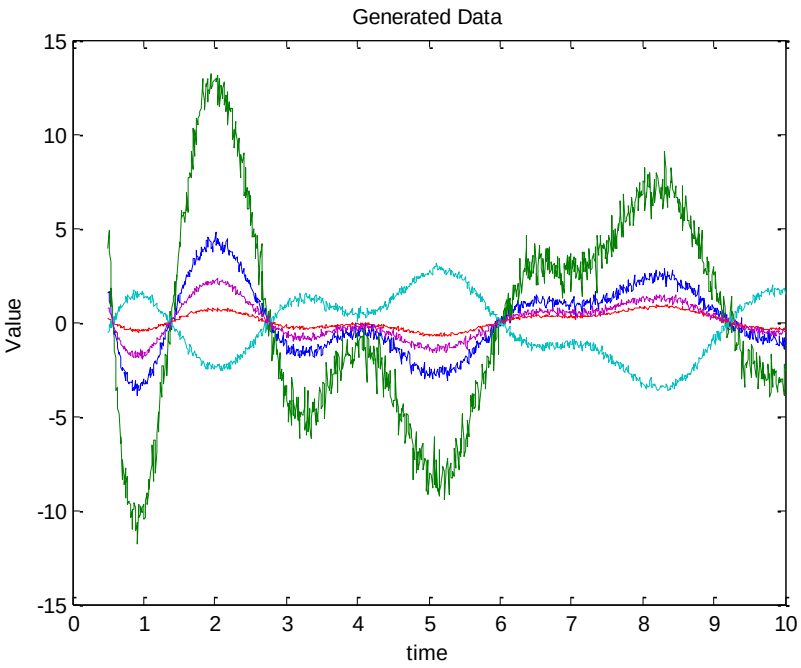


Figure 23: Generated Data

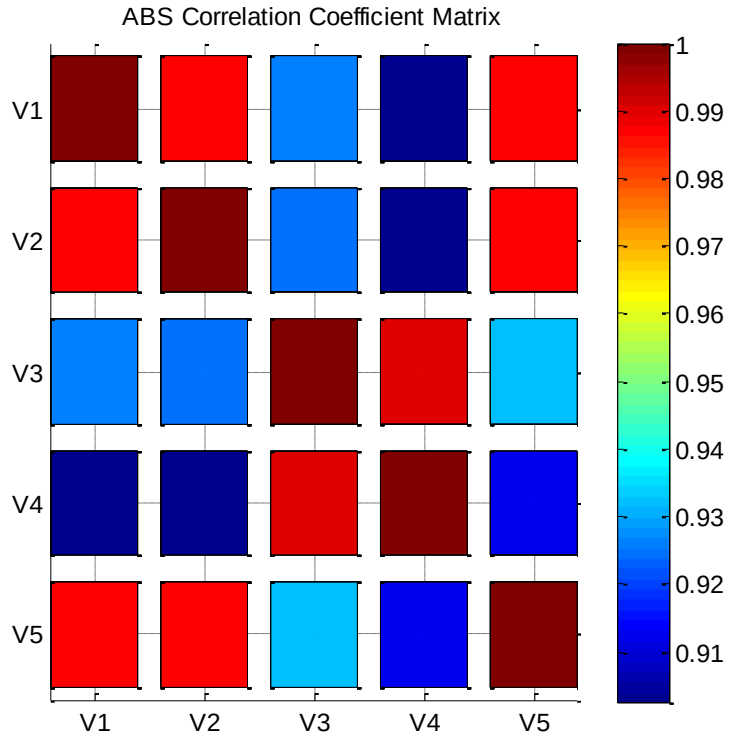


Figure 24:Correlation Matrix

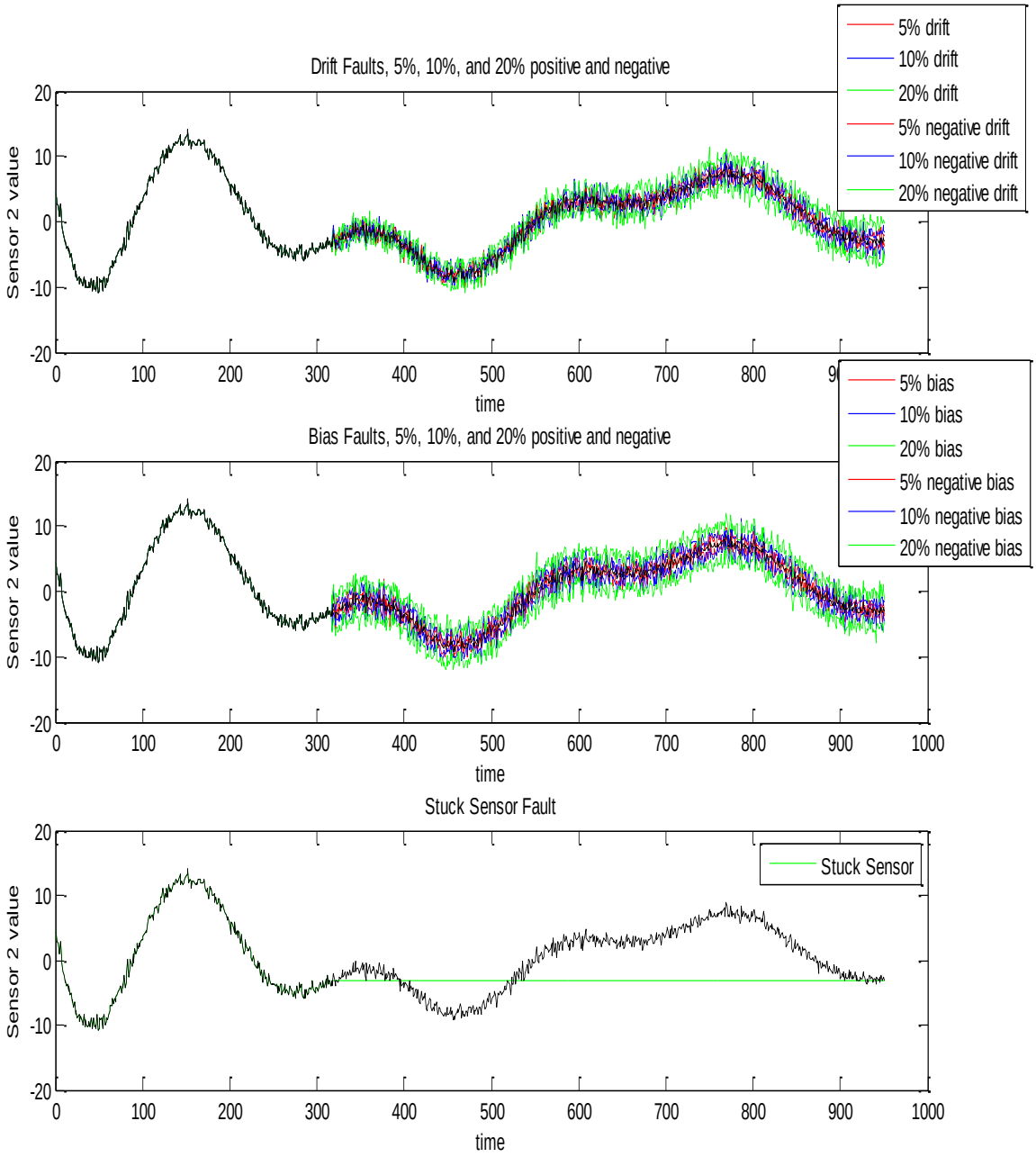


Figure 25:Drift, Bias, and Stuck Sensor Faults

To test the expanded condition differentiation ability of the ECM technique an expanded condition is needed. An expanded condition was generated using the same internal relationships in the data. This was done by using signals that were time dependent; using different time frames in the simulation allowed for an expanded condition. The differences between the variable 2 values are shown below in Fig.26. The range of the expanded condition is shown to be larger than the normal data range, hence classifying it as an expanded condition.

To test its application, the PCA ECM technique was applied to both the faulty data and the expanded condition. The initial generated data was used in creating the scores and loadings; these were applied to the faulty data and the expanded data to test the differentiation ability.

The application of the PCA ECM technique to the positive bias faults results in the ability to correctly detect the faults within the data. This is shown in Fig. 27, with the Q-statistic values for the faulty data exceeding the maximum Q-statistic for the non-fault condition. The T^2-statistics for the faulty conditions stay within the maximum range of the original data T^2-statistics. These faults would be categorized in the scenario 3 fault verdicts, where the Q-statistic is outside the limits and the T^2-statistic is within the limits.

Applying the PCA ECM technique to the positive drift faults results in a similar result to that of the bias faults. The ability to correctly detect the faults within the data is shown. Fig. 28 shows the Q-statistic values for the faulty data exceeding the maximum Q-statistic for the non-fault condition. As the drift increases, the Q-statistics increases showing a more severe fault. The T^2-statistics for the faulty conditions stay within the maximum range of the original data T^2-statistics. These faults would be categorized in the scenario 3 fault verdict, where the Q-statistic is outside the limits and the T^2-statistic is within the limits.

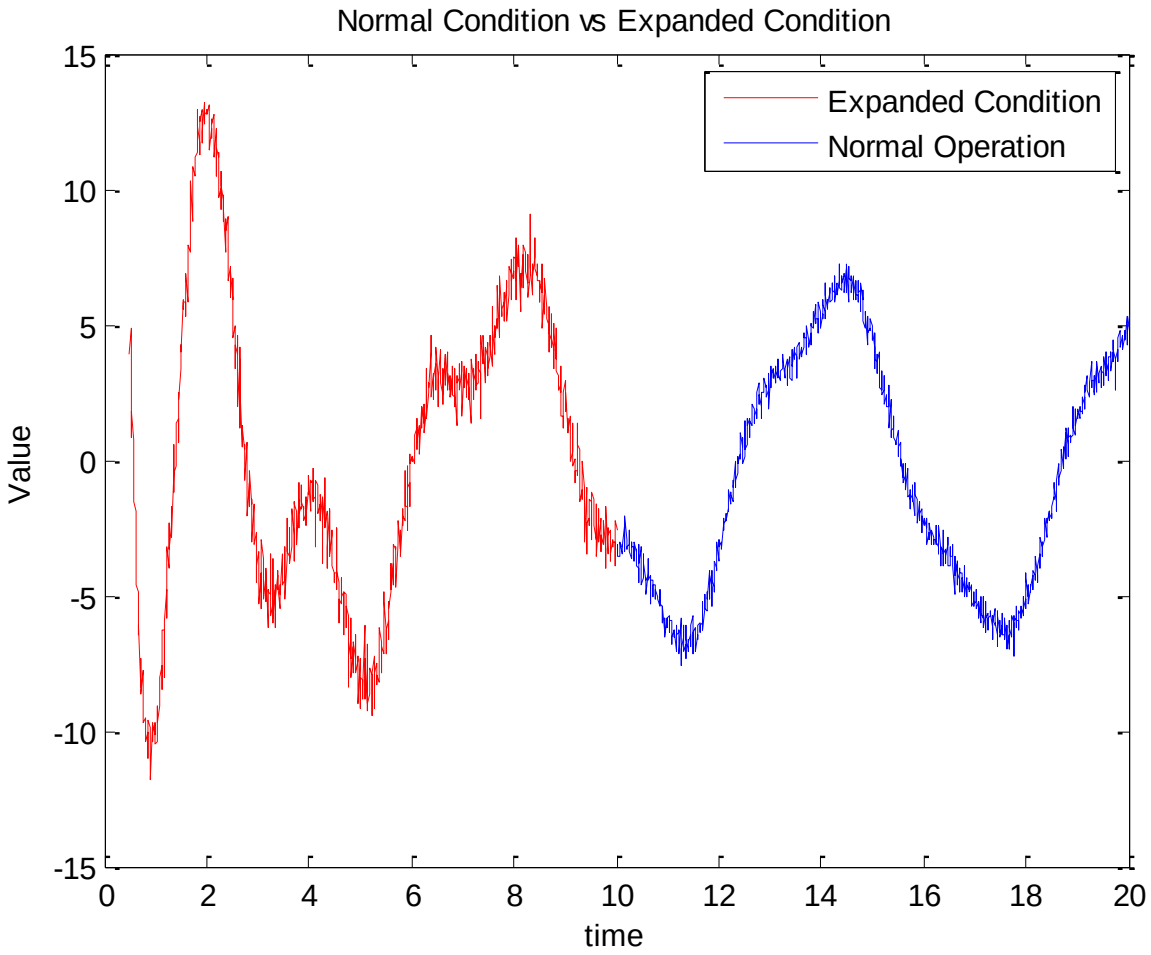


Figure 26: Normal Condition vs Expanded Condition

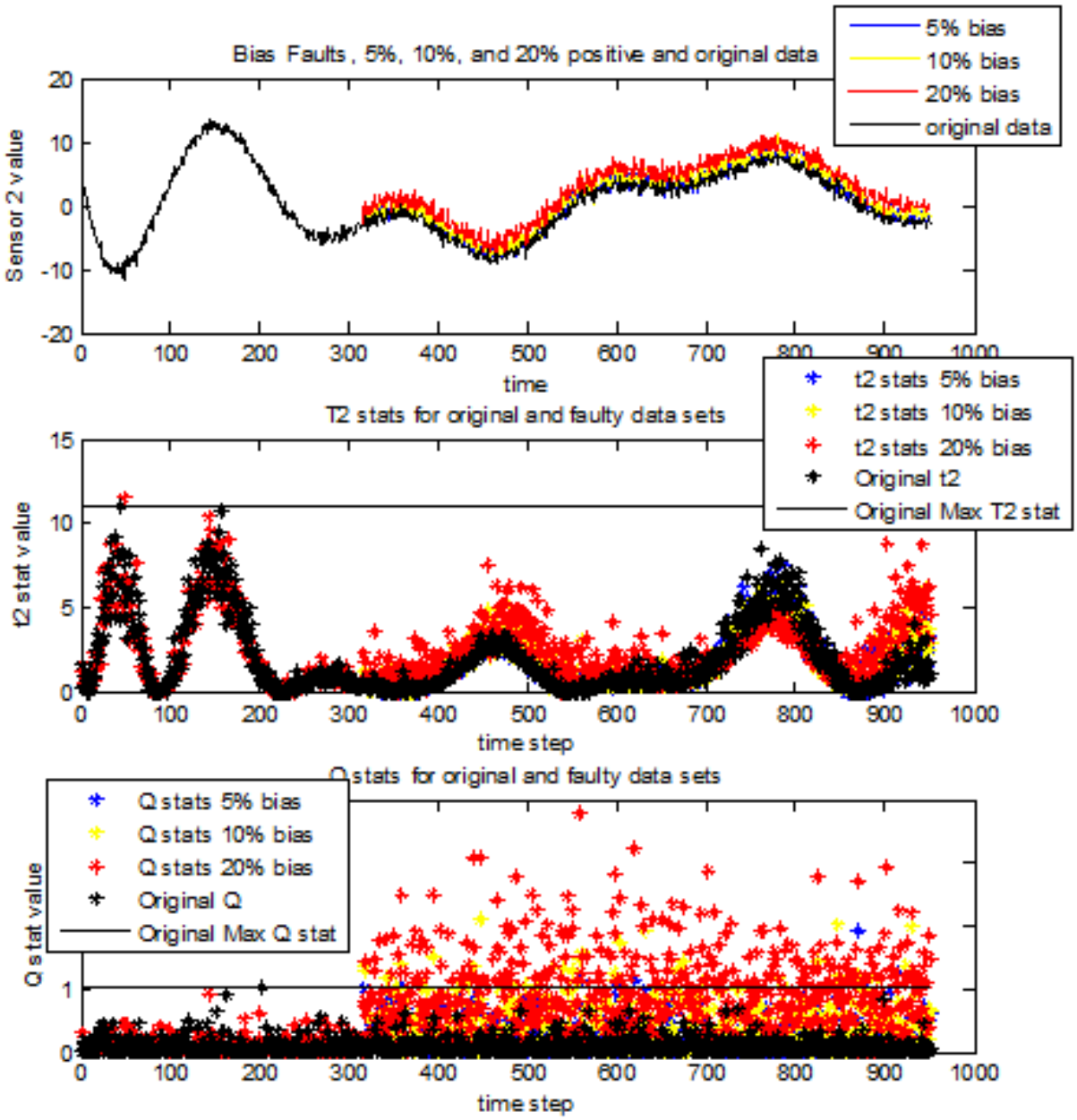


Figure 27: Positive Bias PCA ECM results

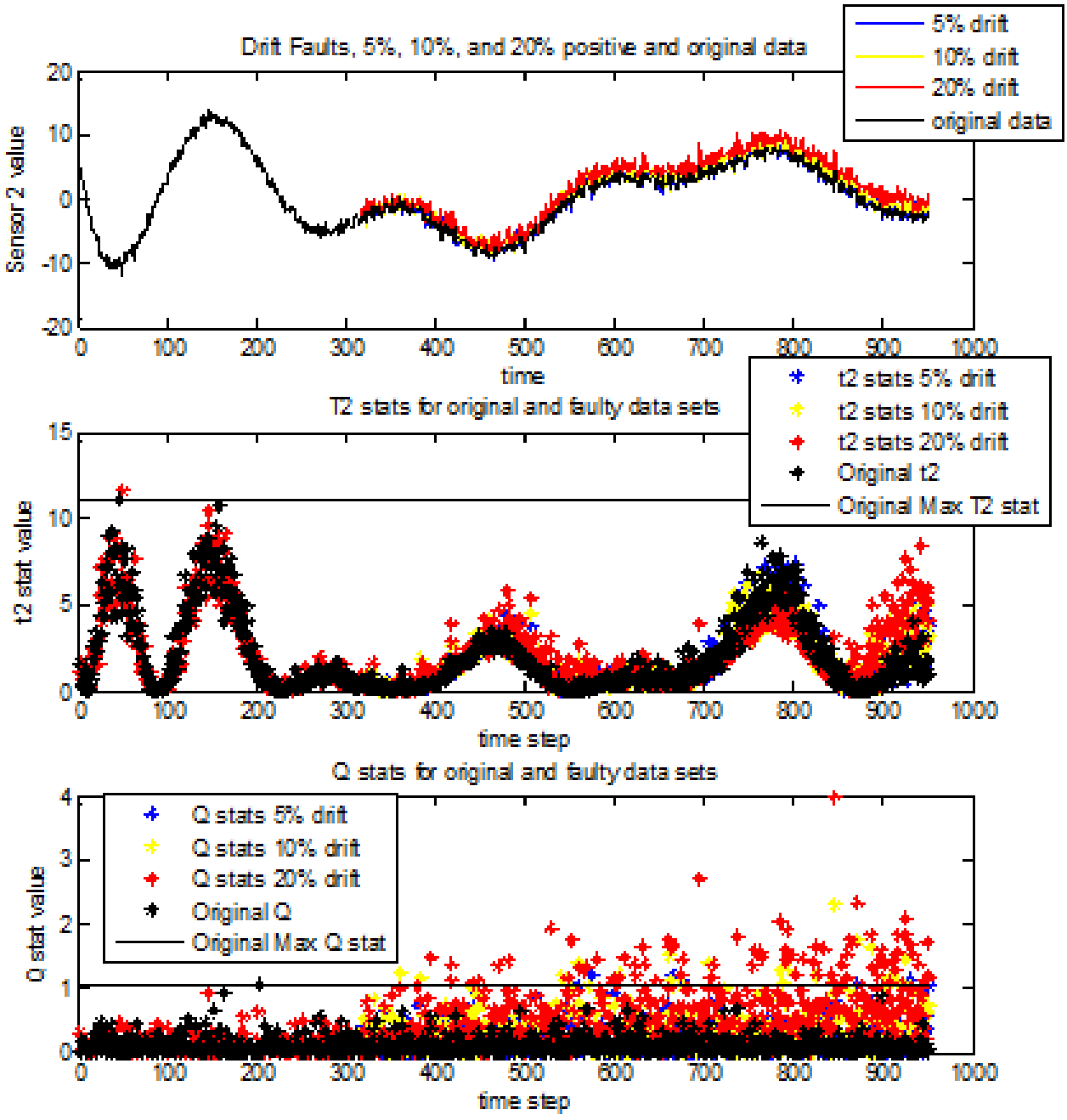


Figure 28: Positive Drifts PCA ECM results

The negative bias and drift data had similar results to the positive bias and drift data. The T^2-statistics increase beyond the limits for both of the negative fault cases. This changes the results from a scenario 3 fault verdict to a scenario 4 fault verdict, which has both the Q-statistic and the T^2-statistic increasing beyond their nominal limits.

Applying the PCA ECM technique to the stuck sensor fault results in a scenario 4 fault verdict. The ability to correctly detect the fault within the data is still shown; however, both the Q-statistics and the T^2-statistics increase beyond their nominal limits (Fig. 29).

All three of the faults were detected using the PCA ECM technique. This shows that the PCA ECM technique was useful in detecting different magnitudes of drifts and bias faults along with a stuck sensor fault. Fault scenarios 3 and 4 were used in detecting the faults. There are other fault detection techniques that could detect these faults as well; this ability of the PCA ECM is not what makes it useful in the adaptive environment. The ability to detect expanded conditions makes it useful in the adaptive environment. Expanded conditions are designated as scenario 2, where the Q-statistic is within the limits and the T^2-statistic increases beyond the limits. The data for variable 2 for the expanded condition was shown in Fig. 8. The results of applying the PCA ECM are shown in Fig. 30. This shows that the Q-statistic is within the limits and the T^2-statistic increases beyond its training limit. A scenario 2, as expected, is the result of applying the detection method to the expanded condition.

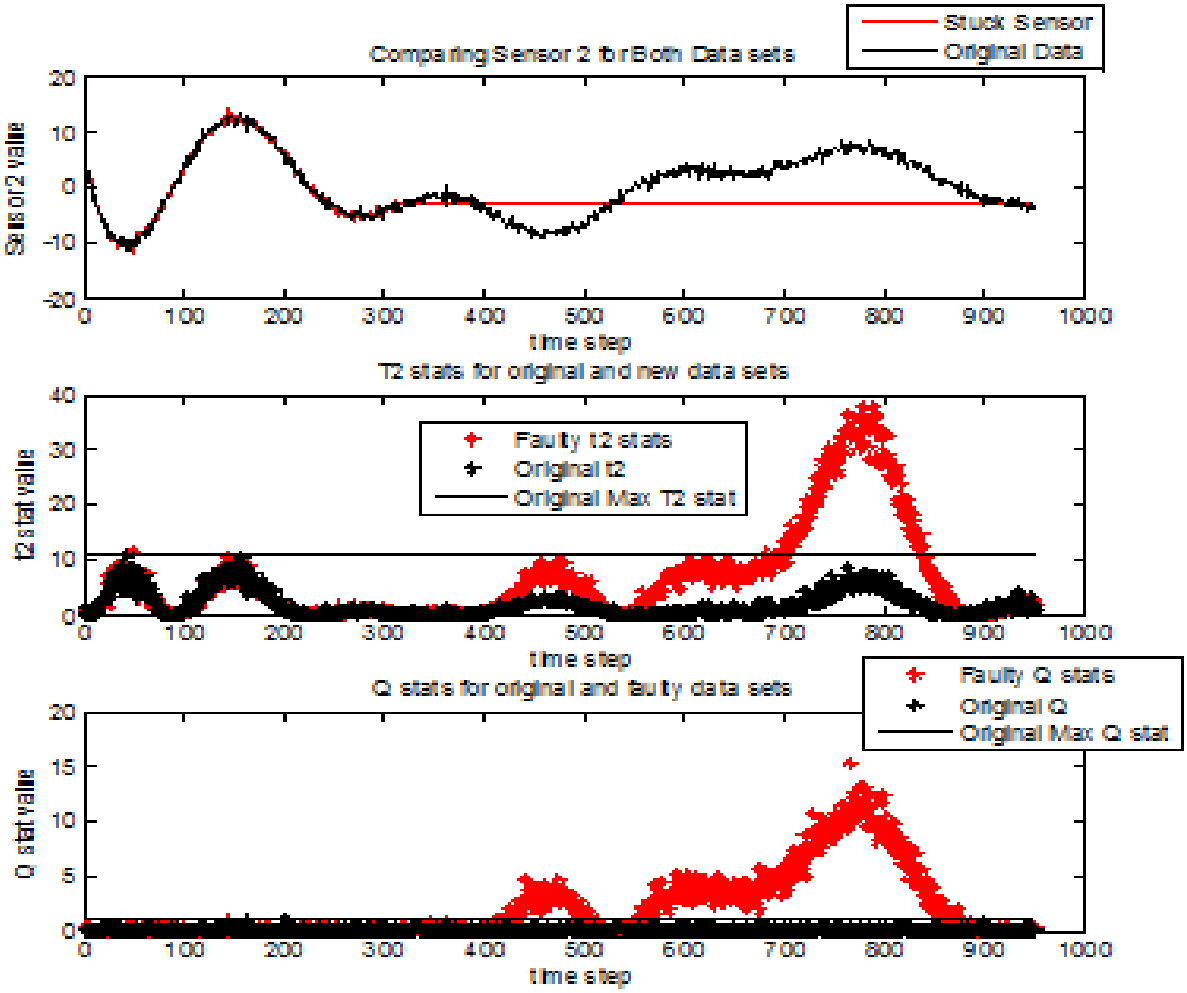


Figure 29: Stuck Sensor PCA ECM results

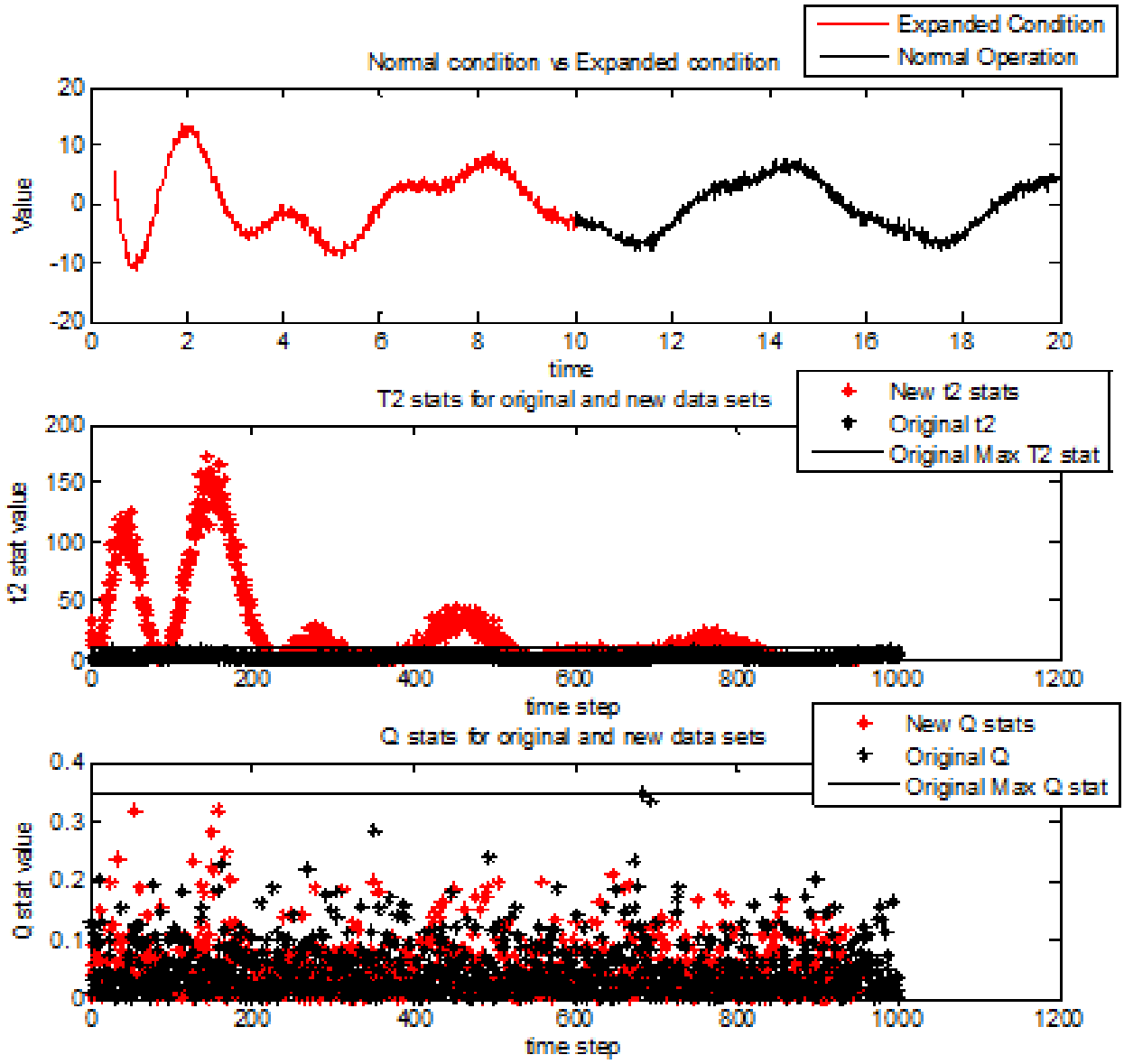


Figure 30: Normal Condition vs Expanded Condition

4.1.4 Principal Component Analysis Expanded Condition Monitoring test on flow loop data

The flow loop was built by Sergio Perillo and operated by Brian Wood, who supplied the data for this research. The SIMULINK model of the flow loop was designed by James Henkel. The flow loop, shown in Fig. 31, consists of a sump tank that contains the loop's water supply, a pump that pushes the water through the loop, and three computer controlled valves that control the flow from the pump to either of two vertical holding tanks. Flow rate sensors are placed in line with both tank inputs and each of their return lines back to the sump tank. The analog signals from each sensor: input, output flow rates, input, output valve positions, tank levels, and water and pump temperatures, are sent to the data acquisition boards where they are recorded.

The flow loop is controlled by the user to either manually adjust the valve positions to allow for more or less water to enter the tanks or to use a Proportional Integral (PI) controller to control the tanks' levels. A PI controller can be used to automatically maintain a desired water tank level in either one tank or both tanks. LABView software is used to control the flow loop through the interface shown in Fig. 32.

The test flow loop was used to obtain real data of the system's conditions, flow rates, valve positions, and tank levels, as they change with the PI controller's commands. The flow loop data was then used along with a SIMULINK model of the flow loop for application of the ANPM. The MATLAB SIMULINK model of the two tank control flow loop was developed by James Henkel. This model consists of two tanks with PI controlled valves attached to the tank inputs and gravity drained outputs and a pipe running between the tanks used to complicate the problem.

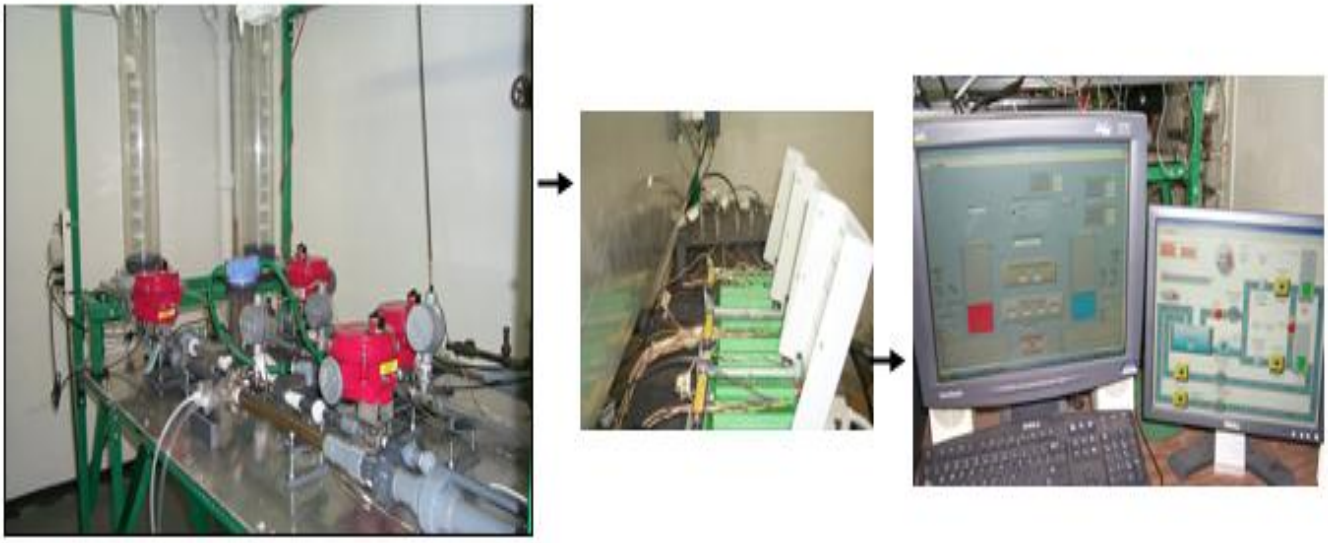


Figure 31: Picture of flow loop and progression of data signals (Wood 2008)

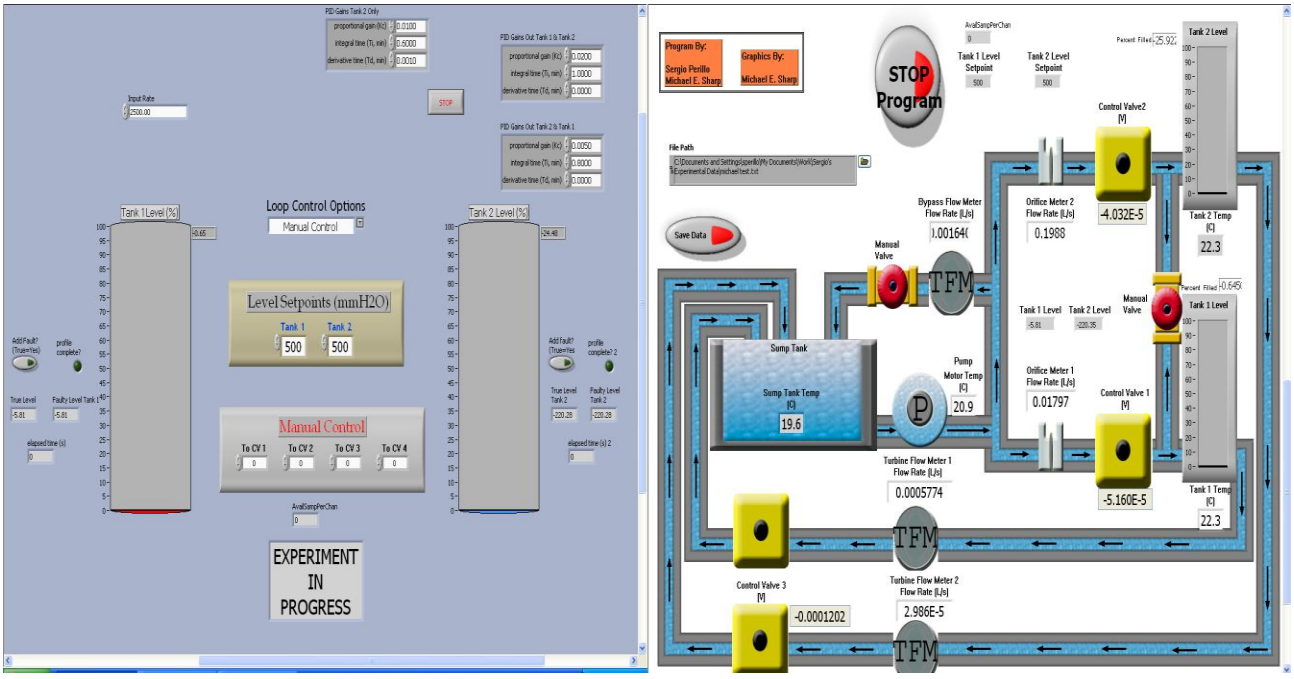


Figure 32: LABView User Interface to control the flow loop and collect data (Wood 2008)

The following equations are used to govern the flow loop process. First the initial parameters of interest are the cross sectional area of the tanks, initial level, and setpoint profile.

$$A = \pi r^2 \tag{4.1}$$

$$m = \rho h A = \rho V \tag{4.2}$$

And the outlet flow, Q in meters cubed per second, is a function of height, h, and can be defined as:

$$Q_{outlet} = vA_{outlet} \tag{4.3}$$

$$v_{outlet} = \sqrt{2gh} \tag{4.4}$$

$$Q_{outlet} = K\sqrt{h} \tag{4.5}$$

Where K is the resistance in the outlet piping and the inlet flow is a function of the valve position .

$$Q_{inlet} = f(u) \tag{4.6}$$

The change in height of the tanks is converted to millimeters and found by:

$$\dot{h} = \frac{(Q_{inlet} - Q_{outlet})}{A} = \frac{(f(u) - K\sqrt{h})}{A} \tag{4.7}$$

Then the inlet flow can be found as:

$$Q_{inlet} = f(u) = \dot{h}(A) + K\sqrt{h} \tag{4.8}$$

When the valve between the tanks is open the one tank equations do not hold and the volumetric flow across the tanks is calculated as:

$$Q_{cross} = A_c v_c = A_c \left[sign(h_1 - h_2) \right] \sqrt{2g\left|h_1 - h_2\right|} \tag{4.9}$$

So the change is height can be found

$$\dot{h} = \frac{(Q_{inlet} - Q_{outlet} + Q_{cross})}{A} = \frac{(f(u) - K\sqrt{h} + A_c \left[sign(h_1 - h_2) \right] \sqrt{2g\left|h_1 - h_2\right|})}{A} \tag{4.10}$$

Then the inlet flow is:

$$Q_{inlet} = f(u) = \dot{h}(A) + K\sqrt{h} - A_c\left[sign(h_1 - h_2)\right]\sqrt{2g\left|h_1 - h_2\right|} \qquad \textbf{(4.11)}$$

A schematic of the two tank setup can be seen in Fig. 33.

The initial parameters such as initial level, valve position, setpoint profile, and pipe size are needed for the SIMULINK model. The same sensors were used within the SIMULINK model as the actual flow loop. The model and the input setup can be seen in more detail in Appendix A. Table 2 below gives a description of the ten sensors used in the flow loop.

Applying the PCA ECM technique to the flow loop data similar results are shown when signals with linear relationships are used. To be able to compare this application to the ANPM use, the original PCA statistics need to be based on the first principle model (FPM) data. The actual flow loop data is monitored for faults or expanded conditions. Three actual faults were added to the flow loop operation. These faults included a return valve blockage, 60mm level drift, and a pump failure simulation. These faults can be detected by analyzing different sensors. Fig. 34 shows the results on the level of tank 1 during normal operation compared to the different faults under the same profile operation.

The first four sensors which are the two tank levels and the two setpoints have linear relationships. The PCA ECM technique was applied to the first four sensors using the FPM data as the basis. The first principal component (PC) describes over 99% of the variance within the data, so only one PC is used. Applying this technique to the normal operation of the flow loop no faults are detected in profile 1 data or profile 2 data. Fig. 35 shows the profile 1 results, and Fig. 36 shows the profile 2 results.

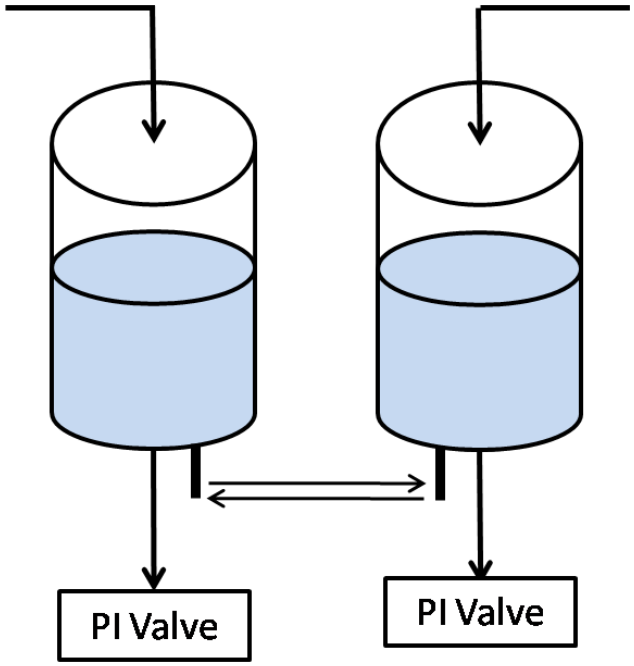


Figure 33: Two tank model

Table 2: Flow Loop Sensor Descriptions

Variable #	Sensor Description
Variable 1	Tank 1 Setpoint
Variable 2	Tank 1 Level
Variable 3	Tank 2 Setpoint
Variable 4	Tank 2 Level
Variable 5	Tank 1 Inlet Flow Rate
Variable 6	Tank 2 Inlet Flow Rate
Variable 7	Tank 2 Outlet Flow Rate
Variable 8	Tank 1 Outlet Flow Rate
Variable 9	Voltage to Control Valve 1
Variable 10	Voltage to Control Valve 2

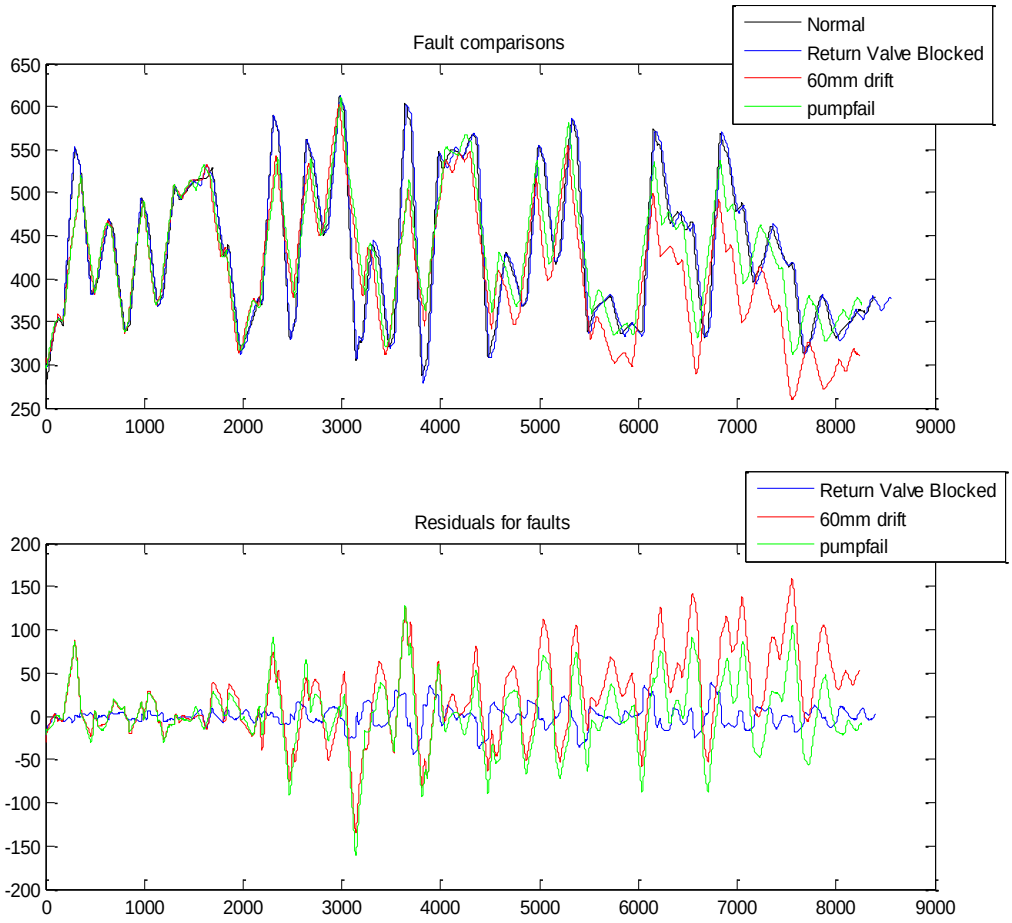


Figure 34: Tank 1 Level Fault Comparison

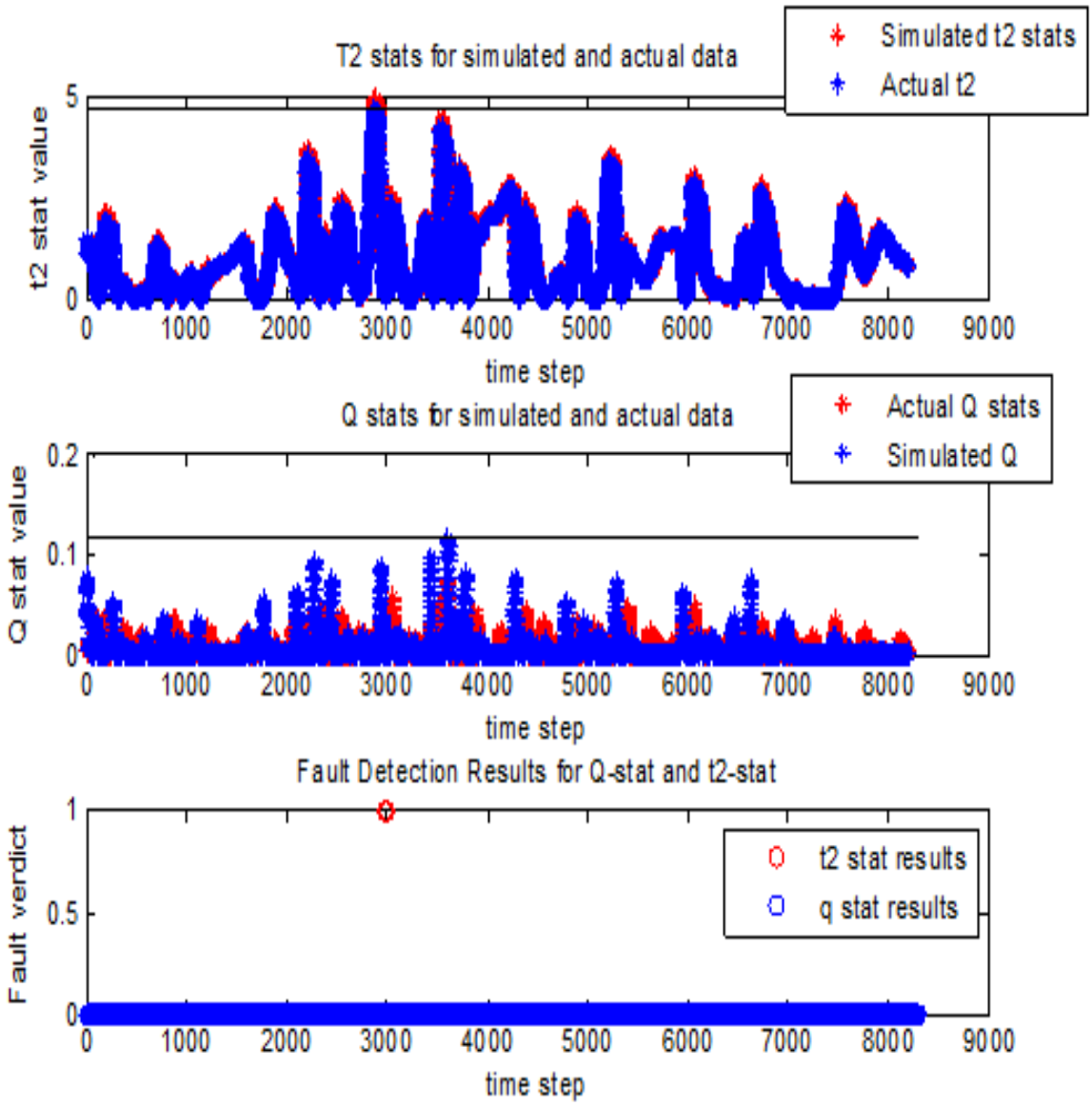


Figure 35: PCA ECM application on profile 1

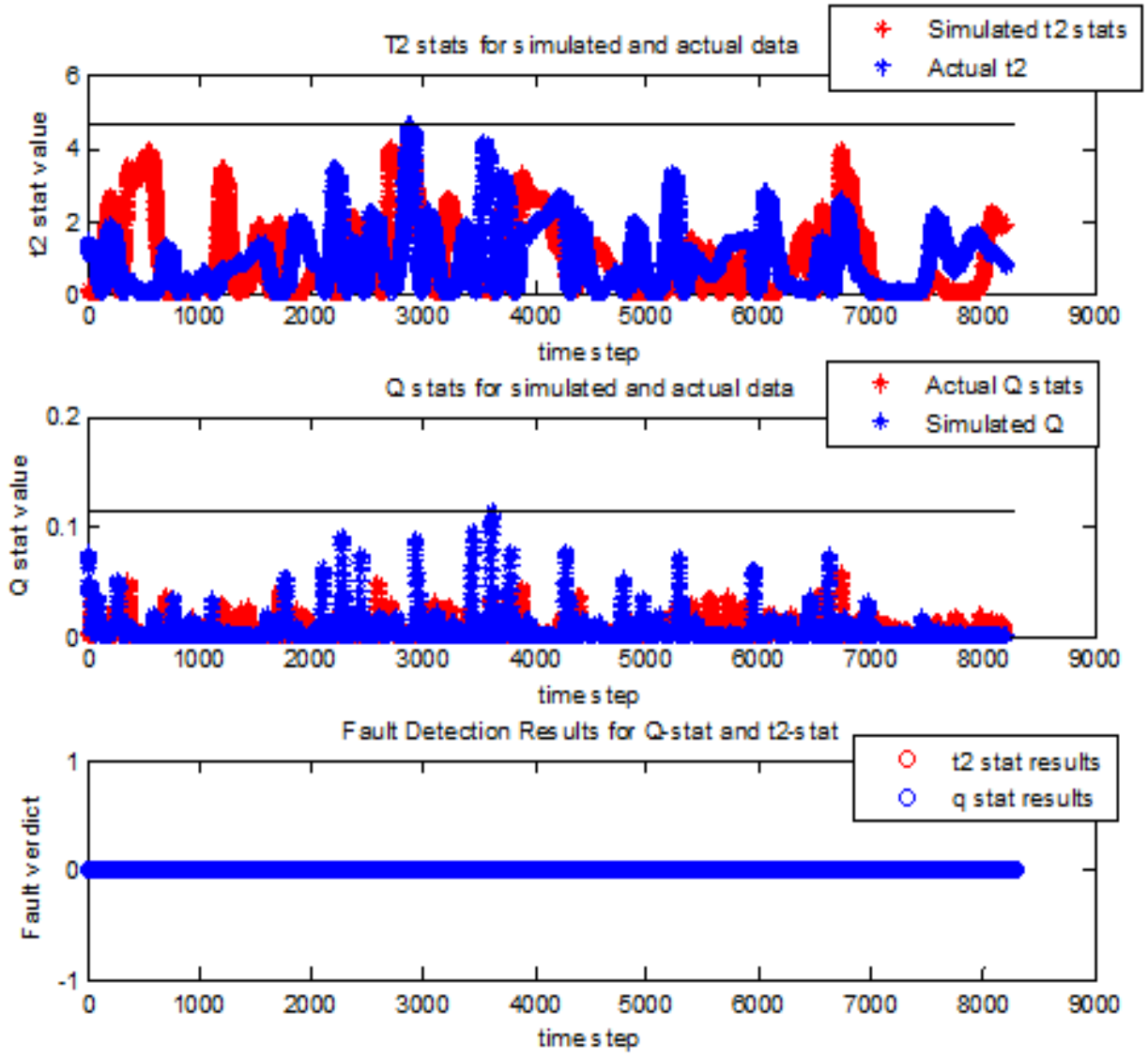


Figure 36: PCA ECM application on profile 2

Applying this technique to the 60mm level drift fault condition of the flow loop the faults were detected in profile 1 faulty data and profile 2 faulty data. Fig. 37 shows the profile 1 results, and Fig. 38 shows the profile 2 results.

The addition of the two outlet flow sensors creates problems for the PCA ECM technique. This problem is created by the non-linear relationships between the outlet flow and the tank levels. With only the linear relationships included the faults were detected and normal operation data was shown to have no faults. The addition of the two flow sensors breaks down the basis for the PCA ECM technique. Fig. 39 shows the application of the PCA ECM technique on the profile 1, 60mm drift fault. Fig. 40 shows the application of the PCA ECM technique on the profile 2, 60mm drift fault.

This demonstration highlights the importance of having linear relationships within the data when using the PCA ECM technique. It was shown on actual data from the flow loop the ability of this technique to detect faults. The detection of faults within the ANPM is very important. This fault detection ability will govern the adaptive ability of the model.

2.1 KERNEL PRINCIPAL COMPONENT ANALYSIS EXPANDED CONDITION MONITORING RESULTS

To test the KPCA ECM technique a simple nonlinear data set was created. This data set contains three sensors with the following relationships shown in Fig. 41.

To test the expanded condition monitoring capability of KPCA it is essential for this technique to differentiate between expanded conditions and faults. Drift and bias faults were

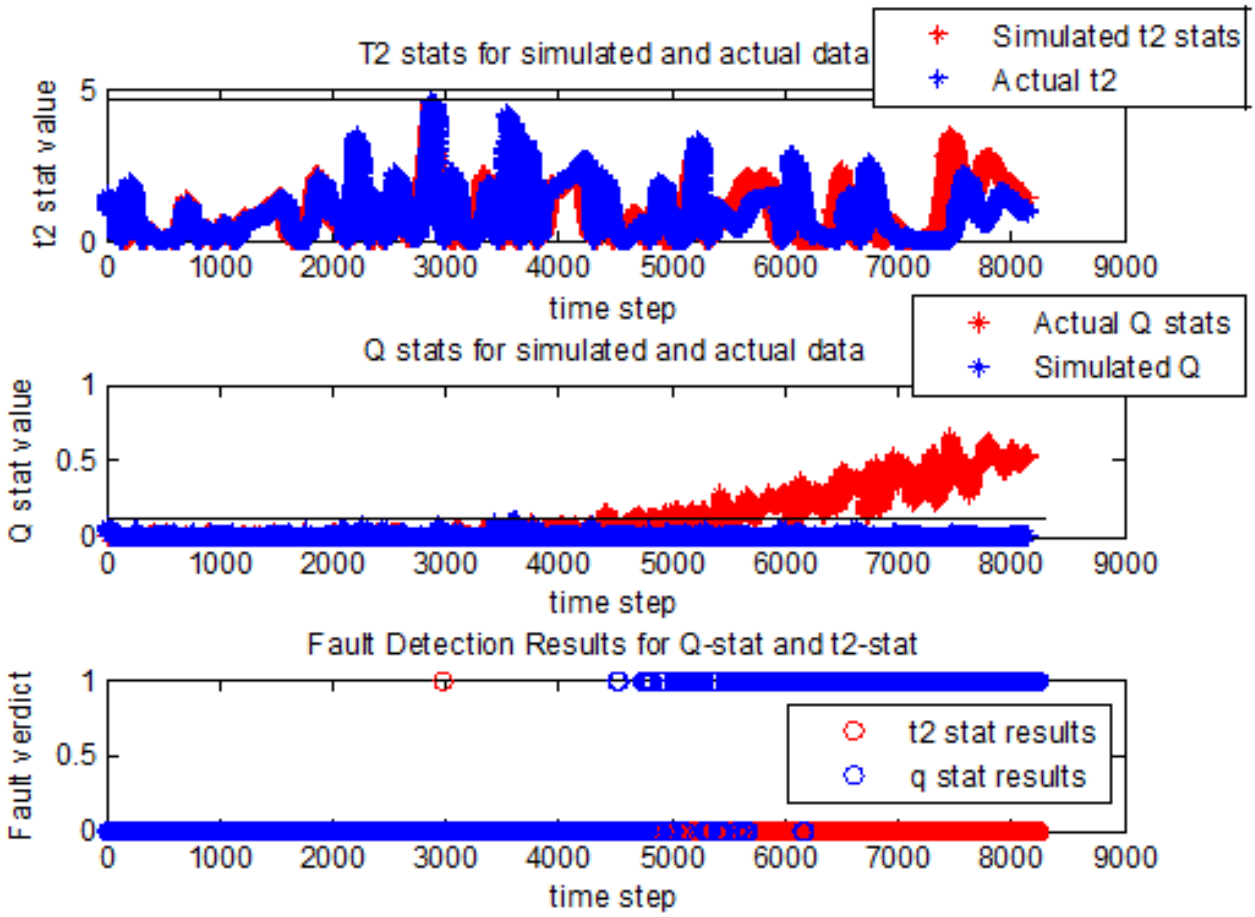


Figure 37: PCA ECM application on profile 1 60mm drift

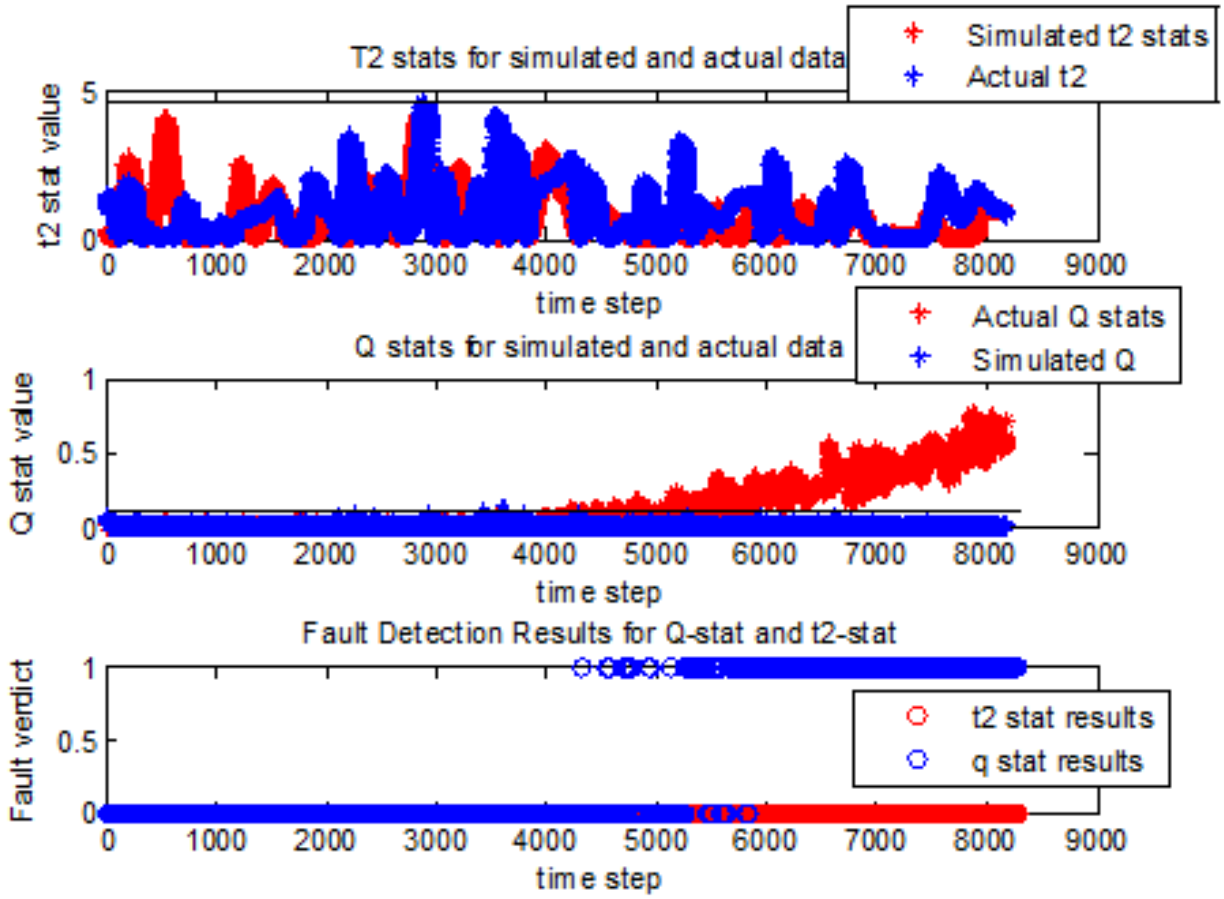


Figure 38: PCA ECM application on profile 2 60mm drift

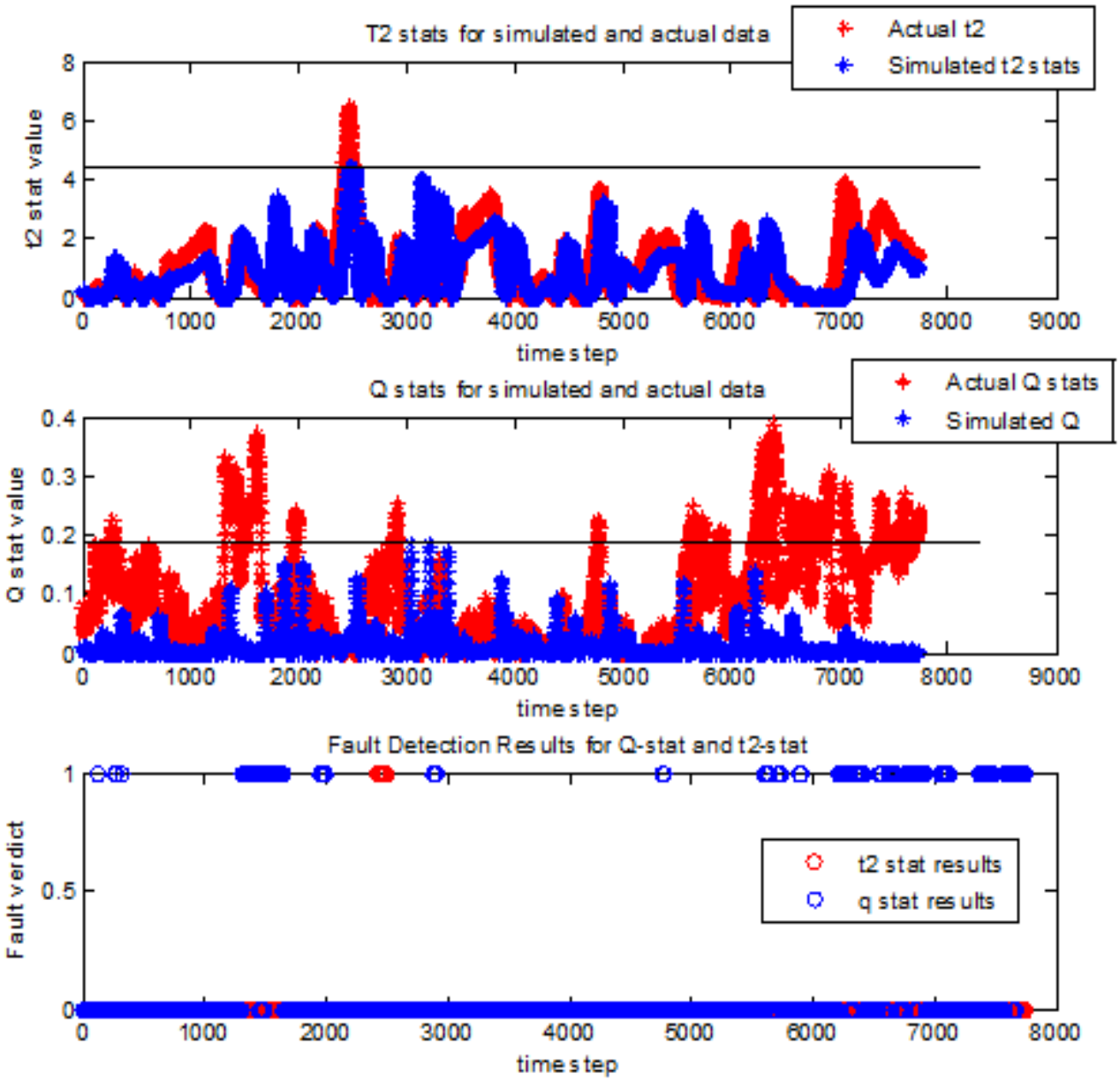


Figure 39: PCA ECM application profile 1 60mm drift 6 sensors

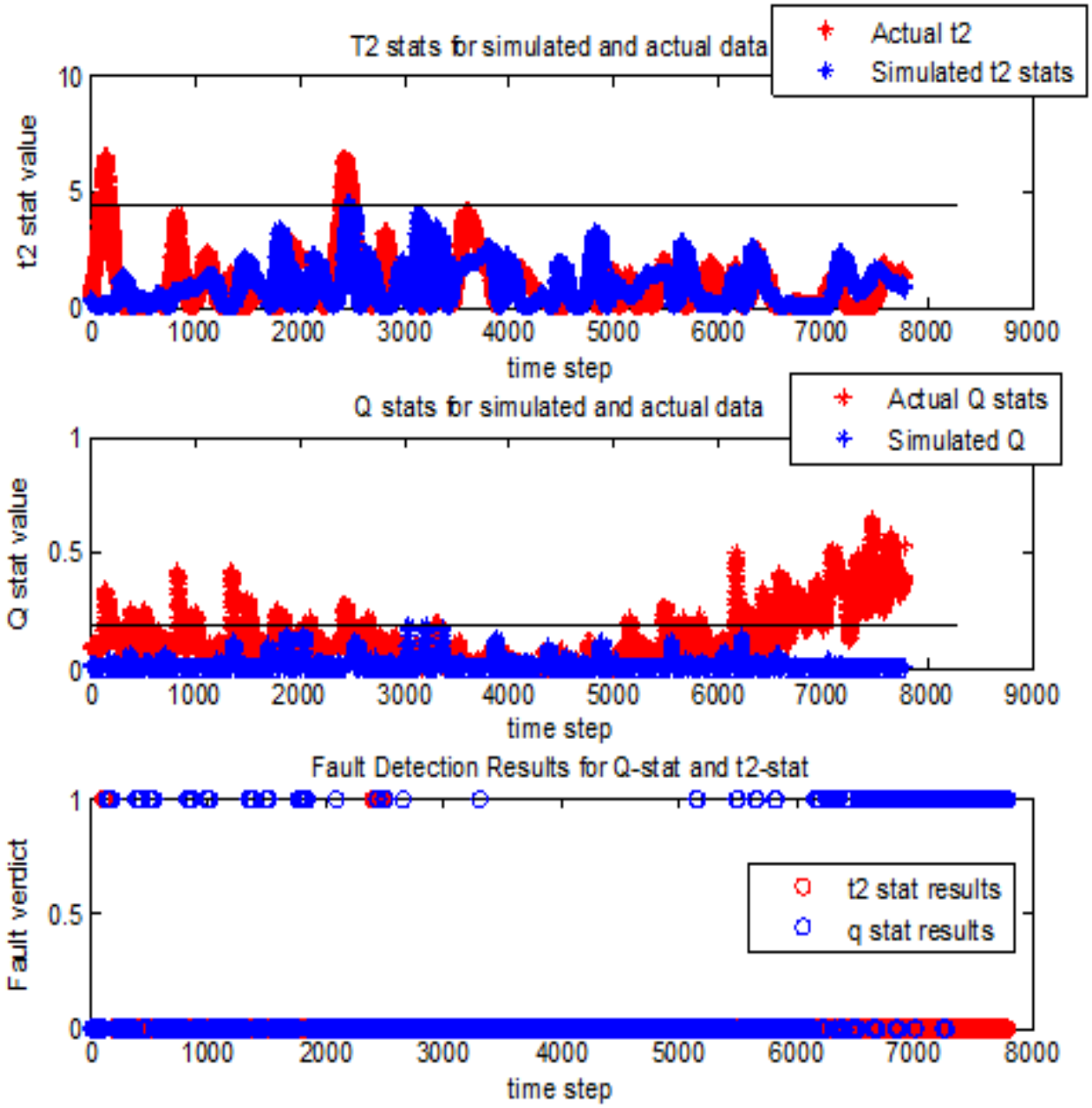


Figure 40: PCA ECM application profile2 60mm drift 6 sensors

added to sensor 2 of the nonlinear relationship data set to test the fault detection capability of the KPCA technique (Fig.42).

Since there are a number of possible kernels the following kernels were tested (Table 3). The expanded condition was created by an extension of the normal data for t=950 to t=1450, which can be seen in Fig. 43.

The following steps were followed to test the KPCA ECM.

Initial Steps:

1) Mean center the normal operational data.

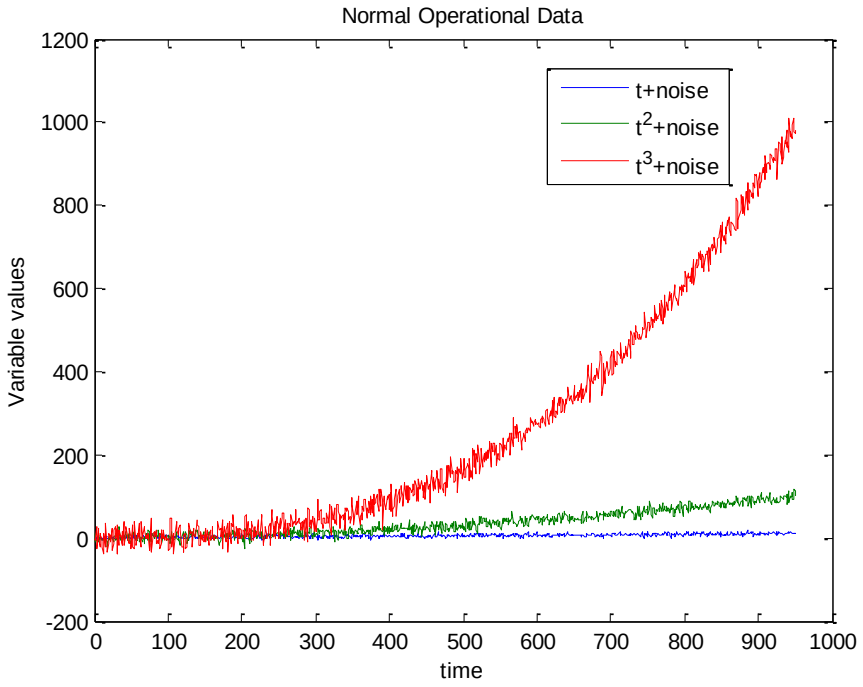


Figure 41: Nonlinear Relationship Data Set

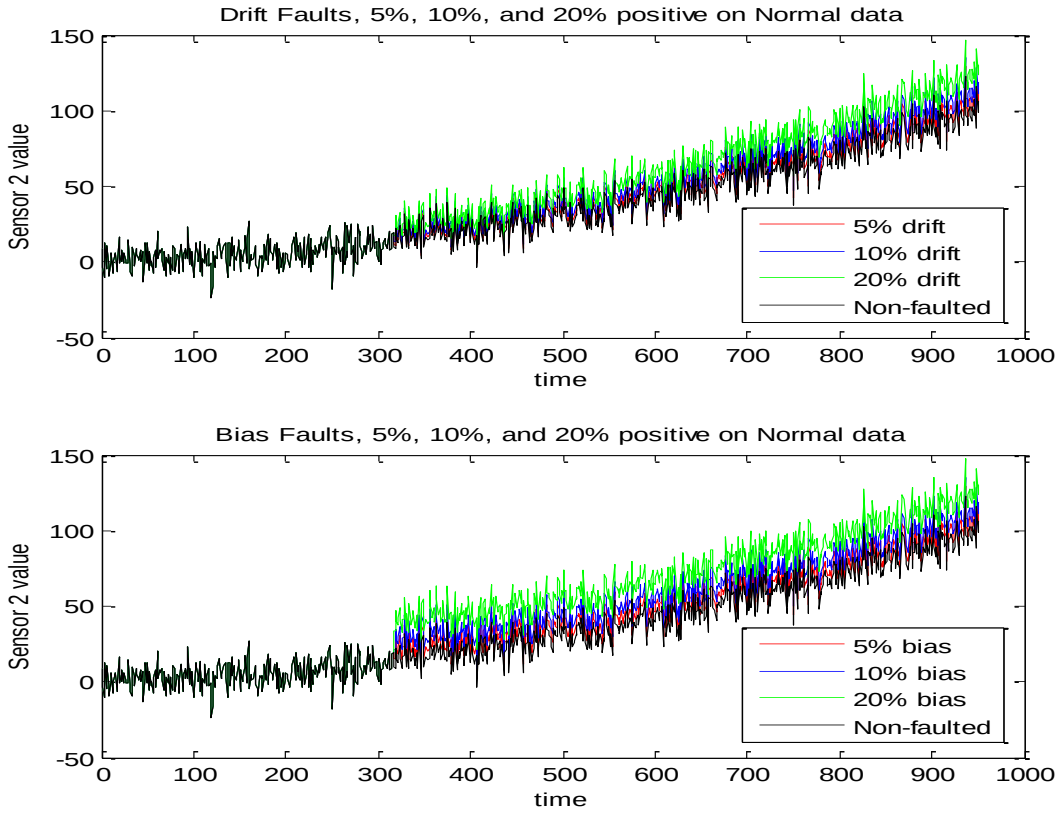


Figure 42: Faulted data for nonlinear relationships

Table 3: Kernels choices

Kernels	Variations
$k(x_i, x_j) = \left(x_i \bullet x_j\right)^d$	d=1,2,3
$k(x_i, x_j) = \exp\left(-\frac{\left(x_i - x_j\right)^2}{c}\right)$	c= 1, 2, 4, 8

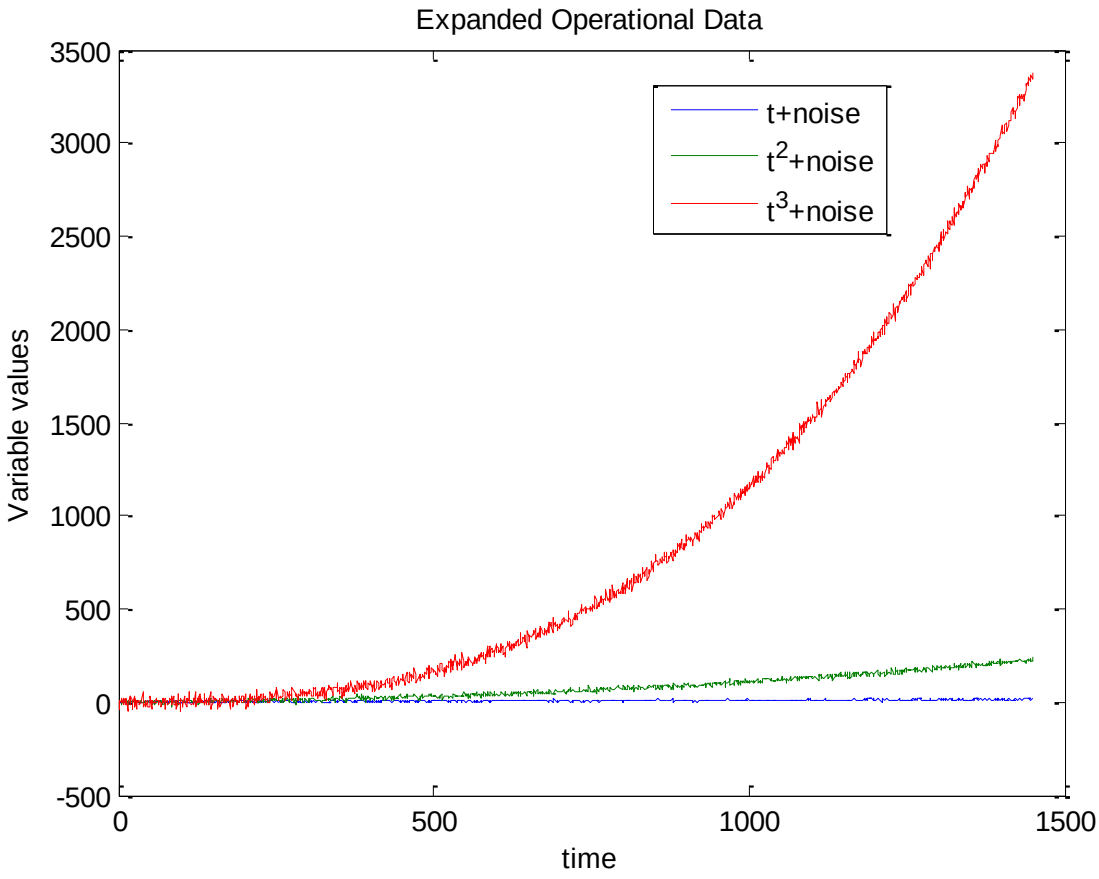


Figure 43: Expanded Operational Data

2) Apply the Kernel function to transform the data from the input space to the feature space.

3) Mean center the feature space data.

4) Apply PCA to this data and follow the discussed PCA ECM approach

ECM Technique Steps:

5) The data is first mean centered using the mean from step 1 in the initial steps.

6) The Kernel function is applied to transform the data from the input space to the feature space.

7) The feature space data is then mean centered using the means from step 3 in the initial steps.

8) PCA is applied to the feature space.

9) The same procedure as discussed for PCA ECM is then followed.

These steps were applied for the different kernels to first test the monitoring capability of KPCA by applying it to faulty data that is not from the expanded condition. The first kernel tested it the polynomial kernel with d=1. Fig. 44 shows the results of applying this to a bias on sensor 2. Other drifts and bias faults were tested with similar results and the interested reader is referred to Appendix B. The KPCA technique can detect the bias, this shows that KPCA can be applied for monitoring puposes. To test the expanded condition monitoring capability this same kernel is applied to an expanded condition without an added fault, these results can be seen in Fig. 45.

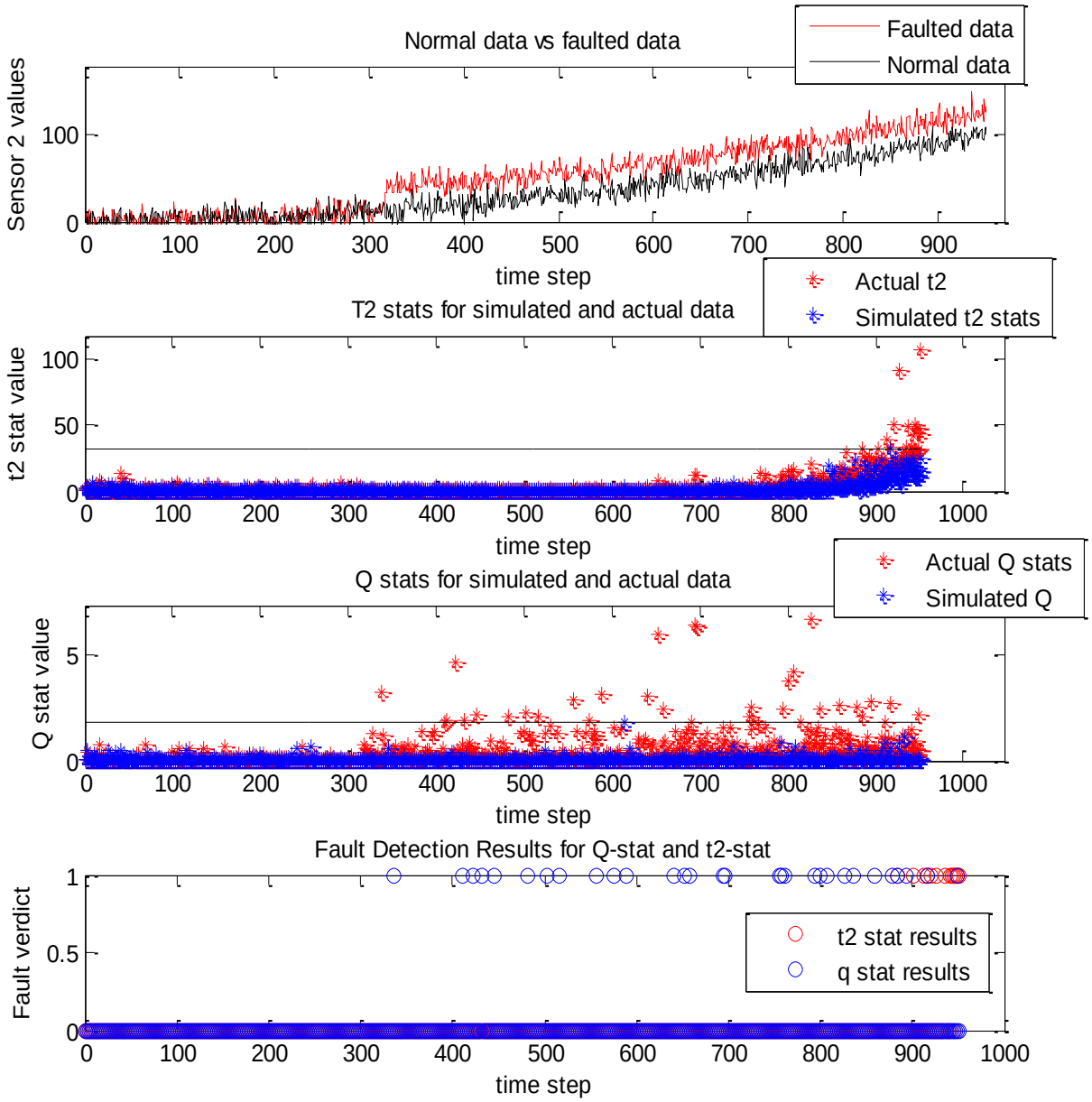


Figure 44: KPCA results for fault detection without an expanded condition

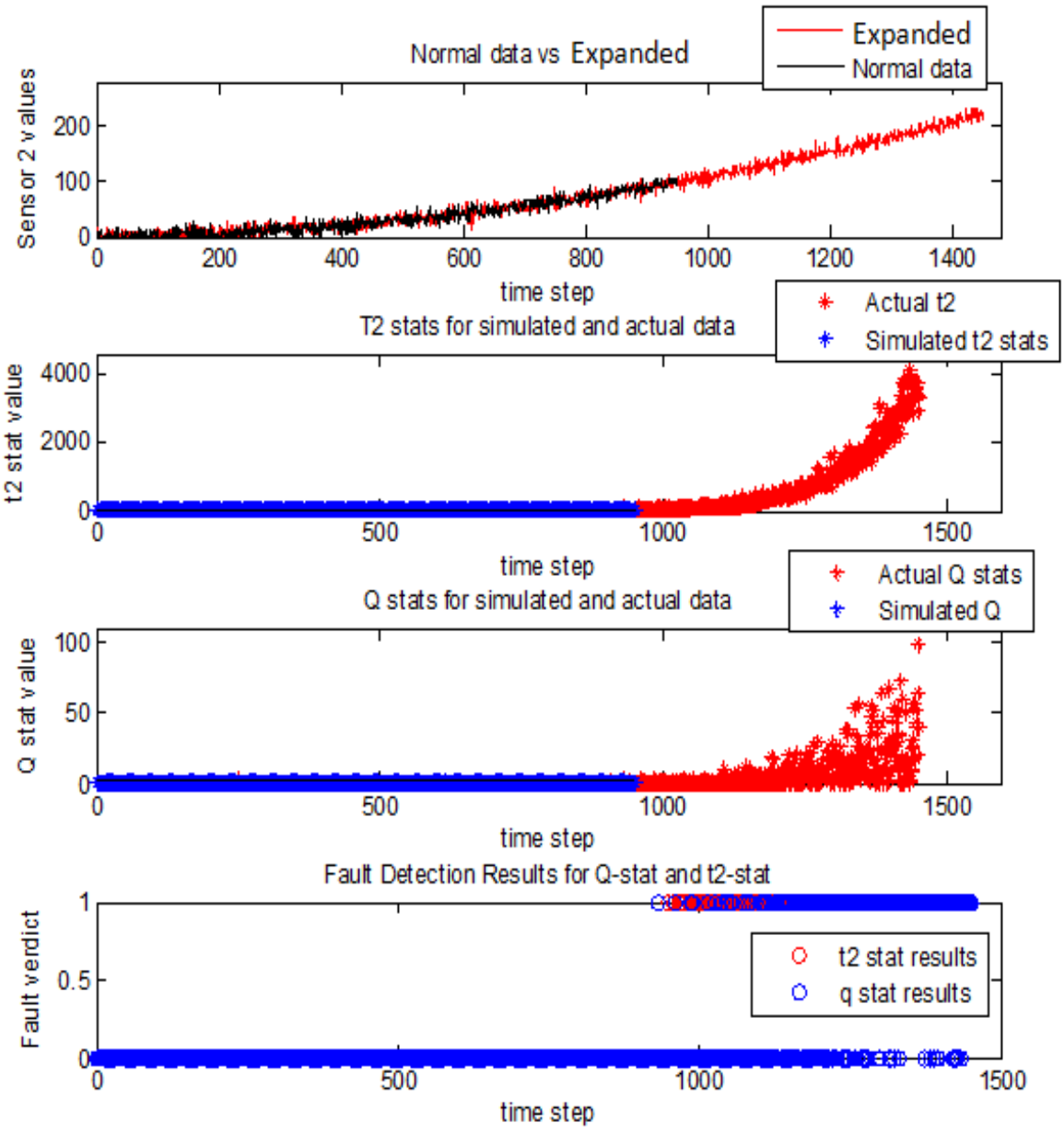


Figure 45: KPCA results on an expanded condition

For the ECM technique it is essential that an expanded condition can be differentiated from a fault. The indication that an expanded condition has occurred is the increase in the T squared statistic while the Q statistic is still within the bounds. As can be seen, that is not the case for applying this kernel, both statistics increase beyond the bounds as soon as the data exceeds the previously definded range. So this indicates that this choice for a kernel is not a suitable case for ECM. There were similar results for the other values of d in the polynomial kernel, this can be seen in Appendix B.

Another kernel that was tested is the radial basis kernel, first investigated is the case with c=4. Applying KPCA using this kernel to faulted data within the normal operating conditions is shown in Fig 46.

As with the polynomial kernel, the radial basis kernel with c=4, is able to detect the fault. Other faults were tested with similar results. The KPCA technique can detect the bias, this shows that KPCA can be applied for monitoring puposes. To test the expanded condition monitoring capability this same kernel is applied to an expanded condition without an added fault, these results can be seen in Fig. 47.

For the ECM technique it is essential that an expanded condition can be differentiated from a fault. As can be seen, that is not the case for applying this kernel, both statistics increase beyond the bounds as soon as the data exceeds the previously definded range. So this indicates that this choice for a kernel is not a suitable case for ECM. There were similar results for the other values of c in the radial basis kernel, this can be seen in Appendix B.

These results for a simple nonlinear relationship show that the KPCA technique is not a suitable choice for ECM applications. Testing different kernel functions and data sets have

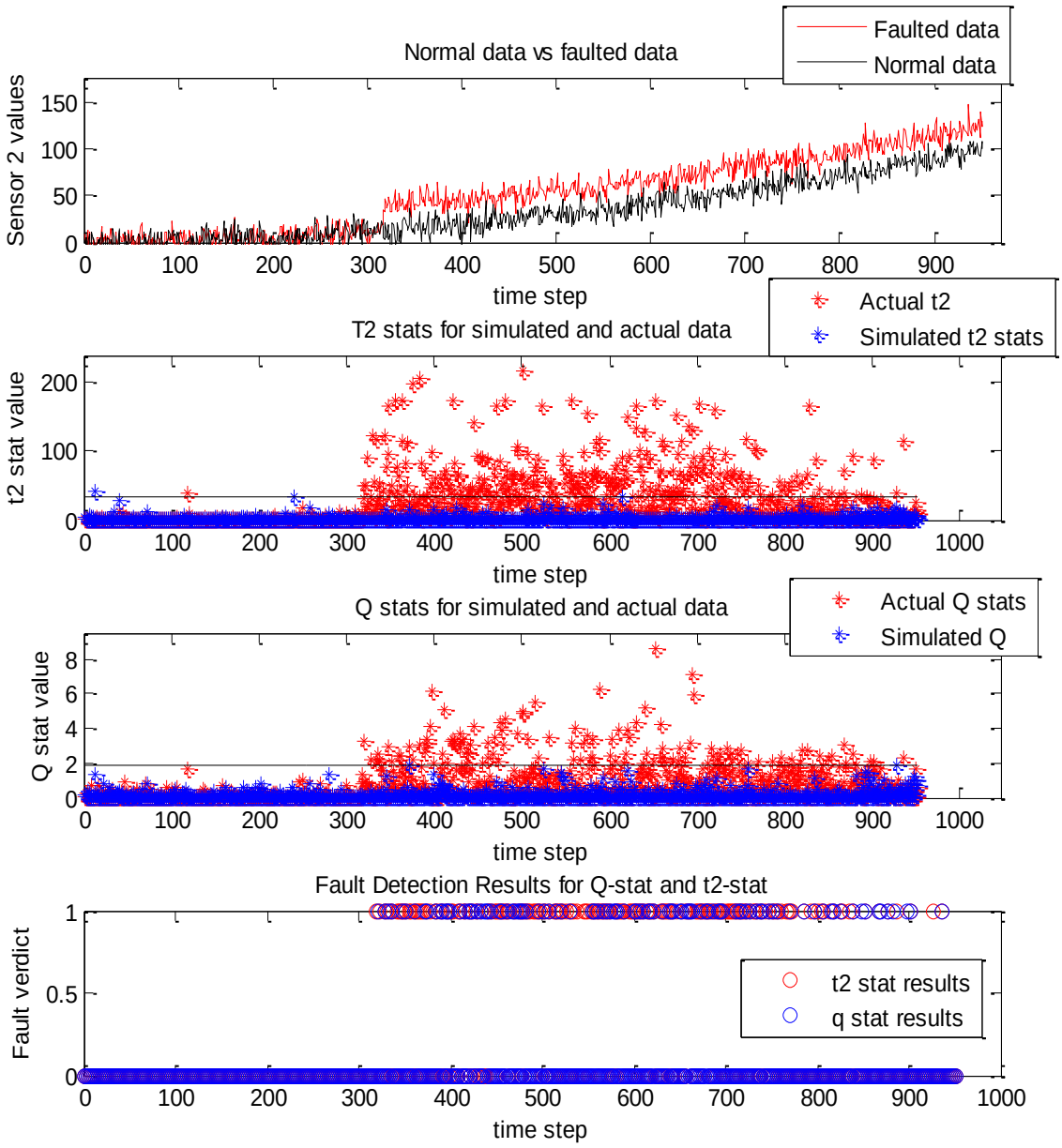


Figure 46: KPCA fault detection capability

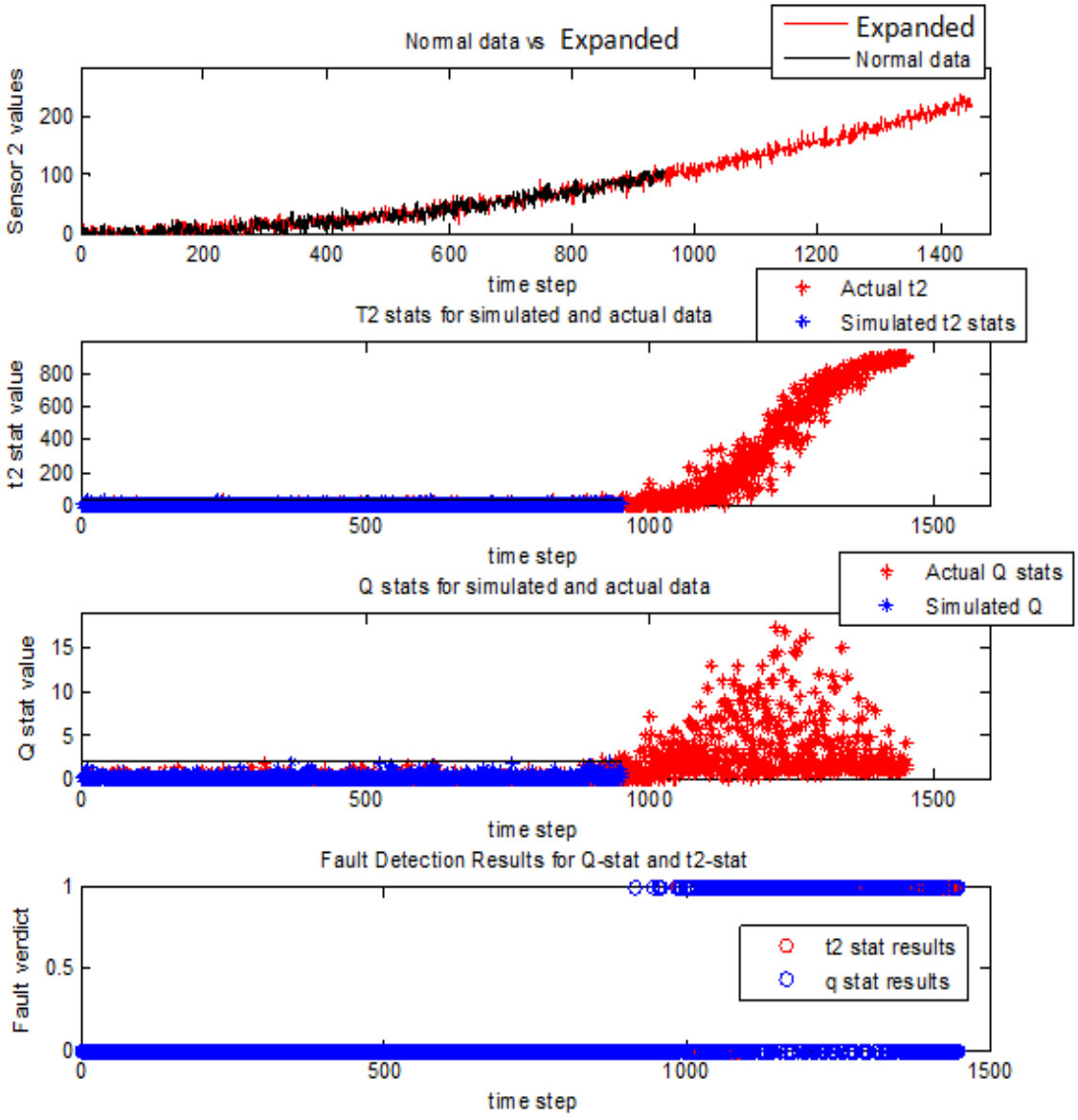


Figure 47: KPCA application to expanded condition

shown that using KPCA in and ECM does not provide the ability to differentiate between faults and expanded conditions. Expanded conditions and found to exceed the Q statistic beyond the bounds inticating a fault has occurred. This could be because KPCA is a nonlinear technique, when extending nonlinear models beyond their training ranges the results are not to be trusted.

The use of the PCA ECM technique in differentiating between fault conditions and expanded conditions was demonstrated. This is useful within an adaptive model, such as the ANPM to differentiate between the three conditions. The system will be in the normal condition, expanded condition, or a fault condition. The adaptation will depend on which condition the data is in; it is important to have a method for expanded condition differentiation. However, there can be issues with the PCA ECM method when non-linear relationships are present within the data. The KPCA techniques were investigated for use in this area; however, these techniques have been shown not to work in ECM. The method for handling this problem was a combination of the PCA ECM technique were applicable and absolute value checks.

2.2 HEAT EXCHANGER MODEL

A simple heat exchanger model was created by James Henkel using MATLAB's SIMULINK platform. The heat exchanger modeled is a copper tube and shell structure, 24" long, with 31 internal tubes. The internal tubes have a 0.25" diameter and the shell had a 2.5" diameter. The hot water flows through the tube side and the cold water flows through the shell side. The SIMULINK model includes 6 outputs: an inlet cold leg flow rate, an inlet cold leg temperature, an outlet cold leg temperature, an inlet hot leg flow rate, an inlet hot leg temperature, and an outlet hot leg temperature (Fig. 48). To better simulate real data acquisition,

independent sensor noise is added to each sensor, 2.0% noise to the flow meters and 0.2% noise to the thermometers.

The Log Mean Temperature or the Effectiveness-NTU method can be used for the heat exchanger first principle model representation. Both these methods use energy balance equations that describe the energy transferred to the cold fluid equal to the energy transferred from the hot fluid.

$$Q = \dot{m}c_p^h \Delta T^h = -\dot{m}c_p^c \Delta T^c \tag{4.12}$$

Where Q = heat transfer

$\dot{m}$ = mass flow rate

c_p = specific heat

ΔT = temperature difference

h, c = superscripts for hot and cold

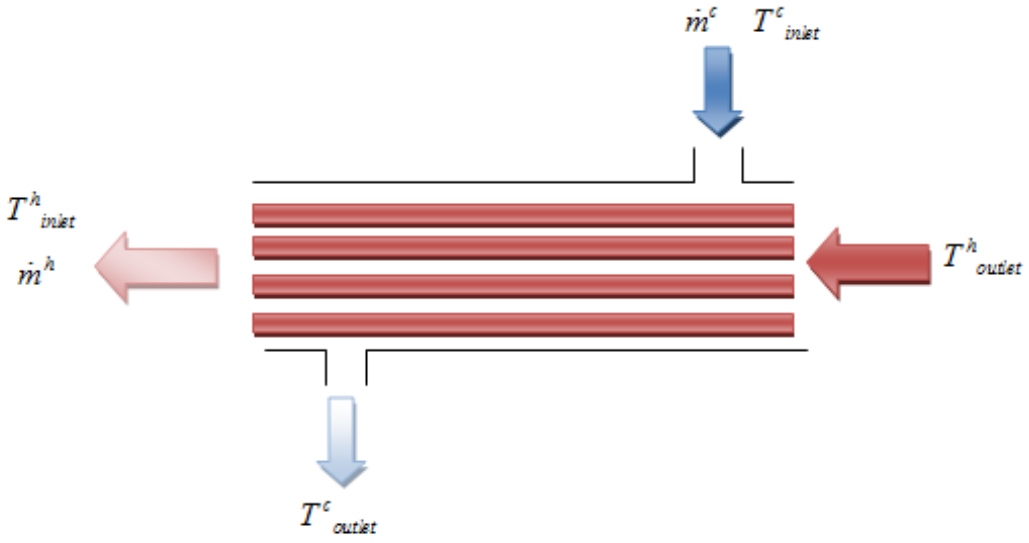


Figure 48: Heat Exchanger

For this model, it is assumed that the inlet flows and inlet temperatures are known; therefore, the outlet flow temperatures are calculated using the Effectiveness-NTU method described in (Schmidt et al. 1993, Penha et al. 2001). This method is useful when the output temperatures are unkown and is based on the effectiveness of the heat exchanger.

This method is based on the effectiveness of the heat exchanger in transferring a given amount of heat, with effectiveness defined as the ratio between the actual heat transfer and the maximum possible heat transfer.

$$e \equiv \frac{Q}{Q_{\max}} \tag{4.13}$$

Where the maximum heat transfer is:

$$Q_{\max} = \dot{m}c_{p_{\min}}(T_{h_{inlet}} - T_{c_{inlet}}) \tag{4.14}$$

The heat exchanger effectiveness can then be defined as:

$$e = \frac{\dot{m}c_p^h(T_{h_{inlet}} - T_{h_{outlet}})}{\dot{m}c_{p_{\min}}(T_{h_{inlet}} - T_{c_{inlet}})} \text{ or } \quad e = \frac{\dot{m}c_p^c(T_{c_{inlet}} - T_{c_{outlet}})}{\dot{m}c_{p_{\min}}(T_{h_{inlet}} - T_{c_{inlet}})} \tag{4.15}$$

The effectiveness can be written as a function of Number of Thermal Units (NTU), $\dot{m}c_{p_{\min}}$, and $\dot{m}c_{p_{\max}}$.

$$NTU \equiv \frac{UA}{\dot{m}c_{p_{\min}}} \tag{4.16}$$

The overall heat transfer coefficients are calculated by summing the thermal resistances:

$$UA = \frac{1}{R_1 + R_{f1} + R_w + R_{f2} + R_2} \tag{4.17}$$

where:

U = Overall heat transfer coefficient

A = Heat exchanger surface area

R_1 = Resistance due to convection for fluid 1

R_{f1} = Fouling resistance for fluid 1

R_w = Thermal resistance due to wall

R_{f2} = Fouling resistance for fluid 2

R_2 = Resistance due to convection for fluid 2

R_1, R_w, and R_2 are determined from fluid properties at a given flow and temperature and from the heat exchanger material properties. R_{f1}, and R_{f2}, are usually taken as constants from based on historical data. This technique is described in detail in (Schmidt et al. 1993).

Ideally, the SIMULINK model would be compared to actual operating data for the ANPM. However, actual operating data is not available for this heat exchanger system, so the SIMULINK model was used to create two datasets: a low fidelity model simulation and a high fidelity model simulation. The difference in the two datasets is the number of nodes used when running the model. More nodes are analogous to a finer mesh and therefore should yield slightly more accurate outlet temperatures when compared to a dataset with fewer nodes.

The final inputs to the SIMULINK model are: cold leg inlet flow, hot leg inlet flow, cold leg inlet temperature, hot leg inlet temperature, the number of nodes used, R_{f1} and R_{f2}, and the number of tubes plugged. R_{f1} and R_{f2}, and the number of tubes plugged are included as input parameters in anticipation of creating faulted data sets representing tube fouling or plugging. The outputs of the model are the cold leg outlet temperature and the hot leg outlet temperature.

The simple heat exchanger model described above was used to create two data sets. The low fidelity data was used to populate the original ANPM memory matrix, and the high fidelity data is used as the actual system data to run and update the ANPM. Fig. 49 below shows the differences between the FPM data and the heat exchanger data for the first three sensors. There is a discrepancy between the two data sets, which the ANPM will compensate for by adapting the model to fit the high fidelity heat exchanger data. Fig. 50 shows the residuals of sensor 2 obtained from three different models; the original model built on FPM data, the ANPM residuals obtained while the model was adapting, and the residuals form the final model built on actual system data. This shows that the ANPM model residuals, which are in blue, slowly approach zero while the FPM based model does a poor job of predicting sensor 2 during the entire phase 1 operation. Table 4 shows the mean squared error (MSE) results for the heat exchanger data. The ANPM predictions are much better than the FPM prediction, which is evident by the mean square error. The final model has the best results, which is expected but is not useful in prediction during phase 1 operation.

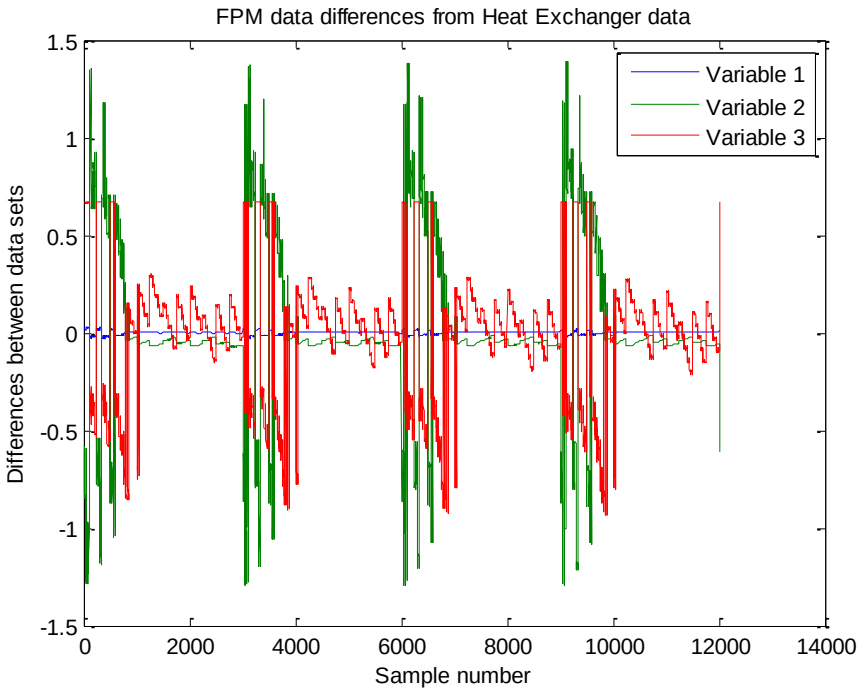


Figure 49: Differences between data sets

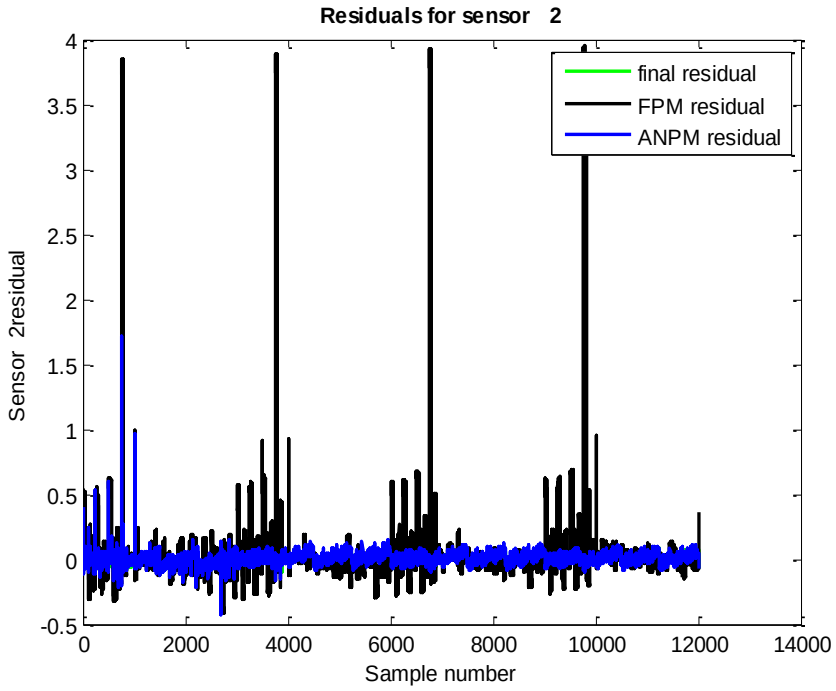


Figure 50: Residuals for Variable 2

Table 4: MSE results for heat exchanger model

Predictions	sensor 1	sensor 2	sensor 3	sensor 4	sensor 5	sensor 6
FPM predictions	0.0104	0.0268	0.428	0.0308	0.1132	0.2382
ANPM predictions	0.004	0.005	0.003	0.005	0.006	0.008
final model	0.0009	0.0022	0.0004	0.0016	0.002	0.0022

2.3 ADAPTIVE NONPARAMETRIC MODEL RESULTS ON THE FLOW LOOP

There were two non-faulted profiles created using the flow loop and the SIMULINK model. The actual flow loop data and the SIMULINK model data are shown in Fig 51 for the first tank level profile. The SIMULINK flow loop model does a good job of predicting the flow loop operating conditions. The residuals between the actual flow loop data and the SIMULINK simulation are shown in Fig. 52. This magnifies the differences between the FPM data and the actual flow loop data.

All the sensors were kept in the model. The predictions from the FPM based AAKR model and the ANPM for sensor 2, the actual tank level, are shown in Fig. 53 for profile 1. Fig. 54 shows the residuals for the FPM-based model, the ANPM, and the final system data-based model.

Table 5 gives the MSE for profile 1 for each of the three models. This shows that the accuracy of the predictions is improved by adapting the ANPM to use actual system data instead

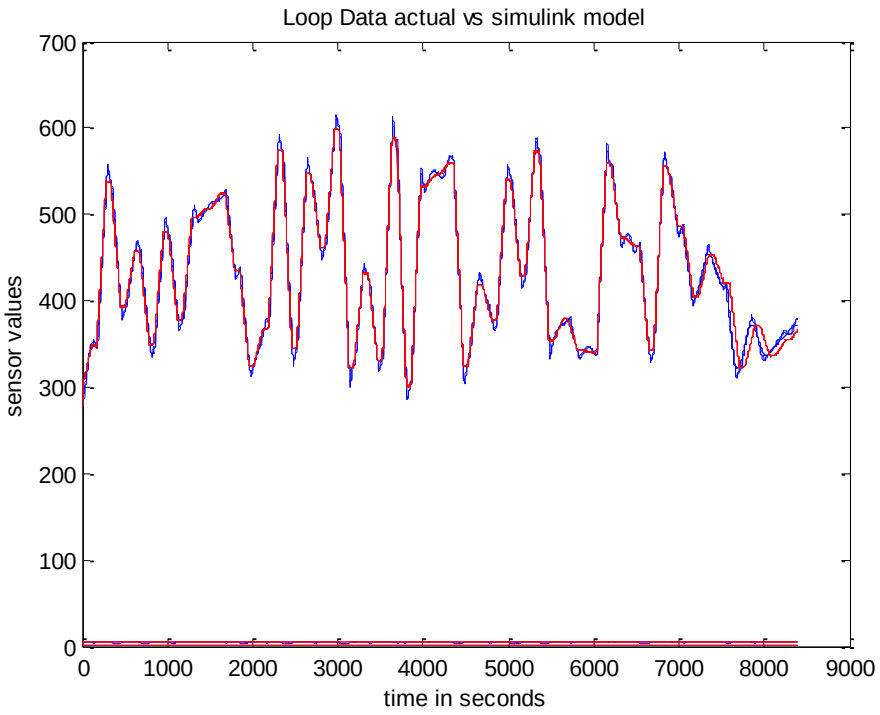


Figure 51: Flow loop data comparison profile 2

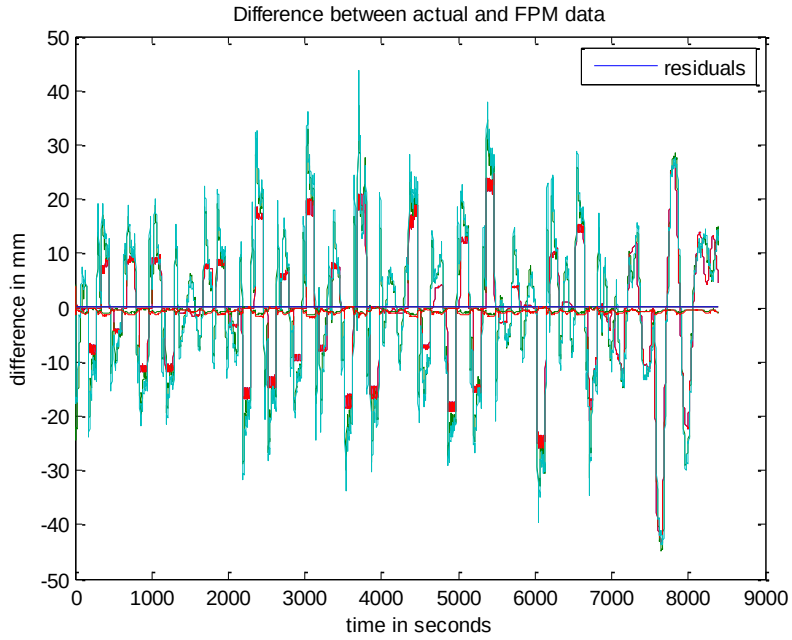


Figure 52: Residuals profile 1

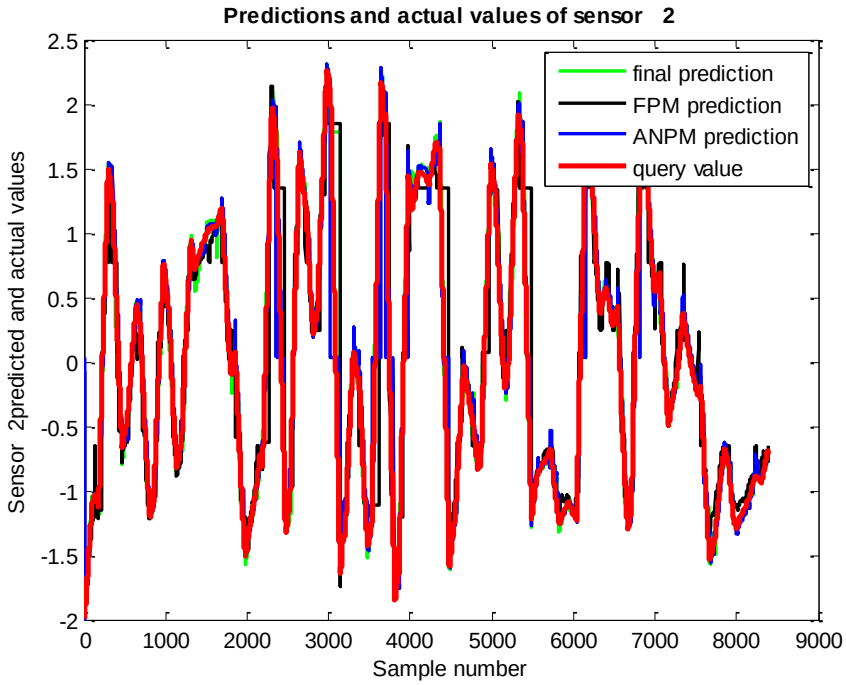


Figure 53: Predictions for sensor 2

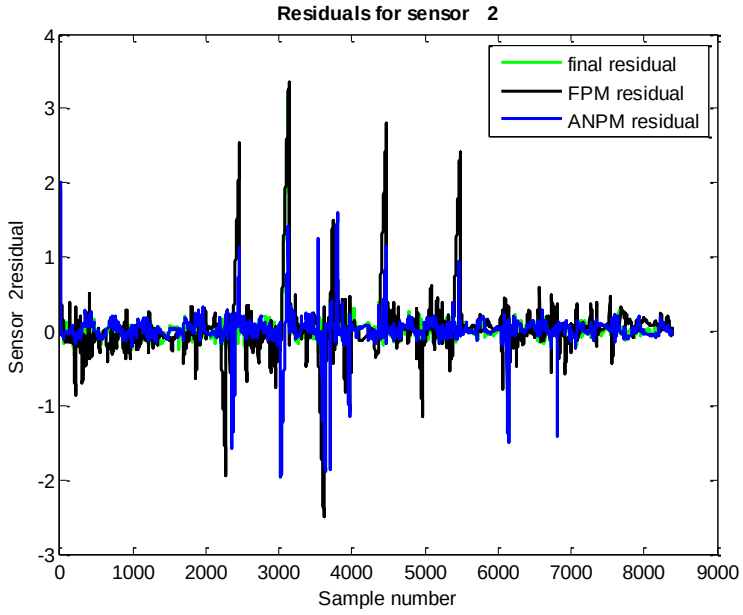


Figure 54: Residuals sensor 2

Table 5: MSE results for profile 1

Predictions	sensor 1	sensor 2	sensor 3	sensor 4	sensor 5	sensor 6	sensor 7	sensor 8	sensor 9	sensor 10
FPM predictions	0.218	0.2031	0.218	0.2004	0.5394	0.6878	0.1965	0.2145	0.5105	0.6411
ANPM predictions	0.0432	0.0434	0.0432	0.0438	0.2193	0.2305	0.0447	0.0467	0.2225	0.2427
Final predictions	0.0458	0.0511	0.0458	0.0546	0.0513	0.0458	0.0536	0.0642	0.0801	0.1057

of FPM data. The MSE decreases for each sensor, showing that the ANPM has increased the accuracy of the model by adapting it from the FPM to the actual data. Similar results are seen for profile 2.

The ANPM has been shown to adapt a nonparametric model from a FPM base to a base of the actual system data. Two examples have been shown where no faults are included in the system and the ANPM increases the overall performance of the model. This was shown for the heat exchanger data sets and the flow loop data sets. The ANPM increased the accuracy of the monitoring model on the heat exchanger model from an average MSE of 0.1451 to 0.0028, and on the flow loop data from an average MSE of 0.302 to an MSE of 0.013 over the adaptation phase of operation. Testing the ANPM on a faulty system shows the ability of the model to detect faults while adapting.

The 60mm drift failure that was shown in Fig. 27 was used to test the ANPM's ability to detect faults while adapting. This integrates the ANPM adaptation with the PCA ECM technique to give a complete monitoring and fault detection capability. The predictions, residuals, and fault hypothesis is shown for the first two sensors where this failure is evident. Fig. 55 shows the query values with the predictions from the final model, FPM model, and ANPM model for

sensor 1. Fig. 56 has the residuals for the final model, FPM model, and the ANPM model. The ANPM residual follows the same path as the FPM residuals indicating a deviation from the nominal conditions. This highlights that the ANPM has the same ability as the FPM at detecting the 60mm drift fault which can be seen in the residual for sensor 1.

The fault detection hypothesis can be seen in Fig. 57. The fault is first detected breifly around the 2700 time step and then detected around the 4000 time step and beyond. The ANPM uses the fault hypothesis to govern the adaptive ability, so after a fault is detected the model stops adaptation until normal operation begins. The fault hypothesis is for the complete system the remaining sensors share the same fault hypothesis.

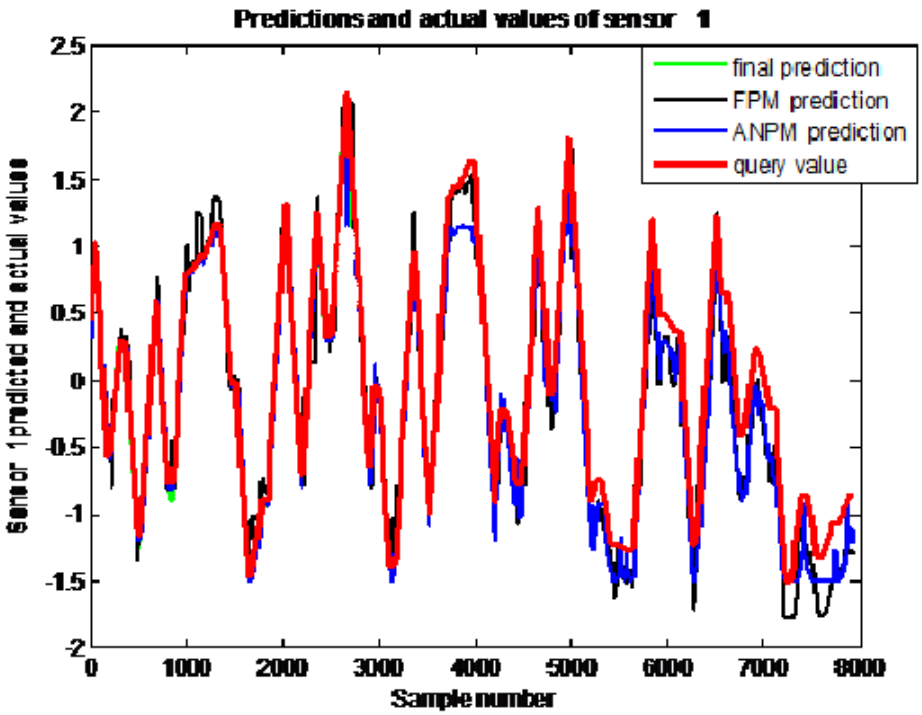


Figure 55: ANPM sensor 1 predictions for the 60mm drift failure

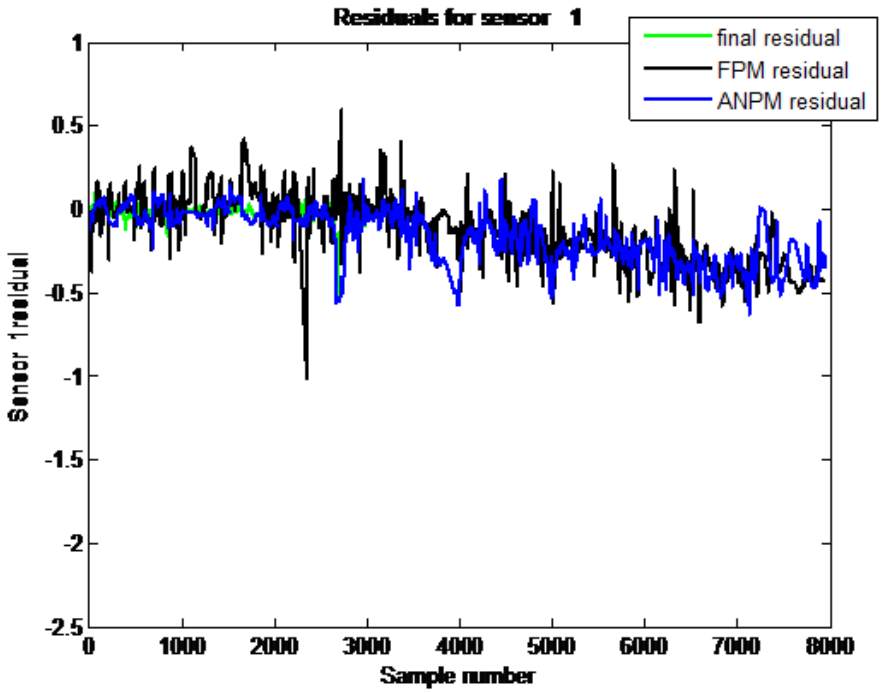


Figure 56: Residuals for sensor 1

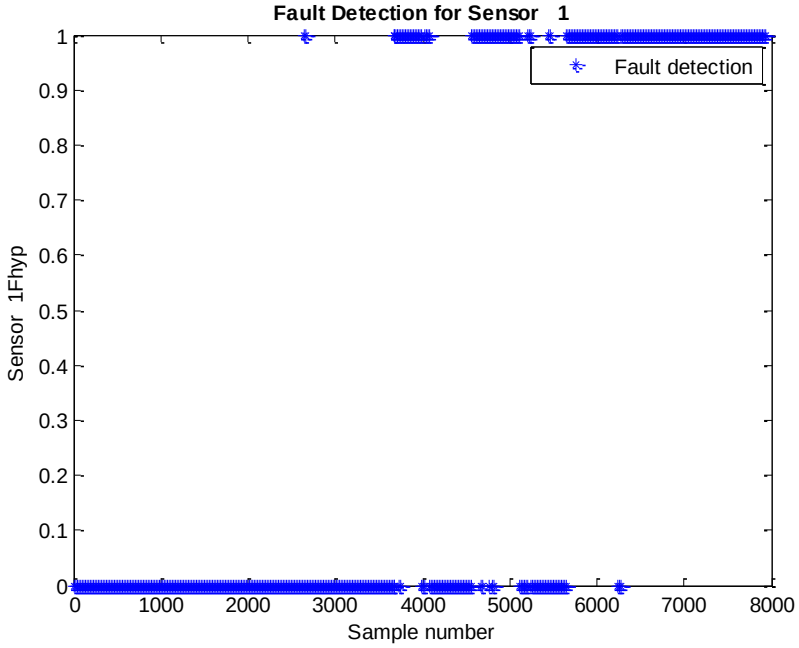


Figure 57: Fault hypothesis

Sensor 2 predictions have a similar result to sensor 1 predictions and can be seen in Fig. 58. The predictions follow a similar path as the FPM predictions which then can be seen in the residuals that are shown in Fig. 59. These residuals show a deviation from the nominal conditions and the fault hypothesis shows the ANPM has detected this abnormality.

The 60mm drift fault was correlty detected and the use of the ANPM on the faulty flow loop data was shown to have accurate results. Both sensors 1 and 2 were shown to have residuals that deviated from the nominal conditions indicating a fault.

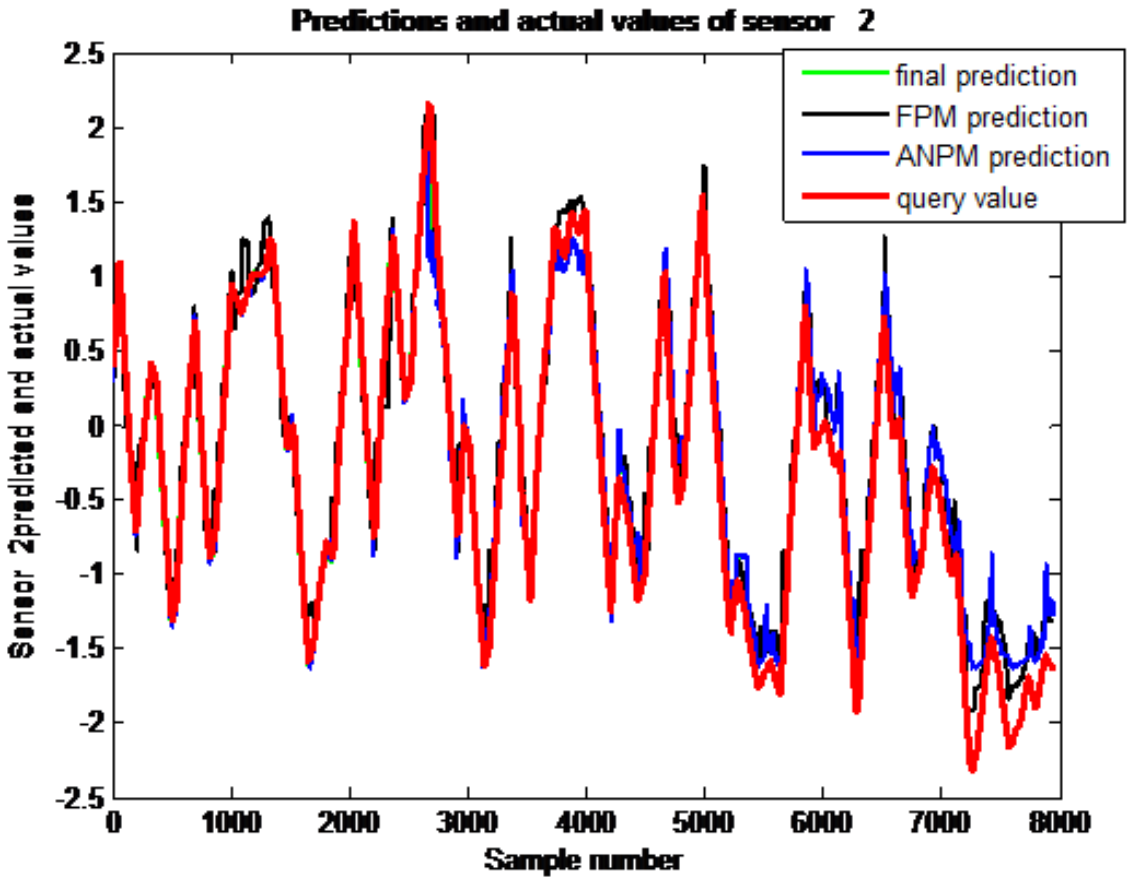


Figure 58: ANPM sensor 2 predictions for the 60mm drift failure

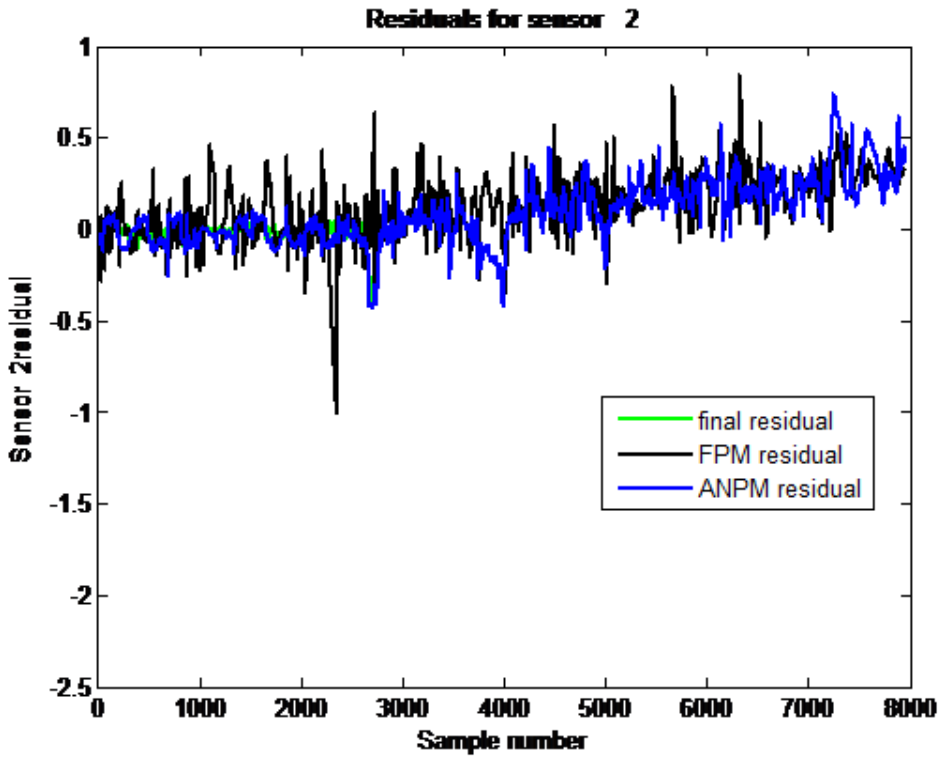


Figure 59: Residuals for sensor 2

2.4 ELIMINATION AND SIMILARITY DIAGNOSTIC TECHNIQUE

A data set was created to test the Elimination and Similarity (ES) diagnostic technique. There are 10 residuals with a mean of zero and different variances which represent a nominal condition. Below in Fig. 60 are the non-faulty residuals, which is normally distributed noise with a mean of zero. This is expected when a good monitoring model is used for prediction. The deviation from nominal conditions of the variance or mean of the residual shows some abnormality in the system.

A historical residual memory matrix of failures was created with fifteen possible failures. These failures map to a corresponding fault signature vector. The historical residual memory matrix consists of ten observations for each failure.

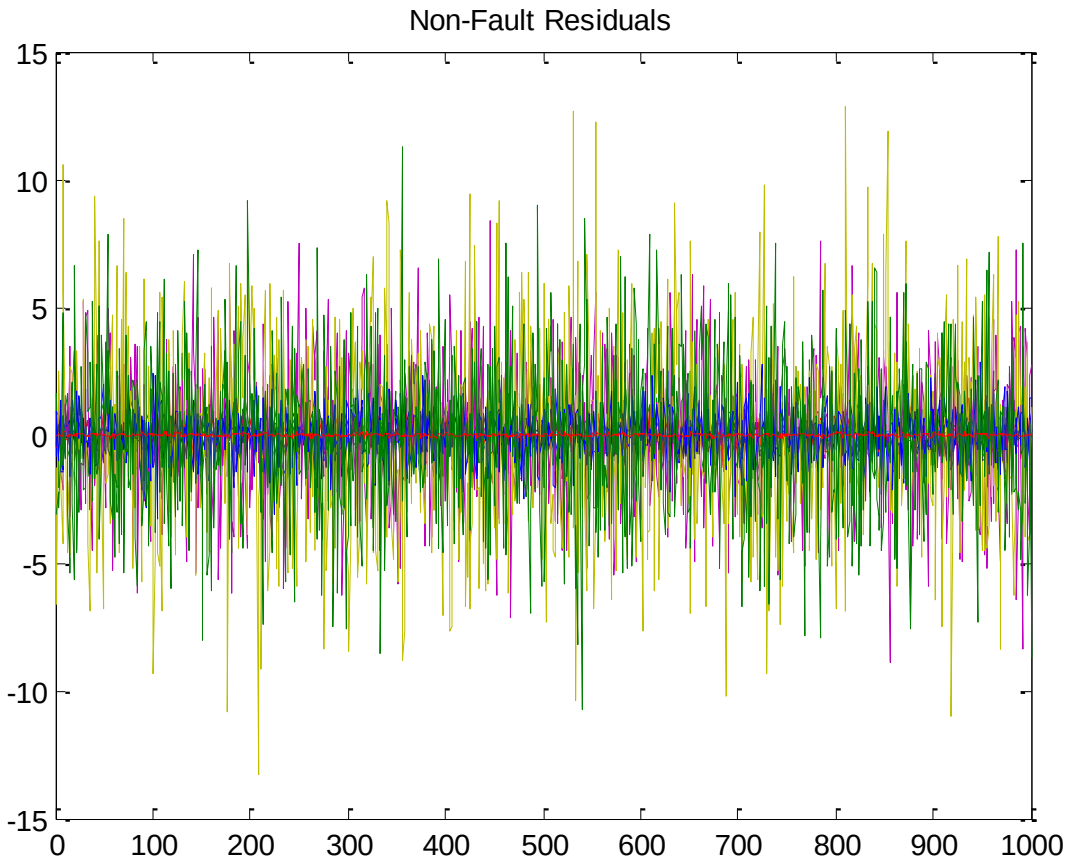


Figure 60: Non-faulty residuals

Table 6: Fault Signature 1

Sensors and corresponding fault signatures									
1	2	3	4	5	6	7	8	9	10
1	0	0	1	0	0	0	0	0	0

To test the complete capability of the ES diagnostic technique a number of different failures need to have the same fault signature vector. The first two tested failures have the same fault signature vectors which can be seen in table 6. This fault signature vector shows that a positive drift or bias was found in sensors 1 and 4. All the other sensors had no detected deviations from nominal conditions. However, for this example, two failures were created that have this fault signature vector, failure **a** and failure **b**. These residuals for sensors 1 and 4 can be seen in Fig. 61.

The ES diagnostic technique outputs the fault signature vector followed by the failures that have that fault signature vector. Then the final output is the similarity parameters which has lower values for failures that are more similar to the observed failure. Testing the ES diagnostic technique on the actual failure **a** gives the following results:

Fault signature vector = 1 0 0 1 0 0 0 0 0 0

Possible Failures = 2 13

Failure Similarities = 115.5691 132.1960

For this specific fault signature there are two possible failures, failure 2 and 13. The other 13 possible failures have been eliminated from the possible failure list. A similarity comparison is conducted and failure 2 has the lowest similarity parameter of 115.57 which shows this failure is most similar to the observed failure. Where actual failure **a** is failure 2 so this failure was correctly classified. The classification process for this example can be seen in Fig. 62.

Testing the ES diagnostic technique on the actual failure **b** gives the following results:

Fault signature vector =1 0 0 1 0 0 0 0 0 0

Possible Failures = 2 13

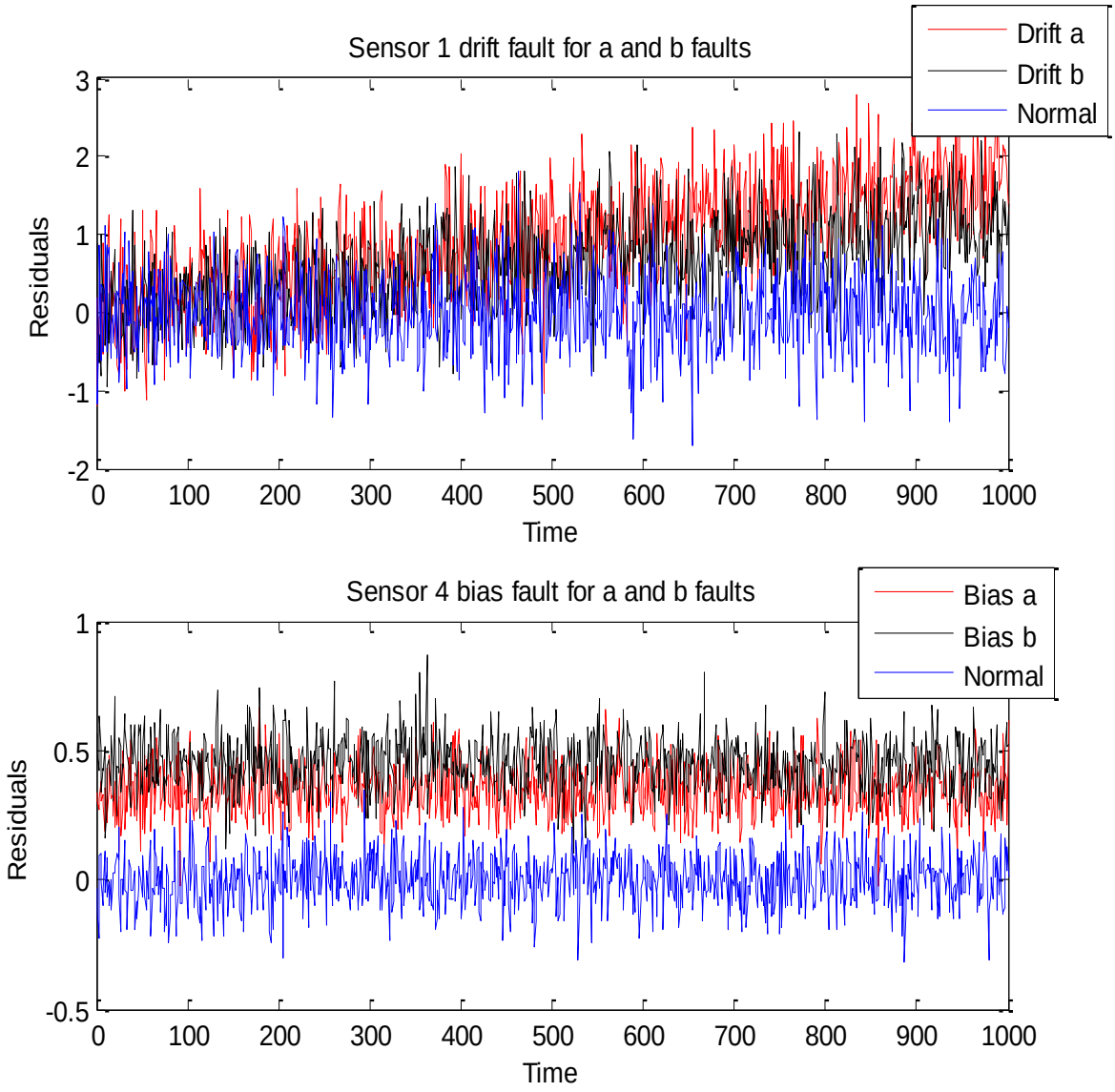


Figure 61: System failures

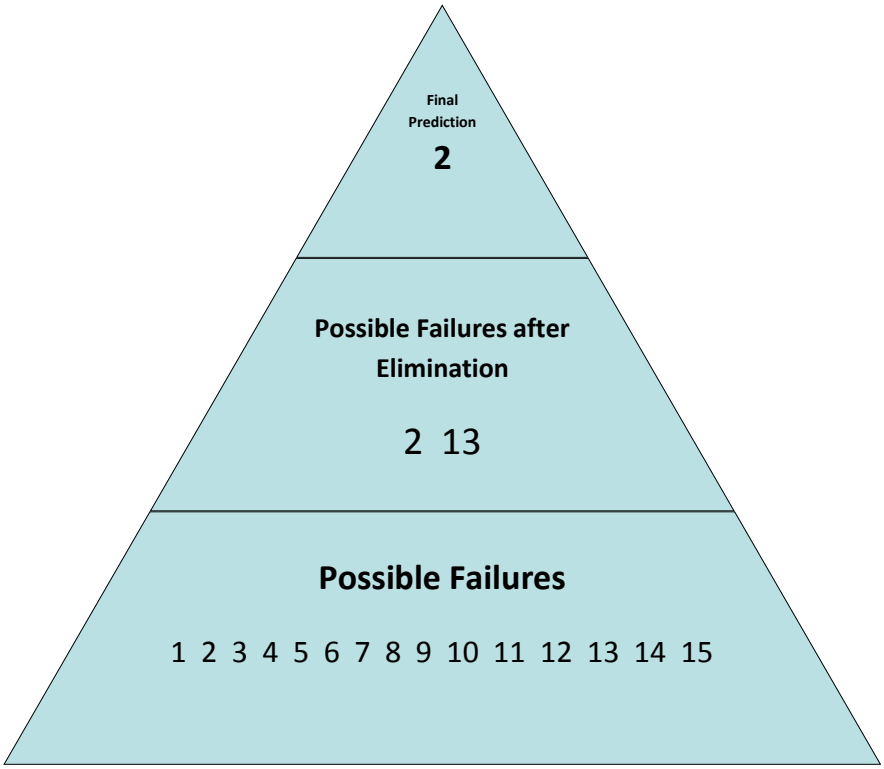


Figure 62: Failure Elimination Process

Failure Similarities =140.4501 122.4915

For this specific fault signature there are the same two possible failures, failure 2 and 13. The other 13 possible failures have been eliminated from the possible failure list. A similarity comparison is conducted and failure 13 has the lowest similarity parameter of 122.49 which shows this failure is most similar to the observed failure. Where actual failure **b** is failure 13 so this failure was correctly classified.

Three more system failures with the same fault signatures were created to test the ES diagnostic technique, the fault signature vector that these faults can be seen below in table 7.This fault signature vector has noise disturbance in sensor 7 and a positive drift or bias in sensors 3 and 5. All the other sensors had no detected deviations from nominal conditions. However, for this example, three failures were created that have this fault signature vector, failure **a,** failure **b** and failure **c**. The residuals for sensors 3, 5 and 7 can be seen in Fig 63.

Table 7: Fault signature 2

Sensors and corresponding faults									
1	2	3	4	5	6	7	8	9	10
0	0	1	0	1	0	3	0	0	0

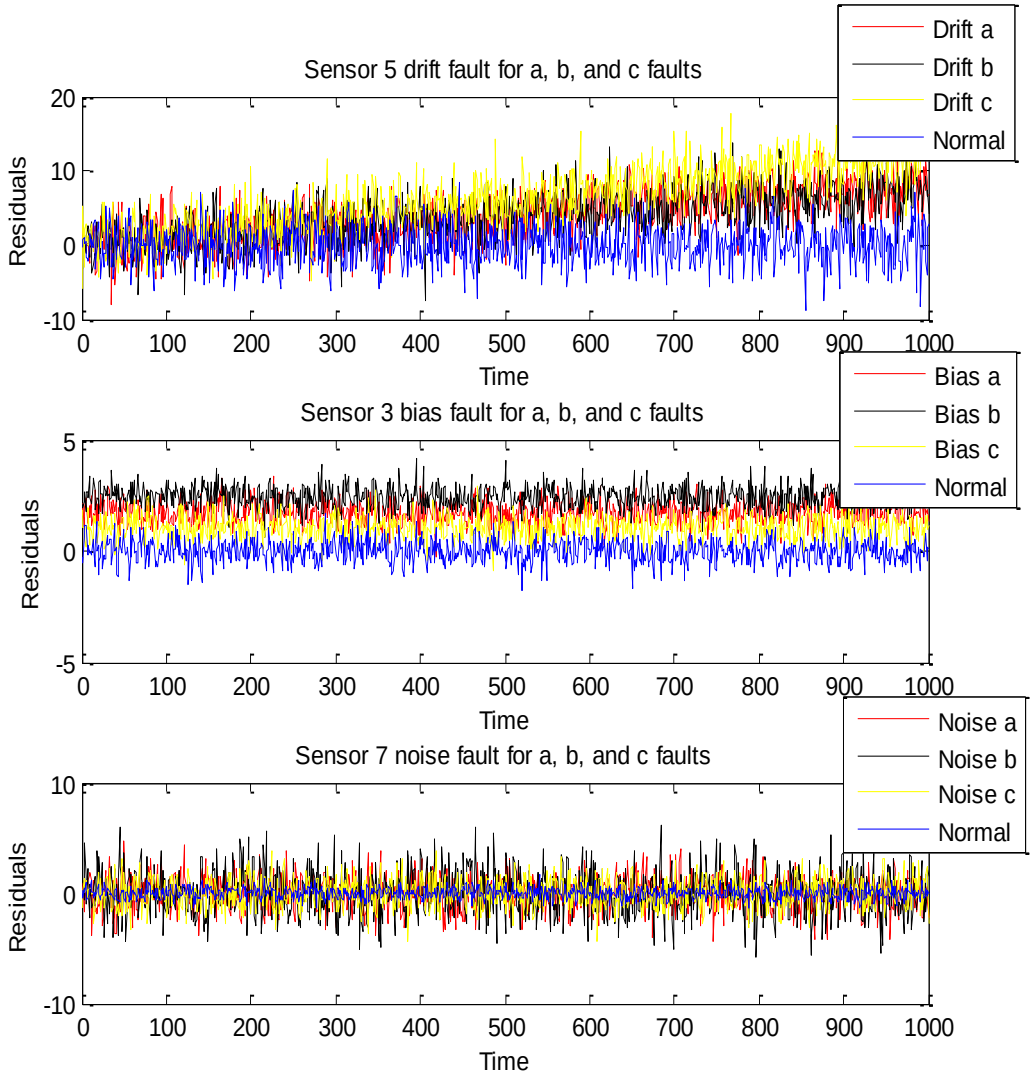


Figure 63: Three failures

Applying the ES diagnostic technique to the actual failure **a** gives the following results:

Fault signature vector= 0 0 1 0 1 0 3 0 0 0

Possible Failures = 5 11 15

Failure Similarities =119.9847 171.9808 137.0913

For this specific fault signature there are the three possible failures, failure 5, failure 11 and failure 13. The other 12 possible failures have been eliminated from the possible failure list. A similarity comparison is conducted and failure 5 has the lowest similarity parameter of 119.98 which shows this failure is most similar to the observed failure. The actual failure **a** is failure 5 so this failure was correctly classified. The classification process for this example can be seen in Fig. 64.

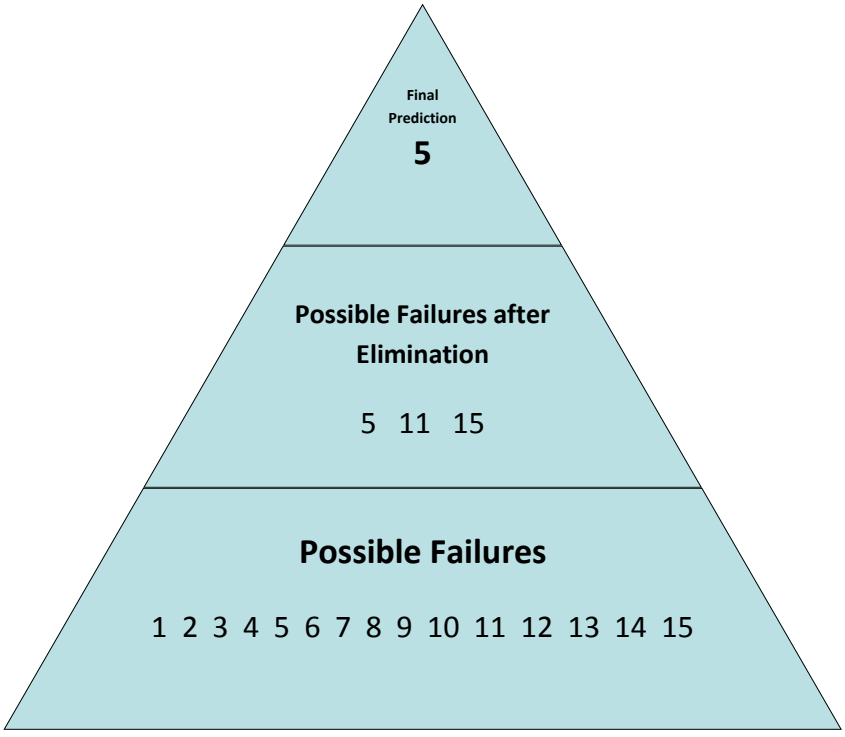


Figure 64: Failure Elimination Process 2

The ES diagnostic technique results for failure **b** are:

Fault signature vector= 0 0 1 0 1 0 3 0 0 0

Possible Failures = 5 11 15

Failure Similarities = 197.7658 158.9452 168.6357

There are three possible failures, failure 5, failure 11 and failure 13. A similarity comparison is conducted and failure 11 has the lowest similarity parameter of 158.95 which shows this failure is most similar to the observed failure. The actual failure **b** is failure 11 so this failure was correctly classified.

The ES diagnostic technique results for failure **c** are:

Fault signature vector= 0 0 1 0 1 0 3 0 0 0

Included =5 11 15

Failure Similarities = 135.1732 152.8051 121.5484

There are three possible failures, failure 5, failure 11 and failure 13. A similarity comparison is conducted and failure 15 has the lowest similarity parameter of 121.55 which shows this failure is most similar to the observed failure. The actual failure **c** is failure 15 so this failure was correctly classified.

A fault diagnostic module was shown that can be implemented into a complete equipment surveillance system which will be capable of determining the most likely fault given the faulty residuals. A two step process which consists of fault signature elimination and a weighted path similarity are used to give the best identification. This technique is referred to as

the elimination and similarity (ES) diagnostic technique. A detailed description of both steps was provided. Starting with faulty residuals and ending with a most likely fault. A couple of example problems show the diagnostic capabilities of the complete ES diagnostic system.

Results were shown to highlight the use of the proposed methods on actual data sets and to prove the proposed concepts. Results were shown for the application of the ANPM to non-faulty data sets and faulty data sets. This highlighted the improved accuracy of the adaptive model when applied to systems but emphasized the fault detection capability. The two step ES diagnostic technique was shown to correctly identify the faults. The next chapter is the conclusion which summarizes the research and describes the original contributions and the research done that corresponds accordingly. Then a suggested future work section follows with a description of key areas of future work in the adaptive modeling research.

3 CONCLUSIONS AND SUGGESTIONS FOR FUTURE WORK

A detailed study of an adaptive modeling technique was given, with an emphasis on the development of the ANPM approach. The need for such a modeling technique was discussed in regard to condition monitoring within the nuclear power industry and the application of new dynamic modeling structures for next generation reactors. Despite the origin for this research there are numerous areas of application for adaptive modeling capabilities. A complete literature review was given on the four main areas of a complete surveillance system, monitoring, fault detection, diagnostics, and prognostics. Testing of the proposed ANPM with an expanded condition monitoring capability was given on two major system data sets. The goals of this research have been accomplished through the five original contributions and a number of other non-original contributions.

3.1 SUMMARY AND ORIGINAL CONTRIBUTIONS

There are five areas of original contribution that were discussed. Each of these original contributions were covered with a thorough literature review and an indepth study of each research topic.

The first contribution is the development of an adaptive approach that can be used in a nonparametric modeling environment. This approach allows the model to evolve over time. This increases the predictive capability of the model. Included in the ANPM is a sequence of optimum vector addition techniques, FPM vector deletion techniques, evolving vector selection, and a combination of fault detection and expanded condition monitoring techniques. These techniques

were discussed in depth with several applications shown. This adaptive modeling technique was tested on two systems. The first system is a heat exchanger model that was simulated in both a low and high fidelity way in SIMULINK. The second system is a flow loop that is built at the University of Tennessee and simulated in SIMULINK. The ANPM architecture is used to adapt the model's memory matrix from data simulated by an FPM to that collected from actual system operation. The ANPM increased the accuracy of the monitoring model on the heat exchanger model from an average MSE of 0.1451 to 0.0028, and on the flow loop data from an average MSE of 0.302 to an MSE of 0.013 over the adaptation phase of operation. Both tests highlighted the benefits of the adaptive modeling technique by increasing model performance and detecting faults during operation.

The second original contribution is the PCA ECM technique. This technique is used for the differentiation between expanded nominal conditions and an expanded fault condition. The PCA ECM technique is an integral part of the ANPM, it allows adaptation of the model beyond the initial range of data without losing fault detection capability.This dissertation partially focused on the basic principles behind PCA ECM and the demonstration of this technique on simulated data. The four different fault scenarios were discussed and their resulting conclusions. The PCA ECM technique was applied to a five variable data set with linear relationships. Three different fault types were tested, including drift faults, bias faults, and a stuck sensor fault. There were a total of six bias faults, six drift faults, and one stuck sensor fault. All these faults were on one of the five variables and the other variables were left unchanged. The results of the PCA ECM technique were the correct classification of the faults. This was shown by either fault scenario 3 or fault scenario 4. An expanded condition was shown to have a larger range than the normal condition. The application of the PCA ECM technique to the expanded condition data

resulted in the correct classification of an expanded condition. This PCA ECM technique was integrated into the ANPM as the expanded condition monitoring technique used in adaptation decisions.

The third original contribution is the integration of the ANPM into a complete monitoring, detection, and identification system. This integrates a dynamic modeling structure into a complete equipment surveillance system hence increasing the performance of the system. The basic structure of such a system was highlighted and discussed in detail. The monitoring, detection, and identification systems were tested and shown to accurately perform. Traditional prognostic techniques were discussed and their applicability into such a system was highlighted with an emphasis on different techniques that fit different systems.

The fourth original contribution is the development of a KPCA ECM technique for use on non-linear sensor relationships. This was investigated and discussed showing that KPCA is not capable for ECM applications. The nonlinear nature of KPCA made expanded condition monitoring unsuccessful. A discussion on KPCA was given with the testing of such a technique with different kernel selections. Such a technique was shown to be unreliable and not useful in an adaptive environment.

The fifth and final original contribution is the development of MATLAB based software that implements the ANPM and ECM technique. This contribution was highlighted through the use of the software to test and complete the results given. This MATLAB based software was used in the operaton of the ANPM and ECM techniques.

This dissertation gave an overview of the development of the ANPM. The focus was on the method used to create a model that can operate over a phase of adaptation. The results for testing

the ANPM on non-faulted conditions for the heat exchanger model and the flow loop were shown. The ANPM was tested on faulted data and shown to accurately detect faults while being able to differentiate expanded nominal conditions. All five of the original contributions were completed and discussed in detail througout the dissertation. These contributions are the results of in depth research into adaptive modeling and the techniques needed to accomplish an accurate result. This research opens the door to a plethora of other research opportunities in the adaptive modeling arena.

3.2 FUTURE WORK

A complete study of adaptive modeling was given, which resulted in the creation of the ANPM. This dissertation covered the development of the ANPM with a focus on the method used to create a model that can operate over a phase of adaptation. The need for such a modeling technique was discussed in detail with an emphasis on applications within the nuclear power industry. All five of the original contributions were completed and discussed in detail througout the dissertation. This research opens the door to a plethora of other research opportunities in the adaptive modeling arena. A few of these opportunities are discussed below.

1. Traditionally residuals are used within a number of fault detection and diagnostic algorithms. Adaptive models provide the ability to have two separate residuals, the residuals from the FPM, and the residuals from the adaptive model. An area that has not been investigated for use in fault detection and diagnostics is the difference between the adaptive model predictions and the FPM predictions. This would create a third type of residual that would be helpful in distinguishing the differences between the adaptive

model and the FPM. These new residuals could be usefule in a traditional fault detection technique such as the SPRT, or some new type of fault detection technique. This could also be used as to improve fault detection and diagnostic capabilities with providing a better understanding of the relationships behind the different modeling techniques.

2. The development of other non-linear ECM techniques to solve the difficulties that PCA ECM has with non-linear relationships. Discovering a way to either transform the data so the PCA ECM technique will work properly or a different type of non-linear PCA that could solve the issues that the kernel PCA had with the expanded condition monitoring.

3. Another opportunity lies with the optimizing of the ANPM for case specific applications, with regards to the vector addition technique. The proposed ANPM was shown to perform well for a number of different systems. Two vector addition techniques were discussed in this dissertation, the continuous addition technique, and the step addition technique. Most of the research focused on the continuous addition technique; however, there is promise in the step addition technique when tied to different fault detection techniques such as the SPRT. As more specific cases are investigated, the optimization of the ANPM by investigating a number of vector addition techniques and the ability of these techniques to perform for these specific cases.

There are numerous areas of research that could come from the adaptive modeling environment and the optimization and application of these techniques to specific systems. Three areas were briefly discusssed for future work; however, there are many other ideas that could be drawn from a similar basis.

REFERENCES

Abernethy, R. B. (1996). The New Weibull Handbook, 2nd edn. ISBN 0 9653062 0 8. Abernethy, North Palm Beach, 1996.

Alessandri, A., & Parisini, T. (1997). Non-Linear Modeling of Complex Large –Scale Plants Using Neural Networks and Stochastic-Approximation. *IEEE Transactions on Systems, Man and Cybernetics, Part A-Systems and Humans*, 6, 750-757.

Atkeson, C. G., Moore, A. W., & Schaal, S. (1997a). Locally Weighted Learning. *Artificial Intelligence Review*, 11, 11-73.

Atkeson, C. G., Moore, A. W., & Schaal, S. (1997b). Locally Weighted Learning for Control. *Artificial Intelligence Review*, 11, 75-113.

Azzam, H. (1997). A Practical Approach for the Indirect Prediction of Structural Fatigue from Measured Flight Parameters. *Journal of Aerospace Engineering* ,211, 29 – 38.

Bakshi, B.R. (1998). Multiscale PCA with application to multivariate statistical process monitoring. *AIChE J.*, 44, 1596-1610.

Basir, O. A. (2004). Vibro-Acoustic Engine Diagnostic System. *U.S. Patent Application 20040260454*, Carlson, Gaskey & Olds, P.C., Birmingham, MI: June 14.

Ben-Abdennour, Adel, & Lee, K. Y. (1996). An Autonomous Control System For Boiler-Turbine Units. *IEEE Transactions on Energy Conversion*, 11, 401-406.

Bickel, P., J. & Doksum, K. A. (2001). *Mathematical Statistics: Basic and Selected Topics, Volume 1*, Pearson Prentice-Hall.

Bogdanoff, J.L. & Kozin, F. (1985). *Probabilistic Models of Cumulative Damage*. John Wiley and Sons, New York.

Breed, D. S. (2002). Telematics System for Vehicle Diagnostics. *U.S. Patent 6,738,697*, Automotive Technologies International, Inc., Denville, NJ: July 3.

Breed, D. S. (2005). Vehicular Information and Monitoring System and Methods. *U.S. Patent Application 20050125117*, Brian Roffe, ESQ, Valley Stream, NY: January 19.

Brown, D.W., Kalgren, P.W., Byington, C.S., & Roemer, M.J. (2007). Electronic Prognostics – A Case Study using Global Positioning System (GPS). *Microelectronics Reliability*, 47, 1874 – 1881.

Byington, C.S., Watson,M., Edwards, D., & Stoelting, P. (2004). A Model-Based Approach to Prognostics and Health Management for Flight Control Actuators. *2004 IEEE Aerospace Conference Proceedings*. 3551 – 3562.

Carden, E.P. & Fanning, P. (2004). Vibration Based Condition Monitoring: A Review. *Structural Health Monitoring*, 4, 355 – 377.

Catbas, F.N. & Atkan, A.E. (2002). Condition and Damage Assessment: Issues and Some Promising Indices. *Journal of Structural Engineering*, 8, 1026 – 1036.

Chinnam, R.B. (1999). On-line Reliability Estimation of Individual Components, Using Degradation Signals. *IEEE Transactions on Reliability*, 4, 403 – 412.

Cho, J.H., Lee, J.M., Choi, S.W., Lee, D., & Lee, I.B. (2005). Fault identification for process monitoring using kernel principal component analysis. *Chemical engineering science*, 60, 279-288.

Choi, S. W., & Lee, I.B. (2004). Nonlinear dynamic process monitoring based on dynamic KPCA. *Chemical Engineering Science*, 59, 5897-5908.

Choi, S. W., Lee, C., Lee, J.M.,Park, J.H., & Lee, I.B. (2005). Fault detection and identification of nonlinear processes based on KPCA. *Chemometrics and Intelligent Laboratory Systems*, 75, 55-67.

Christianini, N. & Shawe-Taylor, J. (2000). *An Introduction to Support Vector Machines and Other Kernel-Based Learning Methods*. Cambridge University Press, UK.

Cleveland, W. S. & Loader, C. (1994a).Computational methods for local regression. *Technical Report 11*, AT&T Bell Laboratories, Statistics Department, Murray Hill, NJ.

Cleveland, W. S. & Loader, C. (1994b). Smoothing by local regression: Principles and methods. *Technical Report 95.3*, AT&T Bell Laboratories, Statistics Department, Murray Hill, NJ.

Cover, T. M. & Hart, P. E. (1967). Nearest Neighbor Pattern Classification. *IEEE Transactions on Information Theory*, 13, 1.

Cox, D.R. & Oakes, D. (1984). *Analysis of Survival Data*. Chapman and Hall.

Cui, P., Li, J., & Wang, G. (2008). Improved kernel principal component analysis for fault detection. *Expert system with application*, 34, 1210-1219.

Davidson, A.C. (2003). *Statistical Models*. Cambridge University Press.

Del Amo, A., Keller, K., & Swearingen, K. (2005). General Reasoning System for Health Management. *Proceedings of the Annual Meeting of the North American Fuzzy Information Processing Society*, 19-24: June 26-28.

Dong, D. & McAvoy, T. J. (1996). Nonlinear principal component analysis based on principal curves and neural networks. *Computers and Chemical Engineering*, 20, 65-78.

Dong, M., Xu, D. K., Li, M. H., & Yan, X. (2004). Fault Diagnosis Model for Power Transformer Based on Statistical Learning Theory and Dissolved Gas Analysis. *Proceedings of the IEEE International Symposium on Electrical Insulation*, 85-88, Indianapolis, IN: September 19-22.

Draper, N. R., & Smith, H. (1966). *Applied Regression Analysis*. John Wiley & Sons, New York.

Dunia, R., Qin, S. J., Edgar T. F., & McAvoy, T.J. (1996). Identification of faulty sensors using principal component analysis. *AIChE Journal*, 42, 2797-2812

Dunia, R., & Qin, S. J. (1998). Joint Diagnosis of Process and Sensor Faults Using Principal Component Analysis. *Control Engineering Practice*, 6, 457-469.

Ebeling, C. E. (2005). *An Introduction to Reliability and Maintainability Engineering*. Waveland Press, Inc., Long Grove, IL.

Engel, S., Gilmartin, B., Bongort, K., & Hess, A. (2000). Prognostics, the Real Issues Involved with Predicting Life Remaining. *Proceedings of the IEEE Aerospace Conference*, 457-469.

Esary, J.D. & Marshall, A.W. (1973). Shock Models and Wear Processes. *The Annals of Probability*, 4, 627 – 649.

Fan, J., & Gijbels, I. (1995). Data-driven bandwidth selection in local polynomial fitting: Variable bandwidth and spatial adaptation. *J. Roy. Statist. Soc.*, 57, 371-394.

Fan, J., & Gijbels, I. (1996a). *Local Polynomial Modeling and Its Applications*. Chapman & Hall/CRC, New York, NY.

Fan, J., Gijbels, I., Hu, T-C., & Huang, L.(1996b). A study of variable bandwidth selection for local polynomial regression. *Statistica Sinica.*, 6, 113-127.

Fenu, G. & Parisini, F. (1998). Model-Free Fault Diagnosis for Nonlinear Systems: A Combined Kernel-Regression and Neural Networks Approach. *Proceedings of the American Control Conference*, Philadelphia, PA.

Ferrell, B.L. (1999). JSF Prognostics and Health Management. *IEEE Aerospace Conference*, 471.

Ferrell, B.L. (2000). Air Vehicle Prognostics and Health Management. *IEEE Aerospace Conference*, 145 – 146.

Fourie, S.H. & Vaal, P.de. (2000). Advanced process monitoring using an on-line non-linear multiscale principal component analysis methodology. *Computers and Chemical Engineering,* 24, 755-760.

Frank, P.M. (1990). Fault diagnosis in dynamic systems using analytical and knowledge base redundancy–A survey and some new results. *Automatica*, 26, 459–474.

Freedman, D.A. (2009). Statistical Models: Theory and Practice, Second Edition. *Cambridge University Press*.

Garvey, D. & Hines, J.W. (2006). An Adaptive Distance Measure for Use with Nonparametric Models. *5th International Topical Meeting on Nuclear Plant Instrumentation, Control and Human-Machine Interface Technologies,* Albuquerque, NM: November 12-14.

Garvey, D., & Hines, J.W. (2006). Development and Application of Fault Detectability Performance Metrics for Instrument Calibration Verification and Anomaly Detection. *Journal of Pattern Recognition Research*, 1, 2-15.

Garvey, D. (2007). An Adaptive Integrated Fuzzy Inference Based Monitoring, Diagnostic, and Prognostic System. *Dissertation*, Nuclear Engineering Department, University of Tennessee.

Gayme, D. F., Menon, S. K., Nwadiogbu, E. O., Mukavetz, D. W., & Ball, C. M. (2003). Fault Detection System and Method Using Augmented Data and Fuzzy Logic. *U.S. Patent Application 20050021212*, Honeywell International Inc., Morristown, NJ: July 24.

Germond, A. J. & Niebur, D. (1992). Survey of Knowledge-Based Systems in Power Systems: Europe. *Proceedings of the IEEE*, 80, 5: May.

Gertler, J. (1988). Survey of model-based failure detection and isolation in complex plants. *IEEE Control Syst. Mag*, 8, 3–11.

Geva, S. & Sitte, J. (1991). Adaptive Nearest Neighbor Pattern Classification. *IEEE Transactions on Neural Networks*, 2, 318-322.

Gibbons, J. D. & Chakraborti, S. (2003). *Nonparametric Statistical Inference, 4th Edition*. CRC.

Gribok, A.V., Attieh, I., Hines, J.W., & Uhrig, R.E. (2001). Stochastic Regularization of Feedwater Flow Rate Evaluation for the Venturi Meter Fouling Problem in Nuclear Power Plants. *Inverse Problems in Engineering*, 00, 1-26.

Gut, A.(1990). Cumulative Shock Models. *Advances in Applied Probability*, 2, 504 – 507.

Haykin, Simon (1994). *Neural Networks: A Comprehensive Foundation*. 2nd Edition, Prentice Hall.

Heng, A., Zhang, S., Tan, A., & Mathew, J. (2009). Rotating Machinery Prognostics: State of the Art, Challenges, and Opportunities. *Mechanical Systems and Signal Processing,* 23, 724 – 739.

Heo, G.Y. (2008). Condition Monitoring Using Empirical Models: Technical Review and Prospects for Nuclear Applications. *Nuclear Engineering and Technology*, 1, 49 – 68.

Hess, A. & Fila, L. (2002). The Joint Strike Fighter (JSF) PHM Concept: Potential Impact on Aging Aircraft Problems. *IEEE Aerospace Conference,* 3021 – 3026.

Hines, J.W. & Garvey, D. (2005). The Development of a Process and Equipment Monitoring (PEM) Toolbox and its Application to Sensor Calibration Monitoring. *Fourth International Conference on Quality and Reliability (ICQR4)*. Beijing, China, August 9 - 11.

Hines, J.W. & Seibert, R. (2006a). Technical Review of On-Line Monitoring Techniques for Performance Assessment. *NUREG/CR-6895 ORNL/TM-188, Vol 1*

Hines, J. W. & Dustin, G. (2006b), Traditional and Robust Vector Selection Methods for Use with Similarity Based Models. *5th International Topical Meeting on Nuclear Plant Instrumentation, Control and Human-Machine Interface Technologies*, Albuquerque, NM: November 12-14.

Hines, J.W., Garvey, D., Seibert, R., & Usynin, A. (2007a). Technical Review of On-Line Monitoring Techniques for Performance Assessment, *NUREG/CR-6895 ORNL/TM-188, Vol 2*

Hines, J.W., Garvey, J., Garvey, D., & Seibert, R. (2007b). Technical Review of On-Line Monitoring Techniques for Performance Assessment, *NUREG/CR-6895 ORNL/TM-188,* Vol 3

Hines, J.W. (2006). Empirical Methods for Process and Equipment Condition Monitoring. *52nd Annual Reliability and Maintainability Symposium (RAMS),* Newport Beach CA, Jan 23-26.

Hines, J.W., Garvey, J., Preston, J., & Usynin, A. (2008a). Empirical Methods for Process and Equipment Condition Monitoring. *53rd Annual Reliability and Maintainability Symposium (RAMS)*, Las Vegas, NV, Jan.

Hines, J.W. (2008b). *Class notes for NE 671*, Empirical Modeling

Hines, J.W. & Usynin, A. (2008). Current Computational Trends in Equipment Prognostics. *International Journal of Computational Intelligence Systems,* 1, 1 – 9.

Humberstone, M., Wood, B., Henkel, J., & Hines, J.W. (2009). An adaptive model for expanded process monitoring. *Sixth American Nuclear Society International Topical Meeting on Nuclear Plant Instrumentation, Control, and Human-Machine Interface Technologies,* Knoxville, TN, April.

Humberstone, M., Wood, B., Henkel, J., & Hines, J.W. (2009). Differentiating between expanded and fault conditions using principal component analysis. *Journal of Intelligent Manufacturing*, 2009, 10-1007/s10845-009-0343-1

Isermann, R. (1984). Process Fault Detection Based on Modeling and Estimation Methods , A Survey. *Automatica*, 20, 387-404.

Isermann, R. (1995). Model Based Fault Detection and Diagnosis Methods. *Proceedings of the American Control Conference*, 1605-1609, Seattle, WA.

Isermann, R. (2004). Model-Based Fault Detection and Diagnosis, Status and Applications. *Proceedings of the 16th International Federation of Automatic Control (IFAC) Symposium on Automatic Control in Aerospace,* St. Petersburg, Russia: June 14-18.

Jackson, J. E. (1991). *A user's Guide to Principal Components*. Wiley, New York

Jackson, J.E. & Mudholkar, G.S. (1979). Control Procedures for Residuals Associated with Principal Component Analysis. *Technometrics,* 21, 341-349.

Johnson, R. A. & Wichern, D. W. (1992). *Applied Multivariate Statistical Analysis, 3rd Edition*. Prentice Hall, New Jersey.

Jolliffe, I. T. (2002). *Principal Component Analysis, Second Edition*. Springer Series in Statistics, New York, Springer.

Kacprzynski, G.J., Roemer, M.J., Modgil, G., Palladino, A., & Maynard, K. (2002). Enhancement of Physics-of-Failure Prognostic Models with System Level Features. *IEEE Aerospace Conference.*

Kacprzynski, G.J., Sarlashkar, A., Roemer, M.J., Hess, A., & Hardman, W. (2004). Predicting Remaining Life by Fusing the Physics of Failure Modeling with Diagnostis. *Journal of the Mineral, Metals, and Materials Society,* 3, 29 – 35.

Kalman, R.E. (1960). A new approach to linear filtering and prediction problems. *Journal of Basic Engineering,* 1, 35-45.

Kavuri, S. N. & Venkatasubramanian, V. (1993). Using Fuzzy Clustering with Ellipsoidal Units in Neural Networks for Robust Fault Classification. *Computers & Chemical Engineering*, 17, 765-784.

Kharoufeh, J.P. & Cox, S.M. (2005). Stochastic Models for Degradation-based Reliability. *IIE Transactions,* 37, 533 – 542.

Kotschenreuther, M., Dorland, W., Beer, M. A., & Hammett, G.W. (1995). Quantitative Predictions of Tokamak Energy Confinement from First-Principles Simulations with Kinetic Effects. *Phys. Plasmas,* 2, 2381.

Koutroumbas, K. & Kalouptsidis, N. (1994). Nearest Neighbor Pattern Classification Neural Networks. *Proceedings of the IEEE World Congress on Computational Intelligence and Neural Networks*, 5, 2911-2915, Orlando, FL.

Kramer, M.A., Thompson, M.L., & Bhagat, P.M. (1992). Embedding Theoretical Models in Neural Networks. *Proc. Amer. Control Conference*, 475-479.

Kresta, J., & MacGregor, J. F. (1991). Marlin multivariable statistical monitoring of process operating performance. *Can. J. Chem. Eng.,* 69, 35-47.

Kreutzer, S. E. & Hakimi, S. L. (1987). System-Level Fault Diagnosis: A Survey. *Microprocessing and Microprogramming*, 20, 323-330.

Kutner, M.H., Nachtsheim, C.J., & Neter, J. (2004). *Applied Linear Regression Models,* McGraw-Hill/Irwin, New York, New York.

Kwan, C., Zhang, X., Xu, R. & Haynes, L. (2003). A Novel Approach to Fault Diagnostics and Prognostics. *2003 IEEE International Conference on Robotics & Automation*, 604 – 609.

Lall, P., Pecht, M., & Harkim, E. (1997). *Influence of Temperature on Microelectronics and System Reliability*, New York: CRC Press.

Lee J.M., Qian, Yoo, Choi, S.W., Vanrolleghem, P.A., & Lee, I.B. (2004).Nonlinear process monitoring using kernel principal component analysis, *Chemical Engineering Science,* 59, 223-234.

Lin W., Qian, Y., & Li, X. (2000). Nonlinear dynamic principal component analysis for on-line process monitoring and diagnosis, *Computers and Chemical Engineering,* 24, 423-429.

Line, J.K. & Clements, N.S. (2005). A Systematic Approach for Developing Prognostic Algorithms on Large Complex Systems. *IEEE Aerospace Conference*, 1 – 7.

Lu, C. J. & Meeker, Q. W. (1993). Using Degradation Measures to Estimate a Time-to-Failure Distribution. *Technometrics,* 35, 161-174.

Mackenzie, M., & Tieu, A. K. (2004). Asymmetric Kernel Regression, *IEEE Transactions on Neural Networks*, 15, 2.

MacGregor, J. F., Jaeckle, C., Kiparissides, C., & Koutodi, M. (1994). Process monitoring and diagnosis by multiblock PLS methods. *AIChE Journal,* 40, 826-828.

Mallor, F. & Santos, J. (2003). Classification of Shock Models in System Reliability. *Monograf´ıas del Semin. Matem. Garc´ıa de Galdeano*, 27, 405–412.

Maybeck, P.S. (1979). *Stochastic Models, Estimation, and Control, Volume 1*. Academic Press, New York, NY.

Meeker, W. Q., Escobar, L. A., & Lu, C. J. (1998). Accelerated Degradation Tests: Modeling and Analysis. *Technometics*, 40, 89-99.

Mehra, R.K. & Peshon, I. (1971). An innovations approach to fault detection and diagnosis in dynamic systems. *Autmatica*, **7**, 637–640.

Mika, S., Scholkopf, B., Smola, A. J., Muller, K.R., Scholz, M., & Ratsch, G. (1999). Kernel PCA and de-noising in feature spaces. *Advances in Neural Information Processing Systems*, 11, 536-542.

Milne, R. (1987). Strategies for diagnosis. *IEEE Trans. Syst., Man Cybern*, 17, 333–339.

Mishra, S. & Pecht, M. (2002). In-situ Sensors for Product Reliability Monitoring. *Proceedings of SPIE*, 4755, 10-19.

Mishra, S., Ganesan, S., Pecht, M., & Xie, J. (2004) Life Consumption Monitoring for Electronics Prognostics. *IEEE Aerospace Conference Proceedings*, 3455 – 3467.

Murphy, O. J. (1990). Nearest Neighbor Pattern Classification Perceptrons. *Proceedings of the IEEE*, 78, 1595-1598.

Nadaraya, E.A.(1964). On estimating regression. *Theory of Probability and Its Applications*, 10, 186-190.

Natke, H. G. (1997). Symptom-Based Fault Diagnosis of Vibrating Structures. *Proceedings of the 51st Meeting of the MFTP Society*, 3-7, Virginia Beach, VA: April 14-18.

Nelson, W. (1990). *Accelerated Testing: Statistical Models, Test Plans, and Data Analysis*. John Wiley & Sons, New York, NY.

Neyman, J., & Pearson, E. (1933). On the Problem of the Most Efficient Tests of Statistical Hypotheses. *Philosophical Transaction of the Royal Society of London. Series A, Containing Papers of the Mathematical or Physical Character*, 231, 289-337.

Orchard, M. & Vachtsevanos, G. (2007). A particle filtering approach for on-line failure prognosis in a planetary carrier plate. *International Journal of Fuzzy Logic and Intelligent Systems*, 7, 221-227.

Patton, R. J., Uppal, F. J. , & Lopez-Toribio, C. J. (2000). Soft Computing Approaches to Fault Diagnosis for Dynamic Systems: A Survey. *Proceedings of the IFAC Symposium SAFEPROCESS* , 298-311, Budapest: June 14-16.

Pecht, M. & ARINC Inc (1995a). *Product Reliability, Maintainability, and Supportability Handbook*. CRC Press, New York, NY.

Pecht, M. & Dasgupta, A. (1995b). Physics of Failure: An Approach to Reliable Product Development. *Journal of the Institute of Environmental Sciences*, 38, 30 – 34.

Pecht, M., Das, D., & Ramakrishnan, A. (2002). The IEEE Standards on Reliability Program and Reliability Prediction Methods for Electronic Equipment. *Microelectronic Reliability*, 42 1259 – 1266.

Penha, R. & Hines, J. W. (2001). Using principal component analysis modeling to monitor temperature sensors in a nuclear research reactor. *In Proc. of the Maintenance and Reliability Center*.

Piovoso, M. J., Kosanovich, K. A., & Pearson, R. K. (1992). Monitoring process performance in real time. *Proc. American Control Conference*, 2359-2364.

Pook, P. K. & Ballard, D. H. (1994). Deictic Teleassistance. *Proceedings of the IEEE/RSJ/GI International Conference on Intelligent Robots and Systems*, 1, 245-252, Munich, Germany: September 12-16.

Psichogios, D.C. & Ungar, L.H. (1992). A Hybrid Neural Networks First Principles Approach to Process Modeling. *AIChE Journal*, 38, 1499-1511.

Ramakrishnan, A. & Pecht, M.G. (2003) A Life Consumption Monitoring Methodology for Electronic Systems. *IEEE Transactions on Components and Packaging Technologies*, 26, 625 – 634.

Rice, J. (1995). *Mathematical Statistics and Data Analysis*. Second Edition, Duxbury Press.

Riedesel, J. (1989). A Survey of Fault Diagnosis Technology. *Proceedings of the 24th Intersociety Energy Conversion Engineering Conference*, 1, 183-188, Washington, D.C.: August 6-11.

Romdhani, S., Gong, S., & Psarrou, A. (1999). A muti-view nonlinear active shape model using kernel PCA. *Proceedings of BMVC*, Nottingham, U.K., 483-492.

Roemer, M.J., Dzakowic, J., Orsagh, R.F., Byington, C.S., & Vachtsevanos, G. (2005). Validation and Verification of Prognostic and Health Management Technologies. *IEEE Aerospace Conference*, 3941 – 3947.

Russell, E., Chiang, L. H., & Braatz, R.D. (2000). *Data-Driven Techniques for Fault Detection and Diagnosis in Chemical Processes*. A Volume in the Advances in Industrial Control Series, Springer, New York, NY.

Seber, G. A. F., & Lee, A. J. (2003). Linear Regression Analysis, 2nd Edition, *Wiley-Interscience*.

Scholkopf, B., Smola, A. J. & Muller, K. (1998). Nonlinear component analysis as a kernel eigenvalue problem, *Neural Computation*, 10, 1299-1399.

Simani, S., Fantuzzi, C., & Patton, R. J. (2003). *Model-based Fault Diagnosis in Dynamic Systems Using Identification Techniques*, Springer-Verlag London Limited, London.

Storrick, G.D.,Petrovic, B., Oriani, L., Conway, L.E., & Conti, D., & Westinghouse Electric Company LLC, STD-AR-05-01 (2005). Instrumentation Needs for Integral Primary System Reactors (IPSRs). *USDOE*. Sept.

Schmidt, F., Henderson, R., & Wolgemuth, C. (1993). *Introduction to .Thermal Sciences*. New York: John Wiley & Sons, Inc.

Strid, I & Walentin, K. (2009). Block Kalman filtering for large scale DSGE models. *Computational Economics*, 3, 277-304.

Tarifa, E. E. & Nicolas J. S. (1997). Fault Diagnosis, Direct Graphs, and Fuzzy Logic. *Computers & Chemical Engineering*, 21, S649-S654.

te Braake, H.A.B., van Can, H.J.L.V., & Verbruggen, H.B. (1998). Semi-mechanistic Modeling of Chemical Processes with Neural Networks. *Engineering Applications of Artificial Intelligence*, 11, 507-515.

Thompson, M.L. & Kramer, M.A. (1994). Modeling Chemical Processes Using Prior Knowledge and Neural Networks. *AIChE Journal*, 4, 1328-1340.

Upadhyaya, B.R., Naghedolfeizi, M., & Raychaudhuri, B. (1994). Residual Life Estimation of Plant Components. *Periodic and Predictive Maintenance Technology*, 22 – 29.

Vachtsevanos, G., Kim, W., Al-Hasan, S., Rufus, F., Simon, M., Schrage, D., & Prasad, J.V.R. (1997). Mission Planning and Flight Control: Meeting the Challenge with Intelligent Techniques. *Journal of Advanced Computational Intelligence,* 1, 62-70.

Valentin, R., Newman, B., & Osterman, M. (2003). Remaining Life Assessment of Aging Electronics in Avionic Applications. *Proceedings of the Annual Reliability and Maintainability Symposium (RAMS),* 313 – 318.

Venkatasubramanian, V., Rengaswamy, R., Kavuri, S.N., & Yin, K. (2003). A Review of Process Fault Detection and Diagnosis Part III: Process 324 History Based Methods. *Computers & Chemical Engineering,* 27, 327-346.

Vichare, N., Rodgers, P., Eveloy, V., & Pecht, M. (2004). In Situ Temperature Measurement of a Notebook Computer – A Case Study of Health and Usage Monitoring of Electronics. *IEEE Transactions on Device and Materials Reliability,* 4, 658 – 663.

Vichare, N., & Pecht, M. (2006). Prognostics and Health Management of Electronics, *IEEE Transactions on Components and Packaging Technologies,* 29, 222 – 229.

Wald A. (1945). Sequential Tests of Statistical Hypotheses. *Annals of Mathematical Statistics,* 16, 117 – 186.

Wald, A. (1947). *Sequential Analysis*. John Wiley & Sons, New York, NY.

Wang H., Song, Z., & Li, P., (2001a). Improved PCA with application to process monitoring and diagnosis. *Journal of Chemical Industry and Engineering,* 52, 471-475.

Wang, P. & Vachtsevanos, G. (2001b). Fault Prognostics Using Dynamic Wavelet Neural Networks. *Journal of Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, 15, 349-365.

Wang, H., Song, Z., & Wang, H. (2002). Fault detection behavior analysis of PCA-based process monitoring approach. *Journal of Chemical Industry and Engineering*, 53, 297-301.

Wang, H., Li, P., & Yuan, Z. (2002). Understanding PCA fault detection results by using expectation analysis method. *Proceedings of the 41st IEEE Conference on Decision and Control.*, Las Vegas, NV, Dec.

Wang, P., & Coit, W. D., (2007). Reliability and Degradation Modeling with Random or Uncertain Failure Threshold. *Proceeding of the Annual Reliability and Maintainability Symposium*, Las Vegas, NV, January 28-31.

Wasserman, L. (2007). All of the Nonparametric Statistics. *Springer*.

Watson, G. S. (1964). Smooth Regression Analysis. *Indian Journal of Statistics*, 26, 359-372.

Welch, G. & Bishop, G. (2001). *An Introduction to the Kalman Filter*. SIGGRAPH, Los Angeles, CA.

Willsky, A. S. (1976). A survey of design methods for failure detection in dynamic systems. *Automatica*, 12, 601.

Wilsky, A.S. & Jones, H.L. (1974). A generalized likelihood ratio approach to state estimation in linear systems subject to abrupt changes. *Proc IEEE Conf. on Decision and Control*, Pheonix, Arizona.

Wilson, J.A., & Zorsetto, L.F.M. (1997). A Generalized approach to Process State Estimation Using Hybrid Artificial Neural Network Mechanistic Models. *Computers & Chemical Engineering*, 21, 951-963.

Wood, B. (2008). NERI-C Quarterly Report, Flow Loop Section.

Wong, D. K. (2001). Geo-Location in Urban Areas Using Signal Strength Repeatability. *IEEE Communications Letters*, 5, 411-413.

Zavaljevski, N. & Gross, K. C. (2000a). Support Vector Machines for Nuclear Reactor State Estimation. *Proceedings of the International Topical Meeting on Advances in Reactor Physics and Mathematics and Computation (PHYSOR 2000)*, Pittsburg, Pennsylvania.

Zavaljevski, N. & Gross, K. C. (2000b). Sensor Fault Detection in Nuclear Power Plants Using Multivariate State Estimation Technique and Support Vector Machines. *Proceedings of the Third International Conference of the Yugoslav Nuclear Society (YUNSC 2000)*, Belgrade, Yugoslavia.

Zhang, Y. & Qin, S. J. (2009). Enhanced statistical analysis of nonlinear processes using KPCA, KICA and SVM. *Chemical Engineering Science*, 64, 801-811.

APPENDICES

APPENDIX A: MATLAB CODE

ANPM MATLAB Code

There are a number of versions of the ANPM that have different capabilities.

The ANPM_PCA_T function provides the ANPM capability with the integrated PCA ECM technique is a total combined format.

```
function [totquery,
nmat,oldmodel,finalmodel,totpredict,finalpredict,FPMpredict,Fhypmat,Totres] =
ANPM_PCA_T(data,query,perday,cycle)

%Adaptation Model Function ANPM1
%   This function is to be used for model adaptation.
%
%   [totquery,
nmat,oldmodel,finalmodel,totpredict,finalpredict,FPMpredict,Fhypmat]
%   = ANPM1(data,query,perday,cycle) creates a new
%   model finalmodel that represents the data from the actual process of
%   interest and slowly deletes the original model, which is first created
%   using aakrmodcreate. nmat is the data matrix created by the new data,
%   this is without any faults detected, this can be used to create
%   a model solely dependent on the new data
%
%   [totquery,
nmat,oldmodel,finalmodel,totpredict,finalpredict,FPMpredict,Fhypmat]
%   = ANPM1(data,query,totcyclenum) this is if the total cycle number is
%   known but the perday of cycle by themselves is not known.
%
%   DESCRIPTION:
%   This is the first working ANPM model. It deletes the original memory
%   matrix with a linear deletion method in which the vector with the
%   highest sum of weights is deleted at a certain interval. This interval
%   is determined by the number of steps within the adaptation and the
%   number of original memory vectors within the FPM (First Principle Model).
%   ANPM1 uses the bandwidth that was optimized with the FPM and not
%   updated within the adaptation. The vector addition technique used was
%   a continual addition with periodic vector selection using
%   Adeli Hung technique.


%   Inputs
%   data- The FPM data used to create the inital aakr model.
%   query- Is the new input data that is going to be stored in nmat and
%       used to create the new model
%   perday- Is the number of data collection rows in a day
%   cycle- The number of days that the adapt model will be running.
```

```
%   Matt Humberstone
%   The University of Tennessee, Knoxville
%   Nuclear Engineering Department
%   Last Update:    2/26/2008
%

%   Using aakrmodcreate function to create the original model.
[te,ve,test,train,valid,origmemmat,model] = aakrmodcreate(data);

% Original model saved for comparison purposes
oldmodel=model;

%  Decayed Original Memory matrix that will be modified
domm=origmemmat;

%  Row deleted Original Memory matrix

rdomm=origmemmat;

% Size of the original memory matrix

[rows, columns]=size(origmemmat);


%   Number of Steps during adapt procedure
if nargin<4;
    steps=perday;
else
    steps = perday*cycle;
end;

%   Size of the model matrix
nmem = model.architecture.nmem;
ro=nmem;

split=floor(steps/nmem);
          %_____________________________________________________________
          %                          PCA ECM
          %_____________________________________________________________

          x=data(:,1:4);

          [xs meanx stdx]=zscore1(x);
          [xsq meanx1 stdx1]=zscore1(query(:,1:4));
          [pc,latent,explained] = pcacov(cov(xs));

          explained;
          pc_used=1;

          %The first statistic is Hotelling's T2 statistic
          % This measures the variation within a PCA model
```

```
            t2=tstat1(xs,pc,latent,pc_used);


            %The Q statistic measure the variation outside of the PCA model.

            q=qstat(xs,pc,pc_used);


            QL=max(q);
            TL=max(t2);


                QLmax_Vec(1:steps)=QL;
                TLmax_Vec(1:steps)=TL;



%  Initializing matrix
totquery=[];      % The query vectors all in the same matrix
totquery_s=[];    % The standardized query vectors all in the same matrix
nmat=[];          % The new query vectors without errors
queryadd=[];      % The new query vectors that will be added to the model
modmat=[];        % Matrix that is the model
totpredict=[];    % Total prediction matrix for the ANPM
Fhypmat=[];       % A matrix of Fhyp (fault detection) values
FhypmatT=[];       % A matrix of Fhyp (fault detection) values
totw=[];          % The sum of all the vector weights for the original memmat
newmodmat=[];     % The model memory matrix at the end of ANPM
TotFres=[];       % Total Fault residuals
Totres=[];        % Total residuals


t=0;
z=0;
l=0;
q=0;
h=0;
p=0;
nmodel=model;
sizeD=nmem;
Nofault=0;

% Threshold vector

Tvect=[0.5 0.5 0.5 0.5 1 1 0.5 0.5 1 1];
%Tvect=[1 1 1 1 3 3 1 1 3 3];
%Tvect=[0.25 0.25 0.25 0.25 0.5 0.5 0.25 0.25 0.5 0.5];
%   Create wait bar if necessary
display=true;
if display;
    wb = waitbar(0,'Running ANPM1');
end;
```

```
% The steps+100 upper limit is for the first 100 samples to be run through
% the ANPM, so at steps+100 i-100=steps.
for i=1:steps+100;

 % This develops an initial query matrix that gives certain values that
 % will be needed for calculations such as the mean and std for
 % standardization of the data

   if i<=99;

      % Total initial query matrix creation
        totquery=[totquery;query(i,:)];

   elseif i==100;
      % Total initial query matrix creation
        totquery=[totquery;query(i,:)];
        Meanq=mean(data);
        Stdvq=std(data);
   else


      % Standardizes the query vector
        query_s(i-100,:)=zscore1(query(i-100,:),Meanq, Stdvq);

      % Total query matrix creation continued
           if i<=steps;
              totquery=[totquery;query(i,:)];
           end

      % A standardized totquery matrix
        totquery_s =[totquery_s;query_s(i-100,:)];




%_____________________________________________________________________
  % Optimizing the new model
  %nmodel = optmodel1(nmodel,query(i-100,:),'error','bandwidth',0.1:0.2:3.9);
  %nmodel.architecture  ;
 %_____________________________________________________________________

       % Prediction and weight calculation
         [predict, reliability, w] = runmodel(nmodel,query_s(i-100,:),false);

       % Saved predictions for analysis
         totpredict=[totpredict;predict];

         % Analysis of query prediction
         residual=predict-query_s(i-100,:);

         Totres=[Totres;unscore(residual,Meanq, Stdvq)];
```

```
    columns=length(residual);

  %_____________________________________________________________________________
  % HR based on max weight not each individual weight
  % Find the weight and index
  %  [M ind]=max(w);

  % Find the hit rate contribution to the decay vector

   %HR(ind)=HR(ind)*0.9
  %_____________________________________________________________________________

%_____________________________________________________________________
%                 PCA ECM
%_____________________________________________________________________



  sdata=zscore1(query(i-100,1:4),meanx,stdx1);
      t2v(i)=tstat1(sdata,pc,latent,pc_used);
      qv(i)=qstat(sdata,pc,pc_used);

      if qv(i)>QL;
          FhypP=1;
      else
          FhypP=0;
      end;



  if FhypP==1;
      Fhyp(1:columns)=1; Nofault=Nofault+1;
  else
      Fhyp(1:columns)=0;
  end;

       % Saving a matrix of Fhyp values
    Fhypmat=[Fhypmat;Fhyp];

%_____________________________________________________________________
%                 Threshold Fault Detection
%_____________________________________________________________________

for h=1:columns;
    if residual(h)>=Tvect(h);
        FhypT(1:h)=1; Nofault=Nofault+1;
    else
        FhypT(1:h)=0;
    end
end

FhypmatT=[FhypmatT;FhypT];
```

```
      % Size of Row Deleted Original Memory Matrix
      [Rrdomm Cdomm]=size(rdomm);

           % Initializing the totw vector
             p=p+1;
             if p==1;
                for j=1:Rrdomm;
                    totw(j)=0;
                end;
             end;

         % Addition of the weights for only the original memory matrix
portion
            for x=1:Rrdomm;
                totw(x)=totw(x)+w(x);
            end;

      % Decay of initial matrix
        h=h+1;
          if h==split;
             h=0;

             % max weigth and index found
             [M ind]=max(totw);

             % Max weight vector deleted from rdomm
             rdomm(ind,:)=[];
             totw(ind)=[];
          end;



        %_____________________________________________________________________
        % For Dynamic SPRT calculations
        % Initial Tolerances for SPRT, set as constants but could be made
        % dependent on i

        %False Alarm Probability
        %   alpha = 0.0001;  %0.01%

        %Missed Alarm Probability
        %  beta = 0.10;    % 10%

        % Fault Detection
        % SPRT (Sequential Probability Ratio Test)

        % Training the SPRT for dynamic alpha and beta
        %  [ErrM ErrV Tol] = trainsprt(te,ve,alpha,beta);
        %_____________________________________________________________________
```

```
%_____________________________________________________________________
% Apply a logic screen to limit the number of alarms
% 3 out of 5 before a fault is declared
% CFhyp = confh(Fhyp, [1 2]); %possibility
%_____________________________________________________________________



 % Sum of the Fhyp for each sensor
   for k=1:columns;
       sumFhyp=sum(Fhyp(k));
       sumFhypT=sum(FhypT(k));
   end

% Add query if no fault detected within vector
  if sumFhyp == 0 && sumFhypT == 0;
      if Nofault<=2;
     z=z+1;
    % Query matrix with no faults detected
     nmat =[nmat;query_s(i-100,:)];

    % The non fault query vector is added to the queryadd vector
     queryadd=[queryadd;query_s(i-100,:)];


     % A 1500 vector memory matrix limit is set, then the Adeli-Hung
     % clusstering vector selection is used to set the queryadd matrix
     % back to 500 memory vector

          if z==1500;
               z=0;

             % New queryadd matrix
               queryadd=vectsel(nmat, 'h', 500);

             % Model matrix is first defined as the deleted original
             % memory matrix
               modmat=rdomm;

             % Then the queryadd is added onto the model matrix
               modmat=[modmat;queryadd];

          else
             % Then the queryadd matrix is added to the decayed original
             % memory matrix
               modmat=rdomm;

               modmat=[modmat;queryadd];
          end;

     % Fault detected
```

```
            else
              TotFres=[TotFres;residual];
            end;
          elseif  sumFhyp >=1;
              TotFres=[TotFres;residual];
                  for j=1:columns;
                      if Fhyp(j)==1;
                         fprintf(['Sensor  ', num2str(j) , ' has a fault
detected']);
                         fprintf([' at query vector ', num2str(i),'\n']);
                         modmat=modmat;
                     end;
                  end;
        %_____________________________________________________________
        % Need to determine if this fault is an actual fault
        %_____________________________________________________________
          end;


     % Set the new matrix into the model
     nmodel.data.strain = modmat;

    %   Display progress
        if display;
          m = mat2str(['Running ANPM (' mat2str(round(100*i/(steps+100)))
'%)']);
            waitbar(i/(steps+100),wb,m);
        end;
    end;
end;

%   Close the wait bar if necessary
if display;
    close(wb);
end;

% New queryadd matrix
  [colsnmat rowsnmat]=size(nmat);
  if rowsnmat>500;
  newmodmat=vectsel(nmat, 'h', 500);
  else
      newmodmat=modmat;
  end;
% Set the new matrix into the model
  nmodel.data.strain = newmodmat;


% Final model

finalmodel=nmodel;


%_____________________________________________________________________________
```

```
% Results Analysis

% Comparing the adaptive model predictions, new model predictions and
% the old model predicitons to the query actual values

% Standardized query

% Standardize with respect to the mean and std of the FPM data
query_s=zscore1(query,Meanq,Stdvq);

% Standardized with respect to the new datas mean and std
%query_s=zscore1(query);

% Final Model Predictions
finalpredict=runmodel(finalmodel, query_s);

% Original Model (FPM) Predictions
FPMpredict = runmodel(oldmodel, query_s);

% Residuals
Rfinal= finalpredict-query_s;
RFPM=FPMpredict-query_s;
RANPM=totpredict-query_s;

% For each sensor residuals and predictions plotted
for i=1:columns;

    % Prediction comparisons
    figure;
        plot(finalpredict(:,i),'g','linewidth',1.5);
        hold on;
        plot(FPMpredict(:,i) ,'k','linewidth',1.5);
        hold on;
        plot(totpredict(:,i) ,'b','linewidth',1.5);
        hold on;
        plot(query_s(:,i) ,'r','linewidth',2.5);
        hold off;
        legend('final prediction','FPM prediction','ANPM prediction','query
value','location','NorthEast');
        xlabel('Sample number');
        ylabel(['Sensor  ',num2str(i), 'predicted and actual values']);
        title(['Predictions and actual values of sensor   ',num2str(i)],
'FontWeight','Bold');


        %Residual Comparisons
        figure;
        plot(Rfinal(:,i),'g','linewidth',1.5);
        hold on;
        plot(RFPM(:,i) ,'k','linewidth',1.5);
        hold on;
        plot(RANPM(:,i) ,'b','linewidth',1.5);
        hold off;
```

```
        legend('final residual','FPM residual','ANPM
residual','location','NorthEast');
        xlabel('Sample number');
        ylabel(['Sensor  ',num2str(i), 'residual']);
        title(['Residuals for sensor   ',num2str(i)], 'FontWeight','Bold');

        % SPRT plots
        figure;
        plot(Fhypmat(:,i),'b*');
        hold on;
        plot(FhypmatT(:,i),'r*');
        legend('Fault detection','location','NorthEast');
        xlabel('Sample number');
        ylabel(['Sensor  ',num2str(i), 'Fhyp']);
        title(['Fault Detection for Sensor   ',num2str(i)],
'FontWeight','Bold');
end;
```

ANPM1 function is the first basic ANPM function with a continuous vector addition technique. This function was used for testing the adaptation of the model not the fault detection capability.

```
function [totquery,
nmat,oldmodel,finalmodel,totpredict,finalpredict,FPMpredict,Fhypmat] =
ANPM1(data,query,perday,cycle)

%Adaptation Model Function ANPM1
%   This function is to be used for model adaptation.
%
%   [totquery,
nmat,oldmodel,finalmodel,totpredict,finalpredict,FPMpredict,Fhypmat]
%   = ANPM1(data,query,perday,cycle) creates a new
%   model finalmodel that represents the data from the actual process of
%   interest and slowly deletes the original model, which is first created
%   using aakrmodcreate. nmat is the data matrix created by the new data,
%   this is without any faults detected, this can be used to create
%   a model solely dependent on the new data
%
%   [totquery,
nmat,oldmodel,finalmodel,totpredict,finalpredict,FPMpredict,Fhypmat]
%   = ANPM1(data,query,totcyclenum) this is if the total cycle number is
%   known but the perday of cycle by themselves is not known.
%
%   DESCRIPTION:
%   This is the first working ANPM model. It deletes the original memory
%   matrix with a linear deletion method in which the vector with the
%   highest sum of weights is deleted at a certain interval. This interval
```

```
%   is determined by the number of steps within the adaptation and the
%   number of original memory vectors within the FPM (First Principle Model).
%   ANPM1 uses the bandwidth that was optimized with the FPM and not
%   updated within the adaptation. The vector addition technique used was
%   a continual addition with periodic vector selection using
%   Adeli Hung technique.


%   Inputs
%   data- The FPM data used to create the inital aakr model.
%   query- Is the new input data that is going to be stored in nmat and
%       used to create the new model
%   perday- Is the number of data collection rows in a day
%   cycle- The number of days that the adapt model will be running.


%   Matt Humberstone
%   The University of Tennessee, Knoxville
%   Nuclear Engineering Department
%   Last Update:    2/26/2008
%

%   Using aakrmodcreate function to create the original model.
[te,ve,test,train,valid,origmemmat,model] = aakrmodcreate(data);

% Original model saved for comparison purposes
oldmodel=model;

%  Decayed Original Memory matrix that will be modified
domm=origmemmat;

%  Row deleted Original Memory matrix

rdomm=origmemmat;

% Size of the original memory matrix

[rows, columns]=size(origmemmat);

% Training SPRT for constant alpha and beta, if dynamic need to train
% within the loop
alpha=0.0001;
beta=0.1;
[ErrM ErrV Tol] = trainsprt(te,ve,alpha,beta);

%   Number of Steps during adapt procedure
if nargin<4;
    steps=perday;
else
    steps = perday*cycle;
end;

%   Size of the model matrix
```

```
nmem = model.architecture.nmem;
ro=nmem;

split=floor(steps/nmem);

%  Initializing matrix
totquery=[];      % The query vectors all in the same matrix
totquery_s=[];    % The standardized query vectors all in the same matrix
nmat=[];          % The new query vectors without errors
queryadd=[];      % The new query vectors that will be added to the model
modmat=[];        % Matrix that is the model
totpredict=[];    % Total prediction matrix for the ANPM
Fhypmat=[];       % A matrix of Fhyp (fault detection) values
totw=[];          % The sum of all the vector weights for the original memmat
newmodmat=[];     % The model memory matrix at the end of ANPM


t=0;
z=0;
l=0;
q=0;
h=0;
p=0;
nmodel=model;
sizeD=nmem;

%   Create wait bar if necessary
display=true;
if display;
    wb = waitbar(0,'Running ANPM1');
end;

% The steps+100 upper limit is for the first 100 samples to be run through
% the ANPM, so at steps+100 i-100=steps.
for i=1:steps+100;

 % This develops an initial query matrix that gives certain values that
 % will be needed for calculations such as the mean and std for
 % standardization of the data

   if i<=99;

      % Total initial query matrix creation
        totquery=[totquery;query(i,:)];

   elseif i==100;
      % Total initial query matrix creation
        totquery=[totquery;query(i,:)];
        Meanq=mean(data);
        Stdvq=std(data);
   else
```

```
      % Standardizes the query vector
        query_s(i-100,:)=zscore1(query(i-100,:),Meanq, Stdvq);

      % Total query matrix creation continued
           if i<=steps;
              totquery=[totquery;query(i,:)];
           end

      % A standardized totquery matrix
        totquery_s =[totquery_s;query_s(i-100,:)];




%_____________________________________________________________________________
  % Optimizing the new model
  %nmodel = optmodel1(nmodel,query(i-100,:),'error','bandwidth',0.1:0.2:3.9);
  %nmodel.architecture  ;
 %_____________________________________________________________________________

       % Prediction and weight calculation
         [predict, reliability, w] = runmodel(nmodel,query_s(i-100,:),false);

       % Saved predictions for analysis
         totpredict=[totpredict;predict];

      %_____________________________________________________________________
      % HR based on max weight not each individual weight
      % Find the weight and index
      %  [M ind]=max(w);

      % Find the hit rate contribution to the decay vector

       %HR(ind)=HR(ind)*0.9
      %_____________________________________________________________________

    % Size of Row Deleted Original Memory Matrix
      [Rrdomm Cdomm]=size(rdomm);

           % Initializing the totw vector
             p=p+1;
             if p==1;
                for j=1:Rrdomm;
                    totw(j)=0;
                end;
             end;

         % Addition of the weights for only the original memory matrix
portion
           for x=1:Rrdomm;
               totw(x)=totw(x)+w(x);
           end;
```

```
 % Decay of initial matrix
   h=h+1;
     if h==split;
        h=0;

        % max weigth and index found
        [M ind]=max(totw);

        % Max weight vector deleted from rdomm
        rdomm(ind,:)=[];
        totw(ind)=[];
     end;

  % Analysis of query prediction
    residual=predict-query_s(i-100,:);

    %_____________________________________________________________________
    % For Dynamic SPRT calculations
    % Initial Tolerances for SPRT, set as constants but could be made
    % dependent on i

    %False Alarm Probability
    %   alpha = 0.0001;  %0.01%

    %Missed Alarm Probability
    %  beta = 0.10;    % 10%

    % Fault Detection
    % SPRT (Sequential Probability Ratio Test)

    % Training the SPRT for dynamic alpha and beta
    %  [ErrM ErrV Tol] = trainsprt(te,ve,alpha,beta);
    %_____________________________________________________________________

 % Running SPRT based on the new prediction
    [Fhyp Fscore]= sprtn(ErrM, ErrV, residual, alpha, beta, Tol);

 % Saving a matrix of Fhyp values
    Fhypmat=[Fhypmat;Fhyp];
%_____________________________________________________________________
% Apply a logic screen to limit the number of alarms
% 3 out of 5 before a fault is declared
% CFhyp = confh(Fhyp, [1 2]); %possibility
%_____________________________________________________________________


[rF cF]=size(Fhyp);
% Sum of the Fhyp for each sensor
  for k=1:cF;
      sumFhyp=sum(Fhyp(k));
  end
```

```
    % Add query if no fault detected within vector
      if sumFhyp == 0;
         z=z+1;
       % Query matrix with no faults detected
         nmat =[nmat;query_s(i-100,:)];

       % The non fault query vector is added to the queryadd vector
         queryadd=[queryadd;query_s(i-100,:)];


        % A 1500 vector memory matrix limit is set, then the Adeli-Hung
        % clusstering vector selection is used to set the queryadd matrix
        % back to 500 memory vector

             if z==1500;
                  z=0;

                % New queryadd matrix
                  queryadd=vectsel(nmat, 'h', 500);

                % Model matrix is first defined as the deleted original
                % memory matrix
                  modmat=rdomm;

                % Then the queryadd is added onto the model matrix
                  modmat=[modmat;queryadd];

             else
                % Then the queryadd matrix is added to the decayed original
                % memory matrix
                  modmat=rdomm;

                  modmat=[modmat;queryadd];
             end;

        % Fault detected
          elseif  sumFhyp >=1;
                  for j=1:columns;
                      if Fhyp(j)==1;
                         fprintf(['Sensor  ', num2str(j) , ' has a fault
detected']);
                         fprintf([' at query vector ', num2str(i),'\n']);
                         modmat=modmat;
                     end;
                  end;
        %_____________________________________________________________________
        % Need to determine if this fault is an actual fault
        %_____________________________________________________________________
          end;


     % Set the new matrix into the model
     nmodel.data.strain = modmat;
```

```
    %   Display progress
        if display;
          m = mat2str(['Running ANPM (' mat2str(round(100*i/(steps+100)))
'%)']);
            waitbar(i/(steps+100),wb,m);
        end;
    end;
end;

%   Close the wait bar if necessary
if display;
    close(wb);
end;

% New queryadd matrix
  newmodmat=vectsel(nmat, 'h', 500);

% Set the new matrix into the model
  nmodel.data.strain = newmodmat;


% Final model

finalmodel=nmodel;


%_____________________________________________________________________
% Results Analysis

% Comparing the adaptive model predictions, new model predictions and
% the old model predicitons to the query actual values

% Standardized query

% Standardize with respect to the mean and std of the FPM data
query_s=zscore1(query,Meanq,Stdvq);

% Standardized with respect to the new datas mean and std
%query_s=zscore1(query);

% Final Model Predictions
finalpredict=runmodel(finalmodel, query_s);

% Original Model (FPM) Predictions
FPMpredict = runmodel(oldmodel, query_s);

% Residuals
Rfinal= finalpredict-query_s;
RFPM=FPMpredict-query_s;
RANPM=totpredict-query_s;
```

```
% For each sensor residuals and predictions plotted
for i=1:columns;

    % Prediction comparisons
    figure;
        plot(finalpredict(:,i),'g','linewidth',1.5);
        hold on;
        plot(FPMpredict(:,i) ,'k','linewidth',1.5);
        hold on;
        plot(totpredict(:,i) ,'b','linewidth',1.5);
        hold on;
        plot(query_s(:,i) ,'r','linewidth',2.5);
        hold off;
        legend('final prediction','FPM prediction','ANPM prediction','query
value','location','NorthEast');
        xlabel('Sample number');
        ylabel(['Sensor  ',num2str(i), 'predicted and actual values']);
        title(['Predictions and actual values of sensor   ',num2str(i)],
'FontWeight','Bold');


        %Residual Comparisons
        figure;
        plot(Rfinal(:,i),'g','linewidth',1.5);
        hold on;
        plot(RFPM(:,i) ,'k','linewidth',1.5);
        hold on;
        plot(RANPM(:,i) ,'b','linewidth',1.5);
        hold off;
        legend('final residual','FPM residual','ANPM
residual','location','NorthEast');
        xlabel('Sample number');
        ylabel(['Sensor  ',num2str(i), 'residual']);
        title(['Residuals for sensor   ',num2str(i)], 'FontWeight','Bold');

        % SPRT plots
        figure;
        plot(Fhypmat(:,i),'*');
        legend('SPRT Fault detection','location','NorthEast');
        xlabel('Sample number');
        ylabel(['Sensor  ',num2str(i), 'Fhyp']);
        title(['SPRT fault detection for sensor   ',num2str(i)],
'FontWeight','Bold');
end;
```

ANPM1_5 function is a basic ANPM function with a step vector addition technique. This function is similar to ANPM1 function as in it was used for testing the adaptation of the model not the fault detection capability.

```
function [totquery,
nmat,oldmodel,finalmodel,totpredict,finalpredict,FPMpredict,Fhypmat] =
ANPM1_5(data,query,perday,cycle)

%Adaptation Model Function ANPM1
%   This function is to be used for model adaptation.
%
%   [totquery,
nmat,oldmodel,finalmodel,totpredict,finalpredict,FPMpredict,Fhypmat]
%   = ANPM1(data,query,perday,cycle) creates a new
%   model finalmodel that represents the data from the actual process of
%   interest and slowly deletes the original model, which is first created
%   using aakrmodcreate. nmat is the data matrix created by the new data,
%   this is without any faults detected, this can be used to create
%   a model solely dependent on the new data
%
%   [totquery,
nmat,oldmodel,finalmodel,totpredict,finalpredict,FPMpredict,Fhypmat]
%   = ANPM1(data,query,totcyclenum) this is if the total cycle number is
%   known but the perday of cycle by themselves is not known.
%
%   DESCRIPTION:
%   This is the first working ANPM model. It deletes the original memory
%   matrix with a linear deletion method in which the vector with the
%   highest sum of weights is deleted at a certain interval. This interval
%   is determined by the number of steps within the adaptation and the
%   number of original memory vectors within the FPM (First Principle Model).
%   ANPM1 uses the bandwidth that was optimized with the FPM and not
%   updated within the adaptation. The vector addition technique used was
%   a continual addition with periodic vector selection using
%   Adeli Hung technique.


%   Inputs
%   data- The FPM data used to create the inital aakr model.
%   query- Is the new input data that is going to be stored in nmat and
%       used to create the new model
%   perday- Is the number of data collection rows in a day
%   cycle- The number of days that the adapt model will be running.


%   Matt Humberstone
%   The University of Tennessee, Knoxville
%   Nuclear Engineering Department
%   Last Update:    2/26/2008
%

%   Using aakrmodcreate function to create the original model.
[te,ve,test,train,valid,origmemmat,model] = aakrmodcreate(data);

% Original model saved for comparison purposes
oldmodel=model;

%  Decayed Original Memory matrix that will be modified
```

```
domm=origmemmat;

%  Row deleted Original Memory matrix

rdomm=origmemmat;

% Size of the original memory matrix

[rows, columns]=size(origmemmat);

% Training SPRT for constant alpha and beta, if dynamic need to train
% within the loop
alpha=0.0001;
beta=0.1;
[ErrM ErrV Tol] = trainsprt(te,ve,alpha,beta);

%   Number of Steps during adapt procedure
if nargin<4;
    steps=perday;
else
    steps = perday*cycle;
end;

%   Size of the model matrix
nmem = model.architecture.nmem;
ro=nmem;

split=floor(steps/nmem);

%  Initializing matrix
totquery=[];      % The query vectors all in the same matrix
totquery_s=[];    % The standardized query vectors all in the same matrix
nmat=[];          % The new query vectors without errors
queryadd=[];      % The new query vectors that will be added to the model
modmat=[];        % Matrix that is the model
totpredict=[];    % Total prediction matrix for the ANPM
Fhypmat=[];       % A matrix of Fhyp (fault detection) values
totw=[];          % The sum of all the vector weights for the original memmat
newmodmat=[];     % The model memory matrix at the end of ANPM


t=0;
z=0;
l=0;
q=0;
h=0;
p=0;
nmodel=model;
sizeD=nmem;

%   Create wait bar if necessary
display=true;
if display;
```

```
    wb = waitbar(0,'Running ANPM1');
end;

% The steps+100 upper limit is for the first 100 samples to be run through
% the ANPM, so at steps+100 i-100=steps.
for i=1:steps+100;

 % This develops an initial query matrix that gives certain values that
 % will be needed for calculations such as the mean and std for
 % standardization of the data

   if i<=99;

      % Total initial query matrix creation
        totquery=[totquery;query(i,:)];

   elseif i==100;
      % Total initial query matrix creation
        totquery=[totquery;query(i,:)];
        Meanq=mean(data);
        Stdvq=std(data);
   else


      % Standardizes the query vector
        query_s(i-100,:)=zscore1(query(i-100,:),Meanq, Stdvq);

      % Total query matrix creation continued
           if i<=steps;
              totquery=[totquery;query(i,:)];
           end

      % A standardized totquery matrix
        totquery_s =[totquery_s;query_s(i-100,:)];




%_____________________________________________________________________
  % Optimizing the new model
  %nmodel = optmodel1(nmodel,query(i-100,:),'error','bandwidth',0.1:0.2:3.9);
  %nmodel.architecture  ;
 %_____________________________________________________________________

       % Prediction and weight calculation
         [predict, reliability, w] = runmodel(nmodel,query_s(i-100,:),false);

       % Saved predictions for analysis
         totpredict=[totpredict;predict];

      %_____________________________________________________________________
      % HR based on max weight not each individual weight
```

```
      % Find the weight and index
      %  [M ind]=max(w);

      % Find the hit rate contribution to the decay vector

       %HR(ind)=HR(ind)*0.9
      %_____________________________________________________________________

    % Size of Row Deleted Original Memory Matrix
      [Rrdomm Cdomm]=size(rdomm);

           % Initializing the totw vector
             p=p+1;
             if p==1;
                for j=1:Rrdomm;
                    totw(j)=0;
                end;
             end;

         % Addition of the weights for only the original memory matrix
portion
           for x=1:Rrdomm;
               totw(x)=totw(x)+w(x);
           end;

      % Decay of initial matrix
        h=h+1;
          if h==split;
             h=0;

             % max weigth and index found
             [M ind]=max(totw);

             % Max weight vector deleted from rdomm
             rdomm(ind,:)=[];
             totw(ind)=[];
          end;

       % Analysis of query prediction
         residual=predict-query_s(i-100,:);

        %_____________________________________________________________________
        % For Dynamic SPRT calculations
        % Initial Tolerances for SPRT, set as constants but could be made
        % dependent on i

        %False Alarm Probability
        %   alpha = 0.0001;  %0.01%

        %Missed Alarm Probability
        %  beta = 0.10;    % 10%

        % Fault Detection
```

```
    % SPRT (Sequential Probability Ratio Test)

    % Training the SPRT for dynamic alpha and beta
    %  [ErrM ErrV Tol] = trainsprt(te,ve,alpha,beta);
    %_____________________________________________________________________

  % Running SPRT based on the new prediction
    [Fhyp Fscore]= sprtn(ErrM, ErrV, residual, alpha, beta, Tol);

  % Saving a matrix of Fhyp values
    Fhypmat=[Fhypmat;Fhyp];
%_____________________________________________________________________
% Apply a logic screen to limit the number of alarms
% 3 out of 5 before a fault is declared
% CFhyp = confh(Fhyp, [1 2]); %possibility
%_____________________________________________________________________


[rF cF]=size(Fhyp);
% Sum of the Fhyp for each sensor
  for k=1:cF;
      sumFhyp=sum(Fhyp(k));
  end

% Add query if no fault detected within vector
  if sumFhyp == 0;
     z=z+1;
   % Query matrix with no faults detected
     nmat =[nmat;query_s(i-100,:)];

   % The non fault query vector is added to the queryadd vector
     %queryadd=[queryadd;query_s(i-100,:)];


    % A 1500 vector memory matrix limit is set, then the Adeli-Hung
    % clusstering vector selection is used to set the queryadd matrix
    % back to 500 memory vector


         modmat=nmodel.data.strain;
         if z==1500;
              z=0;

            % New queryadd matrix
              queryadd=vectsel(nmat, 'h', 500);

            % Model matrix is first defined as the deleted original
            % memory matrix
              modmat=rdomm;

            % Then the queryadd is added onto the model matrix
              modmat=[modmat;queryadd];
```

```
            else
               % Then the queryadd matrix is added to the decayed original
               % memory matrix
                 %modmat=rdomm;
                  modmat=modmat;
                 %modmat=[modmat;queryadd];
            end;

        % Fault detected
          elseif  sumFhyp >=1;
                  for j=1:columns;
                      if Fhyp(j)==1;
                         fprintf(['Sensor  ', num2str(j) , ' has a fault
detected']);
                         fprintf([' at query vector ', num2str(i),'\n']);
                         modmat=modmat;
                     end;
                  end;
        %_____________________________________________________________
        % Need to determine if this fault is an actual fault
        %_____________________________________________________________
          end;


     % Set the new matrix into the model
     nmodel.data.strain = modmat;

    %   Display progress
        if display;
          m = mat2str(['Running ANPM (' mat2str(round(100*i/(steps+100)))
'%)']);
            waitbar(i/(steps+100),wb,m);
        end;
    end;
end;

%   Close the wait bar if necessary
if display;
    close(wb);
end;

% New queryadd matrix
  newmodmat=vectsel(nmat, 'h', 500);

% Set the new matrix into the model
  nmodel.data.strain = newmodmat;


% Final model

finalmodel=nmodel;
```

```
%_____________________________________________________________________
% Results Analysis

% Comparing the adaptive model predictions, new model predictions and
% the old model predicitons to the query actual values

% Standardized query

% Standardize with respect to the mean and std of the FPM data
query_s=zscore1(query,Meanq,Stdvq);

% Standardized with respect to the new datas mean and std
%query_s=zscore1(query);

% Final Model Predictions
finalpredict=runmodel(finalmodel, query_s);

% Original Model (FPM) Predictions
FPMpredict = runmodel(oldmodel, query_s);

% Residuals
Rfinal= finalpredict-query_s;
RFPM=FPMpredict-query_s;
RANPM=totpredict-query_s;

% For each sensor residuals and predictions plotted
for i=1:columns;

    % Prediction comparisons
    figure;
        plot(finalpredict(:,i),'g','linewidth',1.5);
        hold on;
        plot(FPMpredict(:,i) ,'k','linewidth',1.5);
        hold on;
        plot(totpredict(:,i) ,'b','linewidth',1.5);
        hold on;
        plot(query_s(:,i) ,'r','linewidth',2.5);
        hold off;
        legend('final prediction','FPM prediction','ANPM prediction','query
value','location','NorthEast');
        xlabel('Sample number');
        ylabel(['Sensor  ',num2str(i), 'predicted and actual values']);
        title(['Predictions and actual values of sensor   ',num2str(i)],
'FontWeight','Bold');


        %Residual Comparisons
        figure;
        plot(Rfinal(:,i),'g','linewidth',1.5);
        hold on;
        plot(RFPM(:,i) ,'k','linewidth',1.5);
        hold on;
```

```
        plot(RANPM(:,i) ,'b','linewidth',1.5);
        hold off;
        legend('final residual','FPM residual','ANPM
residual','location','NorthEast');
        xlabel('Sample number');
        ylabel(['Sensor  ',num2str(i), 'residual']);
        title(['Residuals for sensor   ',num2str(i)], 'FontWeight','Bold');

        % SPRT plots
        figure;
        plot(Fhypmat(:,i),'*');
        legend('SPRT Fault detection','location','NorthEast');
        xlabel('Sample number');
        ylabel(['Sensor  ',num2str(i), 'Fhyp']);
        title(['SPRT fault detection for sensor   ',num2str(i)],
'FontWeight','Bold');
end;
```

The function aakrmodcreate inputs the original data from the system of interest and separates the data into a train, test, and validation data set. It then optimizes the bandwidth and outputs the model the original memory matrix, the train, test, and validation data sets, and the train and validation error for use in an SPRT algorithm.

```
function [te,ve,test,train,valid,origmemmat,model] = aakrmodcreate(data)

%   aakr Standard Model Function
%   This function is to be used for standard aakr model creation.
%
%   [test,train,validation,model] = aakrmodcreate(data) creates an aakr
%   based on the data given. The data should include the variables of
%   interest,so vector selection will not be done.
%
%   The data will be split into a test train and validation data set using
%   the function initial4. The data will also be cleaned using cleandata
%   function. The model characteristics and certain model attributes that
%   will be needed for adapt model will be passed on.
%
%   Inputs
%   Data matrix data, which is the data matrix that includes the vectors of
%   interest.
%
%   Outputs
%   This function outputs the test, train, and validation data sets, the
%   final aakr model.


%   Matt Humberstone
%   The University of Tennessee, Knoxville
```

```
%   Nuclear Engineering Department
%   Last Update:    2/26/2008
%

% This standardizes the data and cleans the data
data_s=zscore1(data);
xc=cleandata(data_s);

% This seperates the data into a train, test, and validation data sets
% also plots the train, test , and validation data sets
[train test valid]=initial4(xc,100);

% Initializes the aakr with the train data set
model = initmodel('aakr',train,'bandwidth',0.2);
model = setmsa(model,'plotresults',false);

%Sets the number of memory vectors in the model to 500
model.architecture.nmem=500;



% Vector Selection method used is Adeli-Hung Clustering
%  model.architecture.vsmethod=h; %Not needed for initial model creation

% Model architecture
model.architecture

% optimize the bandwidth %
model = optmodel(model,test,'error','bandwidth',.1:.1:4);
model.architecture

% Model characteristics
model=modchar(model);

% extract analytic uncertainty %
pua = ones(length(test),1)*model.attributes.uncertainty;
%uncertainty = model.attributes.uncertainty./span.*100; % percent span %
avg_accuracy = mean(model.attributes.accuracy.*100); % percent %
avg_autosens = mean(model.attributes.autosensitivity);
avg_crosssens = mean(model.attributes.crosssensitivity);
avg_eulmdet = mean(model.attributes.eulmdetectability.*100); % percent %
avg_sprtdet = mean(model.attributes.sprtdetectability.*100); % percent %

% The original memory matrix
origmemmat=model.data.strain;

% Training predictions
tp=runmodel(model,train);

% Test prediction
vp = runmodel(model,test);

% Train Error
te = tp-train;
```

```
% Test Error
ve = vp-test;

%_____________________________________________________________________________
```

Operational and PCA ECM Code

The code below was used for preprocessing of the flow loop data and the operation of the ANPM and the PCA ECM techniques.

```
clear;
load profile1;
load profile2;
load Prof1_RVB_10;
load Prof2_RVB_10;
load Prof1_fault60_10;
load Prof2_fault60_10;
load Prof1_pumpfail_10;
load Prof2_pumpfail_10;
load Profile1_simulator;
load Profile2_simulator;

P1_L=profile1_loop(300:8300,2:11);
P2_L=profile2_loop(300:13000,2:11);
P1_S=[profile1_simulink_reduced(300:8300,2:9)
profile1_simulink_reduced(300:8300,10:11)];
P2_S=[profile2_simulink_reduced(300:13000,2:9)
profile2_simulink_reduced(300:13000,10:11)];

[totquery,
nmat,oldmodel,finalmodel,totpredict,finalpredict,FPMpredict,Fhypmat,TotFres]
= ANPM_PCA(P1_S,P1_L,8001)

[totquery,
nmat,oldmodel,finalmodel,totpredict,finalpredict,FPMpredict,Fhypmat,TotFres]
= ANPM_PCA_T(P1_S,P1_L,8001)

MSE_FPM = MSE_ANPM1(P1_L, FPMpredict, P1_S)
MSE_ANPM = MSE_ANPM1(P1_L, totpredict, P1_S)
MSE_final = MSE_ANPM1(P1_L, finalpredict, P1_S)

fault60_prof1=Prof1_fault60_10(300:8242,:);
```

```
[totquery,
nmat,oldmodel,finalmodel,totpredict,finalpredict,FPMpredict,Fhypmat,Totres] =
ANPM_PCA(P1_S,fault60_prof1,7943)

[totquery,
nmat,oldmodel,finalmodel,totpredict,finalpredict,FPMpredict,Fhypmat,Totres] =
ANPM_PCA_T(P1_S,fault60_prof1,7943)



Totres_centered=Totres-ones(7943,1)*mean(Totres(1:3000,:));

[rows cols]=size(Totres);

n=4000;

stepsize1=ceil((rows-n)/3600);

Totres2=-Totres_centered(1:stepsize1:(rows-n),:);


[faults possibilities] = fault_identify(Totres2)


Poss_Matrix=[0.0124 0.0129 0.0112 0.0112 0.0125; 0.1111 0.1198 0.1313 0.1523
0.1841;
             0.1205 0.1318 0.1399 0.1379 0.1130; 0.1184 0.1328 0.1453 0.1633
0.1841;
             0.0925 0.0951 0.0989 0.1006 0.0927; 0.1800 0.1495 0.1399 0.1090
0.0927;
             0.0954 0.0998 0.1072 0.1050 0.0962; 0.1815 0.1667 0.1496 0.1277
0.1121;
             0.0679 0.0703 0.0698 0.0721 0.0735; 0.0204 0.0213 0.0206 0.0209
0.0206];

timesteps=[4000 5000 6000 7000 8000];

figure;
plot(timesteps, Poss_Matrix);
title('Fault Possibilities Over Time')
legend('1','2','3','4','5','6','7','8','9','10');
xlabel('Time Step')
ylabel('Possibility')


% RVB fault detection

RVB_Prof1=Prof1_RVB_10(300:8242,:);

[totquery,
nmat,oldmodel,finalmodel,totpredict,finalpredict,FPMpredict,Fhypmat,Totres] =
ANPM_PCA_T(P1_S,RVB_Prof1,7943)
```

```
Totres_centered=Totres-ones(7943,1)*mean(Totres(1:3000,:));

[rows cols]=size(Totres);

n=7800;

stepsize1=ceil((rows-n)/3600);

Totres2=-Totres_centered(1:stepsize1:(rows-n),:);


[faults possibilities] = fault_identify(Totres2)

Poss_Matrix=[0.1027 0.0861 0.0160 0.0116 0.0200; 0.0997 0.1021 0.0890 0.0706
0.0778;
             0.0980 0.1009 0.1239 0.0831 0.0710; 0.1007 0.1020 0.0890 0.0727
0.0895;
             0.0994 0.1027 0.0852 0.0544 0.0614; 0.0978 0.1021 0.1879 0.2837
0.2071;
             0.0986 0.1014 0.0994 0.0657 0.0923; 0.0204 0.1015 0.2087 0.2975
0.1954;
             0.1022 0.1047 0.0658 0.0437 0.0735; 0.1026 0.0965 0.0251 0.0170
0.0512];

timesteps=[4000 5000 6000 7000 7800];

figure;
plot(timesteps, Poss_Matrix);
title('Fault Possibilities Over Time')
legend('1','2','3','4','5','6','7','8','9','10');
xlabel('Time Step')
ylabel('Possibility')

% Without the Threshold Fault detect
[totquery,
nmat,oldmodel,finalmodel,totpredict,finalpredict,FPMpredict,Fhypmat,Totres] =
ANPM_PCA(P1_S,RVB_Prof1,7943)


% Pumpfail fault detection

pumpfail_10=Prof1_pumpfail_10(300:8242,:);

[totquery,
nmat,oldmodel,finalmodel,totpredict,finalpredict,FPMpredict,Fhypmat,Totres] =
ANPM_PCA_T(P1_S,pumpfail_10,7943)


%_____________________________________________________________________
%                        PCA ECM
```

```
%_____________________________________________________________________
%_____________________________________________________________________
% For Profile 1
%_____________________________________________________________________

B=P1_S;
x=B(:,1:4);
[rows cols]=size(x);
[xs meanx stdx]=zscore1(x);
[pc,latent,explained] = pcacov(cov(xs));

explained;
pc_used=1;

%The first statistic is Hotelling's T2 statistic
% This measures the variation within a PCA model

t2=tstat1(xs,pc,latent,pc_used);

% This is the 95% confidence on the t2 statistic
tlimit=tlim(size(xs,1),pc_used);

%The Q statistic measure the variation outside of the PCA model.

q=qstat(xs,pc,pc_used);

% This is the 95% confidence on the Q statistic

qlimit=qlim(xs,pc_used);

start=1;

QL=max(q(start:rows));
TL=max(t2(start:rows));


QLmax_Vec(1:rows)=QL;
TLmax_Vec(1:rows)=TL;

% _____________________ Data Being tested_____________________
%                   Loop Data No Faults Profile 1
%_____________________________________________________________
B1=P1_L(:,1:4);
%sdata=zscore1(B1,meanx,stdx);
sdata=zscore1(B1,meanx, stdx);
t2_N2=tstat1(sdata,pc,latent,pc_used);
q_N2=qstat(sdata,pc,pc_used);


[rows cols]=size(B);

[rows2 cols2]=size(B1);
```

```
if rows<=rows2;
    rows =rows;
else
    rows=rows2;
end;


for j=start:rows;
    if t2_N2(j)>=TLmax_Vec(j);
       TF(j)=1;
    else
        TF(j)=0;
    end;
    if q_N2(j)>=QLmax_Vec(j);
       QF(j)=1;
    else
        QF(j)=0;
    end;
end;


figure;

subplot(3,1,1)
plot(t2_N2(start:rows),'*r');
hold on;
plot(t2(start:rows),'*b');
hold on;
plot(TLmax_Vec,'k');
title('T2 stats for simulated and actual data');
legend('Actual t2','Simulated t2 stats');
xlabel('time step');
ylabel('t2 stat value');


subplot(3,1,2)
plot(q_N2(start:rows),'*r');
hold on;
plot(q(start:rows),'*b');
hold on;
plot(QLmax_Vec,'k');
title('Q stats for simulated and actual data');
legend('Actual Q stats','Simulated Q');
xlabel('time step');
ylabel('Q stat value');

subplot(3,1,3)
plot(TF(start:rows),'or');
hold on;
plot(QF(start:rows),'ob');
title('Fault Detection Results for Q-stat and t2-stat');
legend('t2 stat results','q stat results');
xlabel('time step');
```

```
ylabel('Fault verdict');

% _____________________ Data Being tested_____________________
%                    Loop Data No Faults Profile 2
%_____________________________________________________________
B1=P2_L(:,1:4);
%sdata=zscore1(B1,meanx,stdx);
sdata=zscore1(B1);
t2_N2=tstat1(sdata,pc,latent,pc_used);
q_N2=qstat(sdata,pc,pc_used);


[rows cols]=size(B);

[rows2 cols2]=size(B1);

if rows<=rows2;
    rows =rows;
else
    rows=rows2;
end;


for j=start:rows;
    if t2_N2(j)>=TLmax_Vec(j);
       TF(j)=1;
    else
        TF(j)=0;
    end;
    if q_N2(j)>=QLmax_Vec(j);
       QF(j)=1;
    else
        QF(j)=0;
    end;
end;


figure;

subplot(3,1,1)
plot(t2_N2(start:rows),'*r');
hold on;
plot(t2(start:rows),'*b');
hold on;
plot(TLmax_Vec,'k');
title('T2 stats for simulated and actual data');
legend('Actual t2','Simulated t2 stats');
xlabel('time step');
ylabel('t2 stat value');


subplot(3,1,2)
plot(q_N2(start:rows),'*r');
```

```
hold on;
plot(q(start:rows),'*b');
hold on;
plot(QLmax_Vec,'k');
title('Q stats for simulated and actual data');
legend('Actual Q stats','Simulated Q');
xlabel('time step');
ylabel('Q stat value');

subplot(3,1,3)
plot(TF(start:rows),'or');
hold on;
plot(QF(start:rows),'ob');
title('Fault Detection Results for Q-stat and t2-stat');
legend('t2 stat results','q stat results');
xlabel('time step');
ylabel('Fault verdict');

% _____________________ Data Being tested_____________________
%             Loop Data Faults 60mm drift Profile 1
%_____________________________________________________________
B1=Prof1_fault60_10(:,1:4);
%sdata=zscore1(B1,meanx,stdx);
sdata=zscore1(B1);
t2_N2=tstat1(sdata,pc,latent,pc_used);
q_N2=qstat(sdata,pc,pc_used);


[rows cols]=size(B);

[rows2 cols2]=size(B1);

if rows<=rows2;
    rows =rows;
else
    rows=rows2;
end;


for j=start:rows;
    if t2_N2(j)>=TLmax_Vec(j);
       TF(j)=1;
    else
        TF(j)=0;
    end;
    if q_N2(j)>=QLmax_Vec(j);
       QF(j)=1;
    else
        QF(j)=0;
    end;
end;
```

```
figure;

subplot(3,1,1)
plot(t2_N2(start:rows),'*r');
hold on;
plot(t2(start:rows),'*b');
hold on;
plot(TLmax_Vec,'k');
title('T2 stats for simulated and actual data');
legend('Actual t2','Simulated t2 stats');
xlabel('time step');
ylabel('t2 stat value');


subplot(3,1,2)
plot(q_N2(start:rows),'*r');
hold on;
plot(q(start:rows),'*b');
hold on;
plot(QLmax_Vec,'k');
title('Q stats for simulated and actual data');
legend('Actual Q stats','Simulated Q');
xlabel('time step');
ylabel('Q stat value');

subplot(3,1,3)
plot(TF(start:rows),'or');
hold on;
plot(QF(start:rows),'ob');
title('Fault Detection Results for Q-stat and t2-stat');
legend('t2 stat results','q stat results');
xlabel('time step');
ylabel('Fault verdict');

% _____________________ Data Being tested_____________________
%             Loop Data Faults 60mm drift Profile 2
%_____________________________________________________________
B1=Prof2_fault60_10(:,1:4);
%sdata=zscore1(B1,meanx,stdx);
sdata=zscore1(B1);
t2_N2=tstat1(sdata,pc,latent,pc_used);
q_N2=qstat(sdata,pc,pc_used);


[rows cols]=size(B);

[rows2 cols2]=size(B1);

if rows<=rows2;
    rows =rows;
else
    rows=rows2;
end;
```

```
for j=start:rows;
    if t2_N2(j)>=TLmax_Vec(j);
       TF(j)=1;
    else
        TF(j)=0;
    end;
    if q_N2(j)>=QLmax_Vec(j);
       QF(j)=1;
    else
        QF(j)=0;
    end;
end;


figure;

subplot(3,1,1)
plot(t2_N2(start:rows),'*r');
hold on;
plot(t2(start:rows),'*b');
hold on;
plot(TLmax_Vec,'k');
title('T2 stats for simulated and actual data');
legend('Actual t2','Simulated t2 stats');
xlabel('time step');
ylabel('t2 stat value');


subplot(3,1,2)
plot(q_N2(start:rows),'*r');
hold on;
plot(q(start:rows),'*b');
hold on;
plot(QLmax_Vec,'k');
title('Q stats for simulated and actual data');
legend('Actual Q stats','Simulated Q');
xlabel('time step');
ylabel('Q stat value');

subplot(3,1,3)
plot(TF(start:rows),'or');
hold on;
plot(QF(start:rows),'ob');
title('Fault Detection Results for Q-stat and t2-stat');
legend('t2 stat results','q stat results');
xlabel('time step');
ylabel('Fault verdict');


% _____________________ Data Being tested_____________________
%             Loop Data Faults RVB Profile 1
```

```
%_____________________________________________________________
B1=Prof1_RVB_10(:,1:4);
%sdata=zscore1(B1,meanx,stdx);
sdata=zscore1(B1);
t2_N2=tstat1(sdata,pc,latent,pc_used);
q_N2=qstat(sdata,pc,pc_used);


[rows cols]=size(B);

[rows2 cols2]=size(B1);

if rows<=rows2;
    rows =rows;
else
    rows=rows2;
end;


for j=start:rows;
    if t2_N2(j)>=TLmax_Vec(j);
       TF(j)=1;
    else
        TF(j)=0;
    end;
    if q_N2(j)>=QLmax_Vec(j);
       QF(j)=1;
    else
        QF(j)=0;
    end;
end;


figure;

subplot(3,1,1)
plot(t2_N2(start:rows),'*r');
hold on;
plot(t2(start:rows),'*b');
hold on;
plot(TLmax_Vec,'k');
title('T2 stats for simulated and actual data');
legend('Actual t2','Simulated t2 stats');
xlabel('time step');
ylabel('t2 stat value');


subplot(3,1,2)
plot(q_N2(start:rows),'*r');
hold on;
plot(q(start:rows),'*b');
hold on;
plot(QLmax_Vec,'k');
```

```
title('Q stats for simulated and actual data');
legend('Actual Q stats','Simulated Q');
xlabel('time step');
ylabel('Q stat value');

subplot(3,1,3)
plot(TF(start:rows),'or');
hold on;
plot(QF(start:rows),'ob');
title('Fault Detection Results for Q-stat and t2-stat');
legend('t2 stat results','q stat results');
xlabel('time step');
ylabel('Fault verdict');

% _____________________ Data Being tested_____________________
%             Loop Data Faults RVB Profile 2
%_____________________________________________________________
B1=Prof2_RVB_10(:,1:4);
%sdata=zscore1(B1,meanx,stdx);
sdata=zscore1(B1);
t2_N2=tstat1(sdata,pc,latent,pc_used);
q_N2=qstat(sdata,pc,pc_used);


[rows cols]=size(B);

[rows2 cols2]=size(B1);

if rows<=rows2;
    rows =rows;
else
    rows=rows2;
end;


for j=start:rows;
    if t2_N2(j)>=TLmax_Vec(j);
       TF(j)=1;
    else
        TF(j)=0;
    end;
    if q_N2(j)>=QLmax_Vec(j);
       QF(j)=1;
    else
        QF(j)=0;
    end;
end;


figure;

subplot(3,1,1)
plot(t2_N2(start:rows),'*r');
```

```
hold on;
plot(t2(start:rows),'*b');
hold on;
plot(TLmax_Vec,'k');
title('T2 stats for simulated and actual data');
legend('Actual t2','Simulated t2 stats');
xlabel('time step');
ylabel('t2 stat value');


subplot(3,1,2)
plot(q_N2(start:rows),'*r');
hold on;
plot(q(start:rows),'*b');
hold on;
plot(QLmax_Vec,'k');
title('Q stats for simulated and actual data');
legend('Actual Q stats','Simulated Q');
xlabel('time step');
ylabel('Q stat value');

subplot(3,1,3)
plot(TF(start:rows),'or');
hold on;
plot(QF(start:rows),'ob');
title('Fault Detection Results for Q-stat and t2-stat');
legend('t2 stat results','q stat results');
xlabel('time step');
ylabel('Fault verdict');


% _____________________ Data Being tested_____________________
%             Loop Data Faults pump fail Profile 1
%_____________________________________________________________
B1=Prof1_pumpfail_10(:,1:4);
%sdata=zscore1(B1,meanx,stdx);
sdata=zscore1(B1);
t2_N2=tstat1(sdata,pc,latent,pc_used);
q_N2=qstat(sdata,pc,pc_used);


[rows cols]=size(B);

[rows2 cols2]=size(B1);

if rows<=rows2;
    rows =rows;
else
    rows=rows2;
end;


for j=start:rows;
```

```
    if t2_N2(j)>=TLmax_Vec(j);
       TF(j)=1;
    else
        TF(j)=0;
    end;
    if q_N2(j)>=QLmax_Vec(j);
       QF(j)=1;
    else
        QF(j)=0;
    end;
end;


figure;

subplot(3,1,1)
plot(t2_N2(start:rows),'*r');
hold on;
plot(t2(start:rows),'*b');
hold on;
plot(TLmax_Vec,'k');
title('T2 stats for simulated and actual data');
legend('Actual t2','Simulated t2 stats');
xlabel('time step');
ylabel('t2 stat value');


subplot(3,1,2)
plot(q_N2(start:rows),'*r');
hold on;
plot(q(start:rows),'*b');
hold on;
plot(QLmax_Vec,'k');
title('Q stats for simulated and actual data');
legend('Actual Q stats','Simulated Q');
xlabel('time step');
ylabel('Q stat value');

subplot(3,1,3)
plot(TF(start:rows),'or');
hold on;
plot(QF(start:rows),'ob');
title('Fault Detection Results for Q-stat and t2-stat');
legend('t2 stat results','q stat results');
xlabel('time step');
ylabel('Fault verdict');


% _____________________ Data Being tested_____________________
%             Loop Data Faults pump fail Profile 2
%_____________________________________________________________
B1=Prof2_pumpfail_10(:,1:4);
%sdata=zscore1(B1,meanx,stdx);
sdata=zscore1(B1);
```

```
t2_N2=tstat1(sdata,pc,latent,pc_used);
q_N2=qstat(sdata,pc,pc_used);


[rows cols]=size(B);

[rows2 cols2]=size(B1);

if rows<=rows2;
    rows =rows;
else
    rows=rows2;
end;


for j=start:rows;
    if t2_N2(j)>=TLmax_Vec(j);
       TF(j)=1;
    else
        TF(j)=0;
    end;
    if q_N2(j)>=QLmax_Vec(j);
       QF(j)=1;
    else
        QF(j)=0;
    end;
end;


figure;

subplot(3,1,1)
plot(t2_N2(start:rows),'*r');
hold on;
plot(t2(start:rows),'*b');
hold on;
plot(TLmax_Vec,'k');
title('T2 stats for simulated and actual data');
legend('Actual t2','Simulated t2 stats');
xlabel('time step');
ylabel('t2 stat value');


subplot(3,1,2)
plot(q_N2(start:rows),'*r');
hold on;
plot(q(start:rows),'*b');
hold on;
plot(QLmax_Vec,'k');
title('Q stats for simulated and actual data');
legend('Actual Q stats','Simulated Q');
xlabel('time step');
ylabel('Q stat value');
```

```
subplot(3,1,3)
plot(TF(start:rows),'or');
hold on;
plot(QF(start:rows),'ob');
title('Fault Detection Results for Q-stat and t2-stat');
legend('t2 stat results','q stat results');
xlabel('time step');
ylabel('Fault verdict');



%_____________________________________________________________________
%                    Data Comparison
%_____________________________________________________________________



clear;
load profile1;
load profile2;
load Prof1_RVB_10;
load Prof2_RVB_10;
load Prof1_fault60_10;
load Prof2_fault60_10;
load Prof1_pumpfail_10;
load Prof2_pumpfail_10;

    [rows cols]=size(profile1);
    [rowsRVB1 colsRVB1]=size(Prof1_RVB_10);
    [rowsF1 colsF1]=size(Prof1_fault60_10);
    [rowsP1 colsP1]=size(Prof1_pumpfail_10);

    RVB1n=min([rows rowsRVB1]);
    F1n=min([rows rowsF1]);
    P1n=min([rows rowsP1]);

    [rows cols]=size(profile2);
    [rowsRVB2 colsRVB1]=size(Prof2_RVB_10);
    [rowsF2 colsF1]=size(Prof2_fault60_10);
    [rowsP2 colsP1]=size(Prof2_pumpfail_10);

    RVB2n=min([rows rowsRVB2]);
    F2n=min([rows rowsF2]);
    P2n=min([rows rowsP2]);

for i=1:10;
    figure;
    subplot(2,1,1);
    plot(profile1(:,i),'k');
    hold on
    plot(Prof1_RVB_10(:,i),'b');
    hold on
```

```
    plot(Prof1_fault60_10(:,i),'r');
    hold on
    plot(Prof1_pumpfail_10(:,i),'g');
    title('Fault comparisons');
    legend('Normal','Return Valve Blocked','60mm drift','pumpfail');

    subplot(2,1,2);
    plot(profile1(1:RVB1n,i)-Prof1_RVB_10(1:RVB1n,i),'b');
    hold on
    plot(profile1(1:F1n,i)-Prof1_fault60_10(1:F1n,i),'r');
    hold on
    plot(profile1(1:P1n,i)-Prof1_pumpfail_10(1:P1n,i),'g');
    title(' Residuals for faults');
    legend('Return Valve Blocked','60mm drift','pumpfail');

end;
for i=1:10;

    figure;
    subplot(2,1,1);
    plot(profile2(300:8000,i),'k');
    hold on
    plot(Prof2_RVB_10(300:8000,i),'b');
    hold on
    plot(Prof2_fault60_10(300:8000,i),'r');
    hold on
    plot(Prof2_pumpfail_10(300:8000,i),'g');
    title('Fault comparisons');
    legend('Normal','Return Valve Blocked','60mm drift','pumpfail');

    subplot(2,1,2);
    plot(profile2(300:8000,i)-Prof2_RVB_10(300:8000,i),'b');
    hold on
    plot(profile2(300:8000,i)-Prof2_fault60_10(300:8000,i),'r');
    hold on
    plot(profile2(300:8000,i)-Prof2_pumpfail_10(300:8000,i),'g');
    title(' Residuals for faults');
    legend('Return Valve Blocked','60mm drift','pumpfail');
end;

clear;
load Profile1_simulator;
load Profile2_simulator;

P1_L=profile1_loop(300:8300,2:11);
P2_L=profile2_loop(300:13000,2:11);
P1_S=[profile1_simulink_reduced(300:8300,2:9)
profile1_simulink_reduced(300:8300,10:11)];
P2_S=[profile2_simulink_reduced(300:13000,2:9)
profile2_simulink_reduced(300:13000,10:11)];

for i=1:10;
    figure;
    subplot(2,1,1);
```

```
    plot(P1_L(:,i),'k');
    hold on
    plot(P1_S(:,i),'b');
    title('Simulink model comparisons');
    legend('Loop','Simulated');

    subplot(2,1,2);
    plot(P1_L(:,i)-P1_S(:,i),'b');
    title(' Difference between simulated and actual loop data');
    legend('Residual');
end;

for i=1:10;
    figure;
    subplot(2,1,1);
    plot(P2_L(:,i),'k');
    hold on
    plot(P2_S(:,i),'b');
    title('Simulink model comparisons');
    legend('Loop','Simulated');

    subplot(2,1,2);
    plot(P2_L(:,i)-P2_S(:,i),'b');
    title(' Difference between simlated and actual loop data');
    legend('Residual');
end;
```

```
%
%_____________________________________________________________
%
%_____________________________________________________________
% Adding faults to the lowfid data
% Negative 5% drift
%_____________________________________________________________
lowfd_neg5v6=[];
faultrw=[];

[rowshf colhf]=size(lowfid);
```

The below code was used for fault addition and PCA ECM testing.

```
load lowfid;
load highfid;

lowfids=zscore1(lowfid);
```

```
meanl=mean(lowfid);
stdl=std(lowfid);


[tlimit,qlimit,pc1,latent,t2,q]=PCAfault_limit2(lowfids,3);
qlimitvec=[];
qmaxvec=[];
qhighvec=[];
qmaxlowfid=max(q);
tlimitvec=[];
tmaxvec=[];
thighvec=[];
tmaxlowfid=max(t2);

t=lowfids*pc1;

[rows cols]=size(t);

qlimitvec=qlimit*ones(rows,1);
tlimitvec=tlimit*ones(rows,1);
qmaxvec=qmaxlowfid*ones(rows,1);
tmaxvec=tmaxlowfid*ones(rows,1);
timesteps=[1:1:rows];
qhighvec=qmaxvec;

t1limit=0.95*max(abs(t(:,1)));
t2limit=0.95*max(abs(t(:,2)));
t3limit=0.95*max(abs(t(:,3)));

figure;
subplot(2,1,1);
plot(timesteps,q,'*r');
hold on;
plot(timesteps,qlimitvec,'b');
hold on;
plot(timesteps,qmaxvec,'g');
legend('q values','qlimit','qmax');
title('timesteps vs Q-stat');
xlabel('timesteps');
ylabel('Q-stat');
axis ([-10 13000 -max(q)*0.1 max(q)*1.1]);

subplot(2,1,2);
plot(timesteps,t2,'*r');
hold on;
plot(timesteps,tlimitvec,'b');
hold on;
plot(timesteps,tmaxvec,'g');
legend('t2 values','tlimit','t2max');
title('timesteps vs t^2-stat');
xlabel('timesteps');
ylabel('t^2-stat');
axis ([-10 13000 -max(t2)*0.1 max(t2)*1.1]);
```

```
%_____________________________________________________________
%
%
%         Negative Drifts
%

thirddata=ceil(1/3*rowshf);

faultrw(:,1)=lowfid(:,6);
for i=thirddata:rowshf;

    % 5% increase per step

    faultrw(i,1)=faultrw(i,1)-faultrw(i,1)*i*.05/rowshf;

end;

figure;

plot(faultrw(:,1));
hold on;
plot(lowfid(:,6),'r');
hold off;
legend('faulty data','original data','location','NorthEast');
xlabel('Sample number');
ylabel('Standardized Value');
title('Fault Negative 5% drift vs Actual data', 'FontWeight','Bold');


nonfaultvect=[];

nonfaultvect=lowfid;

nonfaultvect(:,6)=[];

lowfd_neg5v6=[ nonfaultvect faultrw(:,1)];

save lowfd_neg5v6;

figure;

plot(lowfd_neg5v6(:,6)-highfid(:,6));

legend('Difference between fault and actual data','location','NorthEast');
xlabel('Sample number');
ylabel('difference');
title('Difference between Fault Negative 5% drift and Actual data',
'FontWeight','Bold');

%_____________________________________________________________________
%_____________________________________________________________________
```

```
%_____________________________________________________
% Adding faults to the lowfid data
% Negative 10% drift
%_____________________________________________________
lowfd_neg10v6=[];
faultrw=[];

[rowshf colhf]=size(lowfid);


thirddata=ceil(1/3*rowshf);

faultrw(:,1)=lowfid(:,6);
for i=thirddata:rowshf;

    % 10% increase per step

    faultrw(i,1)=faultrw(i,1)-faultrw(i,1)*i*.10/rowshf;

end;

figure;

plot(faultrw(:,1));
hold on;
plot(lowfid(:,6),'r');
hold off;
legend('faulty data','original data','location','NorthEast');
xlabel('Sample number');
ylabel('Standardized Value');
title('Fault Negative 10% drift vs Actual data', 'FontWeight','Bold');


nonfaultvect=[];

nonfaultvect=lowfid;

nonfaultvect(:,6)=[];

lowfd_neg10v6=[ nonfaultvect faultrw(:,1)];

save lowfd_neg10v6;

figure;

plot(lowfd_neg10v6(:,6)-highfid(:,6));

legend('Difference between fault and actual data','location','NorthEast');
```

```
xlabel('Sample number');
ylabel('difference');
title('Difference between Fault Negative 10% drift and Actual data',
'FontWeight','Bold');

%_____________________________________________________________________
%_____________________________________________________________________

%_____________________________________________________
% Adding faults to the lowfid data
% Negative 20% drift
%_____________________________________________________
lowfd_neg20v6=[];
faultrw=[];

[rowshf colhf]=size(lowfid);


thirddata=ceil(1/3*rowshf);

faultrw(:,1)=lowfid(:,6);
for i=thirddata:rowshf;

    %20% increase per step

    faultrw(i,1)=faultrw(i,1)-faultrw(i,1)*i*.20/rowshf;

end;

figure;

plot(faultrw(:,1));
hold on;
plot(lowfid(:,6),'r');
hold off;
legend('faulty data','original data','location','NorthEast');
xlabel('Sample number');
ylabel('Standardized Value');
title('Fault Negative 20% drift vs Actual data', 'FontWeight','Bold');


nonfaultvect=[];

nonfaultvect=lowfid;

nonfaultvect(:,6)=[];

lowfd_neg20v6=[ nonfaultvect faultrw(:,1)];

save lowfd_neg20v6;

figure;
```

```
plot(lowfd_neg20v6(:,6)-highfid(:,6));

legend('Difference between fault and actual data','location','NorthEast');
xlabel('Sample number');
ylabel('difference');
title('Difference between Fault Negative 20% drift and Actual data',
'FontWeight','Bold');

%_____________________________________________________________________________
%_____________________________________________________________________________









%_____________________________________________________________
%
%
%         Positive Drifts
%
%
%_____________________________________________________________
%
%_____________________________________________________
% Adding faults to the lowfid data
% Positive 5% drift
%_____________________________________________________
lowfd_5v6=[];
faultrw=[];

[rowshf colhf]=size(lowfid);


thirddata=ceil(1/3*rowshf);

faultrw(:,1)=lowfid(:,6);
for i=thirddata:rowshf;

    % 5% increase per step

    faultrw(i,1)=faultrw(i,1)+faultrw(i,1)*i*.05/rowshf;

end;

figure;

plot(faultrw(:,1));
hold on;
plot(lowfid(:,6),'r');
hold off;
```

```
legend('faulty data','original data','location','NorthEast');
xlabel('Sample number');
ylabel('Standardized Value');
title('Fault 5% drift vs Actual data', 'FontWeight','Bold');


nonfaultvect=[];

nonfaultvect=lowfid;

nonfaultvect(:,6)=[];

lowfd_5v6=[ nonfaultvect faultrw(:,1)];

save lowfd_5v6;

figure;

plot(lowfd_5v6(:,6)-highfid(:,6));

legend('Difference between fault and actual data','location','NorthEast');
xlabel('Sample number');
ylabel('difference');
title('Difference between Fault 5% drift and Actual data',
'FontWeight','Bold');

%_____________________________________________________________________________
%_____________________________________________________________________________



%_____________________________________________________
% Adding faults to the lowfid data
% Positive 10% drift
%_____________________________________________________
lowfd_10v6=[];
faultrw=[];

[rowshf colhf]=size(lowfid);


thirddata=ceil(1/3*rowshf);

faultrw(:,1)=lowfid(:,6);
for i=thirddata:rowshf;

    % 10% increase per step

    faultrw(i,1)=faultrw(i,1)+faultrw(i,1)*i*.10/rowshf;

end;
```

```
figure;

plot(faultrw(:,1));
hold on;
plot(lowfid(:,6),'r');
hold off;
legend('faulty data','original data','location','NorthEast');
xlabel('Sample number');
ylabel('Standardized Value');
title('Fault 10% drift vs Actual data', 'FontWeight','Bold');


nonfaultvect=[];

nonfaultvect=lowfid;

nonfaultvect(:,6)=[];

lowfd_10v6=[ nonfaultvect faultrw(:,1)];

save lowfd_10v6;

figure;

plot(lowfd_10v6(:,6)-highfid(:,6));

legend('Difference between fault and actual data','location','NorthEast');
xlabel('Sample number');
ylabel('difference');
title('Difference between Fault 10% drift and Actual data',
'FontWeight','Bold');

%_____________________________________________________________________________
%_____________________________________________________________________________

%_____________________________________________________
% Adding faults to the lowfid data
% Positve 20% drift
%_____________________________________________________
lowfd_20v6=[];
faultrw=[];

[rowshf colhf]=size(lowfid);


thirddata=ceil(1/3*rowshf);

faultrw(:,1)=lowfid(:,6);
for i=thirddata:rowshf;

    %20% increase per step
```

```
    faultrw(i,1)=faultrw(i,1)+faultrw(i,1)*i*.20/rowshf;

end;

figure;

plot(faultrw(:,1));
hold on;
plot(lowfid(:,6),'r');
hold off;
legend('faulty data','original data','location','NorthEast');
xlabel('Sample number');
ylabel('Standardized Value');
title('Fault 20% drift vs Actual data', 'FontWeight','Bold');


nonfaultvect=[];

nonfaultvect=lowfid;

nonfaultvect(:,6)=[];

lowfd_20v6=[ nonfaultvect faultrw(:,1)];

save lowfd_20v6;

figure;

plot(lowfd_20v6(:,6)-highfid(:,6));

legend('Difference between fault and actual data','location','NorthEast');
xlabel('Sample number');
ylabel('difference');
title('Difference between Fault 20% drift and Actual data',
'FontWeight','Bold');

%_____________________________________________________________________________
%_____________________________________________________________________________









%_____________________________________________________________
%
%
%        Negative Bias
```

```
%
%
%_____________________________________________________________
%
%_____________________________________________________________
% Adding faults to the lowfid data
% Negative 5% bias
%_____________________________________________________________
lowfd_negB5v6=[];
faultrw=[];

[rowshf colhf]=size(lowfid);


thirddata=ceil(1/3*rowshf);

faultrw(:,1)=lowfid(:,6);
for i=thirddata:rowshf;

    % 5% increase per step

    faultrw(i,1)=faultrw(i,1)-min(faultrw(:,1))*.05;

end;

figure;

plot(faultrw(:,1));
hold on;
plot(lowfid(:,6),'r');
hold off;
legend('faulty data','original data','location','NorthEast');
xlabel('Sample number');
ylabel('Standardized Value');
title('Fault Negative 5% bias vs Actual data', 'FontWeight','Bold');


nonfaultvect=[];

nonfaultvect=lowfid;

nonfaultvect(:,6)=[];

lowfd_negB5v6=[ nonfaultvect faultrw(:,1)];

save lowfd_negB5v6;

figure;

plot(lowfd_negB5v6(:,6)-highfid(:,6));

legend('Difference between fault and actual data','location','NorthEast');
```

```
xlabel('Sample number');
ylabel('difference');
title('Difference between Fault Negative 5% bias and Actual data',
'FontWeight','Bold');

%_____________________________________________________________________
%_____________________________________________________________________



%_____________________________________________________
% Adding faults to the lowfid data
% Negative 10% bias
%_____________________________________________________
lowfd_negB10v6=[];
faultrw=[];

[rowshf colhf]=size(lowfid);


thirddata=ceil(1/3*rowshf);

faultrw(:,1)=lowfid(:,6);
for i=thirddata:rowshf;

    % 10% increase per step

    faultrw(i,1)=faultrw(i,1)-min(faultrw(:,1))*.10;

end;

figure;

plot(faultrw(:,1));
hold on;
plot(lowfid(:,6),'r');
hold off;
legend('faulty data','original data','location','NorthEast');
xlabel('Sample number');
ylabel('Standardized Value');
title('Fault Negative 10% bias vs Actual data', 'FontWeight','Bold');


nonfaultvect=[];

nonfaultvect=lowfid;

nonfaultvect(:,6)=[];

lowfd_negB10v6=[ nonfaultvect faultrw(:,1)];

save lowfd_negB10v6;
```

```
figure;

plot(lowfd_negB10v6(:,6)-highfid(:,6));

legend('Difference between fault and actual data','location','NorthEast');
xlabel('Sample number');
ylabel('difference');
title('Difference between Fault Negative 10% bias and Actual data',
'FontWeight','Bold');

%_____________________________________________________________________________
%_____________________________________________________________________________

%_____________________________________________________
% Adding faults to the lowfid data
% Negative 20% bias
%_____________________________________________________
lowfd_negB20v6=[];
faultrw=[];

[rowshf colhf]=size(lowfid);


thirddata=ceil(1/3*rowshf);

faultrw(:,1)=lowfid(:,6);
for i=thirddata:rowshf;

    %20% increase per step

    faultrw(i,1)=faultrw(i,1)-min(faultrw(:,1))*.20;

end;

figure;

plot(faultrw(:,1));
hold on;
plot(lowfid(:,6),'r');
hold off;
legend('faulty data','original data','location','NorthEast');
xlabel('Sample number');
ylabel('Standardized Value');
title('Fault Negative 20% bias vs Actual data', 'FontWeight','Bold');


nonfaultvect=[];

nonfaultvect=lowfid;

nonfaultvect(:,6)=[];
```

```
lowfd_negB20v6=[ nonfaultvect faultrw(:,1)];

save lowfd_negB20v6;

figure;

plot(lowfd_negB20v6(:,6)-highfid(:,6));

legend('Difference between fault and actual data','location','NorthEast');
xlabel('Sample number');
ylabel('difference');
title('Difference between Fault Negative 20% bias and Actual data',
'FontWeight','Bold');

%_____________________________________________________________________________
%_____________________________________________________________________________









%_____________________________________________________________
%
%
%        Positive Bias
%
%
%_____________________________________________________________
%
%_____________________________________________________________
% Adding faults to the lowfid data
% Positive 5% bias
%_____________________________________________________________
lowfd_B5v6=[];
faultrw=[];

[rowshf colhf]=size(lowfid);


thirddata=ceil(1/3*rowshf);

faultrw(:,1)=lowfid(:,6);
for i=thirddata:rowshf;

    % 5% increase per step

    faultrw(i,1)=faultrw(i,1)+min(faultrw(:,1))*.05;

end;
```

```
figure;

plot(faultrw(:,1));
hold on;
plot(lowfid(:,6),'r');
hold off;
legend('faulty data','original data','location','NorthEast');
xlabel('Sample number');
ylabel('Standardized Value');
title('Fault 5% bias vs Actual data', 'FontWeight','Bold');


nonfaultvect=[];

nonfaultvect=lowfid;

nonfaultvect(:,6)=[];

lowfd_B5v6=[ nonfaultvect faultrw(:,1)];

save lowfd_B5v6;

figure;

plot(lowfd_B5v6(:,6)-highfid(:,6));

legend('Difference between fault and actual data','location','NorthEast');
xlabel('Sample number');
ylabel('difference');
title('Difference between Fault 5% bias and Actual data',
'FontWeight','Bold');

%_____________________________________________________________________________
%_____________________________________________________________________________



%_____________________________________________________
% Adding faults to the lowfid data
% Positive 10% bias
%_____________________________________________________
lowfd_B10v6=[];
faultrw=[];

[rowshf colhf]=size(lowfid);


thirddata=ceil(1/3*rowshf);

faultrw(:,1)=lowfid(:,6);
for i=thirddata:rowshf;
```

```
    % 10% increase per step

    faultrw(i,1)=faultrw(i,1)+min(faultrw(:,1))*.10;

end;

figure;

plot(faultrw(:,1));
hold on;
plot(lowfid(:,6),'r');
hold off;
legend('faulty data','original data','location','NorthEast');
xlabel('Sample number');
ylabel('Standardized Value');
title('Fault 10% bias vs Actual data', 'FontWeight','Bold');


nonfaultvect=[];

nonfaultvect=lowfid;

nonfaultvect(:,6)=[];

lowfd_B10v6=[ nonfaultvect faultrw(:,1)];

save lowfd_B10v6;

figure;

plot(lowfd_B10v6(:,6)-highfid(:,6));

legend('Difference between fault and actual data','location','NorthEast');
xlabel('Sample number');
ylabel('difference');
title('Difference between Fault 10% bias and Actual data',
'FontWeight','Bold');

%_____________________________________________________________________
%_____________________________________________________________________

%_____________________________________________________
% Adding faults to the lowfid data
% Positve 20% bias
%_____________________________________________________
lowfd_B20v6=[];
faultrw=[];

[rowshf colhf]=size(lowfid);
```

```
thirddata=ceil(1/3*rowshf);

faultrw(:,1)=lowfid(:,6);
for i=thirddata:rowshf;

    %20% increase per step

    faultrw(i,1)=faultrw(i,1)+min(faultrw(:,1))*.20;

end;

figure;

plot(faultrw(:,1));
hold on;
plot(lowfid(:,6),'r');
hold off;
legend('faulty data','original data','location','NorthEast');
xlabel('Sample number');
ylabel('Standardized Value');
title('Fault 20% bias vs Actual data', 'FontWeight','Bold');


nonfaultvect=[];

nonfaultvect=lowfid;

nonfaultvect(:,6)=[];

lowfd_B20v6=[ nonfaultvect faultrw(:,1)];

save lowfd_B20v6;

figure;

plot(lowfd_B20v6(:,6)-highfid(:,6));

legend('Difference between fault and actual data','location','NorthEast');
xlabel('Sample number');
ylabel('difference');
title('Difference between Fault 20% bias and Actual data',
'FontWeight','Bold');

%_____________________________________________________________________
%_____________________________________________________________________


% ---------------------------------------------------------------------
%----------------------------------------------------------------------
%
%
%                        Looking at the Faults
```

```
%_____________________________________________________________________________


figure;
subplot(3,1,1);
plot(lowfid(:,6),'b');
hold on;
plot(lowfd_neg5v6(:,6),'g');
hold on;
plot(lowfd_neg10v6(:,6),'g');
hold on;
plot(lowfd_neg20v6(:,6),'g');
hold on;
plot(lowfd_5v6(:,6),'g');
hold on;
plot(lowfd_10v6(:,6),'g');
hold on;
plot(lowfd_20v6(:,6),'g');
hold off;
title('Faulty Drift Data Sets');
xlabel('time step');
ylabel('Value');

subplot(3,1,2);
plot(lowfid(:,6),'b');
hold on;
plot(lowfd_negB5v6(:,6),'r');
hold on;
plot(lowfd_negB10v6(:,6),'r');
hold on;
plot(lowfd_negB20v6(:,6),'r');
hold on;
plot(lowfd_B5v6(:,6),'r');
hold on;
plot(lowfd_B10v6(:,6),'r');
hold on;
plot(lowfd_B20v6(:,6),'r');
hold off;

subplot(3,1,3);
plot(lowfid(:,6),'b');
hold on;
plot(highfid(:,6),'k');
title('Faulty Bias Data Sets');
xlabel('time step');
ylabel('Value');

%_____________________________________________________


[rows cols]=size(lowfid);
```

```
timesteps=[1:1:rows];


%_____________________________________________________________
%
%
%          Looking at the change in the Q and t^2 stats
%
%
%_____________________________________________________________



%_____________________________________________
%
%
%           t^2 and q fault error for faulty data
%_____________________________________________

lowfd_5v6s=zscore1(lowfd_5v6,meanl,stdl);

q=qstat(lowfd_5v6s,pc1,3);
t2=tstat1(lowfd_5v6s,pc1,latent,3);

%  Percent Change in q-stat and t^2 stat
q5v6=max(q)/qmaxlowfid-1
t5v6=max(t2)/tmaxlowfid-1

figure;
subplot(3,1,1);
plot(timesteps,q,'*r');
hold on;
plot(timesteps,qhighvec,'k');
hold off;
legend('q values');
title('timesteps vs Q-stat');
xlabel('timesteps');
ylabel('Q-stat');
axis ([-10 13000 -max(q)*0.1 max(q)*1.1]);

subplot(3,1,2);
plot(timesteps,t2,'*r');
hold on;
plot(timesteps,tmaxvec,'k');
legend('t values','tlimit','tmax');
hold off;
legend('t^2 values');
title('timesteps vs t^2-stat');
xlabel('timesteps');
ylabel('t^2-stat');
axis ([-10 13000 -max(t2)*0.1 max(t2)*1.1]);
```

```
subplot(3,1,3)
plot(timesteps, lowfd_5v6s(:,6)-lowfids(:,6),'b');

title('Difference between faultdata and lowfid data');
legend('difference');
xlabel('timesteps');
ylabel('residual');




lowfd_10v6s=zscore1(lowfd_10v6,meanl,stdl);

q=qstat(lowfd_10v6s,pc1,3);
t2=tstat1(lowfd_10v6s,pc1,latent,3);

%  Percent Change in q-stat and t^2 stat
q10v6=max(q)/qmaxlowfid-1
t10v6=max(t2)/tmaxlowfid-1

figure;
subplot(3,1,1);
plot(timesteps,q,'*r');
hold on;
plot(timesteps,qhighvec,'k');
hold off;
legend('q values');
title('timesteps vs Q-stat');
xlabel('timesteps');
ylabel('Q-stat');
axis ([-10 13000 -max(q)*0.1 max(q)*1.1]);

subplot(3,1,2);
plot(timesteps,t2,'*r');
hold on;
plot(timesteps,tmaxvec,'k');
legend('t values','tlimit','tmax');
hold off;
legend('t^2 values');
title('timesteps vs t^2-stat');
xlabel('timesteps');
ylabel('t^2-stat');
axis ([-10 13000 -max(t2)*0.1 max(t2)*1.1]);

subplot(3,1,3)
plot(timesteps, lowfd_10v6s(:,6)-lowfids(:,6),'b');

title('Difference between faultdata and lowfid data');
legend('difference');
xlabel('timesteps');
ylabel('residual');
```

```
lowfd_20v6s=zscore1(lowfd_20v6,meanl,stdl);

q=qstat(lowfd_20v6s,pc1,3);
t2=tstat1(lowfd_20v6s,pc1,latent,3);

%  Percent Change in q-stat and t^2 stat
q20v6=max(q)/qmaxlowfid-1
t20v6=max(t2)/tmaxlowfid-1

figure;
subplot(3,1,1);
plot(timesteps,q,'*r');
hold on;
plot(timesteps,qhighvec,'k');
hold off;
legend('q values');
title('timesteps vs Q-stat');
xlabel('timesteps');
ylabel('Q-stat');
axis ([-10 13000 -max(q)*0.1 max(q)*1.1]);

subplot(3,1,2);
plot(timesteps,t2,'*r');
hold on;
plot(timesteps,tmaxvec,'k');
legend('t values','tlimit','tmax');
hold off;
legend('t^2 values');
title('timesteps vs t^2-stat');
xlabel('timesteps');
ylabel('t^2-stat');
axis ([-10 13000 -max(t2)*0.1 max(t2)*1.1]);

subplot(3,1,3)
plot(timesteps, lowfd_20v6s(:,6)-lowfids(:,6),'b');

title('Difference between faultdata and lowfid data');
legend('difference');
xlabel('timesteps');
ylabel('residual');


%_____________________________________________________________
%_____________________________________________________________

lowfd_neg5v6s=zscore1(lowfd_neg5v6,meanl,stdl);

q=qstat(lowfd_neg5v6s,pc1,3);
t2=tstat1(lowfd_neg5v6s,pc1,latent,3);

%  Percent Change in q-stat and t^2 stat
qneg5v6=max(q)/qmaxlowfid-1
tneg5v6=max(t2)/tmaxlowfid-1
```

```
figure;
subplot(3,1,1);
plot(timesteps,q,'*r');
hold on;
plot(timesteps,qhighvec,'k');
hold off;
legend('q values');
title('timesteps vs Q-stat');
xlabel('timesteps');
ylabel('Q-stat');
axis ([-10 13000 -max(q)*0.1 max(q)*1.1]);

subplot(3,1,2);
plot(timesteps,t2,'*r');
hold on;
plot(timesteps,tmaxvec,'k');
legend('t values','tlimit','tmax');
hold off;
legend('t^2 values');
title('timesteps vs t^2-stat');
xlabel('timesteps');
ylabel('t^2-stat');
axis ([-10 13000 -max(t2)*0.1 max(t2)*1.1]);

subplot(3,1,3)
plot(timesteps, lowfd_neg5v6s(:,6)-lowfids(:,6),'b');

title('Difference between faultdata and lowfid data');
legend('difference');
xlabel('timesteps');
ylabel('residual');




lowfd_neg10v6s=zscore1(lowfd_neg10v6,meanl,stdl);

q=qstat(lowfd_neg10v6s,pc1,3);
t2=tstat1(lowfd_neg10v6s,pc1,latent,3);

%  Percent Change in q-stat and t^2 stat
qneg10v6=max(q)/qmaxlowfid-1
tneg10v6=max(t2)/tmaxlowfid-1

figure;
subplot(3,1,1);
plot(timesteps,q,'*r');
hold on;
plot(timesteps,qhighvec,'k');
hold off;
legend('q values');
title('timesteps vs Q-stat');
```

```
xlabel('timesteps');
ylabel('Q-stat');
axis ([-10 13000 -max(q)*0.1 max(q)*1.1]);

subplot(3,1,2);
plot(timesteps,t2,'*r');
hold on;
plot(timesteps,tmaxvec,'k');
legend('t values','tlimit','tmax');
hold off;
legend('t^2 values');
title('timesteps vs t^2-stat');
xlabel('timesteps');
ylabel('t^2-stat');
axis ([-10 13000 -max(t2)*0.1 max(t2)*1.1]);

subplot(3,1,3)
plot(timesteps, lowfd_neg10v6s(:,6)-lowfids(:,6),'b');

title('Difference between faultdata and lowfid data');
legend('difference');
xlabel('timesteps');
ylabel('residual');


lowfd_neg20v6s=zscore1(lowfd_neg20v6,meanl,stdl);

q=qstat(lowfd_neg20v6s,pc1,3);
t2=tstat1(lowfd_neg20v6s,pc1,latent,3);

%  Percent Change in q-stat and t^2 stat
qneg20v6=max(q)/qmaxlowfid-1
tneg20v6=max(t2)/tmaxlowfid-1

figure;
subplot(3,1,1);
plot(timesteps,q,'*r');
hold on;
plot(timesteps,qhighvec,'k');
hold off;
legend('q values');
title('timesteps vs Q-stat');
xlabel('timesteps');
ylabel('Q-stat');
axis ([-10 13000 -max(q)*0.1 max(q)*1.1]);

subplot(3,1,2);
plot(timesteps,t2,'*r');
hold on;
plot(timesteps,tmaxvec,'k');
legend('t values','tlimit','tmax');
hold off;
legend('t^2 values');
title('timesteps vs t^2-stat');
```

```
xlabel('timesteps');
ylabel('t^2-stat');
axis ([-10 13000 -max(t2)*0.1 max(t2)*1.1]);

subplot(3,1,3)
plot(timesteps, lowfd_neg20v6s(:,6)-lowfids(:,6),'b');

title('Difference between faultdata and lowfid data');
legend('difference');
xlabel('timesteps');
ylabel('residual');



%_____________________________________________________
%
%
%
%
% For Bias Results
%
%
%_____________________________________________________




lowfd_B5v6s=zscore1(lowfd_B5v6,meanl,stdl);

q=qstat(lowfd_B5v6s,pc1,3);
t2=tstat1(lowfd_B5v6s,pc1,latent,3);

%  Percent Change in q-stat and t^2 stat
qB5v6=max(q)/qmaxlowfid-1
tB5v6=max(t2)/tmaxlowfid-1

figure;
subplot(3,1,1);
plot(timesteps,q,'*r');
hold on;
plot(timesteps,qhighvec,'k');
hold off;
legend('q values');
title('timesteps vs Q-stat');
xlabel('timesteps');
ylabel('Q-stat');
axis ([-10 13000 -max(q)*0.1 max(q)*1.1]);

subplot(3,1,2);
plot(timesteps,t2,'*r');
hold on;
plot(timesteps,tmaxvec,'k');
```

```
legend('t values','tlimit','tmax');
hold off;
legend('t^2 values');
title('timesteps vs t^2-stat');
xlabel('timesteps');
ylabel('t^2-stat');
axis ([-10 13000 -max(t2)*0.1 max(t2)*1.1]);

subplot(3,1,3)
plot(timesteps, lowfd_B5v6s(:,6)-lowfids(:,6),'b');

title('Difference between faultdata and lowfid data');
legend('difference');
xlabel('timesteps');
ylabel('residual');




lowfd_B10v6s=zscore1(lowfd_B10v6,meanl,stdl);

q=qstat(lowfd_B10v6s,pc1,3);
t2=tstat1(lowfd_B10v6s,pc1,latent,3);

%  Percent Change in q-stat and t^2 stat
qB10v6=max(q)/qmaxlowfid-1
tB10v6=max(t2)/tmaxlowfid-1

figure;
subplot(3,1,1);
plot(timesteps,q,'*r');
hold on;
plot(timesteps,qhighvec,'k');
hold off;
legend('q values');
title('timesteps vs Q-stat');
xlabel('timesteps');
ylabel('Q-stat');
axis ([-10 13000 -max(q)*0.1 max(q)*1.1]);

subplot(3,1,2);
plot(timesteps,t2,'*r');
hold on;
plot(timesteps,tmaxvec,'k');
legend('t values','tlimit','tmax');
hold off;
legend('t^2 values');
title('timesteps vs t^2-stat');
xlabel('timesteps');
ylabel('t^2-stat');
axis ([-10 13000 -max(t2)*0.1 max(t2)*1.1]);

subplot(3,1,3)
```

```
plot(timesteps, lowfd_B10v6s(:,6)-lowfids(:,6),'b');

title('Difference between faultdata and lowfid data');
legend('difference');
xlabel('timesteps');
ylabel('residual');


lowfd_B20v6s=zscore1(lowfd_B20v6,meanl,stdl);

q=qstat(lowfd_B20v6s,pc1,3);
t2=tstat1(lowfd_B20v6s,pc1,latent,3);

%  Percent Change in q-stat and t^2 stat
qB20v6=max(q)/qmaxlowfid-1
tB20v6=max(t2)/tmaxlowfid-1

figure;
subplot(3,1,1);
plot(timesteps,q,'*r');
hold on;
plot(timesteps,qhighvec,'k');
hold off;
legend('q values');
title('timesteps vs Q-stat');
xlabel('timesteps');
ylabel('Q-stat');
axis ([-10 13000 -max(q)*0.1 max(q)*1.1]);

subplot(3,1,2);
plot(timesteps,t2,'*r');
hold on;
plot(timesteps,tmaxvec,'k');
legend('t values','tlimit','tmax');
hold off;
legend('t^2 values');
title('timesteps vs t^2-stat');
xlabel('timesteps');
ylabel('t^2-stat');
axis ([-10 13000 -max(t2)*0.1 max(t2)*1.1]);

subplot(3,1,3)
plot(timesteps, lowfd_B20v6s(:,6)-lowfids(:,6),'b');

title('Difference between faultdata and lowfid data');
legend('difference');
xlabel('timesteps');
ylabel('residual');


%_____________________________________________________________
%_____________________________________________________________
```

```
lowfd_negB5v6s=zscore1(lowfd_negB5v6,meanl,stdl);

q=qstat(lowfd_negB5v6s,pc1,3);
t2=tstat1(lowfd_negB5v6s,pc1,latent,3);

%  Percent Change in q-stat and t^2 stat
qnegB5v6=max(q)/qmaxlowfid-1
tnegB5v6=max(t2)/tmaxlowfid-1

figure;
subplot(3,1,1);
plot(timesteps,q,'*r');
hold on;
plot(timesteps,qhighvec,'k');
hold off;
legend('q values');
title('timesteps vs Q-stat');
xlabel('timesteps');
ylabel('Q-stat');
axis ([-10 13000 -max(q)*0.1 max(q)*1.1]);

subplot(3,1,2);
plot(timesteps,t2,'*r');
hold on;
plot(timesteps,tmaxvec,'k');
legend('t values','tlimit','tmax');
hold off;
legend('t^2 values');
title('timesteps vs t^2-stat');
xlabel('timesteps');
ylabel('t^2-stat');
axis ([-10 13000 -max(t2)*0.1 max(t2)*1.1]);

subplot(3,1,3)
plot(timesteps, lowfd_negB5v6s(:,6)-lowfids(:,6),'b');

title('Difference between faultdata and lowfid data');
legend('difference');
xlabel('timesteps');
ylabel('residual');




lowfd_negB10v6s=zscore1(lowfd_negB10v6,meanl,stdl);

q=qstat(lowfd_negB10v6s,pc1,3);
t2=tstat1(lowfd_negB10v6s,pc1,latent,3);

%  Percent Change in q-stat and t^2 stat
qnegB10v6=max(q)/qmaxlowfid-1
tnegB10v6=max(t2)/tmaxlowfid-1
```

```
figure;
subplot(3,1,1);
plot(timesteps,q,'*r');
hold on;
plot(timesteps,qhighvec,'k');
hold off;
legend('q values');
title('timesteps vs Q-stat');
xlabel('timesteps');
ylabel('Q-stat');
axis ([-10 13000 -max(q)*0.1 max(q)*1.1]);

subplot(3,1,2);
plot(timesteps,t2,'*r');
hold on;
plot(timesteps,tmaxvec,'k');
legend('t values','tlimit','tmax');
hold off;
legend('t^2 values');
title('timesteps vs t^2-stat');
xlabel('timesteps');
ylabel('t^2-stat');
axis ([-10 13000 -max(t2)*0.1 max(t2)*1.1]);

subplot(3,1,3)
plot(timesteps, lowfd_negB10v6s(:,6)-lowfids(:,6),'b');

title('Difference between faultdata and lowfid data');
legend('difference');
xlabel('timesteps');
ylabel('residual');


lowfd_negB20v6s=zscore1(lowfd_negB20v6,meanl,stdl);

q=qstat(lowfd_negB20v6s,pc1,3);
t2=tstat1(lowfd_negB20v6s,pc1,latent,3);

%  Percent Change in q-stat and t^2 stat
qnegB20v6=max(q)/qmaxlowfid-1
tnegB20v6=max(t2)/tmaxlowfid-1

figure;
subplot(3,1,1);
plot(timesteps,q,'*r');
hold on;
plot(timesteps,qhighvec,'k');
hold off;
legend('q values');
title('timesteps vs Q-stat');
xlabel('timesteps');
ylabel('Q-stat');
axis ([-10 13000 -max(q)*0.1 max(q)*1.1]);
```

```
subplot(3,1,2);
plot(timesteps,t2,'*r');
hold on;
plot(timesteps,tmaxvec,'k');
legend('t values','tlimit','tmax');
hold off;
legend('t^2 values');
title('timesteps vs t^2-stat');
xlabel('timesteps');
ylabel('t^2-stat');
axis ([-10 13000 -max(t2)*0.1 max(t2)*1.1]);

subplot(3,1,3)
plot(timesteps, lowfd_negB20v6s(:,6)-lowfids(:,6),'b');

title('Difference between faultdata and lowfid data');
legend('difference');
xlabel('timesteps');
ylabel('residual');

%_____________________________________________________
%               For Highfid data
%_____________________________________________________

highfids=zscore1(highfid,meanl,stdl);

q=qstat(highfids,pc1,3);
t2=tstat1(highfids,pc1,latent,3);

%  Percent Change in q-stat and t^2 stat
qhigh=max(q)/qmaxlowfid-1
thigh=max(t2)/tmaxlowfid-1

figure;
subplot(3,1,1);
plot(timesteps,q,'*r');
hold on;
plot(timesteps,qhighvec,'k');
hold off;
legend('q values');
title('timesteps vs Q-stat');
xlabel('timesteps');
ylabel('Q-stat');
axis ([-10 13000 -max(q)*0.1 max(q)*1.1]);

subplot(3,1,2);
plot(timesteps,t2,'*r');
hold on;
plot(timesteps,tmaxvec,'k');
legend('t values','tlimit','tmax');
hold off;
legend('t^2 values');
title('timesteps vs t^2-stat');
xlabel('timesteps');
```

```
ylabel('t^2-stat');
axis ([-10 13000 -max(t2)*0.1 max(t2)*1.1]);

subplot(3,1,3)
plot(timesteps, highfids(:,6)-lowfids(:,6),'b');

title('Difference between faultdata and lowfid data');
legend('difference');
xlabel('timesteps');
ylabel('residual');
```

Diagnostic MATLAB Code

```
% Fault Detection and Identification
clear
% Fault Detection portion

load NonFaultResid
load SystemFail_1aData
load SystemFail_1bData
load SystemFail_2aData
load SystemFail_2bData
load SystemFail_2cData


[rowsNF colsNF]=size(NonFaultResid);

Var=[];
SensVar=[];
for i=1:colsNF;
    SensVar=[];
    for j=1:rowsNF-10;
        SensVar=[SensVar var(NonFaultResid(j:j+10, i))];
    end;
    Var(i)=max(SensVar);
end;




residuals=SystemFail_2bData;



Tvect=[1.2 3.5 1.2 0.35 6 14 1.8 3.5 12 0.18;0.5*Var];


[rows cols]=size(residuals);
```

```
%NoFault=0;

for h=1:cols;
        if residuals(rows,h)>=Tvect(1,h)||residuals(rows-
2,h)>=Tvect(1,h)||residuals(rows-5,h)>=Tvect(1,h)||...
                residuals(rows-6,h)>=Tvect(1,h)||residuals(rows-
10,h)>=Tvect(1,h)||residuals(rows-12,h)>=Tvect(1,h);

            if var(residuals(rows-10:rows,h))>Tvect(2,h);
                FhypT(h)=3; %Nofault=Nofault+1;
            else
                FhypT(h)=1; %Nofault=Nofault+1;
            end
        elseif residuals(rows,h)<=-Tvect(h);

            if var(residuals(rows-10:rows,h))>Tvect(2,h);
                FhypT(h)=3; %Nofault=Nofault+1;
            else
                FhypT(h)=-1; %Nofault=Nofault+1;
            end;
        else

            FhypT(h)=0;
        end
end



FaultPossibilities=[0 0 1 1 0 0 0 -1 3 0;1 0 0 1 0 0 0 0 0 0;0 3 -1 -1 3 0 0
0 0 0;
                    1 0 0 0 0 0 1 0 0 0; 0 0 1 0 1 0 3 0 0 0; 0 0 0 0 0 1 0 0
-1 0;
                    -1 -1 0 0 0 0 0 0 0 0; 3 0 0 0 0 0 0 0 0 -1; 0 0 0 3 0 0
1 0 0 3;
                    0 0 0 0 0 0 0 1 0 -1; 0 0 1 0 1 0 3 0 0 0;0 0 1 1 0 0 0 -
1 3 0;
                    1 0 0 1 0 0 0 0 0 0;0 3 -1 -1 3 0 0 0 0 0; 0 0 1 0 1 0 3
0 0 0];

[rowsFP colsFP]=size(FaultPossibilities);
rowsI=rowsFP;
Included=[1:rowsFP];
 Deleted=0;
 faultresidualnumbers=[];
for i=1:cols;
    Included1=Included;
    if FhypT(i)~=0;
        faultresidualnumbers=[faultresidualnumbers i];
           for j=1:rowsI;
               if FaultPossibilities(Included1(j),i)~= FhypT(i);

                  Included(j-Deleted)=[];
                  Deleted=Deleted+1;
                end;
```

```
            end;
            rowsI=length(Included);
            Deleted=0;

    end;
end;

FhypT
Included



load FaultyResidualMemoryMatrix;


PossibleFaults=length(Included);
SensorFaults=length(faultresidualnumbers);
FaultyMemMat2=[];

% Creating the memory matrix
for t=1:PossibleFaults;
         if Included(t)==1;
            FaultyMemMat2=[FaultyMemMat2;
FaultyResidualsMemoryMatrix(((Included(t))):(Included(t)*10000),:)];
         else
            FaultyMemMat2=[FaultyMemMat2;
FaultyResidualsMemoryMatrix(((Included(t)-
1)*10000+1):(Included(t)*10000),:)];
         end;
end;

FaultyMemMat=[];
FaultyResidualMat=[];
for j=1:SensorFaults;
    FaultyMemMat=[FaultyMemMat FaultyMemMat2(:,j)];
    FaultyResidualMat=[FaultyResidualMat residuals(:,j)];
end;


[rowsFMM colsFMM]=size(FaultyMemMat);
[rowsFRM colsFRM]=size(FaultyResidualMat);
EuclidDist=zeros(1,PossibleFaults);

FaultyMemMat_S=expfilt(FaultyMemMat);
FaultyResidualMat_S=expfilt(FaultyResidualMat);
FaultsEuclid=zeros(1,PossibleFaults);

for l=1:PossibleFaults
    EuclidDist=zeros(1,PossibleFaults);
for t=1:((rowsFMM/PossibleFaults)/rowsFRM);
    if l==1;
        if t==1;
```

```
       EuclidDist=EuclidDist+sum((FaultyMemMat_S(t:rowsFRM,:)-
FaultyResidualMat_S).^2);
        else
       EuclidDist=EuclidDist+sum((FaultyMemMat_S((t-1)*1000:rowsFRM*t-1,:)-
FaultyResidualMat_S).^2);
        end;
    elseif l==2;
        if t==1;

EuclidDist=EuclidDist+sum((FaultyMemMat_S(t*rowsFMM/PossibleFaults:t*rowsFMM/
PossibleFaults+rowsFRM-1,:)-FaultyResidualMat_S).^2);
        else
       EuclidDist=EuclidDist+sum((FaultyMemMat_S((t-
1)*rowsFRM+rowsFMM/PossibleFaults:rowsFRM*t-1+rowsFMM/PossibleFaults,:)-
FaultyResidualMat_S).^2);
        end;
    else
        if t==1;

EuclidDist=EuclidDist+sum((FaultyMemMat_S(t*2*rowsFMM/PossibleFaults:t*2*rows
FMM/PossibleFaults+rowsFRM-1,:)-FaultyResidualMat_S).^2);
        else
       EuclidDist=EuclidDist+sum((FaultyMemMat_S((t-
1)*rowsFRM+2*rowsFMM/PossibleFaults:rowsFRM*t-1+2*rowsFMM/PossibleFaults,:)-
FaultyResidualMat_S).^2);
        end;
    end
end;
FaultsEuclid(l)=sum(sqrt(EuclidDist/(((rowsFMM/PossibleFaults)/rowsFRM))));
end;

FaultsEuclid




% This is used for producing example residuals for the fault identification
% module


rows=1000;  % Number of observations
cols=10;    % Number of residuals

resid_mag=[ 1 2 1 0.2 5 7 1 2 6 0.1];




% Creating the data
for i=1:cols;
    for j=1:rows;
        NonFaultResid(j,i)=0+random('norm',0,0.5*resid_mag(i));
    end;
```

```
end;

% NonFault residuals
figure;
plot(NonFaultResid);
title('Non-Fault Residuals');

for j=1:cols;
    figure;
    plot(NonFaultResid(:,j));
    title('Non-Fault Residuals');
end;

MaxSens1=max(NonFaultResid(:,1));
MaxSens5=max(NonFaultResid(:,5));
Sensor7BiasValue=max(NonFaultResid(:,7));
Sensor4BiasValue=max(NonFaultResid(:,4));

Sen5DriftData=NonFaultResid;
Sen7BiasData=NonFaultResid;
Sen3NoiseData=NonFaultResid;
SystemFail_1aData=NonFaultResid;
SystemFail_1bData=NonFaultResid;
SystemFail_2aData=NonFaultResid;
SystemFail_2bData=NonFaultResid;
SystemFail_2cData=NonFaultResid;



%_____________________________________________________________________
% Creating 5 different Faults
%_____________________________________________________________________
for i=1:rows;
% Sensor 5 drift
    Sen5DriftData(i,5)=i/rows*MaxSens5+random('norm',0,0.5*resid_mag(5));

% Sensor 7 bias
    Sen7BiasData(i,7)=Sensor7BiasValue+random('norm',0,0.5*resid_mag(7));

% Sensor 3 noise
    Sen3NoiseData(i,3)=0+3*random('norm',0,0.5*resid_mag(7));

% Two different faults with the same fault signatures

% System component 1a failure
% Sensor 1 drift
% Sensor 4 bias
SystemFail_1aData(i,1)=i/rows*MaxSens1+random('norm',0,0.5*resid_mag(1));
SystemFail_1aData(i,4)=Sensor4BiasValue+random('norm',0,0.5*resid_mag(4));

% System component 1b failure
SystemFail_1bData(i,1)=i/rows*0.7*MaxSens1+random('norm',0,0.5*resid_mag(1));
SystemFail_1bData(i,4)=1.3*Sensor4BiasValue+random('norm',0,0.5*resid_mag(4))
;
```

```
% Three different faults with the same fault signatures

% System component 2a failure
% Sensor 5 drift
% Sensor 3 noise
% Sensor 7 bias
SystemFail_2aData(i,5)=i/rows*MaxSens5+random('norm',0,0.5*resid_mag(5));
SystemFail_2aData(i,3)=Sensor7BiasValue+random('norm',0,0.5*resid_mag(7));
SystemFail_2aData(i,7)=0+3*random('norm',0,0.5*resid_mag(7));

% System component 2b failure
SystemFail_2bData(i,5)=i/rows*0.9*MaxSens5+random('norm',0,0.5*resid_mag(5));
SystemFail_2bData(i,3)=1.4*Sensor7BiasValue+random('norm',0,0.5*resid_mag(7))
;
SystemFail_2bData(i,7)=0+4*random('norm',0,0.5*resid_mag(7));

% System component 2c failure
SystemFail_2cData(i,5)=i/rows*1.5*MaxSens5+random('norm',0,0.5*resid_mag(5));
SystemFail_2cData(i,3)=0.6*Sensor7BiasValue+random('norm',0,0.5*resid_mag(7))
;
SystemFail_2cData(i,7)=0+2.6*random('norm',0,0.5*resid_mag(7));
end;


% Figures of the faulty residuals
figure;
plot(Sen5DriftData(:,5),'r');
hold on;
plot(NonFaultResid(:,5),'b');
title('Sensor 5 drift fault');
legend('Drift', 'Normal');
xlabel('Time')
ylabel('Residuals')


figure;
plot(Sen7BiasData(:,7),'r');
hold on;
plot(NonFaultResid(:,7),'b');
title('Sensor 7 bias fault');
legend('Bias', 'Normal');
xlabel('Time')
ylabel('Residuals')

figure;
plot(Sen3NoiseData(:,3),'r');
hold on;
plot(NonFaultResid(:,3),'b');
title('Sensor 3 noise fault');
legend('Noise', 'Normal');
xlabel('Time')
```

```
ylabel('Residuals')

figure;
subplot(2,1,1)
plot(SystemFail_1aData(:,1),'r');
hold on;
plot(SystemFail_1bData(:,1),'k');
hold on;
plot(NonFaultResid(:,1),'b');
title('Sensor 1 drift fault for a and b faults');
legend('Drift a','Drift b', 'Normal');
xlabel('Time')
ylabel('Residuals')

subplot(2,1,2)
plot(SystemFail_1aData(:,4),'r');
hold on;
plot(SystemFail_1bData(:,4),'k');
hold on;
plot(NonFaultResid(:,4),'b');
title('Sensor 4 bias fault for a and b faults');
legend('Bias a','Bias b', 'Normal');
xlabel('Time')
ylabel('Residuals')



figure;
subplot(3,1,1)
plot(SystemFail_2aData(:,5),'r');
hold on;
plot(SystemFail_2bData(:,5),'k');
hold on;
plot(SystemFail_2cData(:,5),'y');
hold on;
plot(NonFaultResid(:,5),'b');
title('Sensor 5 drift fault for a, b, and c faults');
legend('Drift a','Drift b','Drift c', 'Normal');
xlabel('Time')
ylabel('Residuals')

subplot(3,1,2)
plot(SystemFail_2aData(:,3),'r');
hold on;
plot(SystemFail_2bData(:,3),'k');
hold on;
plot(SystemFail_2cData(:,3),'y');
hold on;
plot(NonFaultResid(:,3),'b');
title('Sensor 3 bias fault for a, b, and c faults');
legend('Bias a','Bias b','Bias c', 'Normal');
xlabel('Time')
ylabel('Residuals')
```

```
subplot(3,1,3)
plot(SystemFail_2aData(:,7),'r');
hold on;
plot(SystemFail_2bData(:,7),'k');
hold on;
plot(SystemFail_2cData(:,7),'y');
hold on;
plot(NonFaultResid(:,7),'b');
title('Sensor 7 noise fault for a, b, and c faults');
legend('Noise a','Noise b','Noise c', 'Normal');
xlabel('Time')
ylabel('Residuals')

save NonFaultResid
save Sen5DriftData
save Sen7BiasData
save Sen3NoiseData
save SystemFail_1aData
save SystemFail_1bData
save SystemFail_2aData
save SystemFail_2bData
save SystemFail_2cData


%_____________________________________________________________________
% Faulty Residual Memory Matrix creation
%_____________________________________________________________________


ResSample=10;
FaultyResidualsMemoryMatrix=[];
% Every fault needs 10 faulty residuals


rows=1000;  % Number of observations
cols=10;    % Number of residuals

resid_mag=[ 1 2 1 0.2 5 7 1 2 6 0.1];


% Creating the data
for i=1:cols;
    for j=1:rows;
        NonFaultResid(j,i)=0+random('norm',0,0.5*resid_mag(i));
    end;
end;


% Faulty data

MaxSens1=max(NonFaultResid(:,1));
MaxSens5=max(NonFaultResid(:,5));
```

```
Sensor7BiasValue=max(NonFaultResid(:,7));
Sensor4BiasValue=max(NonFaultResid(:,4));


SystemFail_1aData1=NonFaultResid;
SystemFail_1aData2=NonFaultResid;
SystemFail_1aData3=NonFaultResid;
SystemFail_1aData4=NonFaultResid;
SystemFail_1aData5=NonFaultResid;
SystemFail_1aData6=NonFaultResid;
SystemFail_1aData7=NonFaultResid;
SystemFail_1aData8=NonFaultResid;
SystemFail_1aData9=NonFaultResid;
SystemFail_1aData10=NonFaultResid;

SystemFail_1bData1=NonFaultResid;
SystemFail_1bData2=NonFaultResid;
SystemFail_1bData3=NonFaultResid;
SystemFail_1bData4=NonFaultResid;
SystemFail_1bData5=NonFaultResid;
SystemFail_1bData6=NonFaultResid;
SystemFail_1bData7=NonFaultResid;
SystemFail_1bData8=NonFaultResid;
SystemFail_1bData9=NonFaultResid;
SystemFail_1bData10=NonFaultResid;


SystemFail_2aData1=NonFaultResid;
SystemFail_2aData2=NonFaultResid;
SystemFail_2aData3=NonFaultResid;
SystemFail_2aData4=NonFaultResid;
SystemFail_2aData5=NonFaultResid;
SystemFail_2aData6=NonFaultResid;
SystemFail_2aData7=NonFaultResid;
SystemFail_2aData8=NonFaultResid;
SystemFail_2aData9=NonFaultResid;
SystemFail_2aData10=NonFaultResid;


SystemFail_2bData1=NonFaultResid;
SystemFail_2bData2=NonFaultResid;
SystemFail_2bData3=NonFaultResid;
SystemFail_2bData4=NonFaultResid;
SystemFail_2bData5=NonFaultResid;
SystemFail_2bData6=NonFaultResid;
SystemFail_2bData7=NonFaultResid;
SystemFail_2bData8=NonFaultResid;
SystemFail_2bData9=NonFaultResid;
SystemFail_2bData10=NonFaultResid;

SystemFail_2cData1=NonFaultResid;
SystemFail_2cData2=NonFaultResid;
SystemFail_2cData3=NonFaultResid;
SystemFail_2cData4=NonFaultResid;
```

```
SystemFail_2cData5=NonFaultResid;
SystemFail_2cData6=NonFaultResid;
SystemFail_2cData7=NonFaultResid;
SystemFail_2cData8=NonFaultResid;
SystemFail_2cData9=NonFaultResid;
SystemFail_2cData10=NonFaultResid;


%_____________________________________________________________________
% Creating Fault Residuals
%_____________________________________________________________________

for i=1:rows;



% System component 1a failure
% Sensor 1 drift
% Sensor 4 bias
SystemFail_1aData1(i,1)=i/rows*MaxSens1+random('norm',0,0.5*resid_mag(1));
SystemFail_1aData1(i,4)=Sensor4BiasValue+random('norm',0,0.5*resid_mag(4));
SystemFail_1aData2(i,1)=i/rows*MaxSens1+random('norm',0,0.5*resid_mag(1));
SystemFail_1aData2(i,4)=Sensor4BiasValue+random('norm',0,0.5*resid_mag(4));
SystemFail_1aData3(i,1)=i/rows*MaxSens1+random('norm',0,0.5*resid_mag(1));
SystemFail_1aData3(i,4)=Sensor4BiasValue+random('norm',0,0.5*resid_mag(4));
SystemFail_1aData4(i,1)=i/rows*MaxSens1+random('norm',0,0.5*resid_mag(1));
SystemFail_1aData4(i,4)=Sensor4BiasValue+random('norm',0,0.5*resid_mag(4));
SystemFail_1aData5(i,1)=i/rows*MaxSens1+random('norm',0,0.5*resid_mag(1));
SystemFail_1aData5(i,4)=Sensor4BiasValue+random('norm',0,0.5*resid_mag(4));
SystemFail_1aData6(i,1)=i/rows*MaxSens1+random('norm',0,0.5*resid_mag(1));
SystemFail_1aData6(i,4)=Sensor4BiasValue+random('norm',0,0.5*resid_mag(4));
SystemFail_1aData7(i,1)=i/rows*MaxSens1+random('norm',0,0.5*resid_mag(1));
SystemFail_1aData7(i,4)=Sensor4BiasValue+random('norm',0,0.5*resid_mag(4));
SystemFail_1aData8(i,1)=i/rows*MaxSens1+random('norm',0,0.5*resid_mag(1));
SystemFail_1aData8(i,4)=Sensor4BiasValue+random('norm',0,0.5*resid_mag(4));
SystemFail_1aData9(i,1)=i/rows*MaxSens1+random('norm',0,0.5*resid_mag(1));
SystemFail_1aData9(i,4)=Sensor4BiasValue+random('norm',0,0.5*resid_mag(4));
SystemFail_1aData10(i,1)=i/rows*MaxSens1+random('norm',0,0.5*resid_mag(1));
SystemFail_1aData10(i,4)=Sensor4BiasValue+random('norm',0,0.5*resid_mag(4));


% System component 1b failure
SystemFail_1bData1(i,1)=i/rows*0.7*MaxSens1+random('norm',0,0.5*resid_mag(1))
;
SystemFail_1bData1(i,4)=1.3*Sensor4BiasValue+random('norm',0,0.5*resid_mag(4)
);
SystemFail_1bData2(i,1)=i/rows*0.7*MaxSens1+random('norm',0,0.5*resid_mag(1))
;
SystemFail_1bData2(i,4)=1.3*Sensor4BiasValue+random('norm',0,0.5*resid_mag(4)
);
SystemFail_1bData3(i,1)=i/rows*0.7*MaxSens1+random('norm',0,0.5*resid_mag(1))
;
SystemFail_1bData3(i,4)=1.3*Sensor4BiasValue+random('norm',0,0.5*resid_mag(4)
);
```

```
SystemFail_1bData4(i,1)=i/rows*0.7*MaxSens1+random('norm',0,0.5*resid_mag(1))
;
SystemFail_1bData4(i,4)=1.3*Sensor4BiasValue+random('norm',0,0.5*resid_mag(4)
);
SystemFail_1bData5(i,1)=i/rows*0.7*MaxSens1+random('norm',0,0.5*resid_mag(1))
;
SystemFail_1bData5(i,4)=1.3*Sensor4BiasValue+random('norm',0,0.5*resid_mag(4)
);
SystemFail_1bData6(i,1)=i/rows*0.7*MaxSens1+random('norm',0,0.5*resid_mag(1))
;
SystemFail_1bData6(i,4)=1.3*Sensor4BiasValue+random('norm',0,0.5*resid_mag(4)
);
SystemFail_1bData7(i,1)=i/rows*0.7*MaxSens1+random('norm',0,0.5*resid_mag(1))
;
SystemFail_1bData7(i,4)=1.3*Sensor4BiasValue+random('norm',0,0.5*resid_mag(4)
);
SystemFail_1bData8(i,1)=i/rows*0.7*MaxSens1+random('norm',0,0.5*resid_mag(1))
;
SystemFail_1bData8(i,4)=1.3*Sensor4BiasValue+random('norm',0,0.5*resid_mag(4)
);
SystemFail_1bData9(i,1)=i/rows*0.7*MaxSens1+random('norm',0,0.5*resid_mag(1))
;
SystemFail_1bData9(i,4)=1.3*Sensor4BiasValue+random('norm',0,0.5*resid_mag(4)
);
SystemFail_1bData10(i,1)=i/rows*0.7*MaxSens1+random('norm',0,0.5*resid_mag(1)
);
SystemFail_1bData10(i,4)=1.3*Sensor4BiasValue+random('norm',0,0.5*resid_mag(4
));


% System component 2a failure
% Sensor 5 drift
% Sensor 3 noise
% Sensor 7 bias
SystemFail_2aData1(i,5)=i/rows*MaxSens5+random('norm',0,0.5*resid_mag(5));
SystemFail_2aData1(i,3)=Sensor7BiasValue+random('norm',0,0.5*resid_mag(7));
SystemFail_2aData1(i,7)=0+3*random('norm',0,0.5*resid_mag(7));
SystemFail_2aData2(i,5)=i/rows*MaxSens5+random('norm',0,0.5*resid_mag(5));
SystemFail_2aData2(i,3)=Sensor7BiasValue+random('norm',0,0.5*resid_mag(7));
SystemFail_2aData2(i,7)=0+3*random('norm',0,0.5*resid_mag(7));
SystemFail_2aData3(i,5)=i/rows*MaxSens5+random('norm',0,0.5*resid_mag(5));
SystemFail_2aData3(i,3)=Sensor7BiasValue+random('norm',0,0.5*resid_mag(7));
SystemFail_2aData3(i,7)=0+3*random('norm',0,0.5*resid_mag(7));
SystemFail_2aData4(i,5)=i/rows*MaxSens5+random('norm',0,0.5*resid_mag(5));
SystemFail_2aData4(i,3)=Sensor7BiasValue+random('norm',0,0.5*resid_mag(7));
SystemFail_2aData4(i,7)=0+3*random('norm',0,0.5*resid_mag(7));
SystemFail_2aData5(i,5)=i/rows*MaxSens5+random('norm',0,0.5*resid_mag(5));
SystemFail_2aData5(i,3)=Sensor7BiasValue+random('norm',0,0.5*resid_mag(7));
SystemFail_2aData5(i,7)=0+3*random('norm',0,0.5*resid_mag(7));
SystemFail_2aData6(i,5)=i/rows*MaxSens5+random('norm',0,0.5*resid_mag(5));
SystemFail_2aData6(i,3)=Sensor7BiasValue+random('norm',0,0.5*resid_mag(7));
SystemFail_2aData6(i,7)=0+3*random('norm',0,0.5*resid_mag(7));
SystemFail_2aData7(i,5)=i/rows*MaxSens5+random('norm',0,0.5*resid_mag(5));
SystemFail_2aData7(i,3)=Sensor7BiasValue+random('norm',0,0.5*resid_mag(7));
SystemFail_2aData7(i,7)=0+3*random('norm',0,0.5*resid_mag(7));
```

```
SystemFail_2aData8(i,5)=i/rows*MaxSens5+random('norm',0,0.5*resid_mag(5));
SystemFail_2aData8(i,3)=Sensor7BiasValue+random('norm',0,0.5*resid_mag(7));
SystemFail_2aData8(i,7)=0+3*random('norm',0,0.5*resid_mag(7));
SystemFail_2aData9(i,5)=i/rows*MaxSens5+random('norm',0,0.5*resid_mag(5));
SystemFail_2aData9(i,3)=Sensor7BiasValue+random('norm',0,0.5*resid_mag(7));
SystemFail_2aData9(i,7)=0+3*random('norm',0,0.5*resid_mag(7));
SystemFail_2aData10(i,5)=i/rows*MaxSens5+random('norm',0,0.5*resid_mag(5));
SystemFail_2aData10(i,3)=Sensor7BiasValue+random('norm',0,0.5*resid_mag(7));
SystemFail_2aData10(i,7)=0+3*random('norm',0,0.5*resid_mag(7));


% System component 2b failure
SystemFail_2bData1(i,5)=i/rows*1.5*MaxSens5+random('norm',0,0.5*resid_mag(5))
;
SystemFail_2bData1(i,3)=1.4*Sensor7BiasValue+random('norm',0,0.5*resid_mag(7)
);
SystemFail_2bData1(i,7)=0+4*random('norm',0,0.5*resid_mag(7));
SystemFail_2bData1(i,5)=i/rows*1.5*MaxSens5+random('norm',0,0.5*resid_mag(5))
;
SystemFail_2bData1(i,3)=1.4*Sensor7BiasValue+random('norm',0,0.5*resid_mag(7)
);
SystemFail_2bData1(i,7)=0+4*random('norm',0,0.5*resid_mag(7));
SystemFail_2bData1(i,5)=i/rows*1.5*MaxSens5+random('norm',0,0.5*resid_mag(5))
;
SystemFail_2bData1(i,3)=1.4*Sensor7BiasValue+random('norm',0,0.5*resid_mag(7)
);
SystemFail_2bData1(i,7)=0+4*random('norm',0,0.5*resid_mag(7));
SystemFail_2bData1(i,5)=i/rows*1.5*MaxSens5+random('norm',0,0.5*resid_mag(5))
;
SystemFail_2bData1(i,3)=1.4*Sensor7BiasValue+random('norm',0,0.5*resid_mag(7)
);
SystemFail_2bData1(i,7)=0+4*random('norm',0,0.5*resid_mag(7));
SystemFail_2bData1(i,5)=i/rows*1.5*MaxSens5+random('norm',0,0.5*resid_mag(5))
;
SystemFail_2bData1(i,3)=1.4*Sensor7BiasValue+random('norm',0,0.5*resid_mag(7)
);
SystemFail_2bData1(i,7)=0+4*random('norm',0,0.5*resid_mag(7));
SystemFail_2bData1(i,5)=i/rows*1.5*MaxSens5+random('norm',0,0.5*resid_mag(5))
;
SystemFail_2bData1(i,3)=1.4*Sensor7BiasValue+random('norm',0,0.5*resid_mag(7)
);
SystemFail_2bData1(i,7)=0+4*random('norm',0,0.5*resid_mag(7));
SystemFail_2bData1(i,5)=i/rows*1.5*MaxSens5+random('norm',0,0.5*resid_mag(5))
;
SystemFail_2bData1(i,3)=1.4*Sensor7BiasValue+random('norm',0,0.5*resid_mag(7)
);
SystemFail_2bData1(i,7)=0+4*random('norm',0,0.5*resid_mag(7));
SystemFail_2bData1(i,5)=i/rows*1.5*MaxSens5+random('norm',0,0.5*resid_mag(5))
;
SystemFail_2bData1(i,3)=1.4*Sensor7BiasValue+random('norm',0,0.5*resid_mag(7)
);
SystemFail_2bData1(i,7)=0+4*random('norm',0,0.5*resid_mag(7));
SystemFail_2bData1(i,5)=i/rows*1.5*MaxSens5+random('norm',0,0.5*resid_mag(5))
;
```

```
SystemFail_2bData1(i,3)=1.4*Sensor7BiasValue+random('norm',0,0.5*resid_mag(7)
);
SystemFail_2bData1(i,7)=0+4*random('norm',0,0.5*resid_mag(7));
SystemFail_2bData1(i,5)=i/rows*1.5*MaxSens5+random('norm',0,0.5*resid_mag(5))
;
SystemFail_2bData1(i,3)=1.4*Sensor7BiasValue+random('norm',0,0.5*resid_mag(7)
);
SystemFail_2bData1(i,7)=0+4*random('norm',0,0.5*resid_mag(7));


% System component 2c failure
SystemFail_2cData1(i,5)=i/rows*1.5*MaxSens5+random('norm',0,0.5*resid_mag(5))
;
SystemFail_2cData1(i,3)=0.6*Sensor7BiasValue+random('norm',0,0.5*resid_mag(7)
);
SystemFail_2cData1(i,7)=0+2.6*random('norm',0,0.5*resid_mag(7));
SystemFail_2cData2(i,5)=i/rows*1.5*MaxSens5+random('norm',0,0.5*resid_mag(5))
;
SystemFail_2cData2(i,3)=0.6*Sensor7BiasValue+random('norm',0,0.5*resid_mag(7)
);
SystemFail_2cData2(i,7)=0+2.6*random('norm',0,0.5*resid_mag(7));
SystemFail_2cData3(i,5)=i/rows*1.5*MaxSens5+random('norm',0,0.5*resid_mag(5))
;
SystemFail_2cData3(i,3)=0.6*Sensor7BiasValue+random('norm',0,0.5*resid_mag(7)
);
SystemFail_2cData3(i,7)=0+2.6*random('norm',0,0.5*resid_mag(7));
SystemFail_2cData4(i,5)=i/rows*1.5*MaxSens5+random('norm',0,0.5*resid_mag(5))
;
SystemFail_2cData4(i,3)=0.6*Sensor7BiasValue+random('norm',0,0.5*resid_mag(7)
);
SystemFail_2cData4(i,7)=0+2.6*random('norm',0,0.5*resid_mag(7));
SystemFail_2cData5(i,5)=i/rows*1.5*MaxSens5+random('norm',0,0.5*resid_mag(5))
;
SystemFail_2cData5(i,3)=0.6*Sensor7BiasValue+random('norm',0,0.5*resid_mag(7)
);
SystemFail_2cData5(i,7)=0+2.6*random('norm',0,0.5*resid_mag(7));
SystemFail_2cData6(i,5)=i/rows*1.5*MaxSens5+random('norm',0,0.5*resid_mag(5))
;
SystemFail_2cData6(i,3)=0.6*Sensor7BiasValue+random('norm',0,0.5*resid_mag(7)
);
SystemFail_2cData6(i,7)=0+2.6*random('norm',0,0.5*resid_mag(7));
SystemFail_2cData7(i,5)=i/rows*1.5*MaxSens5+random('norm',0,0.5*resid_mag(5))
;
SystemFail_2cData7(i,3)=0.6*Sensor7BiasValue+random('norm',0,0.5*resid_mag(7)
);
SystemFail_2cData7(i,7)=0+2.6*random('norm',0,0.5*resid_mag(7));
SystemFail_2cData8(i,5)=i/rows*1.5*MaxSens5+random('norm',0,0.5*resid_mag(5))
;
SystemFail_2cData8(i,3)=0.6*Sensor7BiasValue+random('norm',0,0.5*resid_mag(7)
);
SystemFail_2cData8(i,7)=0+2.6*random('norm',0,0.5*resid_mag(7));
SystemFail_2cData9(i,5)=i/rows*1.5*MaxSens5+random('norm',0,0.5*resid_mag(5))
;
SystemFail_2cData9(i,3)=0.6*Sensor7BiasValue+random('norm',0,0.5*resid_mag(7)
);
```

```
SystemFail_2cData9(i,7)=0+2.6*random('norm',0,0.5*resid_mag(7));
SystemFail_2cData10(i,5)=i/rows*1.5*MaxSens5+random('norm',0,0.5*resid_mag(5)
);
SystemFail_2cData10(i,3)=0.6*Sensor7BiasValue+random('norm',0,0.5*resid_mag(7
));
SystemFail_2cData10(i,7)=0+2.6*random('norm',0,0.5*resid_mag(7));

end;



FaultyResidualsMemoryMatrix=[NonFaultResid;
    NonFaultResid;
    NonFaultResid;
    NonFaultResid;
    NonFaultResid;
    NonFaultResid;
    NonFaultResid;
    NonFaultResid;
    NonFaultResid;
    SystemFail_1aData1;
    SystemFail_1aData2;
    SystemFail_1aData3;
    SystemFail_1aData4;
    SystemFail_1aData5;
    SystemFail_1aData6;
    SystemFail_1aData7;
    SystemFail_1aData8;
    SystemFail_1aData9;
    SystemFail_1aData10;
    NonFaultResid;
    NonFaultResid;
    NonFaultResid;
    NonFaultResid;
    NonFaultResid;
    NonFaultResid;
    NonFaultResid;
    NonFaultResid;
    NonFaultResid;
    NonFaultResid;
    NonFaultResid;
    NonFaultResid;
    NonFaultResid;
    NonFaultResid;
    NonFaultResid;
    NonFaultResid;
    NonFaultResid;
    NonFaultResid;
    NonFaultResid;
    NonFaultResid;
    SystemFail_2aData1;
    SystemFail_2aData2;
    SystemFail_2aData3;
    SystemFail_2aData4;
```

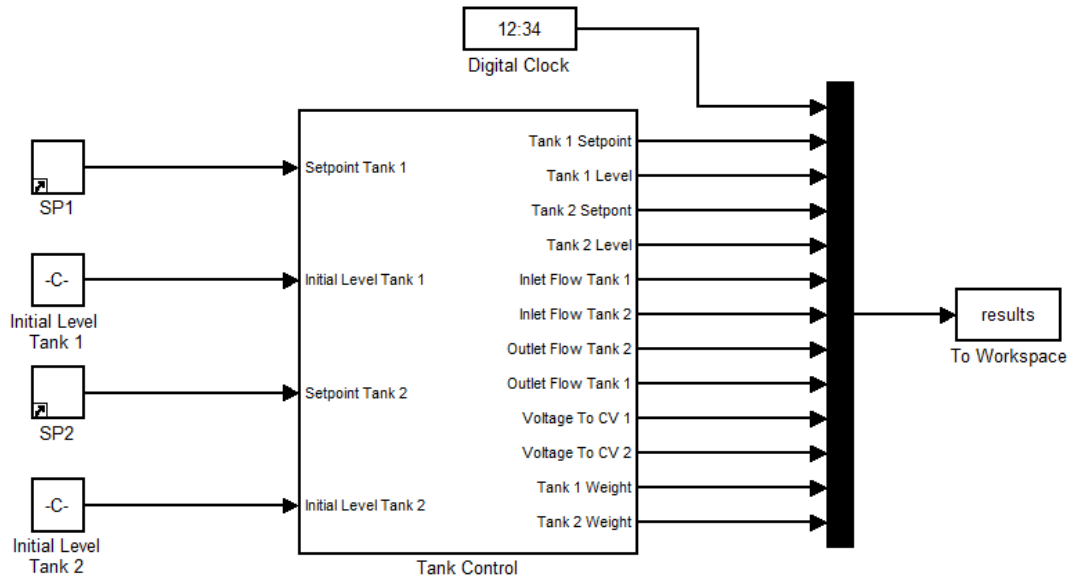
```
SystemFail_2aData5;
SystemFail_2aData6;
SystemFail_2aData7;
SystemFail_2aData8;
SystemFail_2aData9;
SystemFail_2aData10;
NonFaultResid;
NonFaultResid;
NonFaultResid;
NonFaultResid;
NonFaultResid;
NonFaultResid;
NonFaultResid;
NonFaultResid;
NonFaultResid;
NonFaultResid;
NonFaultResid;
NonFaultResid;
NonFaultResid;
NonFaultResid;
NonFaultResid;
NonFaultResid;
NonFaultResid;
NonFaultResid;
NonFaultResid;
NonFaultResid;
NonFaultResid;
NonFaultResid;
NonFaultResid;
NonFaultResid;
NonFaultResid;
NonFaultResid;
NonFaultResid;
NonFaultResid;
NonFaultResid;
NonFaultResid;
NonFaultResid;
NonFaultResid;
NonFaultResid;
NonFaultResid;
NonFaultResid;
NonFaultResid;
NonFaultResid;
NonFaultResid;
NonFaultResid;
NonFaultResid;
NonFaultResid;
NonFaultResid;
NonFaultResid;
NonFaultResid;
NonFaultResid;
NonFaultResid;
NonFaultResid;
NonFaultResid;
NonFaultResid;
```

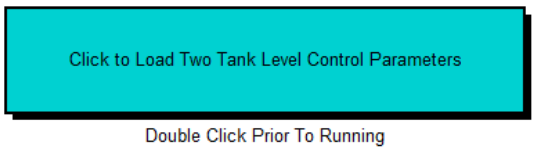
```
    NonFaultResid;
    SystemFail_2bData1;
    SystemFail_2bData2;
    SystemFail_2bData3;
    SystemFail_2bData4;
    SystemFail_2bData5;
    SystemFail_2bData6;
    SystemFail_2bData7;
    SystemFail_2bData8;
    SystemFail_2bData9;
    SystemFail_2bData10;
    NonFaultResid;
    NonFaultResid;
    NonFaultResid;
    NonFaultResid;
    NonFaultResid;
    NonFaultResid;
    NonFaultResid;
    NonFaultResid;
    NonFaultResid;
    NonFaultResid;
    SystemFail_1bData1;
    SystemFail_1bData2;
    SystemFail_1bData3;
    SystemFail_1bData4;
    SystemFail_1bData5;
    SystemFail_1bData6;
    SystemFail_1bData7;
    SystemFail_1bData8;
    SystemFail_1bData9;
    SystemFail_1bData10;
    NonFaultResid;
    NonFaultResid;
    NonFaultResid;
    NonFaultResid;
    NonFaultResid;
    NonFaultResid;
    NonFaultResid;
    NonFaultResid;
    NonFaultResid;
    NonFaultResid;
    SystemFail_2cData1;
    SystemFail_2cData2;
    SystemFail_2cData3;
    SystemFail_2cData4;
    SystemFail_2cData5;
    SystemFail_2cData6;
    SystemFail_2cData7;
    SystemFail_2cData8;
    SystemFail_2cData9;
    SystemFail_2cData10];


save FaultyResidualMemoryMatrix;
```

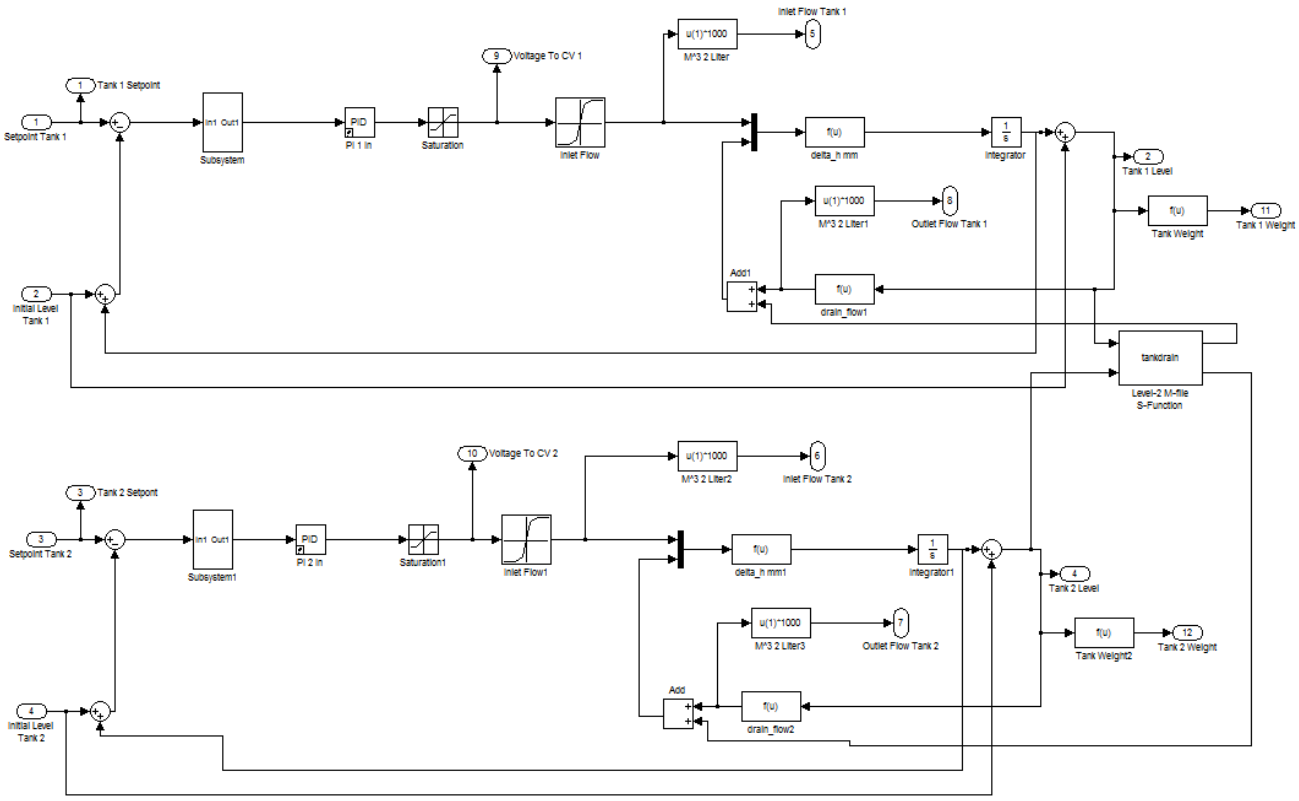
Flow Loop MATLAB SIMULINK model

Running the Simulink Model.

1) Open tankparameters.m file and adjust any constants relevant to the run. This m-file loads all the initial parameters (pipe size, valve position, initial level) and the setpoint profile for each tank.
2) Open the Simulink Model and Double Click the Blue Button
3) The output is a 13 column matrix. The columns are: Time (seconds), Tank 1 setpoint level (mm), Tank 1 measured level (mm), Tank 2 setpoint level (mm), Tank 2 measured level (mm), Inlet flow to tank 1 (Liters/sec), Inlet flow to tank 2 (Liters/sec), Outlet Flow from tank 2 (Liters/sec), Outlet flow from tank 1 (Liters/sec), Voltage to Control Valve 1, Voltage to Control Valve 2, Tank 1 water weight (kg), Tank 2 water weight (kg). The outlet and inlet flows do not count the flow contribution from the between the tank piping.
4) Run Reduce Simulink Data.m. This function down-samples the Simulink data so that the timestamps match the experimental data. This is not a necessary step but was done to make each set of data easier to compare.








```
%Load Data File
%The data file name will have to be changed to load whatever profile.
% the expected structure of the loaded profile is a an 11 column matrix
% (though only the setpoints are required.

% The columns are as follows:
% 1 - Time (seconds)
% 2 - Tank 1 setpoint level (mm)
% 3 - Tank 1 measured level (mm)
% 4 - Tank 2 setpoint level (mm)
% 5 - Tank 2 measured level (mm)
% 6 - Inlet flow to Tank 1 (Liters/sec)
% 7 - Inlet flow to Tank 2 (Liters/sec)
% 8 - Outlet flow from Tank 2 (Liters/sec)
% 9 - Outlet flow from Tank 1 (Liters/sec)
% 10 - Voltage to Control Valve 1
% 11 - Voltage to Control Valve 2
% 12 - Tank 1 water weight (kg)
% 13 - Tank 2 water weight (kg)

load('profile1.mat');
data = profile1_loop;
%constants
g=9.8; %m/s^2 - gravity
```

```
den = 1000 ;%kg/m^3 - water density

%Volumetric Flow
% Q1=2.0E-3*0.5; % unit m3/s
% Q2=2.0E-3*0.5;  % unit m3/s

%Tank Diameter
D10=0.15; % unit m
D20=0.15; % unit m

%Tank Area
A10=pi*(D10/2)^2; % unit m^2
A20=pi*(D20/2)^2; % unit m^2

%Drain Pipes/valve size
d1=0.0127; % unit m, pipe size 1/2 inch
d2=0.0127;
d3=0.0127; %between tanks

% % the drain valves are opened
% 1 = they are all fully open
Cd1=1;
Cd2=1;
Cd3=0;

% Correction factor -
% 1 = they are all fully open
Cf1=0.4014;
Cf2=0.3074;

%Drain Pipes/valve Area
a1=pi*(d1/2)^2*Cd1*Cf1;
a2=pi*(d2/2)^2*Cd2*Cf2;
bb=pi*(d3/2)^2*Cd3;

%Coefficients for outlet flow - linear regression fit.
b0_1 =  5.5423e-005;
b1_1 =  4.4505e-006;

b0_2 =  4.1727e-005;
b1_2 =  3.4736e-006;

%control valve 1 flow for given voltages (units of Liter/sec) converted to
%m^3/sec for simulink input.

voltage = 0:.1:10;
voltage = voltage';

CVIFlow = [
          0         0
          0         0
          0         0
          0         0
```

0	0
0	0
0	0
0	0
0	0
0	0
0	0
0	0
0	0
0	0
0	0
0	0
0	0
0	0
0	0
0	0
0	0
0	0
0	0
0	0
0	0
0	0
0	0
0	0
0	0
0	0
0	0
0	0
0	0
0	0
0	0
0	0.0471
0	0.0527
0	0.0600
0	0.0707
0	0.0797
0.0776	0.0891
0.0885	0.0961
0.1004	0.1009
0.1127	0.1082
0.1258	0.1127
0.1329	0.1145
0.1443	0.1158
0.1488	0.1160
0.1502	0.1154
0.1522	0.1193
0.1566	0.1287
0.1664	0.1433
0.1793	0.1535
0.1898	0.1699
0.2148	0.2028
0.2391	0.2346
0.2391	0.2346
0.2391	0.2346
0.2391	0.2346

```
    0.2391    0.2346
    0.2391    0.2346
    0.2391    0.2346
    0.2391    0.2346
    0.2391    0.2346
    0.2391    0.2346
    0.2391    0.2346
    0.2391    0.2346
    0.2391    0.2346
    0.2391    0.2346
    0.2391    0.2346
    0.2391    0.2346
    0.2391    0.2346
    0.2391    0.2346
    0.2391    0.2346
    0.2391    0.2346
    0.2391    0.2346
    0.2391    0.2346
    0.2391    0.2346
    0.2391    0.2346
    0.2391    0.2346
    0.2391    0.2346
    0.2391    0.2346
    0.2391    0.2346
    0.2391    0.2346
    0.2391    0.2346
    0.2391    0.2346
    0.2391    0.2346
    0.2391    0.2346
    0.2391    0.2346
    0.2391    0.2346
    0.2391    0.2346
    0.2391    0.2346
    0.2391    0.2346
    0.2391    0.2346
    0.2391    0.2346
    0.2391    0.2346
    0.2391    0.2346
    0.2391    0.2346
    0.2391    0.2346
    0.2391    0.2346
    0.2391    0.2346];

%convert to m^3/sec
CVIFlow = CVIFlow*10^(-3);
```

```
function tankdrain(block)
```

```
% Level-2 M file S-Function for times two demo.
%   Copyright 1990-2004 The MathWorks, Inc.
%   $Revision: 1.1.6.1 $

  setup(block);

%endfunction

function setup(block)

  %% Register number of input and output ports
  block.NumInputPorts  = 2;
  block.NumOutputPorts = 2;

  block.NumDialogPrms = 1;

%   block.InputPort(1).Name = 'cl1';
%   block.InputPort(2).Name = 'cl2';
%   block.OutputPort(1).Name = 'df1';
%   block.OutputPort(2).Name = 'df2';
%

  %% Setup functional port properties to dynamically
  %% inherited.
  block.SetPreCompInpPortInfoToDynamic;
  block.SetPreCompOutPortInfoToDynamic;

  block.InputPort(1).DirectFeedthrough = true;

  block.InputPort(1).Complexity   = 'Real';
  block.InputPort(1).DataTypeId   = 0;
  block.InputPort(1).SamplingMode = 'Sample';
  block.InputPort(1).Dimensions   = 1;

  block.InputPort(2).Complexity   = 'Real';
  block.InputPort(2).DataTypeId   = 0;
  block.InputPort(2).SamplingMode = 'Sample';
  block.InputPort(2).Dimensions   = 1;

  block.OutputPort(1).Complexity   = 'Real';
  block.OutputPort(1).DataTypeId   = 0;
  block.OutputPort(1).SamplingMode = 'Sample';
  block.OutputPort(1).Dimensions   = 1;

  block.OutputPort(2).Complexity   = 'Real';
  block.OutputPort(2).DataTypeId   = 0;
  block.OutputPort(2).SamplingMode = 'Sample';
  block.OutputPort(2).Dimensions   = 1;

  %% Set block sample time to inherited
  block.SampleTimes = [-1 0];

  %% Run accelerator on TLC
```

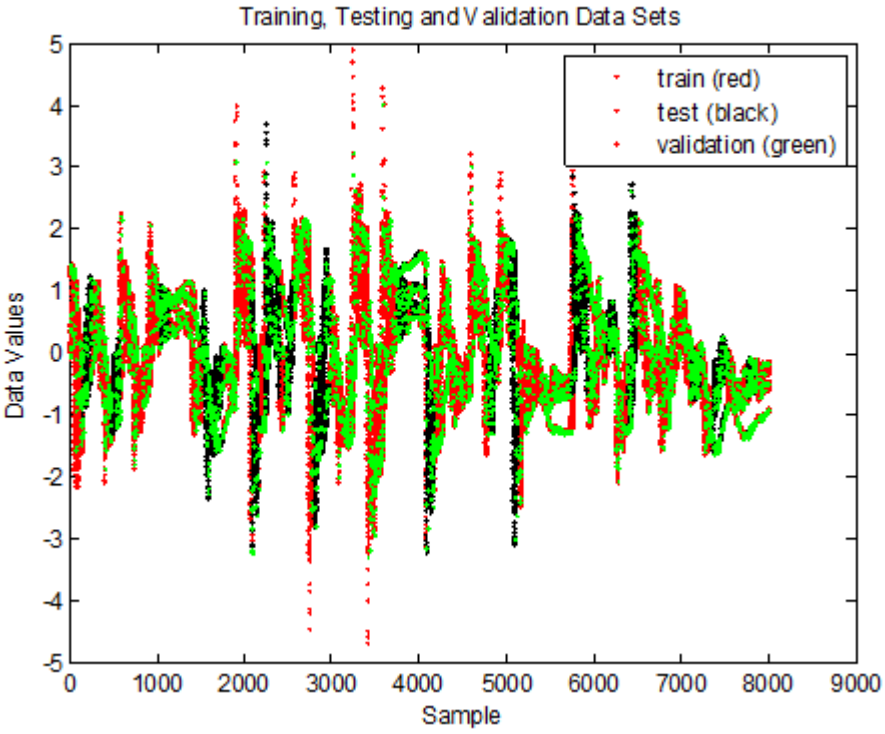
```
  block.SetAccelRunOnTLC(true);

  %% Register methods
  block.RegBlockMethod('Outputs',                 @Output);

%endfunction

function Output(block)


bb = block.DialogPrm(1).Data;

drain = bb*sqrt(2*9.81*abs(block.InputPort(1).Data-
block.InputPort(2).Data)/1000);


if block.InputPort(1).Data > block.InputPort(2).Data
    block.OutputPort(1).Data = drain;
    block.OutputPort(2).Data = -1 * drain;
end

if block.InputPort(1).Data <= block.InputPort(2).Data
    block.OutputPort(1).Data = -1*drain;
    block.OutputPort(2).Data = drain;
end


%endfunction




% load profile1_loop
% load profile1_expanded_simulink
t = profile1_loop(:,1);
[row col]=size(t);
for i =1:1:row
    a=find(results(:,1)==t(i));
%     profile1_expanded_simulink_reduced(i,:)=
profile1_expanded_simulink(a,:);
    profile1_simulink_reduced(i,:)= results(a,:);
end
% save Profile1_expanded profile1_expanded_loop profile1_expanded_simulink
profile1_expanded_simulink_reduced colheaders
```

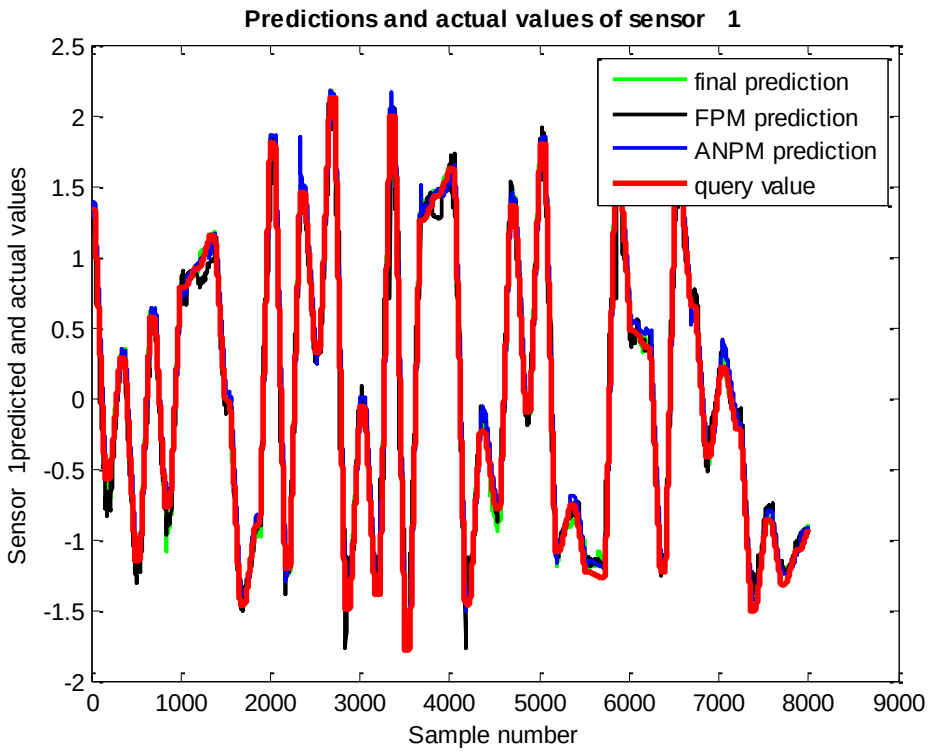
APPENDIX B: FIGURES

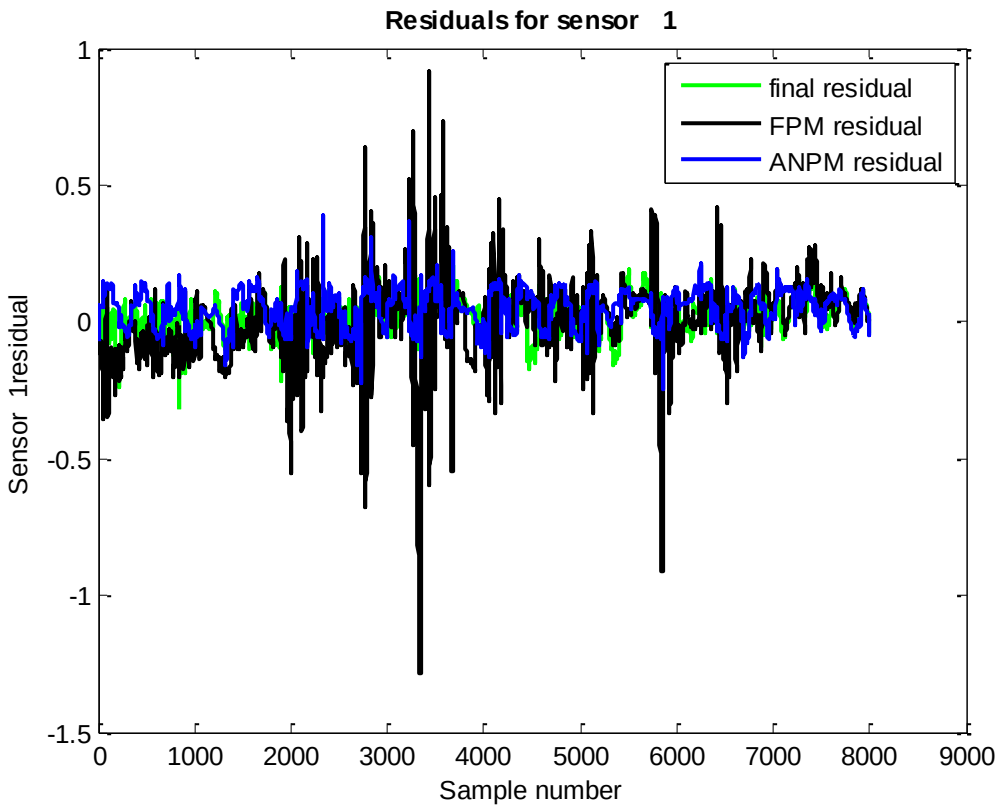
ANPM Figures

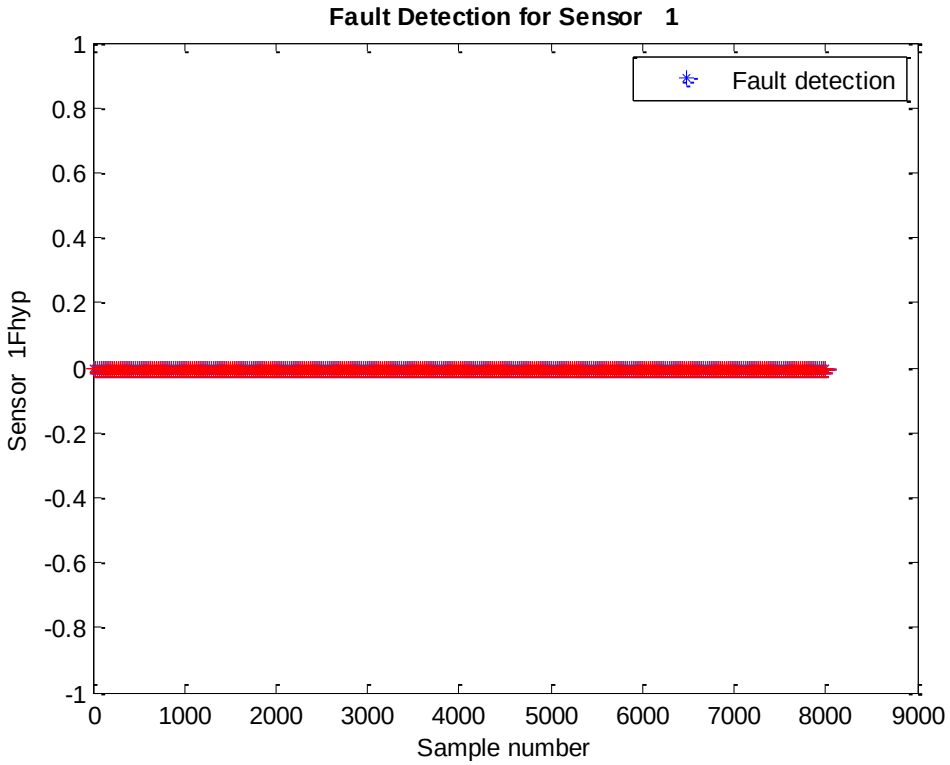
Figures not shown in the dissertation.

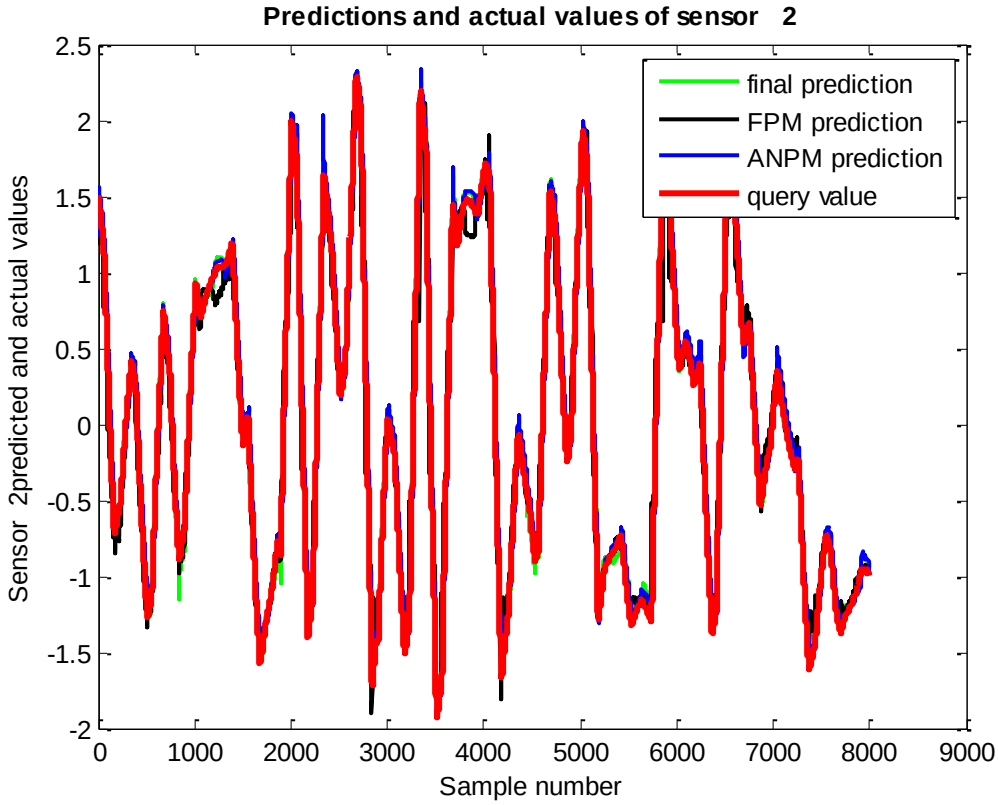
Flow Loop ANPM Predictions, Residuals, and Fault Alarms

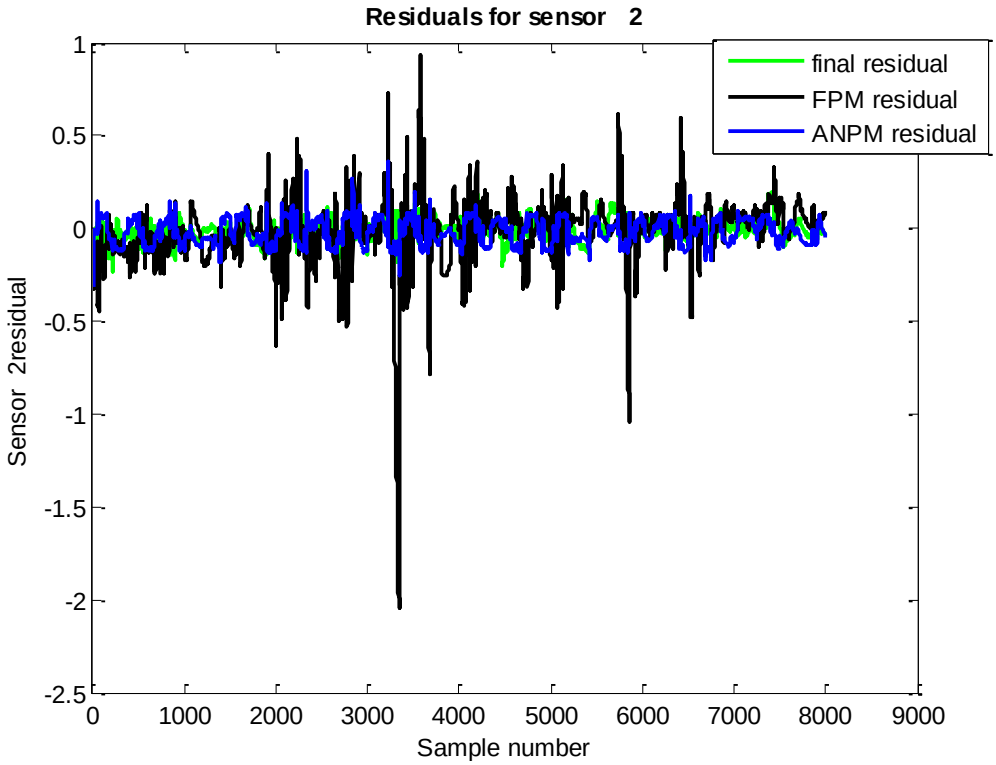


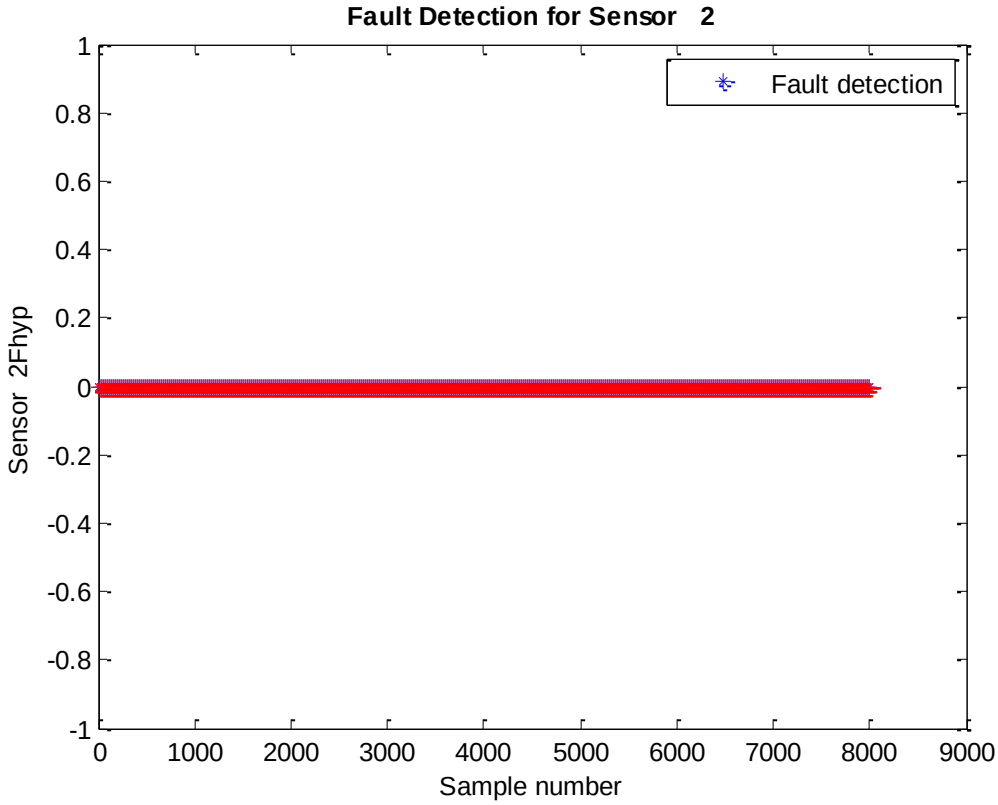

Predictions and actual values of sensor 1
final prediction
FPM prediction
ANPM prediction
query value
Sensor 1predicted and actual values
Sample number
2.5
2
1.5
1
0.5
0
-0.5
-1
-1.5
-2
0
1000
2000
3000
4000
5000
6000
7000
8000
9000

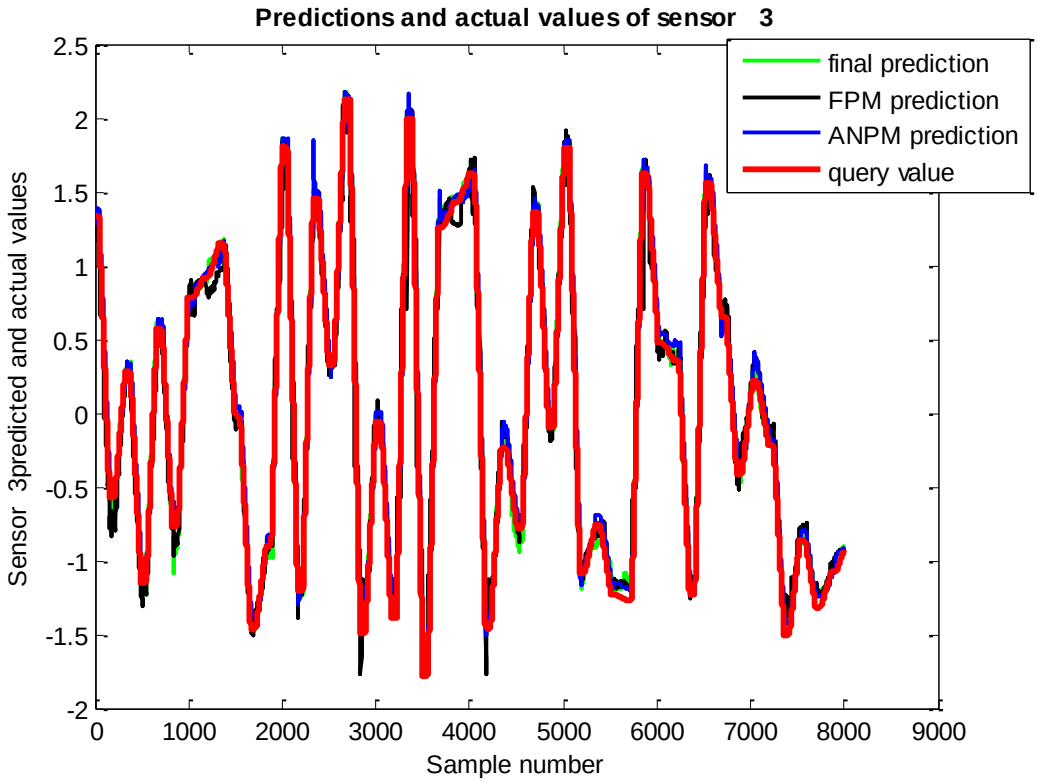

Residuals for sensor 1
final residual
FPM residual
ANPM residual
Sensor 1residual
Sample number
1
0.5
0
-0.5
-1
-1.5
0
1000
2000
3000
4000
5000
6000
7000
8000
9000

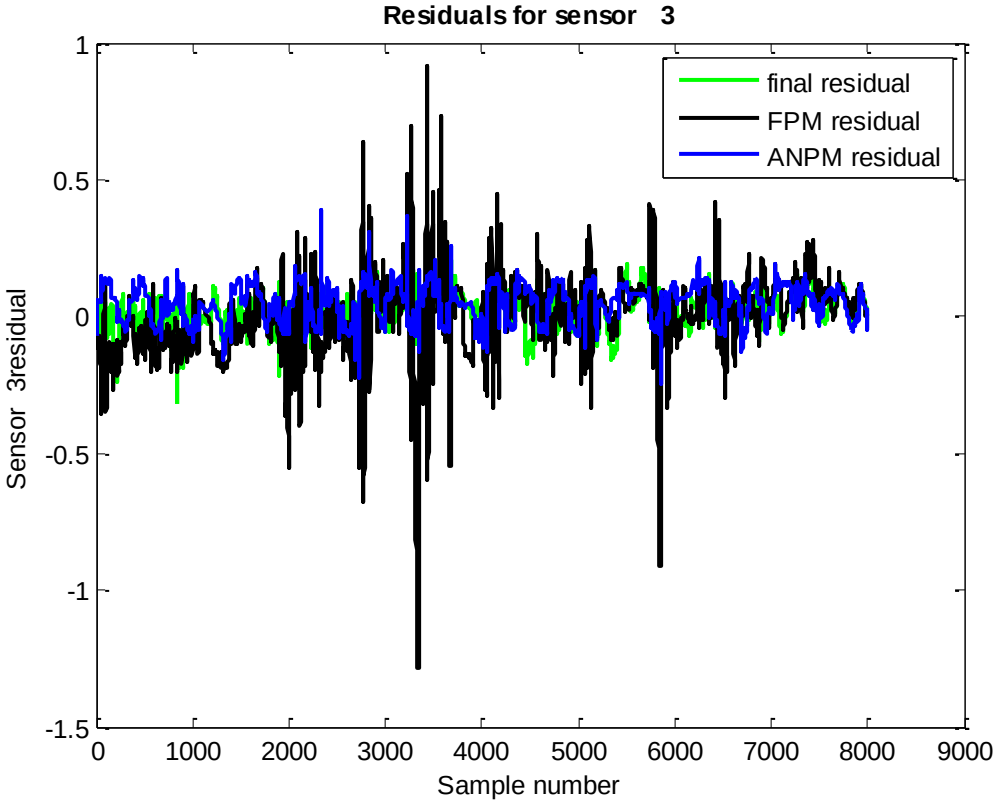

Fault Detection for Sensor 1
Fault detection
Sensor 1Fhyp
Sample number
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1
0
1000
2000
3000
4000
5000
6000
7000
8000
9000

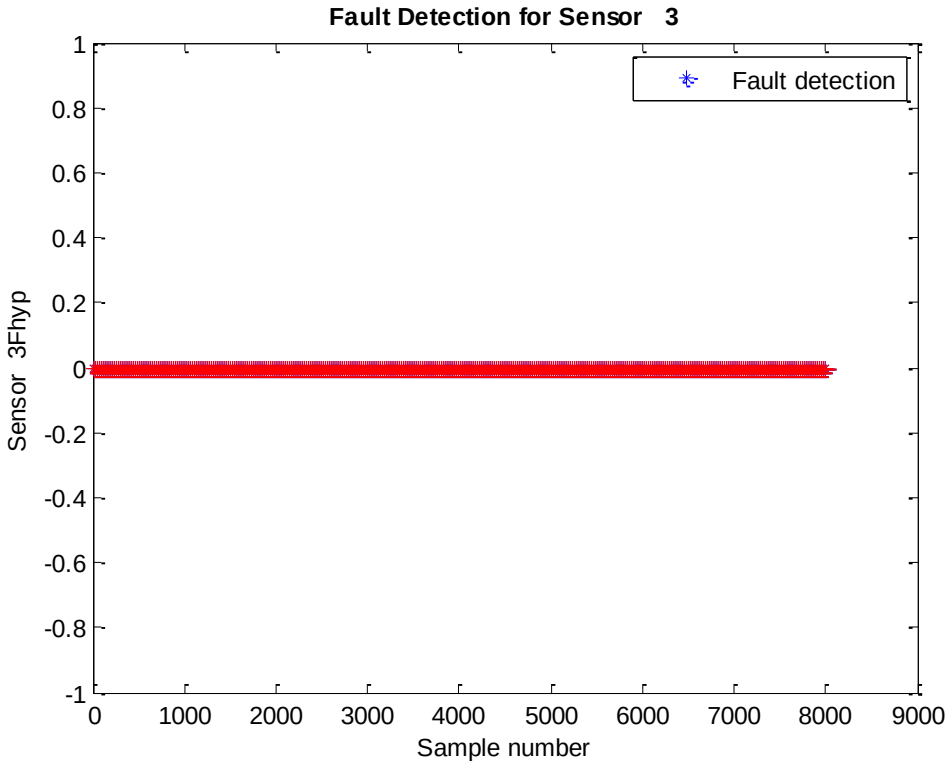

Predictions and actual values of sensor 2
final prediction
FPM prediction
ANPM prediction
query value
Sensor 2predicted and actual values
Sample number
2.5
2
1.5
1
0.5
0
-0.5
-1
-1.5
-2
0
1000
2000
3000
4000
5000
6000
7000
8000
9000

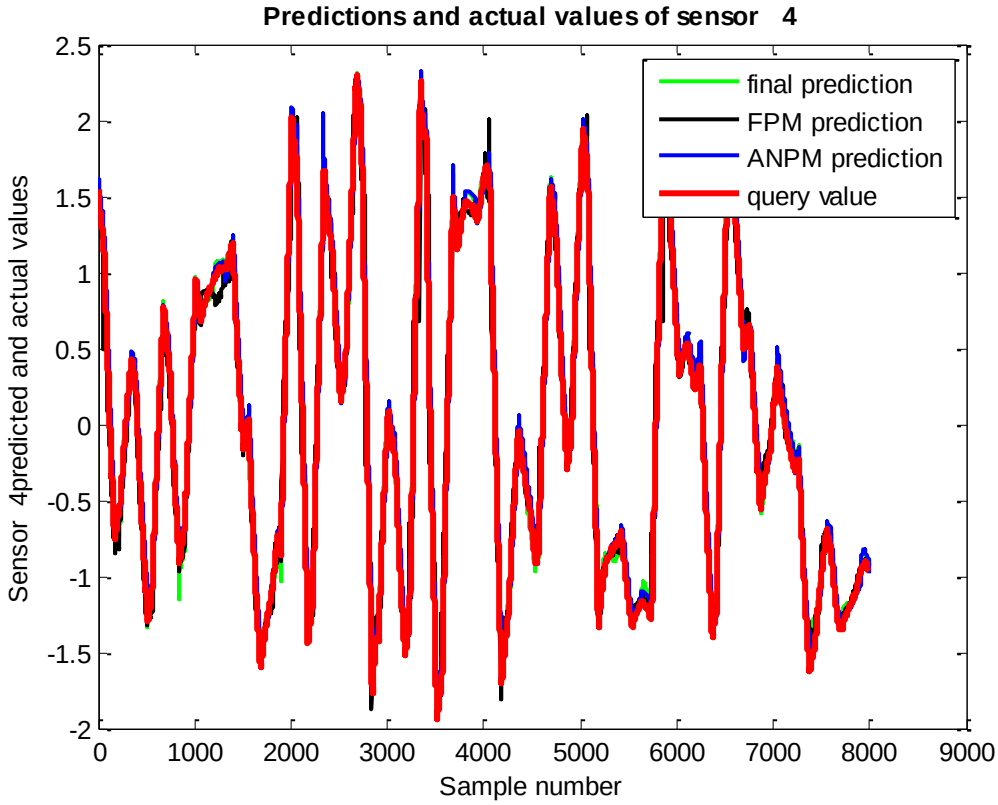

Residuals for sensor 2
final residual
FPM residual
ANPM residual
Sensor 2residual
Sample number
1
0.5
0
-0.5
-1
-1.5
-2
-2.5
0
1000
2000
3000
4000
5000
6000
7000
8000
9000

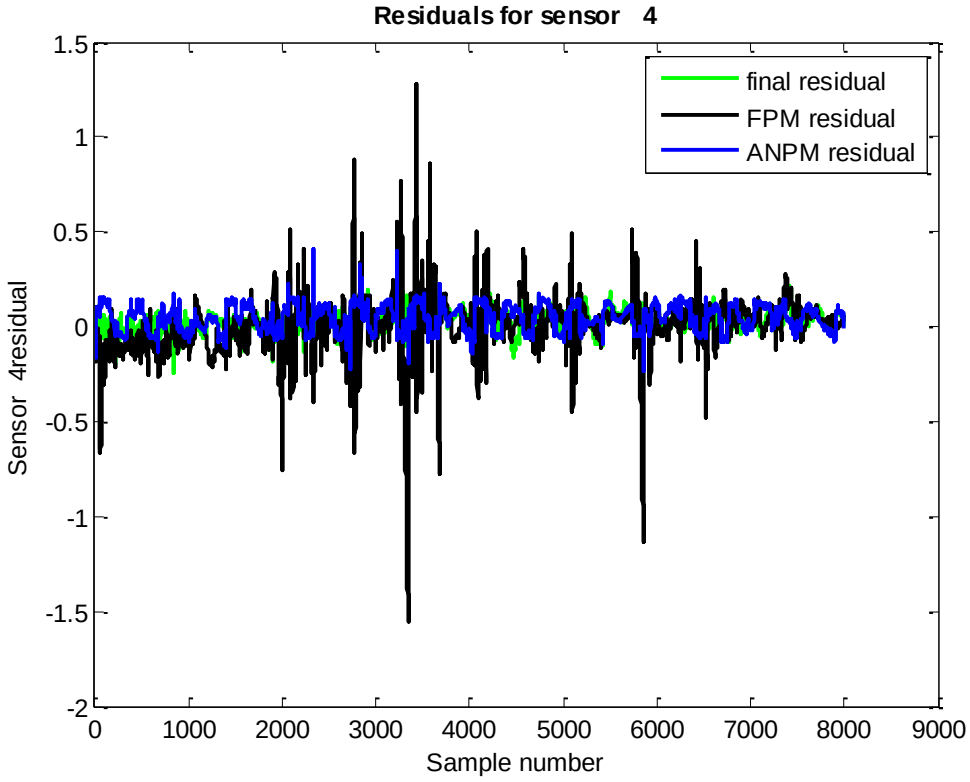

Fault Detection for Sensor 2
Fault detection
Sensor 2Fhyp
Sample number
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1
0
1000
2000
3000
4000
5000
6000
7000
8000
9000

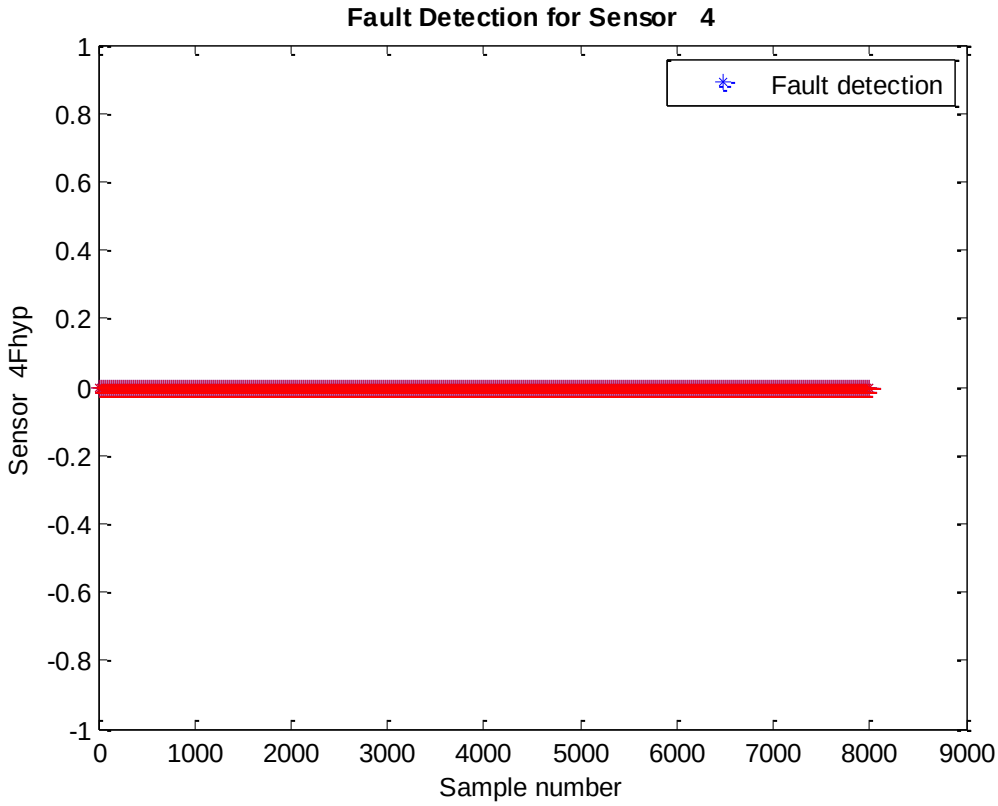

Predictions and actual values of sensor 3
final prediction
FPM prediction
ANPM prediction
query value
Sensor 3predicted and actual values
Sample number
2.5
2
1.5
1
0.5
0
-0.5
-1
-1.5
-2
0
1000
2000
3000
4000
5000
6000
7000
8000
9000

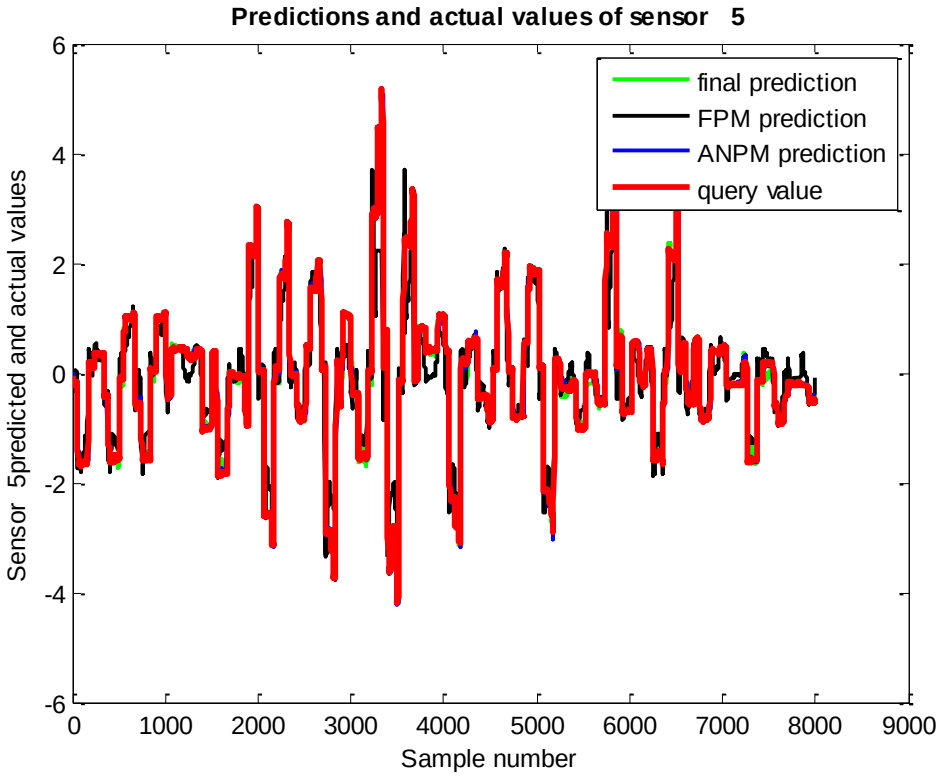

Residuals for sensor 3
final residual
FPM residual
ANPM residual
Sensor 3residual
Sample number
1
0.5
0
-0.5
-1
-1.5
0
1000
2000
3000
4000
5000
6000
7000
8000
9000

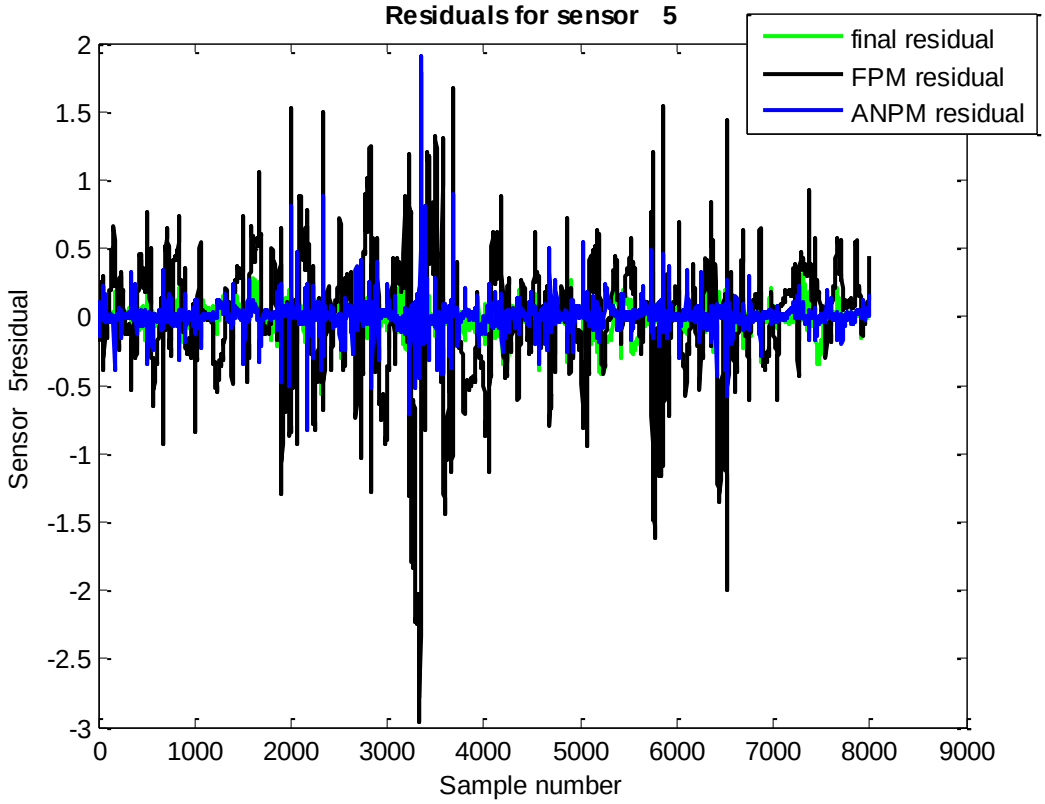

Fault Detection for Sensor 3
Fault detection
Sensor 3Fhyp
Sample number
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1
0
1000
2000
3000
4000
5000
6000
7000
8000
9000

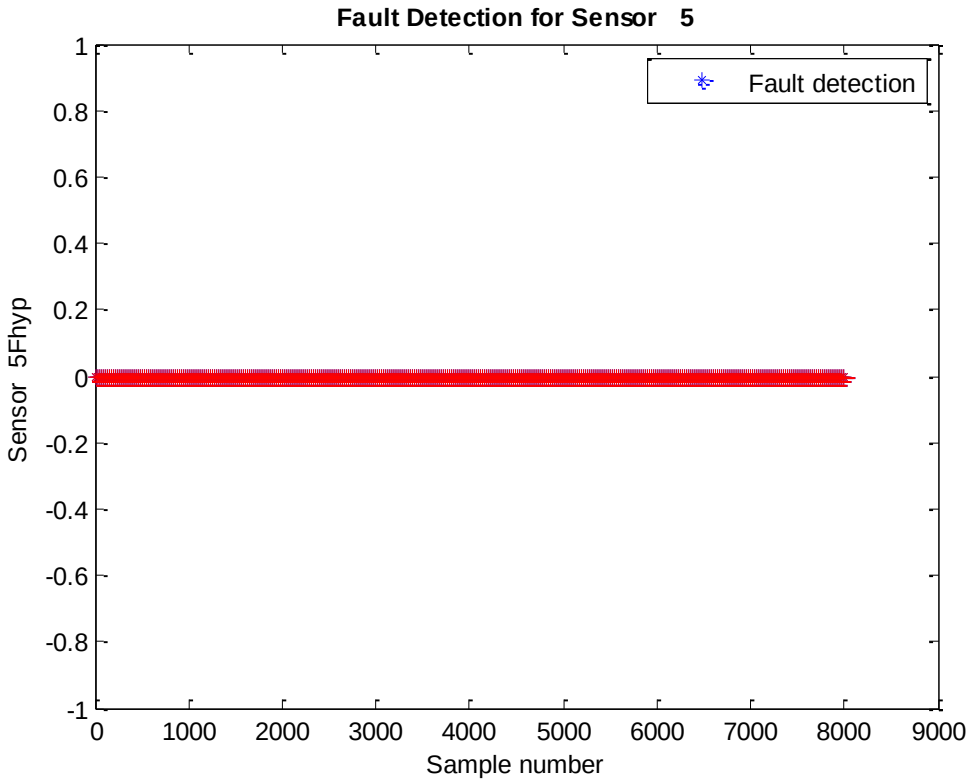

Predictions and actual values of sensor 4
final prediction
FPM prediction
ANPM prediction
query value
Sensor 4predicted and actual values
Sample number
2.5
2
1.5
1
0.5
0
-0.5
-1
-1.5
-2
0
1000
2000
3000
4000
5000
6000
7000
8000
9000

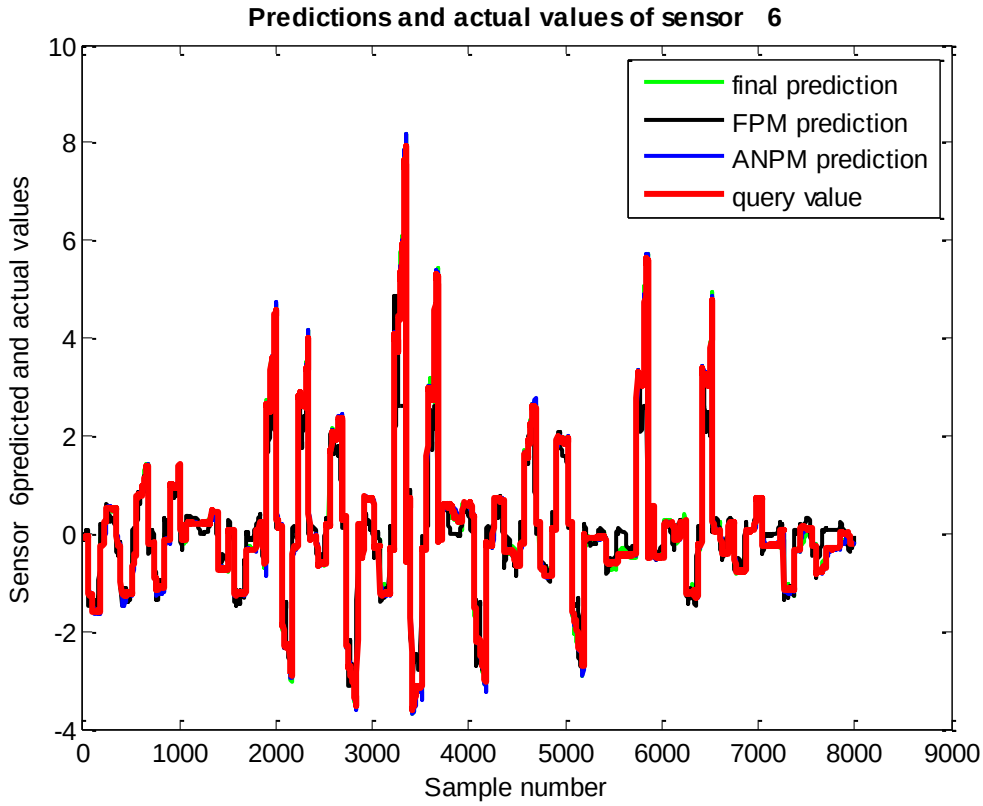

Residuals for sensor 4
final residual
FPM residual
ANPM residual
Sensor 4residual
Sample number
1.5
1
0.5
0
-0.5
-1
-1.5
-2
0
1000
2000
3000
4000
5000
6000
7000
8000
9000

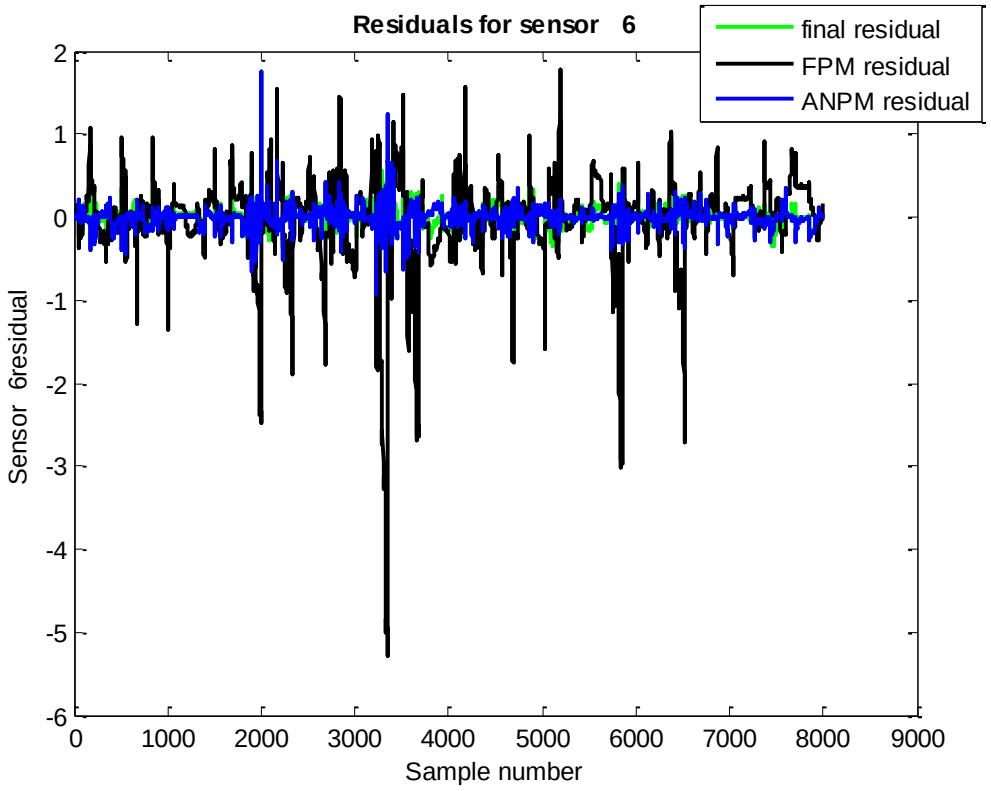

Fault Detection for Sensor 4
Fault detection
Sensor 4Fhyp
Sample number
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1
0
1000
2000
3000
4000
5000
6000
7000
8000
9000

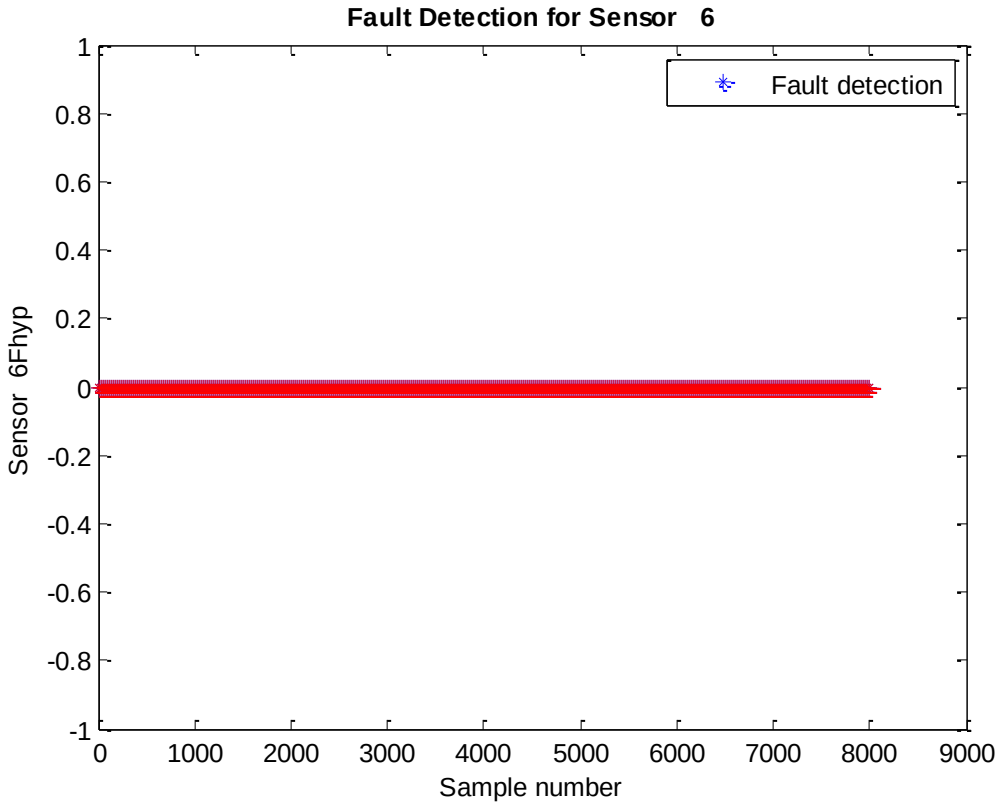

Predictions and actual values of sensor 5
final prediction
FPM prediction
ANPM prediction
query value
Sensor 5predicted and actual values
Sample number
6
4
2
0
-2
-4
-6
0
1000
2000
3000
4000
5000
6000
7000
8000
9000

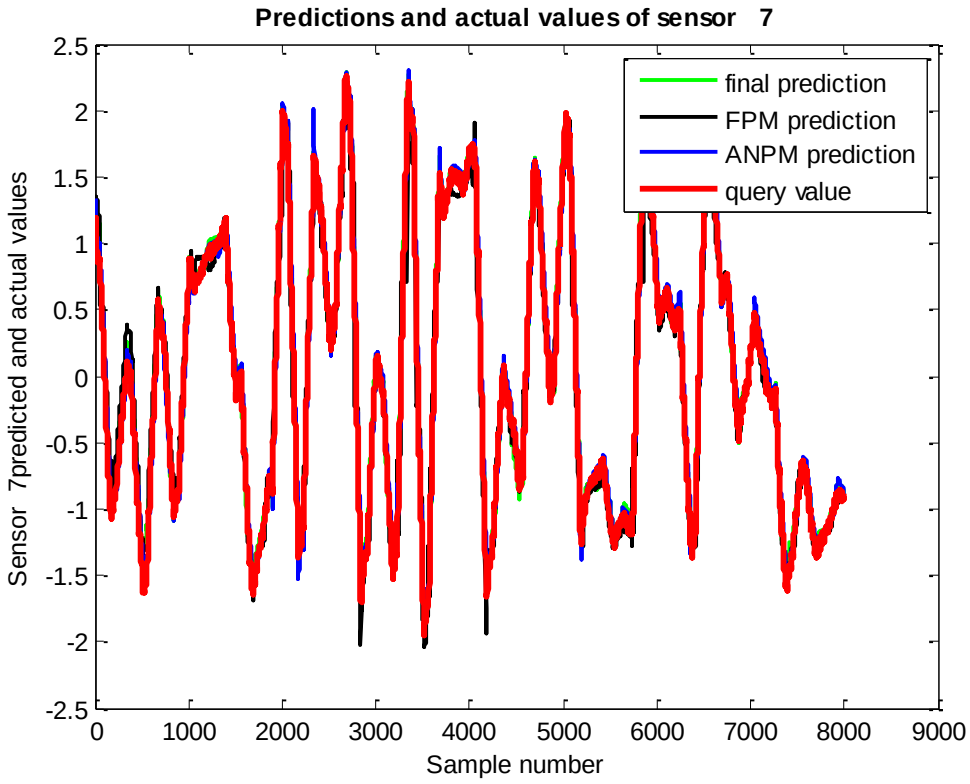

Residuals for sensor 5
final residual
FPM residual
ANPM residual
Sensor 5residual
Sample number
2
1.5
1
0.5
0
-0.5
-1
-1.5
-2
-2.5
-3
0
1000
2000
3000
4000
5000
6000
7000
8000
9000

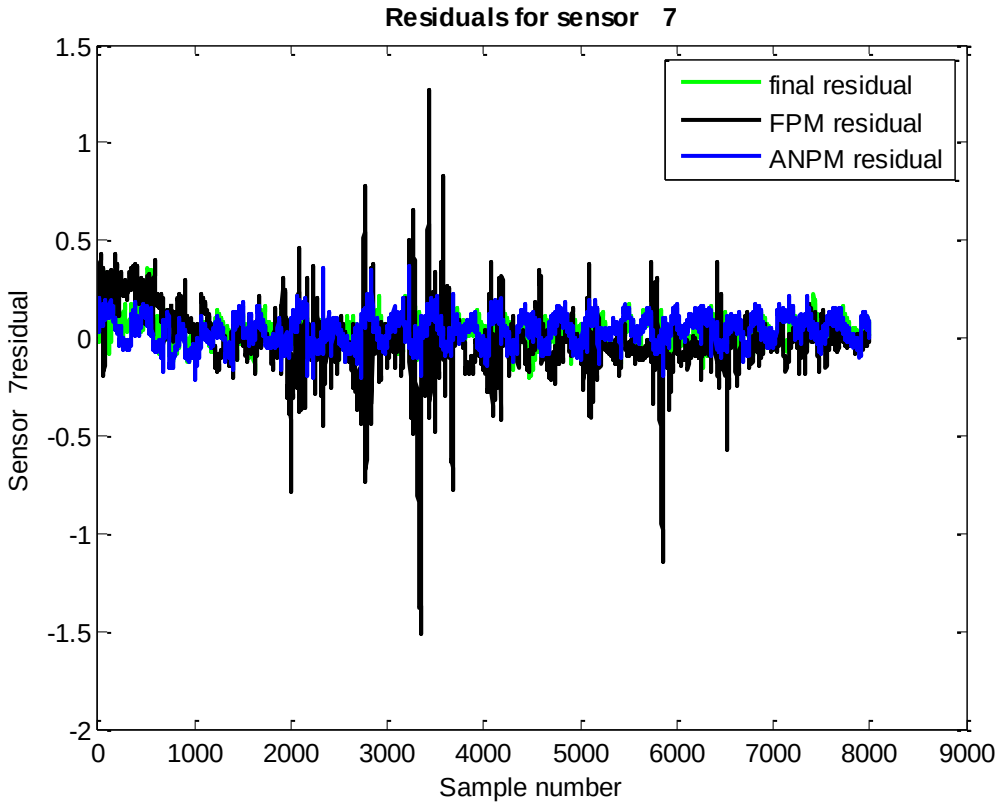

Fault Detection for Sensor 5
Fault detection
Sensor 5Fhyp
Sample number
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1
0
1000
2000
3000
4000
5000
6000
7000
8000
9000

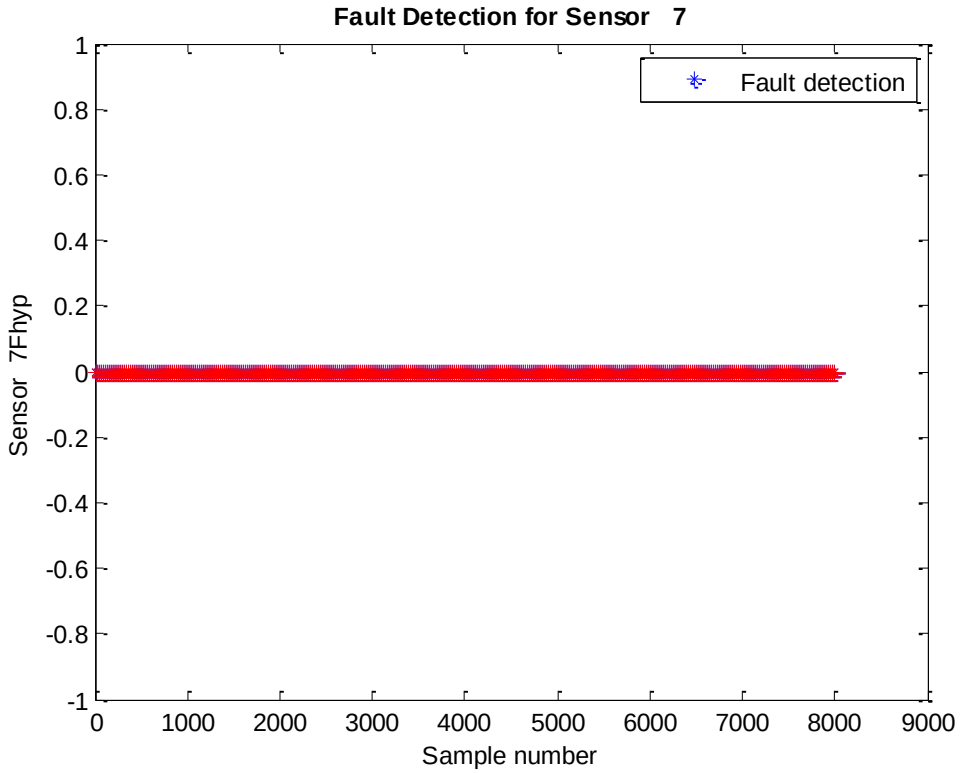

Predictions and actual values of sensor 6
final prediction
FPM prediction
ANPM prediction
query value
Sensor 6predicted and actual values
Sample number
10
8
6
4
2
0
-2
-4
0
1000
2000
3000
4000
5000
6000
7000
8000
9000

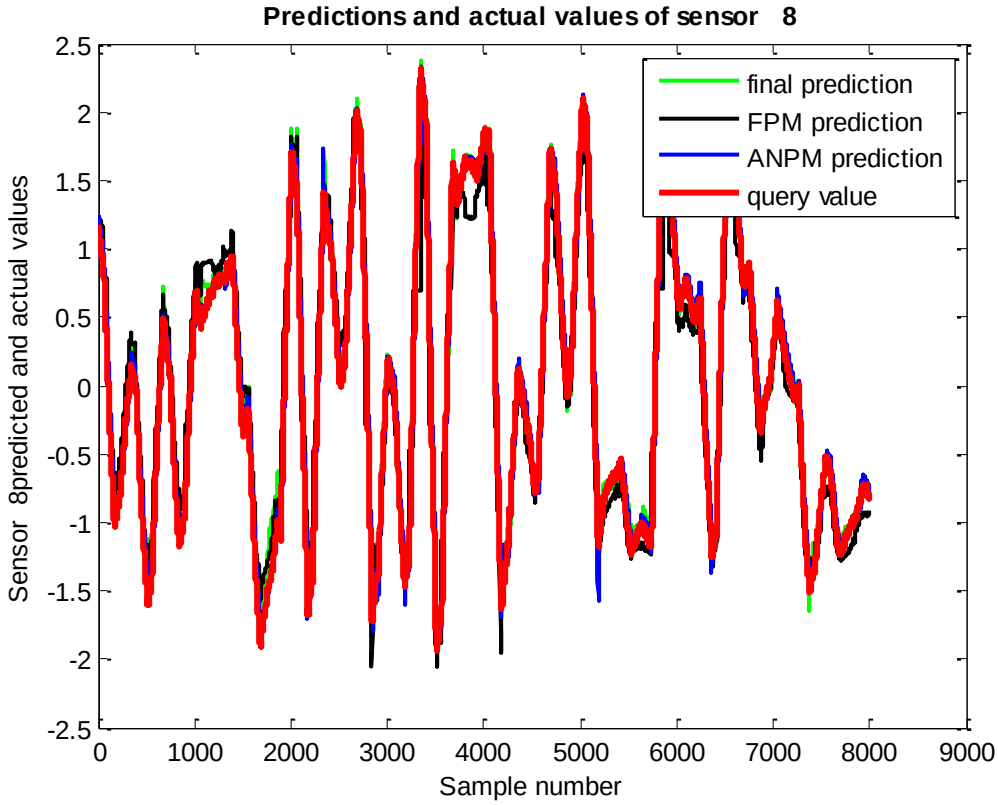

Residuals for sensor 6
final residual
FPM residual
ANPM residual
Sensor 6residual
Sample number
2
1
0
-1
-2
-3
-4
-5
-6
0
1000
2000
3000
4000
5000
6000
7000
8000
9000

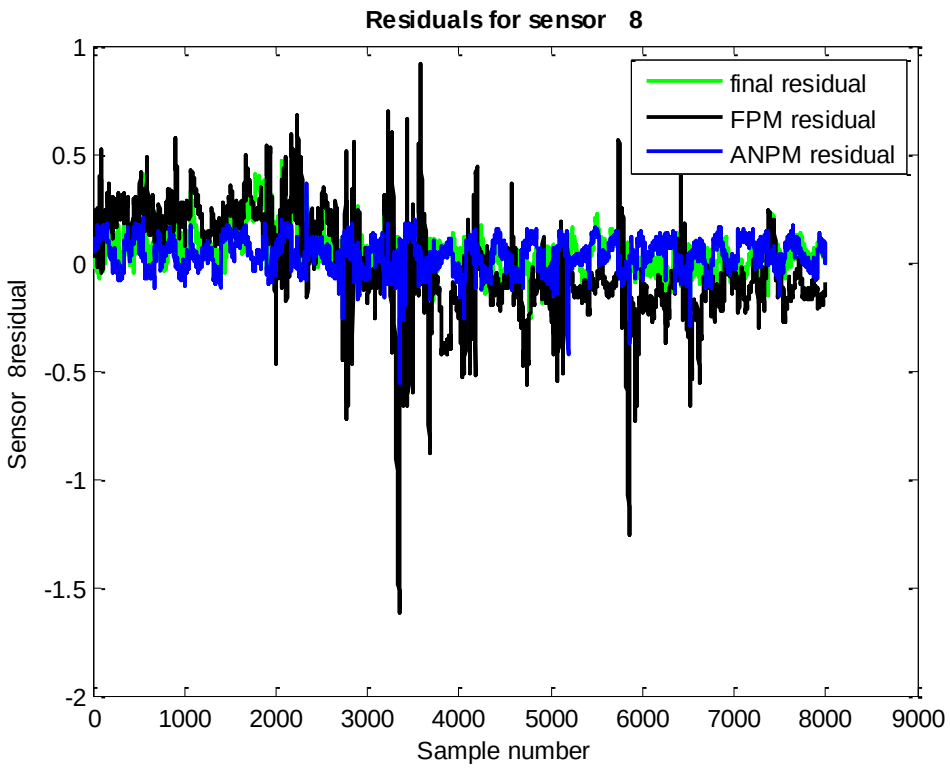

Fault Detection for Sensor 6
Fault detection
Sensor 6Fhyp
Sample number
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1
0
1000
2000
3000
4000
5000
6000
7000
8000
9000

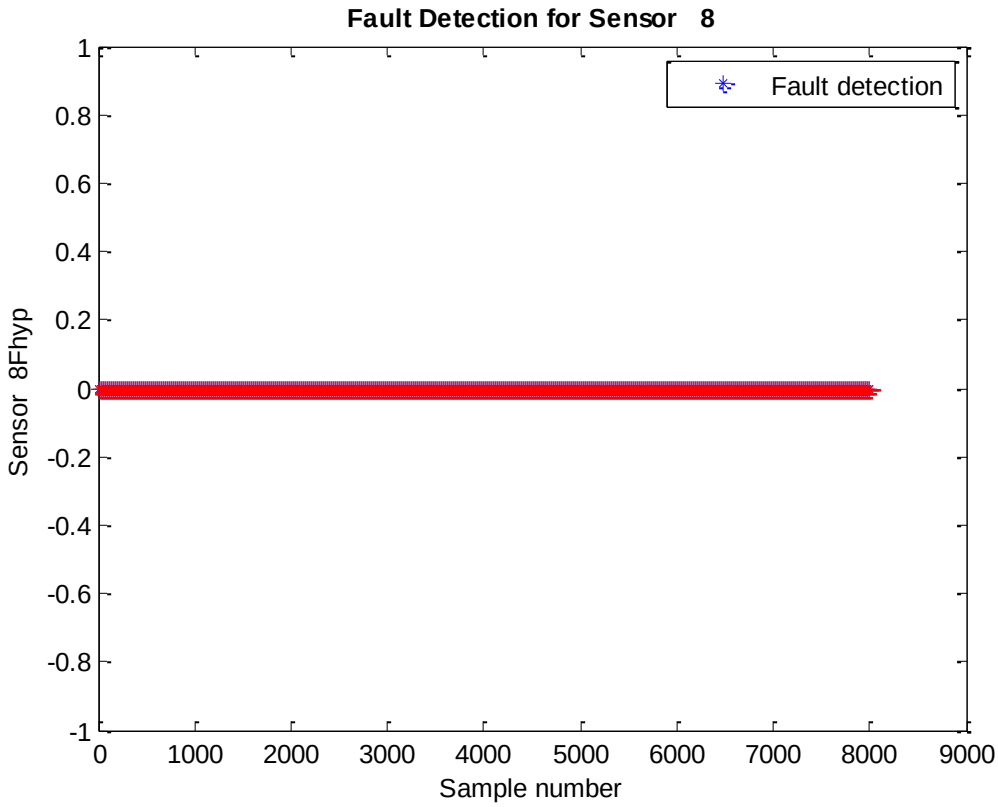

Predictions and actual values of sensor 7
final prediction
FPM prediction
ANPM prediction
query value
Sensor 7predicted and actual values
Sample number
2.5
2
1.5
1
0.5
0
-0.5
-1
-1.5
-2
-2.5
0
1000
2000
3000
4000
5000
6000
7000
8000
9000

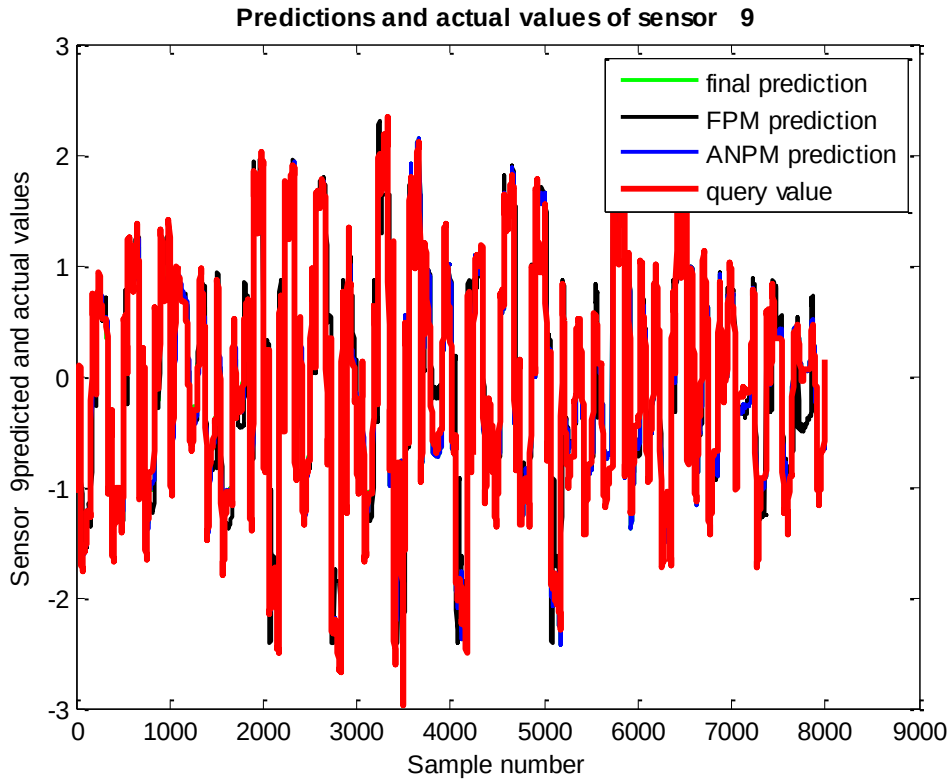

Residuals for sensor 7
final residual
FPM residual
ANPM residual
Sensor 7residual
Sample number
1.5
1
0.5
0
-0.5
-1
-1.5
-2
0
1000
2000
3000
4000
5000
6000
7000
8000
9000

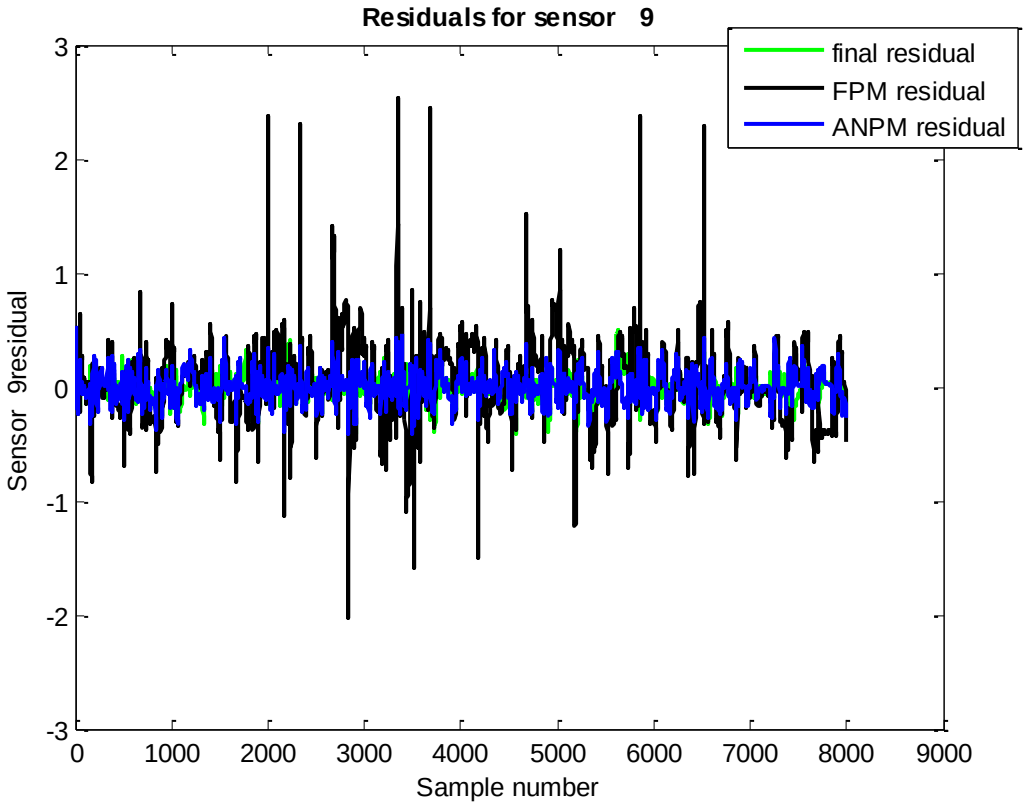

Fault Detection for Sensor 7
Fault detection
Sensor 7Fhyp
Sample number
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1
0
1000
2000
3000
4000
5000
6000
7000
8000
9000

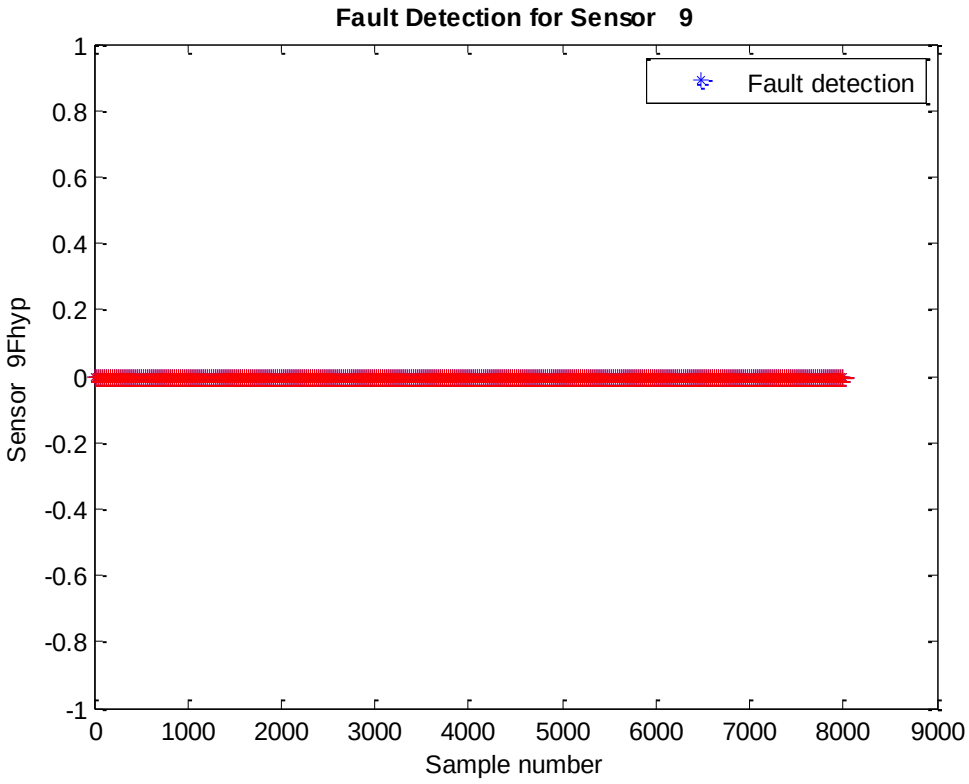

Predictions and actual values of sensor 8
final prediction
FPM prediction
ANPM prediction
query value
Sensor 8predicted and actual values
Sample number
2.5
2
1.5
1
0.5
0
-0.5
-1
-1.5
-2
-2.5
0
1000
2000
3000
4000
5000
6000
7000
8000
9000

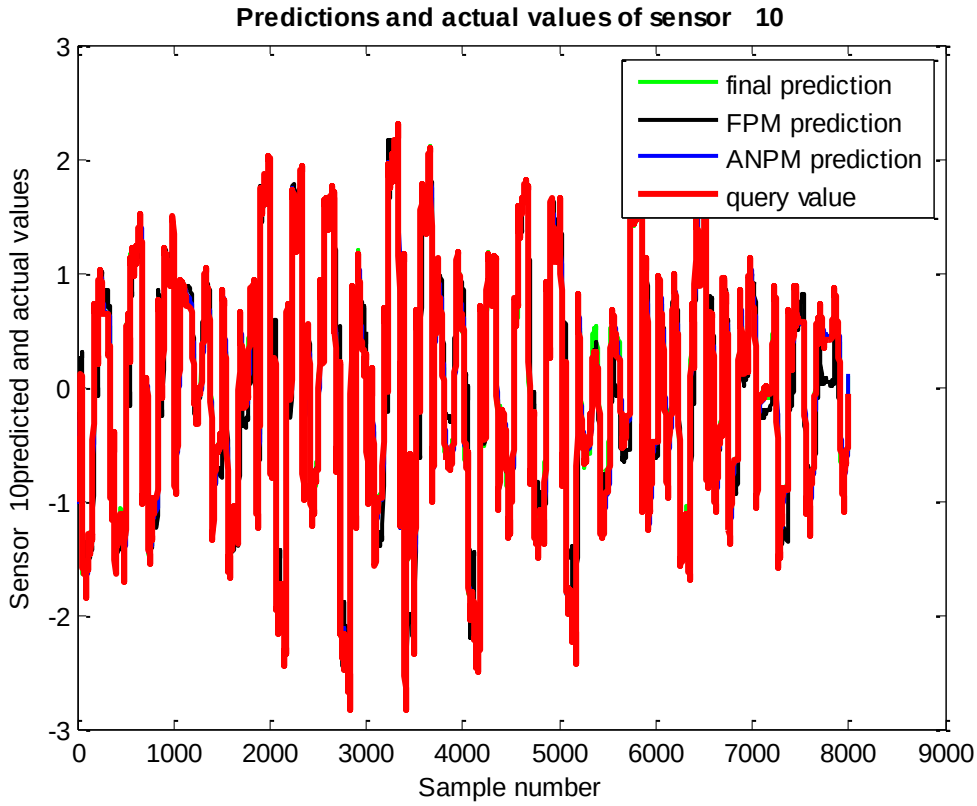

Residuals for sensor 8
final residual
FPM residual
ANPM residual
Sensor 8residual
Sample number
1
0.5
0
-0.5
-1
-1.5
-2
0
1000
2000
3000
4000
5000
6000
7000
8000
9000

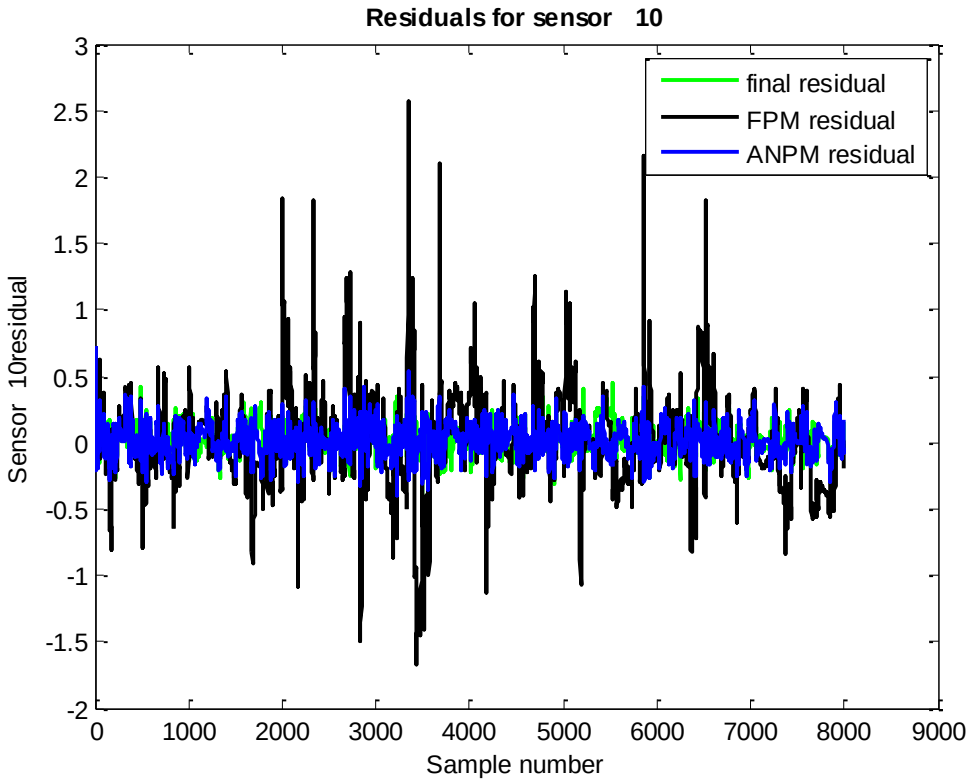

Fault Detection for Sensor 8
Fault detection
Sensor 8Fhyp
Sample number
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1
0
1000
2000
3000
4000
5000
6000
7000
8000
9000

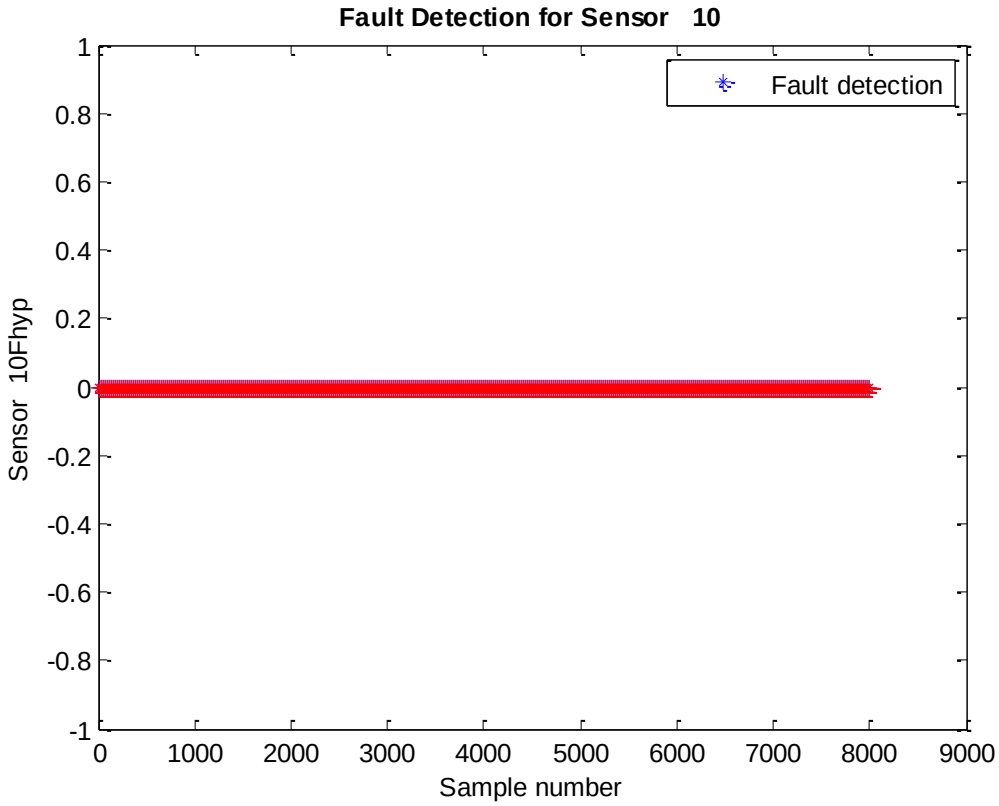

Predictions and actual values of sensor 9
final prediction
FPM prediction
ANPM prediction
query value
Sensor 9predicted and actual values
Sample number
3
2
1
0
-1
-2
-3
0
1000
2000
3000
4000
5000
6000
7000
8000
9000


Residuals for sensor 9
final residual
FPM residual
ANPM residual
Sensor 9residual
Sample number
3
2
1
0
-1
-2
-3
0
1000
2000
3000
4000
5000
6000
7000
8000
9000


Fault Detection for Sensor 9
Fault detection
Sensor 9Fhyp
Sample number
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1
0
1000
2000
3000
4000
5000
6000
7000
8000
9000


Predictions and actual values of sensor 10
final prediction
FPM prediction
ANPM prediction
query value
Sensor 10predicted and actual values
Sample number
3
2
1
0
-1
-2
-3
0
1000
2000
3000
4000
5000
6000
7000
8000
9000

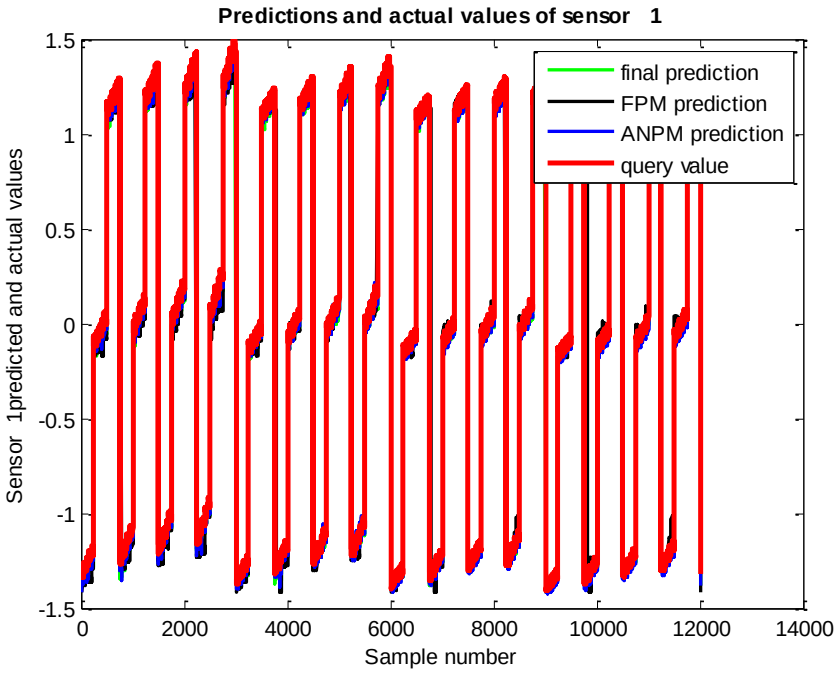

Residuals for sensor 10
final residual
FPM residual
ANPM residual
Sensor 10residual
Sample number
3
2.5
2
1.5
1
0.5
0
-0.5
-1
-1.5
-2
0
1000
2000
3000
4000
5000
6000
7000
8000
9000

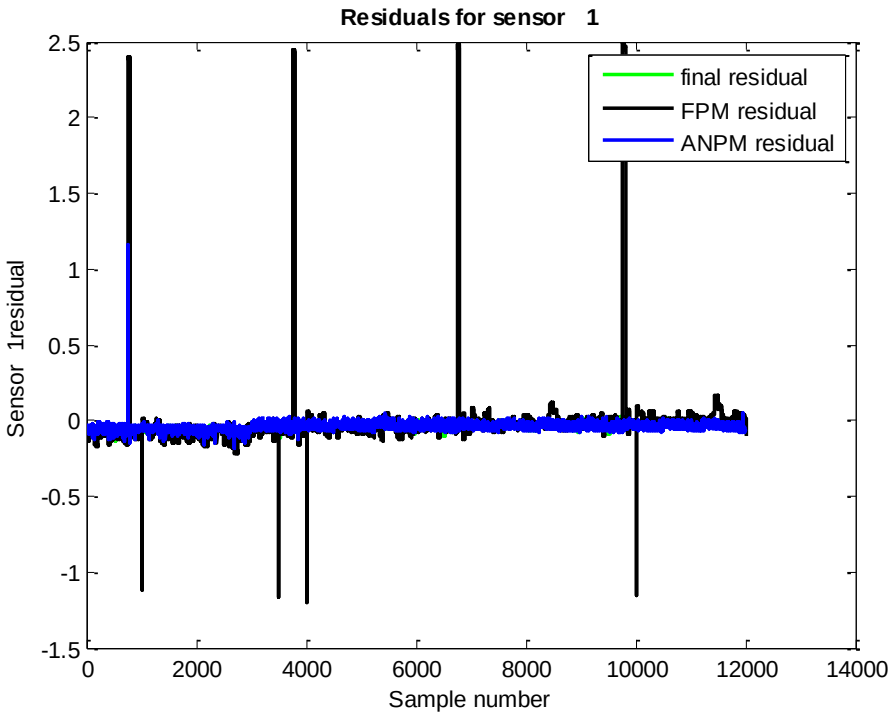

Fault Detection for Sensor 10
Fault detection
Sensor 10Fhyp
Sample number
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1
0
1000
2000
3000
4000
5000
6000
7000
8000
9000

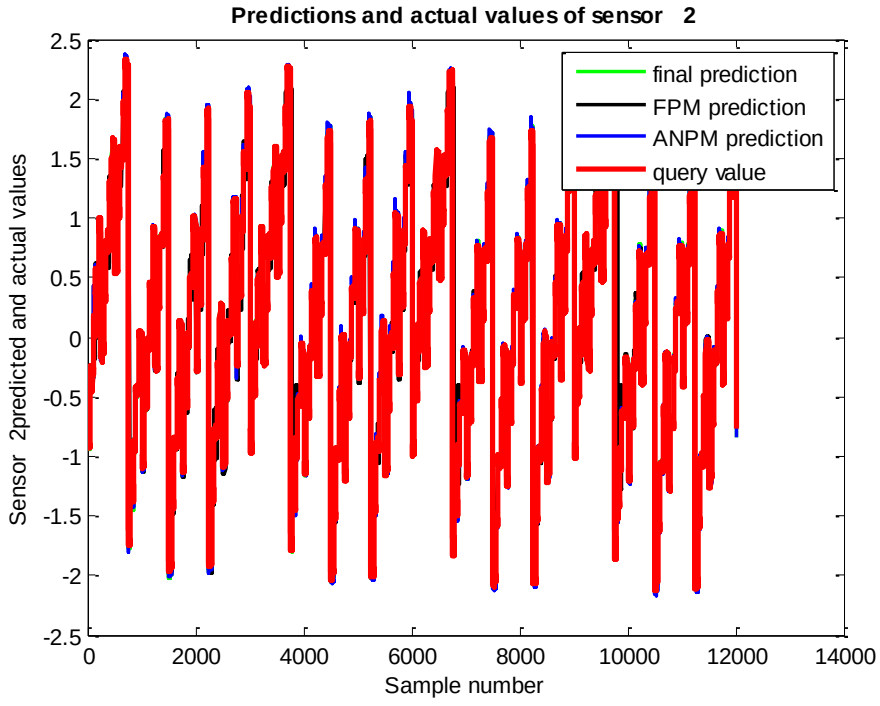
ANPM1 Heat Exchanger Outputs

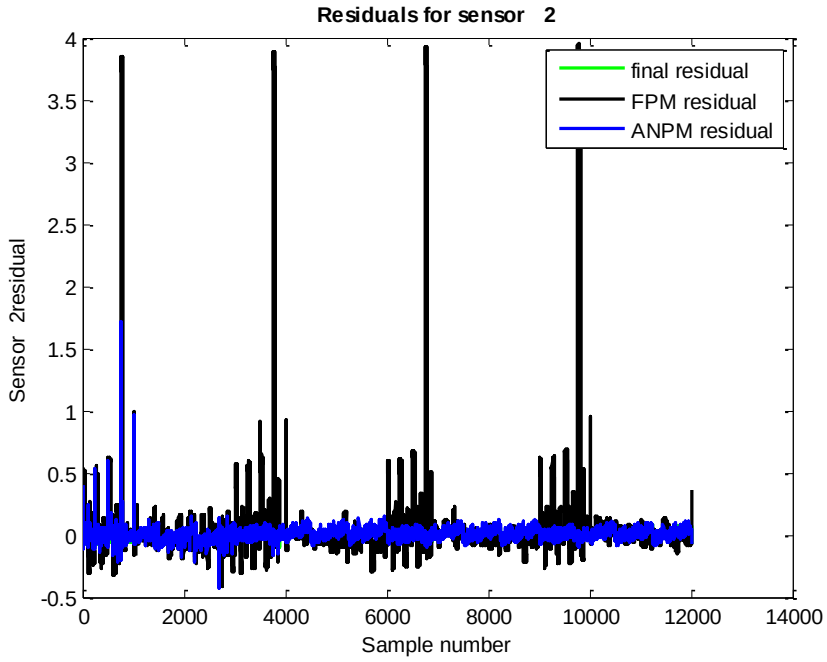
MSE for predictions

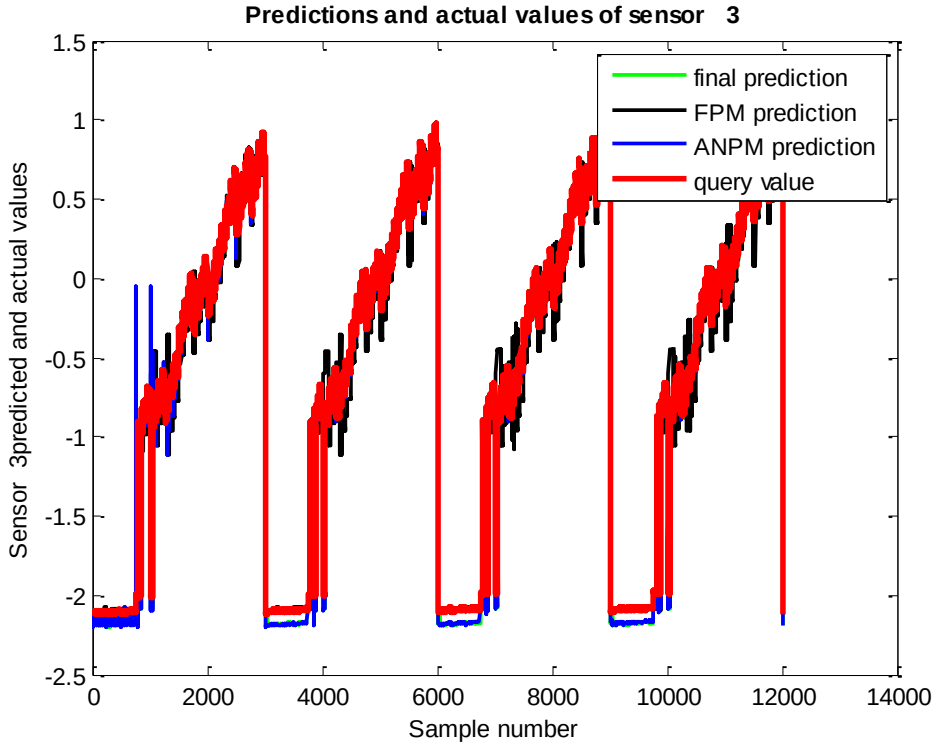
	sensor 1	sensor 2	sensor 3
FPM predictions	0.1092	0.2744	0.0518
ANPM predictions	0.0023	0.0019	0.0043
final predictions	0.0023	0.0011	0.0039

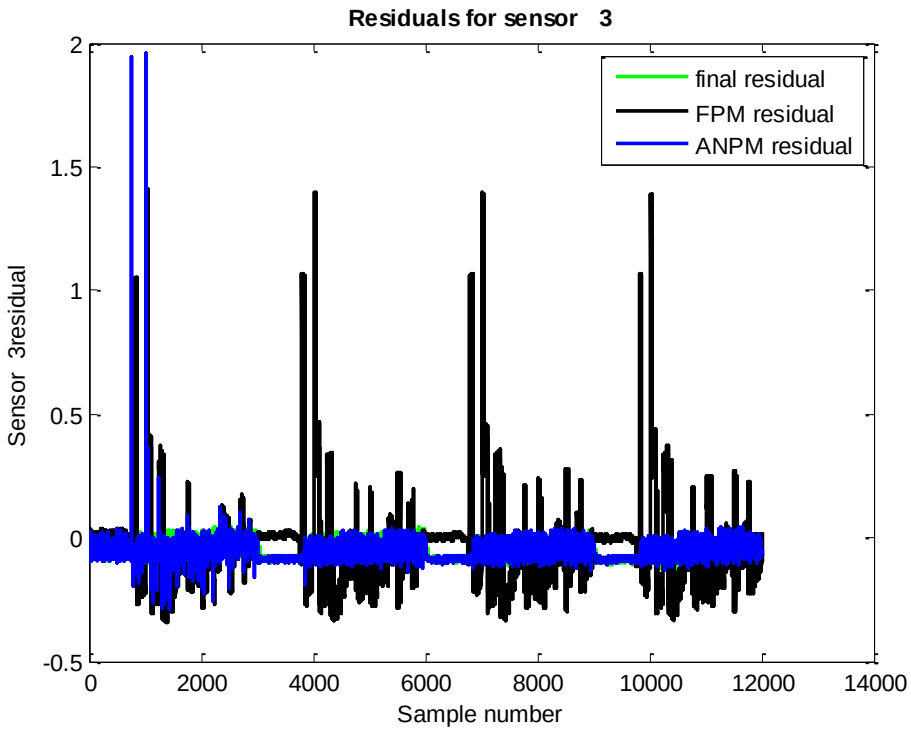



Residuals for sensor 1
final residual
FPM residual
ANPM residual
Sensor 1residual
Sample number
2.5
2
1.5
1
0.5
0
-0.5
-1
-1.5
0
2000
4000
6000
8000
10000
12000
14000


Predictions and actual values of sensor 2
final prediction
FPM prediction
ANPM prediction
query value
Sensor 2predicted and actual values
Sample number
2.5
2
1.5
1
0.5
0
-0.5
-1
-1.5
-2
-2.5
0
2000
4000
6000
8000
10000
12000
14000


Residuals for sensor 2
final residual
FPM residual
ANPM residual
Sensor 2residual
Sample number
4
3.5
3
2.5
2
1.5
1
0.5
0
-0.5
0
2000
4000
6000
8000
10000
12000
14000

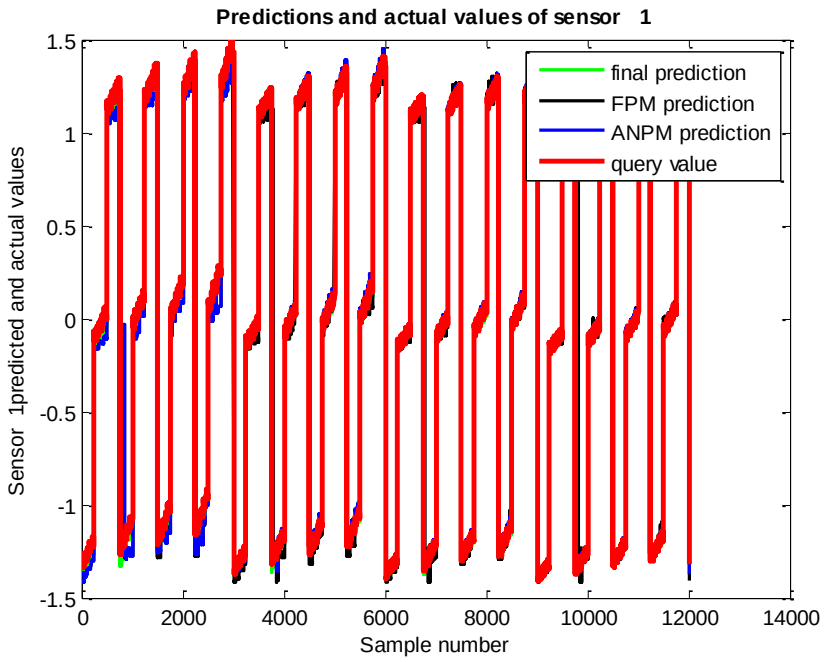

Predictions and actual values of sensor 3
final prediction
FPM prediction
ANPM prediction
query value
Sensor 3predicted and actual values
Sample number
1.5
1
0.5
0
-0.5
-1
-1.5
-2
-2.5
0
2000
4000
6000
8000
10000
12000
14000

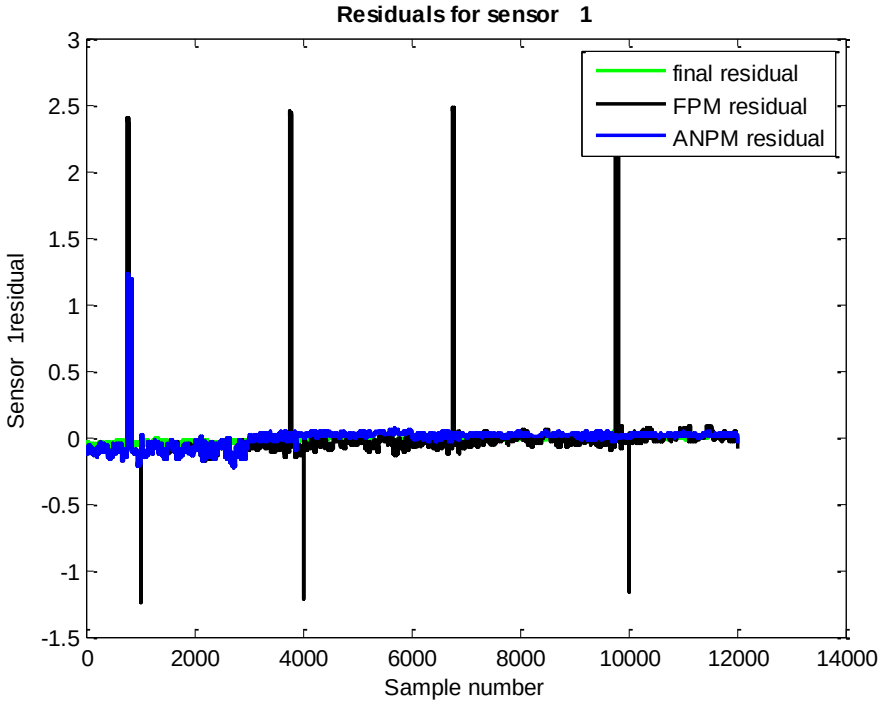


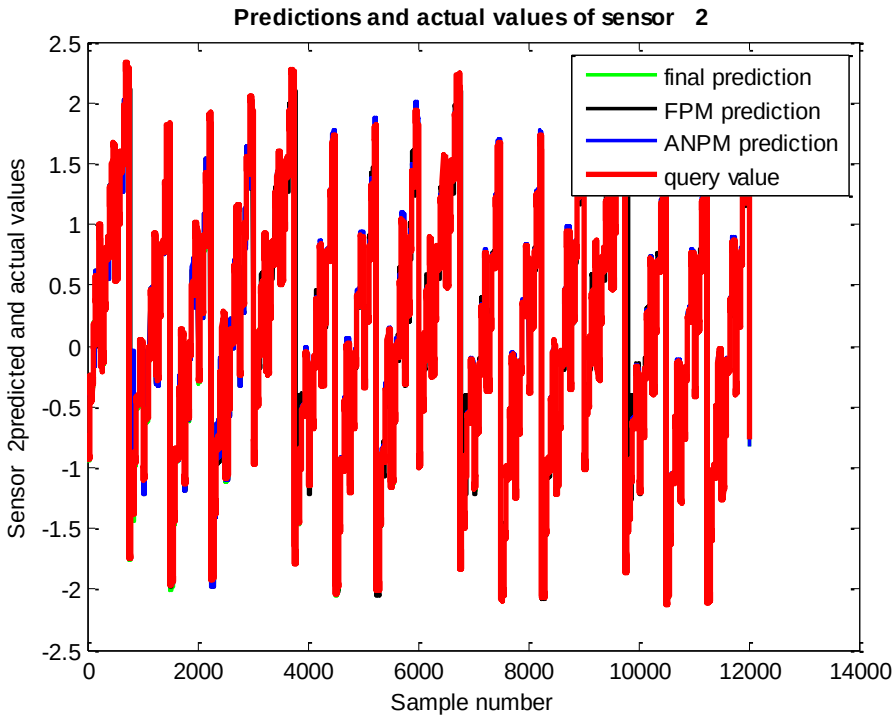
ANPM1_5 outputs for hx_data:

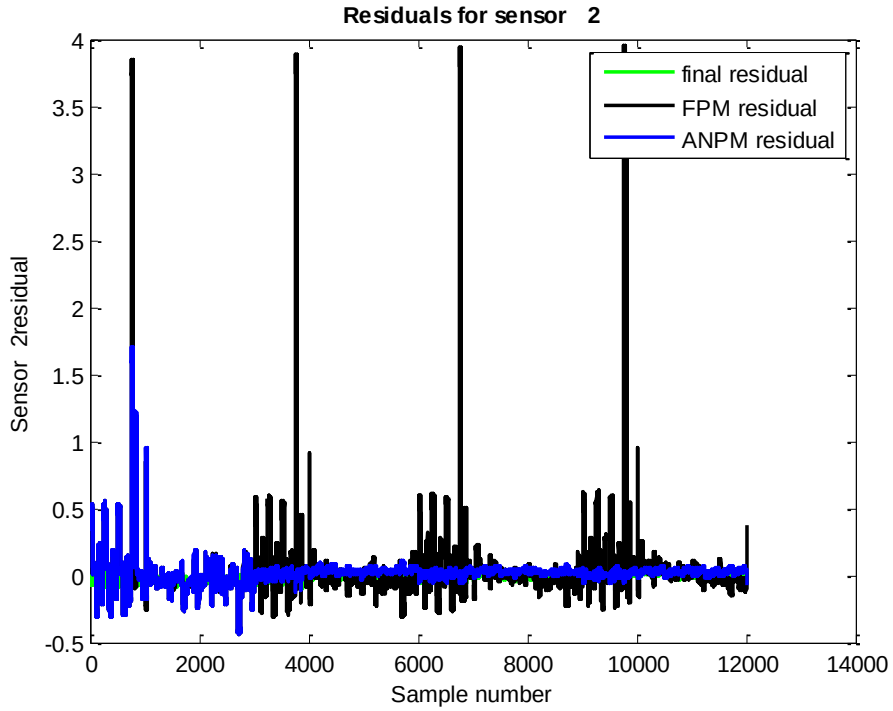
MSE for predictions

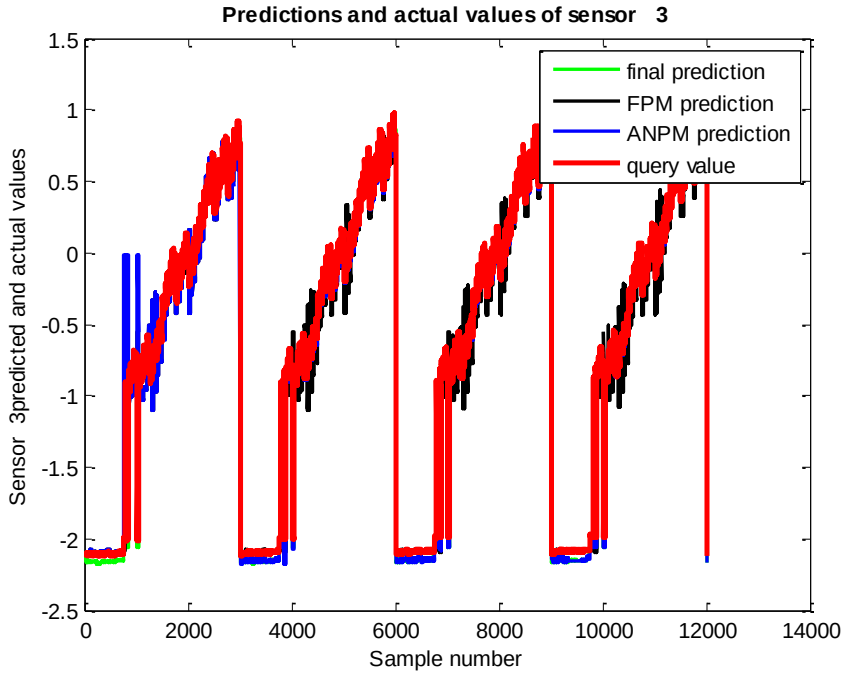
	sensor 1	sensor 2	sensor 3
FPM predictions	0.1092	0.2734	0.06076
ANPM predictions	0.0123	0.0227	0.0384
final predictions	0.0007	0.0006	0.002

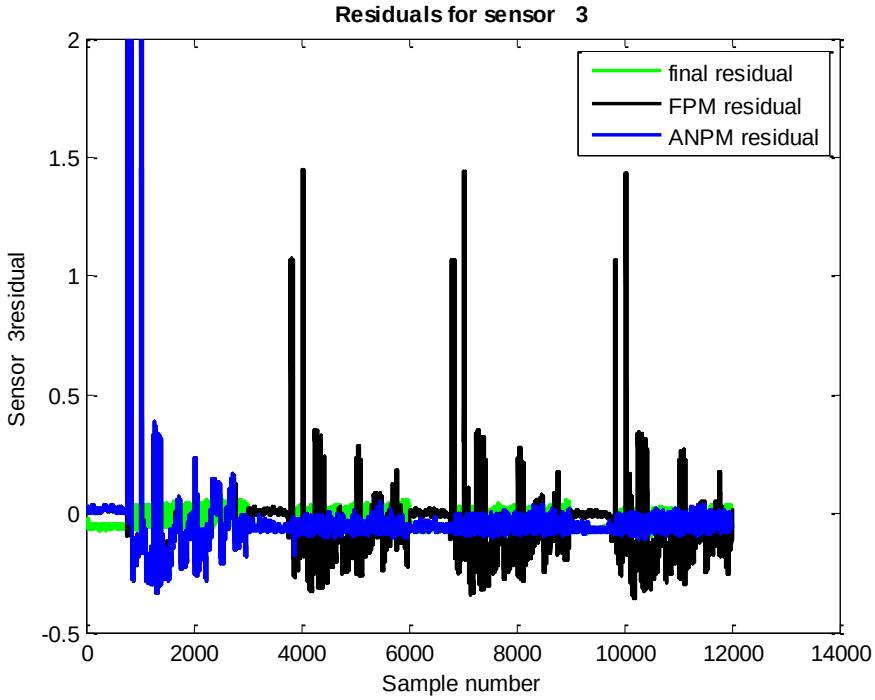

Predictions and actual values of sensor 1
final prediction
FPM prediction
ANPM prediction
query value
Sensor 1predicted and actual values
Sample number
1.5
1
0.5
0
-0.5
-1
-1.5
0
2000
4000
6000
8000
10000
12000
14000


Residuals for sensor 1
final residual
FPM residual
ANPM residual
Sensor 1residual
Sample number
3
2.5
2
1.5
1
0.5
0
-0.5
-1
-1.5
0
2000
4000
6000
8000
10000
12000
14000

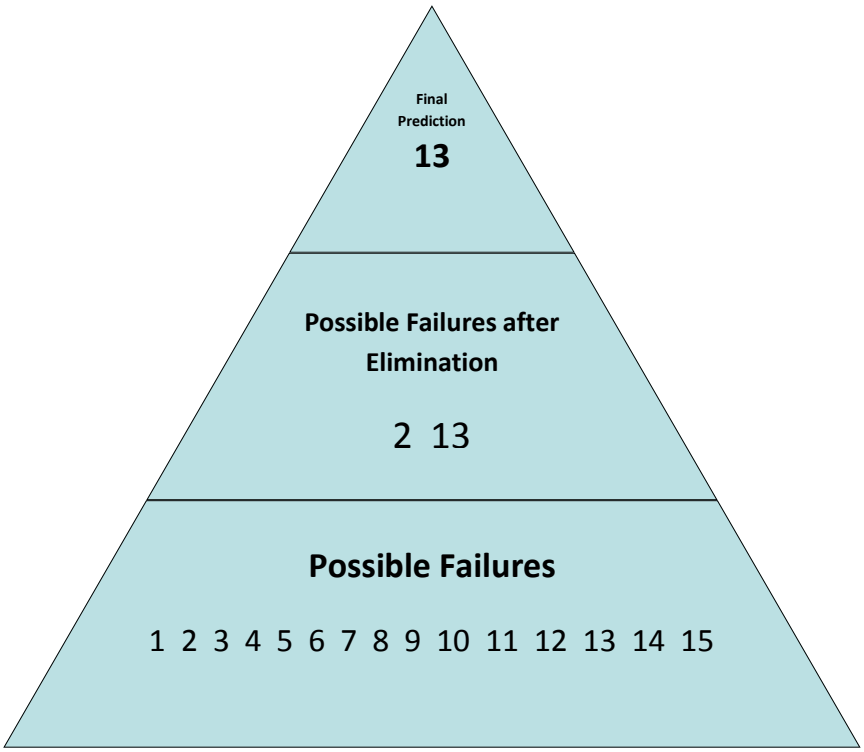

Predictions and actual values of sensor 2
final prediction
FPM prediction
ANPM prediction
query value
Sensor 2predicted and actual values
Sample number
2.5
2
1.5
1
0.5
0
-0.5
-1
-1.5
-2
-2.5
0
2000
4000
6000
8000
10000
12000
14000

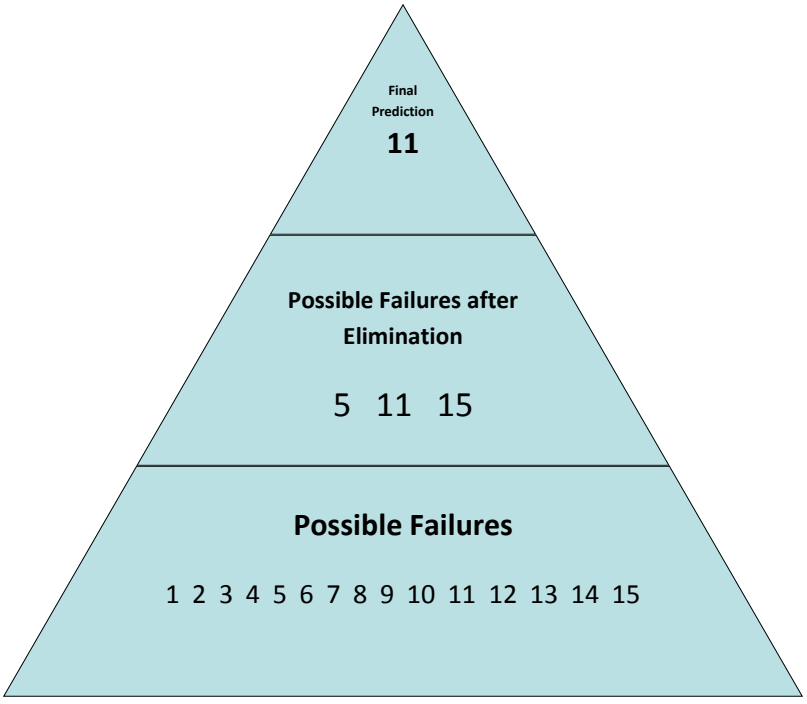

Residuals for sensor 2
final residual
FPM residual
ANPM residual
Sensor 2residual
Sample number
4
3.5
3
2.5
2
1.5
1
0.5
0
-0.5
0
2000
4000
6000
8000
10000
12000
14000


Predictions and actual values of sensor 3
final prediction
FPM prediction
ANPM prediction
query value
Sensor 3predicted and actual values
Sample number
1.5
1
0.5
0
-0.5
-1
-1.5
-2
-2.5
0
2000
4000
6000
8000
10000
12000
14000

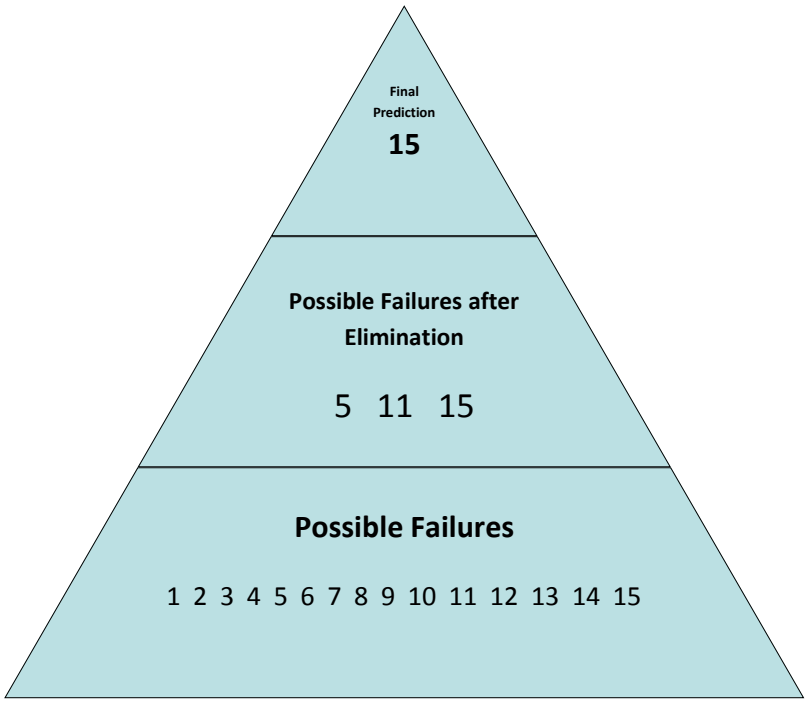

Residuals for sensor 3
final residual
FPM residual
ANPM residual
Sensor 3residual
Sample number
2
1.5
1
0.5
0
-0.5
0
2000
4000
6000
8000
10000
12000
14000

Diagnostic Figures





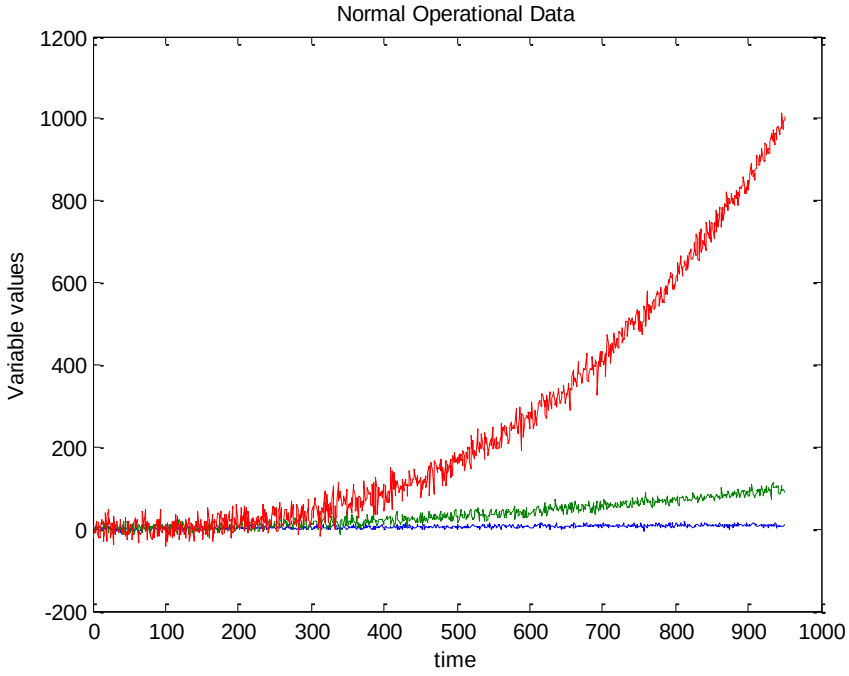
Correct Failure is classified.

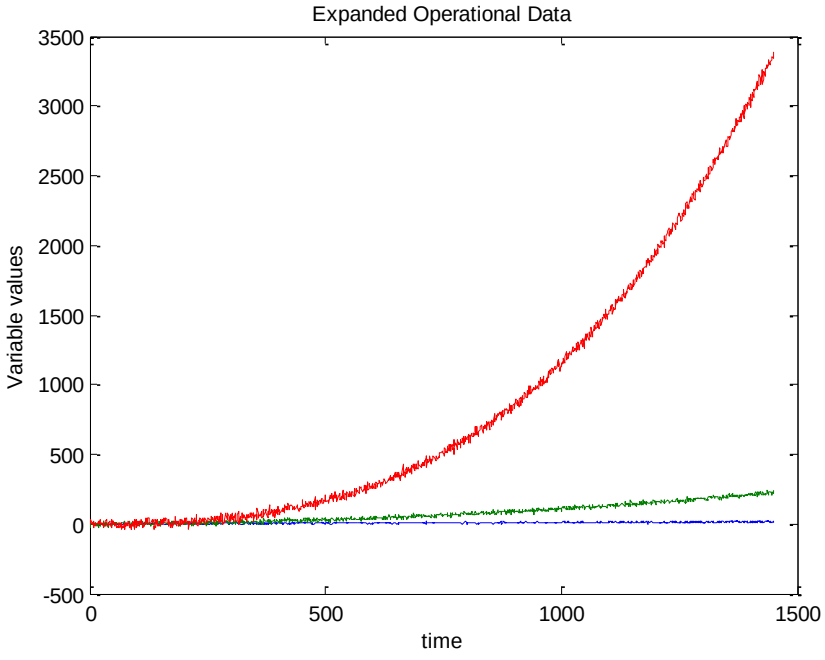


Correct Failure is classified.

KPCA Simple Test

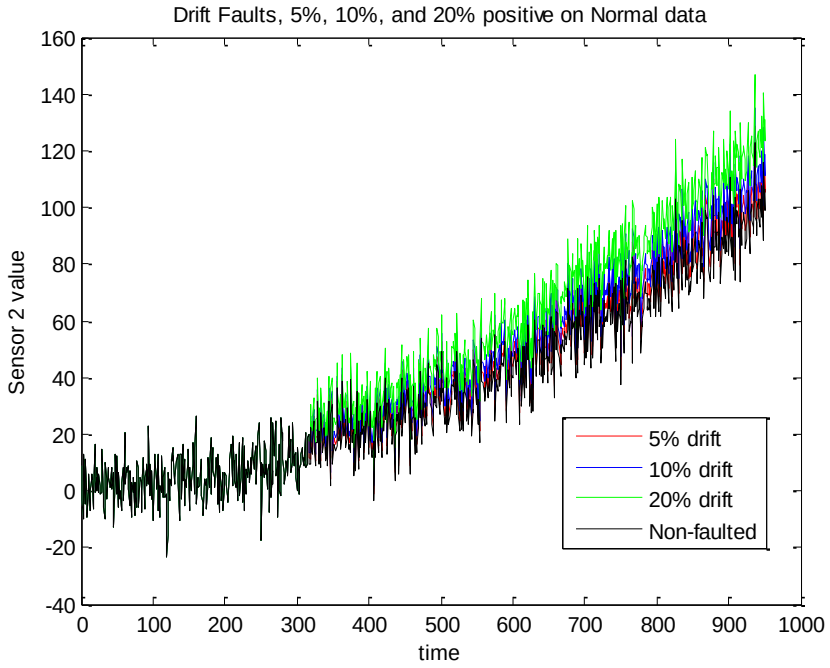
Simple 3 sensor data set.

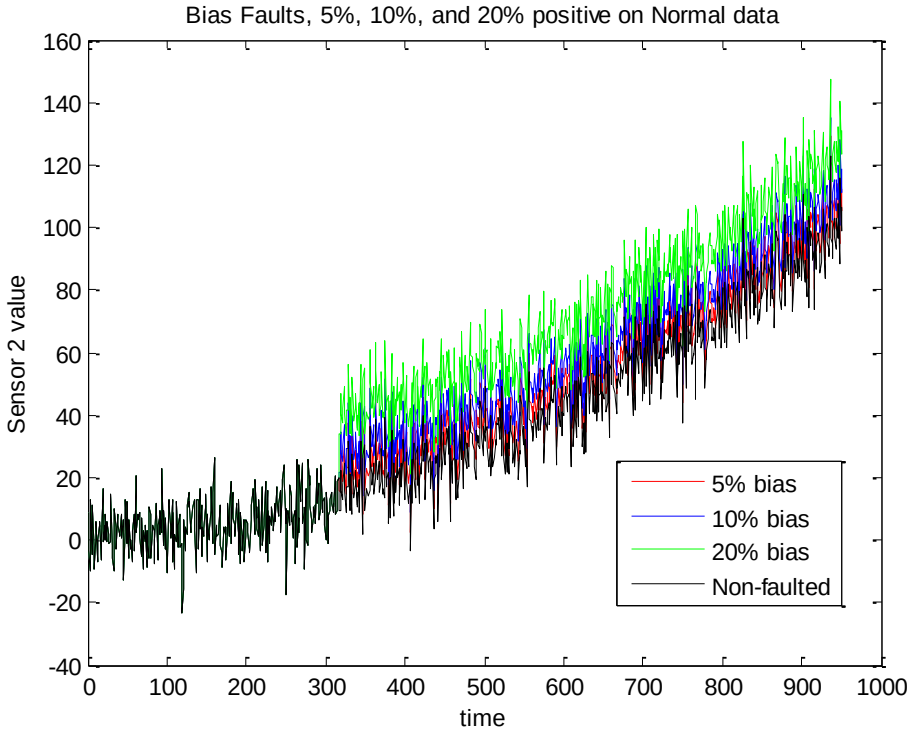


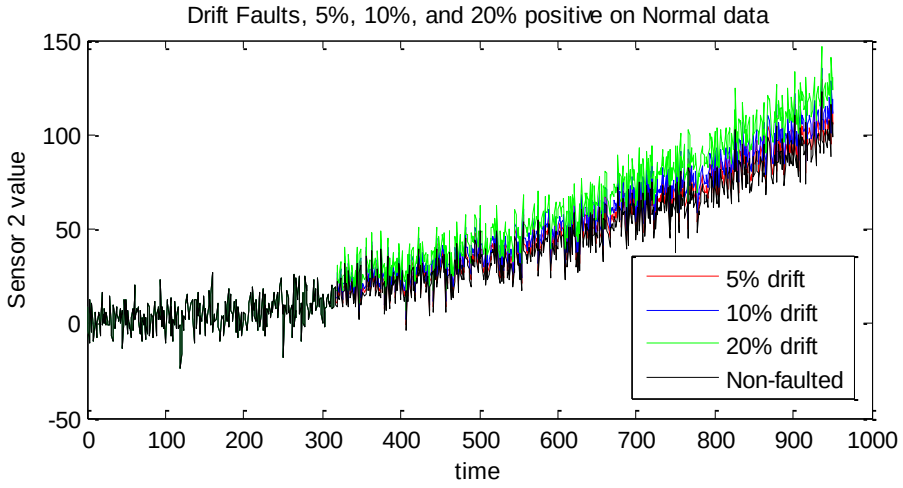


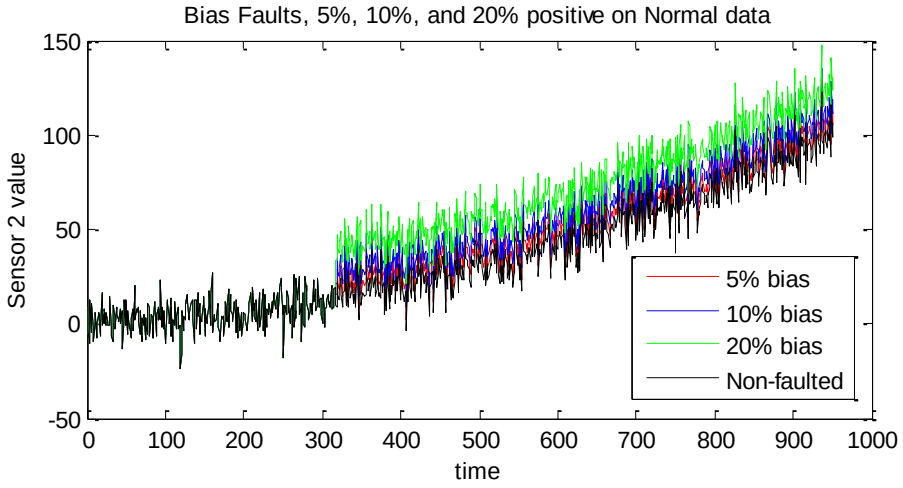
Faults Added to Sensor 2 of both the Normal condition and Expanded Condition

Normal Condition:

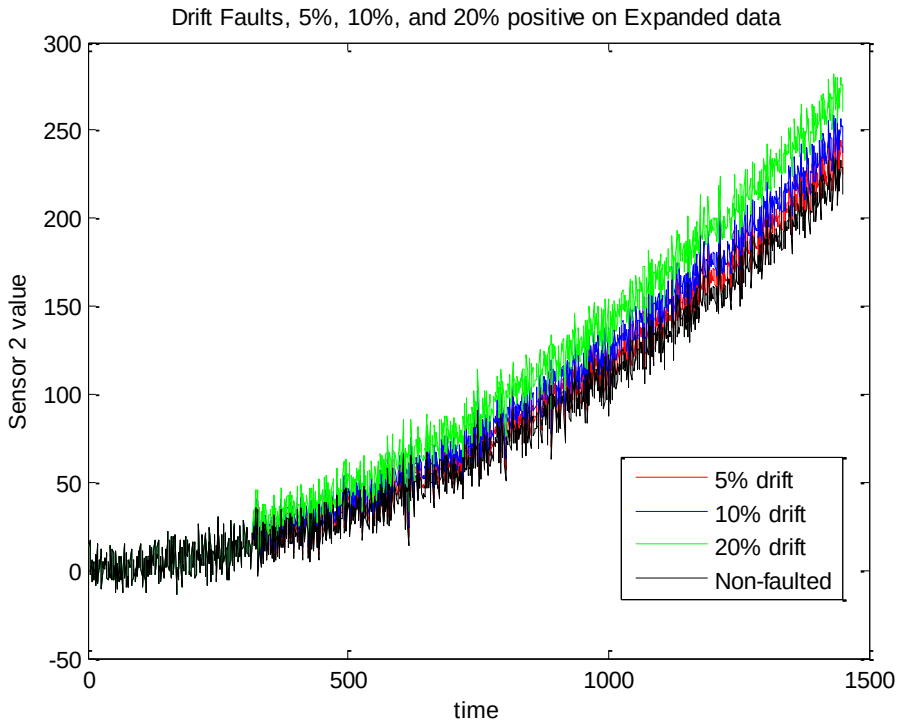


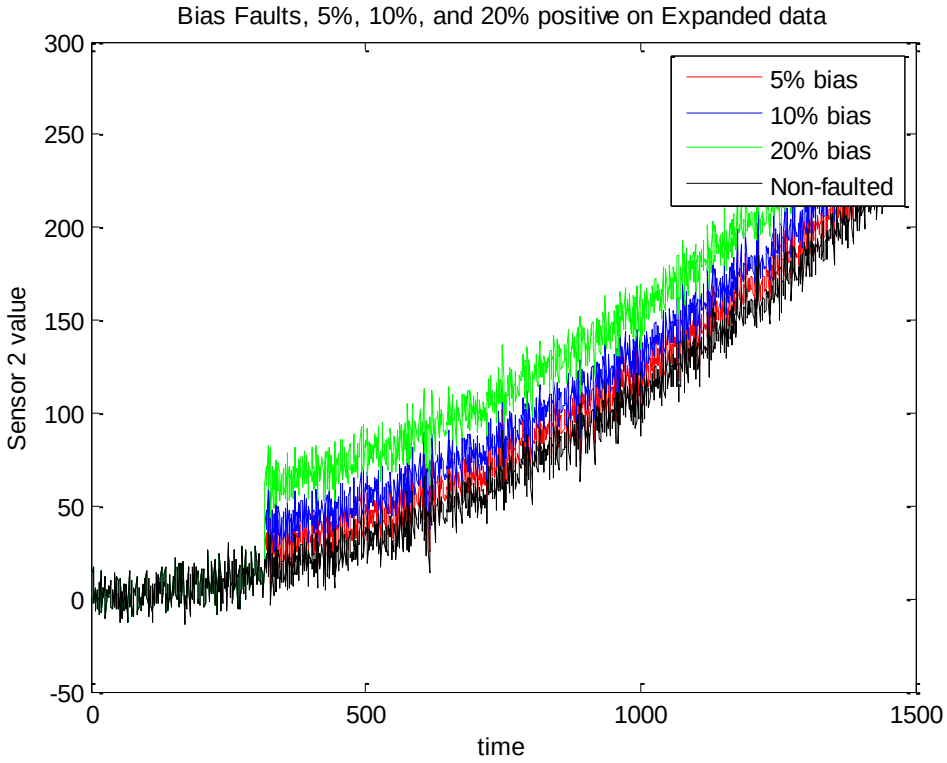


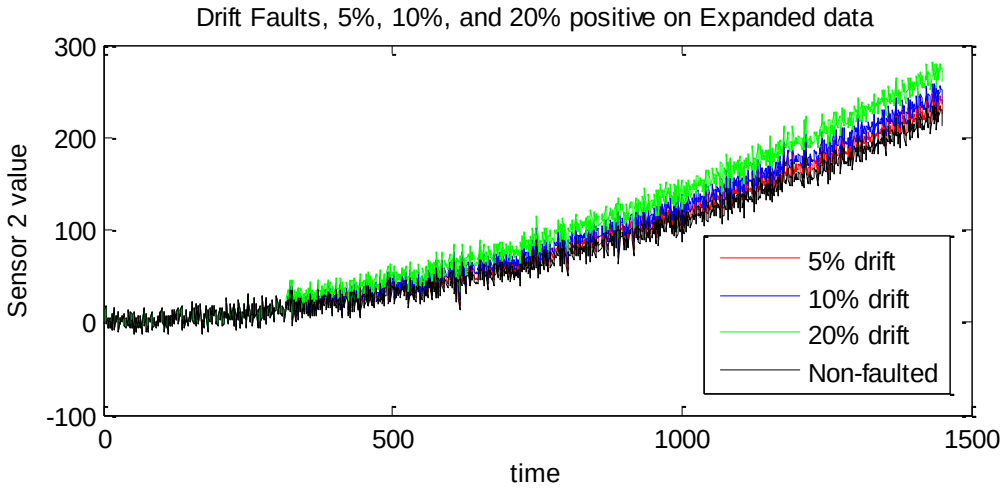

Drift Faults, 5%, 10%, and 20% positive on Normal data
Sensor 2 value
time
5% drift
10% drift
20% drift
Non-faulted

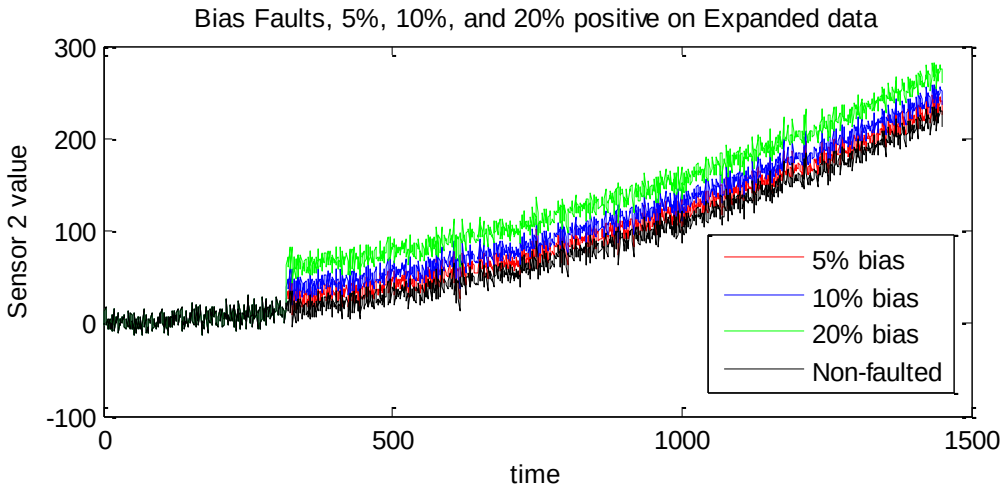

Bias Faults, 5%, 10%, and 20% positive on Normal data
Sensor 2 value
time
5% bias
10% bias
20% bias
Non-faulted

Expanded Condition Faults:






Drift Faults, 5%, 10%, and 20% positive on Expanded data
Sensor 2 value
time
300
200
100
0
-100
0
500
1000
1500
5% drift
10% drift
20% drift
Non-faulted


Bias Faults, 5%, 10%, and 20% positive on Expanded data
Sensor 2 value
time
300
200
100
0
-100
0
500
1000
1500
5% bias
10% bias
20% bias
Non-faulted

Performing PCA ECM to see if the expanded faults can be detected and differentiated from the expanded condition

Applying PCA to the normal condition gives:

explained =

83.4394

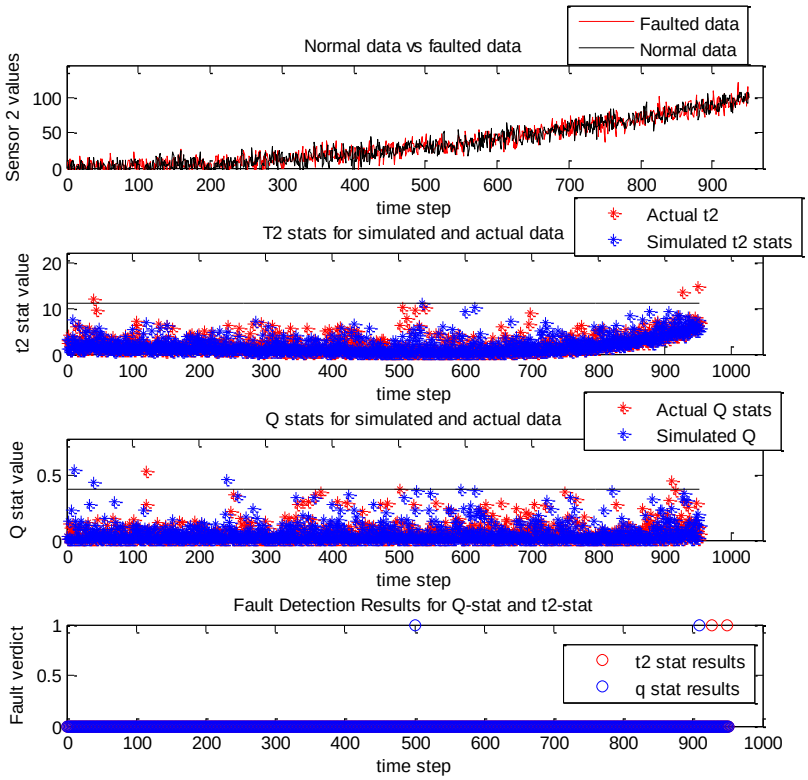
14.9518

1.6087

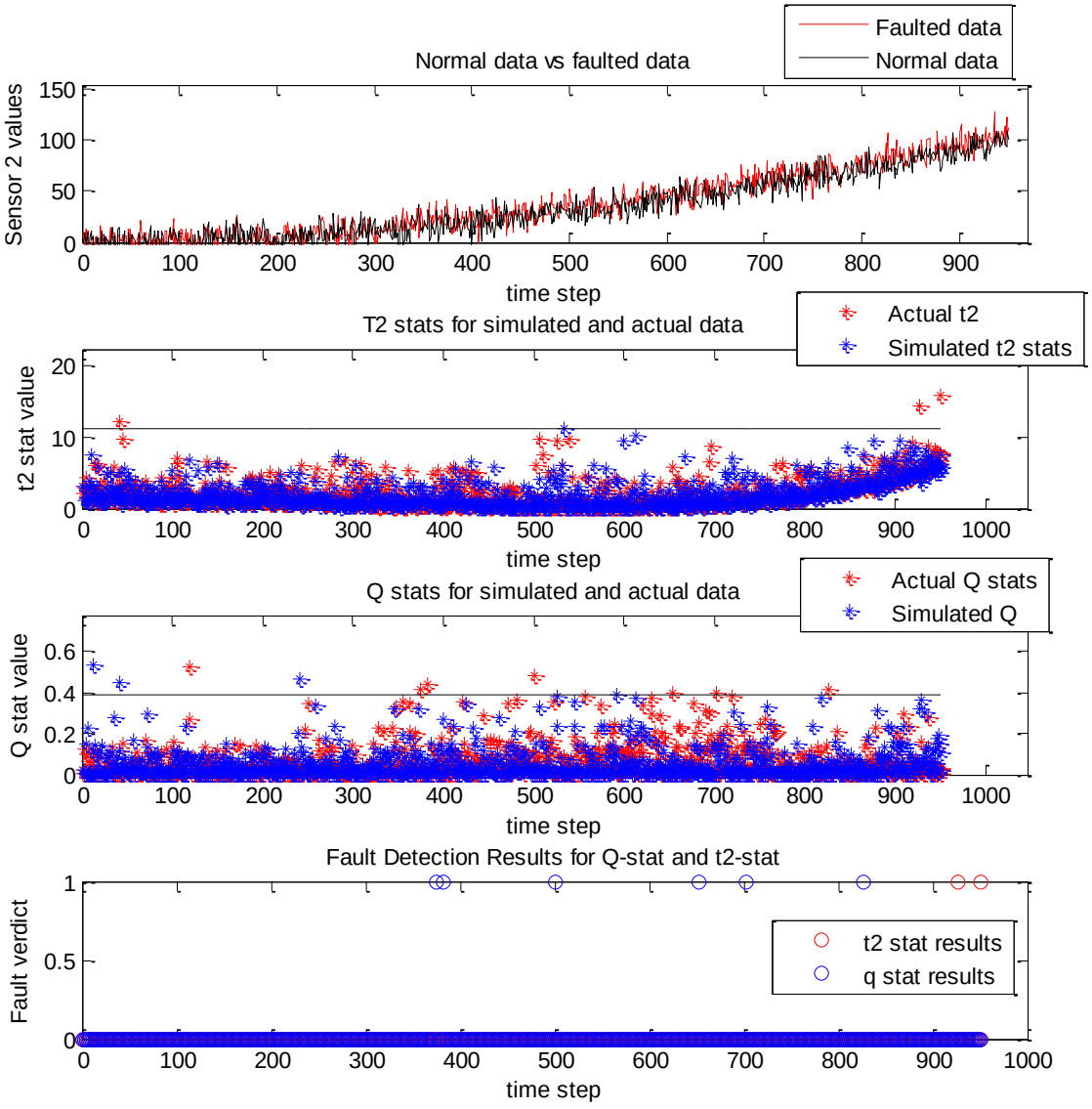
So the first two PCs are used.

Applying the PCA ECM to the normal faulted data:

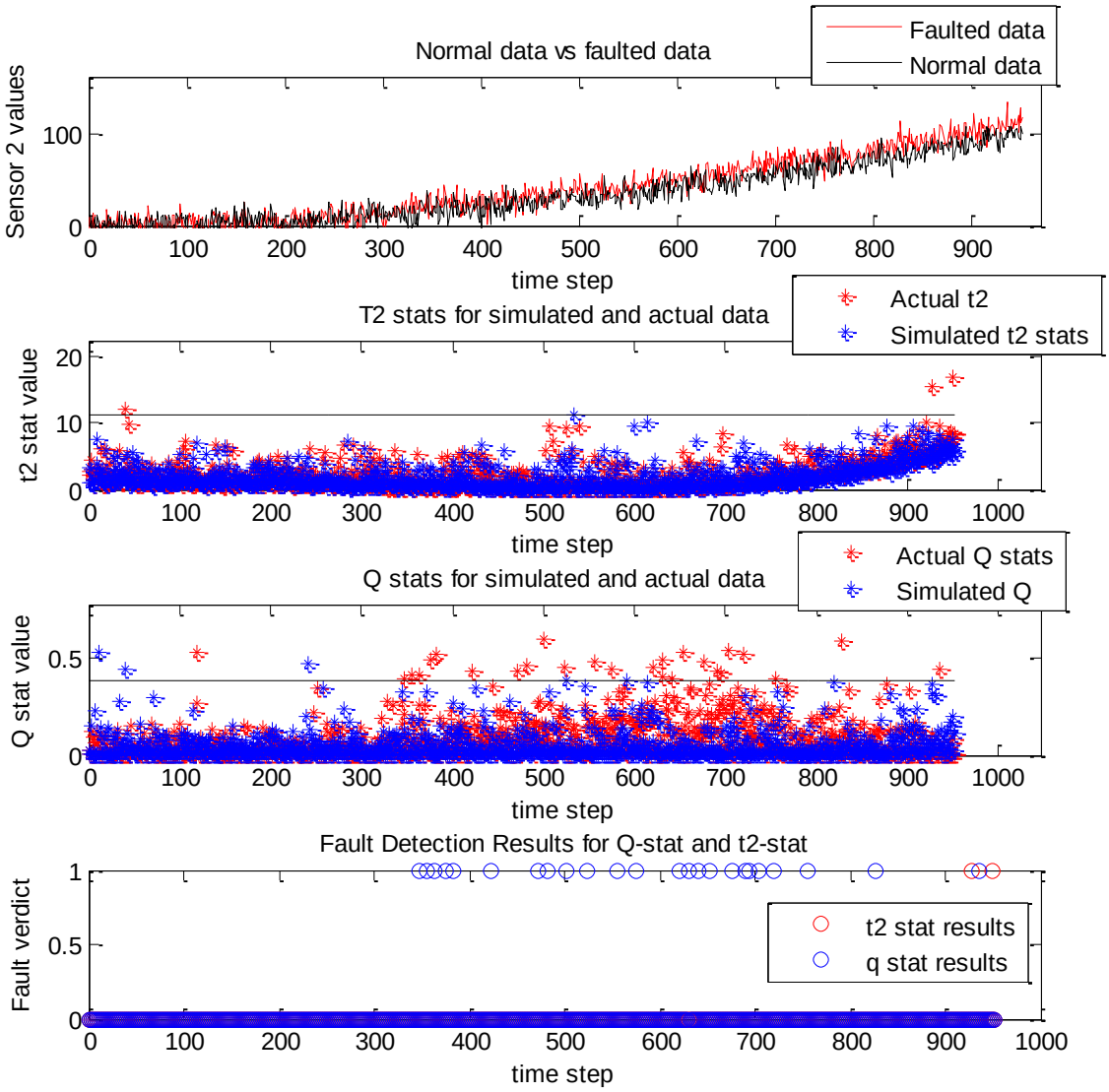
Non-Faulted Data:



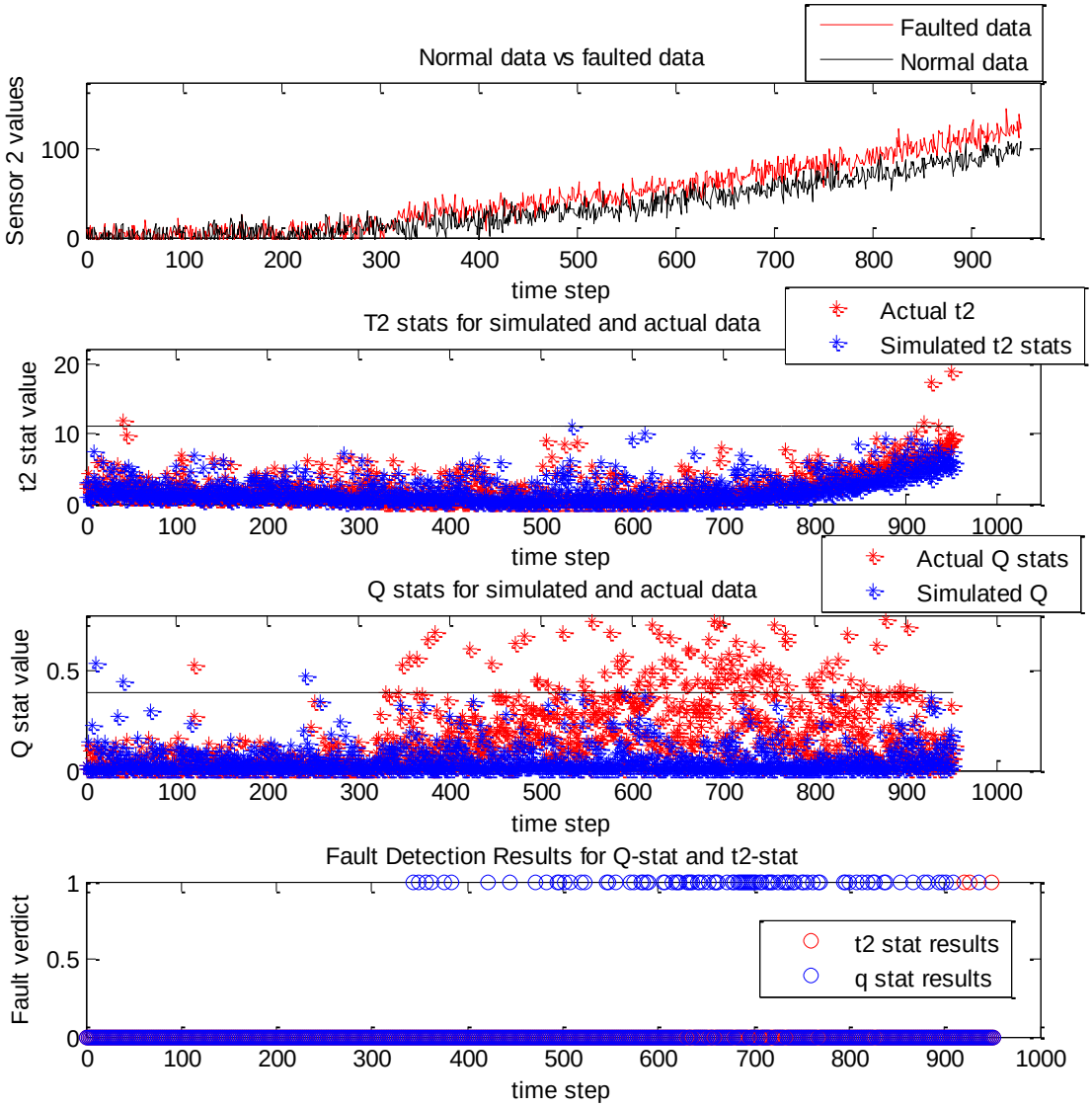
5% Drift:


Normal data vs faulted data
Faulted data
Normal data
Sensor 2 values
time step
T2 stats for simulated and actual data
Actual t2
Simulated t2 stats
t2 stat value
time step
Q stats for simulated and actual data
Actual Q stats
Simulated Q
Q stat value
time step
Fault Detection Results for Q-stat and t2-stat
t2 stat results
q stat results
Fault verdict
time step

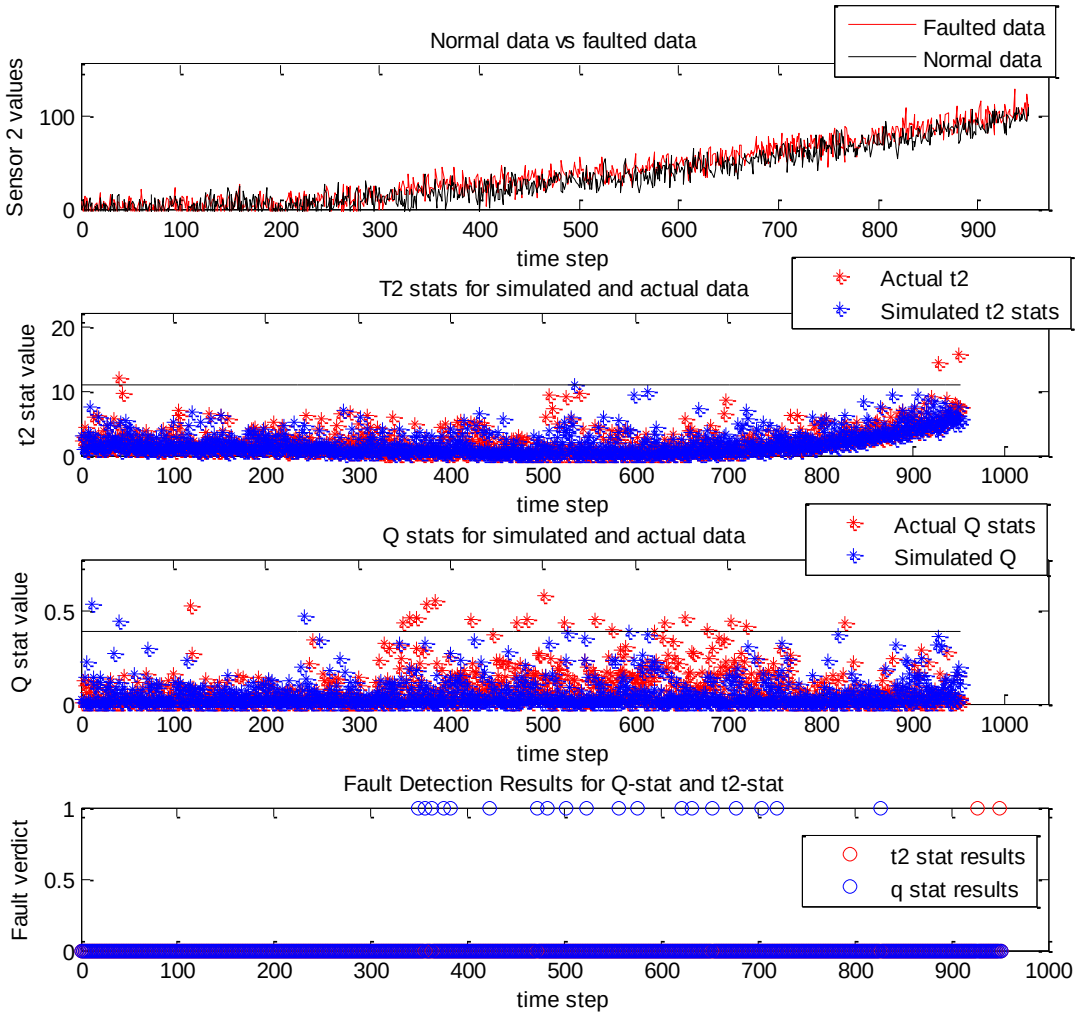
10% Drift:


Normal data vs faulted data
Faulted data
Normal data
Sensor 2 values
time step
T2 stats for simulated and actual data
Actual t2
Simulated t2 stats
t2 stat value
time step
Q stats for simulated and actual data
Actual Q stats
Simulated Q
Q stat value
time step
Fault Detection Results for Q-stat and t2-stat
t2 stat results
q stat results
Fault verdict
time step

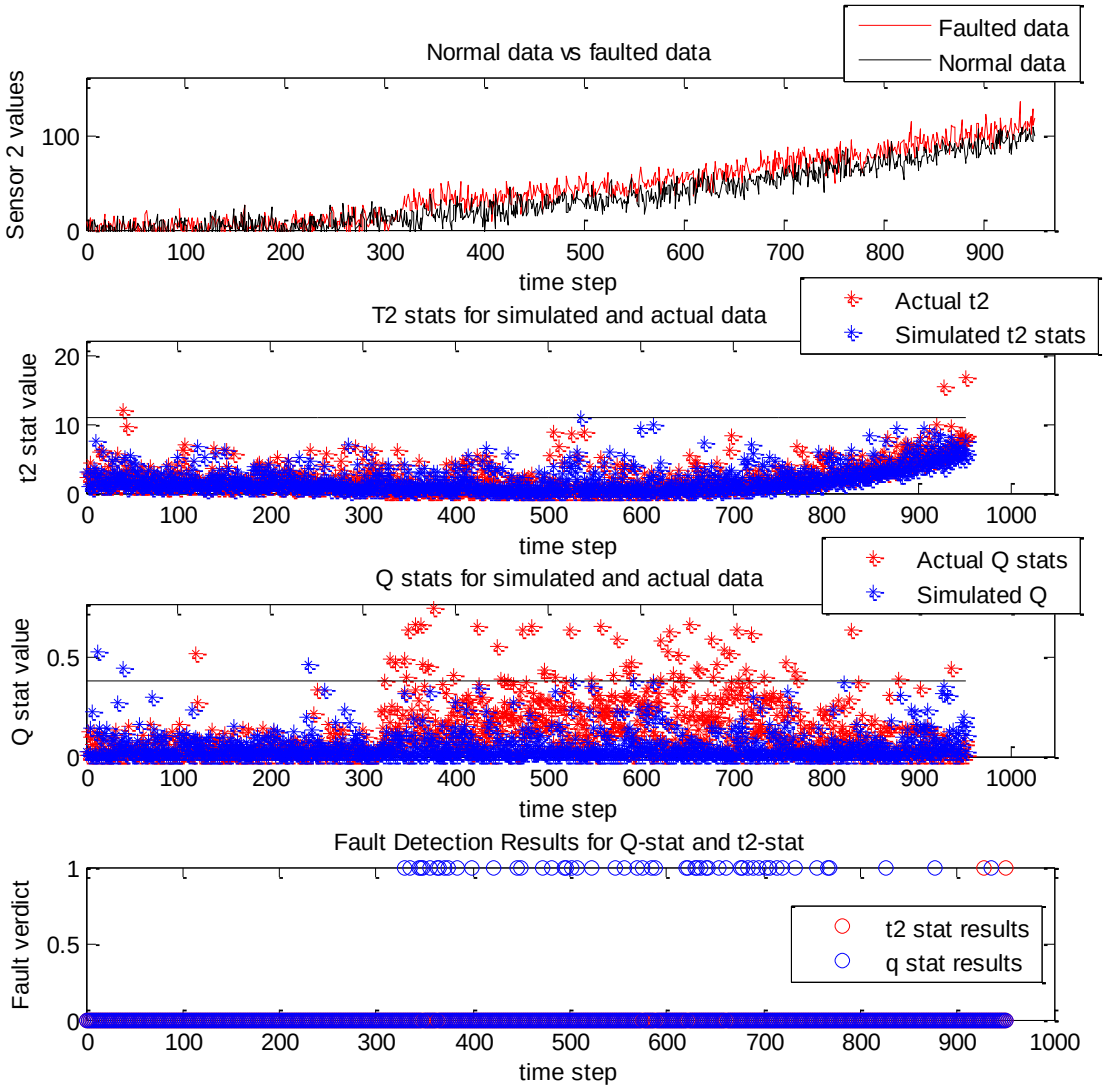
20% Drift:


Normal data vs faulted data
Faulted data
Normal data
Sensor 2 values
time step
T2 stats for simulated and actual data
Actual t2
Simulated t2 stats
t2 stat value
time step
Q stats for simulated and actual data
Actual Q stats
Simulated Q
Q stat value
time step
Fault Detection Results for Q-stat and t2-stat
t2 stat results
q stat results
Fault verdict
time step

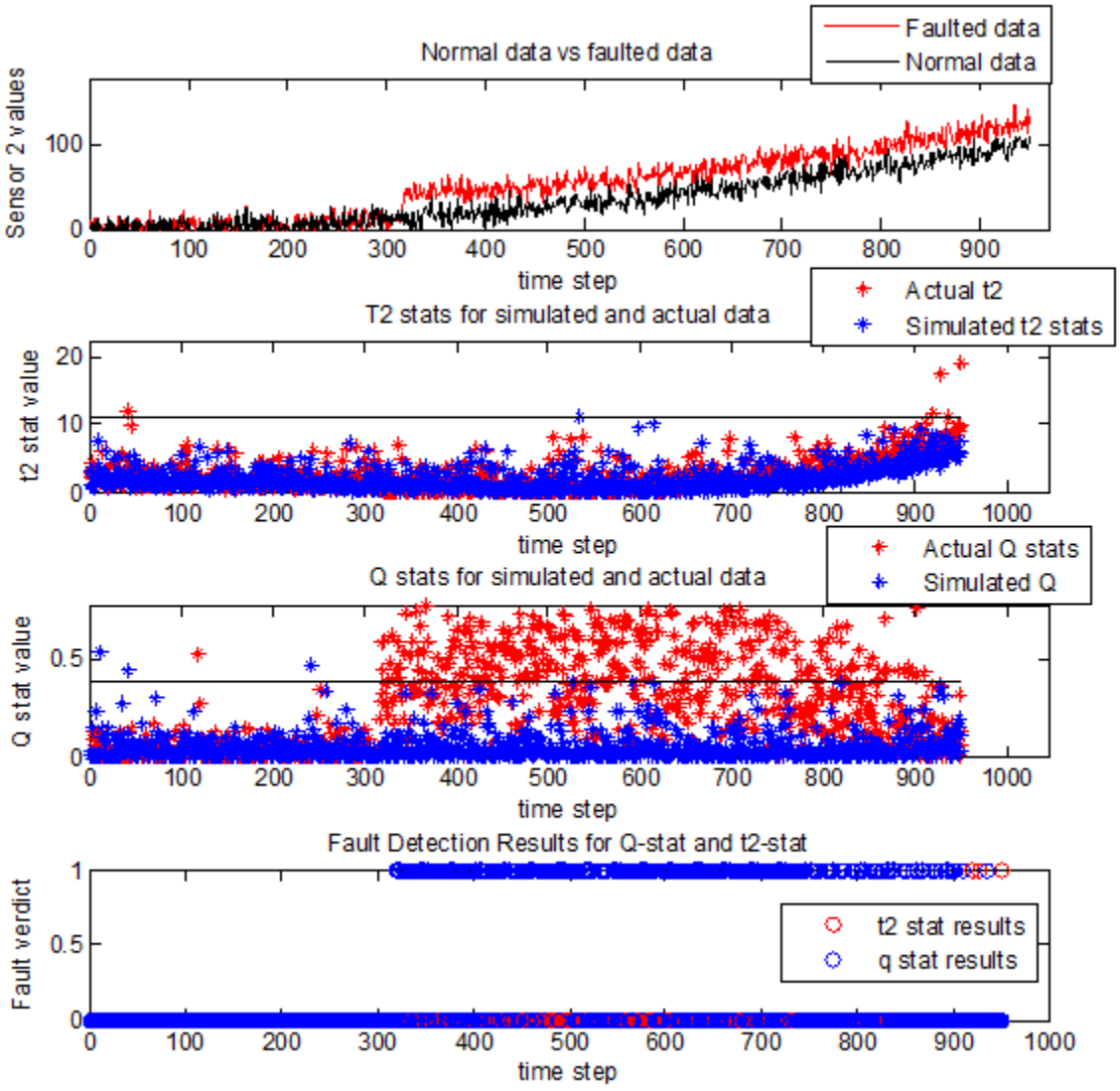
5% Bias Fault:


Normal data vs faulted data
Faulted data
Normal data
Sensor 2 values
time step
T2 stats for simulated and actual data
Actual t2
Simulated t2 stats
t2 stat value
time step
Q stats for simulated and actual data
Actual Q stats
Simulated Q
Q stat value
time step
Fault Detection Results for Q-stat and t2-stat
t2 stat results
q stat results
Fault verdict
time step

10% Bias Fault:

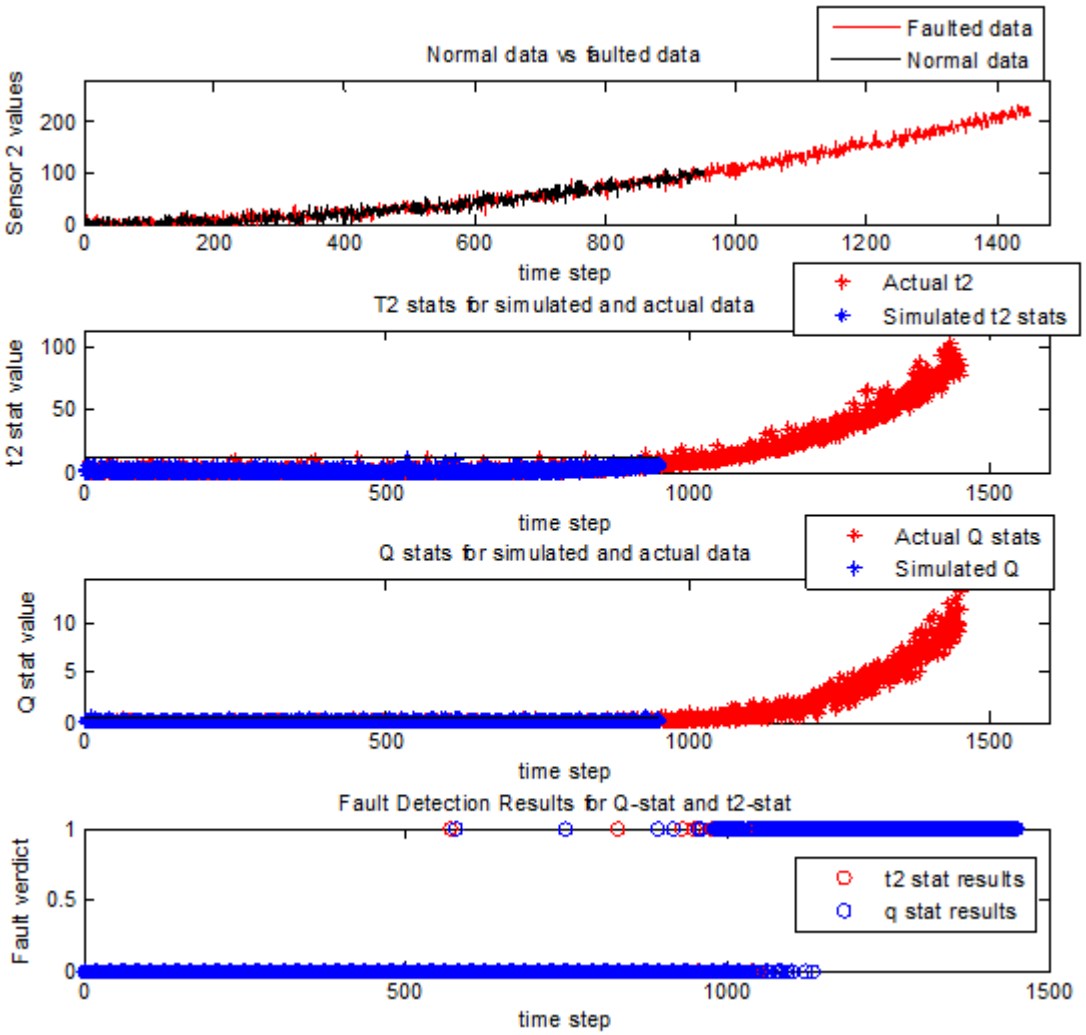


20% Bias Fault:



Now applying the PCA ECM technique to the Expanded Condition, and the Expanded Condition with Faults.

Non Faulted Expanded Condition:



So the normal PCA ECM cannot be used for nonlinear relationships.

So now trying a Kernel PCA ECM to see how it works on the Normal Faulted data and the Expanded Condition Faulted Data. There are a number of kernels that could be applied.

$$k(x_i, x_j) = \left(x_i \bullet x_j\right)^d$$

$$k(x_i, x_j) = \left(x_i \bullet x_j + 1\right)^d$$

$$k(x_i, x_j) = \exp\left(-\frac{\left\|x_i - x_j\right\|^2}{c}\right)$$

$$k(x_i, x_j) = \exp\left(-\frac{\left\|x_i - x_j\right\|^2}{2\sigma^2}\right)$$

So first trying the polynomial kernel: $k(x_i, x_j) = \left(x_i \bullet x_j\right)^d$ **and setting d=1, 2, 3.**

We get the following results for d=1:

explained =

94.6216

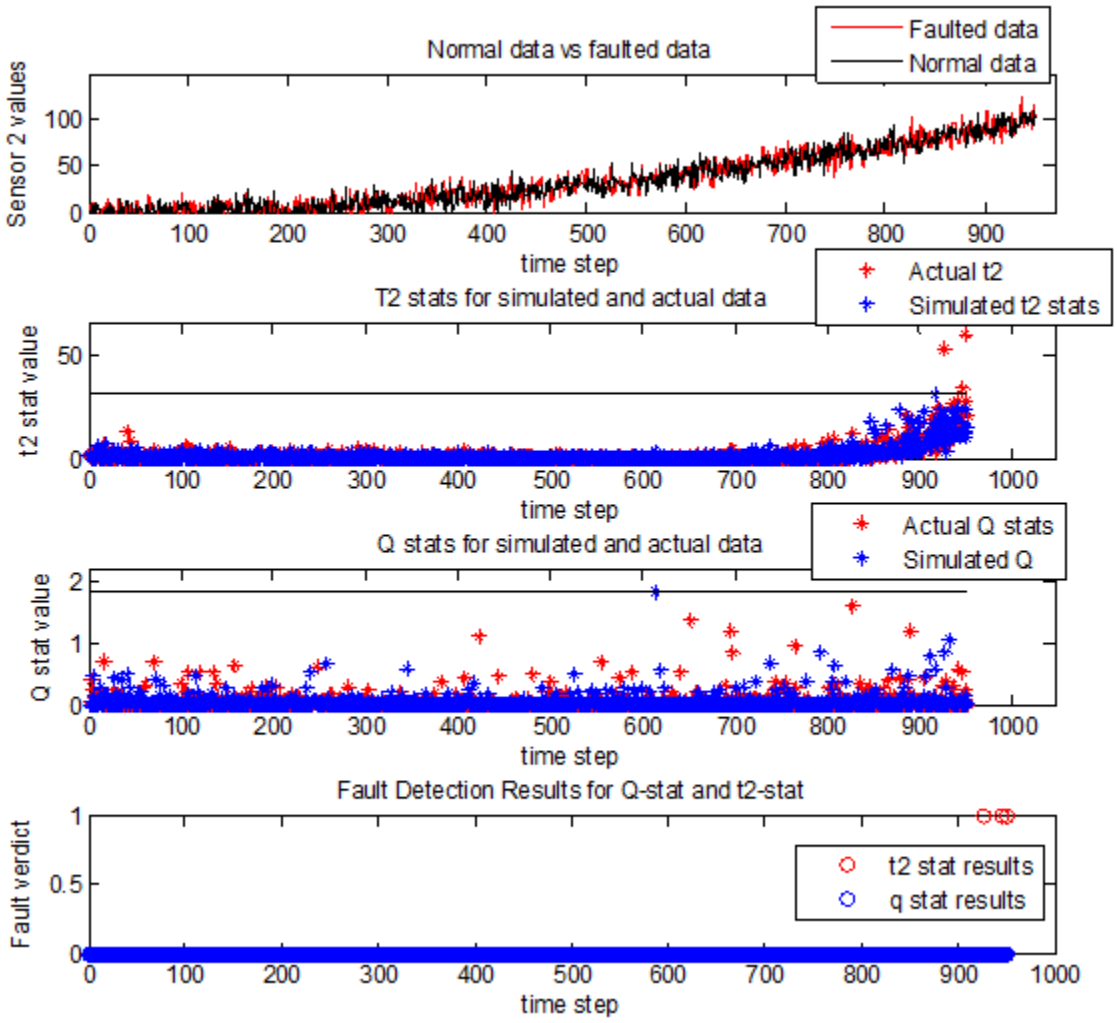
4.6838

0.6946

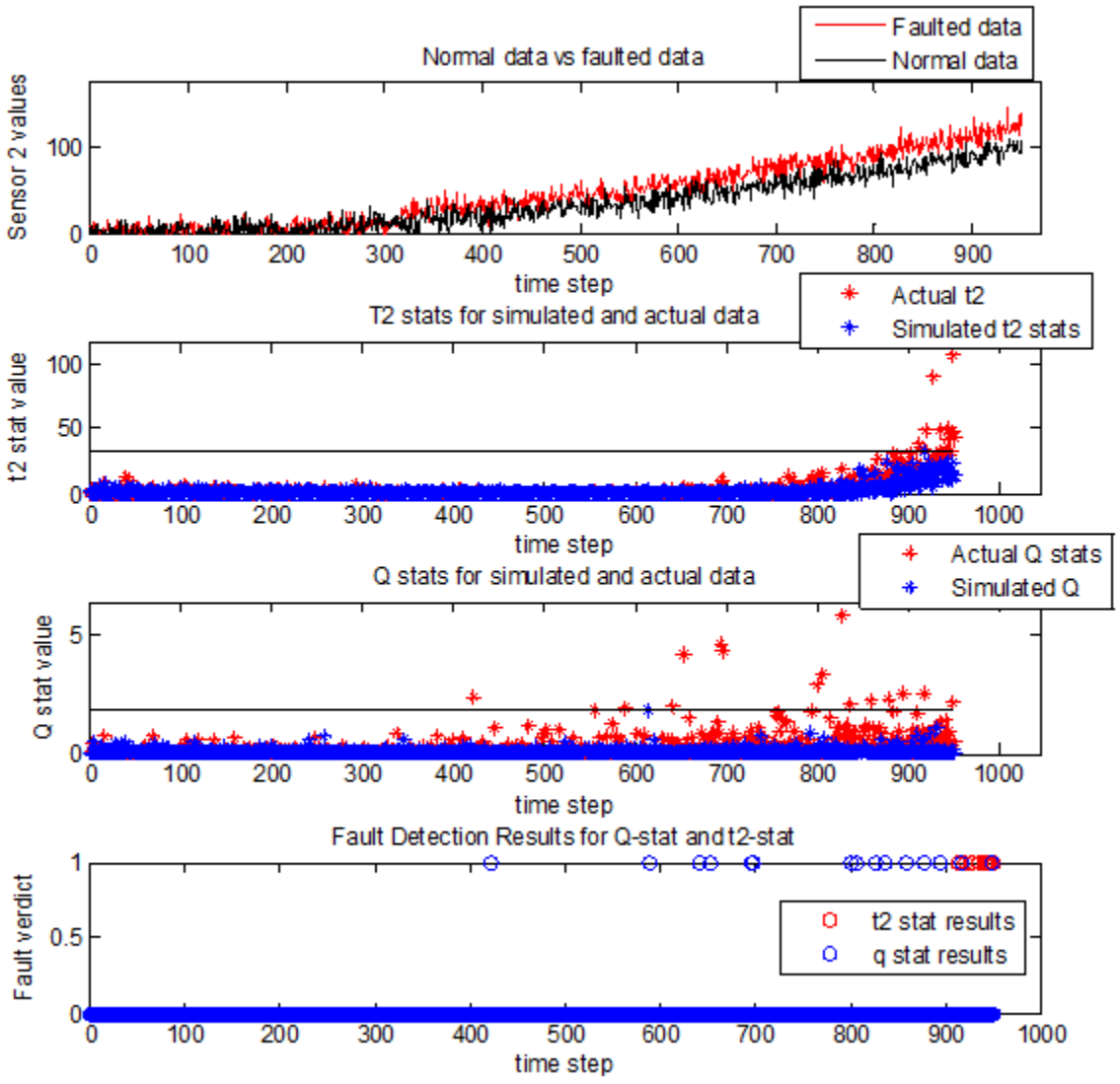
So 2 PCs are used.

Applying this we get:

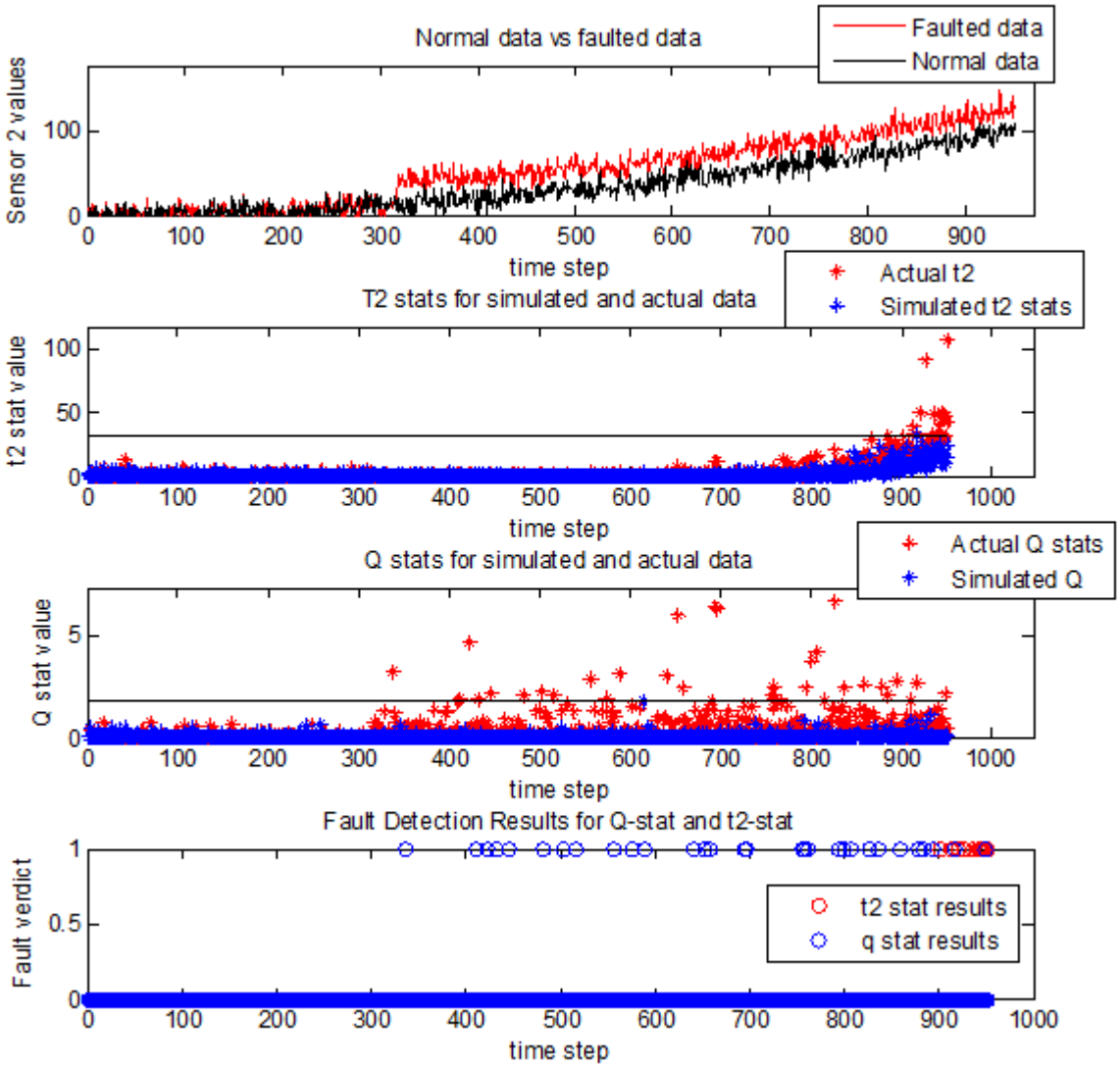
NonFaulted data:



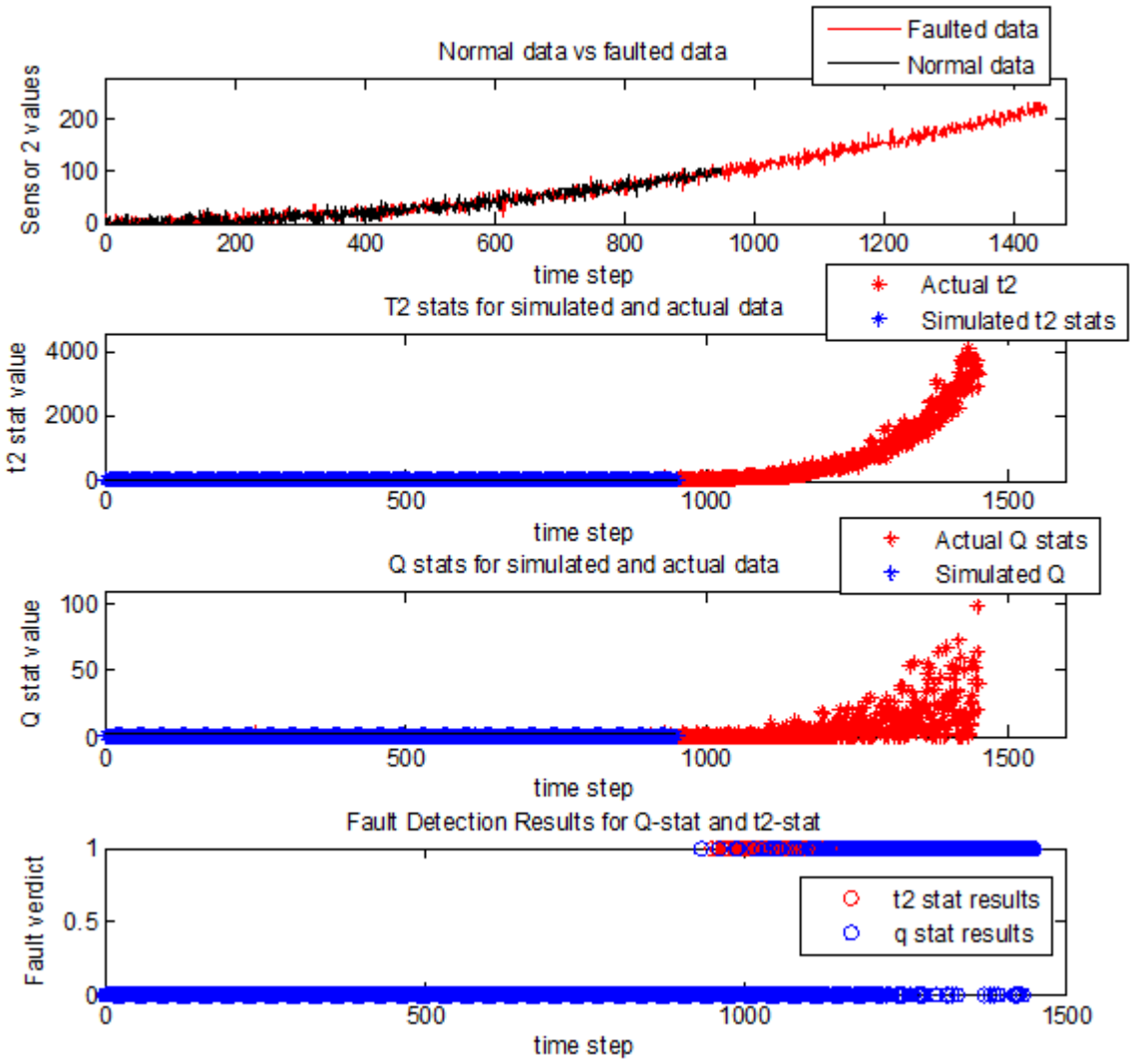
20% Drift:


Normal data vs faulted data
Faulted data
Normal data
Sensor 2 values
time step
T2 stats for simulated and actual data
Actual t2
Simulated t2 stats
t2 stat value
time step
Q stats for simulated and actual data
Actual Q stats
Simulated Q
Q stat value
time step
Fault Detection Results for Q-stat and t2-stat
t2 stat results
q stat results
Fault verdict
time step

20% Bias:


Normal data vs faulted data
Faulted data
Normal data
Sensor 2 values
time step
T2 stats for simulated and actual data
Actual t2
Simulated t2 stats
t2 stat value
time step
Q stats for simulated and actual data
Actual Q stats
Simulated Q
Q stat value
time step
Fault Detection Results for Q-stat and t2-stat
t2 stat results
q stat results
Fault verdict
time step

Expanded Non-Faulted Condition:



So next trying the polynomial kernel with d=2.

explained =

80.4724

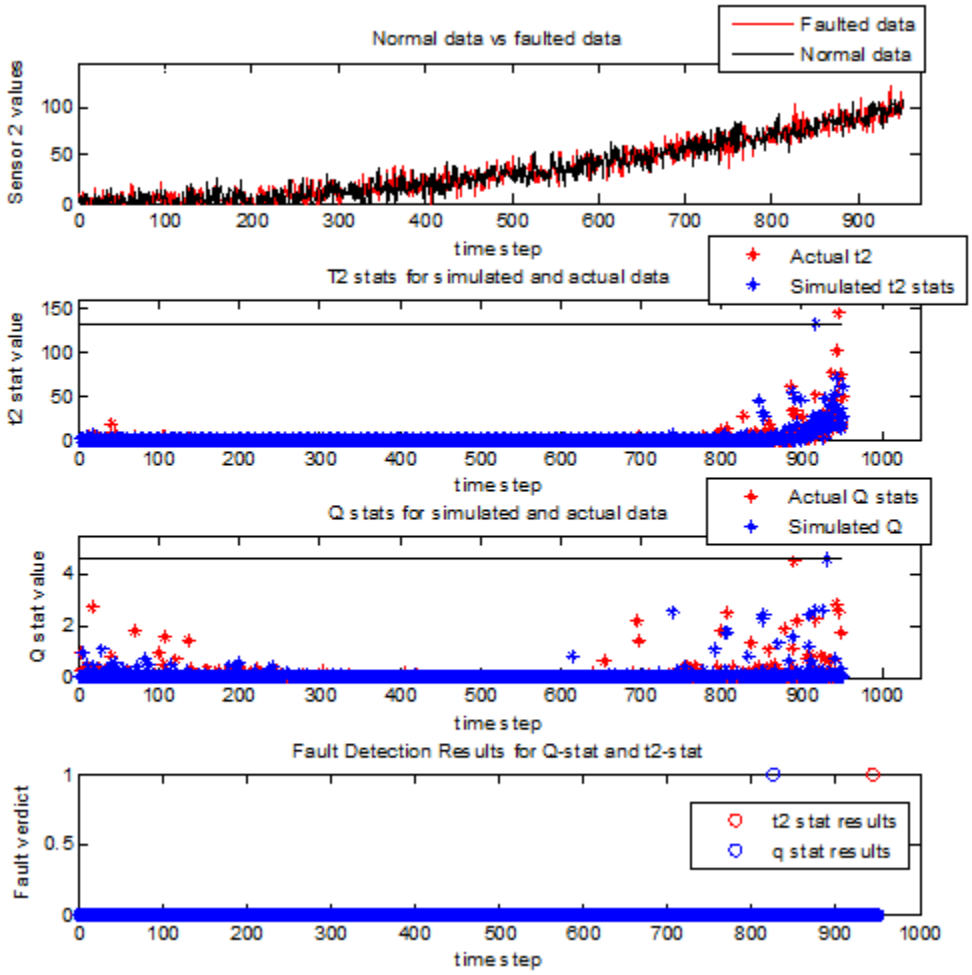
17.4885

2.0392

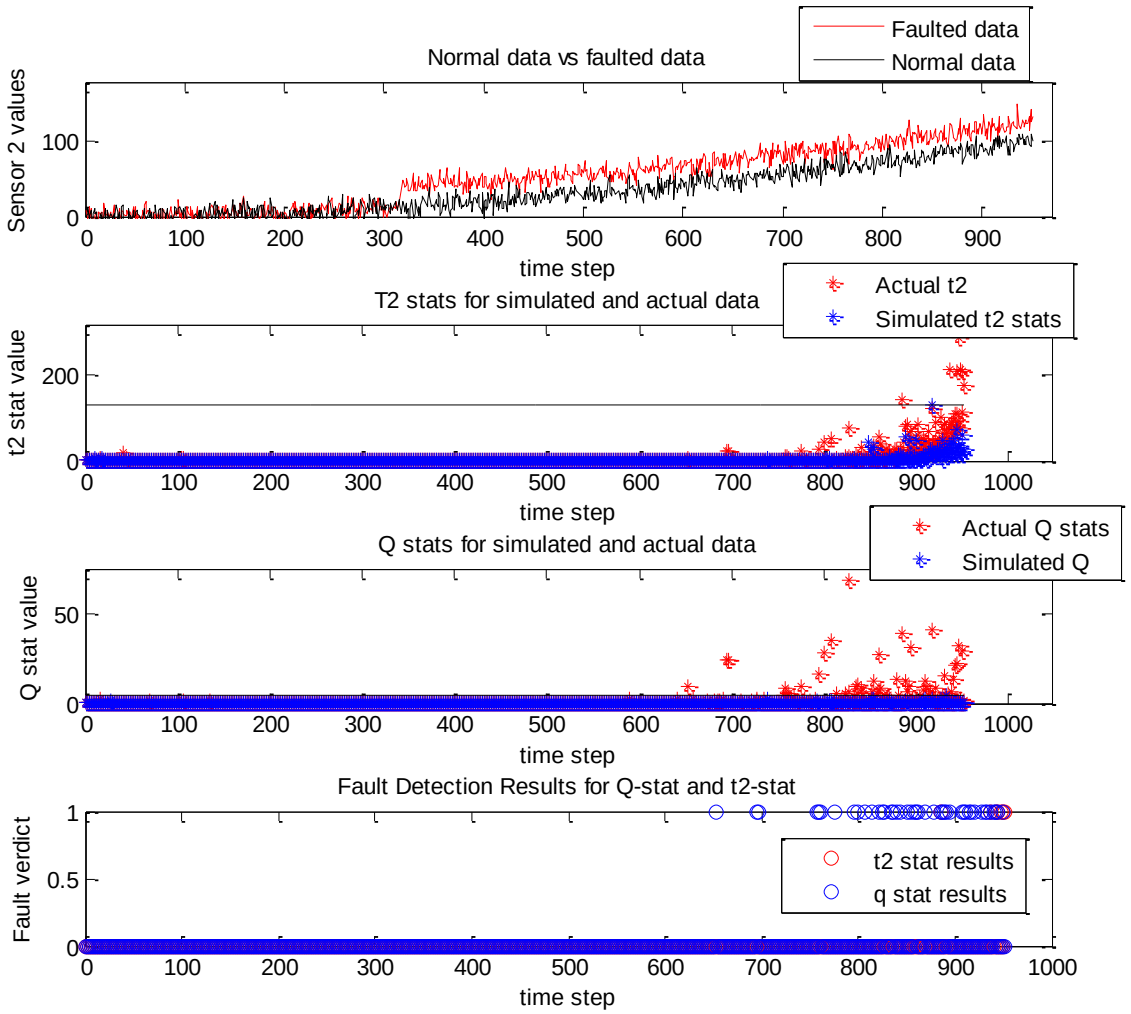
So 2 PCs are used.

Applying this we get:

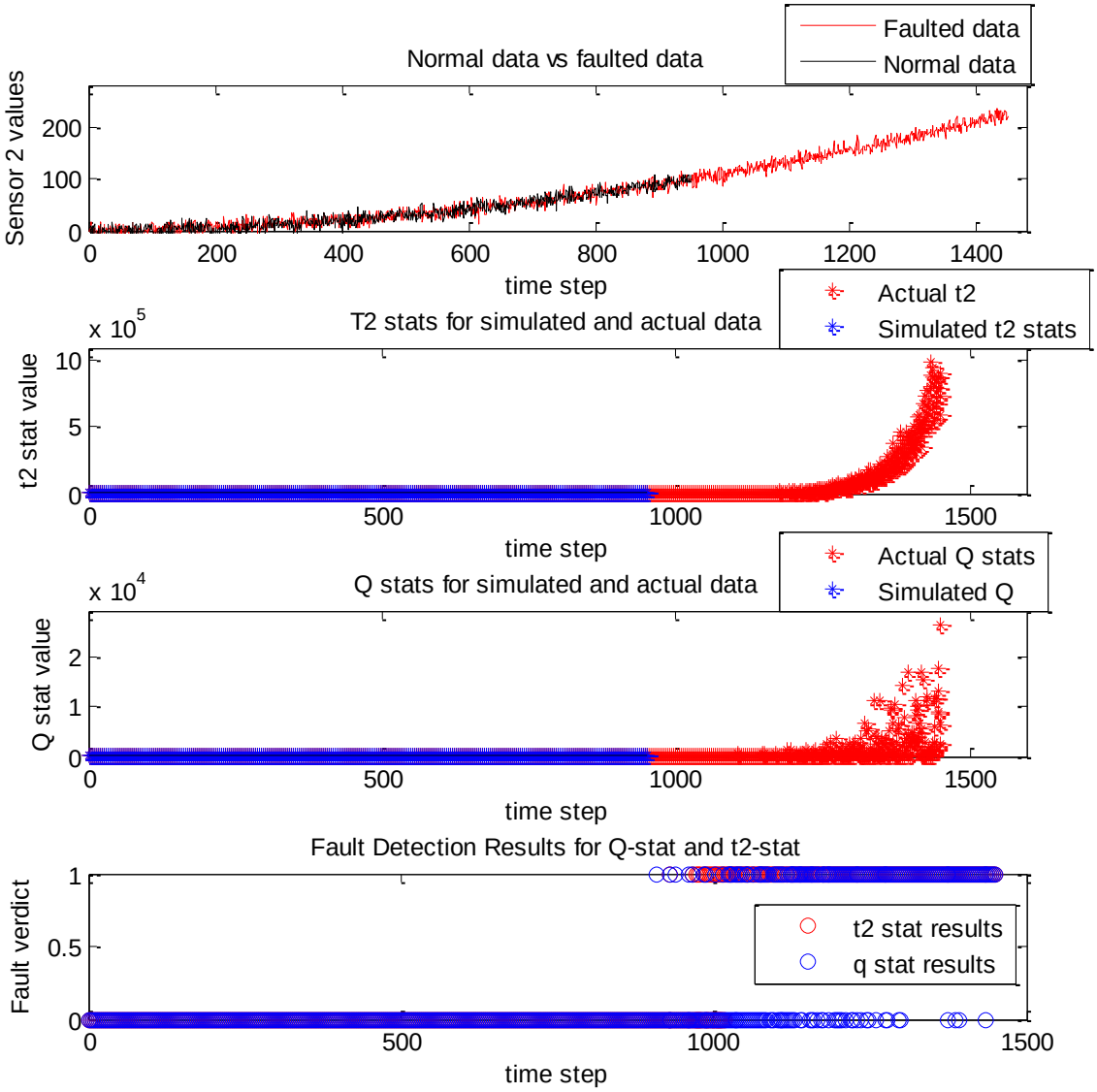
NonFaulted data:



20% Bias:


Normal data vs faulted data
Faulted data
Normal data
Sensor 2 values
time step
T2 stats for simulated and actual data
Actual t2
Simulated t2 stats
t2 stat value
time step
Q stats for simulated and actual data
Actual Q stats
Simulated Q
Q stat value
time step
Fault Detection Results for Q-stat and t2-stat
t2 stat results
q stat results
Fault verdict
time step

Non-Faulted Expanded Condition :



So next trying the polynomial kernel with d=3.

explained =

76.1427

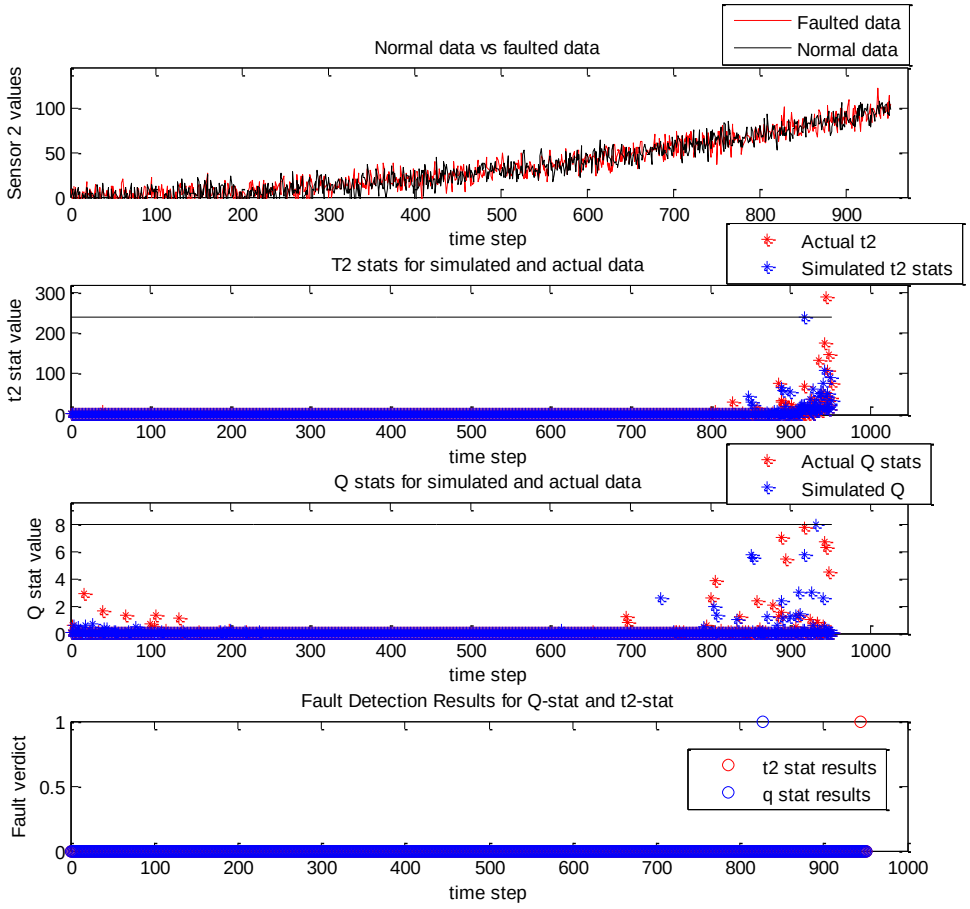
21.6085

2.2488

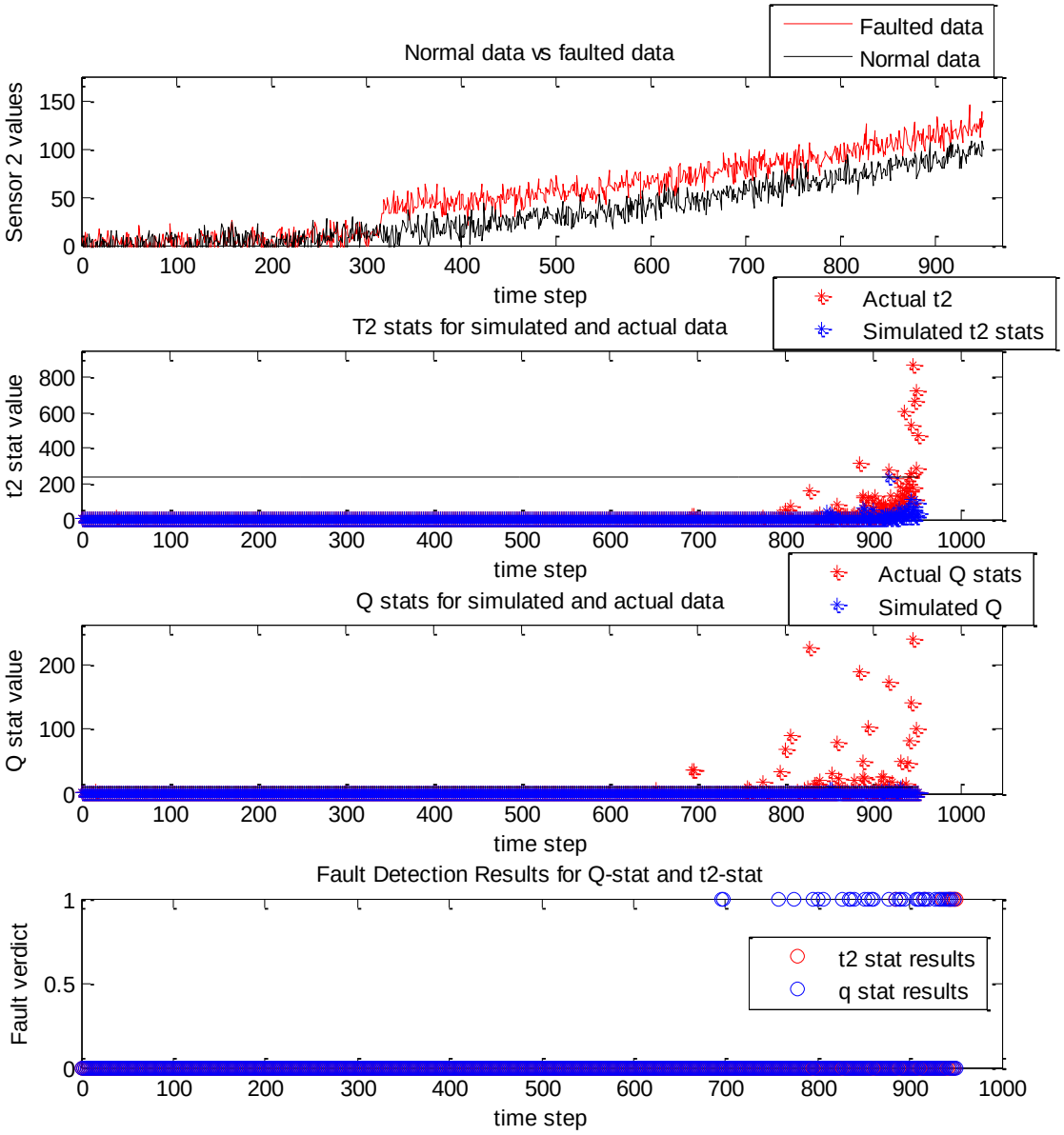
So 2 PCs are used.

Applying this we get:

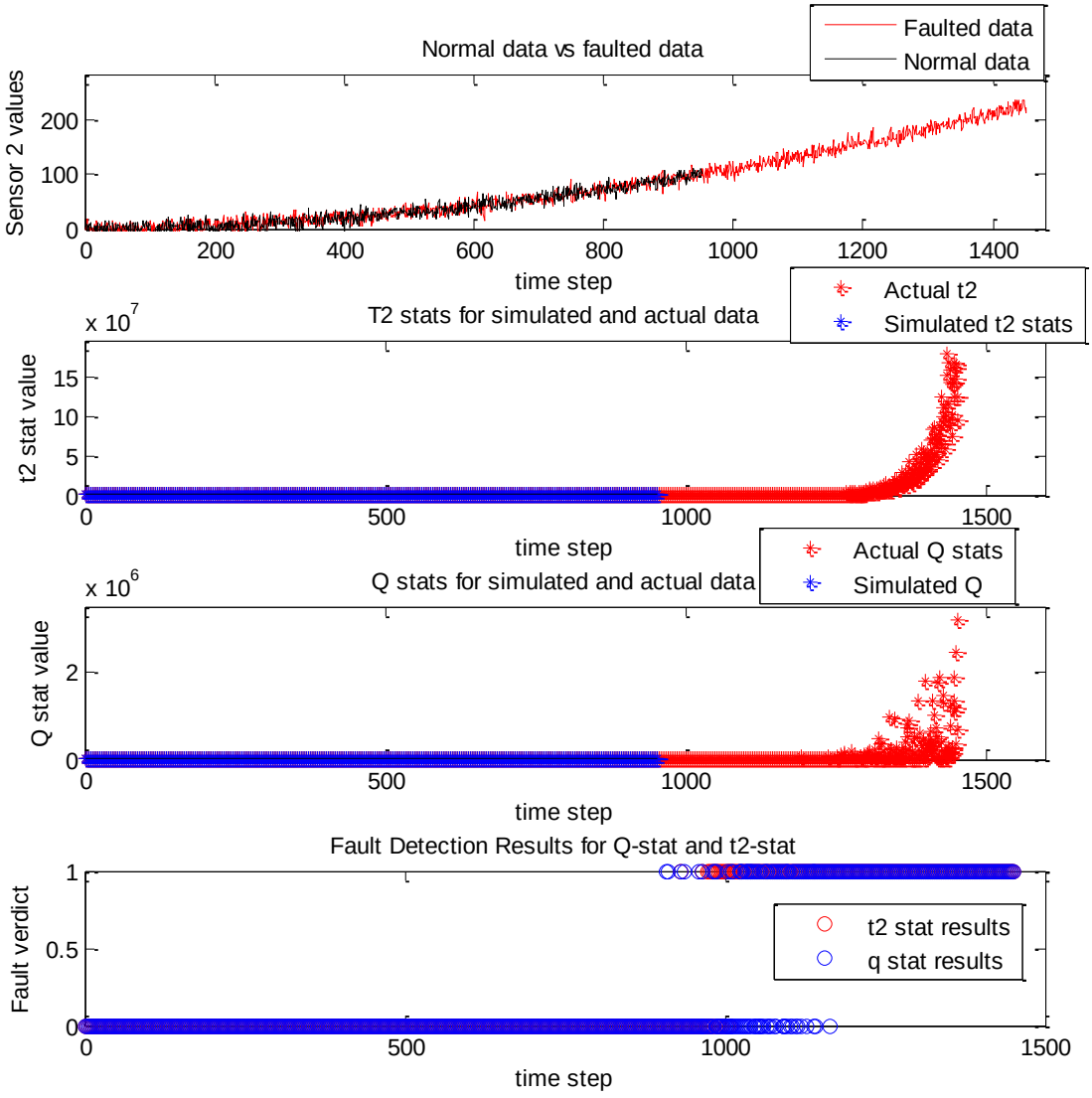
NonFaulted data:



20% Bias:


Normal data vs faulted data
Faulted data
Normal data
Sensor 2 values
time step
T2 stats for simulated and actual data
Actual t2
Simulated t2 stats
t2 stat value
time step
Q stats for simulated and actual data
Actual Q stats
Simulated Q
Q stat value
time step
Fault Detection Results for Q-stat and t2-stat
t2 stat results
q stat results
Fault verdict
time step

Non-Faulted Expanded Condition :



Trying the radial basis kernel: $k(x_i, x_j) = \exp\left(-\frac{\left(x_i - x_j\right)^2}{c}\right)$

We get the following results for c=4:

explained =

62.0419

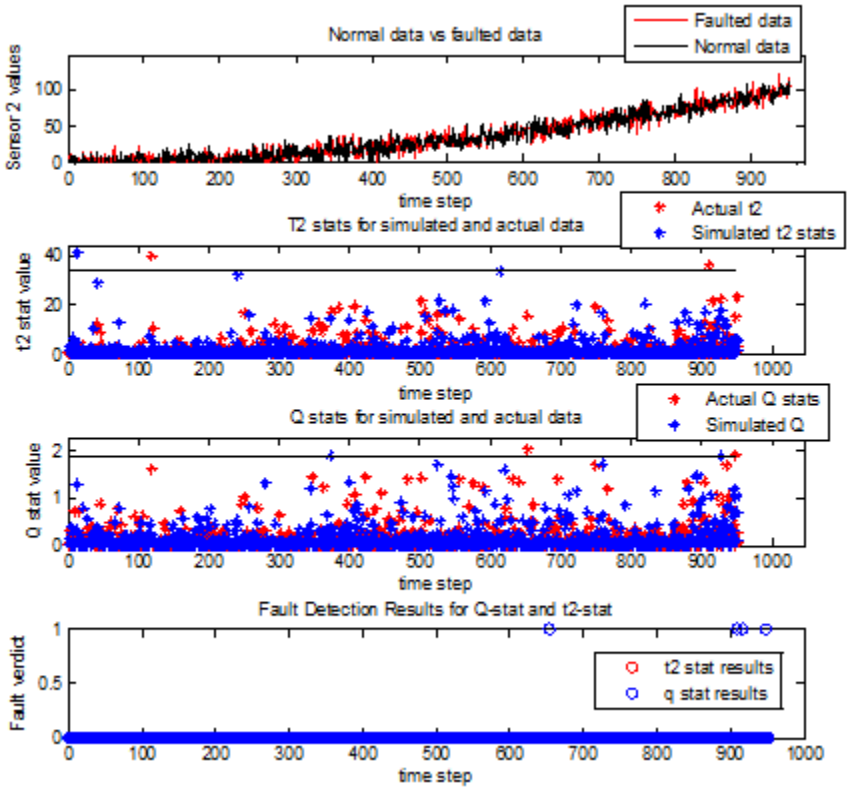
33.2514

4.7067

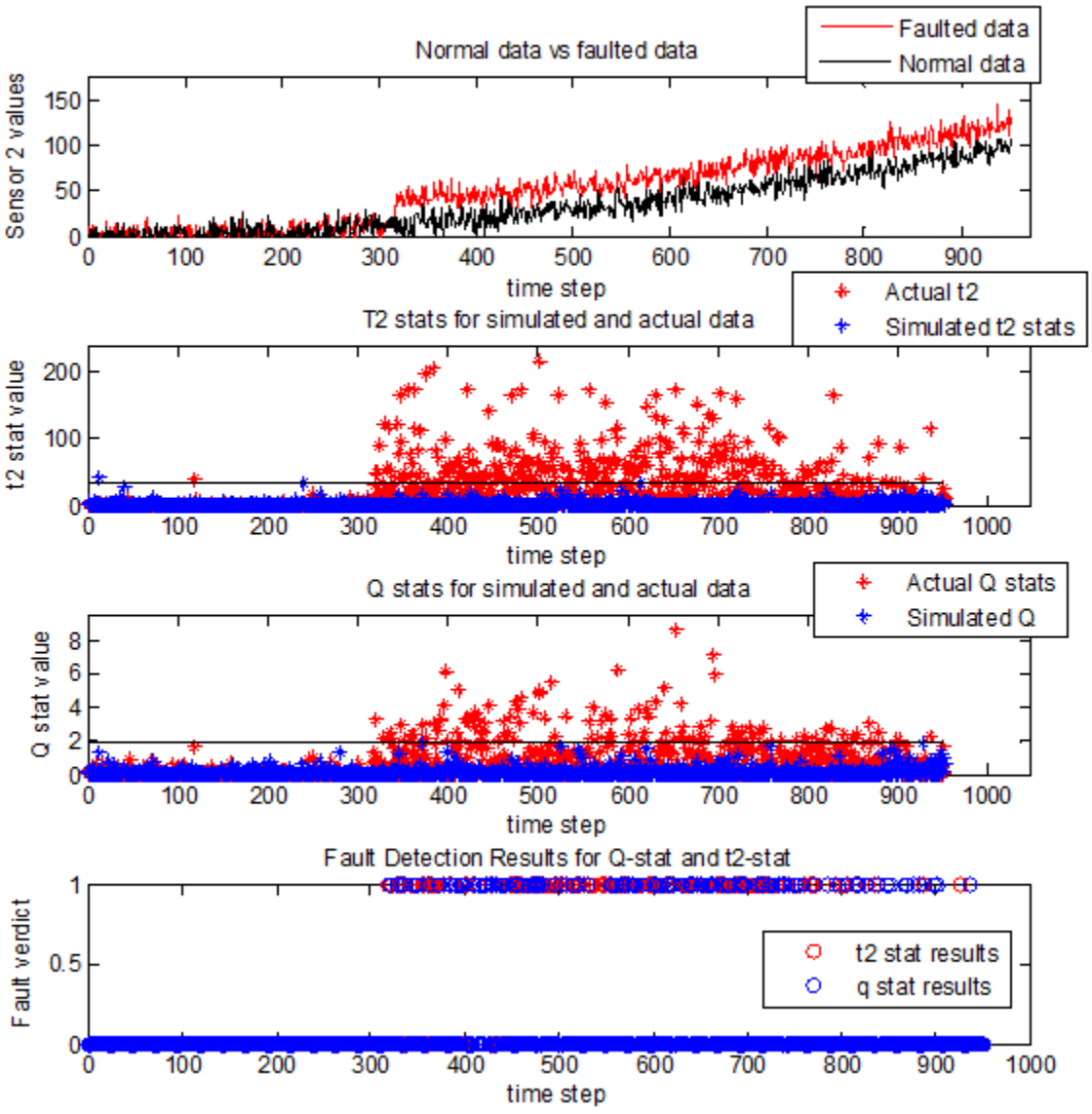
So 2 PCs are used.

Applying this we get:

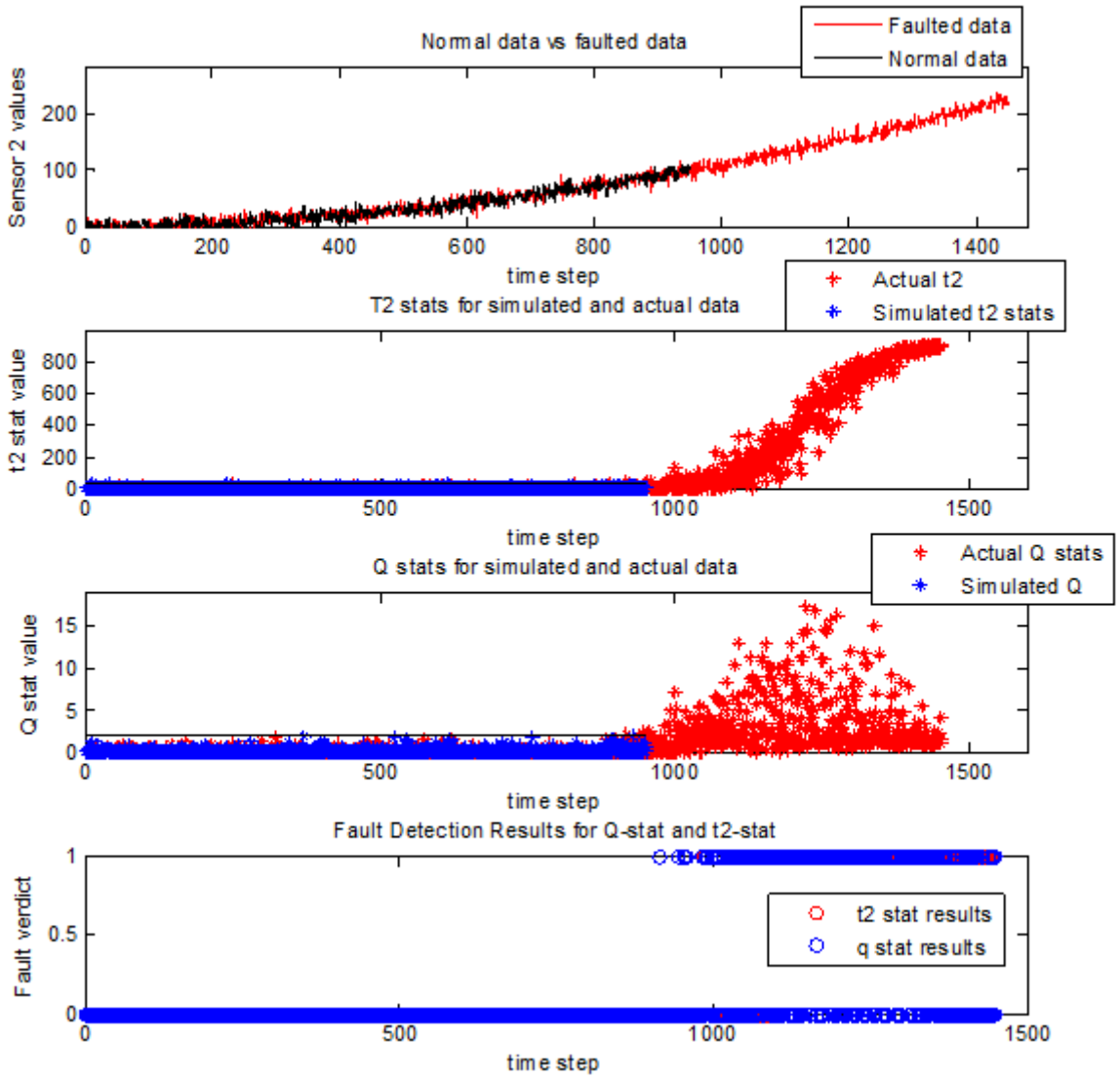
NonFaulted data:



20% Bias:


Normal data vs faulted data
Faulted data
Normal data
Sensor 2 values
time step
T2 stats for simulated and actual data
Actual t2
Simulated t2 stats
t2 stat value
time step
Q stats for simulated and actual data
Actual Q stats
Simulated Q
Q stat value
time step
Fault Detection Results for Q-stat and t2-stat
t2 stat results
q stat results
Fault verdict
time step

Non-Faulted Expanded Condition:



Trying the radial basis kernel with c=2:

explained =

61.7046

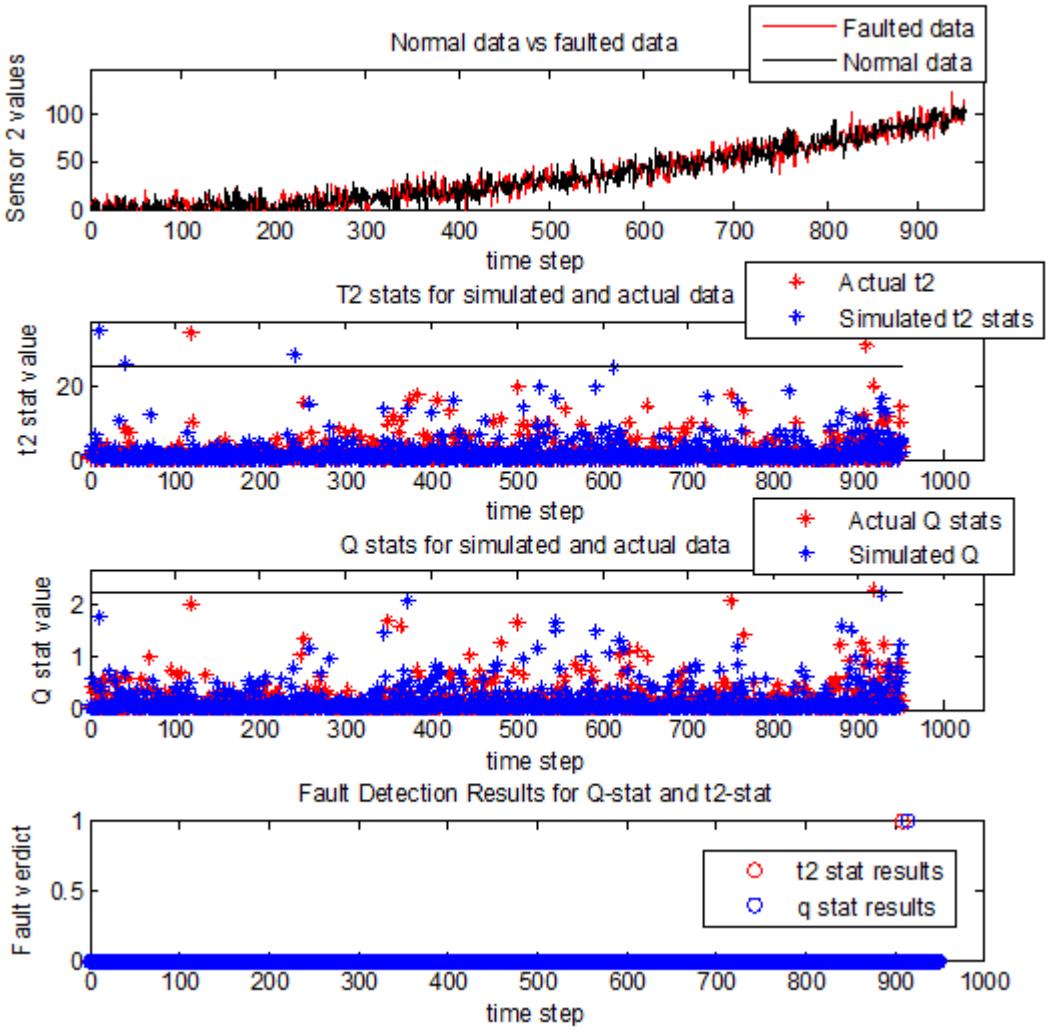
33.2774

5.0181

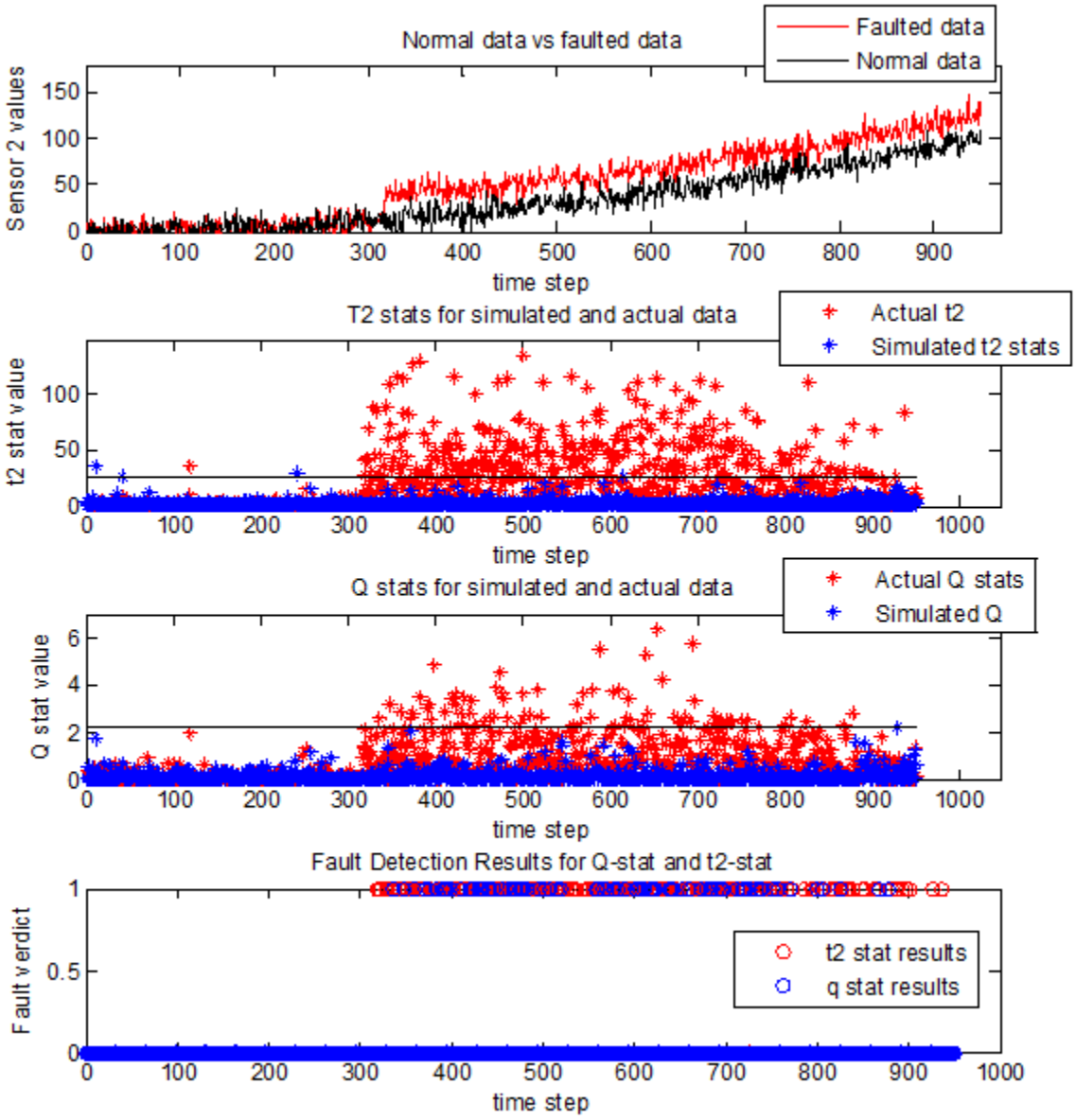
So 2 PCs are used.

Applying this we get:

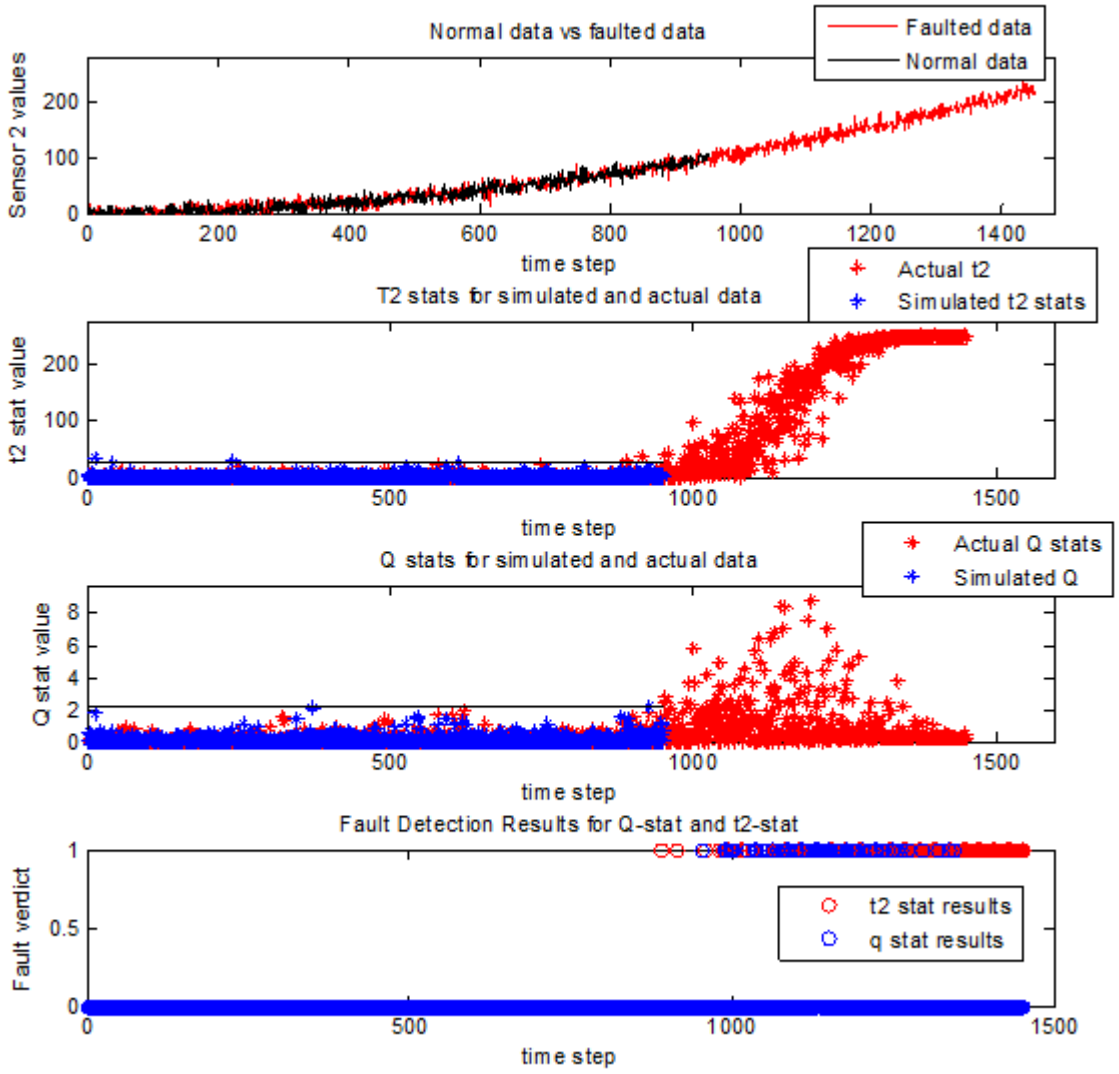
NonFaulted data:



20% Bias:


Normal data vs faulted data
Faulted data
Normal data
Sensor 2 values
time step
T2 stats for simulated and actual data
Actual t2
Simulated t2 stats
t2 stat value
time step
Q stats for simulated and actual data
Actual Q stats
Simulated Q
Q stat value
time step
Fault Detection Results for Q-stat and t2-stat
t2 stat results
q stat results
Fault verdict
time step

Non-Faulted Expanded Condition:



Trying the radial basis kernel with c=1:

explained =

60.8676

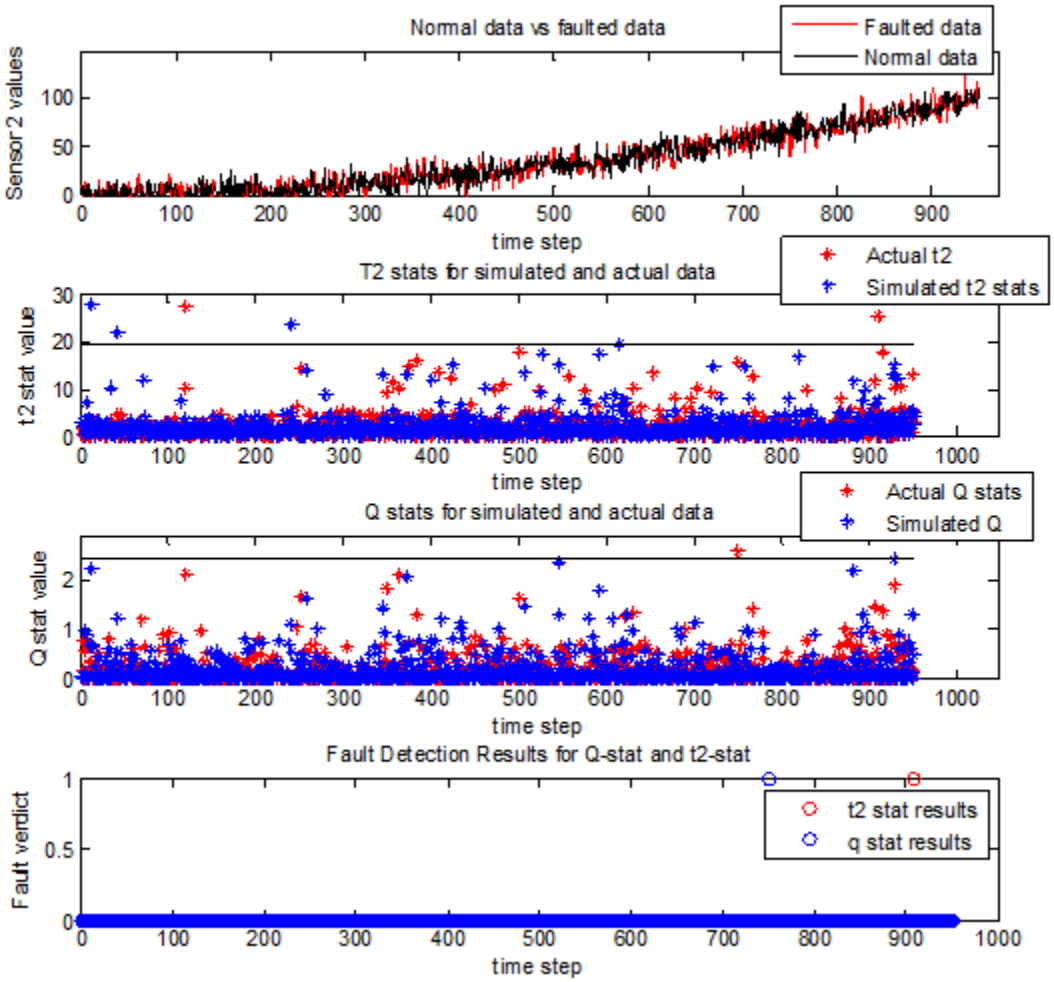
33.3075

5.8250

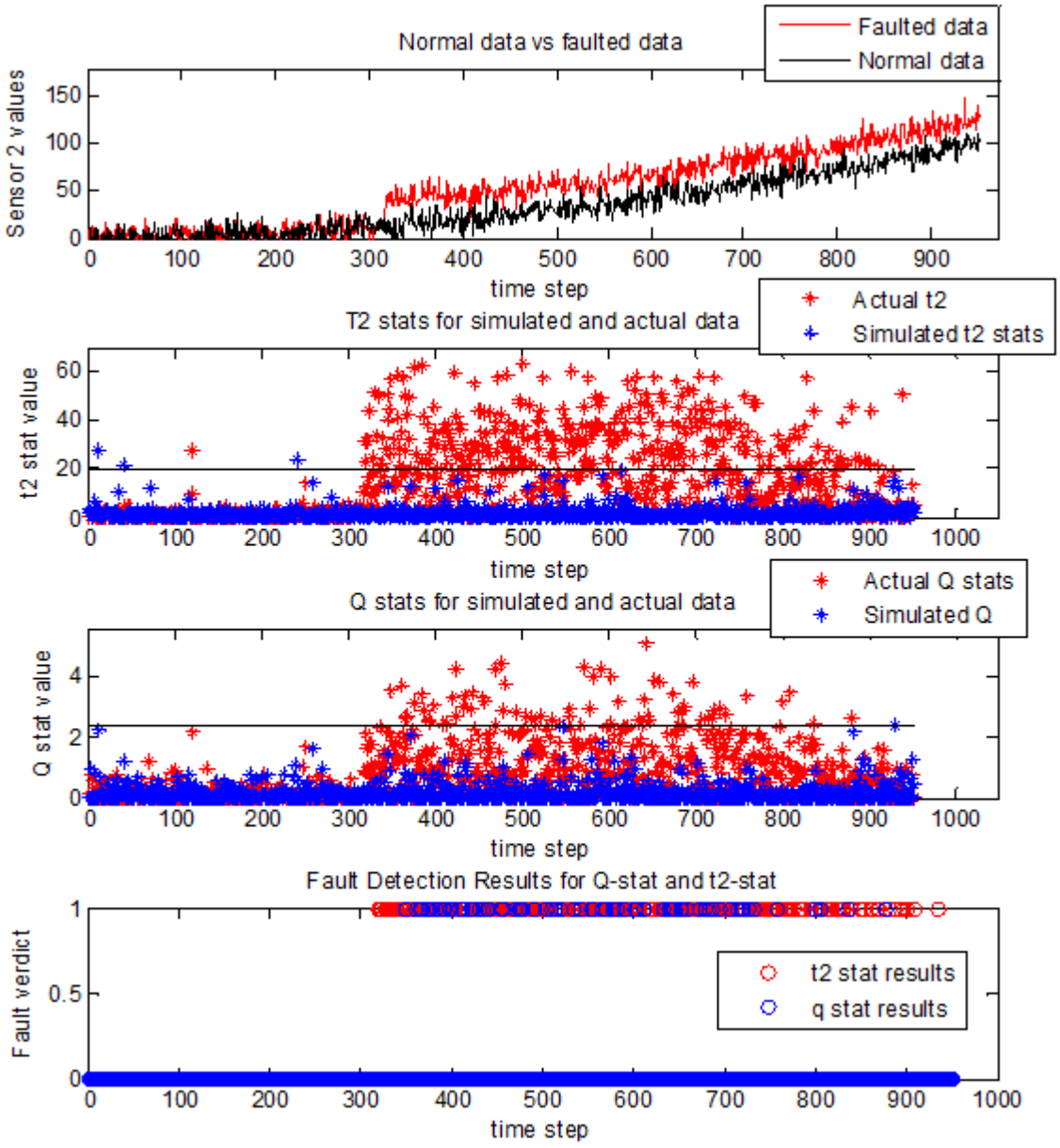
So 2 PCs are used.

Applying this we get:

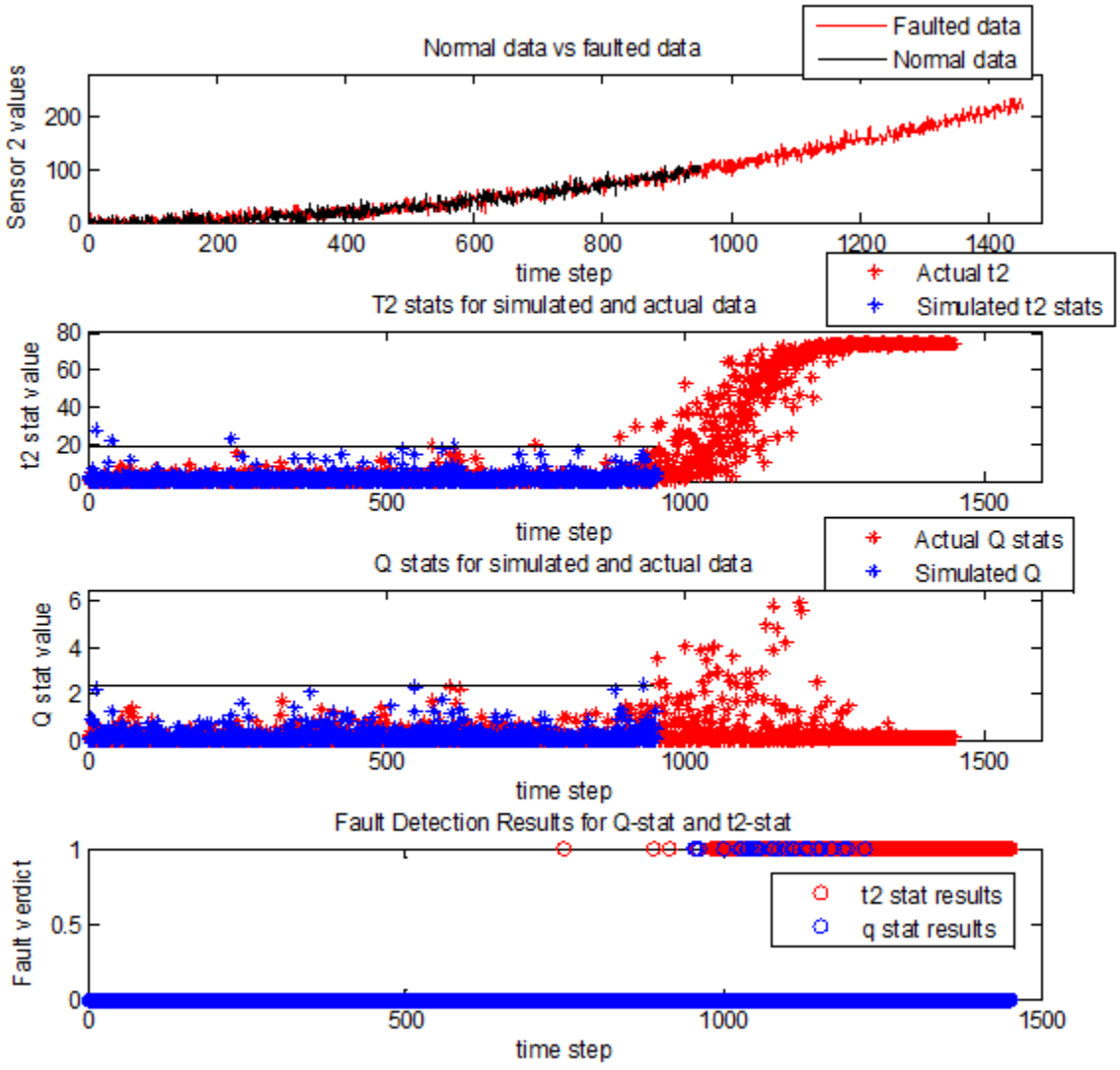
NonFaulted data:



20% Bias:


Normal data vs faulted data
Faulted data
Normal data
Sensor 2 values
time step
T2 stats for simulated and actual data
Actual t2
Simulated t2 stats
t2 stat value
time step
Q stats for simulated and actual data
Actual Q stats
Simulated Q
Q stat value
time step
Fault Detection Results for Q-stat and t2-stat
t2 stat results
q stat results
Fault verdict
time step

Non-Faulted Expanded Condition



VITA

Matthew Humberstone was born on November 3, 1983 in Albuquerque, NM. He attended Moriarty High School in Moriarty, NM where he graduated in May of 2002. Matt studied Engineering Physics at New Mexico State University in Las Cruces, NM. He graduated May 2006 with a Bachelors of Science in Engineering Physics and a minor in Mathematics. He continued on to attend the University of Tennessee in Knoxville, TN for graduate studies in Nuclear Engineering. He graduated with a Masters of Science in Nuclear Engineering in Dec 2007.

He participated in a National Student Exchange program for his junior year where he attended California State University at Northridge. He also was a student intern at Sandia National Laboratory (SNL) for four consecutive summers. While at Sandia he worked for the Advanced Nuclear Concepts division and worked on a number of interesting projects such as reactor impact analysis using Monte Carlo N-Particle (MCNP) and space reactor studies for the now cancelled Jupiter Icy Moon Orbiter (JIMO) project. Under Dr. Miller at the University of Tennessee, Matt studied the isotopic composition of spent fuels as a function of burnup focusing on fast reactors particularly the PRISM reactor. This research was used as the focus of his Master's research.

Matt's current research focuses on an adaptive non-parametric model and its application on the IRIS reactor. He will graduate with a Ph. D. in Nuclear Engineering and a M.S. in Statistics in May 2010. He will then move to North Bethesda, MD where he accepted a position at the Nuclear Regulatory Commission (NRC) in their advanced reactor program.