5-2010

Merging Data Sources to Predict Remaining Useful Life – An Automated Method to Identify Prognostic Parameters

Jamie Baalis Coble
jcoble1@utk.edu

To the Graduate Council:

I am submitting herewith a dissertation written by Jamie Baalis Coble entitled "Merging Data Sources to Predict Remaining Useful Life – An Automated Method to Identify Prognostic Parameters." 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:

J. Wesley Hines, Belle Upadhyaya, Haitao Liao, Xueping Li

Accepted for the Council:
Dixie L. Thompson

Vice Provost and Dean of the Graduate School

(Original signatures are on file with official student records.)

To the Graduate Council:

I am submitting herewith a dissertation written by Jamie Baalis Coble entitled "Merging Data Sources to Predict Remaining Useful Life – An Automated Method to Identify Prognostic Parameters." 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 Upadhyaya

Haitao Liao

Xueping Li

Accepted for the Council:

Carolyn R. Hodges

Vice Provost and Dean of the Graduate School

(Original signatures are on file with official student records.)

Merging Data Sources to Predict Remaining Useful Life – An Automated Method to Identify Prognostic Parameters

A Dissertation Presented for

The Doctor of Philosophy Degree

The University of Tennessee, Knoxville

Jamie Baalis Coble

May, 2010

Dedication

To Lilly, my favorite distraction.

“Cats are dangerous companions for writers because cat watching is a near-perfect method of writing avoidance” – Dan Greenburg

Acknowledgements

I would like to thank the members of my dissertation committee for taking time to review this work: Dr. J. Wesley Hines, Dr. Belle Upadhyaya, Dr. Haitao Liao, and Dr. Xueping Li. I would like to especially thank Dr. Hines for his dedication and guidance performing research throughout the last five years at the University of Tennessee. I'd like to also thank the other faculty members of the Nuclear Engineering department who instructed and helped me during my time at UT: Dr. H.L. Dodds, Dr. M.H. Fontana, Dr. L. Miller, Dr. L. Townsend, Dr. R. Pevey, Dr. J. Gehin, Dr. M.L. Grossbeck, and Dr. R. Uhrig.

The UTNE support staff has also been instrumental in making my tenure as a graduate student in this department fruitful and smooth. A very big thank you goes to both current and past staff members: Kristin England, Lydia Salmon, Dick Bailey, Gary Graves, Kim Scarbrough, Candy Kitts, Ellen Fisher, and Juvy Melton. They have each provided me with invaluable guidance, support, and encouragement over the years. They have also been more than willing to test any vegan baking experiment I've concocted; for this sacrifice, I thank you.

Of course, gratitude is owed to my family and friends who saw me through my education and my life over the last 27 years: my parents: Don and Pat, my brother and his family: Duncan, Sara, Grace, and Sophie, my grandmother: Dot, and my dearest friends: Billy, Sarah, and Julia. Thank you to Matt Humberstone for forcing me to finish my degree even though I didn't want to; it was time. Finally, a great debt of thanks is owed to Lilly Coble for her constant support and caring over the last eight years.

I would also like to thank Dr. Bob Kronick for giving me a chance to help my local community in such a meaningful way. He allowed me to share, and in some cases steal, his vision for full service schools while I volunteered with his program. I enjoyed every minute I spent at Sarah Moore Greene Magnet Technology Academy; I cherish the people I met there, both children and adults. Two SMG faculty made my work there seamless and fun: Mrs. Janice Clark and Ms. Amy Brace. Knoxville is lucky to have had both of them. Thank you to all of you for showing me how much of a difference can be made with so little.

Abstract

The ultimate goal of most prognostic systems is accurate prediction of the remaining useful life (RUL) of individual systems or components based on their use and performance. This class of prognostic algorithms is termed Degradation-Based, or Type III Prognostics. As equipment degrades, measured parameters of the system tend to change; these sensed measurements, or appropriate transformations thereof, may be used to characterize degradation. Traditionally, individual-based prognostic methods use a measure of degradation to make RUL estimates. Degradation measures may include sensed measurements, such as temperature or vibration level, or inferred measurements, such as model residuals or physics-based model predictions. Often, it is beneficial to combine several measures of degradation into a single parameter. Selection of an appropriate parameter is key for making useful individual-based RUL estimates, but methods to aid in this selection are absent in the literature. This dissertation introduces a set of metrics which characterize the suitability of a prognostic parameter. Parameter features such as trendability, monotonicity, and prognosability can be used to compare candidate prognostic parameters to determine which is most useful for individual-based prognosis. Trendability indicates the degree to which the parameters of a population of systems have the same underlying shape. Monotonicity characterizes the underlying positive or negative trend of the parameter. Finally, prognosability gives a measure of the variance in the critical failure value of a population of systems. By quantifying these features for a given parameter, the metrics can be used with any traditional optimization technique, such as Genetic Algorithms, to identify the optimal parameter for a given system. An appropriate parameter may be used with a General Path Model (GPM) approach to make RUL estimates for specific systems or components. A dynamic Bayesian updating methodology is introduced to incorporate prior information in the GPM methodology. The proposed methods are illustrated with two applications: first, to the simulated turbofan engine data provided in the 2008 Prognostics and Health Management Conference Prognostics Challenge and, second, to data collected in a laboratory milling equipment wear experiment. The automated system was shown to identify appropriate parameters in both situations and facilitate Type III prognostic model development.

Executive Summary

Unforeseen equipment failure is costly, both in terms of equipment repair costs and lost revenue. Discovery of unanticipated pressure vessel head degradation at the Davis-Besse nuclear plant led to a 25-month outage and estimated repair costs exceeding $600 million. In September, 2008, a turbine generator malfunction at the D.C. Cook nuclear plant resulted in a fire which led to eventual manual plant shutdown. Turbine repairs totaled $332 million in addition to lost revenue during the one-year outage. Enterprise server downtime can be even more costly, resulting in a possible loss of $6.4 million *per hour* for brokerage operations or $2.6 million per hour for credit card authorization services. Traditional maintenance strategies fall into one of two categories: preventive and corrective. Preventive, or periodic, maintenance occurs on a time-based schedule, such as replacing car tires after 30,000 miles of use. It is completed every cycle, regardless of need, and is intended to occur frequently enough to preclude any failures from occurring. Clearly, this maintenance strategy results in a significant amount of unnecessary maintenance but reduces the occurrence of failure. Corrective maintenance, on the other hand, occurs only when equipment fails or malfunctions, such as replacing car tires only when they are flat. While this maintenance strategy avoids any unnecessary maintenance by only repairing components or systems which have already failed, it also reduces equipment uptime, resulting in lost revenue. A third maintenance strategy, proactive maintenance, attempts to avoid failure modes by performing preventive maintenance to reduce the probability of the fault or redesigning the system to remove the fault. Maintaining proper alignment in a car to reduce the probability of uneven tire wear is a form of proactive maintenance intended to increase the lifetime of the tires.

Condition-based maintenance provides a more elegant and cost-effective maintenance strategy. Falling somewhere between the two extremes of preventive and corrective maintenance, condition-based maintenance involves monitoring equipment health and performing maintenance actions on an as needed basis. Condition-based maintenance is facilitated by a health monitoring system. Health monitoring systems commonly employ several modules, including but not limited to: system monitoring, fault detection, fault diagnostics, prognostics, and operations and maintenance planning. System monitoring and fault detection modules are used to determine if a component or system is operating in a nominal and expected way. If a fault or anomaly is detected by the monitoring system, the diagnostic

system determines the type, and in some cases, the severity of the fault. The prognostics module uses all the available information—including sensed system measurements, monitoring system residuals, fault detection and diagnostic results—to estimate the Remaining Useful Life (RUL) of the system or component along with associated confidence bounds. With this information in hand, system operation may be adjusted to mitigate the effects of failure or to slow the progression of failure, thereby extending the RUL to a point when a scheduled maintenance activity can occur.

Two MATLAB toolboxes have been developed to aid in health monitoring system development. The Process and Equipment Monitoring (PEM) toolbox uses empirical modeling methods to monitor the performance of complex engineering systems and detect deviations from normal operation, or faults. The Process and Equipment Prognostics (PEP) toolbox uses the results of the PEM toolbox to make RUL estimates after a fault has been detected. Prognostic algorithms can be categorized into three classes by the type of information used in making RUL estimates. Type I, or Reliability-based, models use traditional time-to-failure analysis to estimate the RUL of a system or component. These models characterize the lifetime of an average system operating in historically average conditions; they do not consider any specific information from the current system. Type II, or stressor-based, models incorporate operating condition information in RUL estimation. These models estimate the lifetime of an average component or system operating in a specific environment, under specific load and usage conditions. The final type of prognostics, Type III, or degradation-based, models, uses some measure of system degradation to estimate the RUL of that specific system operating in its specific environment. The PEP toolbox supports algorithms in each of the three model types. Type I models include reliability analysis with Weibull, exponential, and Gaussian distributions. Type II models include Markov Chain models, Shock models, and Proportional Hazards models. Type III models include General Path models with Bayesian updating techniques. The PEP toolbox enables the rapid development and comparison of competing prognostic models, as well as full life cycle prognostics. As a system moves through life, different prognostic algorithms may be appropriate for estimating RUL. When the system is newly acquired, Type I models may be the only applicable method. As operations are planned and the system begins to operate, Type II models can be applied to obtain a more accurate estimate. Finally, if it is possible to measure or infer degradation based on sensed measurements of the system, Type III models may be used to give a truly individual-based prognosis.

The ultimate goal of most prognostic systems is accurate prediction of the RUL of individual systems or components based on their specific use and performance. As equipment degrades, measured parameters of the system tend to change; these sensed measurements, or appropriate transformations thereof, may be used to characterize the system degradation. Traditionally, Type III prognostic methods use some measure of degradation to make RUL estimates. Degradation measures may include sensed measurements, such as temperature or vibration level, or inferred measurements, such as model residuals or physics-based model predictions. Often, it is beneficial to combine several measures of degradation into a single parameter to provide a more robust prognostic model. Selection of an appropriate parameter is key for making useful individual-based RUL estimates, but methods and guidelines to aid in this selection have not been developed. Typically, identification of a prognostic parameter is left to expert analysis, visual inspection of available data, and knowledge of the degradation mechanisms. This approach is tedious and costly, and scales with the number of available data sources and possible fault modes.

This dissertation introduces a set of metrics which characterize the suitability of a prognostic parameter. Parameter features such as trendability, monotonicity, and prognosability can be used to compare candidate prognostic parameters to determine which is most useful for individual-based prognosis. Trendability indicates the degree to which the parameters of a population of systems have the same underlying shape. Monotonicity characterizes the underlying positive or negative trend of the parameter. Finally, prognosability gives a measure of the variance in the critical failure value of a population of systems. By quantifying these features for a given parameter, the metrics can be used with any traditional optimization technique, such as Genetic Algorithms, to identify the optimal parameter for a given system. An appropriate parameter may be used with a General Path Model (GPM) to make RUL estimates for specific systems or components. A dynamic Bayesian updating methodology is introduced to incorporate prior information in the GPM regression parameters, thereby capitalizing on all available information. Incorporating prior knowledge into the regression is particularly useful when only a few observations of the degradation parameter are available or the available observations are contaminated with high noise levels. The proposed methods are illustrated with two applications: first, to the simulated turbofan engine data proved in the 2008 Prognostics and Health Management Conference Prognostics Challenge and, second, to data collected in a laboratory milling equipment wear experiment. The automated system was shown to identify appropriate parameters in both situations and facilitate Type III prognostic model development.

Table of Contents

List of Figures

List of Tables

Abbreviations and Symbols

BIST	Built-In Self Test
CBM	Condition Based Maintenance
CBM+	Condition Based Maintenance Plus
CPU	Central Processing Unit
FMEA	Failure Modes and Effects Analysis
GPM	General Path Model
JSF	Joint Strike Fighter
LCM	Life Consumption Model
LEAP	Life Extension Analysis and Prognostics
MAPE	Mean Absolute Percent Error
MRL	Mean Residual Life
MTTF	Mean Time to Failure
PACE	Path Classification and Estimation
PCA	Principle Component Analysis
PDF	Probability Density Function
PEP	Process and Equipment Prognostics Toolbox
PHM	Prognostics and Health Management
PH	Proportional Hazards
PI	Prediction Interval
POF	Probability of Failure
ROI	Return on Investment
RUL	Remaining Useful Life
SPRT	Sequential Probability Ratio Test
ToF	Time of Failure
WPP	Weighted Path Prognostics

1 Introduction

Unexpected equipment failure can lead to large costs, both for maintenance activities and loss of revenue. Discovery of unanticipated pressure vessel head degradation at the Davis-Besse nuclear plant led to a 25-month outage and estimated repair costs exceeding $600 million [1]. In September, 2008, a turbine generator malfunction at the D.C. Cook nuclear plant resulted in a fire which led to eventual manual plant shutdown. Turbine repairs totaled $332 million in addition to lost revenue during the one-year outage. Enterprise server downtime can be even more costly, resulting in a possible loss of $6.4 million per hour for brokerage operations or $2.6 million per hour for credit card authorization services [2]. Obviously, it is of paramount importance to be aware of impending equipment failures so that operations can be adjusted or auxiliary equipment can be employed to avoid these costs when possible.

Traditionally, maintenance activities have taken one of two approaches: preventive and corrective. Preventive maintenance includes routine maintenance activities scheduled on a time basis designed to prevent failure from occurring. While these activities may minimize operating cost, they typically involve the highest maintenance costs. Conversely, corrective maintenance involves performing maintenance only when a failure occurs. While this eliminates unnecessary maintenance, it ensures that every piece of equipment in the system will fail. Between these two extremes lies condition-based maintenance, wherein maintenance actions are performed as needed based on the condition of the equipment. Ideally in this strategy, maintenance is performed after a fault occurs but before failure to reduce any unnecessary maintenance or unplanned downtime. The chart in Figure 1, adapted from [3], shows the relative costs associated with each maintenance strategy. The lowest total costs tend to occur with a condition-based maintenance strategy. However, in order to fully realize the benefit of such a strategy, it is important to accurately trend the effect of a fault on system performance through prognostics.

Prognostics is a term given to equipment life prediction techniques and may be thought of as the "holy grail" of condition based maintenance. Prognostics can play an important role in increasing safety, reducing downtime, and improving the corporate bottom line by facilitating operations planning

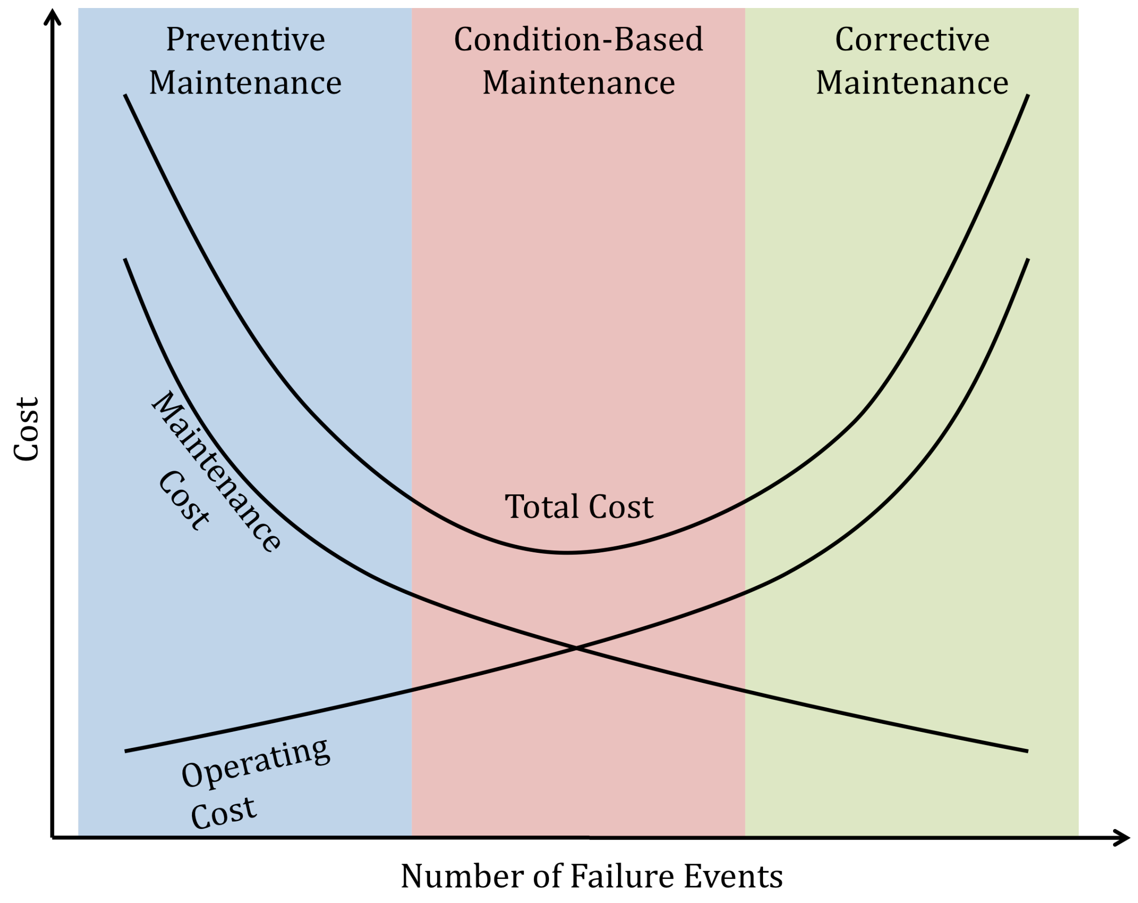


Figure 1: Operating and Maintenance Cost Chart [3]

and timely maintenance. Prognostic systems commonly use several modules which monitor a system's performance, detect changes, identify the root cause of the change, and then predict the remaining useful life (RUL) or probability of failure (POF). Prognostic algorithms can be categorized into three classes based on the type of information used to make RUL estimates. Type I, or reliability-based, prognostics utilize historical failure time data; type II, or stressor-based, prognostics consider operating and environmental conditions; and type III, or degradation-based, prognostics incorporate the measured or inferred condition of the actual system when estimating the remaining system runtime.

Type III prognostics characterize the lifetime of a specific component or system operating in its specific environment. As such, these are the only truly individual-based prognostic models. Individual RUL estimation is the ultimate goal for systems which are expensive, safety-critical, or costly to repair.

Individual-based prognostic methods use a measure of degradation to make estimates of RUL. Degradation measures may be sensed measurements, such as temperature or vibration level, or inferred measurements, such as model residuals or physics-based model predictions using other sensed measurements. Often, it is beneficial to combine several measures of degradation to develop a single parameter. Selection of an appropriate parameter is key for making useful RUL estimates; however, this area of research has been conspicuously lacking in published literature. Typically, identification of a prognostic parameter is left to expert analysis, visual inspection of available data, and knowledge of the degradation mechanisms. This approach is tedious and costly, and scales with the number of available data sources and possible fault modes. Parameter features such as trendability, monotonicity, and prognosability can be used to compare candidate prognostic parameters. Trendability measures how well each parameter in a population is described by the same underlying function. Monotonicity indicates the degree to which a population of parameters is always increasing or decreasing. System damage is generally assumed to be cumulative; because the prognostic parameter tracks degradation as a function of duty cycles, a change in slope would indicate system self-healing. Finally, prognosability characterizes how well defined the critical failure threshold is. Ideally, a prognostic parameter should cross a large range and fail at a well-defined value to allow for extrapolation. With this formalized set of metrics to characterize the goodness of each candidate parameter, traditional optimization methods can be used to automate the identification of prognostic parameters, such as gradient descent methods, genetic algorithms, and machine learning techniques. This automation reduces active model development time and ensures that an optimal or near-optimal prognostic parameter can be found.

Prognostic system development has been a daunting task for several reasons. One is that high value systems are rarely allowed to run to failure once degradation has been detected. This makes the existence of fault-to-failure data rare and the development of degradation-based models difficult. However, current individual prognostic techniques necessitate the availability of a population of exemplar degradation paths for each fault mode of interest. In the absence of real-world data, high fidelity physics of failure models may be used to simulate these paths, but these, too, are rare. Second, if components are allowed to degrade to failure and the failure mode is common, it is often designed out of the system through a continuous improvement process. However, some failure modes are unavoidable, such as pump cavitation or heat exchanger fouling in nuclear power plants. Third, very few legacy systems have the instrumentation required for prognostics. Again, accurate physics of failure models may be used for prognostic system development if they are available, and legacy systems may be re-instrumented to allow for future collection of key data sources. Faced with these and other challenges, evaluation of system health has historically focused on more pedestrian methods, such as time to failure analysis.

Traditional reliability analysis, termed Type I prognostics, uses only failure time data to estimate a time to failure distribution. As equipment components become more reliable, few failure times may be available, even with accelerated testing. Although failure time data becomes more sporadic as equipment reliability rises, often other measures are available which may contain some information about equipment degradation. Lu and Meeker [4] developed the General Path Model (GPM) to assess equipment reliability using these degradation measures, or appropriate functions thereof. This method was originally proposed to move reliability analysis from failure time to failure mechanism analysis. It has since been extended to prognostic applications [5-8]. The GPM assumes that there is some underlying parametric model which describes component degradation. The model may be derived from physical models or directly from available historical degradation data. Typically, this model accounts for both population (fixed) effects and individual (random) effects. Most commonly, the fitted model is extrapolated to some known failure threshold to estimate the RUL of a particular component. By combining the historically determined degradation model with the appropriate degradation measures from the current component or system, the GPM can be used to make truly individual-based prognostic estimates.

A key component missing from the published prognostics research is a method to identify the optimal prognostic parameter for such an extrapolation. This work attempts to fill that hole by developing a methodology for automatically identifying prognostic parameters from data. The following dissertation presents the research completed to develop an automated method for fusing multiple data sources for individual-based prognostics.

1.1 Problem Statement

Type III, or degradation-based, prognostic algorithms are the only truly individual-based estimation methods. Making RUL estimates based on the actual condition of a specific component or system is considered the ultimate goal for safety critical, high risk, and high value systems. Because these algorithms involve trending some measure of degradation herein called a prognostic parameter, identification of an appropriate prognostic parameter is paramount to model performance. Despite the importance of this task in developing individual based prognostic models, it has been conspicuously overlooked in the published literature.

Appropriate measures of degradation do not have to be a directly measured parameter. These measures could be obtained through a function of several measured variables that provide a quantitative measure of degradation. It could also be an empirical model prediction of a key system degradation that cannot be measured. For example, pipe wall thickness may be an appropriate degradation parameter but there may not be an unobtrusive method to directly measure it. However, there may be related measurable variables that can be used to predict the wall thickness, such as temperature, pressure, and flow rate. In this case, the degradation measure is not a directly measurable parameter, but a function of several measurable parameters. Several measures of key degradation sources may be combined to provide a robust prognostic parameter for estimating RUL.

Commonly, identification of prognostic parameters is left to expert analysis and engineering judgment. If any first-principle, or physics of failure, information is available about the system, this knowledge can also be used to inform parameter identification. In the absence of any specific engineering knowledge of the system, however, parameters are typically identified through visual inspection of the available data. Both of these methods are time consuming, tedious, and expensive, and the effort necessary for identifying parameters compounds with additional data sources, fault modes, failure mechanisms, and confounding factors such as discrete operating conditions.

Because the effort needed to identify appropriate prognostic parameters from data can quickly make the problem intractable for a manual approach, an automated approach which results in an optimal, or near-optimal, parameter is very attractive. Developing this methodology is the focus of this research. Prognostic parameter suitability metrics are defined to capture the monotonicity, prognosability, and trendability of a population of prognostic parameters. These metrics can be used with any traditional optimization routine to identify an appropriate prognostic parameter; Genetic Algorithm optimization is used here. A feature selection method is developed and applied to reduce the computational load of the optimization routine when many sources of data are available. Finally, the identified parameter is applied to a GPM prognostic model with Bayesian updating to give prognostic estimates.

1.2 Original Contributions

The research described herein culminates in several original contributions to the field of empirical-based prognostic methods. These contributions lie mainly in the area of identifying prognostic parameters from many data sources for use in individual-based prognostics. Additionally, a platform was developed to aid in the rapid prototyping of prognostic models for many different situations. The main foci of these contributions are enumerated here.

1. Development of a set of metrics to characterize the suitability of a population of parameters for application to the GPM prognostic method, namely monotonicity, prognosability, and trendability.

2. Development of an automated routine to use the proposed suitability metrics to identify an optimal or near-optimal prognostic parameter, including an input selection method to remove un-useful data sources in order to alleviate the computational burden of the optimization routine.

3. Development of the MATLAB-based Process and Equipment Prognostics (PEP) toolbox to facilitate application of the three types of prognostics to general prognostic model development. Methods to aid in, and to some extent automate, prognostic model development are also included in the PEP toolbox, such as the prognostic parameter identification routine.

1.3 Organization of the Document

The next chapter reviews the relevant literature in prognostics algorithms and applications. The structure of the PEP toolbox and its integration with the previously developed Process and Equipment Monitoring (PEM) toolbox is given. Additional information about the PEP toolbox can be found in Appendices C and D. The next chapters outline the specific problem addressed by this work and the methodology proposed to solve this problem. Finally, the results of application of this methodology to two publicly available data sets are given. The first application involves the simulated turbofan engine data used in the PHM '08 Prognostics Challenge Problem; the second application uses milling machine wear data collected in a laboratory setting. Concluding remarks, as well as proposed areas of future work which do not fall under the scope of this dissertation, are also given.

2 Literature Survey

Compared to the more established areas of condition monitoring, fault detection, and diagnostics, prognostics is a fairly immature field. As such, scholarly publications concerning prognostics have focused on the need for prognostics [1, 9-12], the challenges in prognostic model development [1], the many and varied applications of prognostics [8, 13-32], etc. In fact, in many cases, papers purported to study prognostics consider only monitoring, fault detection, and diagnostics while pointing to prognostics as an area of future work or simply overlooking the estimation of RUL as a key function of prognostics [11, 24, 33-37]. Only in recent years have specific prognostic methodologies and applications been presented. These papers will be discussed in more detail in subsequent sections.

One cause for the confusion in developing and applying prognostic methodologies is the lack of a unified definition of Prognostics. For instance, the Joint Strike Fighter (JSF) research group includes fault detection and isolation, enhanced diagnostics, material condition assessment, performance monitoring, and estimation of remaining useful life under the umbrella of prognostics [9]. However, this collection of activities seems better suited to the common moniker of Prognostics and Health Management (PHM) or the program suggested by the U.S. Deputy Under Secretary of Defense for Logistics and Material Readiness, Condition Based Maintenance plus (CBM+) [10]. Sheppard, Kaufman, and Wilmering [38] review the relevant standards in IEEE and other organizations related to prognostics and highlight the importance of predicting RUL as a key feature of prognostics; however, they propose no single, unifying standard for prognostics. For this research, a more specific definition of prognostics will be used, namely estimation of RUL and associated uncertainty. Because fault progression is not deterministic, it is impossible to estimate the RUL exactly. Any RUL estimate given by a prognostic algorithm should be accompanied by an estimate of the uncertainty, giving a confidence interval during which failure is expected to occur or a POF distribution. Prognostics applications in the literature are often remiss in this aspect of the analysis; however, the importance of uncertainty estimation is widely discussed [9].

Because prognostics is defined as an estimate of the RUL of a system, the time of failure must also be explicitly defined. Two types of failure are commonly considered: hard failure and soft failure

[39]. Hard failure is generally considered to have occurred when a product stops working, e.g. an incandescent light bulb burns out or a car tire pops. In hard failure, failure time does not usually occur at a particular degradation level. Instead, loss of functionality occurs at widely varying levels across units. These failures are generally modeled as random failure. Conversely, soft failures are said to have occurred when the degradation level of a system reaches some predefined critical failure threshold, e.g. light output from fluorescent light bulbs decreases below a certain level or car tire tread is below some specified depth. These failures generally do not concur with complete loss of functionality, as in hard failure; however, they correspond with the time when an operator is no longer confident that equipment will continue to work to its specifications. Soft failures, or non-catastrophic failures, are the failures of interest in this research. System degradation can be monitored and trended to make estimates of RUL due to soft failures, whereas hard failures are random in nature and usually can only be predicted if some failure precursor can be monitored.

Finally, the exact nature of RUL should be considered. For a failed unit, it is easy to determine the failure time. It is traditional to consider the RUL at any time before failure to be simply the time between the current time and the failure time (Figure 2). Some argue that this simplistic approach is inappropriate. However, Klinger [40] shows that the assumption that failure time is inversely proportional to the degradation rate is valid for systems which are autonomous in the variable representing time. That is, the differential equations describing failure for these systems are independent of any explicit time dependence. The author further argues that this assumption is generally valid. The inverse relationship should be accepted for systems running under a constant load and environment. In some systems, the operating conditions can be altered to mitigate degradation; in this case the system is not autonomous in time and the inverse relationship is not valid. This more complicated RUL relationship can be seen in Figure 3 [41] where a large shock reduces the RUL significantly. Had this shock not occurred, the circuit board under test could be expected to continue to perform for the originally estimated lifetime.

A significant portion of the published literature in prognostics research focuses on solutions to specific problems, such as electronic prognostics [13-16], vibration analysis [18-20], helicopter gearbox monitoring [8, 17, 21-23, 32], JSF applications [24-31], etc. While this approach may result in very good

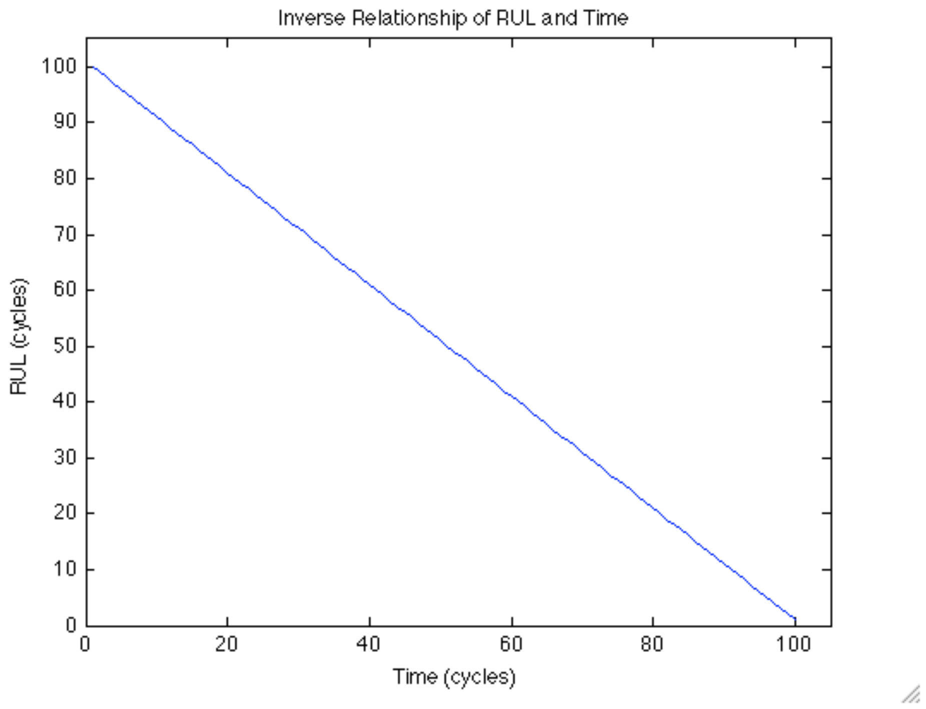


Figure 2: RUL vs Time for time-autonomous systems

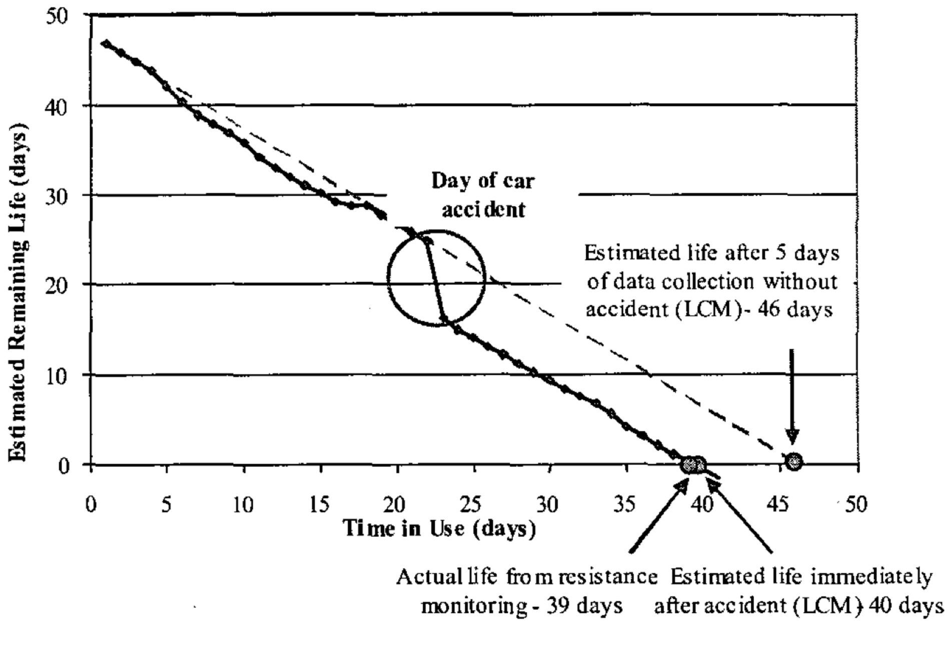


Figure 3: RUL vs Time for a non-time-autonomous system [41]

point solutions for these specific problems, generic prognostic algorithms which may be more broadly applicable are of more interest. The goal in developing generic prognostic algorithms is to develop methods which may be rapidly configured for a new system to allow for effective and efficient deployment of CBM technology on large, complex systems [42].

The following sections discuss specific areas of the prognostics work reported in the literature. First, prognostics as a component of a larger health monitoring system is discussed, such as that suggested by the JSF research group. The next section discusses prognostic algorithms, classifying them into three types based on the information used to make prognostic estimates. Then, metrics for evaluating prognostic model performance are outlined. Finally, data fusion methods for combining information for making prognostic estimates are covered.

2.1 Prognostics in Health Monitoring

Prognostics is one component in a larger health monitoring system which also includes system monitoring, fault detection, and diagnostic modules. Full health monitoring systems, also called Condition Based Maintenance (CBM) systems, are the focus of much research. Kothamasu, et al. [43] describe prognostics as part of a full CBM system. The authors describe using prognostic estimates to aid maintenance scheduling and planning; they also suggest prognostics for optimal control algorithms. Pipe [44] and Hess, et al. [9] suggest the use of RUL estimates for maintenance planning and logistics systems. Callan, et al. [45] outline a five-step CBM system which includes: Data Acquisition, Data Manipulation, Condition Monitoring, Health Assessment, and Prognostics. By applying the entire suite of modules, one can accomplish the goals of most prognostic systems: increased productivity; reduced downtime; reduced number and severity of failures, particularly unanticipated failures; optimized operating performance; extended operating periods between maintenance; reduced unnecessary planned maintenance; and reduced life-cycle cost. Figure 4 gives a diagram of a typical health monitoring system. Data collected from a system of interest is monitored for deviations from normal behavior. Monitoring can be accomplished through a variety of methods, including first principle models, empirical models, and statistical analysis [46]. The monitoring module can be considered an error correction routine; the model gives its best estimate of the true value of the system variables. These estimates are compared to the data collected from the system to generate a time-series of residuals. Residuals characterize system deviations from normal behavior and can be used to determine if the system is operating in an abnormal state. A common test for anomalous behavior is the Sequential Probability Ratio Test (SPRT) [47]. This statistical test considers a sequence of residuals and determines if they are more likely from the distribution that represents normal behavior or a faulted distribution, which may have a shifted mean value or altered standard deviation from the nominal distribution. If a fault is detected, it is often important to identify the type of fault; systems will likely degrade in different ways depending on the type of fault and so different prognostic models will be applicable. Expert systems, such as fuzzy rule-based systems, are common fault diagnosers. Finally, a prognostic model is employed to estimate the Remaining Useful Life (RUL) of the system. This model may include information from the original data, the monitoring system residuals, and the results of the fault detection and isolation routines.

Many prognostic algorithms have been proposed; these will be discussed in more detail in following sections. Despite the many differences between these algorithms, they all have several common features [1, 9, 48]. Prognostic systems must naturally build on monitoring and diagnostic systems. It is near impossible to determine the failure time of a device without first detecting and identifying a failure-causing fault.

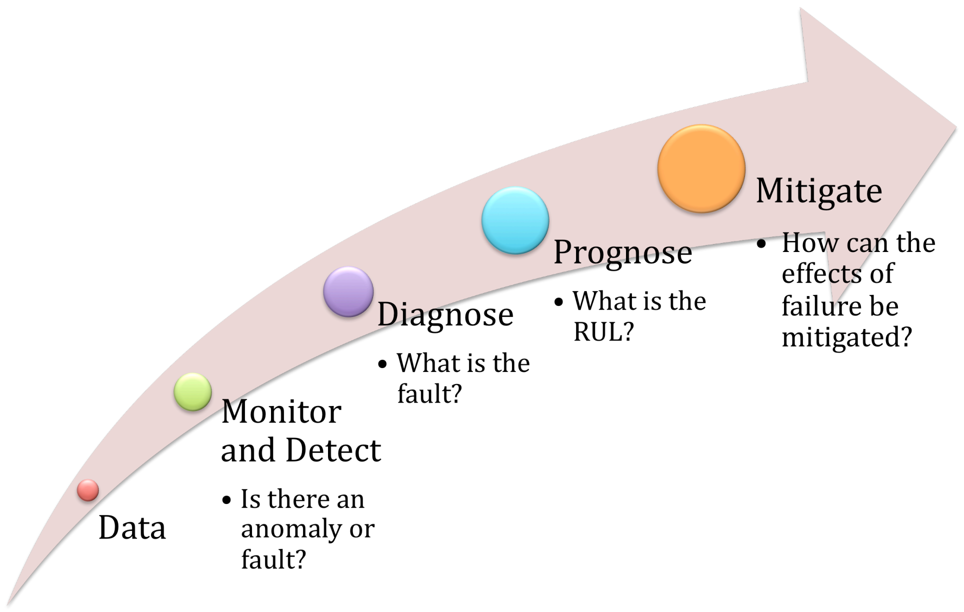


Figure 4: Suite of Modules in a Health Monitoring System

This leads to the first requirement of a prognostic system: notice period [48]. Also called lead time or fault-to-failure time, the notice period is the amount of time between fault detection and predicted failure. It is easy to agree that in most cases a prognostic system which gives only a few seconds of notice before an impending failure is virtually worthless. The notice period must be sufficiently long to allow for operator action, maintenance, automatic reconfiguration, etc.

Beyond this sensitivity, this also means that fault progression must be sufficiently slow for detection, diagnosis, and prognosis to occur with an acceptable notice period. That is, a precipitating fault must occur and become detectable a reasonable time before the resulting failure. This implies an assumption of most prognostic systems: monotonic fault progression. Most prognostics systems assume that once a fault occurs, a failure will also predictably occur in the absence of human intervention; that is, prognostic systems assume that faults cannot self-heal.

A final requirement specific to data-driven prognostics systems is the availability of failure data, specifically several instances of the same type and trend for each failure mode of interest. Failure modes must be carefully identified to differentiate between different fault types and fault progression rates. Similar failure modes which progress at different rates should often be considered different failures. Because of individual, random differences and noisy measurements, it is important to collect many historical fault progressions for each failure mode of interest in order to build a robust and complete prognostic system.

Several procedures have been outlined for developing prognostic systems [49-52]. The specific methodologies vary in the breakdown of steps necessary, but they all generally include the same 3 steps, often subdivided into a total of five to seven steps:

1. Identify important failure modes. This may be accomplished through a failure modes and effects analysis (FMEA) which identifies possible failure modes for a system and ranks them according to severity of failure and risk of occurrence.
2. Identify key parameters that need to be monitored in order to detect and identify these failure modes. This step is sometimes governed by the funds available for the

monitoring sensor suite, the effects sensors may have on performance, and the availability of accurate sensors.

3. Identify and develop appropriate methods to monitor the key failure modes. This may include a variety of approaches, including canary devices or a prognostic algorithm. Typically, one prognostic model should be developed for each anticipated failure mode; significant accuracy may be lost by lumping multiple fault modes into one model.

Prognostic algorithms can be classified in many ways. The following section suggests a classification based on the type of information used to make prognostic estimates: reliability-based, stressor-based, or condition-based. Prognostic algorithms, like all modeling methods, can also be classified as physics-based or empirical. The following sections touch on both types of models; however, the PEP toolbox and the current research focus on empirical prognostics methods. Physics-based models, also called physics of failure models or first principle models, can be expensive and tedious to develop. Often times, the underlying physical processes leading to failure are not completely understood, and simplifying assumptions must be made to facilitate model development. Assumptions made in model development may not be fully applicable to real world systems, which limits the applicability of physics of failure models. In addition, these models are often very computationally expensive, particularly if Monte Carlo simulations are used to estimate confidence intervals about model predictions. Empirical models, on the other hand, use data to fit a model to the relationships seen in real world operation. These models typically provide no additional information about the physical mechanisms leading to failure. Empirical models are preferable to physics of failure models because they are simple to develop, they capture real world relationships, and they require no expertise in the underlying physical phenomena leading to failure. Empirical models have several drawbacks, though. These models rely on operational data for model development. As such, the models are generally only applicable to systems operating within the range of the training data used in model development. This poses additional problems for prognostics models, which rely on run-to-failure data for model training. Very few expensive or safety critical systems are allowed to run to failure; in this case, physics of failure models may be used to simulate failure data for model development. Often, failure data which is available may be the result of accelerated life testing performed during the product design phase.

Several methods are available for extracting useful data which describes actual operation from accelerated testing data [53, 54]. Lemoine and Wenocur [55] give a comprehensive overview of failure modeling. Tang and Chang [56] use the Eyring model to explain degradation measure dependence on accelerated factors. Park and Padgett [57] describe a method for extracting information from accelerated degradation testing with multiple degradation factors. By applying these methods, it is possible to utilize accelerated life test data collected during product design and development phases for prognostics applications. However, care must be given to ensure that the failures seen during accelerated testing are analogous to real-world failures. Accelerated testing conditions can result in fault modes which only occur under the accelerated conditions. The following sections discuss a variety of prognostic models, both physics-based and empirical, classified according to the type of data used to make prognostic estimates.

2.2 Prognostic Algorithms

As suggested by the "No Free Lunch" Theorem, no one prognostic algorithm is ideal for every situation [58, 59]. A variety of models have been developed for application to specific situations or specific classes of systems. The efficacy of these algorithms for a new process depends on the type and quality of data available, the assumptions inherent in the algorithm, and the assumptions which can validly be made about the system. As such, these prognostic algorithms can be categorized according to many criteria. One proposed categorization focuses on the type of information used to make prognostic estimates; this results in three classes of prognostic algorithms (Figure 5) [1, 60]. Type I prognostics is traditional time to failure analysis; this type of prognostic algorithm characterizes the expected lifetime of an average system operating in an historically average environment. These methods may be applied if no data specific to the current system is available. Examples of Type I prognostics include Weibull analysis, exponential or normal distribution analysis, and nonparametric distribution analysis. A readily apparent shortcoming of this group of methods is the absence of consideration for operating conditions and environment in making RUL estimates. Typically, systems operating in harsher conditions will fail more quickly while those in milder environments more slowly. Type II, or stressor-based, prognostics use operational and environmental condition data to estimate RUL. This type of prognostics characterizes the lifetime of an average system or component operating in the specific environment.

Type II methods can be used if operating conditions, such as load, input current and voltage, ambient temperature, vibration, etc., are measurable and correlated to system degradation. Algorithms in this class include specific formulations of the Markov Chain model, shock model, and proportional hazards model, and the path classification and estimation (PACE) [61, 62] or weighted path prognostics (WPP) method. Although more specific than Type I models, Type II models are deficient because they neglect unit-to-unit variance which may be due to manufacturing-, installation-, and maintenance action-variability. The final class of algorithms, Type III or condition-based, prognostics, characterize the lifetime of a specific unit or system operating in its specific environment. Extrapolation of a general path model (GPM) is the most common Type III method. GPM extrapolation involves trending a prognostic parameter and extrapolating it to some predefined failure threshold. A prognostic parameter is a measure, either directly sensed from the system or inferred from a set of sensor readings, which characterizes system degradation or health. System failure is commonly indicated by a soft failure threshold at which the system no longer performs to its specifications or cannot be expected to perform for an appreciable amount of time; this is generally some point before a catastrophic failure occurs. A Bayesian updating technique has been developed to incorporate prior knowledge in the model. Type III prognostics are the only truly individual-based prognostic type. This is generally considered the ultimate goal of prognostics for safety-critical components and systems. Each of these classes of prognostic algorithms, as well as appropriate model architectures, is discussed in the following sections.

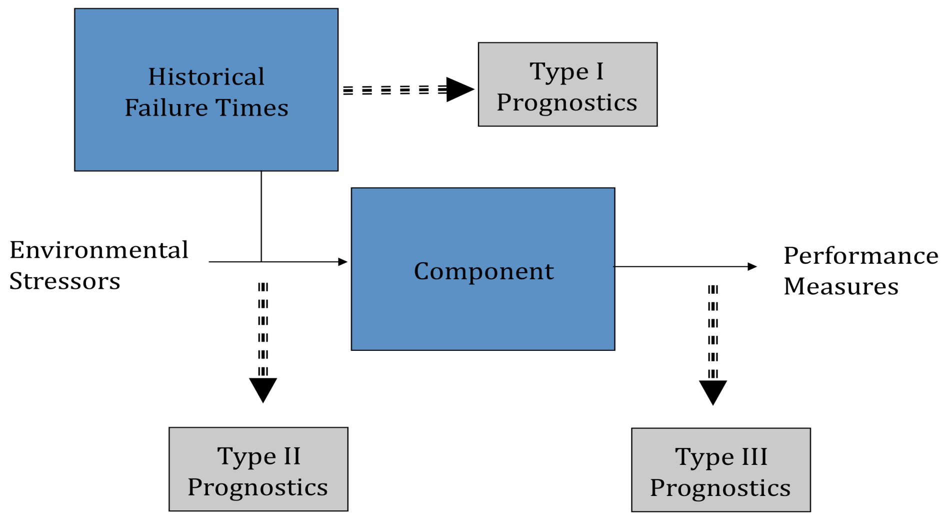


Figure 5: Prognostic Algorithm Categorization

2.2.1 Type I: Traditional Time-to-Failure Analysis

Type I methods are a simple extension of traditional reliability analysis, based entirely on an a priori distribution of failure times for similar systems in the past. Prognostic algorithms in this class characterize the average lifetime of an average system operating in historically average conditions; they do not utilize any information specific to the system at hand. The main assumption made when applying Type I methods is that future systems will operate under similar conditions to those seen in the past and will fail in similar ways.

Typically, Type I prognostic models track a population of systems over their lifetime and record only the failure time of each system. In addition, the total runtime of each system which hasn't failed at the end of the observation is recorded; this is called censored data and is also included in the analysis. A probability distribution is fit to these runtimes to give an estimate of the time of failure (ToF) distribution of the population. The most common parametric model used in reliability analysis is the Weibull distribution. This model is used because it is flexible enough to model a variety of failure rates. The formula for the failure rate (eqn. 1) is a two parameter model with a shape parameter (β) and a characteristic life (θ):

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

These two parameters provide the modeling flexibility for components exhibiting an increasing failure rate (β>1), a constant failure rate (β=1), and a decreasing failure rate (β<1). With the correct choice of shape parameter, the Weibull distribution adequately models the exponential, normal, or Rayleigh distributions. Examples of different shape parameters are given in Figure 6. Additional information on Weibull modeling is available in [63].

Traditional reliability methods consider only the total runtime of a system and the historic total lifetimes of similar systems. However, several methods are available to include additional information in reliability analysis, which may make it more useful for prognostics. Yang and Xue [64] suggest a method for analyzing both catastrophic and soft failures simultaneously using random process simulation and state tree analysis.

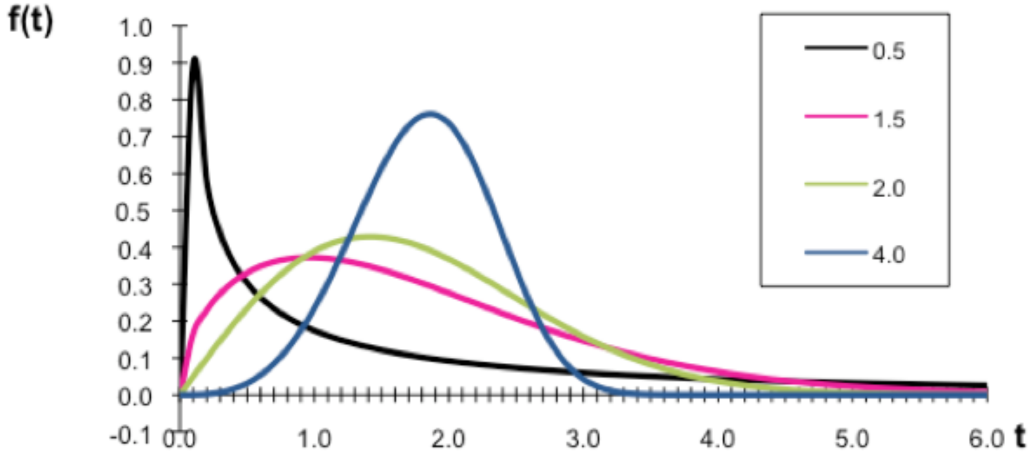


Figure 6: Weibull Failure Distributions with Different Shape Parameters

Several studies suggest the use of degradation data in estimating reliability distributions. Lu and Meeker first proposed the General Path Model (GPM) in [1], which shifts reliability analysis from failure time to failure mode analysis. Improvements to their seminal work are proposed by Girish, Lam, and Jayaram who used neural networks to estimate the failure times for censored systems [65]; Kharoufeh and Cox apply Markovian degradation models to estimate the failure time for censored systems [66]; Chen and Zhang attempt to infer the lifetime distribution itself instead of the distribution parameters from the available data [67]; and Xu and Zhao extend the approach to use multivariate degradation measures [68]. These algorithms are the basis for the Type III prognostic GPM algorithm and are discussed in more detail in that section.

Most commonly, the Mean Residual Life (MRL) is used to estimate the RUL of a system. For an unit of age t, the MRL method assumes that the remaining life is a random variable, and the MRL is given by the expected value of this random variable [69]:

$$MRL(t) = \frac{1}{S(t)}\int_{t}^{\infty} S(u)du$$

where S(·) is the survival function and t is the current time. The MRL at time t can be calculated from either parametric or nonparametric distributions, which makes it particularly flexible for application to real world data.

Studies applying Type I prognostics to estimation of RUL have shown the method to be unsatisfactory [70, 71]. This is to be expected since the MRL method assumes that the remaining life is a random variable; clearly it is not. In fact, because mean residual life is an "average" measure, it is expected to underestimate RUL about half the time and overestimate RUL about half the time. Vichare, et al. [72] show that Type I methods are insufficient for electronic prognostics; the authors go on to suggest that in situ monitoring of operating conditions, such as temperature, humidity, vibration, and shock, may improve prognostic model performance. This leads to Type II prognostic methods.

2.2.2 Type II: Stressor-based Prediction

As suggested by [72-74], it is intuitive to consider usage conditions, both past and future, when estimating the RUL of a system. Type II methods attempt to do this by characterizing the lifetime of an average component operating in the specific environment. Here, it is assumed that systems operating in the same conditions will fail in similar ways; there is little unit-to-unit variance. Methods which commonly fall into this category include regression analysis with prognostic monitors [75-79], a specific formulation of the Markov Chain model [66, 80, 81], Proportional Hazards Models [33, 82, 83], physics-of-failure models [32, 84-87], and Life Consumption models [41,88]. Studies utilizing each of these architectures are described below.

2.2.2.1 Regression Analysis

Electronic system prognostics on the board or circuit level commonly utilize a built-in self test (BIST) prognostic monitor or canary [75-79]. A prognostic monitor is a "pre-calibrated semiconductor cell that is co-located with the actual circuit on a semi-conductor device" [79]. The prognostic cell is designed to experience a higher current level than the actual circuit by decreasing the cross sectional area of the current-carrying path in the canary. Because the canary cell undergoes a higher current density, it is expected to fail in a predictably faster way than the actual circuit. By locating several prognostic monitors on a circuit with different known accelerating factors, the failure times of each of

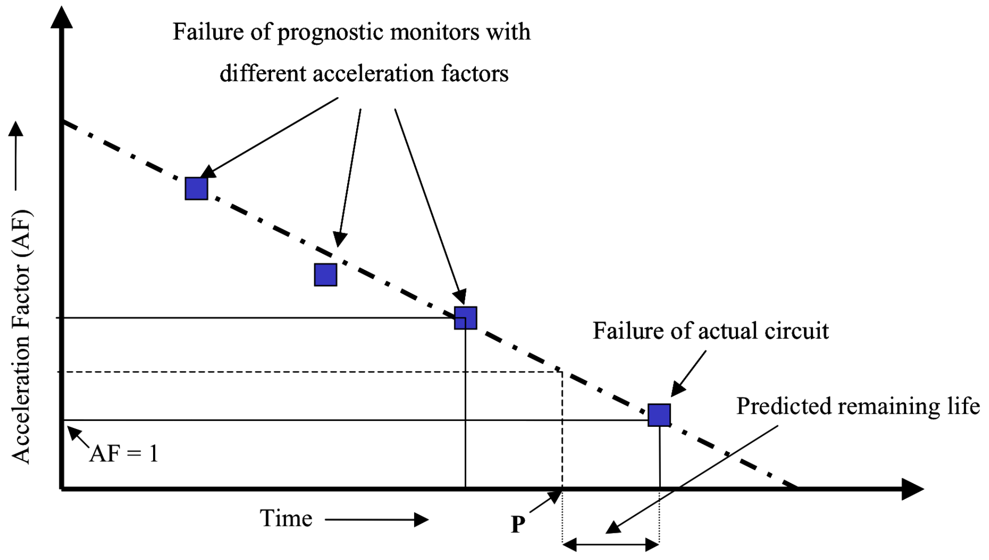

Failure of prognostic monitors with different acceleration factors
Acceleration Factor (AF)
Failure of actual circuit
Predicted remaining life
AF = 1
Time
P

Figure 7: Use of Prognostic Monitor Failure Times to Estimate RUL of Actual Circuit [79]

the cells can be trended to predict failure in the actual circuit (Figure 7). While this method is convenient and uncomplicated, Pecht, et al. [89] argue that BIST monitors are not always sufficient for detecting and identifying failures. The authors found BIST results to suffer from a high false alarm rate and a low correlation between the fault indicated by the BIST and the actual fault. These shortcomings should be considered before applying this type of prognostic monitoring module.

2.2.2.2 Markov Chain Models

Markov Chains and Hidden Markov Chains are common in many simulation exercises [80]. The Markov Chain model is based on the assumption that the next state which a system will occupy depends only on the current state; past states do not affect the probability of transitioning to a new state. There are two types of Markov Chain prognostic models, which vary only in the information they use to simulate possible future states. Type II Markov Chain models, which will be discussed here, really include two models.

The first model, called the environmental model, is a Markov Chain simulation which produces possible future operating state progressions based on transition probabilities seen in the past and the current operating state. The environment model is needed for making a prediction as to how the environment and operating conditions evolve in the future. The environmental model is defined by the transition probability matrix:

$$Q = \begin{bmatrix} p_{11} & \cdots & p_{1n} \\ \vdots & \ddots & \vdots \\ p_{n1} & \cdots & p_{nn} \end{bmatrix}$$

where pij is the probability of transitioning from state *i* to state *j*. Often this probability matrix is assumed to be static, but it is straightforward to extend the method to time-dependent or degradation level-dependent transition probabilities. This model is used to simulate many possible future state progressions beginning at the current state.

These state progressions are then mapped to a degradation measure, which is the second model necessary in the Type II Markov Chain algorithm. The degradation measure is represented as a function of observable environmental conditions. To be useful for making a reliability prediction, the function should reflect the manner in which the environmental conditions affect the component reliability. Usually, environmental stressors tend to deteriorate the component reliability in a cumulative manner. Hence, the function to relate the environment conditions to the prognostic parameter is commonly of a cumulative form:

$$Y(t_k) = \sum_{i=1}^{k} g\big(E(t_i, t_i + \Delta t_i)\big)\Delta t_i$$

where $Y(t_k)$ is the degradation measure value at time t_k, $E(t_i,\ t_i+\Delta t)$ is the environmental condition observed at the time interval $[t_i\ ,\ t_i+\Delta t]$, and *g(.)* is an appropriate function of environmental conditions.

When the estimated degradation measure is found to cross some pre-defined threshold, failure is said to have occurred. At each time of interest, many possible state progressions are simulated and mapped to degradation measures. These measures are then used to define a time of failure (ToF) distribution for the system. Figure 8 shows a typical progression of the environmental conditions in time. An example of trajectories is given in Figure 9 in which the function *g(.)* is assumed to be an identity function.

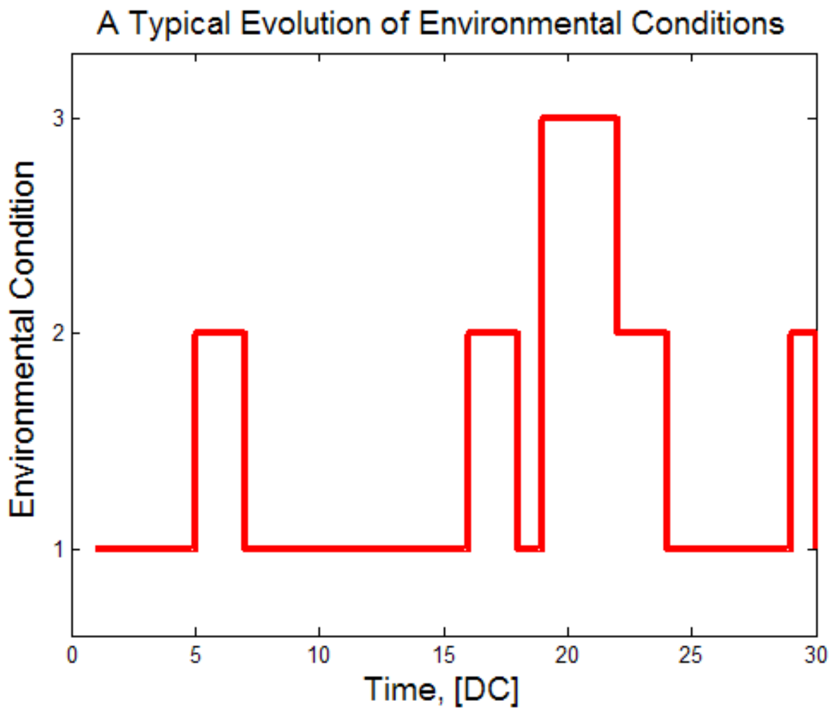


Figure 8: A Typical Progression of the Environmental Conditions

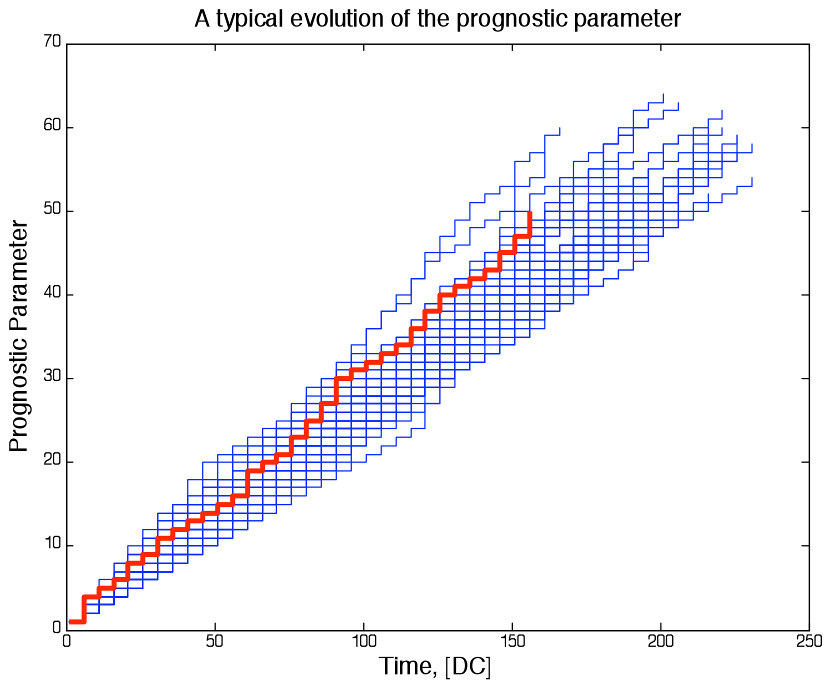


Figure 9: A Collection of Possible Degradation Measures

2.2.2.3 Proportional Hazards Model

The Proportional Hazards (PH) Model developed by Cox [90] merges failure time data and stress data to make RUL estimates. The model uses operating conditions, called covariates, to modify the baseline hazard rate to give a new hazard rate for the system's specific usage conditions:

$$\lambda(t;z) = \lambda_0(t)\exp\left(\sum_{j=1}^{q}\beta_j z_j\right)$$

Failure data collected at covariate operating conditions are used to solve for the parameters (βj) using an ordinary least squares algorithm. The baseline hazard is the hazard rate when covariates have little or no influence on the failure rate. A basic assumption of the proportional hazards model is that the effects of these covariates are multiplicative; this means that when the ratio of two covariates is evaluated, their hazard rates are proportional. A full discussion of developing a proportional hazards model can be found in [83]. Dale [82] applied the proportional hazard model to estimation of product reliability, applying it to heterogeneous data from non-repairable systems. Liao, et al. [33] suggest the use of proportional hazards models for estimating RUL, though the authors give no specific results of such an application.

2.2.2.4 Physics of Failure Models

Physics of failure models, or first principles models, are often desirable in engineering applications because they provide a greater understanding of the mechanisms by which systems and components may fail. Physics of failure models are also desirable for high-cost, high-risk systems that cannot be run to failure many times to collect the data needed for development of empirical models [86]. The use of physics of failure models for estimating RUL has focused mainly on electronic system prognostics [84, 85]. These models may be readily available for single components or single fault modes; however, developing accurate physics of failure models for large, complex systems has always been a daunting task. Physics of failure models often suffer from inaccuracies due to the assumptions made in developing the model, the inclusion of physical interactions which themselves are not completely understood, and long runtimes. Kacprzynski, et al. [87] attempt to alleviate the inaccuracies of physics of failure models by fusing the results with other data sources such as diagnostic results, prognostic architectures, inspection information, and real-time system level features. The authors applied their system to a gear with a seeded fault and found very promising results for the data fusion technique. In [32], the same methodology was applied to helicopter gearboxes with similar success. Physics of failure models offer a better understanding of the mechanisms of failure for a component or system, but they are costly and tedious to develop for large, complex systems which experience many fault modes. In addition, the run time needed for many damage propagation models may make them impractical for real-time analysis. For systems with accurate physics of failure models available, it may be prudent to use these models to simulate system failure data. That simulated data may then be used to develop empirical models which may be run very quickly. This alleviates the burden of collecting failure data for expensive or safety critical systems. Physics of failure models are a key component of Life Consumption Models.

2.2.2.5 Life Consumption Models

Life Consumption models (LCM) were first proposed by Ramakrishnan and Pecht [88] for monitoring RUL in electronic systems. The LCM methodology monitors the environment of a component or system during its entire lifecycle to determine the amount of damage incurred by the various loads and conditions experienced. This damage is translated to lost "life" which is subtracted from the expected life of an average system or component. The incurred damage is estimated through physics of failure models; this damage amount is related to a fraction of life lost by considering the total amount of operation under the same conditions which would result in failure of an undamaged part. LCM is illustrated in [88] and [41] by application to a mounted printed circuit board operated under the hood of a moving vehicle. Both temperature and vibration levels were monitored on the board during use. The methodology was shown to effectively estimate RUL of the circuit board, even in the event of unexpected damage accumulation caused by a large, random shock. The major drawback of the LCM methodology is the need for accurate physics of failure models to estimate the accumulated damage. As mentioned above, physics of failure models are often not available, not accurate, or not time-effective for large, complex systems. Development of a more general LCM methodology which utilizes empirical models, such as neural networks, kernel regression models, or simple regression, for damage estimation would increase the applicability of this algorithm.

2.2.3 Type III: Degradation-based Prediction

Finally, the last category of prognostic algorithms attempts to characterize the lifetime of the specific system operating in its specific environment. These methods are termed degradation-based (or condition-based). The most common method in this class is the General Path model (GPM), which is described in great detail later. Basically, the GPM attempts to track some measure of degradation called a prognostic parameter and extrapolate it to failure [1-8, 11, 24, 91-93]. This measure may be something measured directly from the system that gives information about system condition and fault severity, or it may be inferred from measurements made on the system. The specific details of this method are left to later sections because it is the focus of this work; however, fundamentals and results of this algorithm given in the literature are summarized here. Additional algorithms included in Type III prognostics are a different formulation of Markov Chain models [60, 94, 95] and Shock models [96-98].

2.2.3.1 Parametric Methods

The General Path Model (GPM) was first proposed by Lu and Meeker [1] to move reliability analysis from failure time to failure mode analysis. Extension of their original methodology to prognostics is discussed in detail in later sections. The first work to make this extension was by Upadhyaya, et al. [5]. The authors in that work apply traditional regression models and neural networks to trend system degradation. In later years, the extrapolation methodology of traditional regression models was applied to helicopter gearboxes [8], flight control actuators [92], aircraft power systems [24], computer power supplies [91], and global positioning systems [7]. In addition, work by Chinnam [6] applied the GPM methodology to feed forward neural networks for estimating degradation levels. Each of these studies attempt to model a degradation measure and extrapolate it to some pre-defined threshold to estimate the system RUL. Only a few of the studies consider the problem of uncertainty measurements [8 91, 92]. The studies tend to take two approaches to uncertainty estimation. Uncertainty measurements in [91] and [8] are estimated based on the model architecture used to make the prediction, while Byington, et al. [92] utilize Bayesian belief models to estimate the uncertainty. In addition, bootstrap methods can also be used to make uncertainty estimates. If accurate physics of failure models are available, these may be used to trend the system degradation. In these models, the physics of failure model utilizes measurements directly from the system of interest to estimate the hidden damage [93]. Here, the authors utilize a residual monitoring system to estimate the hidden damage in a system; this basic methodology can be applied to physics of failure models or empirical models. As in the previous discussion, Type III physics of failure models suffer from difficulty in development, inaccuracies due to the assumptions made in model development, and long runtimes. However, if accurate physics of failure models are available, they can be applied to prognostic algorithms and used with the GPM methodology.

Greitzer, et al. [12, 52, 99] propose a slightly different formulation of the GPM, which he coined the Life Extension Analysis and Prognostics (LEAP) method. LEAP differs from GPM primarily in that it is a short-term regression model. Instead of regressing a model onto the entire operating history of a system or component, LEAP utilizes some recent window of data for the regression. Although using more data tends to lead to more stable predictions, using the entire operating history may mask recent changes in behavior which are of critical importance. While these predictions have greater variability, they tend to be more sensitive to recent and abrupt changes in condition, as seen in Figure 10. The LEAP methodology is proposed as a Type II method using operating conditions to estimate some figure of merit and then trending that figure of merit to failure. However, the extension to Type III prognostics is clear, where the figure of merit is directly measured or inferred from direct measurements of the system. This methodology is an improvement on the traditional GPM in that it is more sensitive to abrupt changes, but it is also less robust to noisy measurements. Identification of an optimal window size for regression is a critical task in applying this technique.

As mentioned previously, Markov Chain models can fall into the Type II or Type III category. While Type II Markov Chain models use the Markov assumption to generate possible future operating condition progressions, Type III Markov Chain models use the Markov assumption to generate random shock arrival times based on the current degradation level [60, 94, 95]. These random shocks contribute some deterministic amount of degradation, usually one unit, to the degradation measure. The time of failure is calculated as the time when this simulated degradation measure crosses the failure threshold.

At each time of interest, many degradation paths are simulated, and a probability of failure distribution is estimated from the collection as shown in Figure 11. The Markov Chain model is continuous in the time domain, but discrete in the degradation measure. A more general formulation is

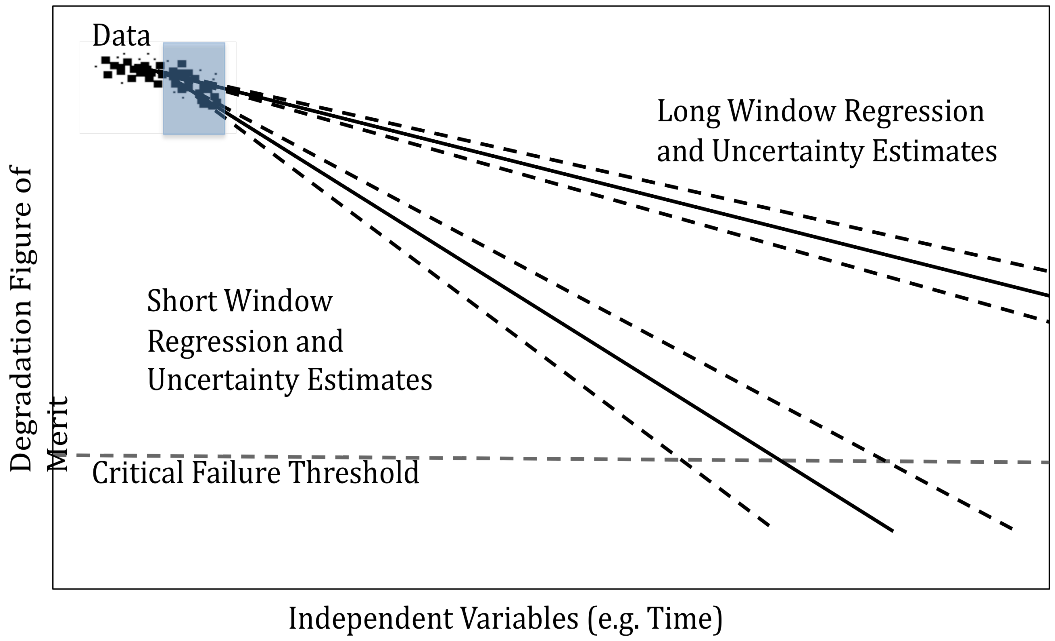

Data
Long Window Regression and Uncertainty Estimates
Short Window Regression and Uncertainty Estimates
Critical Failure Threshold
Degradation Figure of Merit
Independent Variables (e.g. Time)

Figure 10: LEAP Short Term Regression [99] Markov Chain Models

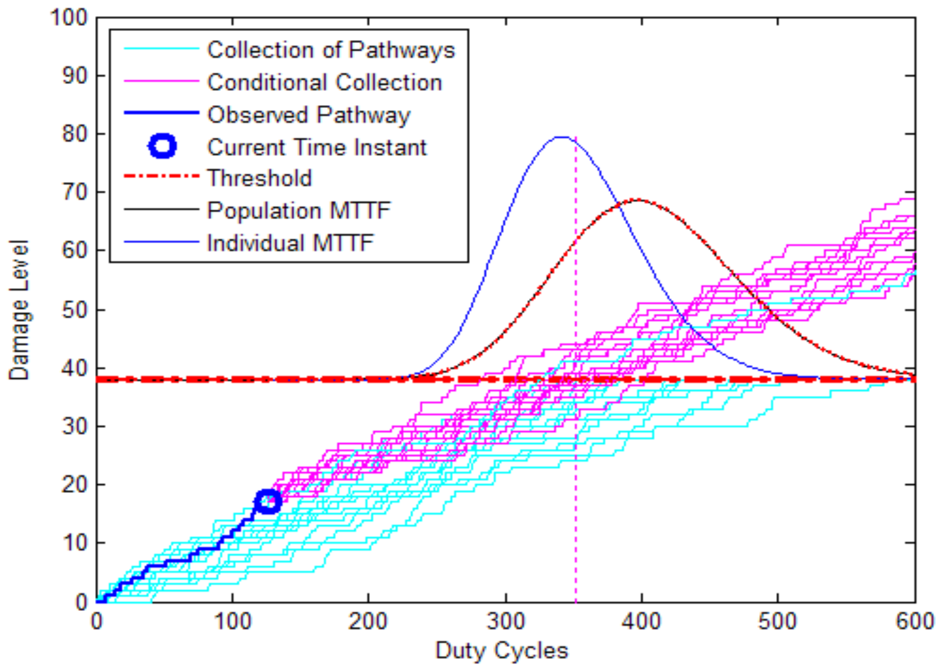


Figure 11: Markov Chain Model PoF estimation

the Shock model [96-98]. Instead of experiencing some known amount of shock at each random shock occurrence, the shock model allows for a shock of random size. In this model, the time between random shocks is a continuous variable, with the probability of shock often determined by the current degradation state, the operating conditions, or some combination thereof. The size of the shock may be based on a single shock size distribution, or other features such as the current degradation measure, the operating condition, or other measures available from the system may define it. Again, when the cumulative degradation measure crosses some pre-determined threshold, failure is said to have occurred; a probability of failure distribution is estimated from multiple simulated degradation measures.

2.3 Data Fusion for Prognostics

In any CBM system, many data sources are available for making prognostic estimates. These data sources may be redundant, such as measurements from a group of redundant sensors; as measurements from different subsystems within the same system. The main concern in any attempt at combining these data sources is to avoid producing a fused result which performs worse than the best performing constituent result [100]. This can happen if data sources with little or no prognostic capability are included in the data fusion set. Several data fusion techniques have been proposed specifically for application in prognostic algorithms. These will be briefly described here.

Jardine, et al. [101] explain the need for appropriate data fusion methods, stating "When multiple sensors are used, data collected from different sensors may contain different partial information about the same machine condition." The authors describe two classes of fusion methods: instantaneous and convolutive. Instantaneous methods involve time-independent, or memoryless, mixing functions; convolutive methods include some time dependence. Choi and Li [102] apply instantaneous mixing functions to fuse time and frequency data to estimate crack size in gear tooth traverse cracks. Goebel and Bonissone [102] use PCA-based data fusion methods to give a strong prognostic indicator when none is available in the data. Their method includes data analysis, PCA, adaptive network based fuzzy inference, and trending analysis. They apply the method to constant load, wet-end papermill breaks. The resulting model minimizes false alarms and provides adequate coverage of multiple types of breaks resulting from unknown causes.

Roemer, et al. [100] attempt to optimize data fusion results by reducing false alarm rates, increasing confidence levels in fault detection, and optimizing time to failure predictions. The authors specify three types of data fusion: feature creation, feature combination, and knowledge fusion. They specify knowledge fusion as a method to incorporate legacy information, physical model predictions, and signal-based information to improve prognostic estimates. The authors give an overview of common data fusion methods, including Bayesian Inference, Dempster-Shafer Method, Weighting/Voting Fusion, Fuzzy Logic Inference, and Neural Network Fusion. The authors illustrate the application of these methods to engine test cell sensor validation. They do not present results specific to prognostic applications.

Hughes, et al. [104] combine multiple fault mode indicators using traditional logical tests. They compare the performance of logical 'OR' tests and rank sum tests. These methods are used to reduce false alarms and improve fault isolation result. They apply their approach to disk drive failures. The two logical tests gave similar performance for each of the cases studied.

Wang and Coit [105] use joint PDFs to determine when any one of a collection of degradation measures crosses its threshold. They apply their methodology to reliability prediction based on degradation modeling of multiple degradation parameters. This allows for the combination of the information from multiple degradation models. The authors apply their methodology to simulated degradation data. Though the method is proposed for reliability monitoring, extension to prognostics is straightforward.

Several methods for prognostic estimation have been described in the previous sections, including algorithms in each of the three types: reliability-based, stressor-based, and degradation-based. Data fusion methods which have been applied to prognostics have also been described. Because a plethora of diverse algorithms and methodologies is available for prognostic estimation, it is necessary to develop a set of performance metrics by which the models can be compared.

2.4 Metrics for Evaluating Algorithm Performance

A major area of ongoing research in prognostics is model performance characterization. Performance metrics are necessary to allow model developers to compare two competing models, to understand the validity of a prognostic estimate, and to characterize model performance over different operating regimes, fault modes, or systems. Performance metrics for monitoring, fault detection, and diagnostic systems are well established [106], including accuracy, robustness or auto-sensitivity, spill-over or cross-sensitivity, fault detectability, uncertainty measures, fault detection time, and false alarm/missed alarm rates. Appropriate prognostic performance metrics, however, are less understood. Research in prognostic model performance metrics has focused on three areas: algorithm performance, computational speed, and economic incentive metrics. Obviously, it is desirable for prognostic algorithms to make accurate and precise RUL estimates. However, by the very nature of prognostics, these estimates ideally are made in an online fashion, which means the algorithm must have an acceptably short computation time. This is of particular import for systems which intend to have real-time data collection and RUL estimation running during system use. Metrics to characterize the computational cost of prognostic algorithms include complexity [107], specificity [108], CPU time [109], and “Big O” runtime notation [110]. It seems reasonable to assume that, given ample funds, the rate of increasing computer speed, and the use of more sophisticated computational methods [111], a system will be available to make an RUL prediction in any prescribed time frame. Economic metrics, such as Return on Investment (ROI), may consider model performance in making cost benefit analyses [95, 109, 113, 114]. Generally, these analyses show that PHM systems decrease maintenance costs, while increasing availability and improving safety. However, these metrics are primarily intended for a management group and are not considered model performance metrics for this research.

Measures of prognostic algorithm performance are of more interest to the current research. Current research largely focuses on method development; however, algorithm performance metrics have been the focus of several studies in recent years [48, 108, 109, 114, 116, 117]. Each of these studies resulted in different, and sometimes competing, performance metric definitions. However, each study highlights key shortcomings in traditional performance metrics. Prognostic algorithm performance metrics tend to fall into one of two categories: accuracy metrics or precision metrics. The field, however, is plagued by a problem common to many areas of prediction: “The more precise the remaining life estimate, the less probability that this estimate will be correct” [1]. Bearing this in mind, prognostic algorithm performance measures must be developed which will allow researchers and developers to accurately choose between competing prognostic models.

Unlike monitoring system models which produce one estimate of the system value at each point in time, prognostic models result in a time-series of RUL estimates. That is, the same value is being estimated at each prediction. In general, we are willing to suffer large errors and uncertainties early in life if a model will provide better performance as the system approaches failure [48, 109, 117]. Traditional error measures do not account for these progressive acceptable accuracy and precision levels. Saxena, et al. [109] suggest several metrics to account for this, the most interesting of which are the α-λ performance metrics for accuracy and precision. The α-λ performance dictates that the accuracy (or uncertainty) should be within some specified α*100% of the actual value within a relative distance, λ, to the actual failure, as shown in Figure 12. The formulation proposed by the authors gives a true or false value for the α-λ performance; namely, are the algorithm predictions correct to within α*100% at the λ-relative distance to failure. It may be more useful to consider when the algorithm is within the α*100% precision level or the fraction of time that the algorithm is within this requirement.

In addition to concerns about the importance of correctly accounting for temporal needs, prognostic models that predict that failure will occur before actual failure are generally considered better than those that predict failure will occur after the actual failure. RUL estimates greater than the actual remaining life leave room for unexpected failures and unplanned maintenance. Saxena, et al. [109], suggest an exponentially weighted accuracy metric to account for this, which gives more weight to late predictions and less weight to early predictions. This metric considers the RUL predictions made at one point in time, instead of the entire time series of predictions. A similar error metric was used in the 2008 PHM data challenge [118].

Although it is not generally applicable, for systems with known RUL, the model accuracy for a specific system at any given prediction time can easily be measured by the absolute percent error:

$$A = \frac{\left| RUL_{actual} - RUL_{estimated} \right|}{RUL_{actual}}$$

The accuracy of the model can be calculated across a population of systems as the mean absolute percent error (MAPE) at a given prediction time (or fraction of total life). This gives a time series of expected accuracy for the algorithm. Along with the measure of accuracy, a measure of RUL uncertainty

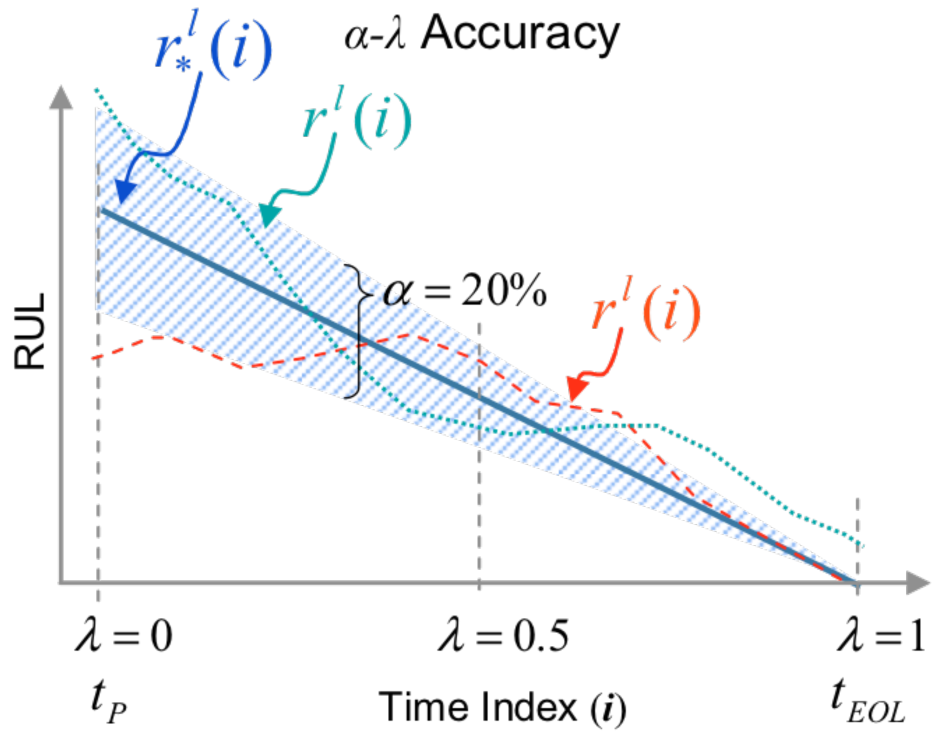


Figure 12: α-λ Performance for Accuracy [109]

should be included. This uncertainty estimate may be derived from the specific model architecture or estimated through a Monte Carlo bootstrap method; these methods of uncertainty estimation are described in great detail in [106, 119, 120].

This chapter has given an overview of the related prognostics research available in the open literature. The next chapter discusses the Process and Equipment Prognostics (PEP) toolbox that has been developed to aid in fast prototyping of prognostic models of the three types. The following chapters identify the specific need remaining in prognostics research which is addressed by this dissertation and the methodology developed to address this need, and results of its application.

3 The Process and Equipment Prognostics Toolbox

Development of an integrated health management system can be daunting because of the high level of complexity involved in identifying appropriate algorithms at each stage. To support this, a suite of MATLAB-based toolboxes have been developed at the University of Tennessee PROaCT lab with a range of monitoring, fault detection and prognostic capabilities. The Process and Equipment Monitoring (PEM) toolbox was developed to facilitate auto-associative modeling of process and system data, and fault detection [121]. The PEM toolbox includes auto-associative kernel regression models, auto-associative neural networks, and linear regression models for system monitoring; and sequential probability ratio test and error uncertainty limit monitoring fault detection methods. The results of both of these modules, as well as an independently developed diagnostic module if available, can be used to facilitate prognostic analysis. The Process and Equipment Prognostics (PEP) toolbox is a MATLAB-based toolbox developed to aid in development of empirical prognostic models of each of the three types. The PEP toolbox is designed to integrate with the previously developed PEM Toolbox. The results of process monitoring and fault detection produced by the PEM toolbox as well as the original system data are utilized by the PEP toolbox to make RUL predictions for the system, as shown in Figure 13.

The purpose of the PEP toolbox is to provide a complete set of tools to facilitate prognostic model development. A myriad of prognostic algorithms have been developed which use a variety of

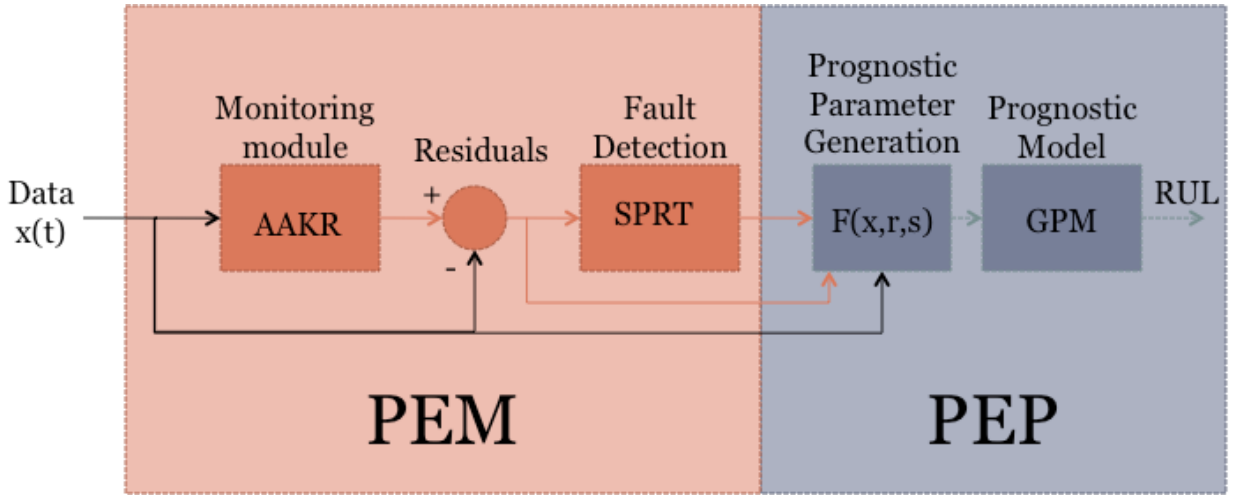


Figure 13: Integration of the PEM and PEP Toolboxes

information sources, models, data processing algorithms, etc. Typically, prognostic model development depends highly on the expertise of the developer. The PEP toolbox reduces the development burden on the system designer and facilitates the rapid development of competing models. As described in the previous chapter, prognostic algorithms can be classified by the type of information used to make RUL estimates. Algorithms in each of the three classes are included in the PEP toolbox. Appendix C gives a list of the functions currently included in the PEP toolbox, as well as the current code, including the help header, for each function. Appendix D includes an example application of the PEM and PEP toolboxes for health monitoring of the turbofan engine data presented in the 2008 PHM Challenge. The appendix includes the code used to develop the health monitoring system using the PEM and PEP toolboxes; the application of three prognostic models, one of each of the three types, to the PHM challenge data; and the resulting output and plots where illustrative.

3.1 Type I: Reliability Data-Based

These methods consider historical time to failure data which are used to model the failure time distribution. They estimate the life of an average component operating under historically average usage conditions. The PEP toolbox includes several statistical analysis techniques for Type I prognostics: Weibull, exponential, and Gaussian distribution models. Because of its flexibility in modeling infant mortality, random failure, and wear-out failure, the Weibull distribution is used most commonly for reliability analysis. These methods do not consider any information specific to the current system, except the total runtime without failure. Type I algorithms result in a distribution of possible failure times; the RUL for a system is determined as the MRL at the current lifetime.

3.2 Type II: Stressor-Based

A readily apparent disadvantage of reliability-based prognostics is that it does not consider the operating condition of the component. However, components operating under harsh conditions would be expected to fail sooner and components operating under mild conditions to last longer. Type II methods account for environmental stresses and usage conditions (e.g. temperature, load, vibration, etc.) when estimating the system RUL. These models estimate the lifetime of an average component operating under the given usage conditions. The PEP toolbox includes three common Type II algorithms: Markov Chain models, Shock models, and Proportional Hazards models. Due to the stochastic nature of these algorithms, each results in a distribution of failure times. The RUL estimate here is estimated for a given reliability confidence level. A confidence level of 0.5 may be chosen to obtain the average RUL for non-safety critical components. For safety-critical or high value systems, a conservative reliability confidence level of 0.95 may be more appropriate.

3.3 Type III: Degradation-Based

The final class of algorithms considers the way in which a specific component responds to its specific usage. It is most commonly constructed through the identification or development of a degradation parameter, which is trended towards failure. The PEP toolbox utilizes a general path model for Type III estimates. It also includes functionality to include prior information in the regression parameters of a new component or system via Bayesian updating techniques. This methodology is used in the current research and is described in great detail in Chapter 0. If a Type III model is applicable to a given system, identification of an appropriate prognostic parameter is paramount to model performance. An automated methodology for identifying prognostic parameters from multiple data sources is the focus of this dissertation. By formalizing the suitability of a prognostic parameter, identification of an appropriate parameter becomes a straightforward optimization problem; the methodology outlined in this research utilizes Genetic Algorithm optimization to find an optimal or near-optimal parameter. The parameter suitability metrics and identification methodology is fully included in the PEP toolbox and can make use of many available data sources, including the sensed measurements from the system, monitoring system residuals and fault detection results obtained from the PEM toolbox.

3.4 Life Cycle Prognostics

As a system moves through life, different prognostic model types may be appropriate, as shown in Figure 14. The PEP toolbox allows a prognostic system to transition between the three prognostics types as more information about the system and its use becomes available. For a newly fabricated component or system, the only possible estimate of RUL is reliability based, because no information is available about how the unit will be used and no data is available from its use. For instance, a new car tire is said to last 40,000 miles based on data collected from similar tires operating in "normal" conditions. However, during operation planning and early life, specific information about the future use

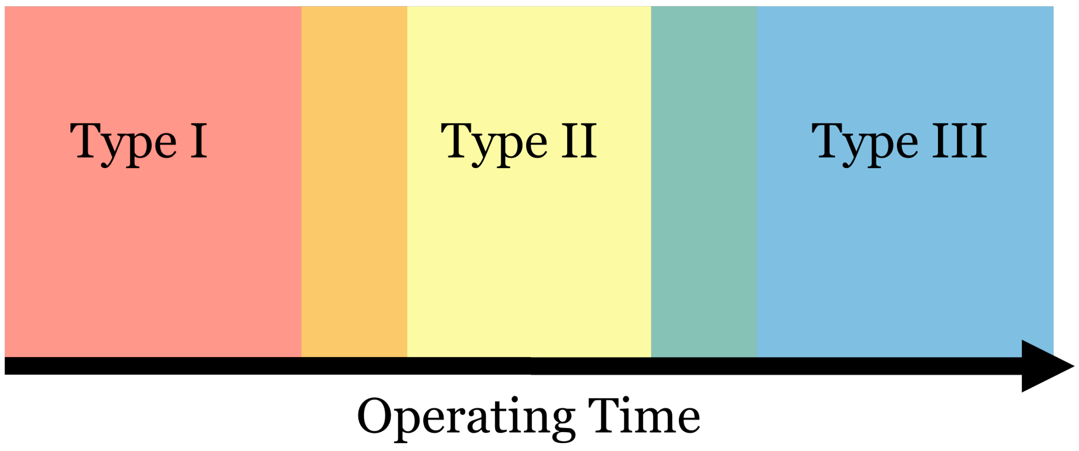


Figure 14: Life Cycle Prognostics

of the unit should be available. At this point, Type II methods can be used to make RUL estimates. Because these estimates consider the specific use of the unit, they should provide more accurate and precise RUL predictions. If it is known that the tire will operating on primarily unpaved roads, the failure time distribution will shift down to account for the harsher usage condition. Finally, as the unit is used and data becomes available from its use and degradation, Type III prognostics may be applicable to obtain an even more accurate estimate of RUL. If performance characteristics such as tread-depth or pressure can be measured on the tire, an individual-based estimate of the RUL can be made for that specific tire. Because the PEP toolbox includes functionality to develop all three types of prognostic models, transitioning between the RUL estimation methods is more readily performed as the operator sees fit. A key area of future work is developing a method to seamlessly shift between the three prognostic types as is appropriate, considering the RUL information of two types of models during the transition period.

The PEP toolbox is a set of MATLAB-based tools designed as a unified platform for developing data-based prognostic models. It is designed to couple with the PEM toolbox to allow for a full monitoring, fault detection, and prognostic system. The toolbox facilitates and automates prognostic model development by implementing the three prognostic types and additional methods to ease the developmental burden on the prognostic system developer. Identification of an appropriate prognostic parameter is key for the application of Type III prognostic models. Development of an automated parameter identification method is the focus of this dissertation. The following section outlines the development of the parameter suitability metrics and the identification methodology. This is followed by example applications of the parameter optimization method and RUL estimation.

4 Methodology

The problem of accurately and precisely predicting remaining useful life is very complicated; as such many methodologies and algorithms have been proposed to address this problem, which were discussed previously. This work focuses on Type III, or condition-based prognostics. Extrapolation of a general path model (GPM) to some pre-defined failure threshold is the most common Type III method. The GPM algorithm employed is a formulation of the model originally proposed by Lu and Meeker [1]. This method is described in detail in the following sections, including a Bayesian updating methodology for incorporating prior information into the GPM fit. This is followed by a discussion of the features necessary in an appropriate prognostic parameter for trending. Finally some discussion of automated methods for identifying the optimal parameter is given.

4.1 The General Path Model

The General Path Model (GPM), also called degradation modeling, was first proposed by Lu and Meeker [4] to move reliability analysis methods from failure-time analysis to failure-process analysis. Traditional methods of reliability estimation use failure times recorded during normal use or accelerated testing to estimate a time of failure (TOF) distribution for a population of identical components. In contrast, GPM uses degradation measures to estimate the TOF distribution. The use of historical degradation measures allows for the direct inclusion of censored data, which gives additional information on unit-wise variations in a population.

GPM analysis begins with some assumption of an underlying functional form of the degradation path for a specific fault mode. The degradation of the ith unit at time tj is given by:

$$y_{ij} = \eta(t_j, \phi, \theta_i) + \varepsilon_{ij}$$

where ϕ is a vector of fixed (population) effects, θi is a vector of random (individual) effects for the ith component, and εij ~ N(0,σ2ε) is the standard measurement error term. Application of the GPM methodology involves several assumptions. First, the degradation data must be describable by a function, η; this function may be derived from physics-of-failure models or from the degradation data itself. In order to fit this model, the second assumption is that historical degradation data from a population of identical components or systems is available. This data should be collected under similar use (or accelerated test) conditions and should reasonably span the range of individual variations between components. Because GPM uses degradation measures instead of failure times, it is also not necessary that all historical units are run to failure; censored data contains information useful to GPM forecasting. The final assumption of the GPM model is that there exists some defined critical level of degradation, D, beyond which a component no longer meets its design specifications, i.e. the component has failed. Therefore, some components should be run to failure in order to quantify this degradation level. Alternatively, engineering judgment may be used if the nature of the degradation parameter is explicitly known.

Several methods are available to estimate the degradation model parameters, ϕ and θ. In some cases, the population parameters may be known in advance, such as the initial level of degradation. If the population parameters are unknown, estimation of the vector of population characteristics, ϕ, is trivial; by fitting the model to each exemplar degradation path, the fixed effects parameters can be taken as the mean of the fitted values for each unit. The variance of these estimates should be examined to ensure that the parameters are in fact population-wide parameters. If significant variability is present, the parameters should be considered random and moved to the θ vector. A two-stage method of parameter estimation was proposed by Lu and Meeker [1] to estimate distribution parameters for the random effects.

In the first stage, the degradation model is fit to each degradation path to obtain an estimate of θ for that unit; these θ's are referred to as stage-1 estimates. It is convenient to assume that the stage-1 estimates, or an appropriate transformation, $\Theta=H(\theta)$, is normally (or multivariate normally) distributed so that the random effects can be fully described using only a mean vector and variance-covariance matrix without significant loss of information. If this assumption is not justifiable, the GPM methodology can be extended in a natural way to allow for other random effects distributions.

In the second stage, the stage-1 estimates (or an appropriate transformation thereof) are combined to estimate ϕ, μ_θ, and Σ_θ. At this stage, if the variance of some previously assumed random parameter is effectively zero, this parameter should be considered a fixed effects parameter and should be removed from the random parameter distribution.

In their seminal paper, Lu and Meeker [1] describe Monte Carlo methods for using the GPM parameter estimates to estimate a time to failure distribution and corresponding confidence intervals. Because the focus of this paper is estimating time to failure of an individual component, these methods will not be described here.

Several limitations and areas of future work of the GPM have been identified by Meeker, et al. [118]. Some of these areas have been addressed in work by other authors. First, Meeker, et al. cite the need for more accurate physics of failure models. While such models are helpful for understanding degradation models, they may not be necessary for RUL estimation. In fact, if exemplar data sets cover the range of likely degradation paths, it may be adequate to fit a function which does not explain failure modes but accurately models the underlying relationships. With this idea, neural networks have been applied to GPM reliability analysis [5, 6, 123].

In addition, the GPM was originally developed for reliability analysis of only one fault mode. In practical applications, the system of interest may consist of several components each with different fault modes, or of one component with several possible, even simultaneous fault modes. These multiple degradation paths may be uncorrelated, in which case extension of the GPM is trivial: reliability of a component for all degradation modes is simply the product of the individual reliabilities, and RUL can be considered some function of the RULs for each fault mode, such as the minimum. If, however, the degradation measures are correlated, extension of the GPM is more complicated. For example, in the case of tire monitoring, several degradation measures may contain information about tire reliability, including wall thickness, tire pressure, and tire temperature. However, it is easy to see that these measures may be correlated; a higher temperature would cause a higher pressure, etc. The case of multiple, competing degradation modes is beyond the scope of the current work. A discussion of the problem can be found in Wang and Coit [105].

4.2 GPM for Prognostics

The GPM reliability methodology has a natural extension to estimation of remaining useful life of an individual component or system; the degradation path model, yi, can be extrapolated to the failure threshold, D, to estimate the component's time of failure. This type of degradation extrapolation was proposed early on by Upadhyaya, et al. [5]. In that work, the authors used both neural networks and nonlinear regression models to predict the RUL of a small induction motor. The prognostic methodology used for the current research is described below.

First, exemplar degradation paths are used to fit the assumed model. The stage-1 parameter estimates are used to evaluate the random-effects distributions, to determine the mean population random effects, the mean time to failure (MTTF) and their associated standard deviations, and to estimate the noise variance in the degradation paths. The MTTF distribution can be used to estimate the time of failure for any component which has not yet been degraded.

As data is collected during use, the degradation model can be fit for the individual component. This specific model can be used to project a time of failure for the component. Because of noise in the degradation signal, the projected time of failure is not perfect. A prediction interval (PI) about the estimated parameters can be evaluated as:

$$\theta \in \left[\hat{\theta} - t_{n-1,\alpha/2} s\sqrt{1+1/n}, \hat{\theta} + t_{n-1,\alpha/2} s\sqrt{1+1/n}\right]$$

where $t_{n-1,\alpha/2}$ is the student's t-distribution, n is the number of observations used to fit the model, and s is the standard deviation of the degradation model parameters. The standard deviation of the parameters can be estimated through traditional linear regression techniques. The range of model parameters can be used to project an PI about the estimated time of failure.

The methodology described considers only the data collected on the current unit to fit the degradation model. However, prior information is available from the historic degradation paths used for initial model fitting, including the mean degradation path and associated distributions. This data can provide valuable knowledge for fitting the degradation model of an individual component, particularly when only a few data points have been collected or the collected data suffers from excessive noise. Bayesian updating methods have been developed to incorporate this additional information in the fitted model.

4.3 Incorporating Prior Information with Bayesian Methods

The current research investigates using Bayesian methods to include prior information for linear regression problems. However, as discussed above, the GPM methodology can be applied to nonlinear regression problems as well as other parametric modeling techniques such as Neural Networks. Other Bayesian methods could be applied to these types of models, but such application is beyond the scope of the current research. For a complete discussion of Bayesian statistics including other Bayesian update methods, the interested reader is referred to [124-126]. In addition, work by Robinson and Crowder [126] focuses on Bayesian methods for nonlinear regression GPM.

A linear regression model is given by:

$$Y = bX$$

The model parameters are estimated as:

$$b = \left(X^T \Sigma_y^{-1} X\right)^{-1} X^T \Sigma_y^{-1} Y$$

where Σ_y is the variance-covariance noise matrix for the response observations. It is important to note that the linear regression model is not necessarily a linear model. The data matrix *X* can be populated with any function of degradation measures, including higher order terms, interaction terms, and functions such as *sin(x)* or e^x. If prior information is available for a specific model parameter, i.e. β_j~N(β_{jo},σ^2_β), then the matrix *X* should be appended with an additional row with value one at the *j*th position and zero elsewhere, and the *Y* matrix should be appended with the a priori value of the *j*th parameter.

$$\begin{aligned} X^* &= [X; \quad 0 \quad \cdots \quad 0 \quad 1 \quad 0 \quad \cdots \quad 0] \\ Y^* &= [Y; \quad \beta_j] \end{aligned}$$

Finally, the variance-covariance matrix is augmented with a final row and column of zeros, with the variance of the a priori information in the diagonal element.

$$\Sigma_y^* = \begin{bmatrix} \sigma_y^2 & 0 & \cdots & 0 \\ 0 & \ddots & 0 & \vdots \\ 0 & \cdots & \sigma_y^2 & 0 \\ 0 & \cdots & 0 & \sigma_{\beta_j}^2 \end{bmatrix}$$

If knowledge is available about multiple regression parameters, the matrices should be appended multiple times with one additional row for each parameter.

It is convenient to assume that the noise in the degradation measurements is constant and uncorrelated. Some a priori knowledge of the noise variance is available from the exemplar degradation paths. If this assumption is not valid for a particular problem, then other methods of estimating the noise variance must be used. The assumption of uncorrelated noise allows the variance-covariance matrix to be a diagonal matrix consisting of noise variance estimates and a priori knowledge variance estimates. If this assumption is not valid, including covariance terms is trivial; again these terms can be estimated from historical degradation paths.

After a priori knowledge is used to obtain a posterior estimate of degradation parameters, this estimate becomes the new prior distribution for the next estimation of degradation parameters. The variance of this new knowledge is estimated as:

$$\frac{1}{\sigma_{post\beta_j}^2} = \frac{n}{\sigma_y^2} + \frac{1}{\sigma_{prior\beta_j}^2}$$

where *n* is the number of observations used to fit the current model.

4.4 Choosing a Prognostic Parameter

Identification of an appropriate prognostic parameter is key for applying a GPM prognostic model to a system. An ideal prognostic parameter has three key qualities: monotonicity, prognosability, and trendability [128]. Monotonicity characterizes the underlying positive or negative trend of the parameter. This is an important feature of a prognostic parameter because it is generally assumed that systems do not undergo self-healing, which would be indicated by a non-monotonic parameter. However, this assumption is not valid for some components such as batteries, which may experience some degree of self-repair during short periods of nonuse. The monotonic trend is considered valid when considering an entire system, even if individual components or sub-systems may experience some self-repair. Prognosability gives a measure of the variance in the critical failure value of a population of systems. Ideally, failure should occur at a crisp, well-defined degradation level. A wide spread in critical failure values can make it difficult to accurately extrapolate a prognostic parameter to failure. Finally, trendability indicates the degree to which the parameters of a population of systems have the same underlying shape and can be described by the same functional form. These three intuitive metrics can be formalized to give a quantitative measure of prognostic parameter suitableness. Ideally, these metrics would each range from zero to one, one indicating a very high score on that metric and zero indicating that the parameter is not suitable according to the particular metric. Figures 15 and 16 give two populations of prognostic parameters; the parameter shown in Figure 15 is considered useful for prognostics, while the parameter in Figure 16 is not. These two populations will be used to illustrate the three suitability metrics as they are discussed.

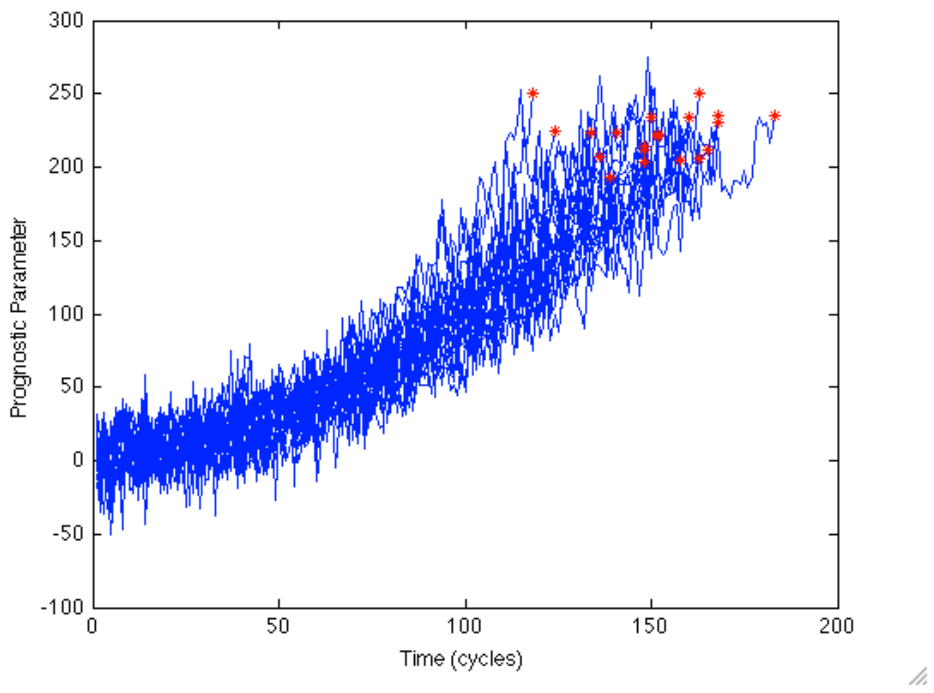


Figure 15: Population of "Good" Prognostic Parameters

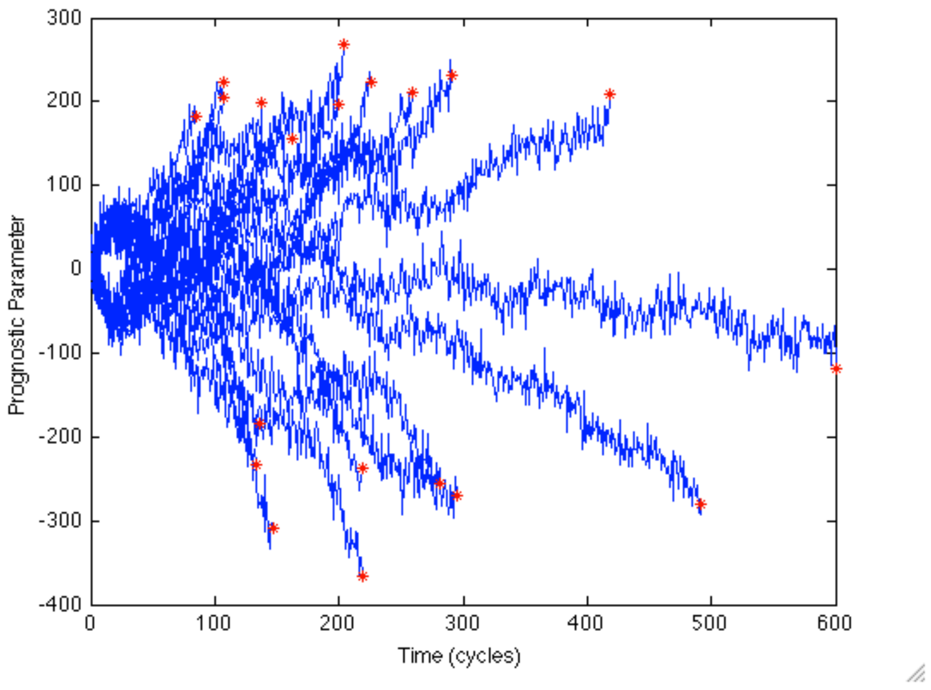


Figure 16: Population of "Bad" Prognostic Parameters

Monotonicity is a straightforward measure given by:

$$Monotonicity = mean\left(\left|\frac{\#\, pos\, {}^{d}\!/_{dx}}{n-1} - \frac{\#\, neg\, {}^{d}\!/_{dx}}{n-1}\right|\right)$$

where n is the number of observations in a particular history. The monotonicity of a population of parameters is given by the average difference of the fraction of positive and negative derivatives for each path. When using data collected or inferred from actual systems, it is important to adequately smooth the data to give more accurate estimates of the derivatives. Numerical calculation of a function derivative should rarely be left to a simple difference function; the addition of noise makes this method inaccurate and impractical. In practice, fitting a line to a small portion of the data, perhaps five or ten observations, and taking the derivative to be the slope of that line will give a more realistic measure of the slope. When this method is employed, the above equation for monotonicity need only be adjusted for the number of calculated derivatives. Instead of n-1 derivatives, n-m derivatives may be calculated, where m is the number of data points used to calculate one derivative. The population of good parameters shown in Figure 15 is clearly monotonically increasing, in the absence of noise; the monotonicity metric for this parameter is 0.940. Conversely, the bad parameter shown in Figure 16 has monotonicity of 0.501. It is important to note that the current formulation of monotonicity does not consider if the entire population is monotonic in the same direction, only that each individual exhibits an either generally increasing or decreasing trend. This is an undesirable feature in the prognostic parameter; however, it is considered in characterizing the prognosability which looks at how well clustered failure values are. If failure values for the entire population are well clustered and the individual parameters are monotonic, then the population must have either an increasing or decreasing monotonicity.

Prognosability is calculated as the deviation of the final failure values for each path divided by the mean range of the path. This is exponentially weighted to give the desired zero to one scale:

$$\Pr ognosability = \exp\left(-\frac{std(failurevalues)}{mean\left(\left|failurevalue - startingvalue\right|\right)} \right)$$

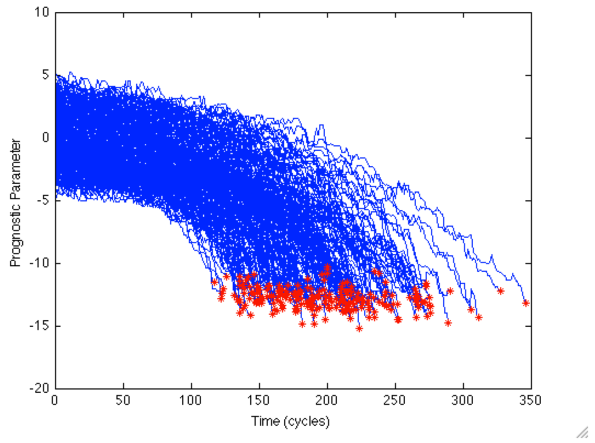
This measure encourages well-clustered failure values, i.e. small standard deviation of failure values, and large parameter ranges. This gives the model a long range to predict a very precise value, which can be related to the notice period discussed previously. The failure values for the good prognostic parameter are very well clustered, following a wide range; the prognosability is 0.930. The failure values of the population of bad prognostic parameters cover a wide range of values; this parameter has prognosability of only 0.346.

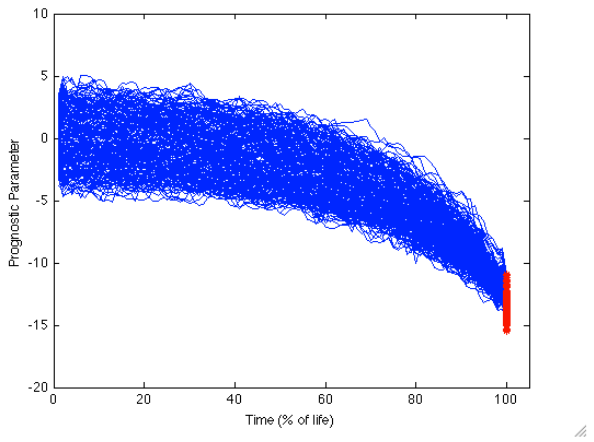
Characterizing the trendability of a population of parameters poses significant difficulty compared to the other two metrics. A candidate parameter is trendable if the same underlying functional form can model each parameter in the population. Initially, trendability was characterized by comparing the fraction of positive first and second derivatives in each parameter. However, this naïve approach was highly susceptible to noise and did not provide a clear distinction between trendable and not-trendable parameters. An improved method for characterizing trendability is used in which prognostic parameters are re-sampled with respect to the fraction of total lifetime. This results in each prognostic parameter containing exactly 100 observations, with each observation corresponding to 1% of lifetime. The linear correlation is calculated across the population of prognostic parameters, and the trendability is given by the smallest absolute correlation:

$$Trendability = \min\left(\left|corrcoef_{ij}\right|\right)$$

Figure 17 gives a comparison of a population of prognostic parameters as a function of cycles and as a function of the percent of life. It is visually obvious in Figure 17(a) that the parameters of this population can be described by the same underlying function. By transforming to the percent lifetime space in Figure 17(b), it is straightforward for a computer to recognize this relationship as well. The "good" population shown in Figure 15 has trendability of 0.984, while the population of "bad" parameters shown in Figure 16 has trendability of 0.288. The three suitability metrics for both populations are summarized in Table 1.

The effect of noise on the parameter suitability metrics is of particular import. Specifically, bad parameters which are noisy should still be identified as bad. It is slightly trickier for good parameters. Some noise contamination can be tolerated by the GPM methodology, but significant levels of noise will appreciably reduce model performance. Three sets of good and bad parameters are tested; each of the six parameters is plotted in Appendix A. The first set is completely noise free; the second set is





(a) (b)

Figure 17: Prognostic Parameter vs (a) Time in Cycles and (b) Time in % of Full Lifetime

Table 1: Example Prognostic Parameter Suitability Metrics

	Monotonicity	Prognosability	Trendability
Good Parameter (Figure 15)	0.940	0.930	0.984
Bad Parameter (Figure 16)	0.510	0.346	0.288

contaminated with Gaussian noise at a level of 20% of the mean parameter value; the third set is contaminated with Gaussian noise at 80% of the mean parameter value. The suitability metrics for each of these parameters are given in Table 2. As the table shows, the parameter suitability metrics for a "good" prognostic parameter are only slightly degraded at a noise level of 20%. The suitability metrics for the very noisy parameter, however, are significantly degraded, indicating that this parameter would result in degraded model performance. The three "good" parameters are used in the following chapter to investigate the effect of noise on model performance. The suitability metrics for the "bad" parameter are not significantly changed by the addition of noise. This is a useful result indicating that even the noise-free bad parameter is poorly suited to prognostics.

Several methods are available for identifying candidate prognostic parameters, including visual inspection of sensed data and model residuals, Principal Component Analysis, traditional optimization method, and Genetic Algorithms approaches. Traditionally, parameter identification is done through visual inspection and engineering judgment. While visual inspection can lead to the identification of useful prognostic parameters, it can be tedious and time consuming when parameters are needed for several components or fault modes, and the optimal parameter may be overlooked in favor of a suitable one. Automated methods for identifying prognostic parameters are possible with a formalized set of metrics to characterize their suitability. By defining a fitness function as a weighted sum of the three metrics:

$$fitness = w_m monotonicity + w_p prognosability + w_t trendability$$

a set of prognostic parameters can be compared to determine the most suitable one. Here, the constants w_m, w_p, and w_t control how important each metric is in the optimization. For most

Table 2: Effect of Noise on Parameter Suitability Metrics

	Monotonicity	Prognosability	Trendability
Good Parameter			
Noise Free	1.000	0.937	1.000
20% Noise	0.955	0.913	0.986
80% Noise	0.715	0.761	0.713
Bad Parameter			
Noise Free	0.439	0.344	0.228
20% Noise	0.409	0.340	0.192
80% Noise	0.441	0.343	0.153

applications, these constants can each be identically one to give equal weight to each parameter feature. However, as discussed previously, monotonicity may not be an appropriate prognostic parameter feature for some applications, such as for battery health monitoring. In that case, monotonicity may be excluded from the parameter fitness calculation by giving it a weight of zero. In addition to optimizing for the best combination of parameter suitability metrics, the prognostic parameter can also be optimized for other features perhaps not directly related to parameter performance. For instance, minimizing the weights of each of the inputs in a linear combination may be of value because it reduces the complexity of the solution and forces unimportant weights to go to zero; this can be achieved by adding an additional term to the fitness function, $w_w \frac{\sum weights}{\sigma}$, where w_w is a constant similar to w_m, w_p, and w_t, and σ is either the standard deviation or the range of the input. Additionally, in order to reduce the uncertainty in the RUL prediction, it is beneficial for the first and second derivatives to have the same sign, i.e. increasing functions are concave up and decreasing functions are concave down. This can be included in the fitness function by simply adding a large penalty for mismatch of first and second derivatives. Other such features may also be included as they apply to the desired parameter optimization. The fitness function is used with optimization techniques such as gradient descent, genetic algorithms, and machine learning methods to identify useful prognostic parameters.

4.5 Prognostic Parameter Optimization

After developing an appropriate fitness function, defined by the suitability metrics given above and other desirable features, any number of well-established optimization methods may be applied to identify the optimal prognostic parameter. Optimization methods can be broken into two major groups: deterministic methods and stochastic methods. Deterministic methods include gradient descent, Newton method, quasi-Newton method, Nelder-Mead, and brute-force search [129]. These methods suffer from high computational cost and sensitivity to local optima. Gradient descent methods attempt to traverse the fitness landscape to identify the optimal solution, but are easily fooled by local optima. Newton's method uses the second order Taylor expansion to find the roots of the derivative of the fitness function, although this method breaks down for derivatives which are not locally well approximated by a quadratic function. The Quasi-Newton method uses a mixed quadratic and cubic line search procedure and the Broyden–Fletcher–Goldfarb–Shanno (BFGS) formula for updating the approximation of the Hessian matrix. Nelder-Mead is a direct-search method that uses only function values, that is, it does not require computation of derivatives; this method works well for non-smooth objective functions. These methods are further complicated by multiple optimization parameters. For complicated fitness functions, these methods are likely to settle in a local optimal solution instead of the global optimum. Brute force methods attempt to identify every possible solution to a given optimization problem and compare the performance of all solutions to identify the optimal one. Brute force methods are obviously computationally intensive, and are well suited to optimizing parameters with discrete values. They are not readily applicable to large, continuous optimization spaces.

Stochastic methods include genetic algorithms, particle swarm, stochastic gradient descent, and simulated annealing [130]. These methods are more robust to local optima because they search the entire solution space stochastically instead of moving along a given path. Given enough time, stochastic methods are guaranteed to find the global optimal solution; given a set amount of time, they are able to find near-optimal solutions [129]. Stochastic methods are often coupled with a traditional gradient descent to refine the results of the optimization. Because stochastic methods are more robust to local optima and handle multiple optimization parameters well, this research will focus on these methods for parameter selection. The most common stochastic optimization method is genetic algorithms (GA), which is described in some detail in the next section. Additional multi-objective stochastic optimization methods include normal boundary intersection method, normal constraint method, and successive pareto optimization method, but GA optimization is the method of choice for the current research. A brief overview of GA optimization is given next.

4.5.1 Genetic Algorithms

Classical optimization attempts to minimize the cost function by starting at an initial set of parameter values and utilizing function and derivative information to hone in on a minimum value. This type of optimization can quickly breakdown if the initial values are close to a local minimum; classical optimization will assume the first minimum it finds is the global minimum which is not necessarily true. Genetic algorithm (GA) optimization works differently by testing a random population of initial parameter values and mimicking the processes of evolution to optimize these values, as outlined in Figure 18. Because of the pseudo-random nature of the parameter populations evaluated, GA optimization is sometimes able to ignore local minima in search of a global minimum. Each of the steps in genetic algorithm optimization is quickly discussed below. A full discussion of continuous genetic algorithms is available in Haupt [129].

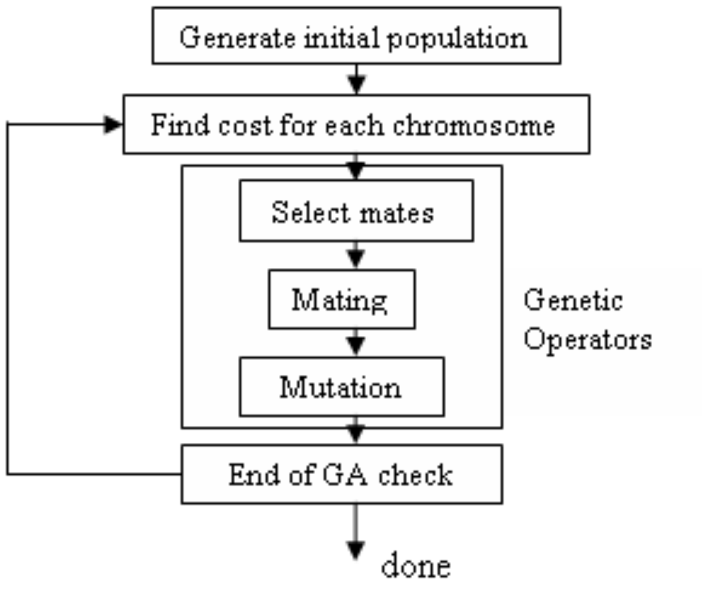


Figure 18: Flowchart of Genetic Algorithm Optimization

4.5.1.1 Initial Population and Cost

The goal of genetic algorithms is to search for an optimal solution of a given problem. This process begins by defining a "chromosome" to be a vector of variable values, a possible solution to the optimization. If a particular optimization problem has Nvar variables, then a possible chromosome for that optimization is

$$chromosome = \left[p_1, p_2, p_3, \ldots, p_{N_{\text{var}}} \right]$$

where p_i is a continuous-valued variable. These pi variables are called "genes." A "population" is the set of all chromosomes in one generation. Each chromosome has an associated fitness value found by evaluating the fitness function for the gene values in that chromosome.

Fitness functions are mathematical functions used to assign a value to a set of variables (chromosome). Fitness functions characterize the values and parameters to be optimized. Development of appropriate fitness functions is critical for genetic algorithm optimization: "The success of the GA on a particular function is certainly related to how the function is 'encoded'" [132]. Overly complex fitness functions can slow GA performance; overly simplified functions may not adequately describe the fitness surface.

After the fitness function has been evaluated for each parameter (or set of parameters), the initial population is ordered in terms of its performance. The genetic algorithm then uses three genetic operators to generate a new, hopefully fitter generation of solution chromosomes.

4.5.1.2 Genetic Operators

Three evolutionary modes exist for generating a subsequent generation: reproduction, crossover, and mutation.

Reproduction, or selection, results in an exact copy of the best performing chromosomes into the new population. This echoes Darwin's theory of Survival of the Fittest. Because of this mode, it is always guaranteed that the best fitness function cost will either improve or at least remain constant between successive generations; the best cost across generations will never worsen. In creating a new generation, a set number of chromosomes are produced via reproduction: the number of elite individuals.

Crossover, or mating, is the creation of offspring chromosomes from two parent chromosomes retained from the previous generation. This method is used to generate a specified fraction of the chromosomes in the new generation, the crossover fraction. Several methods have been developed to perform this crossover. The GA crossover of this research utilizes a "scattered" crossover function. Scattered crossover creates a random binary vector of length Nvar, called the mask. A child is generated by accepting the genes of the first parent when the mask vector is valued 1. Genes of the second parent are selected when the mask vector is values 0:

$$\begin{array}{cc} parent_1 & \left[p_1, p_2, p_3, \ldots, p_{N_{\text{var}}}\right] \\ parent_2 & \left[q_1, q_2, q_3, \ldots, q_{N_{\text{var}}}\right] \\ mask & \left[0,1,1,\ldots,0\right] \\ offspring & \left[q_1, p_2, p_3, \ldots, q_{N_{\text{var}}}\right] \end{array}$$

The final model of new chromosome generation is mutation, which has obvious connections to evolutionary biology. In each generation, random mutations alter a set percent of the variables in the list of chromosomes, the mutation rate. Mutations prevent overly fast convergence and facilitate exploration of other areas of the fitness surface. In general, a gene is randomly to be mutated. This gene can either be replaced by a new, random value or a random value can be added to the gene. For the research presented here, mutation is based on a Gaussian distribution. This distribution uses random numbers taken from a Gaussian distribution centered on zero. These random numbers are added to the genes chosen for mutation.

Through these three methods, a new generation is created which is then evaluated for fitness. This cycle is repeated until one of several stopping criteria is met: a set number of generations is reached, a minimum average change between generations is not met, or the optimal value of the fitness function is obtained.

Like other optimization methods, GA has several advantages and disadvantages. One obvious advantage is the use of a fitness function to determine which parameter values to keep; classical optimization follows gradient curves and can get stuck in a local minimum. Because of the threat of local minimum, the initial parameter values in classical optimization must be close to the optimal values. GA optimization does not have this problem of initial values because of the inherent randomness in generation construction. As mentioned above, this randomness may allow GA to ignore local minima in search of the global minimum. If given enough time, GA will find the optimal, or near optimal solution to most problems. However, the disadvantage is that this can be computationally more intensive than a classical approach. Another disadvantage lies in identifying an appropriate fitness function for the problem. For model optimization, the mean squared error of the model is an apparent and appropriate fitness function. In some situations, however, additional parameters are needed to produce an optimal and appropriate solution. In these cases, multi-objective optimization (MOO) is used to produce a solution which is acceptable for several criteria.

4.5.1.3 Multi-Objective Optimization

In some applications, model optimization has several, sometimes competing objectives. In this case, a solution must be identified which satisfies some or all of these objectives to a greater or lesser degree. Bentley and Wakefield [133] identify six methods for combining multiple objectives in one fitness function. This research will implement the sum of weighted objectives (SWO) method. This is the simplest and, therefore, most common method. To create a SWO fitness function, each objective is given a weight to specify its relative importance; the multiple objectives are then summed to produce a fitness function:

$$fitness = \sum w_i f_i$$

where f_i is the fitness function for the i^{th} objective. The SWO method forces the GA to converge to a compromise solution and results in a best compromise solution based on the weights of each objective. It should be noted that in practice determining appropriate weight values for each objective can be very difficult.

4.5.2 Prognostic Parameter Input Selection

The computation time necessary to execute a GA optimization increases exponentially with the number of parameters being optimized [134]. Many data sources are available as inputs to a prognostic parameter, including sensor readings, monitoring system residuals, fault detection and diagnostic results, and operating condition indicators. In order to alleviate the computational burden, possible prognostic parameter inputs can be pre-screened to determine if they are expected to contribute to the overall prognostic parameter suitability. In fact, removing unnecessary inputs may improve the optimization results. While the optimization should give these inputs weights of zero, this rarely happens before the optimization is terminated. When optimizing a weighted sum of possible inputs, it is straightforward to determine which inputs will improve the prognostic parameter and which will be detrimental. The proposed input selection method involves simply evaluating the fitness function for each candidate input and rejecting any inputs with fitness below a specified threshold. When using the simple sum of the three suitability metrics as a fitness function, a threshold of 1.0 – 1.5 is generally appropriate for identifying useful inputs.

4.6 Combined Monitoring and Prognostic Systems

Figure 19 gives a combined monitoring, fault detection, and prognostics system similar to the one used in this research. The main difference between the system shown here and that shown in Figure 19 is the absence of the fault diagnostic module. The research presented in this dissertation does not consider different fault modes when making prognostic estimations; however, it would be trivial to use diagnostic information to inform the prognostic model. If fault diagnostics are available, then multiple prognostic models should be developed, one for each fault mode of interest. The monitoring system employs an Auto-Associative Kernel Regression (AAKR) model for monitoring and the Sequential Probability Ratio Test (SPRT) for fault detection. Both of these methods are described in broad detail below. The interested reader is referred to [106] for a more complete discussion of AAKR and [47] for SPRT.

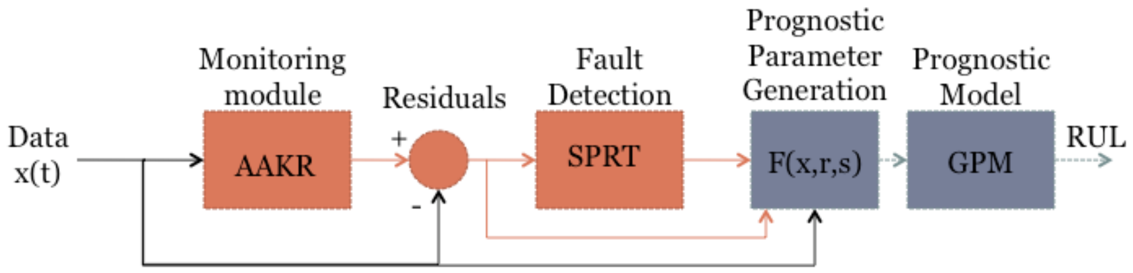


Figure 19: Combined Monitoring and Prognostic System

Auto-Associative models can generally be considered an error correction technique. These models compare a new observation to those seen in the past to estimate how the system "should" be running. These corrected predictions can be compared to the measured data to identify faulted operation. Several auto-associative architectures are available, including auto-associative neural networks, auto-associative kernel regression (AAKR), and multivariate state estimation technique [106]. This research employs the AAKR algorithm for system monitoring.

Auto-associative kernel regression (AAKR) is a non-parametric, empirical technique. Exemplar historical observations of system operation are stored in a data matrix. As a new observation is collected, it is compared to each of the exemplar observations to determine how similar the new observation is to each of the exemplars. This similarity is quantified by evaluating the distance between the new observation and the exemplar. Most commonly, the Euclidean distance is used:

$$d_i = \sqrt{\sum_{j=1}^{m}\left(X_j - x_{ij}\right)^2}$$

where di is the distance between the new observation, X, and the ith exemplar, xi. The distance is converted to a similarity measure through the use of a kernel. Many kernels are available; this research uses the Gaussian kernel:

$$s_i = \exp\left(-\frac{d_i^2}{h^2}\right)$$

where s_i is the similarity of the new observation to the i^{th} exemplar. Finally, the "corrected" observation value is calculated as a weighted average of the exemplar observations:

$$\hat{X} = \frac{\sum s_i x_i}{\sum s_i}$$

Monitoring system residuals are then generated as the difference between the actual observation and the estimated, corrected observation. These residuals are used with a SPRT to determine if the system is operating in a faulted or nominal condition. As the name suggests, the SPRT looks at a sequence of residuals to determine if the time series of data is more likely from a nominal distribution or a pre-specified faulted distribution. As new observations are made, the SPRT compares the cumulative sum of the log-likelihood ratio:

$$s_i = s_{i-1} + \log \Lambda_i$$

to two thresholds, which depend on the acceptable Type I and Type II fault rates:

$$a \approx \log\left(\frac{\beta}{1-\alpha}\right)$$
$$b \approx \log\left(\frac{1-\beta}{\alpha}\right)$$

where α is the acceptable false alarm (false positive) rate and β is the acceptable missed alarm (false negative) rate. For this research, false alarm and missed alarm rates of 1% and 10% respectively are used. If s_i < a, then the null hypothesis is accepted; that is, the system is operating in a nominal condition. If s_i > b, then the alternative hypothesis is accepted; that is, the system is operating in a faulted condition. When a determination is made, the sum, s_i, is reset to zero and the test is restarted.

After a fault is detected in the system, the prognostic system can be engaged to determine the RUL for the system. As discussed, GPM prognostic methods use a measure of system degradation to estimate RUL. Monitoring system residuals, or combinations of residuals, are natural candidates for these prognostic parameters because they inherently give a measure of the deviation of a system from normal operation. The following chapter investigates the application of this combined monitoring/prognostic system and the previously developed prognostic parameter identification method to example data sets.

5 Applications and Results

This section presents the results of applying the described prognostic parameter identification methodology and the GPM/Bayes prognostic model to three data sets. First, the effect of noise in the prognostic parameter on GPM/Bayes model performance is investigated with simulated prognostic parameters. The second example application uses the data set given in the 2008 PHM Challenge Problem, which resulted from a turbofan engine simulator. Additional data from the same simulator, which is available at the NASA Prognostics Data Repository [135], was also used in this application. The second example investigates the milling equipment degradation data set, also obtained from the NASA repository [136].

5.1 The Effect of Noise on Model Performance

Prognostic models using the "good" prognostic parameters discussed in section 4.4 and plotted in Appendix A were developed to investigate the effect of noise on prognostic model performance. As shown in Table 2, the introduction of noise in the prognostic parameter can have a significant effect on parameter suitability metrics. It is shown here that the noisier parameters are useful for prognostics, but show degraded results over the less noisy parameters.

This application of the GPM/Bayes methodology investigated the three populations of "good" parameters given in Appendix A. The first population is noise free; the second is contaminated with Gaussian noise with zero mean and standard deviation equal to 20% of the mean parameter value; and the third set is contaminated with 80% mean value noise. Twenty example parameters are available for each population. For each parameter, a GPM/Bayes model is developed using the remaining 19 paths. This model is used to determine the RUL at 100 equally spaced intervals along the exemplar path; each observation is 1% of the total lifetime. Figure 20 gives the resulting RUL estimates for run 6. The results for each run are plotted in Appendix A. RUL estimates for each case begin at roughly 150 cycles. This is the average lifetime which is predicted by the Bayesian prior distributions. As more observations are collected, the regression coefficients shift from being highly dependent on the prior distributions to depending on the collected data from the actual system. As this happens, the RUL estimates become more appropriate for the specific system being modeled. As indicated in Figure 20 and the figures in

Appendix A, the number of observations needed to shift the model from the population-based prior information to the system-specific observations depends on the noise in the prognostic parameter. The noise-free parameter responds quickly to the collected data while the noisy parameters take longer to reach the correct solution. In addition, the three parameters tend to converge near the end of life. At this point, the regression model has learned the underlying functional form of the prognostic parameter and noise is less of an issue.

These results also highlight the effect of threshold values on RUL bias. For cases with a failure value that is less than the model threshold, the RUL is consistently overestimated, as in run 6. Cases with failure values greater than the model threshold, such as run 2, result in underestimated RUL. This highlights the need for a more sophisticated thresholding technique. The application of a single, static threshold to all runs will often result in a biased RUL estimation. Several solutions to this problem have been proposed, including the use of probabilistic thresholds as opposed to deterministic thresholds [137, 138]. However, a more accurate method may be to account for system-specific features when

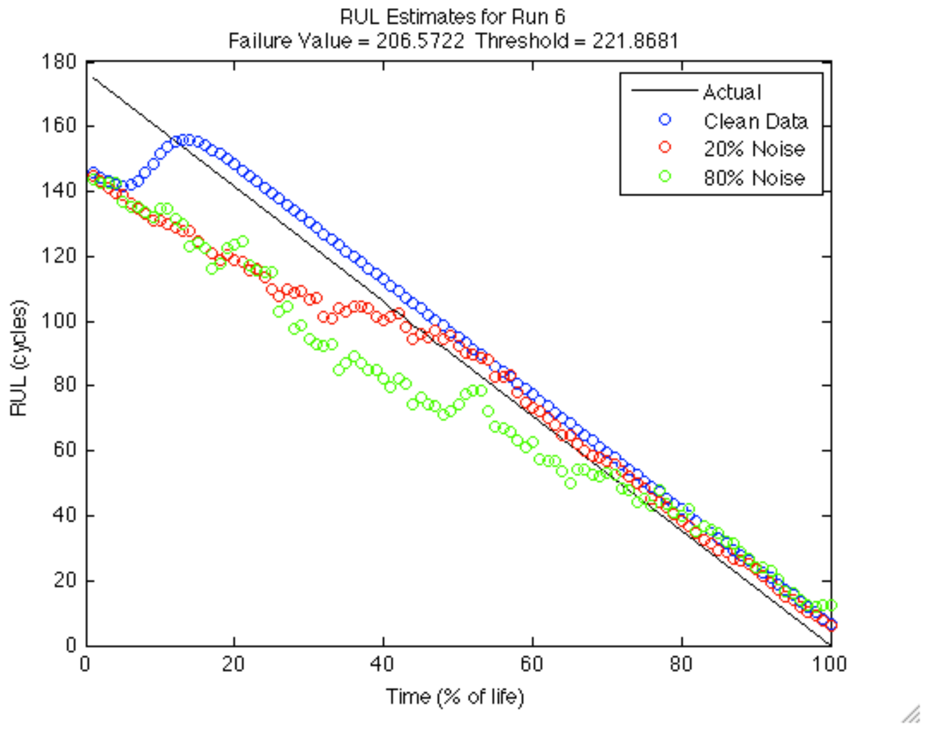


Figure 20: RUL Estimates Using Noisy Parameters

determining the failure threshold. A system-specific failure threshold may be informed by usage conditions or features of the data. Methods which do not apply the same threshold, deterministic or stochastic, have been suggested [61, 139]. Integration of a system-specific threshold with the GPM/Bayes model described here will afford more accurate RUL prediction for systems without a clearly defined failure threshold.

The next example application looks at the full monitoring/prognostic system to make RUL estimates. By monitoring the system measurements, degradation is characterized by monitoring system residuals. The proposed parameter identification method is applied to these residuals to identify an appropriate prognostic parameter and the GPM/Bayes model is applied to several identified parameters for prognostics.

5.2 PHM Challenge Data Description

The PHM Challenge data set consists of 218 cases of multivariate data that track from nominal operation through fault onset to system failure. Data were provided which modeled the damage propagation of aircraft gas turbine engines using the Commercial Modular Aero-Propulsion System Simulation (C-MAPSS). This engine simulator allows faults to be injected in any of the five rotating components and gives output responses for 58 sensed engine variables. The PHM Challenge data set included 21 of these 58 output variables as well as three operating condition indicators. Each simulated engine was given some initial level of wear which would be considered within normal limits, and faults were initiated at some random time during the simulation. Fault propagation was assumed to evolve in an exponential way based on common fault propagation models and the results seen in practice. Engine health was determined as the minimum health margin of the rotating equipment, where the health margin was a function of efficiency and flow for that particular component; when this health indicator reached zero, the simulated engine was considered failed. The interested reader is referred to [118] for a more complete description of the data simulation.

The data have three operational variables – altitude, Mach number, and TRA – and 21 sensor measurements. Initial data analysis resulted in the identification of six distinct operational settings; based on this result, the operating condition indicators were collapsed into one indicator which fully defined in which of the six modes the engine was operating. In addition, ten sensed variables were identified whose residuals changed in a meaningful way through time and were well correlated to each other. In this way, the 24 sensor data set was reduced to 11 variables, with original variable numbers: 1 (the operating condition indicator), 5, 6, 7, 12, 14, 17, 18, 20, 23, and 24. Because the nature of the prognostics challenge was to develop a prognostic algorithm with no knowledge of the system under test or the variables available, no effort is made to physically relate these eleven sensors. However, they are considered suitable for auto-associative modeling due to the strong inter-correlations within the group. The correlations between these eleven variables are shown in Figure 21.

The GPM method uses degradation information, either directly measured or inferred, to estimate the system RUL. Initial analysis of the raw data does not reveal any trendable degradation parameter. That is, no sensed measurement has an identifiable trend toward failure. Figure 22 is a plot of the eleven variables that were determined to statistically change with time. These variables were used to develop a monitoring and prognostics system.

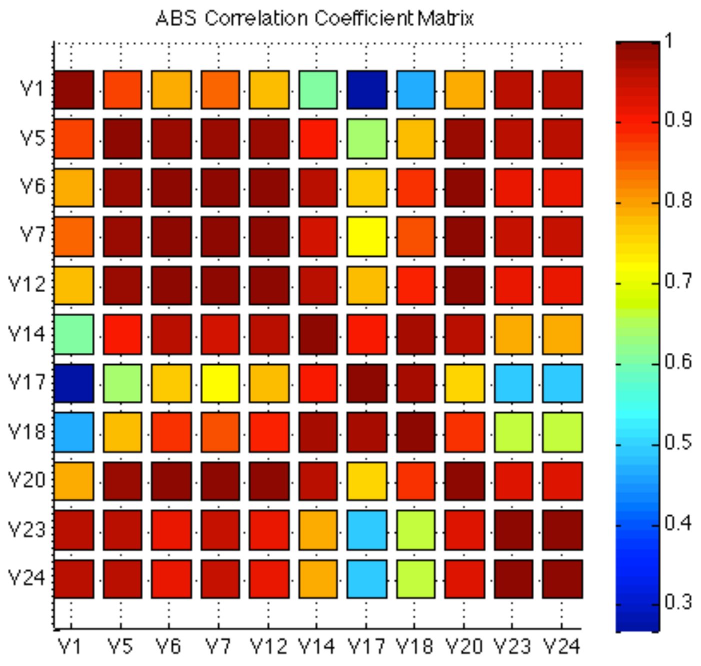


Figure 21: Correlation Coefficient Matrix for Eleven Monitored Variables

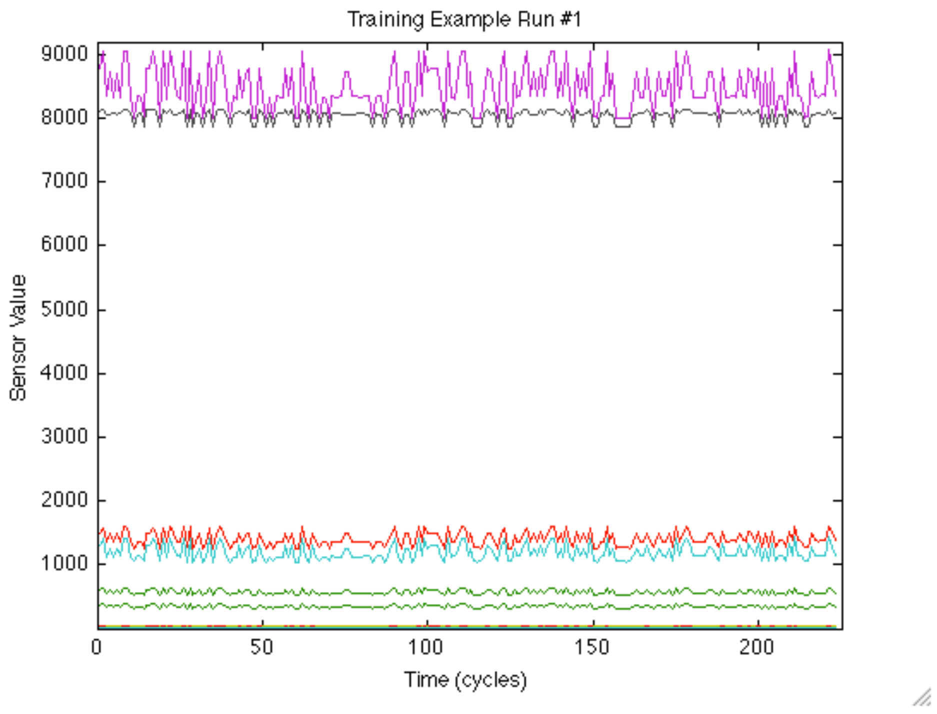
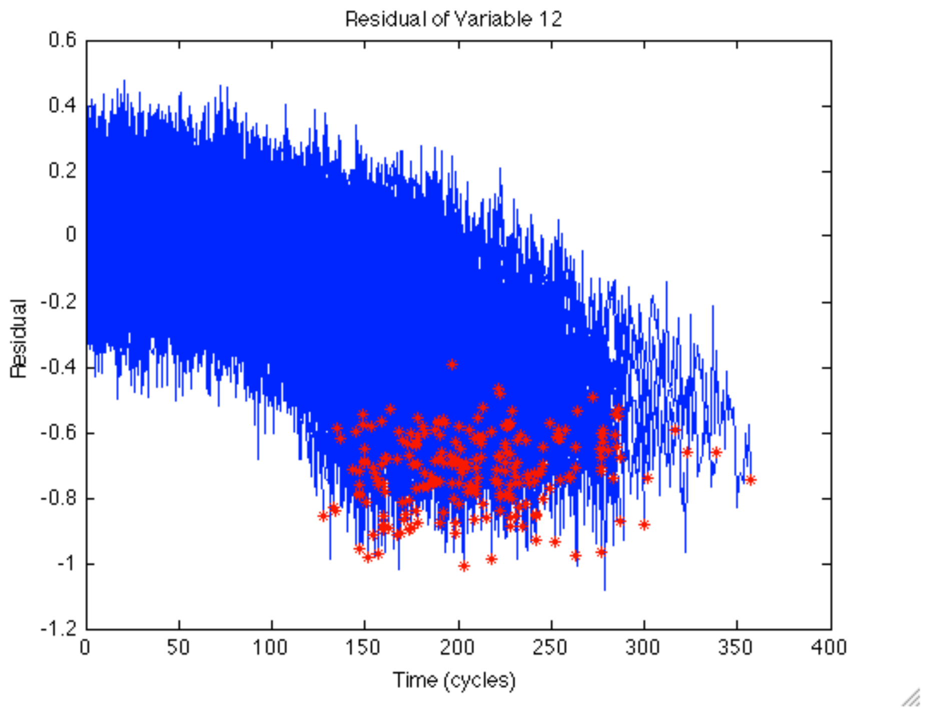
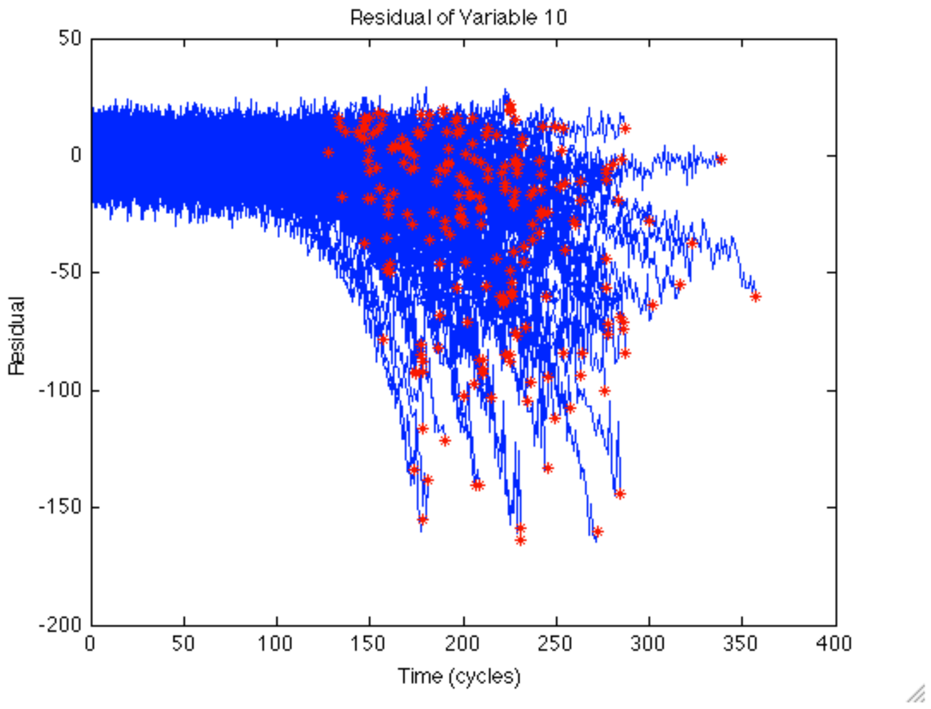


Figure 22: Eleven PHM Data Set Variables

The eleven variables are monitored with an auto associative kernel regression (AAKR) model. The residuals between the measured values and the AAKR "corrected" values are candidates for inclusion in the prognostic parameter. Two of these residuals are shown below in Figure 23. The residual shown on the left is expected to be useful for prognostic predictions, while the residual shown on the right is not expected to be useful because the population of residuals do not all have similar shapes or equal failure values. It may be possible to classify these residuals into several groups of similar residuals; this may be indicative of different failure modes, but this idea was not pursued for this work. All eleven residuals are plotted in Appendix A, for reference. For this simple application, only linear combinations of the residuals are considered for possible prognostic parameters. However, it is a straightforward extension of the method to include other features, such as the measured data or fault detection results, or to allow for higher order terms such as nonlinear combinations of several inputs, exponential terms, etc.



(a)



(b)

Figure 23: Residuals which are expected to be (a) useful and (b) un-useful for prognostics

Three competing prognostic parameters are identified from the monitoring system residuals, one identified through visual inspection and two identified using the proposed automated method. Each of these parameters is used to develop a basic prognostic model and make RUL estimations for the test cases. The following section presents the results of prognostic parameter identification and RUL estimation for both of the methods.

5.2.1 Results

Three competing prognostic parameters are identified. The first parameter is based on visual inspection and expert analysis. The second parameter is identified using the proposed automated identification method. The third parameter is also identified via the automated method, but makes use of the input selection method described to remove un-useful inputs and relieve the computational burden. The three resulting prognostic parameters and their respective prognostic models are given below. A GPM model with dynamic Bayesian updating is developed using each candidate parameter. The prognostic models are then tested on the 218 test cases given in the PHM ’08 challenge. Model performance is characterized using the MAPE of the known RULs for the 218 test cases.

5.2.1.1 Visual Inspection

Visual inspection of the residuals suggests that an appropriate parameter might be a weighted average of the residuals for variables 6, 7, 14, 18, and 20. All eleven model residuals are plotted in Appendix A. The five residuals are weighted by the inverse of their average range and summed to give the prognostic parameter identified through visual inspection. By scaling the residuals, the relative importance of each contributor to the prognostic parameter is equal. Because no engineering judgment can be made concerning the physical characteristics of the system, weighting each input equally is a reasonable approach. The inputs can be scaled with respect to several measures, including the standard deviation, the input range, the correlation to RUL, etc. By combining several similar residuals the spread in the failure value is reduced, as show in Figure 24. This is sometimes referred to as parameter bagging and is a common variance reduction technique. Table 3 gives the parameter suitability metrics for each of the residuals. The five residuals with total suitability (given by the sum of the three suitability metrics) greater than 1.5 were combined to create the final parameter, which has greater suitability metrics than any one constituent residual. Identification of this parameter involved several weeks of expert analysis of the available data.

A second order polynomial model can been used to model the degradation parameter. While an exponential model may be more physically appropriate, and was certainly found to be after the data simulation method was made public, the quadratic model is more robust to noise and better describes the data fit for the chosen prognostic parameter. For the methodology proposed, the model must be linear in parameters; however, simple exponential models, such as *y=exp(ax+b)* parameterized as *ln(y) = ax +b*, cannot be used with negative y-values, because the natural logarithm of a negative number is undefined in the real number system. This adds unnecessary complexity to the modeling method.

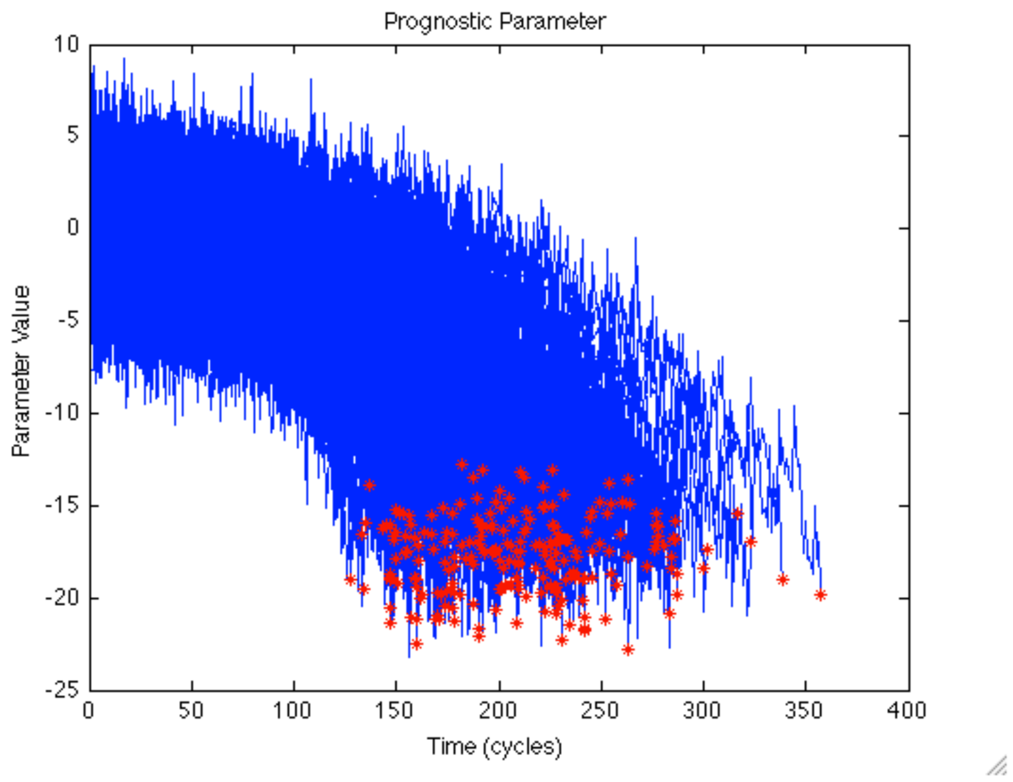


Figure 24: Prognostic Parameter Identified by Visual Inspection

Table 3: Prognostic Parameter Suitability Metrics

Variable	Monotonicity	Prognosability	Trendability	Suitability
3	0.435	0.370	0.000	0.805
5	0.537	0.613	0.018	1.168
6	0.604	0.727	0.291	1.622
7	0.749	0.818	0.726	2.293
12	0.654	0.314	0.001	0.968
14	0.782	0.851	0.814	2.447
17	0.639	0.282	0.000	0.921
18	0.703	0.731	0.649	2.083
20	0.625	0.737	0.476	1.838
23	0.447	0.497	0.000	0.945
24	0.457	0.510	0.000	0.967
VI Parameter	0.846	0.891	0.903	2.640

Quadratic equations, on the other hand, are naturally linear in parameters and can be used without significant concern for the effects of noise on the model fit. Shifting the prognostic parameter to the positive quadrant eliminates the problem of taking the logarithm of negative values; however, the quadratic fit results in a lower fitting error than the exponential fit, with mean squared errors of 1.53 and 2.33 respectively. Because of its robustness to noise and reduced modeling error, the quadratic fit is chosen for this research.

Figure 25 gives an example of a polynomial fit of the prognostic parameter with the time the model crosses the critical failure threshold indicated. The threshold of -13.9 was chosen as the upper 95% level of the distribution of failure values for the known failed cases. This gives an estimated system reliability of 95% which is a conservative estimate of failure time and reduces the possibility of overestimating RUL and having a failure. The time between the last sample and that star is the estimate of RUL, as indicated by the blue area. For this case, the estimated RUL is exactly correct, with an estimated remaining life of 36 cycles.

This prognostic parameter was used to develop a GPM prognostic model with Bayesian updating. The model was developed using the prognostic parameter resulting from 218 training cases which ran from beginning of life to failure, and was tested with 218 test cases which ran from beginning

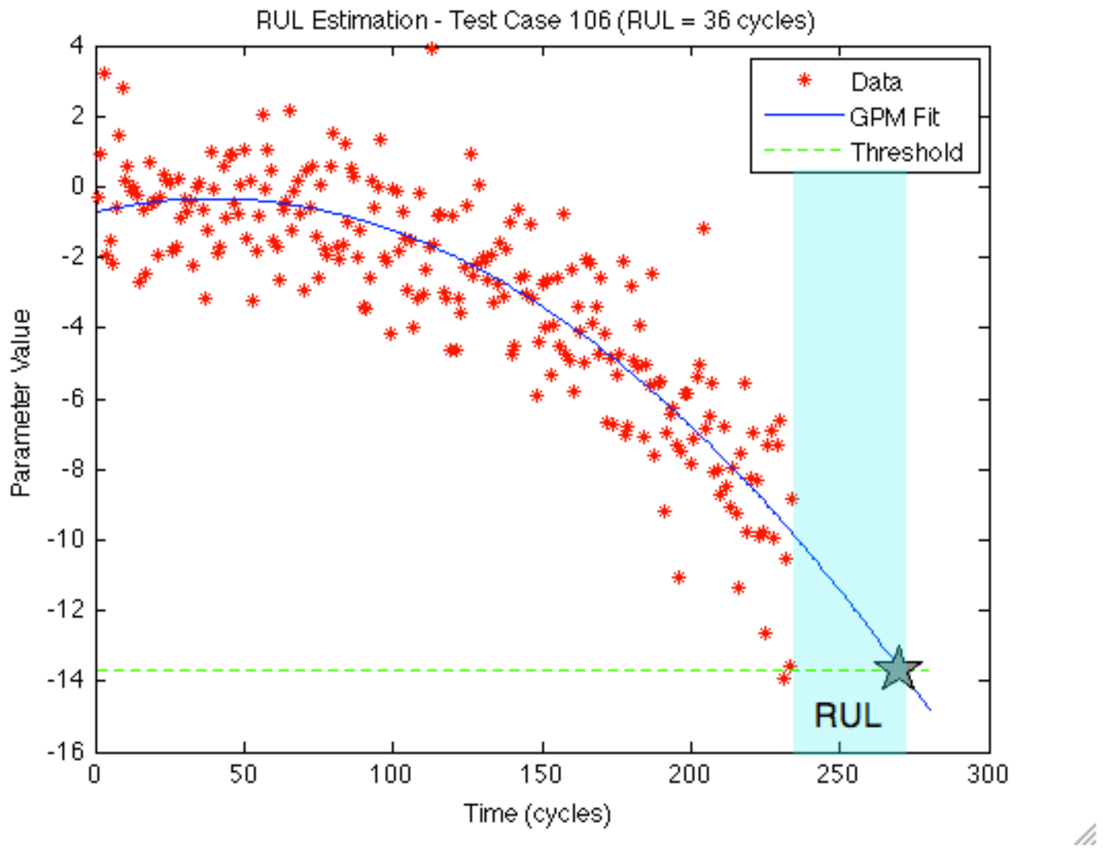


Figure 25: Prognostic parameter trending and RUL estimation

of life to some point after a failure-inducing fault occurred but before actual failure. The result of each of these cases is shown below, in Figure 26. The GPM model developed with this prognostic parameter resulted in a MAPE of 22.8%. While this result is larger than would be accepted in practice, it is important to note that in approximately half of the 218 test cases the RUL is predicted to within 10% accuracy. The large MAPE is due to poor performance on a few cases, which skews the results.

To illustrate the utility of the Bayesian updating method, a GPM model with no updating was also employed. The advantage of including prior information via dynamic Bayesian updating is to improve RUL estimates when very few observations are available, the data are very noisy, or both. A comparison of the performance through time of the straight GPM and the GPM with Bayesian updating is given in Figure 27. In this experiment, the two methodologies were applied to each of the training

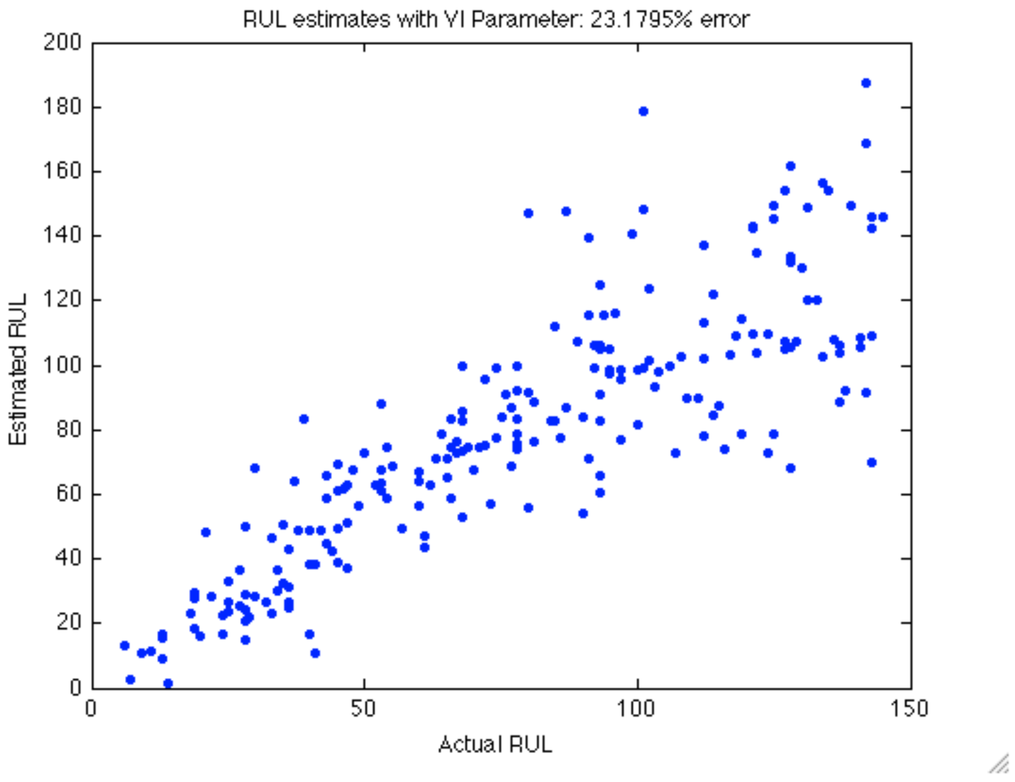


Figure 26: Visual Inspection RUL Estimate Results

cases using only a fraction of the full lifetime. The models were applied to subsets of each lifetime in 5% increments, i.e. the models were run using 5% of the full lifetime, 10%, 15%, etc. The RUL error at each percentage was calculated across the 218 full training cases to determine how the error decreases as more data become available. As was seen in the example case, the non-Bayesian method may result in an undeterminable RUL. In fact, for the data used here, nearly half the runs resulted in an indeterminate RUL estimate using the GPM methodology without Bayesian updating for runs using less than half the total lifetime. For these cases, the RUL is estimated using a Type I, or traditional reliability-based, method in order to give an estimate of RUL prediction error. The mean residual life is found at each time using a Weibull fit of the failure times and the current lifetime. Mean Residual Life (MRL) is found by:

$$MRL(t) = \frac{1}{R(t)} \int_t^{\infty} R(s)ds$$

where R(t) is the reliability function at time t. In practice, the prognostic method would likely fall back to a more rudimentary method such as this if the Type III model did not produce a reasonable answer.

The GPM/Type I model which does not include prior information gives an average error of approximately 55% when only 5% of the full lifetime is available and relies on the Type I method for approximately half of the cases. Conversely, the GPM/Bayes method gives approximately 25% error and is able to predict an RUL for every case. As Figure 27 shows, the average error of both methods decreases as more data becomes available and eventually converges to approximately equal error values when the available data overpowers the prior information.

5.2.1.2 GA Optimized Parameters

Two prognostic parameters were identified using the automated method described with a GA optimization. The GA was used to optimize the weighting coefficients in a weighted sum of the eleven monitoring system residuals. For this application, the fitness function was given by a straight sum of the

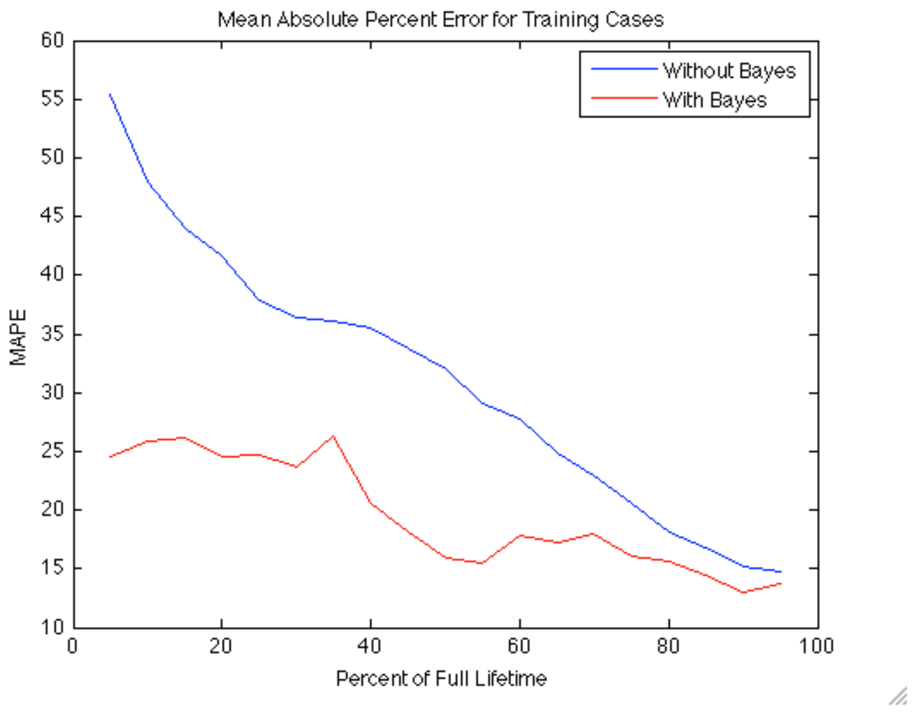


Figure 27: GPM Results With and Without Bayesian Updating

three suitability metrics:

$$fitness = monotonicity + prognosability + trendability$$

This gives equal weight to each of the three parameter suitability measures, but does not consider anything else. While the visual inspection parameter involved several weeks of expert analysis, the GA optimization involved only a few minutes of worker time to set up and approximately 1.68 hours of unsupervised computer runtime. While the time needed for the GA optimization to run will scale with the number of possible inputs, it involves mainly computer runtime and is only a fraction of the time needed for parameter identification through expert opinion. A second GA optimization was run using the input selection method described previously with a suitability cutoff of 1.5 for inclusion in the genetic algorithm. This optimization considered only five inputs and ran in 0.93 hours, nearly half the time the full optimization needed. The parameter identified by the first GA optimization is given in Figure 29, and the second in Figure 29.

Table 4 gives a comparison of the weights for the parameter identified by visual inspection and the two selected by the GA. The parameter suitability metrics for each parameter are given in Table 5; the suitability of each parameter is calculated as the sum of the three suitability metrics, as it was determined in the fitness function. The fitness of both of the GA-optimized parameters is basically equivalent to that of the parameter identified via visual inspection. This may be further improved by standard GA improvement techniques, such as coupling the result with a gradient descent optimization or running the GA several times to find the best result.

These GA-optimized prognostic parameters were also used to develop GPM prognostic models. Figure 30 gives the results for the prognostic model developed with the first parameter population. Figure 31 gives the results for the second parameter. Overall model performance with all three parameters is comparable, and it is nonsensical to claim one parameter as best based on the model performance. However, this application has shown that the proposed automated parameter identification method can identify prognostic parameters comparable to those identified by visual inspection and expert analysis in a fraction of the time.

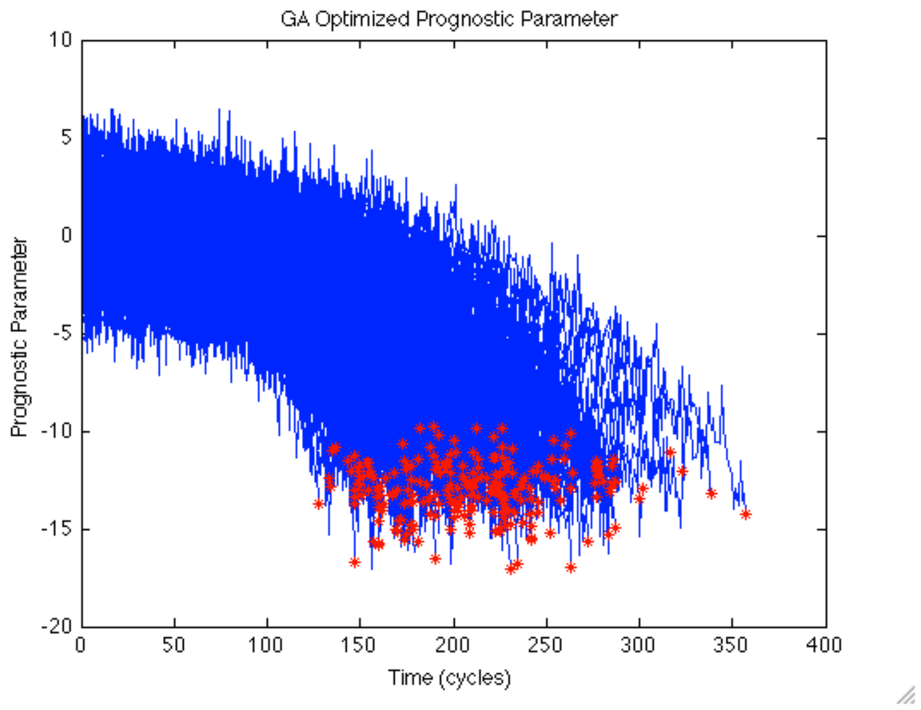


Figure 28: Prognostic Parameter Identified by GA Optimization

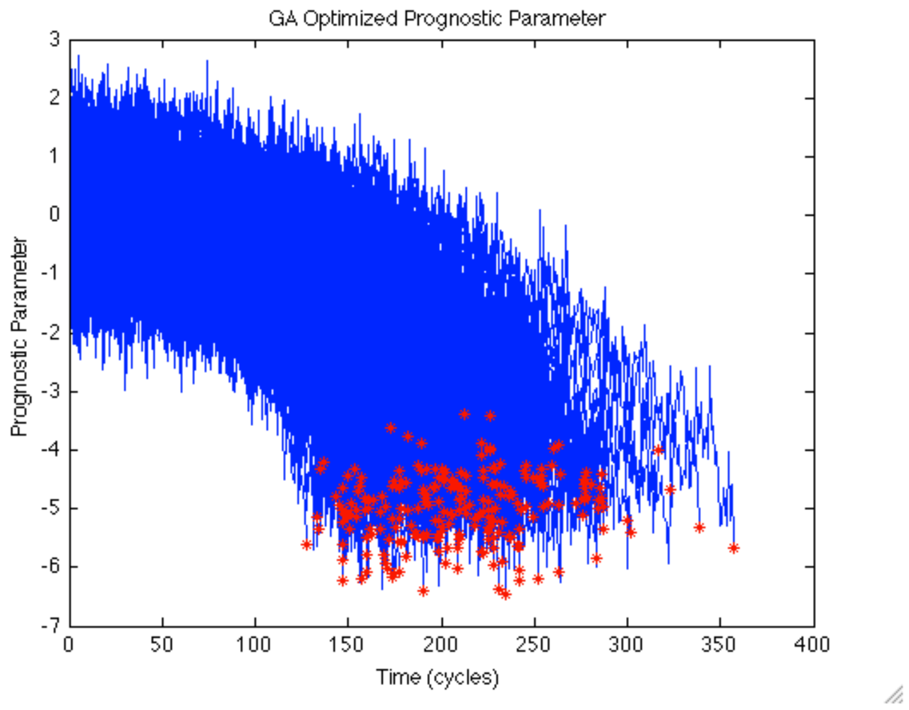


Figure 29: Prognostic Parameter Identified by Reduced GA Optimization

The sample application presented here benefited from the extensive expert analysis needed to identify a parameter through visual inspection, in that a subset of possible parameter inputs had already been identified and parameter optimization was restricted to only consider the monitoring system residuals. In applications with a larger domain of inputs, which could include the actual signals, the monitoring system predictions, information on usage, environment, and load, and fault alarm and diagnostic results, as well as higher order terms of any of these inputs, the input selection technique applied in identifying the second GA parameter will be paramount to timely and accurate parameter identification. It will greatly reduce GA runtime and help ensure that a near optimal parameter is identified.

The prognostic parameters identified here were used to develop a Type III model for prognostic estimation. However, this data set includes both sensed measurements and operating condition measurements. Type I and II models may also be applicable to this type of data. The following section looks at several competing prognostic models of each type and compares their performance on this data set to illustrate the efficacy of the individual-based algorithm.

Table 4: Residual Weightings

	Residual										
Parameter	1	3	4	5	10	12	15	16	18	21	22
VI Parameter	0.00	0.00	0.19	0.14	0.00	4.76	0.00	28.5	0.74	0.00	0.00
GA Parameter 1	-0.28	0.81	0.07	0.14	0.00	4.93	0.02	29.1	0.35	0.32	0.10
GA Parameter 2	0.00	0.00	0.02	0.07	0.00	2.38	0.00	5.92	0.25	0.00	0.00

Table 5: PHM Challenge Data Parameter Suitability Metrics

Parameter	Monotonicity	Prognosability	Trendability	Suitability
VI Parameter	0.846	0.891	0.903	2.640
GA Parameter #1	0.941	0.895	0.931	2.766
GA Parameter #2	0.901	0.889	0.909	2.698

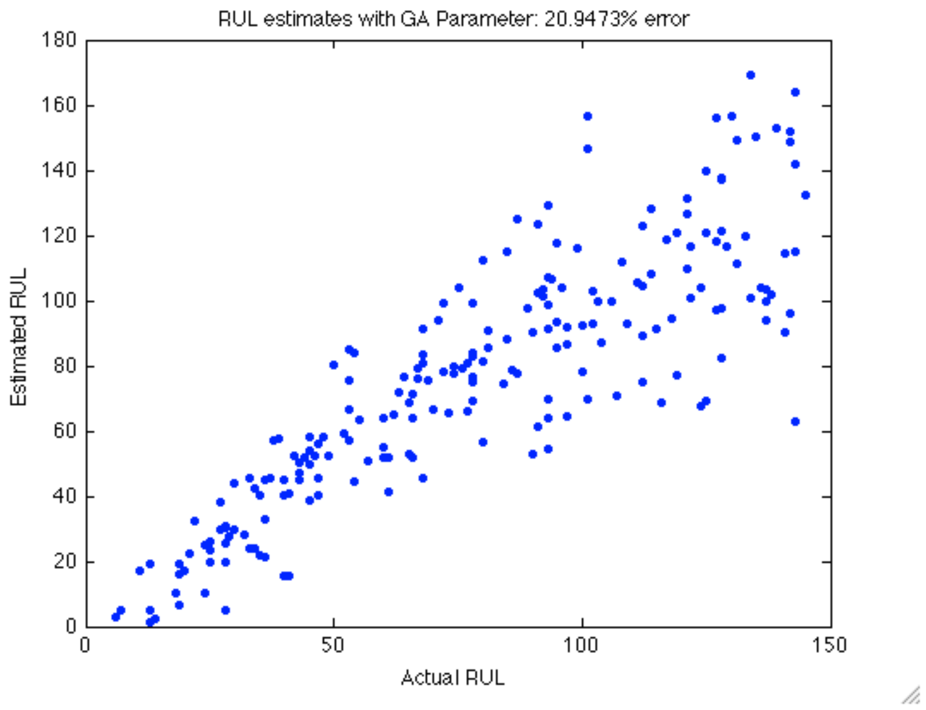


Figure 30: RUL Estimates for First GA-optimized Parameter

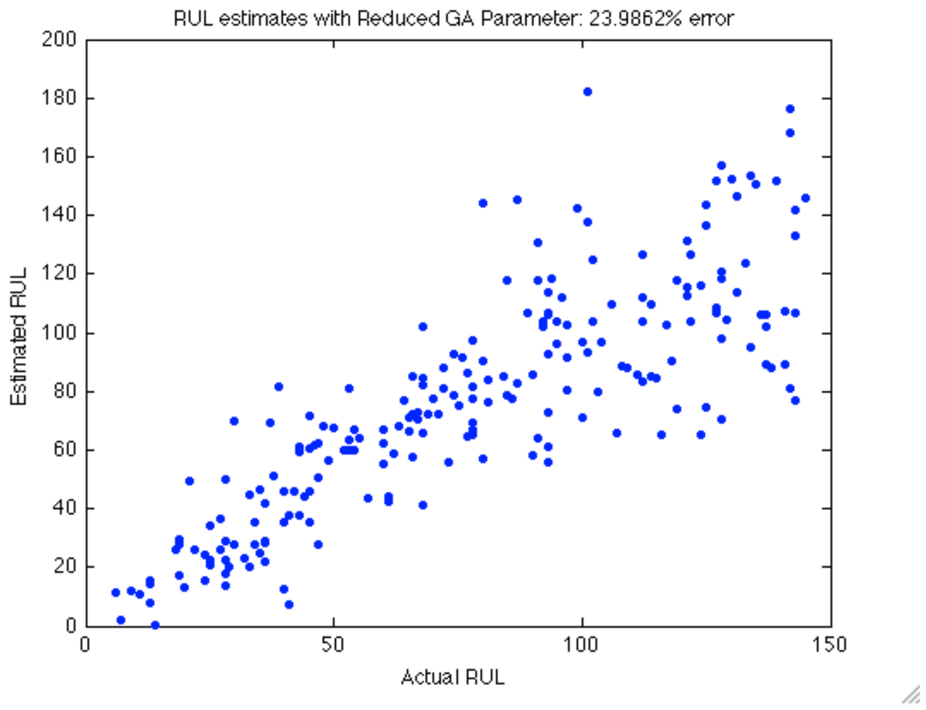


Figure 31: RUL Estimates for Second GA-optimized Parameter

5.2.2 Performance Comparison of the Three Prognostic Types

Three competing prognostic models were developed, one of each prognostic model class, to illustrate the performance improvements offered by including additional information sources. The three models are compared based on the "score" given by the challenge scoring function:

$$\begin{aligned} &d = RUL_{estimated} - RUL_{actual} \\ &score(d < 0) = \exp(-d/13) - 1 \\ &score(d > 0) = \exp(d/10) - 1 \end{aligned}$$

The Type I prognostic model utilizes a Weibull distribution-based reliability analysis. Two Type II models were developed to estimate RUL based on the operating history. The first uses a simple regression model considering the time spent in each operating condition; the second, a Markov Chain model as described earlier. Finally, a GPM model using the GA-optimized prognostic parameter given in Figure 28 gives the Type III estimates.

5.2.2.1 Type I Prognostics

Type I prognostics are traditional time-to-failure analysis methods, such as Weibull analysis. A histogram of the failure times for the 218 training cases is given in Figure 32 along with the Weibull probability density function. The maximum likelihood Weibull fit of this data gives parameter estimates of 228.5 and 5.1 for the scale parameter and shape parameter, respectively. The scale parameter indicates that 63.2% of the observed systems can be expected to fail by cycle 228.5. Since the shape parameter is much greater than 1, failure is occurring due to wear-out degradation. For each new system, the RUL is estimated using the MRL calculation previously discussed. Because the failure data has a large variance, accurate RUL prediction is not possible. From the figure, one can see that the most probably value is around 210 cycles; however, the values have a range that covers approximately two hundred cycles. Using this Weibull fit and the MRL calculation to estimate the RUL for each of the 218 test cases resulted in a score of approximately 22,500.

5.2.2.2 Type II Prognostics

Type II methods use operational data, such as load, environment, input current, etc, to make predictions of RUL. As discussed in section 5.1, the data can easily be divided into six operational

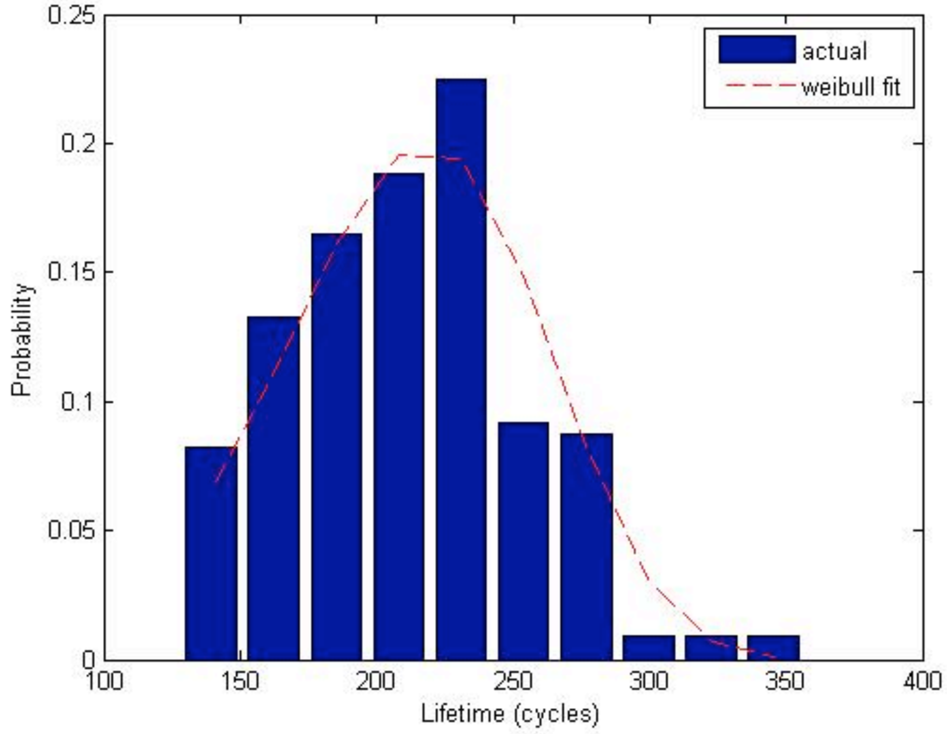


Figure 32: Histogram of failure times and Weibull fit

conditions; the percentage of full lifetime spent in each condition is shown in Figure 33. This figure shows that the variation in the percent of time spent in each condition is small (~10%) , but the variation in failure times is much larger (~20%). Therefore, it is improbable that the variation in time spent in each operational condition is the source for the variation of failure time.

Several methods can be used to estimate the RUL from this usage data. Simplest, perhaps, is a regression using the total amount of time spent in each condition to predict RUL (the percent of time spent in each condition may also be used). Regressions of this type have been performed using a simple linear regression, inferential kernel regression, and neural networks. The results for all regressions have been poor, with little improvement over Type I estimates. This is most likely because the amount of time spent in any one operating condition is not well correlated to the total lifetime, as shown in Table 6. The cost function for an inferential kernel regression was approximately 20,600.

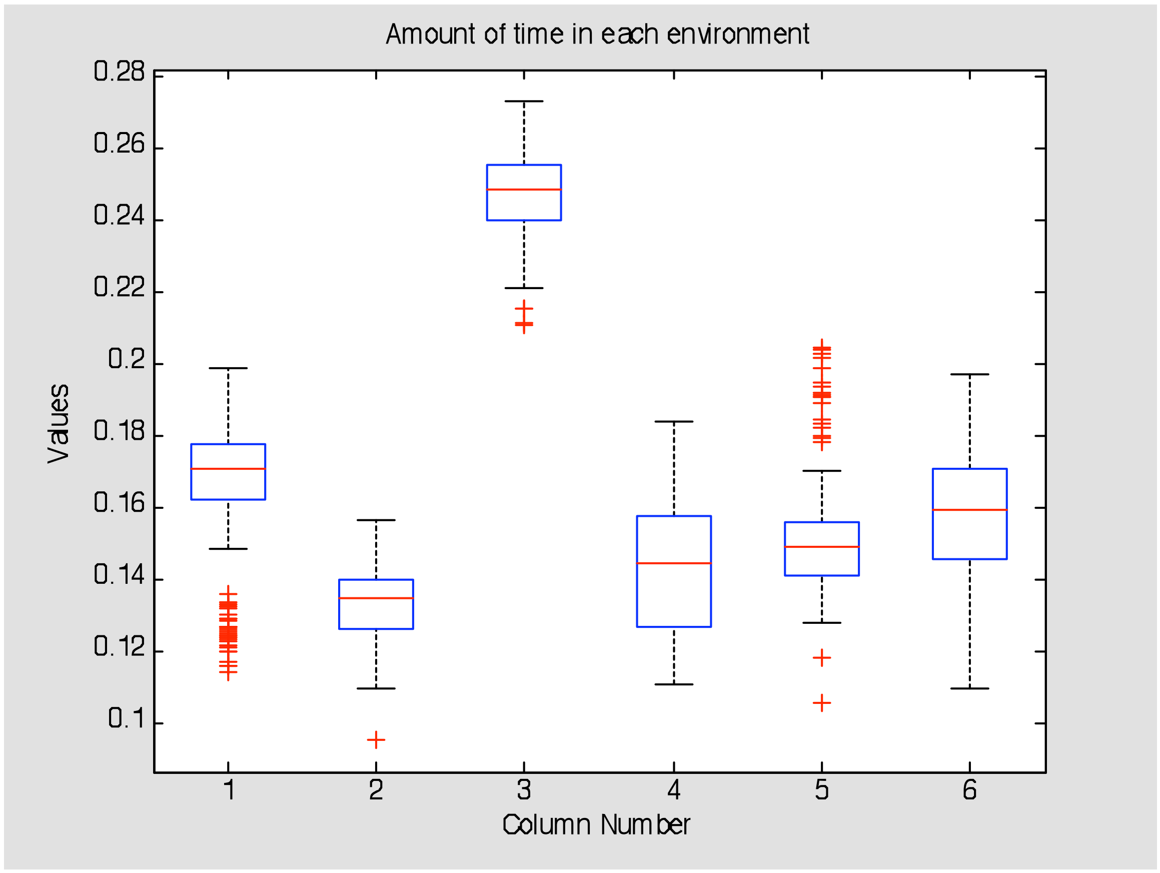


Figure 33: Plot of time spent in each operational condition

Table 6: Correlation of Operating Condition to Lifetime

Correlation to Lifetime	
1	0.12
2	-0.037
3	0.041
4	0.021
5	-0.064
6	-0.046

In addition to simple regression approaches, a Markov Chain (MC) model was developed to simulate possible future operating condition progressions. The transition probably matrix calculated from the training cases is given in Table 7. These operating condition progressions are used to determine a distribution of failure times for the system. However, as with the inferential regression, it was difficult to map the simulated operating condition information to an RUL estimate or degradation parameter to determine end of life for each simulated path. The results of the MC study were also poor, with a score of approximately 19,200.

5.2.2.3 Type III Prognostics

A GPM/Bayes model as described in Chapter 4 was developed using the GA-optimized prognostic parameter given in Figure 28. For the current problem, quadratic regression models were fit to the full degradation parameter for each of the 218 training cases. The resulting prior distributions for the parameter fits are given in Table 8. Very precise estimates of p1 and p2 are available. More variance is seen in p3, which is assumed to correspond to the random level of initial degradation and is retained for that reason. The variance of the degradation parameter can be estimated from the training examples by smoothing each example path and subtracting the smoothed path from the actual path. This gives an estimate of the noise; the noise variance can be estimated directly as the variance of this data set. For this problem, the noise variance in the degradation parameter is estimated to be 0.0588. The cost for the test set is much improved over the previous two methods at approximately 2,500.

Table 7: Markov Chain Transition Probability Matrix

To / From	1	2	3	4	5	6
1	0.14	0.16	0.15	0.16	0.15	0.24
2	0.16	0.15	0.14	0.14	0.15	0.25
3	0.15	0.14	0.15	0.15	0.15	0.26
4	0.16	0.15	0.15	0.15	0.15	0.24
5	0.15	0.15	0.15	0.15	0.15	0.26
6	0.15	0.14	0.15	0.15	0.15	0.26

Table 8: Prior Distribution for Quadratic Parameters

	Mean	Std Dev
p_1	-0.0001	4.30E-05
P_2	0.0075	0.0028
p_3	-0.2057	0.370

The next example application utilizes a data set collected in a laboratory testing experiment. This example will further investigate the efficacy of the parameter identification method and input selection technique.

5.3 Milling Data Set

The second data set used for this method is the Milling Data set collected by Agogino and Goebel, also available at the NASA Prognostic Data Repository [136]. This data set presents milling machine wear measurements made in a laboratory experiment. Figure 34 shows the measured flank wear for each of 15 runs; the original data set includes 16 runs, but Run 6 includes only one set of measurements which do not indicate failure so it is excluded from this analysis. The experimental conditions for each run are given in Table 9. For this application, failure was assumed to occur at flank wear of 0.45 mm, as indicated. Failure times given in Table 9 are interpolated from the available flank wear measurements, assuming that the degradation is linear between measurements. In real-world applications, flank wear measurements would not be available. A prognostic parameter for this system should be able to characterize the milling machine degradation from the available information sources: operating condition indicators, including depth of cut, feed, and material; and available sensor measurements, including AC spindle motor current, DC spindle motor current, table vibration, spindle vibration, table acoustic emissions, and spindle acoustic emissions, or features of these measures.

Figure 35 shows the raw data for run 3. Because each of the sensors measure highly oscillatory variables, this data is not well suited to the AAKR monitoring system described previously. However, there are underlying trends in the data which can be used to monitor the system. Of particular interest here are the mean and standard deviation of the signals. By monitoring these features of the measured

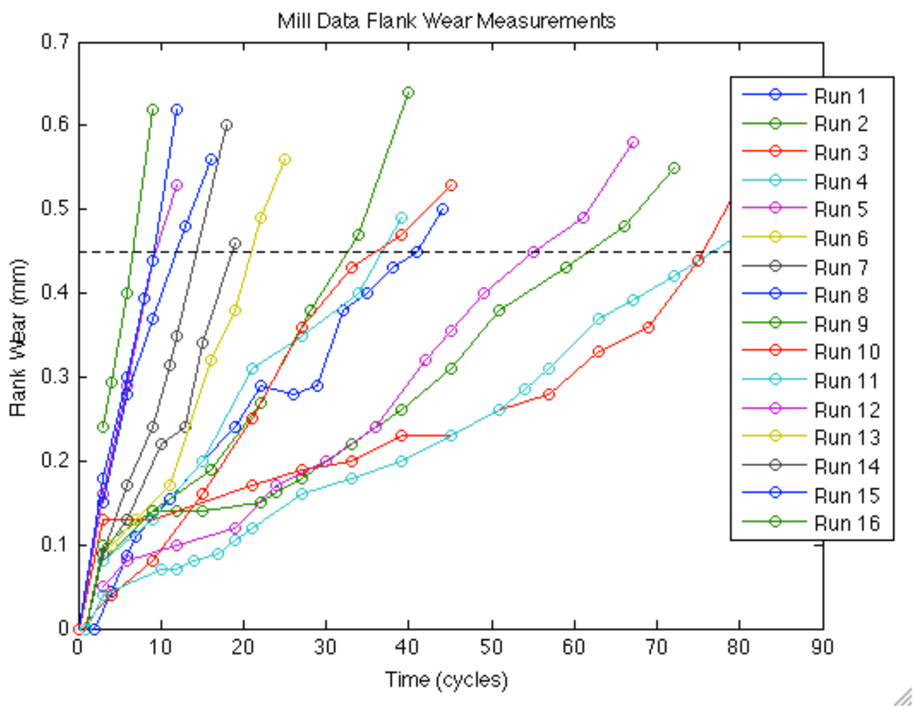


Figure 34: Flank Wear Measurements

Table 9: Milling Data Experimental Conditions

Run	Depth of Cut	Feed	Material	Failure Time
1	1.5	0.5	cast iron	41.00
2	0.75	0.5	cast iron	61.80
3	0.75	0.25	cast iron	75.55
4	1.5	0.25	cast iron	36.78
5	1.5	0.5	steel	9.33
6	1.5	0.25	steel	N/A
7	0.75	0.25	steel	18.67
8	0.75	0.5	steel	9.17
9	1.5	0.5	cast iron	32.67
10	1.5	0.25	cast iron	36.00
11	0.75	0.25	cast iron	76.80
12	0.75	0.5	cast iron	55.00
13	0.75	0.25	steel	20.91
14	0.75	0.5	steel	14.40
15	1.5	0.25	steel	11.91
16	1.5	0.5	steel	6.68

variables, changes in the system can be characterized and trended to failure. The means and standard deviations of the six signals for run 3 are shown in Figure 36. Signal features are calculated for each timestep reported. These steps are not equally spaced in time; however, no information about sampling rate is given to allow for different time averaging. Each time step includes 9000 observations, but time steps span different durations. Because it is not precisely known how these measurements are made, this is considered the best option for this data set. Plots of the mean and standard deviation for all fifteen runs are given in Appendix A.

5.3.1 GA Optimized Parameter

The parameter identification method was applied to these twelve features to identify an optimal prognostic parameter for the system. The parameter identified for the fifteen runs is shown in Figure 1. This parameter has monotonicity, prognosability, and trendability metrics of 0.889, 0.807, and 0.829, respectively. The parameter is used to develop a GPM/Bayes model for the entire population of milling experiments. Because only fifteen example runs are available, the performance of the model is measured through a leave-one-out method. That is, for each of the fifteen cases, the remaining fourteen cases are used with the prognostic parameter to develop a GPM/Bayes model. RUL estimates are made for the left out case using this model. In each case, the RUL is estimated at each observation. Figure 38 gives the results of the prognostic model for Run 3 using the GA optimized parameter. The model on a whole performed well, with an MAPE of 21.5% over the total lifetime. As a component approaches failure, the error decreases to an average of 10.2% or better within the last three cycles before failure.

Because the identification of appropriate prognostic parameters is automated, it is straightforward to develop additional prognostic parameters beyond the single parameter for each system. This data includes three operating conditions: depth of cut, feed, and material. A single mill run involves only one set of operating conditions for each cut. Instead of applying one prognostic model to the entire population, it may be more effective to classify runs according to one or more operating condition and develop multiple prognostic models. The functions available in the PEP toolbox make this extra step trivial for prognostic parameter optimization. The next section presents the results of developing separate prognostic models for each of the two materials being milled: cast iron and steel.

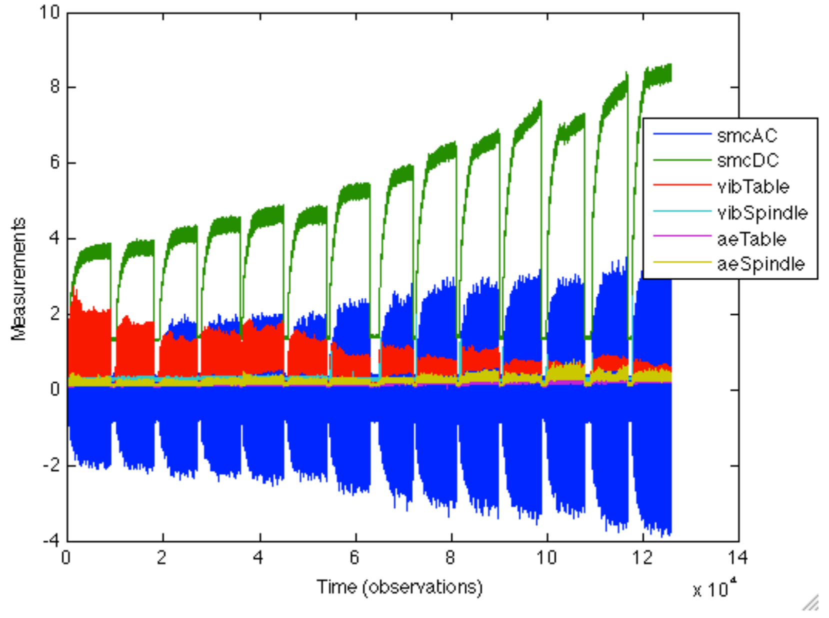


Figure 35: Milling Data, Run 3

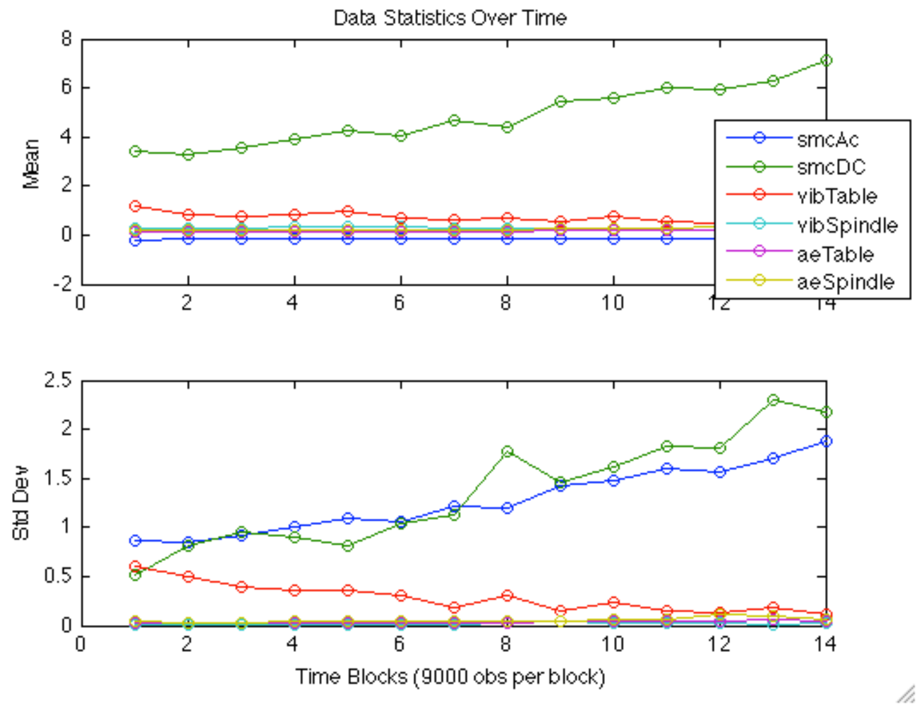


Figure 36: Mean and Standard Deviation of Milling Data, Run 3

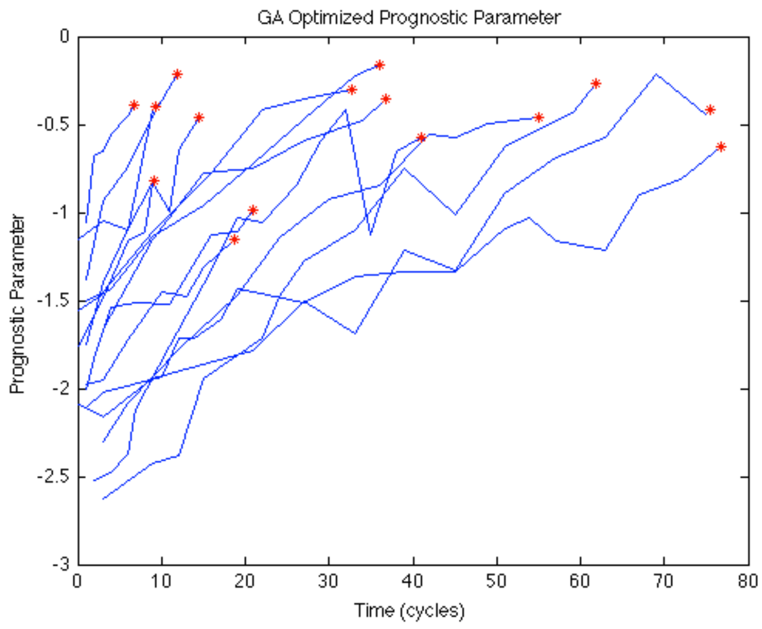


Figure 37: GA-Optimized Prognostic Parameter for Milling Data

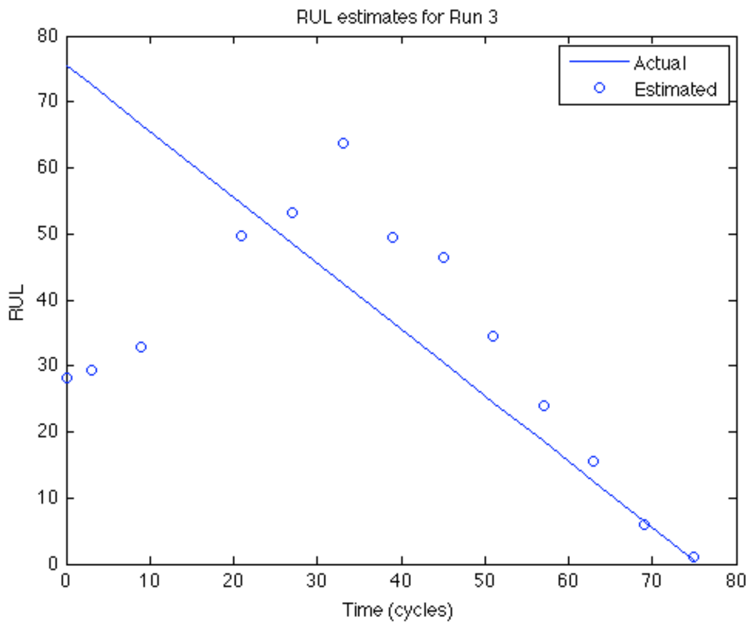


Figure 38: RUL estimates for Milling Data Run 3

5.3.2 Material Specific Parameters

Two additional prognostic parameters are identified for the milling data based on the type of material being milled. As indicated in Table 9, the milling experiments using steel have a much shorter mean lifetime, only 13.0 cycles, compared to those using cast iron, which have a mean life of 51.9 cycles. This indicates that it should be beneficial to separate these cases into two separate models. Three operating conditions total are available; prognostic model performance may be further improved by dividing the data according to two or all of these parameters. However, because only fifteen runs are available, it is not possible with the available data set. Should more data become available, this is an interesting course to pursue.

The optimized parameter for cast iron is shown in Figure 39, and that for steel in Figure 40. The prognostic parameter suitability metrics and total suitability, given by the sum of the three metrics, for these two parameters and the single parameter given earlier are presented in Table 10. By separating the runs according to material, the prognostic parameter suitability increased slightly for both cases. The resulting RUL estimations using these improved parameters showed improvement over the single population parameter. Using the same leave-one-out methodology, each of the fifteen runs were prognosed using the appropriate model. Separating the data according to material improved prognostic performance by nearly 50%, giving an average RUL error of 12.3%, with a reduced error of 8.2% or less within three cycles of failure. Figure 41 compares the performance of the two models. This figure shows that the material-specific models have better performance than the lumped model, which should be expected from the improved parameter suitability metrics.

Using the automated parameter identification method for this data allowed for easily identifying separate prognostic parameters for data in two groups: steel and cast iron. Allowing the computer to do all the computationally heavy work allows a prognostic system designer to investigate many possible competing parameters and choose among them. The PEP toolbox has the ability to identify prognostic parameters for any number of data groups for easy comparison. For this example application, dividing the cases by the material being bored afforded some improvement in the RUL estimation, particularly at end of life when such estimation is most important.

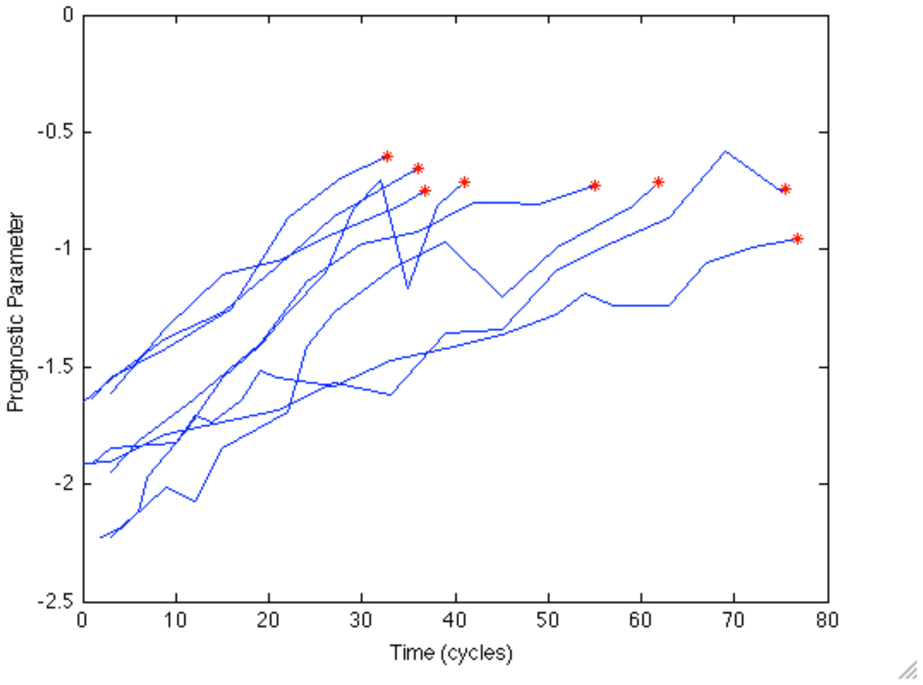


Figure 39: GA-Optimized Prognostic Parameter for Runs Boring Iron

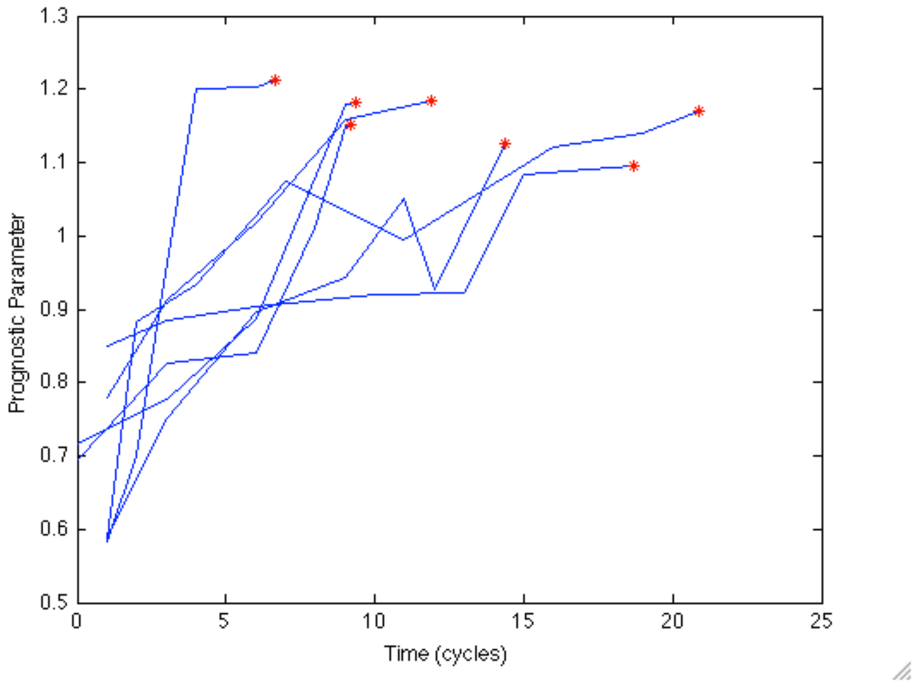


Figure 40: GA-Optimized Prognostic Parameter for Runs Boring Steel

Table 10: Parameter Suitability Metrics for Milling Data

	Monotonicity	Prognosability	Trendability	Suitability
Full Parameter	0.889	0.807	0.829	2.526
Iron Parameter	0.969	0.917	0.898	2.783
Steel Parameter	0.929	0.947	0.768	2.643

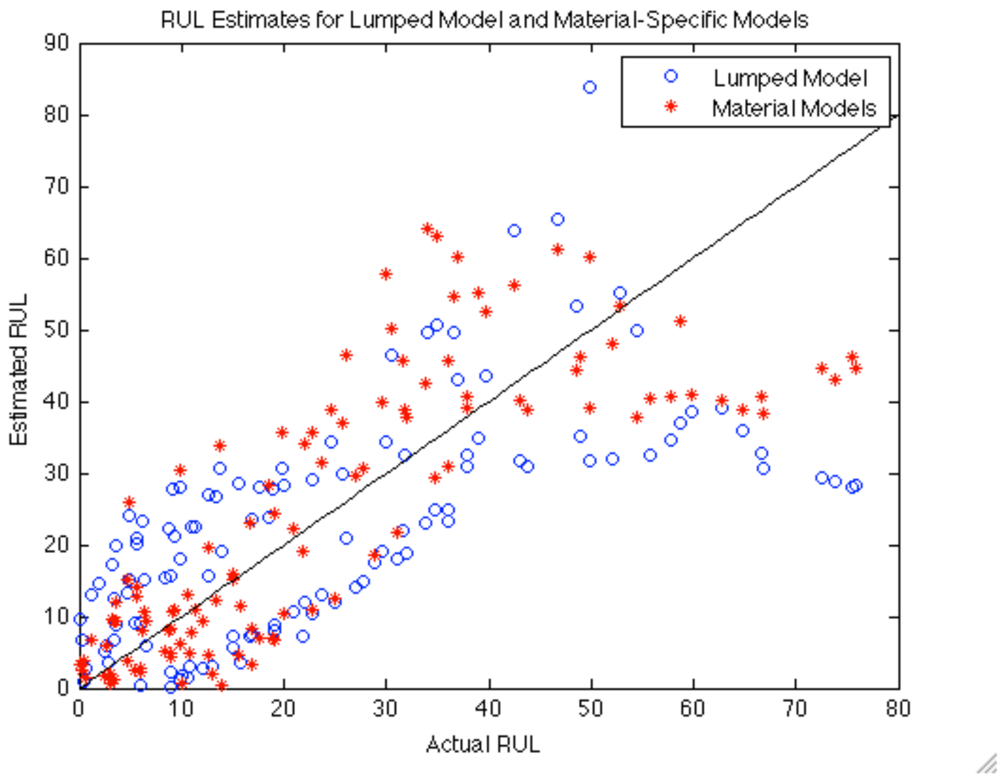


Figure 41: RUL Estimates for Lumped and Material-Specific Models

6 Conclusions

Estimation of remaining useful life is a burgeoning field in the reliability and maintenance community. Prognostics is a key component in a full health monitoring system, which typically includes system monitoring, fault detection and diagnostics, failure prognostics, and operations planning. While system monitoring, fault detection, and diagnostics are well-established fields, prognostics is still in its infancy. Despite prognostics' relative youth as a research area, much attention has been given in the last decade to the development of algorithms for predicting remaining useful life (RUL). These algorithms can be classified into three categories based on the type of information they use to make RUL predictions. Type I prognostics use traditional time-to-failure data to estimate the lifetime of an average component operating in historically average conditions. Type II prognostics, or stressor-based, utilize environmental and operating conditions to characterize the lifetime of an average component operating in a specific environment. Type III prognostics, or degradation-based, are the only truly individual prognostics algorithms. These methods use measures of system health to estimate the lifetime of a specific component operating in its specific environment. As nuclear power plants approach the end of their original design life, individual-based prognosis of the equipment health is paramount to obtaining license extensions. Accurate prognostics can indicate which components need to be replaced for continued safe operation and which are still operating within specifications. The most common Type III algorithm, and the one employed in this research, is the General Path Model (GPM), which tracks a measure of system health called a prognostic parameter. The prognostic parameter may be a direct measurement of system health, such as tire tread depth, or it may be inferred from other measurements made on the system. A parametric model is fit to the prognostic parameter and extrapolated to a pre-defined critical failure level. The RUL is estimated as the difference between the current time and the time at which the extrapolated model crosses the failure threshold. This research proposed a modification to the traditional GPM method which uses Bayesian updating methods to incorporate prior information about the expected prognostic parameter path. Incorporating prior information allows the GPM to be applied to systems with very few observations or whose observations are contaminated by noise. A comparison of the application of the traditional GPM method and the proposed GPM/Bayes method showed that the latter was able to make prognostic estimates in every case, whereas the former method was often unable to make RUL estimates very early in life. In addition, the GPM/Bayes model had smaller Mean Absolute Percent Error (MAPE) for predictions made early in system life; as more information became available, the error of the two methods converged because in both cases prognostic estimates were based primarily on the available prognostic parameter values with little emphasis on the prior information.

Additionally, the effect of noisy prognostic parameters on model performance was investigated. It was shown that the GPM/Bayes methodology can perform in the face of significant noise contamination. Parameter noise slows the response time of the regression model to transition from the population-based prior information to the information contained in the system-specific observations. However, as more observations become available, the model is able to determine the correct underlying regression function despite high levels of noise. The RUL estimates made with clean data, data contaminated with 20% noise, and data contaminated with 80% noise converged as the system aged and approached end of life.

The performance of GPM prognostics depends primarily on the identification of an appropriate prognostic parameter. Ideally, a prognostic parameter should have three key qualities: monotonicity, prognosability, and trendability. Monotonicity is a measure of the general positive or negative trend of the prognostic parameter. Damage is generally assumed to be cumulative and irreversible. Because the prognostic parameter can be considered an indicator of system health, if the parameter were not monotonic, it would indicate some measure of self-healing is taking place. This assumption is not valid for some specific components, such as batteries which do experience some increased capacity after a period of rest. However, it is usually considered valid when dealing with an entire system. Prognosability characterizes how well clustered the failure values are for a population of systems. Because a critical failure threshold must be identified for the GPM model, failure values should be well clustered in order to give a crisp failure value and reduce RUL uncertainty. Finally, trendability measures how well each parameter for a population of systems can be described by the same underlying function. The GPM depends on fitting a parametric model to the prognostic parameter and extrapolating it to failure, so it is key that the same parametric function be applicable to the entire population of systems with the same type of failure mechanism.

By formalizing these metrics so that the suitability of a candidate prognostic parameter can be quantified, it is straightforward to apply conventional optimization methods to identify an optimal, or near-optimal, prognostic parameter from many possible data sources including sensed data, monitoring system residuals, environmental and operational conditions, fault detection and diagnostic results, etc. This research applied Genetic Algorithm (GA) optimization to identify appropriate prognostic parameters for two very different data sets: the turbofan engine simulation data given in the PHM '08 prognostic challenge and milling data, both available at the NASA Prognostic Data Repository. GA runtime scales exponentially with the number of parameters to optimize; an input selection method was presented to alleviate this computational burden. Inputs which have total suitability less than some threshold, generally 1.5, are removed from the candidate parameter inputs. Application to the PHM challenge data set showed that the GA of the full set of possible inputs and the reduced set resulted in prognostic parameters of comparable performance, but with significantly reduced optimization runtime. In addition, the examples investigating the effect of noise on parameter suitability and performance and the milling data set indicate that parameter suitability is positively correlated with model performance; that is, parameters with higher suitability metrics result in greater model performance on the measure of mean absolute percent error of the failure time.

The PHM challenge data application utilized the results of a condition monitoring and fault detection system to characterize the degradation in a given system. Prognostic parameters were generated from a subset of the monitoring system residuals; monitoring system residuals are natural components of a prognostic parameter because they inherently measure the deviation of a system from normal operation. Three prognostic parameters were identified for the PHM '08 challenge data set: one through visual inspection and two using the proposed optimization method. The three parameters had comparable suitability metrics and resulted in comparable model performance on the test runs. The parameter identified through visual inspection involved several weeks of expert analysis to form the final parameter. Conversely, the first GA-optimized parameter, which included 11 monitoring system residuals as possible inputs, took a fraction of a manhour to set up and 1.68 hours of unsupervised computer runtime. The second GA-optimized parameter considered a subset of the eleven residuals, using only five of the possible inputs. This optimization ran in 0.93 hours, nearly half the time of the full optimization. This example application suggests that the proposed prognostic parameter selection method performs comparably to expert analysis in a fraction of the time. This application, however, benefited from extensive analysis given to the system before the optimization method was developed.

The second example application utilized milling data available at the NASA repository. Several prognostic parameters were quickly identified for this data set. The first parameter optimization lumped all fifteen available runs together to develop one parameter for the entire population. In addition, two separate parameters were developed for the two materials being bored: cast iron and steel. By applying the automated method, it is simple to develop many competing prognostic models to determine if additional factors, such as operating conditions, should be considered in model development.

This research presented the development of the GPM/Bayes prognostic model and an automated method for identifying appropriate prognostic parameters from many available data sources. The efficacy of both methodologies was illustrated. Additionally, the development of a MATLAB-based Process and Equipment Prognostics (PEP) toolbox was discussed. The original methods developed for this dissertation, as well as other prognostic algorithms, are available in the PEP toolbox to aid in rapid model prototyping and full health monitoring in conjunction with the Process and Equipment Monitoring toolbox.

7 Future Work

The area of failure prognostics holds many opportunities for continuing research beyond the scope of this dissertation. Several such areas have been mentioned throughout this report; a few key areas are outlined in greater detail here.

This research focused on a Genetic Algorithms approach to identifying prognostic parameters from linear combinations of data sources. However, a more general approach would consider more complicated functions of the available data, such as exponential functions. This may be accomplished through the use of Genetic Programming and would be an interesting advancement on the current work. In addition, a Genetic Programming approach may be useful for identifying the most appropriate parametric function to fit a given prognostic parameter.

All empirical prognostics systems suffer from the need for failure data. A key area of research in prognostics is development and dissemination of failure data for prognostic model development, either collected in real-world or accelerated testing environments or simulated by high-fidelity physics of failure models. Industry's desire for proprietary failure information should become secondary to the need to coalesce available information sources in order to develop accurate prognostic models and algorithms. NASA has begun this push with the Prognostic Data Repository, but the few data sets available there are the result of simulation or laboratory tests. Most of these data sets involve only a few examples of failure, and sometimes failure is not clearly designated so it must be arbitrarily assigned as in the milling data set used in this research. The availability of additional types of failure data which include many examples of failure and give clear definitions of failure will aid researchers in developing generic prognostics algorithms that can be applied to a variety of problems.

The PEP toolbox is designed to aid in the rapid development of empirical prognostic models; however, choosing between competing models is not a trivial task. It is up to the developer to determine the correct prognostic type and algorithm for each system. However, it may be possible to facilitate, and to some extent automate, this task as well. By comparing the available data and information for a particular system to the requirements of each prognostic method, appropriate models for RUL estimation may be identified. As discussed earlier, some work has been completed to develop appropriate performance metrics for prognostic algorithms. However, this remains a grey area in the field. A unified set of performance metrics would also allow for quick comparison of prognostic model performance to aid in identifying the most appropriate model for a particular situation.

Three types of prognostics were introduced based on the information used to make RUL estimations: Type I, or traditional reliability analysis; Type II, or stressor-based; and Type III; or degradation-based. During the course of a system's lifetime, there is a natural progression between the applicability of each of these three types. When a new system or component is acquired, the best estimate of its total lifetime is population-based statistics, or Type I prognostics. As operations are planned for the system and it is put into use, Type II prognostics can be used to evaluate the effect of planned usage on the system's lifetime. Finally, as degradation occurs, if it is possible to quantify this degradation, then Type III prognostics may be applied. Development of an intelligent method to move between the three types of prognostics as they become applicable will make full lifecycle prognostics possible. The proposed GPM/Bayes methodology considers both Type I and Type III prognostics, and transitions between the two as more degradation data becomes available; however, a life-cycle prognostic system should be able to move between the three, considering information from each type as is appropriate.

References

1. Union of Concerned Scientists, *Davis-Besse* Outage Report: http://www.ucsusa.org/assets/documents/nuclear_power/davis-besse-ii.pdf
2. Feng, W.C., "Making a Case for Efficient Super Computing," *Queue* Oct 2003: pp 54 – 64.
3. Toms, L.A., *Machinery Oil Analysis - Methods, Automation & Benefits*, 1st Edition, Larry A Toms Technical Services, Pensacola, Florida, 1994.
4. Lu, C.J., and W. Meeker, "Using Degradation Measures to Estimate a Time-to-Failure Distribution," *Technometrics* 35 (2) 1993: 161 – 174.
5. Upadhyaya, B.R., M. Naghedolfeizi, and B. Raychaudhuri, "Residual Life Estimation of Plant Components," *P/PM Technology* June, 1994: 22 – 29.
6. Chinnam, R.B., "On-line Reliability Estimation of Individual Components, Using Degradation Signals," *IEEE Transactions on Reliability* 48 (4) 1999: 403 – 412.
7. Brown, D.W., P.W. Kalgren, C.S. Byington, and M.J. Roemer, "Electronic Prognostics – A Case Study using Global Positioning System (GPS)," *Microelectronics Reliability* 47 2007: 1874 – 1881.
8. Engel, S., B. Gilmartin, K. Bongort, and A. Hess, "Prognostics, the Real Issues Involved with Predicting Life Remaining," *Proceedings of the IEEE Aerospace Conference*, pp. 457-469, 2000.
9. Hess, A., G. Calvello, and P. Frith, "Challenges, Issues, and Lessons Learned Chasing the 'Big P': Real Predictive Prognostics Part 1," *Proceedings of the IEEE Aerospace Conference*, pp. 3610 – 3619, March 2005.
10. Condition Based Maintenance Plus (2005). Online: http://www.acq.osd.mil/log/mppr/CBM%2B.htm
11. Heo, G.Y., "Condition Monitoring Using Empirical Models: Technical Review and Prospects for Nuclear Applications," *Nuclear Engineering and Technology* 40 (1) 2008: 49 – 68.
12. Greitzer, F.L., E.J. Stahlman, T.A. Ferryman, B.W. Wilson, L.J. Kangas, and D.R. Sisk, "Development of a Framework for Predicting Life of Mechanical Systems: Life Extension Analysis and Prognostics (LEAP)," International Society of Logistics (SOLE) Symposium, Las Vegas, Aug 30 – Sept 2, 1999.
13. Vichare, N., and M. Pecht, "Prognostics and Health Management of Electronics," *IEEE Transactions on Components and Packaging Technologies*, 29 (1), 2006, pp 222 – 229.
14. "Hot Carrier (HC) Prognostic Cell," Ridgetop Semiconductor-Sentinel Silicon Library, Aug 2004.
15. Mishra, S. and M. Pecht, "In-situ Sensors for Product Reliability Monitoring," *Proceedings of SPIE*, vol 4755 (2002) pp 10-19.
16. Kalgren, P.W., M. Baybutt, A. Ginart, C. Minnella, M.J. Roemer, and T. Dabney, "Application of Prognostic Health Management in Digital Electronic Systems," *Proceedings of the 2007 IEEE Aerospace Conference*, March, 2007: 1 – 9.
17. Heng, A., S. Zhang, A. Tan, and J. Mathew, "Rotating Machinery Prognostics: State of the Art, Challenges, and Opportunities," *Mechanical Systems and Signal Processing* 23 2009: 724 – 739.
18. Carden, E.P. and P. Fanning, "Vibration Based Condition Monitoring: A Review," *Structural Health Monitoring* 3 (4) 2004: 355 – 377.
19. Doebling, S.W., C.R. Farrar, M.B. Prime, and D.W. Shevitz, "Damage Identification and Health Monitoring of Structural and Mechanical Systems from Changes in their Vibration Characterisstics: A Literature Review," Technical Report LA-13070-MS, Los Alamos National Lab.
20. Catbas, F.N. and A.E. Atkan, "Condition and Damage Assessment: Issues and Some Promising Indices," *Journal of Structural Engineering* 128 (8) 2002: 1026 – 1036.

21. Vachtsevanos, G., Kim, W., Al-Hasan, S., Rufus, F., Simon, M., Schrage, D. and Prasad, J.V.R., "Mission Planning and Flight Control: Meeting the Challenge with Intelligent Techniques," *Journal of Advanced Computational Intelligence*, Vol. 1, No. 1, pp. 62-70, October 1997.
22. Wang, P. and Vachtsevanos, G. "Fault Prognostics Using Dynamic Wavelet Neural Networks", *Journal of Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, vol.15, pp. 349-365, 2001.
23. Orchard, M., and Vachtsevanos, G., "A particle filtering approach for on-line failure prognosis in a planetary carrier plate," *International Journal of Fuzzy Logic and Intelligent Systems*, vol. 7, no. 4, pp. 221-227, 2007.
24. Keller, K., K. Swearingen, J. Sheahan, M. Baily, J. Dunsdon, B. Cleeve, K.W. Przytula, and B. Jordan, "Aircraft Electrical Power Systems Prognostics and Health Management," IEEE Aerospace Conference, 2006: Big Sky, MT.
25. Orsagh, R., D. Brown, M. Roemer, T. Dabnev, and A. Hess, "Prognostic Health Management for Avionics System Power Supplies," *Proceedings of the IEEE Aerospace Conference*, pp.3585-3591, 5-12 March 2005: Big Sky, MT.
26. Roemer, M.J., J. Dzakowic, R.F. Orsagh, C.S. Byington, G. Vachtsevanos, "Validation and Verification of Prognostic and Health Management Technologies," *Proceedings of the IEEE Aerospace Conference*, 2005: 3941 – 3947.
27. Ferrell, B.L., "JSF Prognostics and Health Management," *Proceedings of the IEEE Aerospace Conference*, 1999: 471.
28. Ferrell, B.L., "Air Vehicle Prognostics and Health Management," *Proceedings of the IEEE Aerospace Conference,* 2000: 145 – 146.
29. Smith, G., J.B. Schroeder, S. Navarro, and D. Haldeman, "Development of a Prognostics and Health Management Capability for the Joint Strike Fighter," *Proceedings of AUTOTESTCON*, 1997: 676 – 682.
30. Hess, A. and L. Fila, "The Joint Strike Fighter (JSF) PHM Concept: Potential Impact on Aging Aircraft Problems," *Proceedings of the IEEE Aerospace Conference*, 2002: 3021 – 3026.
31. Line, J.K. and N.S. Clements, "A Systematic Approach for Developing Prognostic Algorithms on Large Complex Systems," *Proceedings of the IEEE Aerospace Conference*, 2005: 1 – 7.
32. Kacprzynski, G.J., A. Sarlashkar, M.J. Roemer, A. Hess, and W. Hardman, "Predicting Remaining Life by Fusing the Physics of Failure Modeling with Diagnostis," *Journal of the Mineral, Metals, and Materials Society* 56 (3) 2004: 29 – 35.
33. Liao, H., W. Zhao, and H. Guo, "Predicting Remaining Useful Life of an Individual Unit Using Proportional Hazards Model and Logistic Regression Model." *Proceedings of the Reliability and Maintainability Symposium (RAMS)*, 2006: 127 – 132.
34. Goodman, D., B. Vermeire, P. Spuhler, and H. Venkatramani, "Practical Application of PHM/Prognostics to COTS Power Converters," *Proceedings of the IEEE Aerospace Conference*, 2005: 3573 – 3578.
35. Urmanov, A. and J.W. Hines, "Electronic Prognostics Through Continuous System Telemetry and Empirical Models," 53rd Annual Reliability and Maintainability Symposium (RAMS), Orlando, FL, Jan 22-25, 2007.
36. Han, D.K., M. G. Pecht, D.K. Anand, and R. Kavetsky, "Energetic Material/Systems Prognostics," 53rd Annual Reliability and Maintainability Symposium (RAMS), Orlando, FL, Jan 22-25, 2007.

37. Orsagh, R.F., D.W. Brown, P.W. Kalgren, C.S. Byington, A.J. Hess, and T. Dabney, "Prognostic Health Management for Avionic Systems," IEEE Aerospace Conference, 2006: Big Sky, MT.
38. Sheppard, J.W., M.A. Kaufman, and T.J. Wilmering, "IEEE Standards for Prognostics and Health Management," 2008 IEEE Autotestcon, Salt Lake City, UT: Sept 8 – 11, 2008.
39. Meeker, W.Q., and M. Hamada, "Statistical Tools for the Rapid Development and Evaluation of High-Reliability Products," *IEEE Transactions on Reliability* 44 (2) 1995: 187 – 198.
40. Klinger, D.J., "Failure Time and Rate Constant of Degradation: An Argument for the Inverse Relationship," *Microelectronic Reliability* 32 (7) 1992: 987 – 994.
41. Mishra, S., S. Ganesan, M. Pecht, and J. Xie, "Life Consumption Monitoring for Electronics Prognostics," *Proceedings of the IEEE Aerospace Conference*, 2004: 3455 – 3467.
42. Baruah, P., R.B. Chinnam, and D. Filev, "An Autonomous Diagnostics and Prognostics Framework for Condition-Based Maintenance," 2006 International Joint Conference on Neural Networks, Vancouver, BC, Canada: July 16 – 21, 2006.
43. Kothamasu, R., S.H. Huang, and W.H. VerDuin, "System Health Monitoring and Prognostics – A Review of Current Paradigms and Practices," *International Journal of Advanced Manufacturing Technology* 28 2006: 1012 – 1024.
44. Pipe, K., "Practical Prognostics for Condition Based Maintenance," 2008 International Conference on Prognostics and Health Management, Oct 6 – 9, 2008: Denver, CO.
45. Callan, R., B. Larder, and J. Sandiford, "An Integrated Approach to the Development of an Intelligent Prognostic Health Management System," *Proceedings of the IEEE Aerospace Conference*, 2006: 12.
46. Hines, J.W., R. Seibert, S.A. Arndt, "Technical Review of On-Line Monitoring Techniques for Performance Assessment (NUREG/CR-6895) Vol. 1, State-of-the-Art." Published January, 2006.
47. Wald, A. "Sequential Tests of Statistical Hypotheses." Annals of Mathematical Statistics 16 (2): 117–186.
48. Line, J.K., and N.S. Clements, "Prognostics Usefulness Criteria," *Proceedings of the IEEE Aerospace Conference*, 2006: 7.
49. Gu, J., N. Vichare, T. Tracy, and M. Pecht, "Prognostics Implementation Method for Electronics," *Proceedings of the Reliability and Maintainability Symposium (RAMS)*, 2007: 101 – 106.
50. Janasak, K.M., and R.R. Beshears, "Diagnostics to Prognostics – A Product Availability Technology Evolution," *Proceedings of the Reliability and Maintainability Symposium (RAMS)*, 2007: 113 – 118.
51. Kelkar, N., A. Dasgupta, M. Pecht, I.Knowles, M. Hawley, and D. Jennings, "'SMART' Electronic Systems for Condition-Based Health Management," *Proceedings of 9th International Congress and Exhibition on Condition Monitoring and Diagnostic Engineering Management (COMADEM)*, 1996: 591 – 602.
52. Greitzer, F.L., and T.A. Ferryman, "Predicting Remaining Life of Mechanical Systems," Intelligent Ship Symposium IV, April 2 – 3, 2001.
53. Elsayed, E.A., and A.C-K. Chen, "Recent Research and Current Issues in Accelerated Testing," *Proceedings of the IEEE Conference on Systems, Man, and Cybernetics*, 1998: 4704 – 4709.
54. Carey, M.B., and R.H. Koenig, "Reliability Assessment Based on Accelerated Degradation: A Case Study," *IEEE Transactions on Reliability* 40 (5) 1991: 499 – 506.

55. Lemoine, A.J., and M.L. Wenocor, "On Failure Modeling," Navel Research Logistics Quarterly 32 1985: 497 – 508.
56. Tang, L.C., and D.S. Chang, "Reliability Prediction Using Nondestructive Accelerated-Degradation Data: Case Study on Power Supplies," *IEEE Transactions on Reliability* 44 (4) 1995: 562 – 566.
57. Park, C. and W.J. Padgett, "Stochastic Degradation Models with Several Accelerating Variables," *IEEE Transactions on Reliability* 55 (2) 2006: 379 – 390.
58. Ho, Y.-C., and D. L. Pepyne, "Simple Explanation of the No Free Lunch Theorem of Optimization", *Cybernetics and Systems Analysis*, 38 (2), 2002, pp 292 – 298.
59. Koppen, M., "No-Free-Lunch Theorems and the Diversity of Algorithms", *Congress on Evolutionary Computation*, 1, 2004, pp 235 – 241.
60. Hines, J.W. and A. Usynin, "Current Computational Trends in Equipment Prognostics," *International Journal of Computational Intelligence Systems* 1 (1) 2008: 1 – 9.
61. Garvey, D. and J.W. Hines, "Dynamic Prognoser Architecture via the Path Classification and Estimation (PACE) Model," AAAI Fall Symposium on Artificial Intelligence for Prognostics, Nov 9-11, 2007, Arlington, VA.
62. Garvey, Dustin R. (2007), "An Integrated Fuzzy Inference Based Monitoring, Diagnostic, and Prognostic System," Ph.D. Dissertation, Nuclear Engineering Department, University of Tennessee, Knoxville: 2007.
63. Abernethy, R. B., *The New Weibull Handbook*, 2nd edn. ISBN 0 9653062 0 8. Abernethy, North Palm Beach, 1996.
64. Yang, K. and J. Xue, "Continuous State Reliability Analysis," *Proceedings of the Annual Reliability and Maintainability Symposium (RAMS)*, 1996: 251 – 257.
65. Girish, T., S.W. Lam, and S.R. Jayaram, "Reliability Prediction Using Degradation Data – A Preliminary Study Using Neural Network-based Approach," European Safety and Reliability Conference, Jun 15-18, 2003: Maastricht, The Netherlands.
66. Kharoufeh, J.P. and S.M. Cox, "Stochastic Models for Degradation-based Reliability," *IIE Transactions* (2005) 37, 533 – 542.
67. Chen, Z. and S. Zheng, "Lifetime Distribution Based Degradation Analysis," *IEEE Transactions on Reliability*, 54 (1), March 2005, 3 – 10.
68. Xu, D. and W. Zhao, "Reliability Prediction using Multivariate Degradation Data," *Proceedings of the Annual Reliability and Maintainability Symposium (RAMS)*, 2005: 337 – 341.
69. Guess, F. and F. Proschan, "Mean Residual Life: Theory and Applications," FSU Statistics Report M702. AFOSR Technical Report No. 85-178, June 1985.
70. Pecht, M., D. Das, and A. Ramakrishnan, "The IEEE Standards on Reliability Program and Reliability Prediction Methods for Electronic Equipment," *Microelectronic Reliability*, 42 1259 – 1266, 2002.
71. Lall, P., M. Pecht, and E. Harkim, *Influence of Temperature on Microelectronics and System Reliability*, New York: CRC Press, 1997.
72. Vichare, N., P. Rodgers, V. Eveloy, and M. Pecht, "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 (4), December 2004: 658 – 663.
73. Azzam, H., "A Practical Approach for the Indirect Prediction of Structural Fatigue from Measured Flight Parameters," *Journal of Aerospace Engineering* 211 (G): 29 – 38.

74. Baybutt, M., C. Minnella, A. Ginart, P.W. Kalgren, and M.J. Roemer, "Improving Digital System Diagnostics Through Prognostic and Health Management (PHM) Technology," *IEEE Transactions on Instrumentation and Measurement* 58 (2), 2009: 255 – 262.
75. Goodman, D., "Prognostic Techniques for Semiconductor Failure Modes," Ridgetop Group, Inc. white paper, 2000: available at www.Ridgetop-Group.com.
76. Hofmeister, J.P., P. Lall, R. Graves, "In-Situ, Real-Time Detector for Faults in Solder Joint Networks Belonging to Operational, Fully Programmed Field Programmable Gate Arrays (FPGAs)," *Proceedings of the IEEE AUTOTESTCON*, 2006: 237 – 243.
77. Hofmeister, J.P., J. Judkins, E. Oritz, D. Goodman, and P. Lall, "Real-time BIST Detector for BGA Faults in Field Programmable Gate Arrays (FPGAs)," Ridgetop Group, Inc. white paper, 2006: available at www.Ridgetop-Group.com.
78. Goodman, D., B. Vermeire, J. Ralston-Good, and R. Graves, "A Board-Level Prognostic Monitor for MOSFET TDDB," IEEE Aerospace Conference, 2006: Big Sky, MT.
79. Mishra, S., M. Pecht, and D. Goodman, "In-situ Sensors for Product Reliability Monitoring," *Proceedings of SPIE,* 2006: 10 – 19.
80. Bogdanoff, J.L., and F. Kozin, *Probabilistic Models of Cumulative Damage*, John Wiley and Sons, New York: 1985.
81. Kwan, C., X. Zhang, R. Xu, and L. Haynes, "A Novel Approach to Fault Diagnostics and Prognostics," *Proceedings of the IEEE International Conference on Robotics & Automation*, 2005: 604 – 609.
82. Dale, C.J., "Application of the Proportional Hazards Model in the Reliability Field," *Reliability Engineering* 10, 1985:1 – 14.
83. Kumar, D. and B. Klefjo, "Proportional Hazards Model: A Review," *Reliability Engineering and System Safety* 44, 1994: 177 – 188.
84. Oja, M., J.K. Line, G. Krishnan, R.G. Tryon, "Electronic Prognostics with Analytical Models using Existing Measurands," 61st Conference of the Society for Machinery Failure Prevention Technology (MFPT), April 17 – 19, 2007: Virginia Beach, VA.
85. Valentin, R., M. Osterman, B. Newman, "Remaining Life Assessment of Aging Electronics in Avionic Applications," *Proceedings of the Annual Reliability and Maintainability Symposium (RAMS)*, 2003: 313 – 318.
86. Pecht, M. and A. Dasgupta, "Physics of Failure: An Approach to Reliable Product Development," *Journal of the Institute of Environmental Sciences* 38 (5) 1995: 30 – 34.
87. Kacprzynski, G.J., M.J. Roemer, G. Modgil, A. Palladino, K. Maynard, "Enhancement of Physics-of-Failure Prognostic Models with System Level Features," IEEE Aerospace Conference, 2002: Big Sky, MT.
88. Ramakrishnan, A. and M.G. Pecht, "A Life Consumption Monitoring Methodology for Electronic Systems," *IEEE Transactions on Components and Packaging Technologies* 26 (3) 2003: 625 – 634.
89. Pecht, M., M. Dube, M. Natishan, R. Williams, J. Banner, and I. Knowles, "Evaluation of Built-in Test," *IEEE Transactions on Aerospace and Electronic Systems* 37 (1) 2001: 266 – 271.
90. Cox, D.R. and D. Oakes, *Analysis of Survival Data*, Chapman and Hall: 1984.
91. Hines, J.W., A. Usynin, and A. Urmanov, "Prognosis of Remaining Useful Life for Complex Engineering Systems," American Nuclear Society International Topical Meeting on Nuclear Plant

Instrumentation, Control, and Human-Machine Interface Technologies (NPIC&HMIT), 2006: Albuquerue, NM.

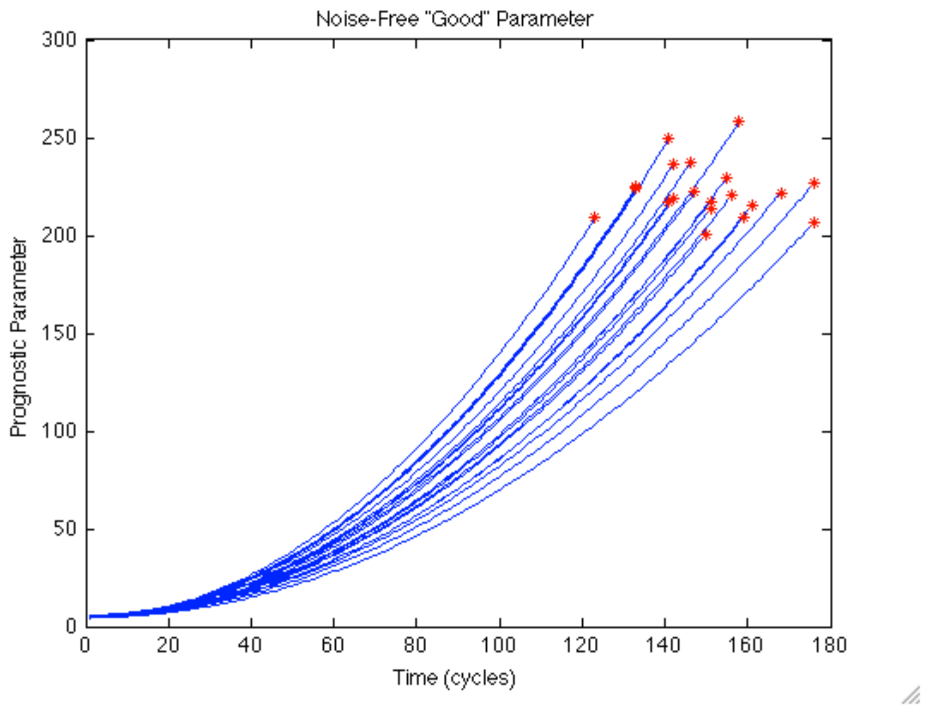
92. Byington, C.S., M. Watson, D. Edwards, P. Stoelting, “A Model-Based Approach to Prognostics and Health Management for Flight Control Actuators,” *Proceedings of the IEEE Aerospace Conference*, 2004: 3551 – 3562.
93. Luo, J., M. Namburu, K. Pattipati, L. Qiao, M. Kawamoto, S. Chigusa, “Model-based Prognostic Techniques,” IEEE Systems Readiness Technology Conference (AUTOTESTCON), Sep 22-25 2003: Anaheim, CA
94. Hines, J.W., J. Garvey, J. Preston, and A. Usynin, “Empirical Methods for Process and Equipment Prognostics,” Tutorial presented at the *IEEE Reliability and Maintainability Symposium (RAMS)*, 2007.
95. He, D., S. Wu, P. Banerjee, and E. Bechhoefer, “Probabilistic Model Based Algorithms for Prognostics,” IEEE Aerospace Conference, 2006: Big Sky, MT.
96. Esary, J.D. and A.W. Marshall, “Shock Models and Wear Processes,” *The Annals of Probability* 1 (4) 1973: 627 – 649.
97. Mallor, F. and J. Santos, “Classification of Shock Models in System Reliability,” Monograf´ıas del Semin. Matem. Garc´ıa de Galdeano. 27 2003: 405–412.
98. Gut, A., “Cumulative Shock Models,” *Advances in Applied Probability* 22 (2) 1990: 504 – 507.
99. Greitzer, F.L., “Life Extension Analysis and Prognostics (LEAP) Architectures,” FY 2001 Laboratory Directed Research and Development Annual Report PNNL: 312 – 316.
100. Roemer, M.J., G.J. Kacprzyniski, and M.H. Schoeller, “Improved Diagnostic and Prognostic Assessments using Health Management Information Fusion,” IEEE Systems Readiness Technology Conference (AUTOTESTCON), 2001: 365 – 377.
101. Jardine, A.K.S., D. Lin, and D. Banjevic, “A Review on Machinery Diagnostics and Prognostics Implementing Condition-Based Maintenance,” *Mechanical Systems and Signal Processing* 20 2006: 1483 – 1510.
102. Choi, S. and C.J. Li, “Estimation of Gear Tooth Transverse Crack Size from Vibration by Fusing Selected Gear Condition Indices,” *Measurement Science and Technology* 17 2006: 2395 – 2400.
103. Goebel, K. and P. Bonissonne, “Prognostic Information Fusion for Constant Load Systems,” 7th International Conference on Information Fusion, 2004: Stockholm, Sweden.
104. Hughes, G.F., J.F. Murray, K. Kreutz-Delgado, and C. Elkan, “Improved Disk Drive Failure Warnings,” *IEEE Transactions on Reliablity* 51 (3) 2002: 350 – 357.
105. Wang, P. and D.W. Coit, "Reliability Prediction based on Degradation Modeling for Systems with Multiple Degradation Measures," *Proceedings of the Reliability and Maintainability Symposium (RAMS),* 2004: 302 – 307.
106. Hines, J.W., D. Garvey, R. Seibert, A. Usynin, and S.A. Arndt, "Technical Review of On-Line Monitoring Techniques for Performance Assessment (NUREG/CR-6895) Vol. 2, Theoretical Issues," Published May, 2008.
107. Byington, C.S., M.J. Roemer, P.W. Kalgren, “Verification and Validation of Diagnostic/Prognostic Algorithms,” Machinery Failure Prevention Technology (MFPT) Conference, 2005: Virginia Beach, VA.
108. Vachtsevanos, G. “Performance Metrics for Fault Prognosis of Complex Systems.” *Proceedings of the IEEE Systems Readiness Technology Conference (AUTOTESTCON),* 2003: 341 – 345.

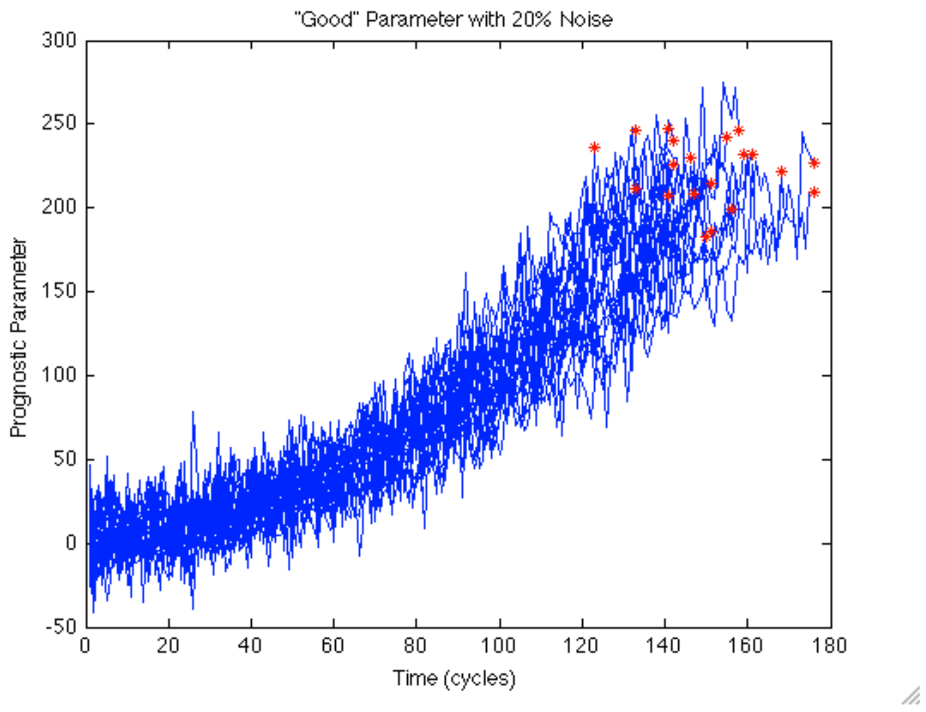
109. Saxena, A., J. Selaya, E. Balaban, K. Goebel, B. Saha, S. Saha, and M. Schwabacher, "Metrics for Evaluating Performance of Prognositc Techniques." International Conference on Prognostics and Health Management, 2008: Denver, CO.
110. Aho, A.V., J.D. Ullman, and J.E. Hopcroft, *Data Structures and Algorithms*. Addison Wesley, 1983.
111. Volkov, V. and J.W. Demmel, "Benchmarking GPUs to Tune Dense Linear Algebra," *Proceedings of the ACM/IEEE Conference on Supercomputing*, 2008: 1 – 11.
112. Goodman, D.L, S. Wood, and A. Turner, "Return-on-Investment (ROI) for Electronic Prognostics in MIL/AERO Systems," *Proceedings of the IEEE Systems Readiness Technology Conference (AUTOTESTCON),* 2005: 73 – 75.
113. Wood, S. and D. L. Goodman, "Return-on-Investment (ROI) for Electronic Prognostics in High Reliability Telecom Applications." *Proceedings of the 28th Annual International Telecommunications Energy Conference (INTELEC)*, 2006: 1 – 3.
114. Banks, J. and J. Merenich, "Cost Benefit Analysis for Asset Health Management Technology," *Proceedings of the IEEE Reliability and Maintainability Symposium (RAMS)*, 2007: 95 – 100.
115. Leao, B. P., T. Yoneyama, G. C. Rocha, and K.T. Fitzgibbon, "Prognostic Performance Metrics and their Relation to Requirements, Design, Verification, and Cost-Benefit." International Conference on Prognostics and Health Management, 2008: Denver, CO.
116. Kacprzynski, G.J., A. Liberson, A. Palladino, M.J. Roemer, A.J. Hess, and M. Begin, "Metrics and Development Tools for Prognostic Algorithms," *Proceedings of the IEEE Aerospace Conference*, 2004: 3809 – 3815.
117. Saxena, A., J. Celaya, B. Saha, S. Saha, and K. Goebel, "Evaluating Algorithm Performance Metrics Tailored for Prognostics," *Proceedings of the IEEE Aerospace Conference*, 2009: 1 – 13.
118. Saxena, A., K. Goebel, D. Simon, N. Eklund, "Prognostics Challenge Competition Summary: Damage Propagation Modeling for Aircraft Engine Run-to-Failure Simulation," International Conference on Prognostics and Health Management, 2008: Denver, CO.
119. Tamhane, A.C. and D.D. Dunlop, *Statistics and Data Analysis from Elementary to Intermediate*, Prentice-Hall, Inc, Upper Saddle River, New Jersey: 2000.
120. Kutner, M.H., C.J. Nachtsheim, and J. Neter, *Applied Linear Regression Models*, McGraw-Hill/Irwin, New York, New York: 2004 (4 ed).
121. J.W. Hines and D. Garvey. "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), August 9 - 11, 2005: Beijing, China.
122. Meeker, W.Q., L.A. Escobar, and C.J. Lu, "Accelerated degradation tests: modeling and analysis," *Technometrics*, vol. 40, no. 2, pp. 89-99, 1998.
123. Girish, T., S.W. Lam, J.S.R. Jayaram, "Reliability Prediction Using Degradation Data – A Preliminary Study Using Neural Network-based Approach," *Proceedings of the European Safety and Reliability Conference (ESREL 2003)*, Jun. 15-18, 2003: Maastricht, The Netherlands.
124. Lindely, D.V. and A.F. Smith, "Bayes Estimates for Linear Models," *Journal of the Royal Statistical Society (B)* 34 (1) 1972: 1 – 41.
125. Gelman, A., J. Carlin, H. Stern, and D. Rubin, *Bayesian Data Analysis* 2nd ed. Boca Raton: Chapman and Hall/CRC: 2004.

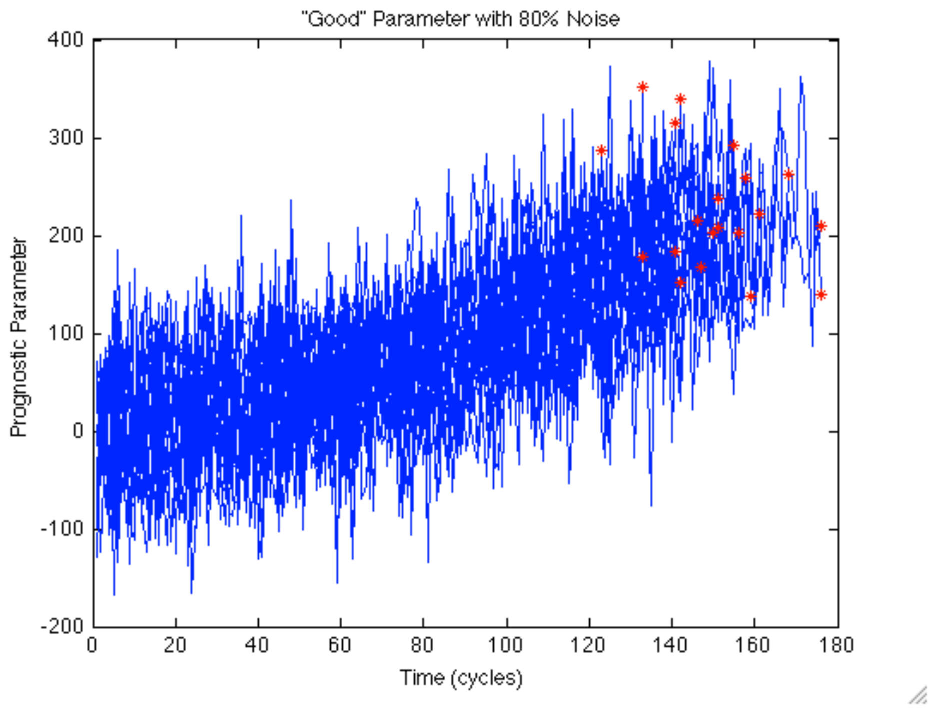
126. Carlin, B.P and T.A. Louis, *Bayes and Empirical Bayes Methods for Data Analysis*, 2nd ed. Boca Raton: Chapman and Hall/CRC: 2000.
127. Robinson, M.E. and M.T. Crowder, "Bayesian Methods for a Growth-Curve Degradation Model with Repeated Measures," *Lifetime Data Analysis* 6, 2000: 357 – 374.
128. Coble, J. and J.W. Hines, "Fusing Data Sources for Optimal Prognostic Parameter Selection." Sixth American Nuclear Society International Topical Meeting on Nuclear Plant Instrumentation, Control, and Human-Machine Interface Technologies NPIC&HMIT 2009, Knoxville, Tennessee, April 5-9, 2009.
129. Horst, R. and T. Hoang, *Global Optimization: Deterministic Approaches* Springer: New York: 1996.
130. Schneider, J.J., *Stochastic Optmization*, Springer-Verlag: Berlin, Heidelberg: 2006.
131. Haupt, R.L. and S.E. Haupt. *Practical Genetic Algorithms*, John Wiley & Sons Ltd, Hoboken, New Jersey: 2004.
132. Mitchell, M., S. Forrest, and J. Holland, "The Royal Road for Genetic Algorithms: Fitness Landscapes and GA Performance," *Toward a Practice of Autonomous Systems: Proceedings of the First European Conference on Artificial Life*, 1992: 245 – 254.
133. Bentley, P. J. and J.P Wakefield, "An Analysis of Multiobjective Optimization Within Genetic Algorithms," Technical Report ENGPJB96, University of Huddersfield, UK: 1996.
134. Ding, L. and J. Yu, "Some Theoretical Results about the Computation Time of Evolutionary Algorithms," *Proceedings of the 2005 conference on Genetic and evolutionary computation*, 2005: 1409 – 1415.
135. Saxena, A. and K. Goebel (2008). "C-MAPSS Data Set," NASA Ames Prognostics Data Repository, [http://ti.arc.nasa.gov/project/prognostic-data-repository], NASA Ames, Moffett Field, CA.
136. A. Agogino and K. Goebel (2007). "Mill Data Set," BEST lab, UC Berkeley. NASA Ames Prognostics Data Repository, [http://ti.arc.nasa.gov/project/prognostic-data-repository], NASA Ames, Moffett Field, CA.
137. Wang, P. and D. Coit, "Reliability and Degradation Modeling with Random or Uncertain Failure Threshold," *Proceedings of the IEEE Reliability and Maintainability Symposium (RAMS)*, 2007: 392 – 397.
138. Usynin, A., J.W. Hines, and A. Urmanov, "Uncertain Failure Thresholds in Cumulative Damage Models, "*Proceedings of the IEEE Reliability and Maintainability Symposium (RAMS)*, 2008: 334 – 340.
139. Sharp, M. and J.W. Hines, "Analysis of Prognostic Opportunities in Power Industry with Demonstration" *Sixth American Nuclear Society International Topical Meeting on Nuclear Plant Instrumentation, Control, and Human-Machine Interface Technologies* NPIC&HMIT 2009, Knoxville, Tennessee, April 5-9, 2009.

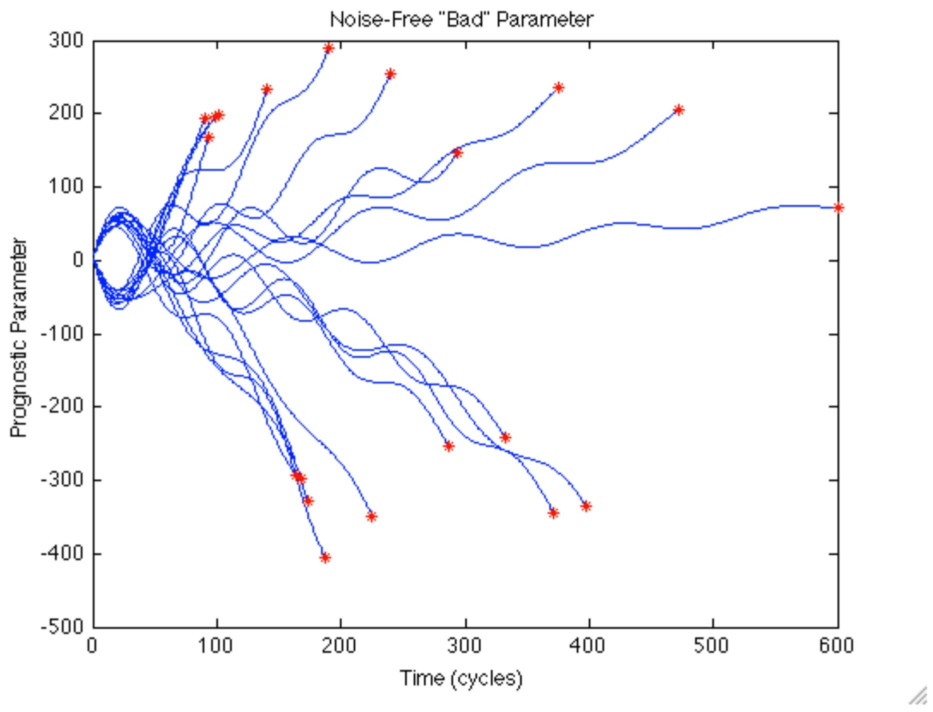
Appendix A: Additional Figures

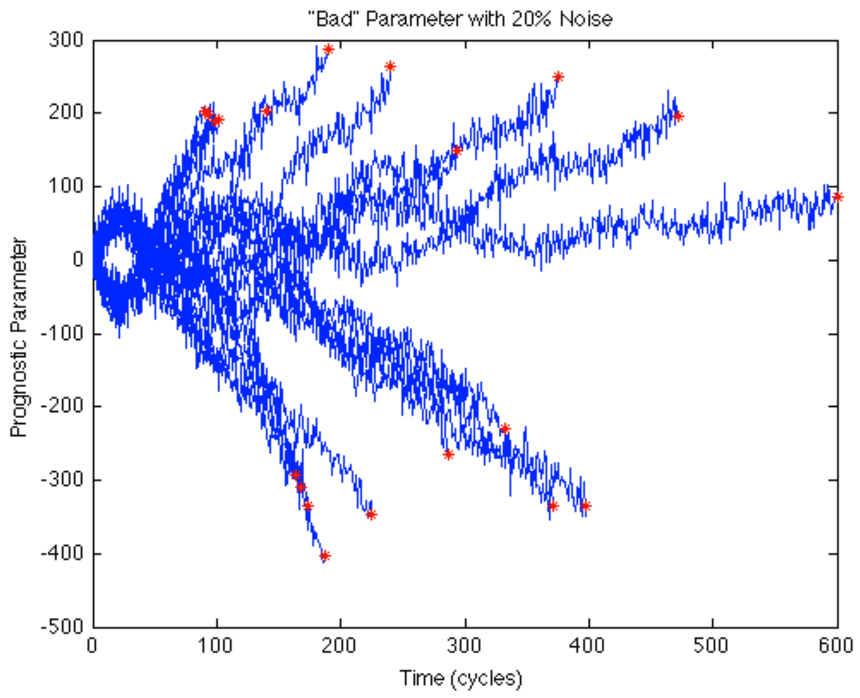
A.1 Noisy Prognostic Parameters

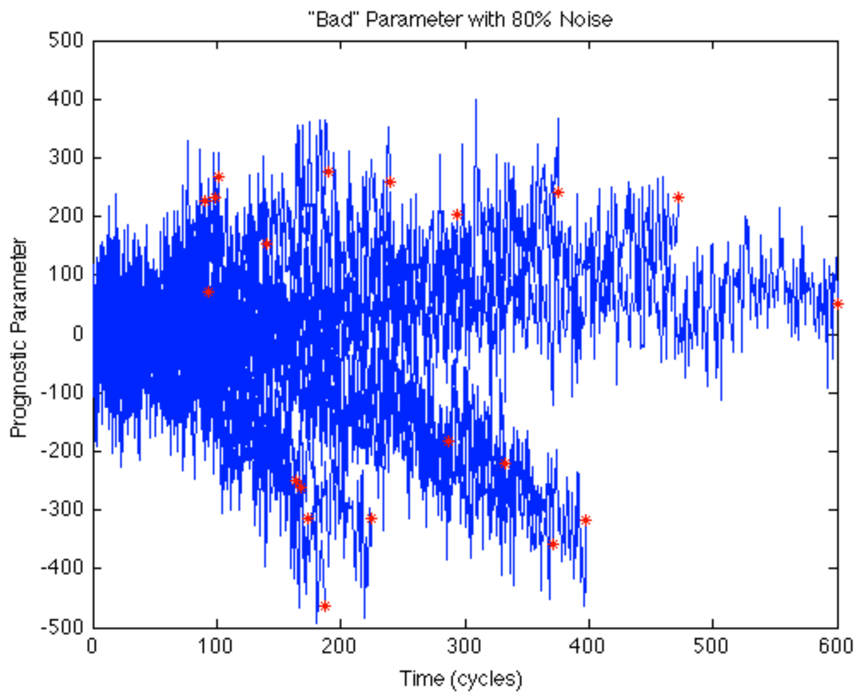




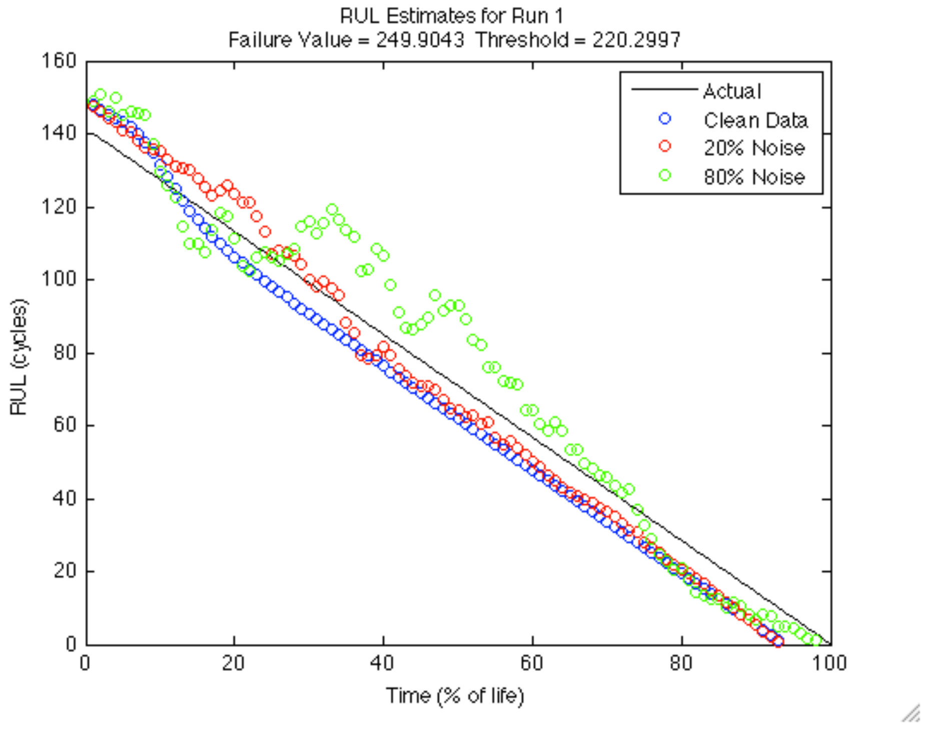

"Good" Parameter with 80% Noise
Prognostic Parameter
Time (cycles)
400
300
200
100
0
-100
-200
0
20
40
60
80
100
120
140
160
180

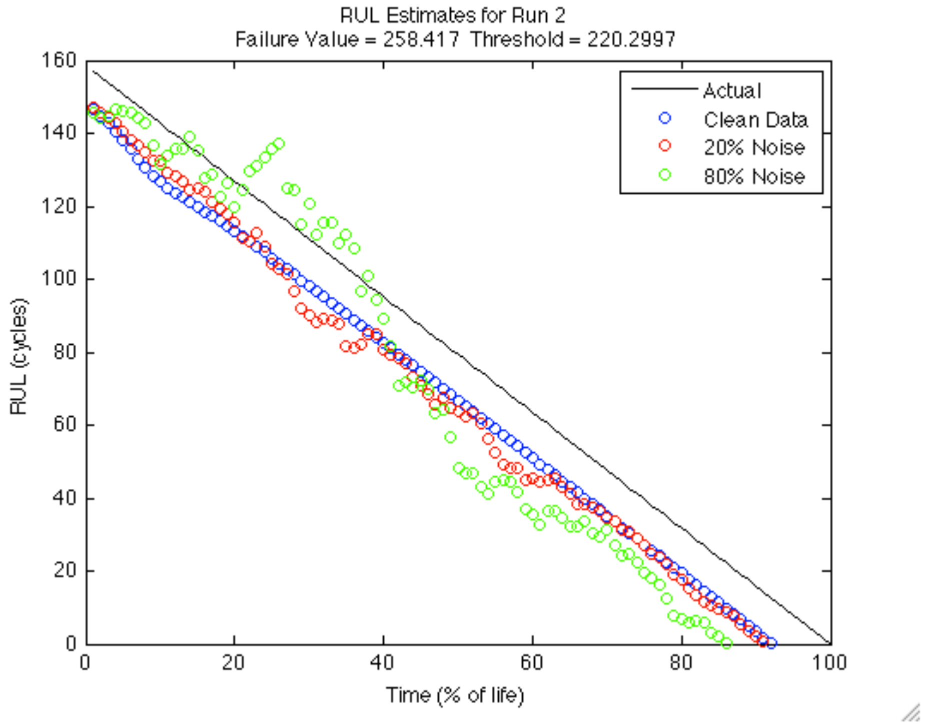

Noise-Free "Bad" Parameter
Prognostic Parameter
Time (cycles)
300
200
100
0
-100
-200
-300
-400
-500
0
100
200
300
400
500
600

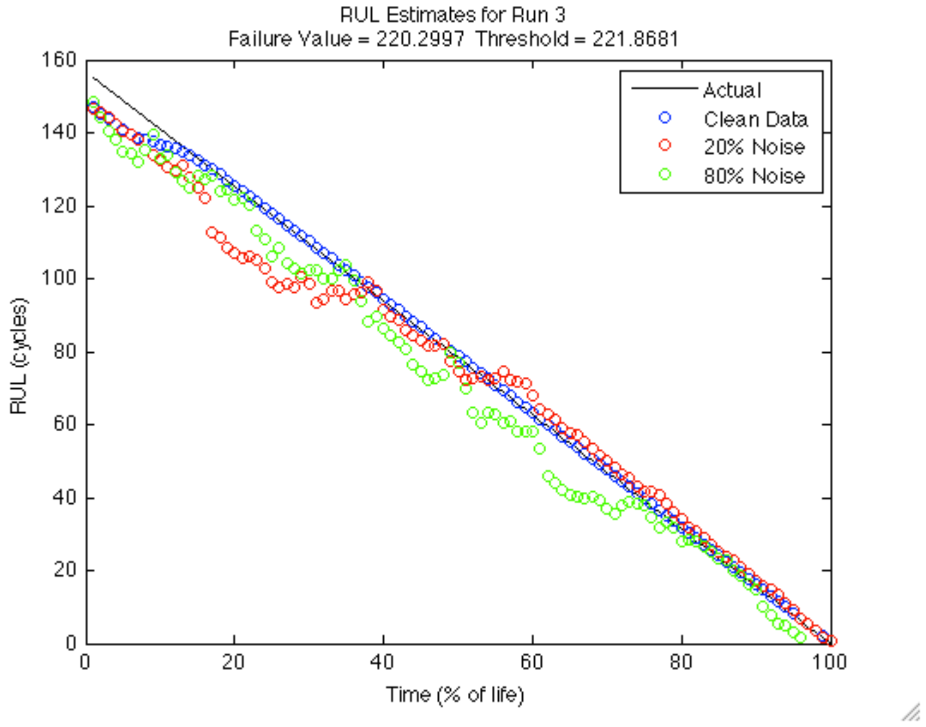

"Bad" Parameter with 20% Noise
Prognostic Parameter
Time (cycles)
300
200
100
0
-100
-200
-300
-400
-500
0
100
200
300
400
500
600

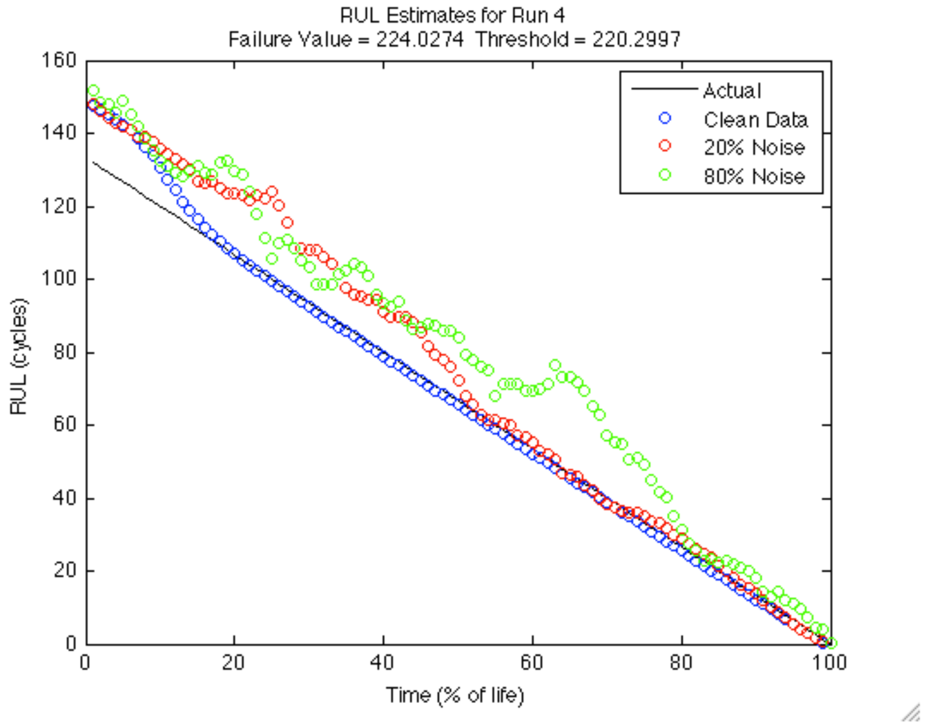

"Bad" Parameter with 80% Noise
Prognostic Parameter
Time (cycles)
500
400
300
200
100
0
-100
-200
-300
-400
-500
0
100
200
300
400
500
600

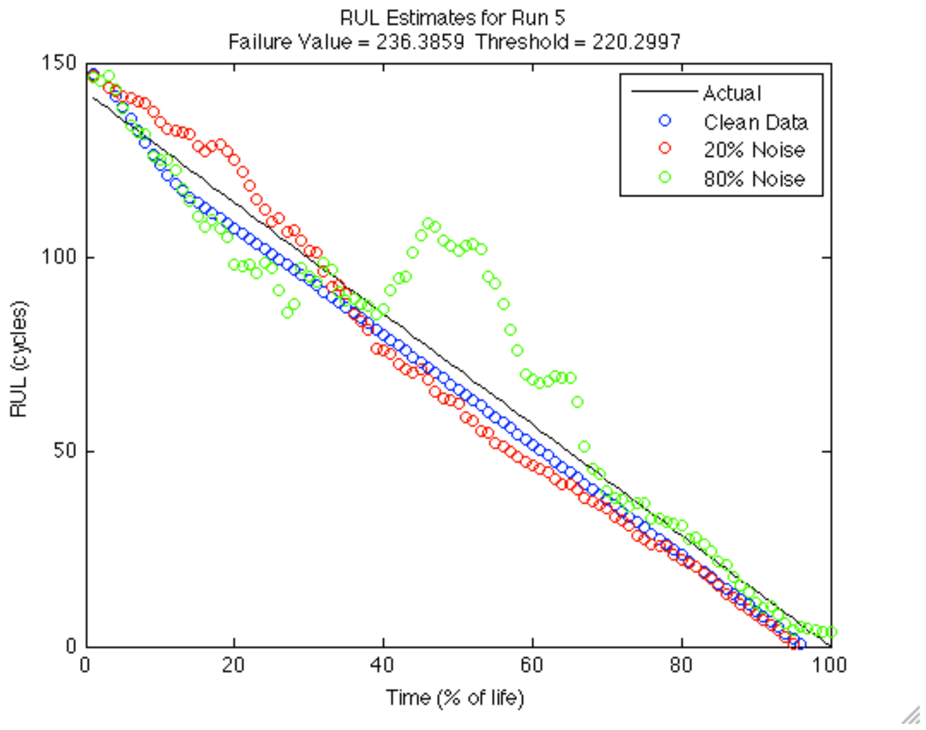
A.2 The Effect of Noise on Model Performance

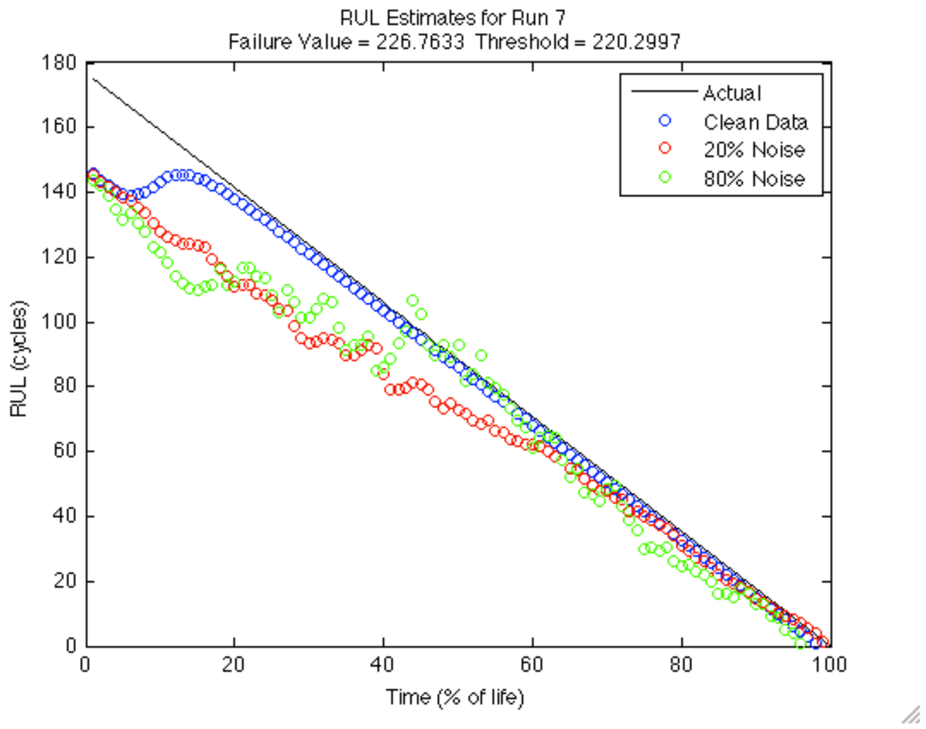


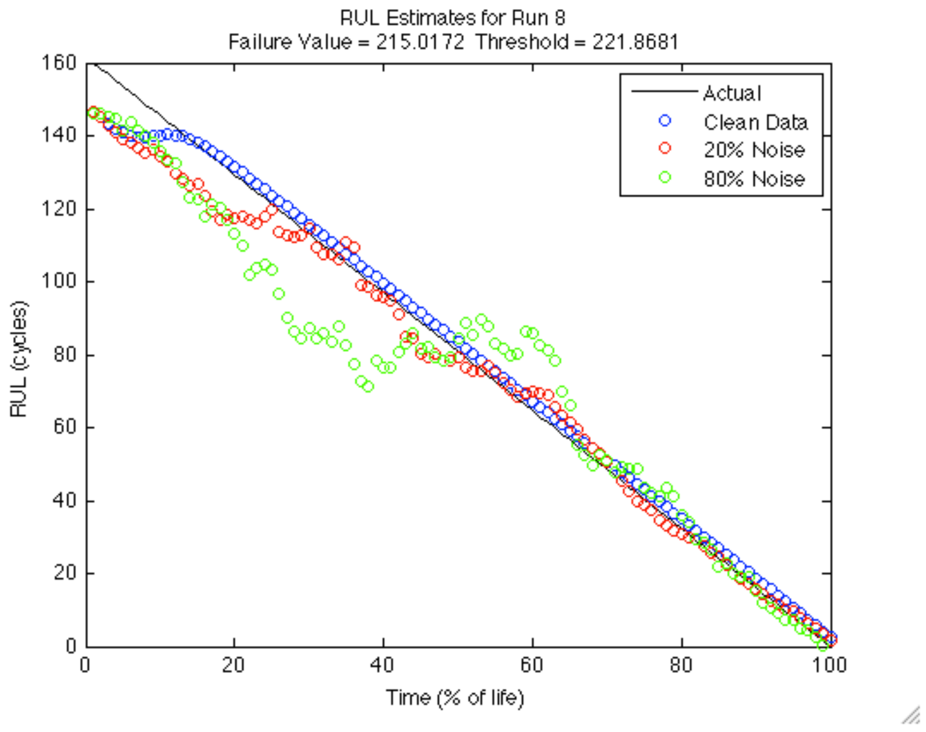



RUL Estimates for Run 3
Failure Value = 220.2997 Threshold = 221.8681
Actual
Clean Data
20% Noise
80% Noise
RUL (cycles)
Time (% of life)
0
20
40
60
80
100
120
140
160

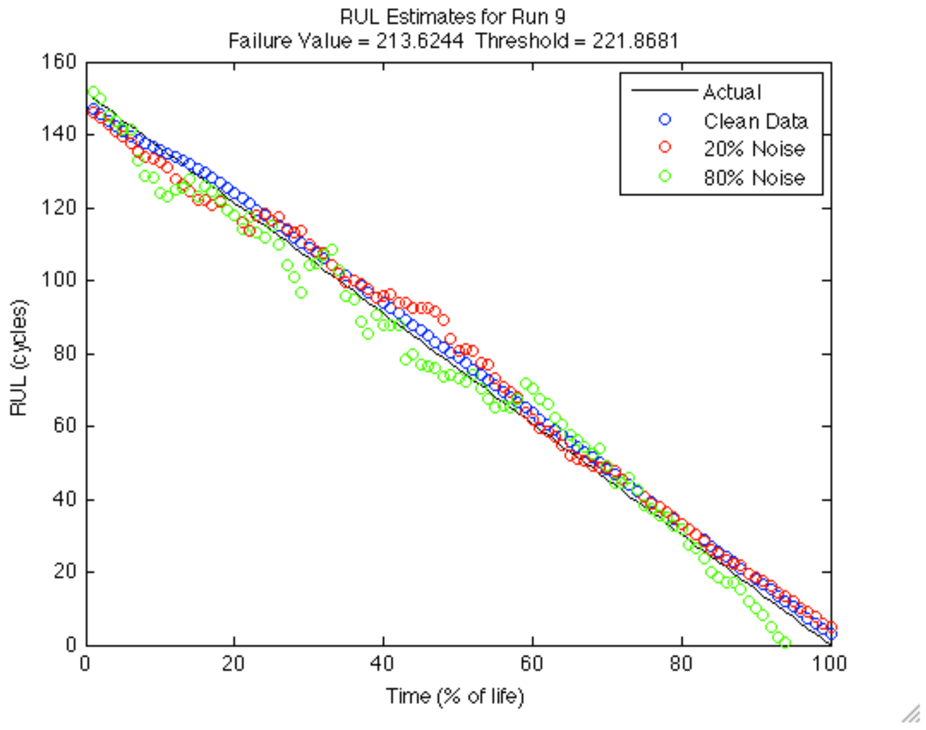

RUL Estimates for Run 4
Failure Value = 224.0274 Threshold = 220.2997
Actual
Clean Data
20% Noise
80% Noise
RUL (cycles)
Time (% of life)
0
20
40
60
80
100
120
140
160


RUL Estimates for Run 5
Failure Value = 236.3859 Threshold = 220.2997
Actual
Clean Data
20% Noise
80% Noise
RUL (cycles)
Time (% of life)
150
100
50
0
0
20
40
60
80
100

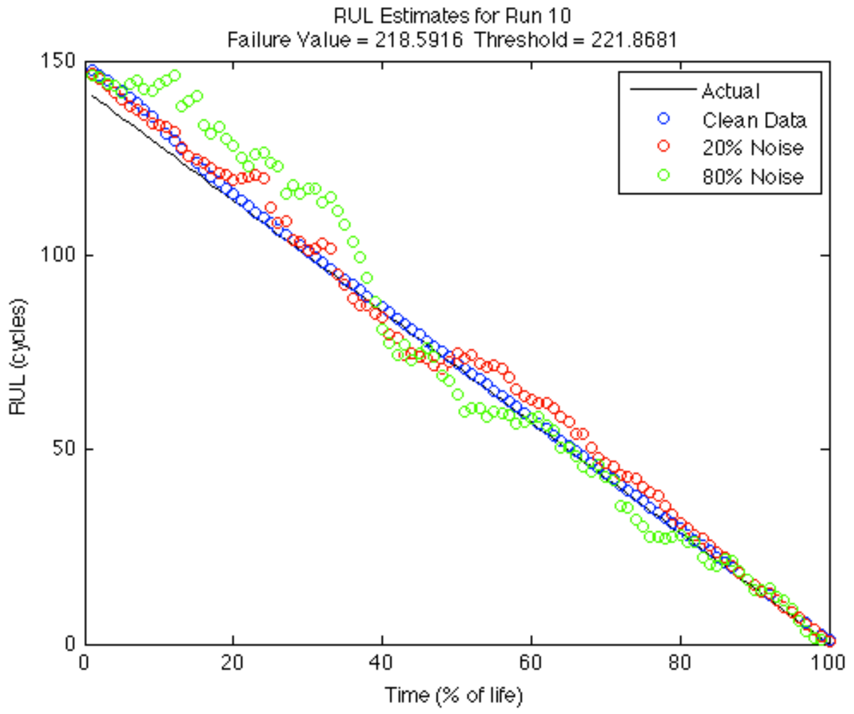

RUL Estimates for Run 7
Failure Value = 226.7633 Threshold = 220.2997
Actual
Clean Data
20% Noise
80% Noise
RUL (cycles)
Time (% of life)
180
160
140
120
100
80
60
40
20
0
0
20
40
60
80
100

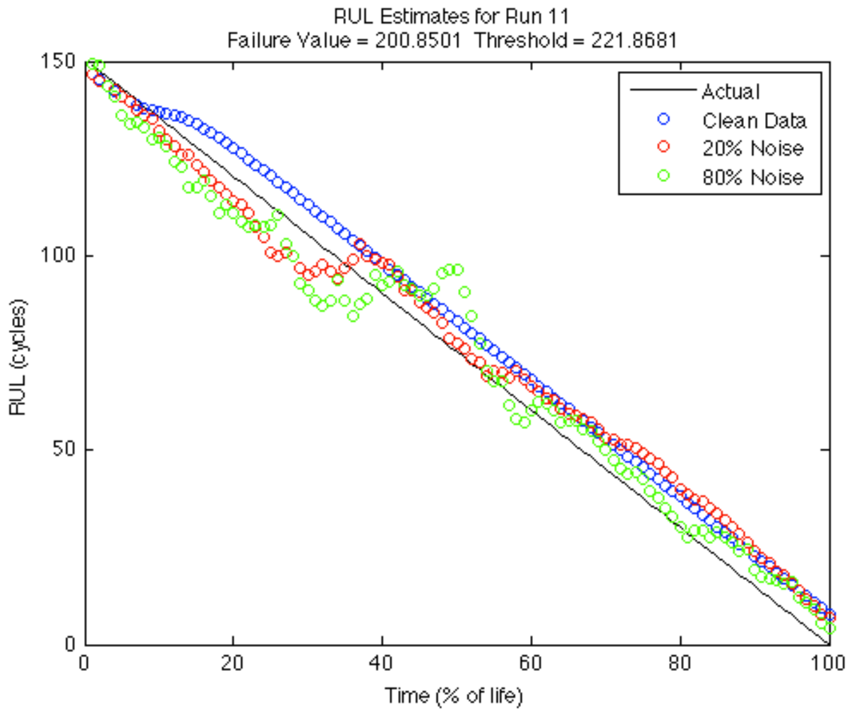


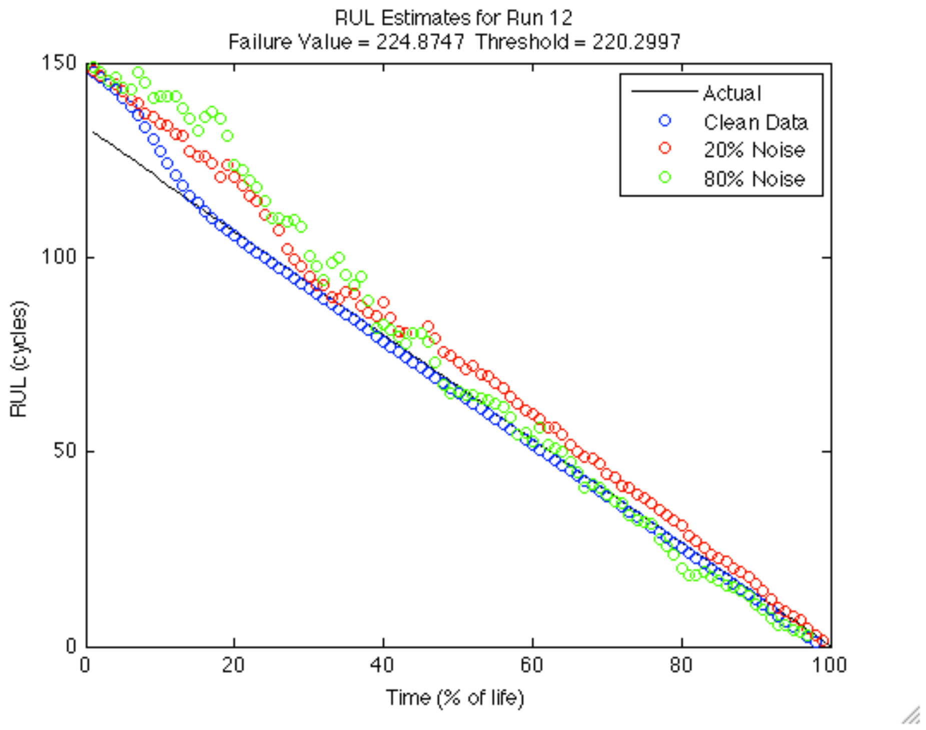
RUL Estimates for Run 8
Failure Value = 215.0172 Threshold = 221.8681
Actual
Clean Data
20% Noise
80% Noise
RUL (cycles)
Time (% of life)

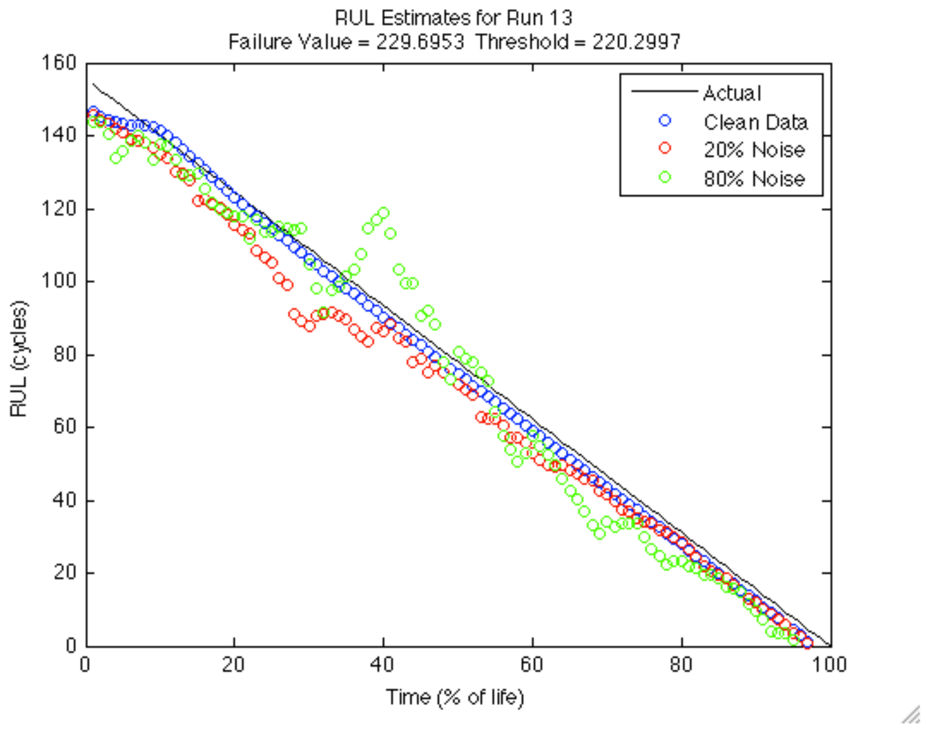


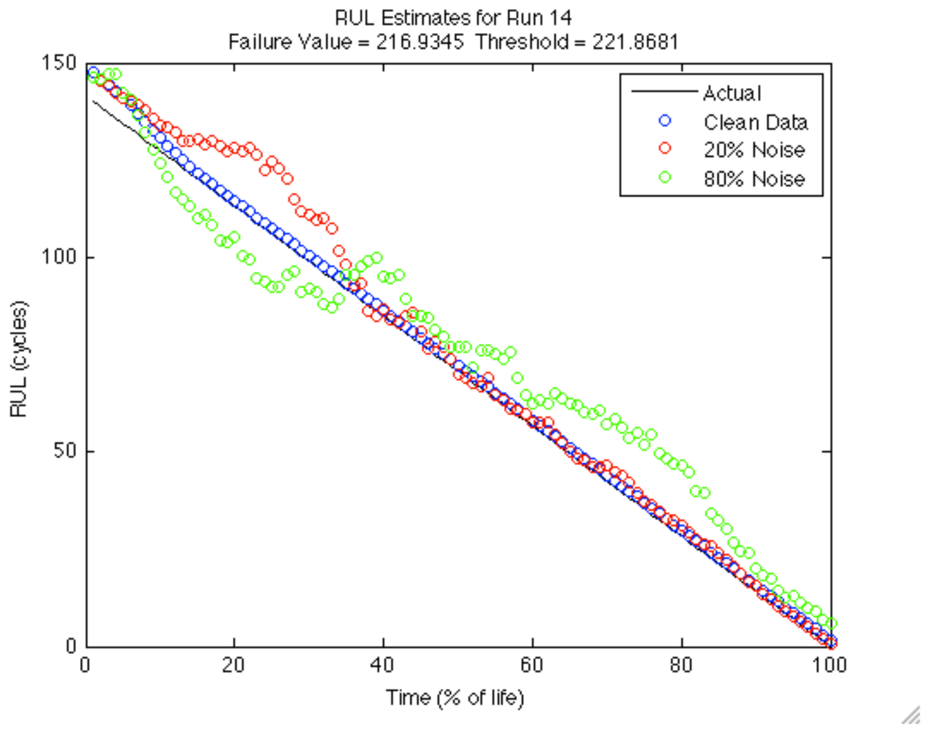
RUL Estimates for Run 9
Failure Value = 213.6244 Threshold = 221.8681
Actual
Clean Data
20% Noise
80% Noise
RUL (cycles)
Time (% of life)


RUL Estimates for Run 10
Failure Value = 218.5916 Threshold = 221.8681
RUL (cycles)
Time (% of life)
Actual
Clean Data
20% Noise
80% Noise
0
50
100
150
0
20
40
60
80
100

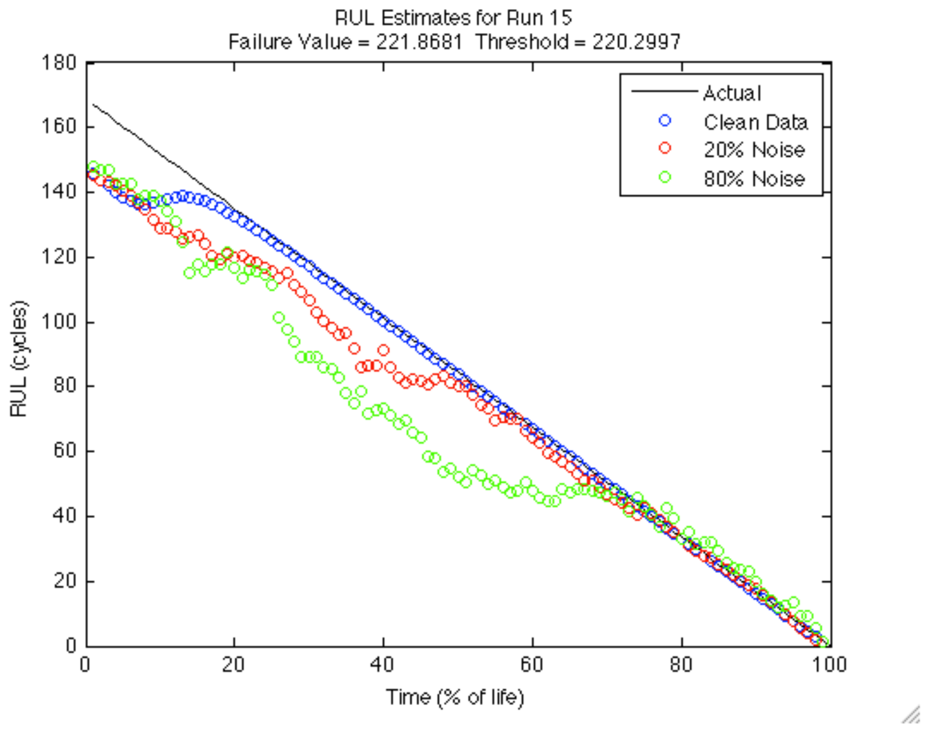

RUL Estimates for Run 11
Failure Value = 200.8501 Threshold = 221.8681
RUL (cycles)
Time (% of life)
Actual
Clean Data
20% Noise
80% Noise
0
50
100
150
0
20
40
60
80
100


RUL Estimates for Run 12
Failure Value = 224.8747 Threshold = 220.2997
Actual
Clean Data
20% Noise
80% Noise
RUL (cycles)
Time (% of life)
150
100
50
0
0
20
40
60
80
100

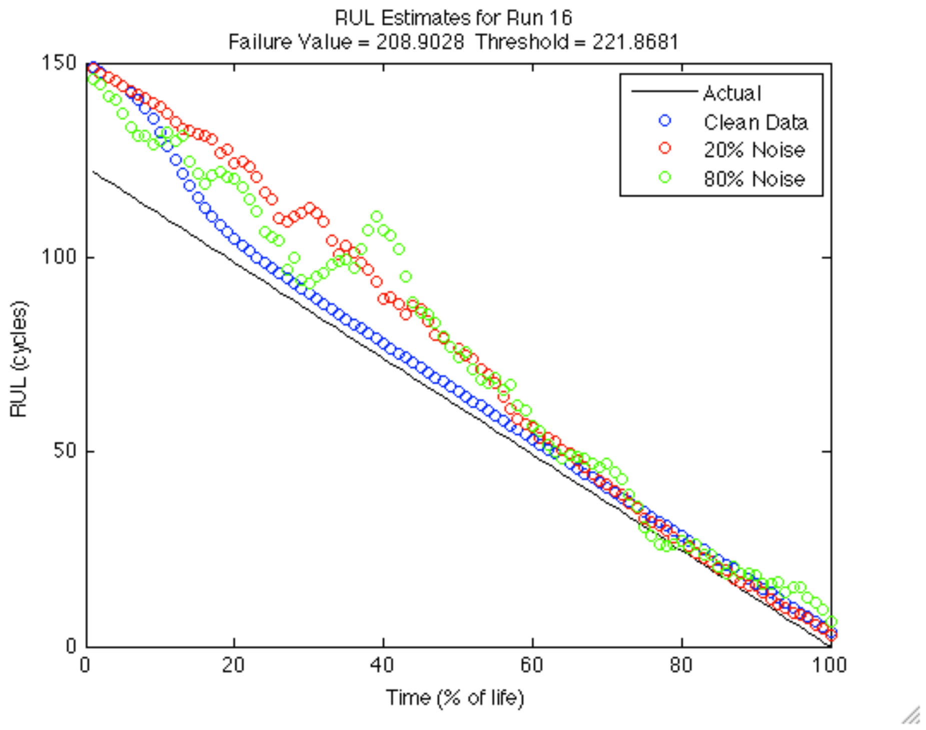

RUL Estimates for Run 13
Failure Value = 229.6953 Threshold = 220.2997
Actual
Clean Data
20% Noise
80% Noise
RUL (cycles)
Time (% of life)
160
140
120
100
80
60
40
20
0
0
20
40
60
80
100

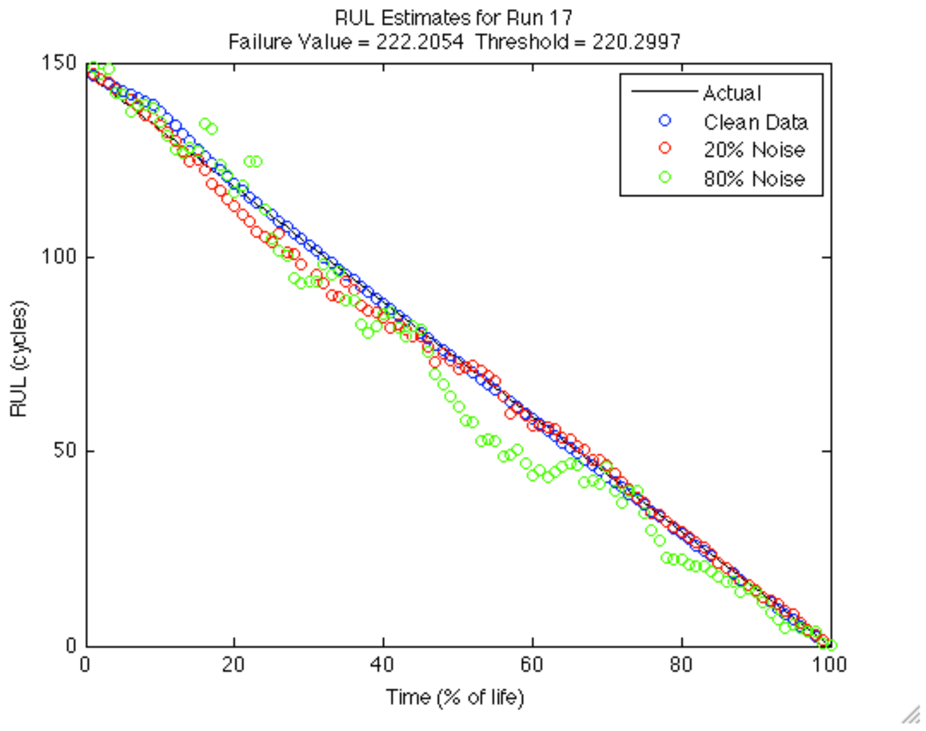


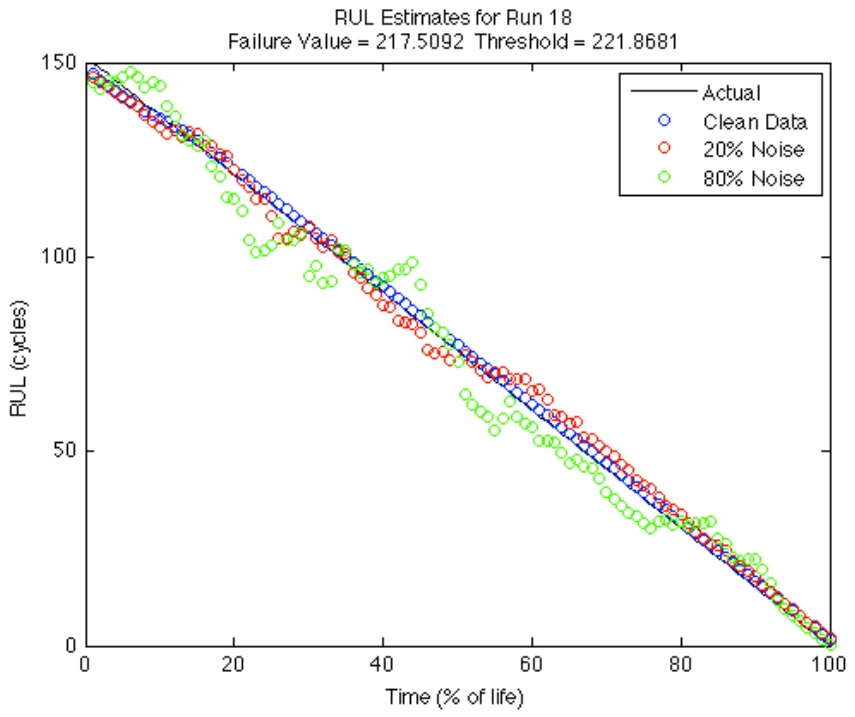
RUL Estimates for Run 14
Failure Value = 216.9345 Threshold = 221.8681
Actual
Clean Data
20% Noise
80% Noise
RUL (cycles)
Time (% of life)
150
100
50
0
0
20
40
60
80
100

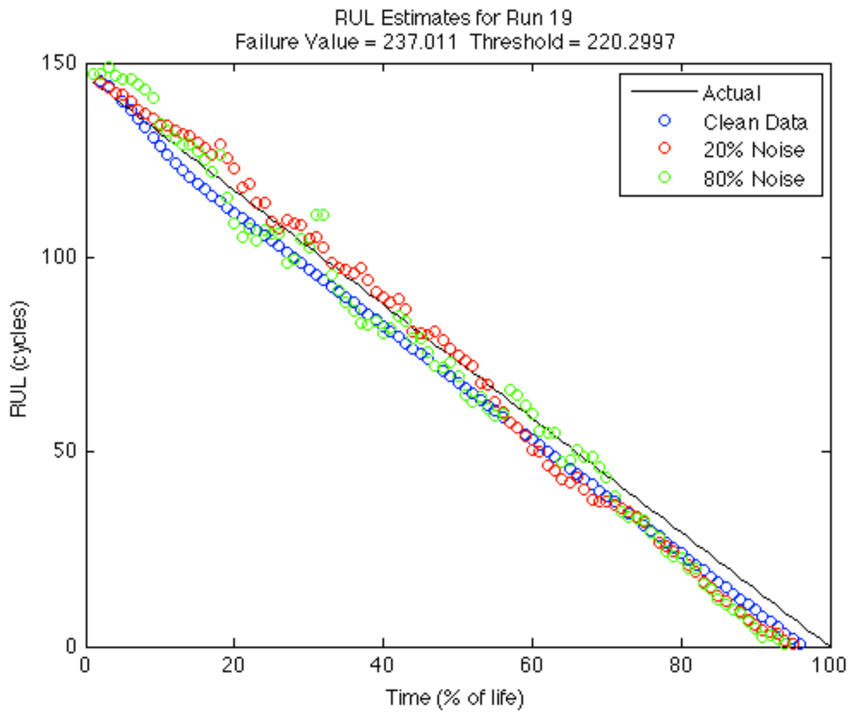


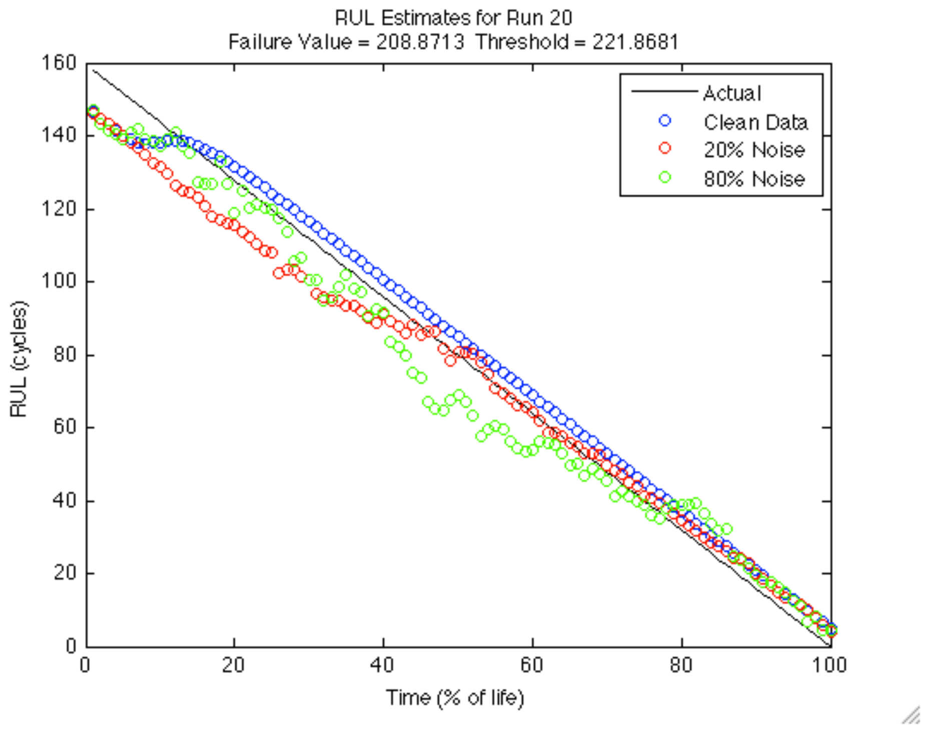
RUL Estimates for Run 15
Failure Value = 221.8681 Threshold = 220.2997
Actual
Clean Data
20% Noise
80% Noise
RUL (cycles)
Time (% of life)
180
160
140
120
100
80
60
40
20
0
0
20
40
60
80
100


RUL Estimates for Run 16
Failure Value = 208.9028 Threshold = 221.8681
Actual
Clean Data
20% Noise
80% Noise
RUL (cycles)
Time (% of life)
0
50
100
150
0
20
40
60
80
100


RUL Estimates for Run 17
Failure Value = 222.2054 Threshold = 220.2997
Actual
Clean Data
20% Noise
80% Noise
RUL (cycles)
Time (% of life)
0
50
100
150
0
20
40
60
80
100

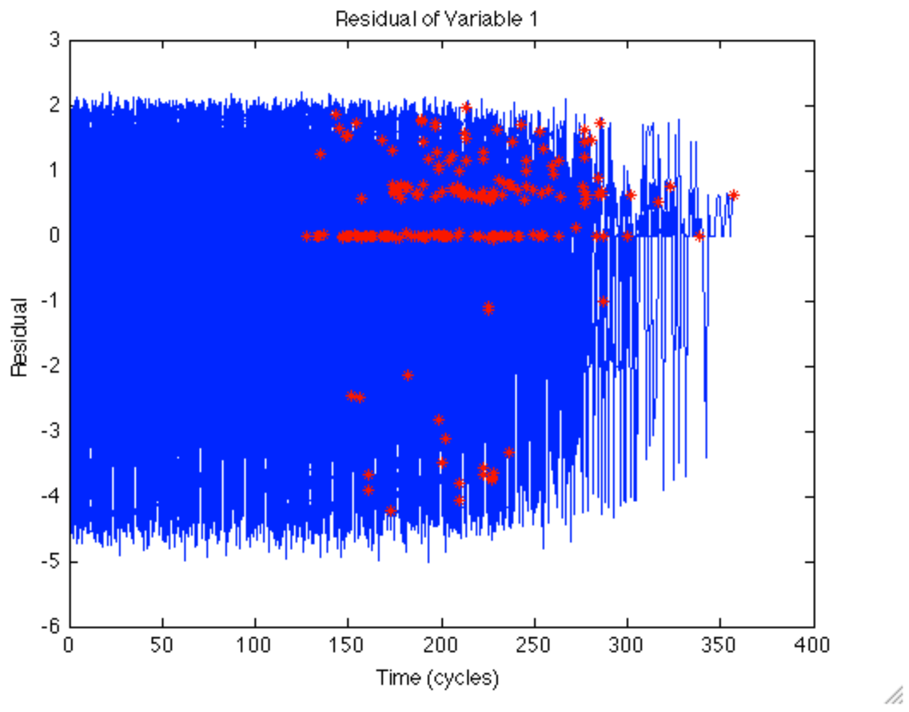

RUL Estimates for Run 18
Failure Value = 217.5092 Threshold = 221.8681
RUL (cycles)
Time (% of life)
Actual
Clean Data
20% Noise
80% Noise
0
50
100
150
0
20
40
60
80
100

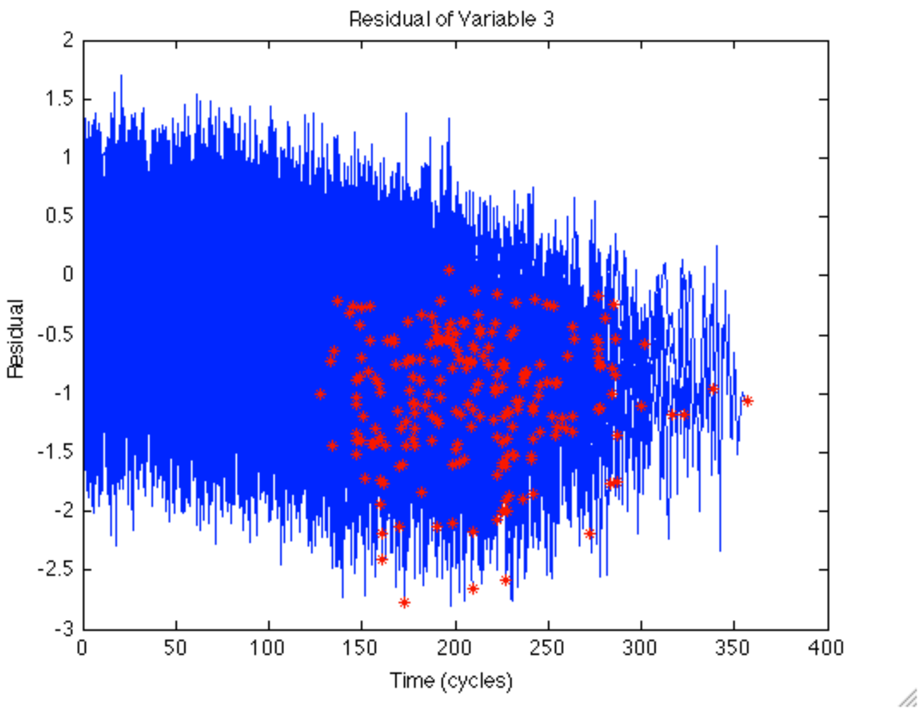

RUL Estimates for Run 19
Failure Value = 237.011 Threshold = 220.2997
RUL (cycles)
Time (% of life)
Actual
Clean Data
20% Noise
80% Noise
0
50
100
150
0
20
40
60
80
100

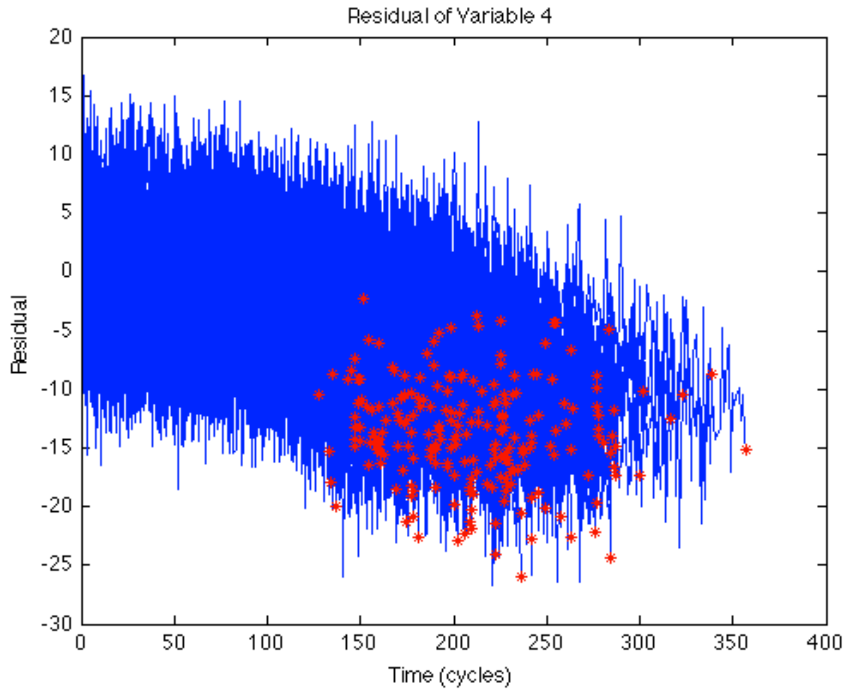


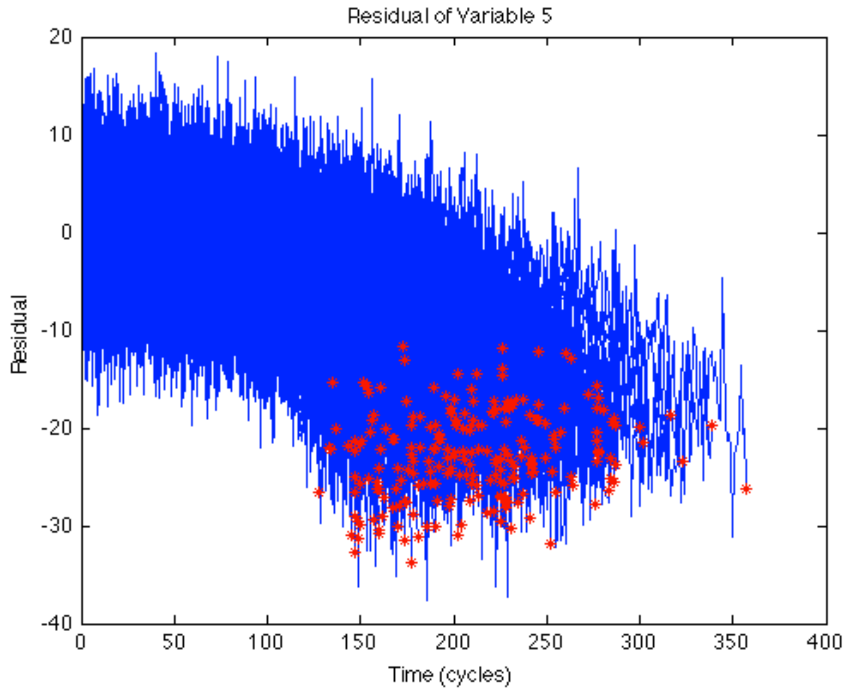
RUL Estimates for Run 20
Failure Value = 208.8713 Threshold = 221.8681
Actual
Clean Data
20% Noise
80% Noise
RUL (cycles)
Time (% of life)
0
20
40
60
80
100
120
140
160

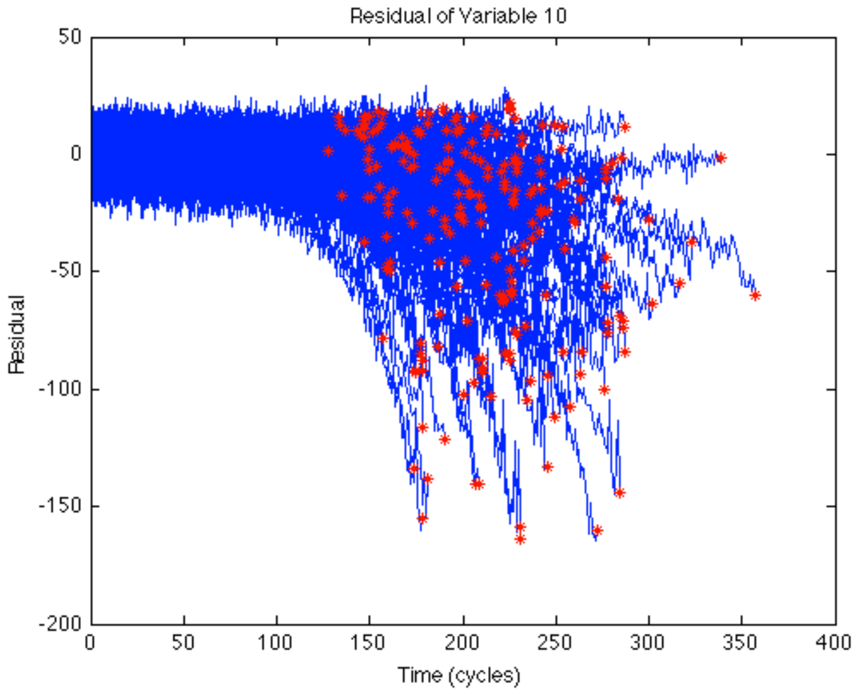
A.3 PHM Challenge Data Monitoring System Residuals

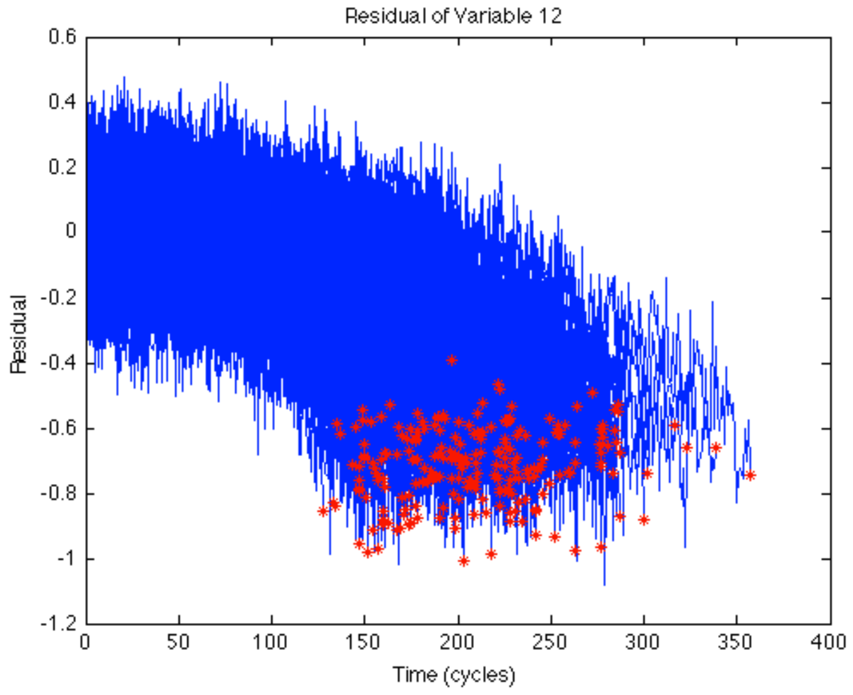


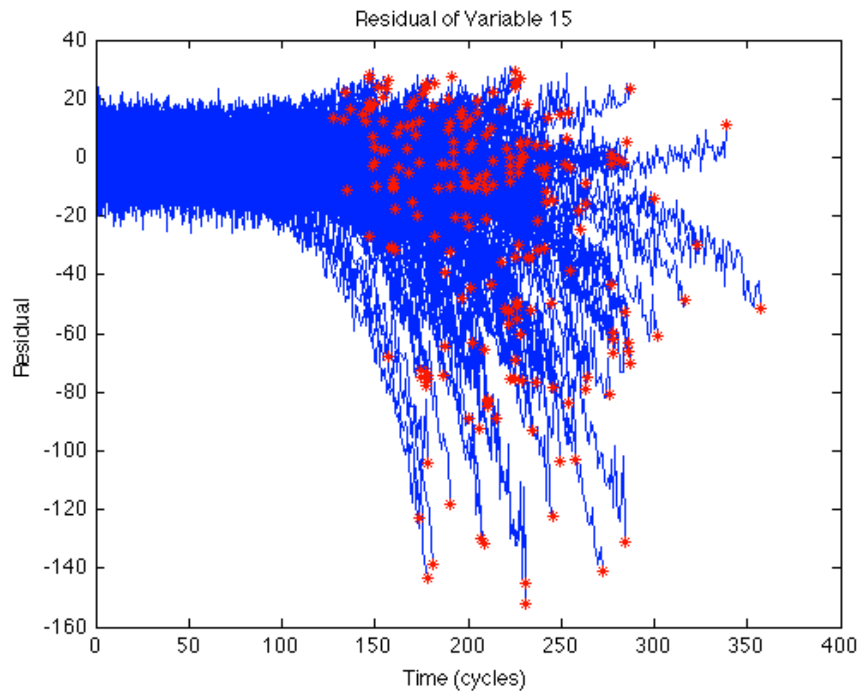


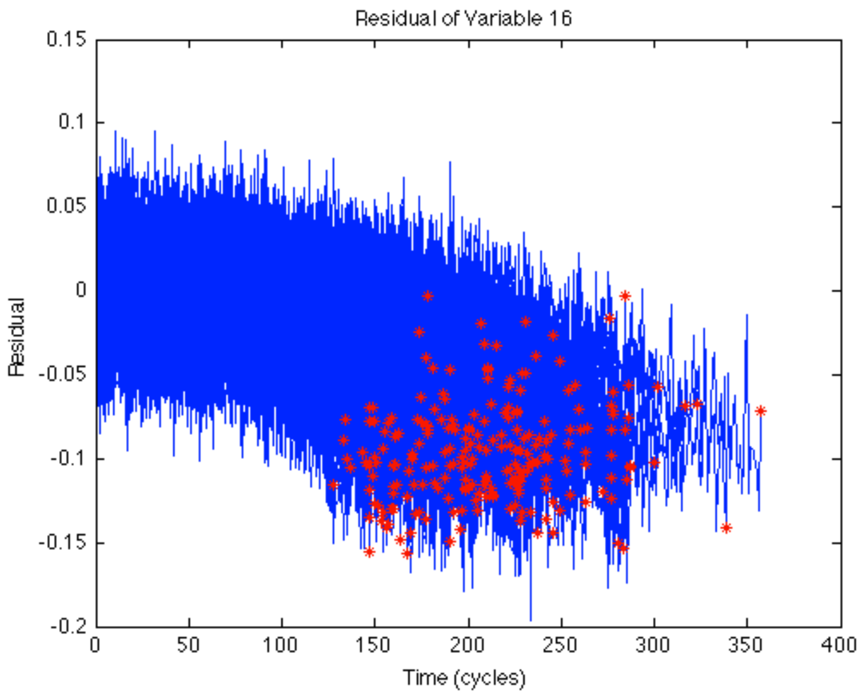

Residual of Variable 4
Residual
Time (cycles)
20
15
10
5
0
-5
-10
-15
-20
-25
-30
0
50
100
150
200
250
300
350
400

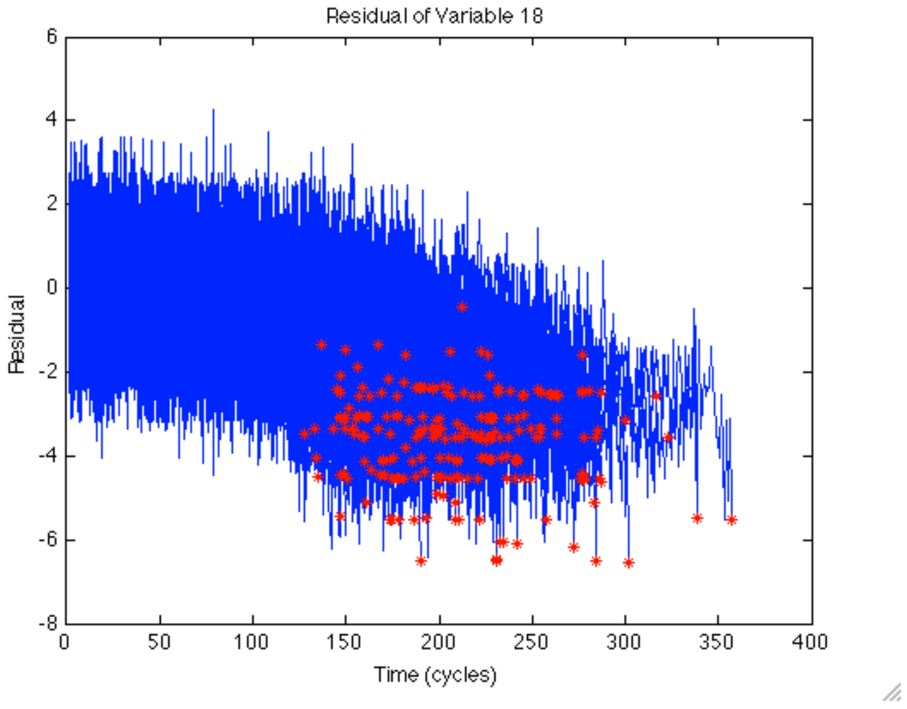

Residual of Variable 5
Residual
Time (cycles)
20
10
0
-10
-20
-30
-40
0
50
100
150
200
250
300
350
400

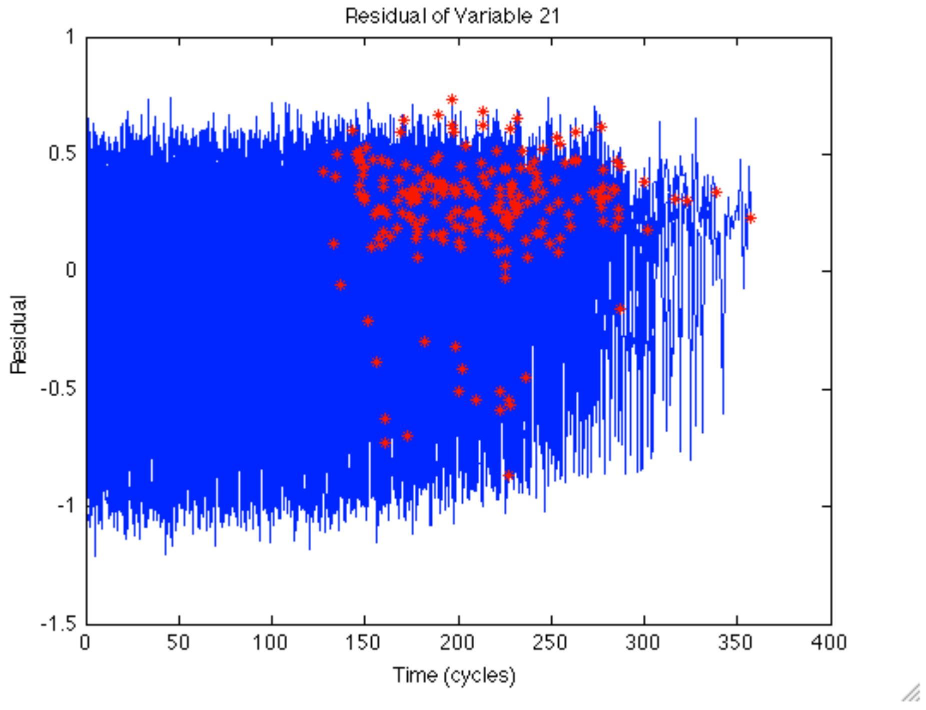

Residual of Variable 10
Residual
Time (cycles)
50
0
-50
-100
-150
-200
0
50
100
150
200
250
300
350
400

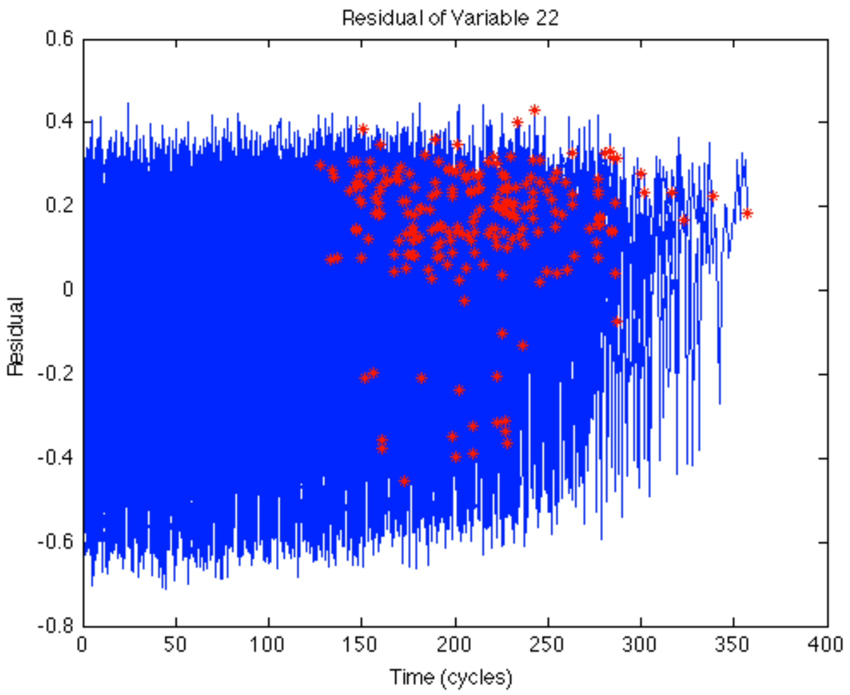

Residual of Variable 12
Residual
Time (cycles)
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1
-1.2
0
50
100
150
200
250
300
350
400


Residual of Variable 15
Residual
Time (cycles)
40
20
0
-20
-40
-60
-80
-100
-120
-140
-160
0
50
100
150
200
250
300
350
400

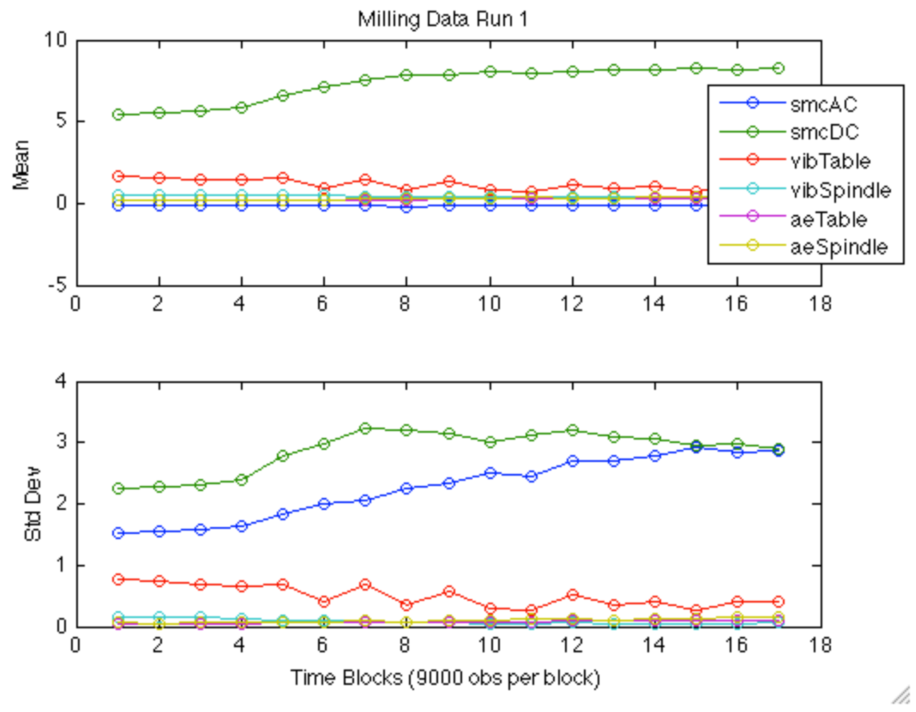

Residual of Variable 16
Residual
Time (cycles)
0.15
0.1
0.05
0
-0.05
-0.1
-0.15
-0.2
0
50
100
150
200
250
300
350
400

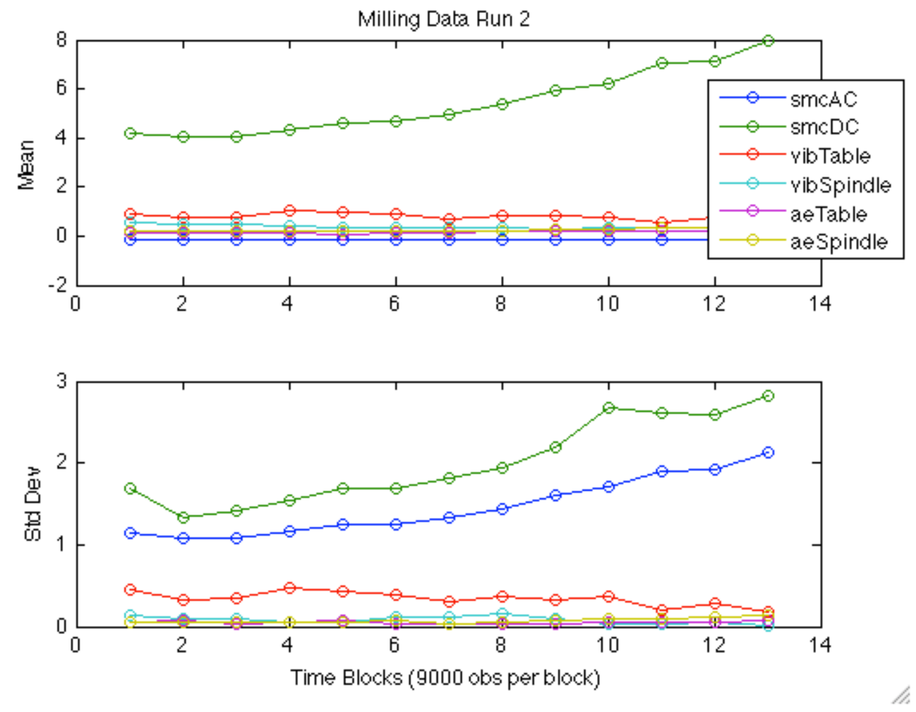

Residual of Variable 18
Residual
Time (cycles)
6
4
2
0
-2
-4
-6
-8
0
50
100
150
200
250
300
350
400

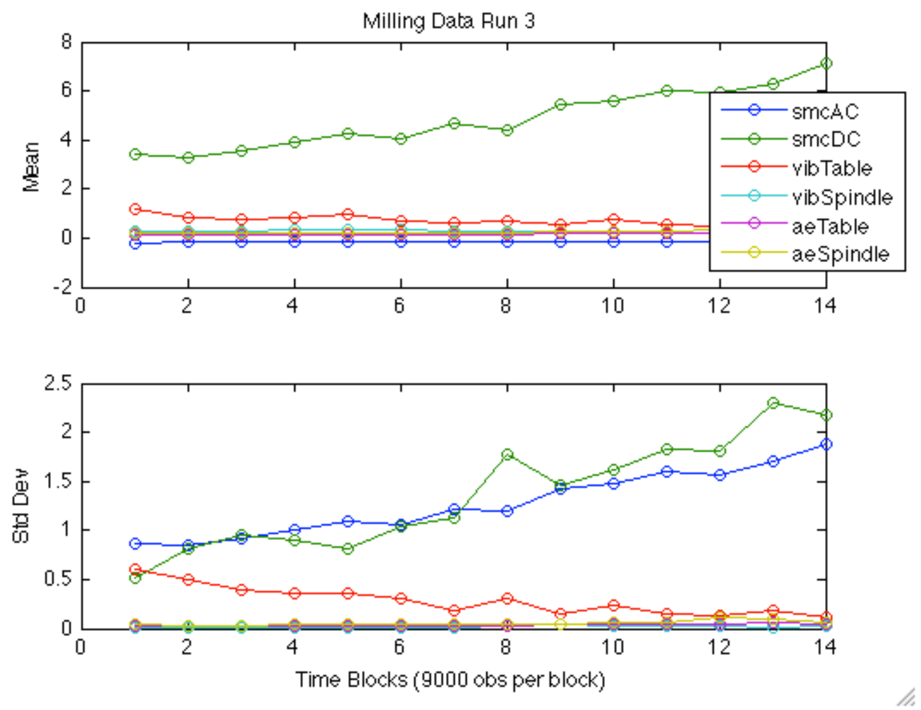

Residual of Variable 21
Residual
Time (cycles)
1
0.5
0
-0.5
-1
-1.5
0
50
100
150
200
250
300
350
400

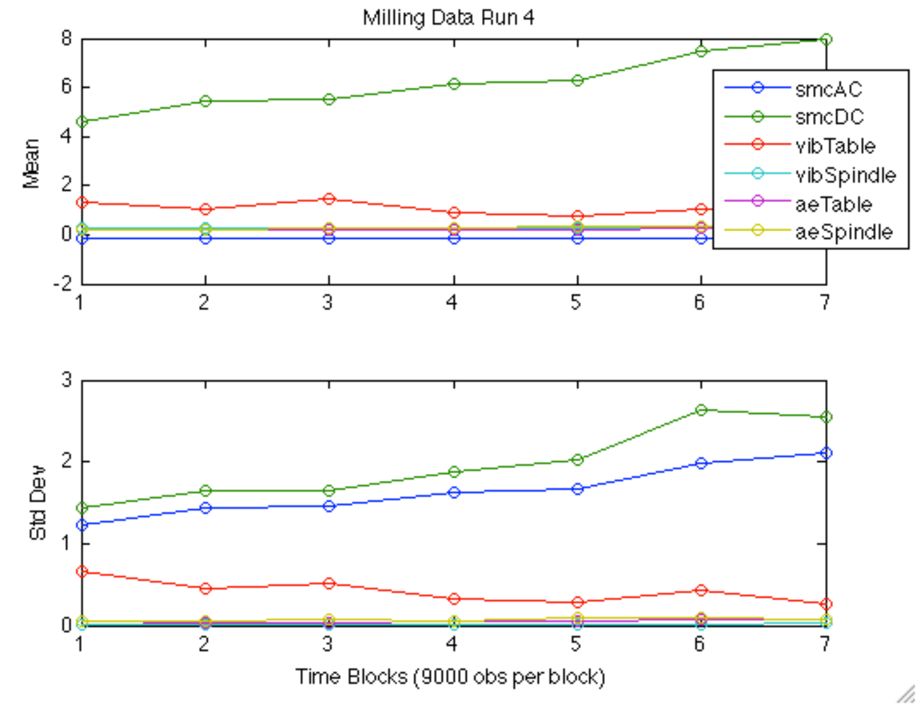

Residual of Variable 22
Residual
Time (cycles)
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
0
50
100
150
200
250
300
350
400

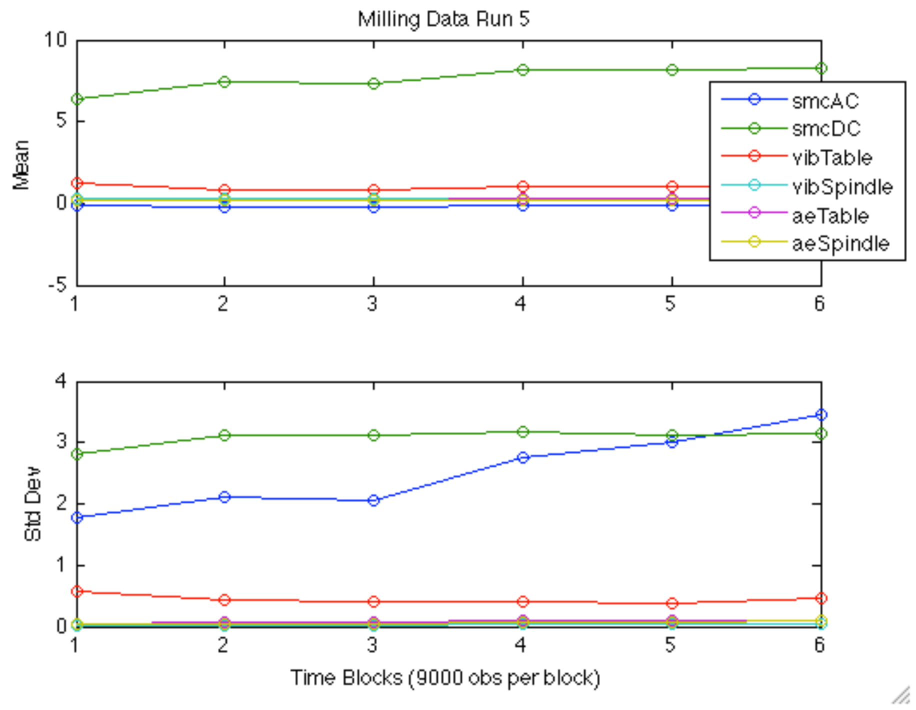
A.4 Milling Data Mean and Standard Deviation Features



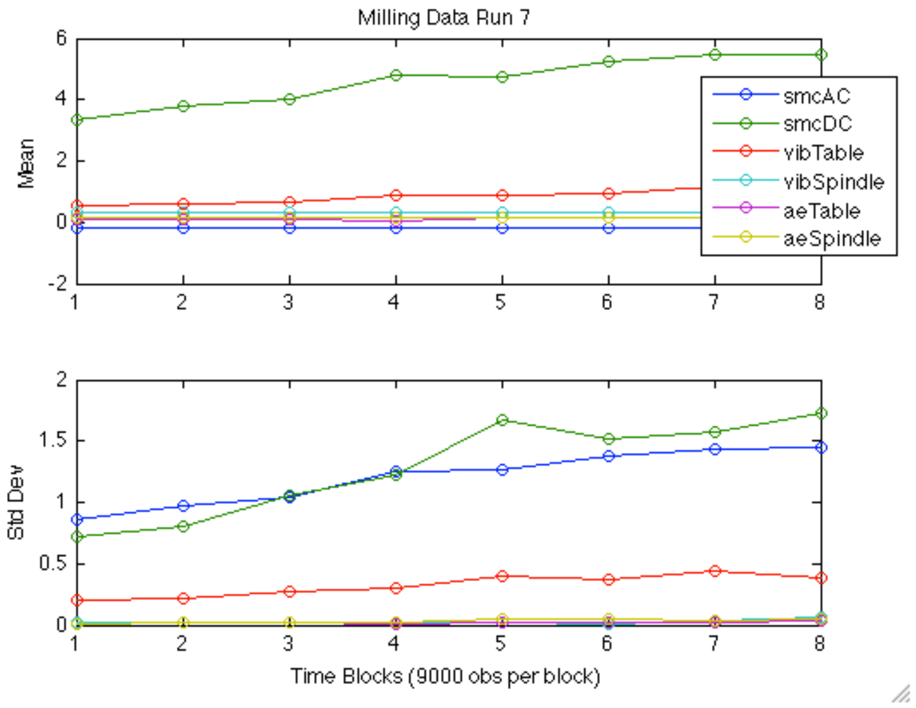



Milling Data Run 3
Mean
Std Dev
Time Blocks (9000 obs per block)
smcAC
smcDC
vibTable
vibSpindle
aeTable
aeSpindle

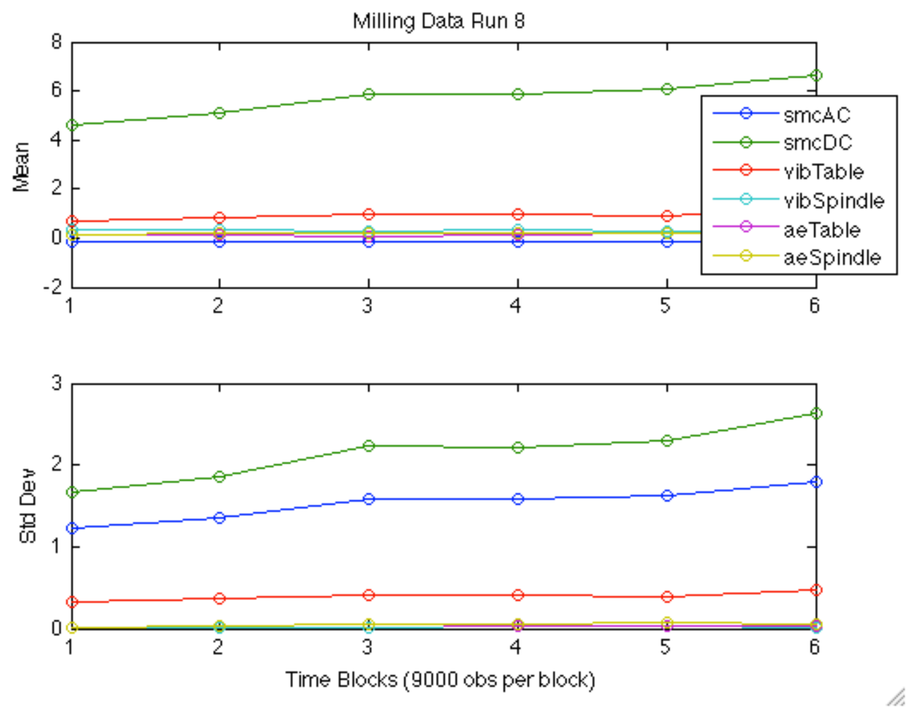

Milling Data Run 4
Mean
Std Dev
Time Blocks (9000 obs per block)
smcAC
smcDC
vibTable
vibSpindle
aeTable
aeSpindle

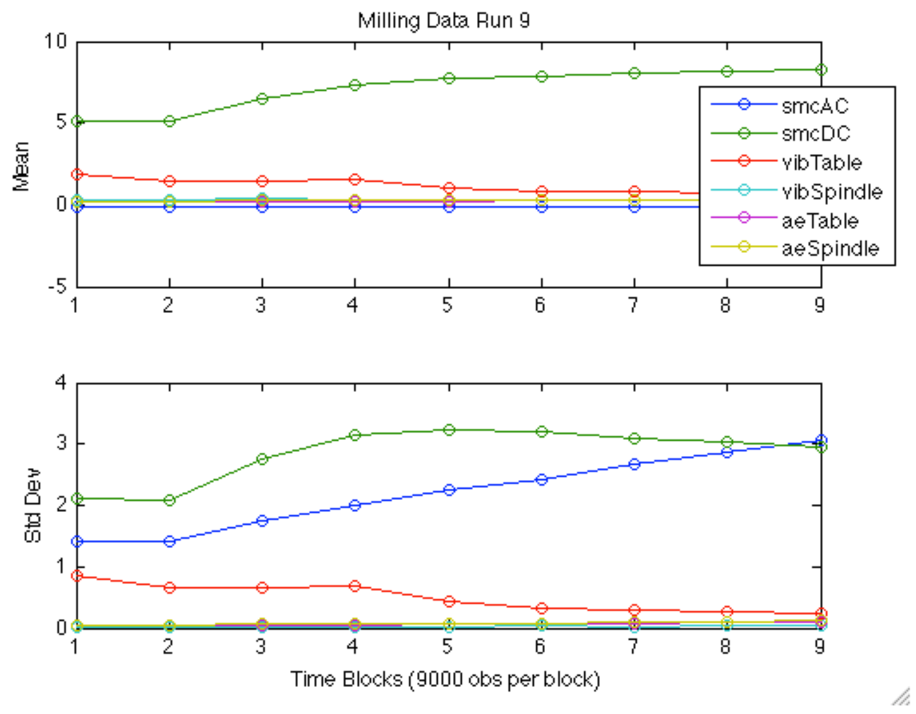


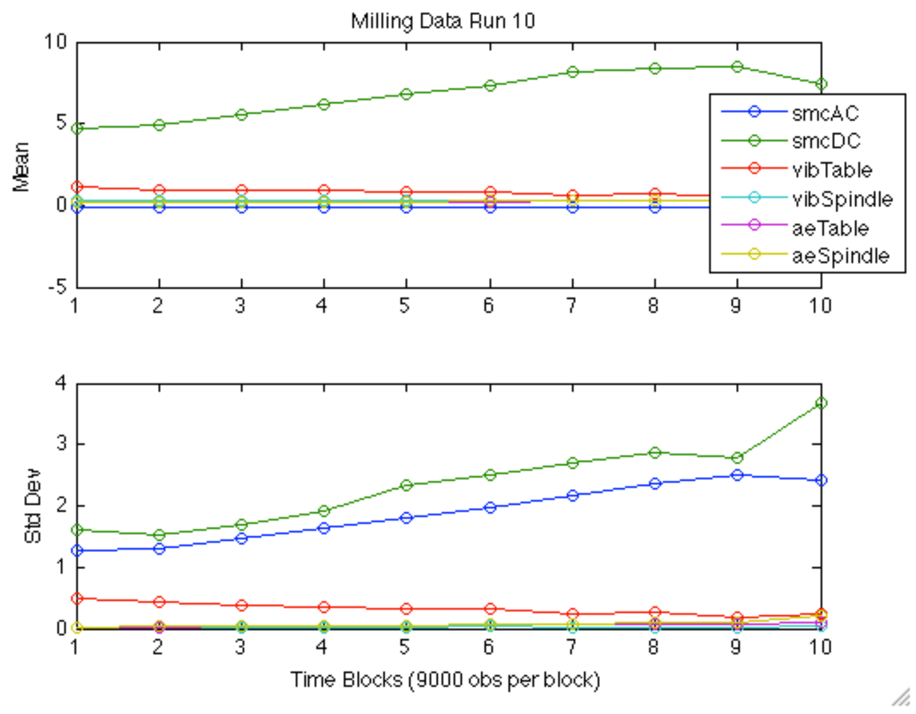
Milling Data Run 5
Mean
Std Dev
Time Blocks (9000 obs per block)
smcAC
smcDC
vibTable
vibSpindle
aeTable
aeSpindle



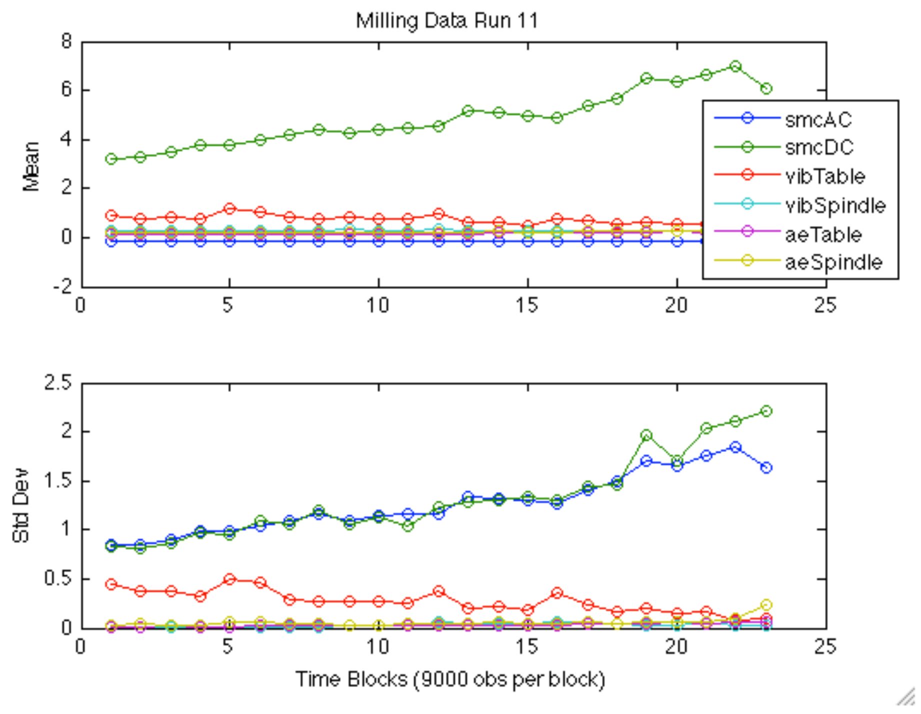
Milling Data Run 7
Mean
Std Dev
Time Blocks (9000 obs per block)
smcAC
smcDC
vibTable
vibSpindle
aeTable
aeSpindle


Milling Data Run 8
Mean
Std Dev
Time Blocks (9000 obs per block)
smcAC
smcDC
vibTable
vibSpindle
aeTable
aeSpindle

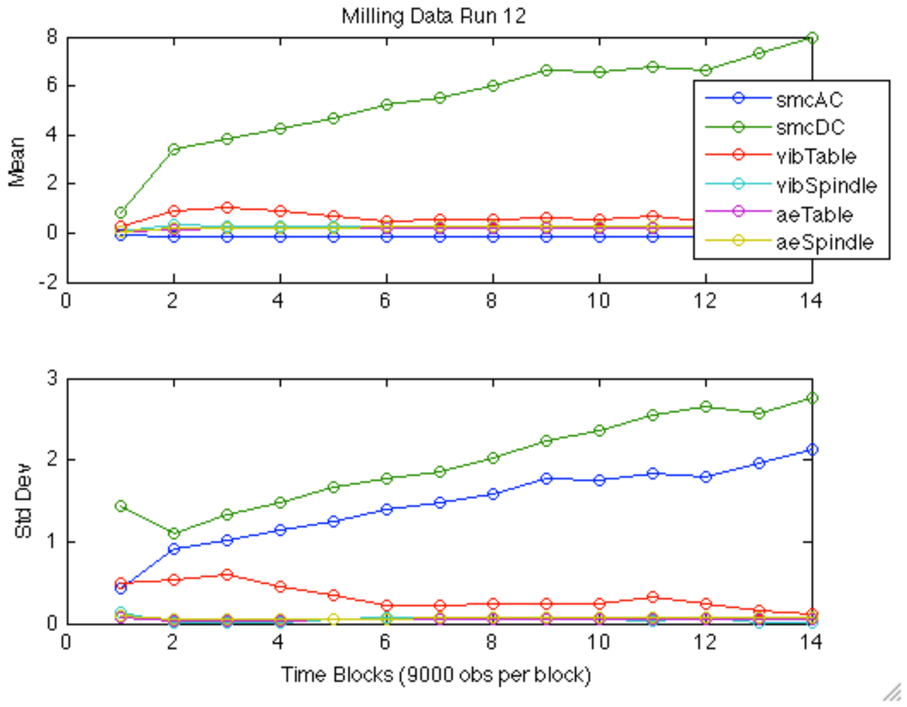

Milling Data Run 9
Mean
Std Dev
Time Blocks (9000 obs per block)
smcAC
smcDC
vibTable
vibSpindle
aeTable
aeSpindle

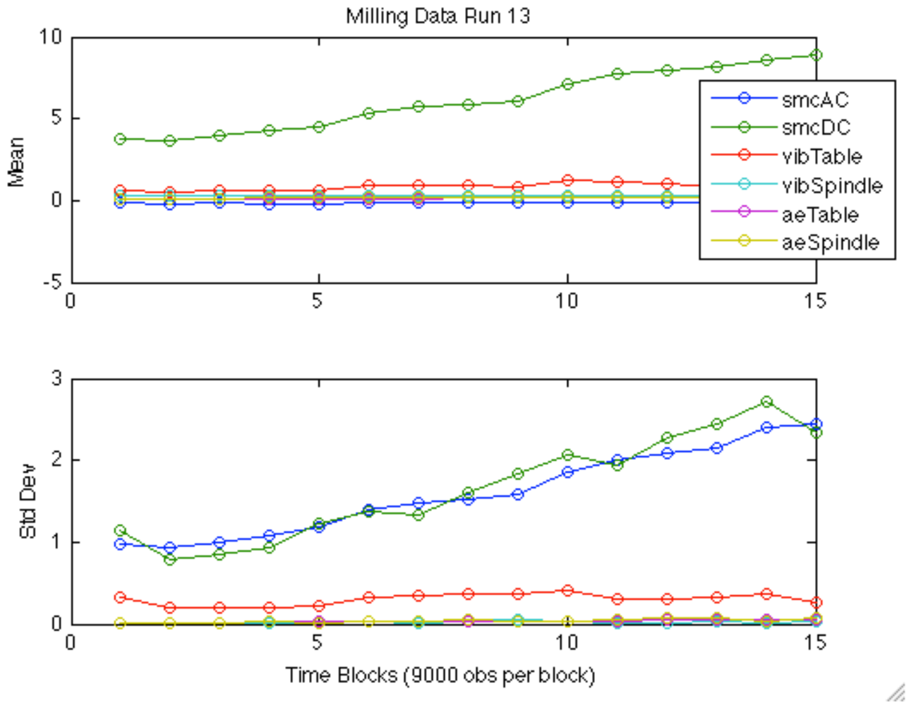


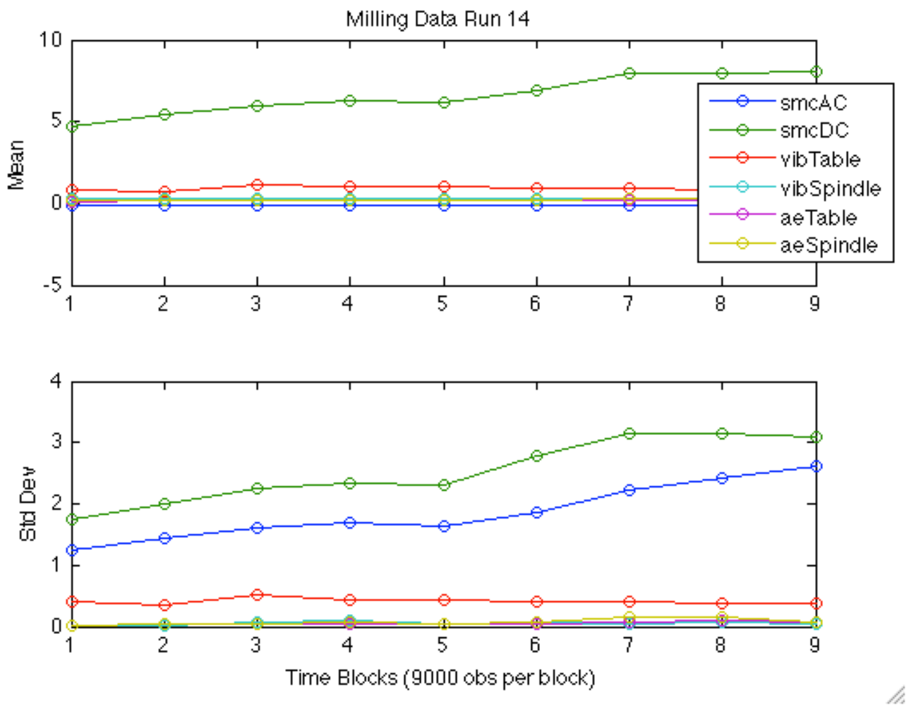
Milling Data Run 10
Mean
Std Dev
Time Blocks (9000 obs per block)
smcAC
smcDC
vibTable
vibSpindle
aeTable
aeSpindle



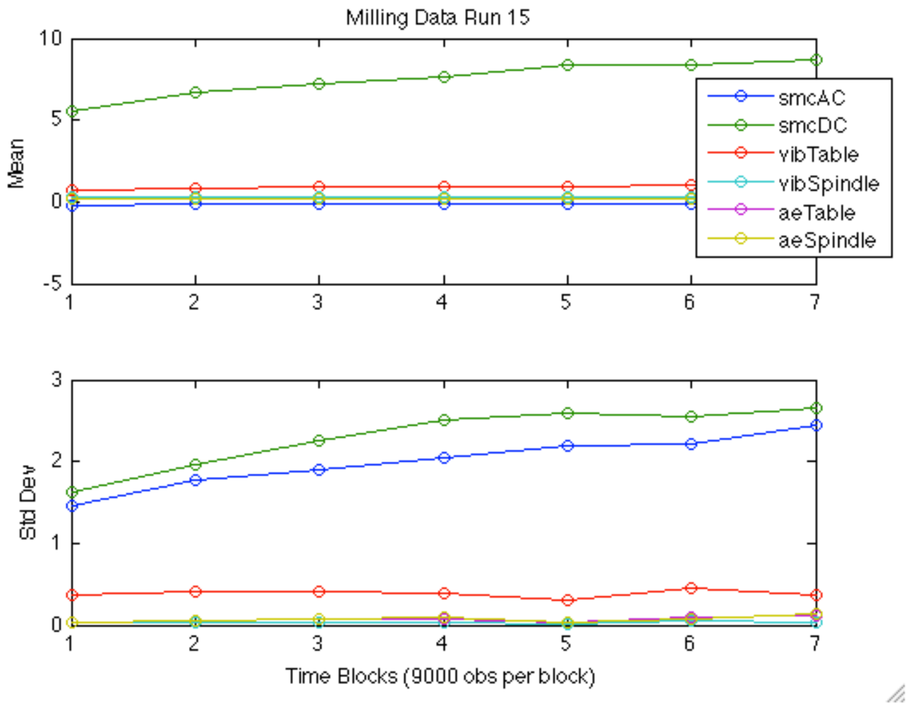
Milling Data Run 11
Mean
Std Dev
Time Blocks (9000 obs per block)
smcAC
smcDC
vibTable
vibSpindle
aeTable
aeSpindle


Milling Data Run 12
Mean
Std Dev
Time Blocks (9000 obs per block)
smcAC
smcDC
vibTable
vibSpindle
aeTable
aeSpindle

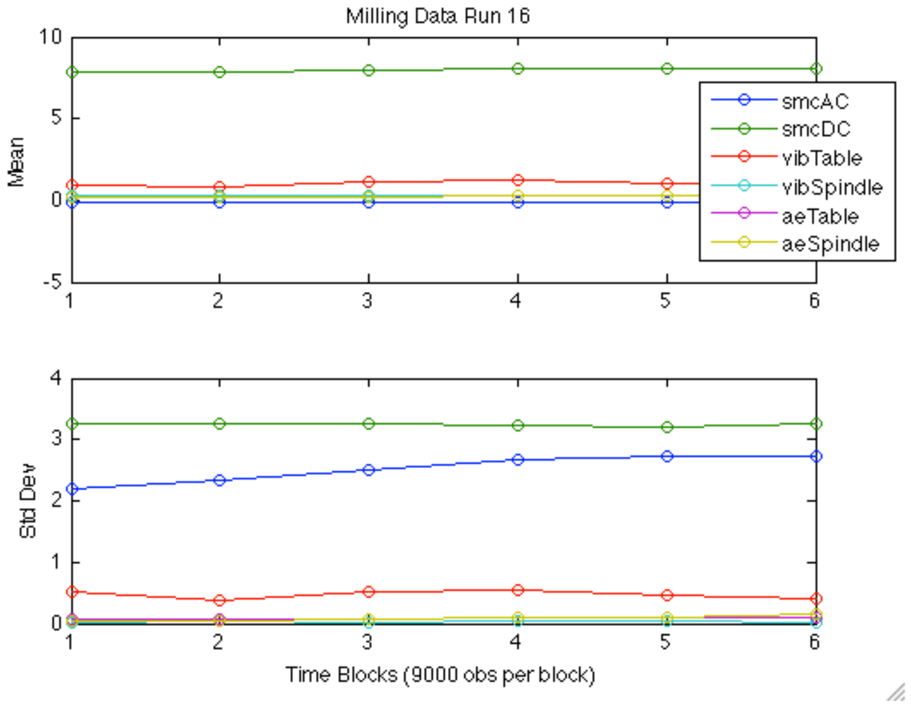

Milling Data Run 13
Mean
Std Dev
Time Blocks (9000 obs per block)
smcAC
smcDC
vibTable
vibSpindle
aeTable
aeSpindle



Milling Data Run 14
Mean
Std Dev
Time Blocks (9000 obs per block)
smcAC
smcDC
vibTable
vibSpindle
aeTable
aeSpindle



Milling Data Run 15
Mean
Std Dev
Time Blocks (9000 obs per block)
smcAC
smcDC
vibTable
vibSpindle
aeTable
aeSpindle



Milling Data Run 16
Mean
Std Dev
Time Blocks (9000 obs per block)
smcAC
smcDC
vibTable
vibSpindle
aeTable
aeSpindle

Appendix B: Related Published Works

Coble, J. and J. Wesley Hines, "Applying the General Path Model to Estimation of Remaining Useful Life", submitted to International Journal of Prognostics and Health Management.

Coble, J. and J.W. Hines, "Identifying Optimal Prognostic Parameters from Data: A Genetic Algorithms Approach," Annual Conference of the Prognostics and Health Management Society PHM 2009, San Diego, CA, Oct. 2009.

Coble, J. and J.W. Hines, "Development of a MATLAB-based Process and Equipment Prognostics Toolbox," 2009 Integrated Structural Health Management (ISHM) Conference, Covington, KY, August, 2009.

Coble, J. and J.W. Hines, "Fusing Data Sources for Optimal Prognostic Parameter Selection," Invited paper in "Best of NPIC/HMIT" section, 2009 American Nuclear Society National Meeting, Atlanta, GA, June, 2009.

Coble, J. and J.W. Hines, "Development of a Prognostics Toolbox with Application to GPS Degradation Data," 2009 Conference of the Society for Machinery Failure Prevention Technology MFPT 2009, Dayton, Ohio, April, 2009.

Coble, J. and J.W. Hines, "Fusing Data Sources for Optimal Prognostic Parameter Selection," Sixth American Nuclear Society International Topical Meeting on Nuclear Plant Instrumentation, Control, and Human-Machine Interface Technologies NPIC&HMIT 2009, Knoxville, Tennessee, April, 2009.

Coble, J. and J.W. Hines, "Metrics for Optimal Prognostic Parameter Selection," Presentation to Propulsion Safety and Affordable Readiness Conference P-SAR 2009, Myrtle Beach, SC, March, 2009.

Coble, J. and J.W. Hines, "Prognostic Algorithm Categorization with Application to the PHM Challenge Problem", Prognostics and Health Management Conference PHM 2008, Denver, Colorado, Oct, 2008.

Garvey, J. and J.W. Hines, "Merging Data Sources for Predicting Remaining Useful Life", Maintenance and Reliability Conference MARCON 2008, Knoxville, Tennessee, May, 2008.

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

Appendix C: The Process and Equipment Prognostics Toolbox

C.1 List of Functions

The Process and Equipment Prognostics (PEP) Toolbox is designed to work with the previously developed Process and Equipment Monitoring (PEM) Toolbox to develop a suite of health management tools. The PEP includes functionality to perform the three types of prognostic models described previously in this report. Each of the functions currently integrated in the PEP Toolbox are described below, categorized by function purpose. Code, including help headers, for each function is included in the following pages in the same order. This list is continuously being updated as the PEP toolbox is developed and improved.

Prognostic Model Function Calls

initprog() – This function initializes a Prognostics Toolbox model structure

runprog() – This function estimates the Remaining Useful Life (RUL) using the model previously developed

Type I Models

initTypeI() – This function initializes a reliability-based prognostic model structure

runTypeI() – This function makes RUL estimates using a Type I (reliability based) prognostics model

Type II Models

initMC() – This function initializes an MC model prognostics toolbox structure

MCprobs() – This function calculates the transition probability matrix and the initial condition probability vector for a Type II Markov Chain model

fitMC() – This function determines the coefficients to map time spent in each operating condition to system degradation

runMC() – This function makes RUL estimates using a Type II MC model

initPHM() – This function initializes a Proportional Hazards Model structure

runPHM() – This function makes RUL estimates using a Proportional Hazards Model

initShock() – This function initializes a shock model prognostics toolbox structure

runShock() – This function makes RUL estimates using a Type II Shock model

Type III Models

initGPM() – This function initializes an GPM prognostic toolbox structure

fitGPM() – This function determines the best fit for a GPM prognostic model between linear, quadratic, and exponential fits

initBayes() – This function calculates the initial Bayesian prior distribution for a GPM model.

threshGPM() – This function calculates the critical failure threshold for a GPM model based on the full historic failure paths contained in param runGPM() – This function makes RUL estimates using a general path model

Parameter Identification

optparam() – This function uses Genetic Algorithms and parameter suitability metrics to identify a near-optimal prognostic parameter from data sources

paramfit() - This function determines the fitness of a candidate prognostic parameter as the weighted sum of monotonicity, prognosability, and trendability

ppmetrics() – This function characterizes the appropriateness of a prognostic parameter based on three metrics

paramgen() – This function generates a population of prognostic parameters according to the options saved in the structure param_struct

Data Preprocessing

MCdata() – This function converts operating condition data into data needed for Markov Chain Models

timestats2() – This function calculates basic time statistics for multivariate data.

C.2 Prognostic Model Function Calls

```
function model = initprog(type,varargin)

% INITPROG Initialize prognostic model structure
%       This function initializes a Prognostics Toolbox model structure
%
%       Model = INITPROG('type',...) initializes a prognostic model of the
%       kind specified by 'type'. 'type' may be set to any of the following
%       strings
%           'typeI'
%           'markovchain' (or 'mc')
%           'shock'
%           'proportionalhazards' (or 'phm')
%           'generalpath' (or 'gpm')
%
%       Model = INITPROG('typeI',TTF) initializes a Type I
%       (reliability-based) model using the failure times contained in the
%       column vector TTF
%
%       Model = INITPROG('typeI',TTF,censored) initializes a Type I model
%       using the failure and censoring times contained in the column
%       vector TTF (nx1). The column vector censored (nx1) indicates
%       whether each value is a failure time (0) or a censored value (1)
%
%       Model =
%       INITPROG('typeI',TTF,censored,'distribution',distributiontype)
%       initializes a Type I model using the failure time distribution
%       indicated by distributiontype.  distributiontype can be set to any
%       of the following strings:
%           'weibull'
%           'exponential' (or 'exp')
%           'normal'
%       The default distribution type is Weibull.  Note that the variable
%       censored may be omitted if all data points are actual failure times
%
%       Model = INITPROG('mc',operatingconditions) initializes an MC model
%       usingthe historic operating parameter progressions to failure
%       contained in the cell array operatingconditions.  These conditions
%       should be discrete and numbered 1-n (no ordinal relation is implied
%       by the numbering). The degradation parameter is assumed to be a
%       weighted linear combination of the time spent in each operating
%       condition. Initial degradation is assumed to be identically zero.
%       The critical failure threshold for this parameter is assumed to be
%       100 (effectively measuring 100% degradation). If the data contained
%       in operatingconditions is not numbered from 1 to n, the data may be
%       reformatted in MCdata() prior to initializing the model.
```

```
%
%       Model = INITPROG('mc',operatingconditions,'flag',value,...)
%       initializes an MC model using the historic operating condition
%       progressions to failure and the user-supplied inputs indicated by
%       the 'flag'-value pairs. 'flag' may be set to any of the following,
%       with the default values given:
%           'Q' (calculated from data)  The transition probability matrix
%                                       for moving between operating
%                                       conditions
%           'u' (calculated from data)  The initial condition probability
%                                       vector
%           'f' (weighted sum of time spent in each condition) A function
%                                       for mapping operating conditions to
%                                       a degradation parameter. Should be
%                                       input as an anonymous function
%                                       @(b,t)f(b,t) where b is a column
%                                       vector of coefficients and t is a
%                                       row vector of time spent in each
%                                       condition, 1-n
%           'b' (calculated from data)  The vector of coefficients which is
%                                       used with f to map the operating
%                                       conditions to a degradation
%                                       parameter
%           'threshold' (100)           The critical failure threshold
%                                       applied to the degradation
%                                       parameter to indicate failure
%           'npop' (1000)               The number of Monte Carlo
%                                       simulations generated for each RUL
%                                       estimation
%           'RULcon' (0.95)             The point in the resulting RUL
%                                       distribution which defines the RUL.
%                                       0.95 indicates that RUL estimate is
%                                       the point where the reliability of
%                                       the system is 0.95 (Failure
%                                       probability is 0.05)
%
%       Model = INITPROG('shock',...) initializes a Shock model prognostic
%       model.
%
%       Model = INITPROG(‘phm’,cov,times) initializes a proportional hazards
%       model with the covariates defined in the nxp matrix cov and the
%       failure times in the nx1 column matrix times.  The baseline value is
%       taken to be zero.
%
%       Model = INITPROG(‘phm’,cov,times,'flag',value...) initializes a
%       proportional hazards model using the covariates and failures times
%       in the matrics cov and times, and the 'flag'-value pairs given.
%       'flag' may be set to any of the following, with the default values
%       given:
%           'frequency' (one for each observation) The jth element of this
%                               nx1 column vector indicates the number of
%                               times that the combination of the jth set
%                               of covariates and the jth failure time are
%                               observed together
%           'censoring' (all zeros) The jth element of this nx1 column
%                               vector indicates whether that observation
%                               is a failure time (0) or a censored time
%                               (1)
%           'baseline' (zero)   This 1xp row vector indicates the baseline
%                               values for each of the covariates
%           'beta' (calculated) The coefficients of the Cox proportional
%                               hazards regression
%           'hazard' (calculated) The baseline cumulative hazard function
%                               given as a 2-column matrix with failure
%                               times in the first column and the
%                               corresponding estimated cumulative hazard
%                               rate in the second column
%           'RULcon' (0.95)     The point in the resulting RUL distribution
%                               which defines the RUL. 0.95 indicates that
%                               RUL estimate is the point where the
%                               reliability of the system is 0.95 (Failure
%                               probability is 0.05)
%
%       Model = INITPROG('gpm',prognosticparameters) initializes a General
%       Path prognostic model using the historical prognostic parameters
%       contained in the cell array prognosticparameters.
%
%       Model = INITPROG('gpm',prognosticparameters,'flag',value...)
%       initializes a GPM using the historical prognostic parameters and
%       the values indicated by the 'flag' - value pairs. 'flag' may be set
%       to any of the following:
%           'bayesian' (true)       Use Bayesian updating to include prior
%                                   information in function fitting (true
%                                   or false)
%           'updateinterval' (1)    Number of time steps between fitting
%                                   updates (n, an integer)
%           'fit' (default is an optimization)  Functional fit to use for
%                                   GPM fitting (entered as a cell array of
%                                   function handles, i.e.
%                                   f = {@(x)f1(x) @x(f2(x) ... @(x)fn(x)}
%                                   where P(x) = f(x)*B). If a non-zero
%                                   intercept is required, an "@(x)1" entry
%                                   must be included.
%            'ytransform' (default is no transformation) Functional
%                                   transformation of y to make the
%                                   prognostic parameter linear in
%                                   parameters, i.e. if y = a*exp(bx) then
%                                   log(y) = log(a)+b*x, entered as
%                                   'ytransform' = @(x)log(x) and 'fit' =
%                                   {@(x)1 @(x)x}
%            'threshold' (default is automatic determination) Specificy the
%                                   critical threshold value (for hard
%                                   thresholds) or the mean and standard
%                                   deviation in the form [m s] (for
%                                   threshold distributions)
%            'thresholdtype' ('hard')Hard threshold or distribution ('hard'
%                                   or 'pdf')
%   See also RUNPROG INITTYPEI INITMC INITSHOCK INITPHM INITGPM

%   Jamie Coble
%   The University of Tennessee, Knoxville
%   Nuclear Engineering Department
%   Last Update:    2/18/2009
%   Copyright (c)

if strcmpi(type,'typeI')
    model = initTypeI(varargin);
    return
elseif strcmpi(type,'markovchain')||strcmpi(type,'mc')
    model = initMC(varargin);
    return
elseif strcmpi(type,'shock')
    model = initShock(varargin);
    return
elseif strcmpi(type,'proportionalhazards')||strcmpi(type,'phm')
    model = initPHM(varargin);
    return
elseif strcmpi(type,'generalpath')||strcmpi(type,'gpm')
    model = initGPM(varargin);
    return
else error('prognostictoolbox:initprog:invalidtype','Invalid Prognostic Model
Type')
end
```

```matlab
function [RUL model] = runprog(model,varargin)

%RUNPROG Make prognostic estimates based on the previously developed model
%       This function estimates the Remaining Useful Life (RUL) using the
%       model previously developed
%
%       RUL = RUNPROG(model,...) estimates the RUL for a component or
%       system based on the model. Additional inputs depend on the type of
%       model used for prognostics.
%
%       RUL = RUNPROG(model,currenttime) makes prognostic estimates for a
%       component or system that has lasted to time currenttime using a
%       Type I model. currenttime can be a scalar indicating the current
%       time for a single component or system, or it may be a vector
%       indicating the current time for a population of components or
%       systems. The RUL estimates for each individual in the population is
%       made independently.
%
%       RUL = RUNPROG(model) makes prognostic estimates using a Type II
%       Markov Chain or Shock model.  This call assumes that no information
%       is available about the actual or planned usage conditions and uses
%       the transition probability matrix in model for monte carlo
%       simulation from time zero.
%
%       RUL = RUNPROG(model,currentenvir) makes prognostic estimates using
%       a Type II Markov Chain or Shock model. The actual usage conditions
%       of the component or system are contained in the column vector
%       currentenvir, where the vector includes one observation for each
%       time step up to the current time. Future usage conditions are
%       estimated using the transition probability matrix contained in the
%       model.
%
%       RUL = RUNPROG(model,currentparam) makes prognostic estimates using
%       a Type III model.  model should be of type 'gpm'. currentparam is
%       matrix, cell array or column vector containing the prognostic
%       parameter values to the current time. For a list of acceptable data
%       entry methods see runGPM.
%
%       [RUL model] = RUNPROG(model,currentparam) makes prognostic
%       estimates using a Type III model and returns an updated model
%       structure.  The only change made to the model structure is to
%       update the bayesianprior field with the calculated posterior
%       distribution which is the new prior.
%
%   See also INITPROG RUNTYPEI RUNGPM RUNMC
%   Jamie Coble
%   The University of Tennessee, Knoxville
%   Nuclear Engineering Department
%   Last Update:    10/21/2008
%   Copyright (c)

type = model.type;

if strcmpi(type,'typeI')
    RUL = runTypeI(model,varargin);
    return
elseif strcmpi(type,'markovchain')||strcmpi(type,'mc')
    RUL = runMC(model,varargin);
    return
elseif strcmpi(type,'shock')
    RUL = runShock(model,varargin);
    return
elseif strcmpi(type,'proportionalhazards')||strcmpi(type,'phm')
    RUL = runPHM(model,varargin);
    return
elseif strcmpi(type,'generalpath')||strcmpi(type,'gpm')
    [RUL model] = runGPM(model,varargin);
    return
else error('prognostictoolbox:runprog:invalidtype','Invalid Prognostic Model
Type')
end
```

C.3 Type I Models

```
function model = initTypeI(inputs)

% INITTYPEI Initialize a Type I (reliability-based) prognostic model
%       This function initializes a reliability-based prognostic model
%       structure
%
%       Model = INITTYPEI(TTF) initializes a reliability-based prognostic
%       model using the failure times in the vector TTF. This model uses a
%       Weibull distribution to model failure times.
%
%       Model = INITTYPEI(TTF,censored) initializes a model using the
%       failure and censoring times in the vector TTF.  The variable
%       censored indicates whether each observed time is a failure (0) or
%       censored (1) time.
%
%       Model = INITTYPEI(TTF,censored,'distribution',distributiontype)
%       initializes a Type I model using the failure time distribution
%       indicated by distributiontype.  distributiontype can be set to any
%       of the following strings:
%           'weibull'
%           'exponential' (or 'exp')
%           'normal'
%       The default distribution type is Weibull.  Note that the variable
%       censored may be omitted if all data points are actual failure times
%

%   Jamie Coble
%   The University of Tennessee, Knoxville
%   Nuclear Engineering Department
%   Last Update:    9/16/2008
%   Copyright (c)

% extract TTF data %
TTF = inputs{1};

flags = [];
% set variable censored and extract flag/value pairs%
if size(inputs,2)>1
    if isnumeric(inputs{2})
        censored = inputs{2};
        if size(inputs,2)>2
            pairs = {inputs{3:end}};
            [flags values] = getfvp(pairs);
        end
    else censored = zeros(size(TTF));
        if size(inputs,2)>2
            pairs = {inputs{2:end}};
            [flags values] = getfvp(pairs);
        end
    end
else censored = zeros(size(TTF));
end

% initialize default model architecture %
model = struct('type','TypeI','distribution','weibull','parameters',[],...
    'data',struct('max',max(TTF),'min',min(TTF),'MTTF',mean(TTF)));

% apply user specifications if necessary %
for i = 1:length(flags)
    if strcmpi(flags{i},'distribution')
        model.distribution = values{i};
    else error('prognostictoolbox:initTypeI:invalidflag','Invalid flag')
    end
end

if strcmpi(model.distribution,'weibull')
    p = wblfit(TTF,0.05,censored);
    model.parameters = struct('beta',p(2),'theta',p(1));
elseif
strcmpi(model.distribution,'exponential')||strcmpi(model.distribution,'exp')
    model.parameters = struct('lambda',expfit(TTF,0.05,censored));
elseif strcmpi(model.distribution,'normal')
    [m s] = normfit(TTF,0.05,censored);
    model.parameters = struct('mean',m,'stddev',s);
else error('prognostictoolbox:initTypeI:invaliddistribution',...
        'Invalid TTF Distribution')
end
```

```
function MRL = runTypeI(model,currenttime)

%RUNTYPEI Make RUL estimates using a Type I prognostics model.
%       This function makes RUL estimats using a Type I (reliability
%       based) prognostics model.
%
%       RUL = RUNTYPEI(model,currenttime)makes prognostic estimates for a
%       component or system that has lasted to time currenttime using a
%       Type I model. currenttime can be a scalar indicating the current
%       time for a single component or system, or it may be a vector
%       indicating the current time for a population of components or
%       systems. The RUL estimates for each individual in the population is
%       made independently.
%
%       currenttime may be input as a column or row vector; the RUL
%       predictions will be returned in the same format.
%
%       RUL is calculated as the mean residual life at time currenttime.
%           MRL(t) = 1/R(t)*int(R(s),s = t->inf)

%   Jamie Coble
%   The University of Tennessee, Knoxville
%   Nuclear Engineering Department
%   Last Update:    10/21/2008
%   Copyright (c)

currenttime = cell2mat(currenttime);
dist = model.distribution;
MRL = zeros(size(currenttime));

if strcmpi(dist,'exp')||strcmpi(dist,'exponential')
    for i = 1:length(currenttime)
        MRL = 1./(1-expcdf(currenttime,model.parameters.lambda)).*trapz(...
            1-expcdf([currenttime(i):model.data.max*100],...
            model.parameters.lambda));
    end
elseif strcmpi(dist,'weibull')
    for i = 1:length(currenttime)
        MRL(i) = 1./(1-wblcdf(currenttime(i),model.parameters.theta,...
            model.parameters.beta)).*trapz(1-wblcdf([currenttime(i):...
            model.data.max*100],model.parameters.theta,...
            model.parameters.beta));
    end
elseif strcmpi(dist,'normal')
    for i = 1:length(currenttime)
        MRL = 1./(1-normcdf(currenttime,model.parameters.mean,...
            model.parameters.stddev)).*trapz(1-normcdf([currenttime(i):...
            model.data.max*100],model.parameters.mean,...
            model.parameters.stddev));
    end
else error('prognostictoolbox:runTypeI:invaliddistribution',...
        'Invalid TTF Distribution')
end
```

C.4 Type II Models

```
function model = initMC(inputs)

% INITMC Initialize a Type II Markov Chain Prognostic Model
%       This function initializes an MC model prognostics toolbox structure
%
%       Model = INITMC(operatingconditions) initializes an MC model using
%       the historic operating parameter progressions to failure contained
%       in the cell array operatingconditions.  These conditions should be
%       discrete and numbered 1-n (no ordinal relation is implied by the
%       numbering). The degradation parameter is assumed to be a weighted
%       linear combination of the time spent in each operating condition.
%       Initial degradation is assumed to be identically zero.  The
%       critical failure threshold for this parameter is assumed to be 100
%       (effectively measuring 100% degradation). If the data contained in
%       operatingconditions is not numbered from 1 to n, the data may be
%       reformatted in MCdata() prior to initializing the model.
%
%       Model = INITMC(operatingconditions,'flag',value,...) initializes
%       an MC model using the historic operating condition progressions to
%       failure and the user-supplied inputs indicated by the 'flag'-value
%       pairs. 'flag' may be set to any of the following, with the default
%       values given:
%           'Q' (calculated from data)  The transition probability matrix
%                                       for moving between operating
%                                       conditions
%           'u' (calculated from data)  The initial condition probability
%                                       vector
%           'f' (weighted sum of time spent in each condition) A function
%                                       for mapping operating conditions to
%                                       a degradation parameter. Should be
%                                       input as an anonymous function
%                                       @(b,t)f(b,t) where b is a column
%                                       vector of coefficients and t is a
%                                       row vector of time spent in each
%                                       condition, 1-n
%           'b' (calculated from data)  The vector of coefficients which is
%                                       used with f to map the operating
%                                       conditions to a degradation
%                                       parameter
%           'threshold' (100)           The critical failure threshold
%                                       applied to the degradation
%                                       parameter to indicate failure
%           'npop' (1000)               The number of Monte Carlo
%                                       simulations generated for each RUL
%                                       estimation
%           'RULcon' (0.95)             The point in the resulting RUL
%                                       distribution which defines the RUL.
%                                       0.95 indicates that RUL estimate is
%                                       the point where the reliability of
%                                       the system is 0.95 (Failure
%                                       probability is 0.05)
%
%   See also INITPROG MCDATA MCPROBS FITMC RUNMC

%   Jamie Coble
%   The University of Tennessee, Knoxville
%   Nuclear Engineering Department
%   Last Update:    6/26/2009
%   Copyright (c)

oc = inputs{1};
flags = [];
if size(inputs,2)>1
    pairs = {inputs{2:end}};
    [flags values] = getfvp(pairs);
end

model = struct('type','MC','Q',[],'u',[],'f',[],'b',[],'threshold',100,...
    'npop',1000,'RULcon',0.95);

for i = 1:length(flags)
    if strcmpi(flags{i},'Q')
        model.Q = values{i};
    elseif strcmpi(flags{i},'u')
        model.u = values{i};
    elseif strcmpi(flags{i},'f')
        model.f = values{i};
    elseif strcmpi(flags{i},'b')
        model.b = values{i};
    elseif strcmpi(flags{i},'threshold')
        model.threshold = values{i};
    elseif strcmpi(flags{i},'npop')
        model.npop = values{i};
    elseif strcmpi(flags{i},'RULcon')
        model.RULcon = values{i};
    else error('prognostictoolbox:initGPM:invalidflag','Invalid flag')
    end
end

if isempty(model.Q)||isempty(model.u)
    [q u] = MCprobs(oc);
    if isempty(model.Q)
        model.Q = q;
    end
    if isempty(model.u)
```

```
        model.u = u;
    end
end

if isempty(model.f)
    model.f = @(b,t)t*b;
end

if isempty(model.b)
    model.b = fitMC(oc,model.f,model.threshold);
end
```

```
function [Q u] = MCprobs(oc)

% MCPROBS calculate the transition and initial condition probabilities
%       This function calculates the transition probability matrix and the
%       initial condition probability vector for a Type II Markov Chain
%       model
%
%       [Q u] = MCPROBS(operatingconditions) calculates the transition
%       probability matrix (Q) and the initial condition vector (u) for a
%       Type II Markov Chain model using the operating condition
%       progressions to failure contained in the cell array
%       operatingconditions
%
%   See also INITMC MCDATA RUNMC


%   Jamie Coble
%   The University of Tennessee, Knoxville
%   Nuclear Engineering Department
%   Last Update:    6/26/2009
%   Copyright (c)

% determine the number of operating conditions
cond = [];
for i = 1:length(oc)
    cond = [cond; unique(oc{i})];
end
cond = unique(cond);
ncond = max(cond);

% initialize count and u with zeros %
count = zeros(ncond,ncond);
u = zeros(1,ncond);

for i = 1:length(oc)
    f = oc{i};
    u(f(1)) = u(f(1))+1;
    for j = 2:length(f)
        count(f(j-1),f(j)) = count(f(j-1),f(j))+1;
    end
end

u = u./sum(u);

countsums = sum(count,2);

Q = count./(countsums*ones(1,ncond));
```

```
function b = fitMC(oc,model,thresh)

% MCFIT Fit the coefficients of a model to map operating conditions to
% degradation
%       This function determines the coefficients to map time spent in each
%       operating condition to system degradation.
%
%       b = MCFIT(operatingconditions,model,threshold) calculates the
%       appropriate coefficients to fit the time spent in each operating
%       condition, calculated from the operating conditions progressions to
%       failure contained in the cell array operatingconditions, to the
%       model (given by an anonymous function of the form @(b,t)f(b,t))
%       with ending value equal to threshold
%
%   See also INITMC MCDATA RUNMC

%   Jamie Coble
%   The University of Tennessee, Knoxville
%   Nuclear Engineering Department
%   Last Update:    6/26/2009
%   Copyright (c)

% Define Y values at end of life to be equal to the failure threshold
y = thresh*ones(length(oc),1);

% Calculate t values, the time spent in each operating condition, for each
% historic failure progression

% determine number of operating conditions
cond = [];
for i = 1:length(oc)
    cond = [cond; unique(oc{i})];
end
cond = unique(cond);
ncond = max(cond);

t = zeros(length(oc),ncond);
for i = 1:length(oc)
    for j = 1:ncond
        t(i,j) = sum(oc{i}==j);
    end
end

b = nlinfit(t,y,model,zeros(ncond,1));
```

```
function [RUL ttf] = runMC(model,inputs)

% RUNMC Calculate Type II Markov Chain prognostic model RUL estimate
%       This function makes RUL estimates using a Type II MC model
%
%       RUL = RUNMC(model,operatingconditions) makes prognostic estimates
%       using a Type II Markov Chain model.  model should be of type 'MC'.
%       operating conditions is a column vector of the operating conditions
%       seen by the unit under test up to the current time.  If
%       operatingconditions is a cell array of column vectors, one vector
%       for the operation of a single unit to the current time, then RUL is
%       a row vector of RUL times.
%
%       RUL = RUNMC(model,'new') calculates the expected RUL of a system
%       starting from new, with no information about its operating states.
%
%   See also RUNPROG INITMC MCDATA MCPROBS FITMC

%   Jamie Coble
%   The University of Tennessee, Knoxville
%   Nuclear Engineering Department
%   Last Update:    6/26/2009
%   Copyright (c)


oc = inputs;
if ~iscell(oc)
    oc = {oc};
end


fit = model.f;
b = model.b;
thresh = model.threshold;
npop = model.npop;
Q = model.Q;
Q2 = cumsum(Q,2);
RULcon = model.RULcon;
ncond = length(Q);
u = model.u;

RUL = NaN(size(oc));
ttf = NaN(npop,size(oc,2));

for i = 1:length(RUL)
    unit = oc{i};

     % determine current degradation level
    t = zeros(1,ncond);
    if ~strcmpi(unit,'new')
        for j = 1:ncond
            t(j) = sum(unit==j);
        end
        deg = fit(b,t);
    else deg = 0;
    end

    for j = 1:npop
        deg_n = deg;
        t_n = t;
        if strcmpi(unit,'new')
            old_state = find(rand<=cumsum(u),1);
        else old_state = unit(end);
        end
         % generate future possible paths
        while deg_n<thresh
            new_state = find(rand<=Q2(old_state,:),1);
            t_n(new_state) = t_n(new_state)+1;
            deg_n = fit(b,t_n);
        end
        if strcmpi(unit,'new')
            ttf(j,i) = sum(t_n);
        else ttf(j,i) = sum(t_n) - sum(t);
        end
    end

    RUL(i) = quantile(ttf(:,i),1-RULcon);
end
```

```
function model = initPHM(inputs)

% INITPHM Initialize a Type II Proportional Hazards Prognostic Model
%       This function initializes an PHM prognostics toolbox structure
%
%       Model = INITPHM(cov,times) initializes a proportional hazards model
%       with the covariates defined in the nxp matrix cov and the failure
%       times in the nx1 column matrix times.  The baseline value is taken
%       to be zero.
%
%       Model = INITPHM(cov,times,'flag',value...) initializes a
%       proportional hazards model using the covariates and failures times
%       in the matrics cov and times, and the 'flag'-value pairs given.
%       'flag' may be set to any of the following, with the default values
%       given:
%           'frequency' (one for each observation) The jth element of this
%                               nx1 column vector indicates the number of
%                               times that the combination of the jth set
%                               of covariates and the jth failure time are
%                               observed together
%           'censoring' (all zeros) The jth element of this nx1 column
%                               vector indicates whether that observation
%                               is a failure time (0) or a censored time
%                               (1)
%           'baseline' (zero)   This 1xp row vector indicates the baseline
%                               values for each of the covariates
%           'beta' (calculated) The coefficients of the Cox proportional
%                               hazards regression
%           'hazard' (calculated) The baseline cumulative hazard function
%                               given as a 2-column matrix with failure
%                               times in the first column and the
%                               corresponding estimated cumulative hazard
%                               rate in the second column
%           'RULcon' (0.95)     The point in the resulting RUL distribution
%                               which defines the RUL. 0.95 indicates that
%                               RUL estimate is the point where the
%                               reliability of the system is 0.95 (Failure
%                               probability is 0.05)
%
%   See also INITPROG

%   Jamie Coble
%   The University of Tennessee, Knoxville
%   Nuclear Engineering Department
%   Last Update:    2/18/2010
%   Copyright (c)

cov = inputs{1};
failtimes = inputs{2};

flags = [];
if size(inputs,2)>2
    pairs = {inputs{3:end}};
    [flags values] = getfvp(pairs);
end

model = struct('type','PHM','beta',[],'hazard',[],...
    'baseline',zeros(1,size(cov,2)),'RULcon',0.95);
freq = ones(size(failtimes));
cens = zeros(size(failtimes));

for i = 1:length(flags)
    if strcmpi(flags{i},'frequency')
        freq = values{i};
    elseif strcmpi(flags{i},'censoring')
        cens = values{i};
    elseif strcmpi(flags{i},'baseline')
        model.baseline = values{i};
    elseif strcmpi(flags{i},'beta')
        model.beta = values{i};
    elseif strcmpi(falgs{i},'hazard')
        model.hazard = values{i};
    elseif strcmpi(flags{i},'RULcon')
        model.RULcon = values{i};
    else error('prognostictoolbox:initPHM:invalidflag','Invalid flag')
    end
end

if isempty(model.beta) || isempty(model.hazard)
    [b logl h stats] = coxphfit(cov,failtimes,'frequency',freq,...
        'censoring',cens,'baseline',model.baseline);
    if isempty(model.beta)
        model.beta = b;
    end
    if isempty(model.hazard)
        model.hazard = h;
    end
end
```

```
function [RUL F] = runPHM(model,inputs)

% RUNMC Calculate Proportional Hazards Model prognostic model RUL estimate
%       This function makes RUL estimates using a Proportional Hazards
%       Model
%
%       RUL = RUNPHM(model,covariates) makes prognostic estimates
%       using a Proportional Hazards model. RUL estimates are made for a
%       system running with the covariate values given in the nxp column
%       vector, covariates, where one row corresponds to the covariates
%       experienced during once cycle.
%
%       [RUL F] = RUNPHM(model,covariates) gives the RUL estimate defined
%       by the appropriate reliability confidence level and the Failure
%       time distribution, F, which is an nx2 matrix, where the first
%       column contains Failure Times (not RULs) and the second column
%       contains the probability of failure at that time.
%
%   See also RUNPROG INITPHM

%   Jamie Coble
%   The University of Tennessee, Knoxville
%   Nuclear Engineering Department
%   Last Update:    2/18/2010
%   Copyright (c)

if iscell(inputs)
    inputs = inputs{1};
end

hazard = model.hazard;
beta = model.beta;
RULcon = model.RULcon;
mcov = mean(inputs);
currenttime = size(inputs,1);
currenttimeind = find(abs(hazard(:,1)-currenttime)==min(abs(hazard(:,1)-
currenttime)));

% find new hazard rate %
haz_prime = hazard(:,2)*exp(mcov*beta);

% find new Reliability function and find conditional reliability at current
time %
R = exp(-haz_prime);
R = R./R(currenttimeind);

% find Time of Failure %
tof = hazard(abs(R-RULcon) == min(abs(R-RULcon)),1);
```

```
RUL = tof - currenttime;

faildist = 1-R(currenttimeind:end,:);
F = [hazard(currenttimeind:end,1) [0;diff(faildist)]];
```

```
function model = initShock(inputs)

% INITSHOCK Initialize a Shock Model
%   This function initializes a shock model prognostics toolbox structure
%
%   Model = INITSHOCK(shocks) initializes a shock model using the
%   progression of random shocks contained in the cell array shocks.
%
%   Model = INITSHOCK(shocks,'flag',value,...) initializes a shock model
%   using the progression of random shocks contained in the cell array
%   shocks and and the values indicated by the 'flag' - value pairs. 'flag'
%   may be set to any of the following:
%
%       'timedist' (static)         Distribution of occurence of shocks
%                                   ('exponential', 'normal', 'lognormal')
%       'timepar' (distribution fit)Parameters of time distribution
%       'magdist' (normal)          Distribution of the magnitude of shocks
%                                   ('constant', 'normal', 'nonparametric')
%       'magpar' (distribution fit) Parameters of magnitude distribution
%       'thresholdtype' ('hard')    Hard threshold or distribution ('hard'
%                                   or 'pdf')
%       'threshold' (calculation)   Specificy the critical threshold value
%                                   (for hard thresholds) or the mean and
%                                   standard deviation in the form [m s]
%                                   (for threshold distributions)

%   Jamie Coble
%   The University of Tennessee, Knoxville
%   Nuclear Engineering Department
%   Last Update:    7/6/2009
%   Copyright (c)

shocks = inputs{1};
flags = [];
if size(inputs,2)>1
    pairs = {inputs{2:end}};
    [flags values] = getfvp(pairs);
end

model = struct('type','Shock','noisevar',[],'timedist','exponential',...
    'timepar',[],'magdist','normal','magpar',[],'thresholdtype','hard',...
    'threshold',[]);

% set flag values %
for i = 1:length(flags)
    if strcmpi(flags{i},'timedist')
        model.timedist = values{i};
    elseif strcmpi(flags{i},'timepar')
        model.timepar = values{i};
    elseif strcmpi(flags{i},'magdist')
        model.magdist = values{i};
    elseif strcmpi(flags{i},'magpar')
        model.magpar = values{i};
    elseif strcmpi(flags{i},'thresholdtype')
        model.thresholdtype = values{i};
    elseif strcmpi(flags{i},'threshold')
        model.threshold = values{i};
    else error('prognostictoolbox:initShock:invalidflag','Invalid flag')
    end
end


% estimate the noise in the data %


nvar = zeros(1,length(shocks));
for i = 1:length(shocks)
    nvar(i) = enovar(shocks{i},'medianfilter',4);
end

noise_var = mean(nvar);

% identify shocks %


for i = 1:length(shocks)
    [peaks{i} times{i}] = findpeaks(shocks{i},'minpeakheight',...
        3*sqrt(noise_var)+mean(shocks{i}));
end

% estimate noise in shock-less data %


nvar2 = zeros(1,length(shocks));
for i = length(shocks)
    data = shocks{i};
    data(times{i}) = [];
    nvar2(i) = enovar(data,'medianfilter'4);
end

model.noisevar = mean(nvar2);

% calculate time between shocks, if necessary %
if isempty(model.timepar)

end

% calculate magnitude of shocks, if necessary %
if isempty(model.magpar)
    peak = cell2mat(peaks);
```

```
end
```

C.5 Type III Models

```
function model = initGPM(inputs)

% INITGPM Initialize a General Path Prognostic Model
%       This function initializes an GPM prognostic toolbox structure
%
%       Model = INITGPM(prognosticparameters) initializes a GPM model using
%       the historic paths contained in the cell array
%       prognosticparameters. The most appropriate fit is chosen from
%       quadratic, linear, and exponential fits.
%
%       Model = INITGPM(prognosticparameters,'flag',value,...) initializes
%       a GPM model using the historical prognostic parameters and the
%       values indicated by the 'flag' - value pairs. 'flag' may be set to
%       any of the following:
%           'bayesian' (true)       Use Bayesian updating to include prior
%                                   information in function fitting (true
%                                   or false)
%           'noise' (calculated using enovar() in PEM) An estimate of the
%                                   noise variance
%           'updateinterval' (1)    Number of time steps between fitting
%                                   updates (n, an integer)
%           'fit' (default is an optimization)  Functional fit to use for
%                                   GPM fitting (entered as a cell array of
%                                   function handles, i.e.
%                                   f = {@(x)f1(x) @x(f2(x) ... @(x)fn(x)}
%                                   where P(x) = f(x)*B). Function must be
%                                   linear in parameters for this type of
%                                   entry. If a non-zero intercept is
%                                   required, an "@(x)1" entry must be
%                                   included. If no fit is given, an
%                                   optimization is performed for the best
%                                   fit among linear, quadratic, and
%                                   exponential models.
%           'ytransform' (default is no transformation) Functional
%                                   transformation of y to make the
%                                   prognostic parameter linear in
%                                   parameters, i.e. if y = a*exp(bx) then
%                                   log(y) = log(a)+b*x, entered as
%                                   'ytransform' = @(y)log(y) and 'fit' =
%                                   {@(x)1 @(x)x}
%           'threshold' (default is automatic determination) Specificy the
%                                   critical threshold value (for hard
%                                   thresholds) or the mean and standard
%                                   deviation in the form [m s] (for
%                                   threshold distributions)
%           'thresholdtype'  ('hard')Hard threshold or distribution ('hard'
%                                    or 'pdf')
%           'threshcon' (0.95)       The confidence level at which the
%                                    threshold is calculated.  Only
%                                    applicable for "hard" type thresholds.
%
%   See also INITPROG INITBAYES FITGPM THRESHGPM RUNGPM PPMETRICS

%   Jamie Coble
%   The University of Tennessee, Knoxville
%   Nuclear Engineering Department
%   Last Update:    2/18/2009
%   Copyright (c)

progparam = inputs{1};
flags = [];
if size(inputs,2)>1
    pairs = {inputs{2:end}};
    [flags values] = getfvp(pairs);
end

model = struct('type','GPM','bayesian',true,'updateinterval',1,'fit',[],...
    'ytransform',@(y)y,'threshold',[],'thresholdtype','hard',...
    'threshcon',0.95);

for i = 1:length(flags)
    if strcmpi(flags{i},'bayesian')
        model.bayesian = values{i};
    elseif strcmpi(flags{i},'updateinterval')
        model.updateinterval = values{i};
    elseif strcmpi(flags{i},'fit')
        model.fit = values{i};
    elseif strcmpi(flags{i},'ytransform')
        model.ytransform = values{i};
    elseif strcmpi(flags{i},'threshold')
        model.threshold = values{i};
    elseif strcmpi(flags{i},'thresholdtype')
        model.thresholdtype = values{i};
    elseif strcmpi(flags{i},'threshcon')
        model.threshcon = values{i};
    else error('prognostictoolbox:initGPM:invalidflag','Invalid flag')
    end
end


if isempty(model.fit)
    [model.fit model.ytransform] = fitGPM(progparam);
end

if isempty(model.threshold)
```

```
    model.threshold = threshGPM(progparam,model.fit,model.ytransform,...
        model.thresholdtype,model.threshcon);
end

if model.bayesian
    model.noisevar = [];
    [model.bayesianprior model.noisevar] =
initBayes(progparam,model.fit,model.ytransform);
end
```

```
function [fit ytransform]= fitGPM(progparam)

%FITGPM Fit a GPM prognostic model
%       This function determines the best fit for a GPM prognostic model
%       between linear, quadratic, and exponential fits
%
%       [Function Ytransform] = FITGPM(prognosticparameters) determines the
%       best fit for the historic paths contained in the cell array
%       prognosticparameters. The function considers linear, quadratic, and
%       exponential fits. Ytransform gives the transformation of y needed
%       to make the fit linear in parameters.  This has value @(y)log(y)
%       for exponential functions and @(y)y (indicating no transformation
%       is needed) for linear and quadratic functions.
%

%   Jamie Coble
%   The University of Tennessee, Knoxville
%   Nuclear Engineering Department
%   Last Update:    2/18/2009
%   Copyright (c)

h = waitbar(0,'Optimizing GPM Fit');
errors = zeros(size(progparam,2),3);

% possible fits are Linear Quadratic Exponential %
yt = {@(y)y @(y)y @(y)log(y)};
f = {{@(x)x @(x)1} {@(x)x.^2 @(x)x @(x)1} {@(x)x @(x)1}};

for i = 1:3
    for j = 1:size(progparam,2)
        p = progparam{j};
        if size(p,2)==2
            t = p(:,1);
            y = p(:,2);
        else y = p; t = [1:size(y,1)]';
        end
        x = zeros(size(t,1),size(f{i},2));
        for k = 1:size(f{i},2)
            x(:,k) = f{i}{k}(t);
        end
        yf = yt{i}(y);
        bfit = inv(x'*x)*(x'*yf);
        g = yinv(yt{i});
        errors(j,i) = sqrt(mean((g(x*bfit)-y).^2));
    end
    waitbar(i/3,h)
end

mses = nanmean(errors);
```

```
fit = f{find(mses==min(mses),1)};
ytransform = yt{find(mses==min(mses),1)};

close(h)

%%
function f = yinv(yt)

if any(strfind(func2str(yt),'log'))
    f = @(x)exp(x);
else f = @(x)x;
end
```

```
function [prior nvar] = initBayes(progparam,f,yt)

%INITBAYES Determine the initial Bayesian prior for a GPM model
%       This function calculates the initial Bayesian prior distribution
%       for a GPM model.
%
%       Prior = INITBAYES(prognosticparameters,fit,ytransform) calculates
%       the Bayesian prior distribution for the coefficients, b, of the
%       functional fit f(b,t) where fit is a cell array of the form
%       f(x) = {@(x)f1(x) @(x)f2(x) ... @(x)fn(x)} and ytransform is a
%       function handle P(y) = @(y)fy(y), where P(y) = f(x)*b for the
%       historic prognostic parameter paths contained in the cell array
%       prognosticparameters. It is assumed that the coefficients are
%       normally distributed with mean and variance calculated from the
%       population of fits. Prior is a matrix which contains the mean value
%       for each parameter in the first row and the standard deviation in
%       the second row:
%
%               prior = [m1 m2 ... mn
%                        s1 s2 ... sn]
%

%   Jamie Coble
%   The University of Tennessee, Knoxville
%   Nuclear Engineering Department
%   Last Update:    9/18/2008
%   Copyright (c)

h = waitbar(0,'Calculating Bayesian Priors');
sizeb = size(f,2);
b_est = zeros(size(progparam,2),sizeb);
noisevar = zeros(size(progparam,2),1);

for i = 1:size(progparam,2)
    p = progparam{i};
    if size(p,2)==2
        t = p(:,1);
        y = p(:,2);
    else y = p; t = [1:size(y,1)]';
    end
    x = zeros(size(t,1),size(f,2));
    for k = 1:size(f,2)
        x(:,k) = f{k}(t);
    end
    y = yt(y);
    b_est(i,:) = inv(x'*x)*(x'*y);
    noisevar(i) = enovar(y,'medianfilter',4);
    waitbar(i/size(progparam,2),h)
```

```
end

prior = [mean(b_est);std(b_est)];
nvar = mean(noisevar);

close(h)
```

```
function thresh = threshGPM(progparam,fit,yt,type,threshcon)

%THRESHGPM Determine the critical failure threshold for GPM model
%       This function calculates the critical failure threshold for a GPM
%       model based on the full historic failure paths contained in
%       progparam.
%
%       Threshold = THRESHGPM(prognosticparameters,fit,yt) determines the
critical
%       failure threshold for the paths contained in the cell array
%       prognosticparameters.  The function calculates a hard threshold as
%       the conservative 95th percentile of the failure values for the
%       historic paths.
%
%       Threshold =
THRESHGPM(prognosticparameters,fit,yt,thresholdtype,threshcon)
%       determines the critical failure threshold using the type specified
%       by thresholdtype. Thresholdtype may be set to either
%           'hard'      A constant value is used for the critical failure
%                       threshold. This value is the conservative 95th
%                       quantile of the failure values for the historic
%                       paths
%           'pdf'       A distribution of critical threshold values is
%                       determined. The failure values are assumed to be
%                       normally distributed with mean and variance as
%                       determined by the historic paths.  The threshold
%                       distribution is returned as a 1x2 vector of the
%                       form [mean standarddeviation]
%       Threshcon indicates the confidence level at which the threshold is
%       calculated from the failure values.
%

%   Jamie Coble
%   The University of Tennessee, Knoxville
%   Nuclear Engineering Department
%   Last Update:    9/18/2008
%   Copyright (c)

h = waitbar(0,'Calculating GPM Threshold');
if nargin == 1
    type = 'hard';
end

% determine parameter value at failure for each historic path %

failval = NaN(size(progparam,2),1);
```

```
for j = 1:size(progparam,2)
    p = progparam{j};
    if size(p,2)==2
        t = p(:,1);
        y = p(:,2);
    else y = p; t = [1:size(y,1)]';
    end
    x = zeros(size(t,1),size(fit,2));
    for k = 1:size(fit,2)
        x(:,k) = fit{k}(t);
    end
    y = yt(y);
    bfit = (x'*x)\(x'*y);
    startval(j) = x(1,:)*bfit;
    failval(j) = x(end,:)*bfit;
    waitbar(j/(size(progparam,2)+1),h)
end

%
% for i = 1:size(progparam,2)
%     failval(i) = progparam{i}(end,end);
%     waitbar(i/(size(progparam,2)+1),h)
% end

if strcmpi(type,'hard')
    if abs(mean(startval)-quantile(failval,threshcon))<abs(mean(startval)-
quantile(failval,1-threshcon))
        thresh = quantile(failval,threshcon);
    else thresh = quantile(failval,threshcon);
    end
elseif strcmpi(type,'pdf')
    thresh = [mean(failval) std(failval)];
else error('prognosticstoolbox:threshGPM:invalidtype','Invalid threshold
type');
end

close(h)
```

```
function [RUL model] = runGPM(model,varargin)

%RUNGPM Make RUL estimates using a GPM model.
%       This function makes RUL estimates using a general path  model.
%
%       RUL = RUNGPM(model,currentparam)  makes prognostic estimates using
%       a Type III model.  model should be of type 'gpm'. currentparam may
%       be a column vector, matrix, or cell array. If only one system is
%       under surveillance, currentparam should be a column vector
%       containing observations of that system up to the current time. If
%       multiple systems are under surveillance, and parameter observations
%       are available for the same time steps for each system, currentparam
%       may be a matrix whose columns contain the parameter observations
%       for a single system. If multiple systems are under surveillance
%       which have been running for different amounts of time, currentparam
%       may be a cell array containing the parameter observations for each
%       system in a column vector contained in separate cells. Here, it is
%       assumed that observations are made every time unit, with an equal
%       sampling interval.
%
%       RUL = RUNGPM(model,currentparam) may also be used for multiple
%       systems under surveillance where the time stamp for each system is
%       not the same. In this case, currentparam should be a cell array
%       whose cells contain an nx2 matrix where the first column is the
%       time stamp for that particular unit and the second column is the
%       parameter values at each time.
%
%       RUL = RUNGPM(model,timestamp,currentparam) can be used when the
%       sampling interval is not equal across observations. If currentparam
%       is a column vector, then timestamp should also be a column vector.
%       If currentparam is a matrix, then timestamp may be a column vector
%       of times (if each unit is surveyed at the same time) or it may be a
%       matrix of times (if surveillance times for each unit are
%       different).
%
%       [RUL model] = RUNGPM(...) returns the estimated RUL and the model
%       structure with an updated Bayesian prior for future estimates.  No
%       other fields are changed.
%
%   See also RUNPROG INITGPM INITBAYES FITGPM THRESHGPM

%   Jamie Coble
%   The University of Tennessee, Knoxville
%   Nuclear Engineering Department
%   Last Update:    10/21/2008
%   Copyright (c)

options = optimset('Display','off'); % set options for use with fzero()

% put data into cell arrays where each cell is an nx2 matrix. the first
% column is the time stamp and the second column is the series of parameter
% values for that system
if size(varargin,2)==1
    tmp = varargin{1};
    if ~iscell(tmp)    % a matrix of param values is supplied, no time %
        for i = 1:size(tmp,2)
            currentparam{i} = [[1:size(tmp,1)]' tmp(:,i)];
        end
    elseif size(tmp{1},2)==1 % a cell array of param values is supplied, no
time %
        for i = 1:size(tmp,1)
            currentparam{i} = [[1:size(tmp{i},1)]' tmp{i}];
        end
    else currentparam = tmp; % a cell array of param values and time is
supplied %
    end
else % separate time and param values are supplied %
    time = varargin{1};
    tmp = varargin{2};
    if size(time,2)~=size(tmp,2)
        time = time*ones(1,size(tmp,2));
    end
    if ~iscell(tmp)
        for i = 1:size(tmp,2)
            currentparam{i} = [time(:,i) tmp(:,i)];
        end
    elseif iscell(tmp) && size(tmp{1},2)==1
        for i = 1:size(tmp,2)
            currentparam{i} = [time(:,i) tmp{i}];
        end
    else error('prognostictoolbox:runGPM:invaliddatatype','Invalid
Time/Parameter Entry');
    end
end

yt = model.ytransform;

if ~model.bayesian
    for i = 1:size(currentparam,2)
        time = currentparam{i}(:,1);
        x = zeros(size(time,1),size(model.fit,2));
        for k = 1:size(model.fit,2)
            x(:,k) = model.fit{k}(time);
        end
        y = yt(currentparam{i}(:,2));
        b = inv(x'*x)*(x'*y);
        if size(model.threshold,2)>1
```

```
            thresh = yt(model.threshold.mean); %apply y transform to 
threshold value %
        else thresh = yt(model.threshold);
        end
        fcn = fsum(model.fit,b);
        try
        [TOF(i) fval flag] = fzero(@(x)fcn(x)-thresh,[time(end) 
1e10],options);
        RUL(i) = TOF(i) - time(end);
        catch
            fprintf('Model fit is not adequate to make RUL estimation. More 
data is needed\n')
            RUL = NaN;
            return
        end
        if flag<0 % value could not be found where param crosses threshold %
            fprintf('Model fit is not adequate to make RUL estimation. More 
data is needed\n')
            return
        end
    end
else
    nvar = model.noisevar;
    prior = model.bayesianprior;
    for i = 1:size(currentparam,2)
        for j = 
model.updateinterval:model.updateinterval:size(currentparam{i},1)
            p = currentparam{i}(1:j,:);
              % parameter
            y = [yt(p(:,2));prior(1,:)'];
            time = p(:,1);
            x = zeros(size(time,1)+size(prior,2),size(model.fit,2));
            for k = 1:size(model.fit,2)
                x(1:length(time),k) = model.fit{k}(time);
            end
            x(length(time)+1:end,:) = eye(size(prior,2));
            invV = diag([ones(1,size(p,1))*nvar prior(2,:).^2].^-1);
            
              % posterior parameter estimates %
            b = inv(x'*invV*x)*(x'*invV*y);
              % sd of posterior
            s = sqrt((1./prior(2,:).^2+j/nvar).^-1);
              % posterior becomes new prior %
            prior = [b';s];
        end
        if size(model.threshold,2)>1
            thresh = yt(model.threshold(1)); %apply y transform to threshold 
value %
        else thresh = yt(model.threshold);
        end
        fcn = fsum(model.fit,b);
        try
        [TOF(i) fval flag] = fzero(@(x)fcn(x)-thresh,[time(end)
1e10],options);
        if flag<0 % value could not be found where param crosses threshold %
            fprintf('Model fit is not adequate to make RUL estimation. More
data is needed\n')
            return
        end
        catch TOF(i) = NaN;
        end
        RUL(i) = TOF(i) - time(end);
        model.bayesianprior = prior;
    end
end



%%
function fcn = fsum(fit,b)

% combine the list of x function handles and the estimated coefficients into
% one function

b = num2str(b);
for i = 1:size(fit,2)
    % convert the function to a string %
    f{i} = char(fit{i});
    f{i}(1:4) = [];
end

y = [];
for j = 1:size(f,2)
    if j~=size(f,2)
        y = strcat(y,b(j,:),'*',f{j},'+');
    else y = strcat(y,b(j,:),'*',f{j});
    end
end

fcn = eval(['@(x)' y]);
```

C.6 Parameter Identification

```
function [param fval] = optparam(input, varargin)

% OPTPARAM Identify a near-optimal prognostic parameter
%       This function uses Genetic Algorithms and parameter suitability
%       metrics to identify a near-optimal prognostic parameter from
%       several data sources.
%
%       param = OPTPARAM(inputs, 'flag', value, ...) identifies a
%       near-optimal prognostic parameter from the available data in the
%       cell array inputs.  The function uses genetic algorithms to
%       optimize the weights in a linear combination of the available
%       signal inputs.  Function returns a structure, param, which includes
%       the resulting weights and all information needed to obtain the
%       parameter from a new data run using function PARAMGEN. Flag/value
%       pairs may be set to any of the
%       following:
%           'inputs' ('all')    Determines how many of the inputs should be
%                               considered by the GA. Can be set to 'all',
%                               indicating that all should be used, or
%                               'subset', indicating that a subset of
%                               useful parameters should be used.
%           'cutoff' (1.5)      Cutoff value for determining which input
%                               parameters are useful when choosing a
%                               subset of inputs for optimization.
%           'fitness' (sum of M,P,T) Identifies the fitness function to be
%                               used, input as @fitness. fitness(w,inputs)
%                               must be a matlab m-file which takes only
%                               the candidate solution from the GA and the
%                               cell array of possible inputs to determine
%                               the fitness of a candidate solution.
%           'fitweights' ([1 1 1]) A 1x3 row vector of weights which gives
%                               the weight of each of the three suitability
%                               metrics in determining the fitness. The
%                               first entry corresponds to monotonicity,
%                               the second to prognosability, and the third
%                               to trendability.
%           'initpop' ([])      Any candidate weightings that the GA should
%                               consider. If visual inspection or expert
%                               analysis has lead to any suitable
%                               parameters, these can be included in the GA
%                               to allow it to explicitly consider them in
%                               the optimization.  They should be included
%                               as an nXm matrix where n is the number of
%                               possible prognostic parameters to be
%                                 included and m is the number of candidate
%                                 inputs to the parameter.  Each row should
%                                 contain the weights associated with one
%                                 possible parameter.
%            'group'  (one group) Multiple prognostic parameters may be
%                                 optimized to compare groups. Associated
%                                 value must be a cell array of matrices
%                                 where each matrix indicates the runs
%                                 included in a specific group. Groups may be
%                                 overlapping. Output in this case will be a
%                                 structure array where the ith entry
%                                 corresponds to the optimum parameter for
%                                 the ith group.
%            'smoothing'  (false) May be set to 'true' or 'false', indicates
%                                 whether the resulting parameter should be
%                                 smoothed prior to final model development.
%
%        SEE ALSO: PARAMGEN

%    Jamie Coble
%    The University of Tennessee, Knoxville
%    Nuclear Engineering Department
%    Last Update:    9/27/2009
%    Copyright (c)

flags = [];
if ~isempty(varargin)
    [flags values] = getfvp(varargin);
end

%% initialize parameter structure


param = struct('weights',[],'inputs',[1:size(input{1},2)],'cutoff',1.5,...
    'fitness',@paramfit,'fitweights',ones(1,3),'initpop',[],...
    'smoothing',false);

if any(strcmpi('fitweights',flags))
    param.fitweights = values{strcmpi('fitweights',flags)};
end

groups = [];

for i = 1:length(flags)
    if strcmpi(flags{i},'inputs')
        if strcmpi(values{i},'subset')
            if any(strcmpi('cutoff',flags))
                cutoff = values{strcmpi('cutoff',flags)};
            else cutoff = 1.5;
            end
            sens = [];
            w = param.fitweights;
            for k = 1:size(input{1},2)
                for j = 1:length(input)
                    temp{j} = input{j}(:,k);
                end
                [m p t] = ppmetrics(temp);
                if w*[m p t]' >= cutoff
                    sens = [sens k];
                end
            end
            param.inputs = sens;
            param.cutoff = cutoff;
        elseif strcmpi(values{i},'all')
        else param.inputs = values{i};
        end
    elseif strcmpi(flags{i},'fitness')
        param.fitness = values{i};
    elseif strcmpi(flags{i},'smoothing')
        param.smoothing = values{i};
    elseif strcmpi(flags{i},'initpop')
        param.initpop = values{i};
    elseif strcmpi(flags{i},'cutoff')
        param.cutoff = values{i};
    elseif strcmpi(flags{i},'group')
        groups = values{i};
    else error('prognostictoolbox:optparam:invalidflag',...
            'Invalid Parameter Optimization Flag');
    end
end

if ~isempty(groups)
    param.group = [];
    p = param;
    for i = 1:length(groups)
        param(i) = p;
        param(i).group = groups{i};
    end
else groups = 1:length(input);
end
%% optimize parameter weights using GA

for j = 1:length(param)
    inputs = input(groups{j});

    options = gaoptimset('plotfcns', {@gaplotbestf},...
        'InitialPopulation',param(j).initpop,'PopulationSize',100);
    for i = 1:length(inputs)
        inputs{i} = inputs{i}(:,param(j).inputs);
    end

    param(j).fitness =
@(x)param(j).fitness(x,inputs,param(j).fitweights,param(j).smoothing);

    [param(j).weights fval flag] =
ga(param(j).fitness,length(param(j).inputs),options);
end
```

```
function fitness = paramfit(x,inputs,w,s)

% PARAMFIT Function to determine fitness of candidate prognostic parameter
%       This function determines the fitness of a candidate prognostic
%       parameter as the sum of monotonicity, prognosability, and
%       trendability
%
%       fitness = PARAMFIT(weights,inputs) calculates the fitness of a
%       linear combination of inputs, weighted according to the row vector
%       weight, as the sum of the parameter suitability metrics
%       monotonicity, prognosability, and trendability.
%
%       fitness = PARAMFIT(weights,inputs,suitabilityweights) calculates
%       the fitness of a linear combination of inputs, weighted according
%       to the row vector weight, as the weighted sum of the parameter
%       suitability metrics monotonicity, prognosability, and trendability,
%       where suitabilityweights is a 1x3 row vector with the weight of
%       monotonicity in the first entry, prognosability in the second, and
%       trendability in the third.
%
%
%   SEE ALSO OPTPARAM PARAMGEN

%   Jamie Coble
%   The University of Tennessee, Knoxville
%   Nuclear Engineering Department
%   Last Update:    9/27/2009
%   Copyright (c)

% set default suitability weights if necessary %
if nargin<3
    w = ones(1,3);
end

% calculate the candidate parameter for each run in inputs
param = cell(size(inputs));
for i = 1:length(inputs)
    param{i} = inputs{i}*x';
    if s
        param{i} = expfilt(param{i});
    end
end

[m p t] = ppmetrics(param);

fitness = -w*[m p t]';
```

```
function [monotonicity prognosability trendability] = ppmetrics(params)

% PPMETRICS characterizes the appropriateness of a prognostic parameter
%       based on three metrics
%
%       [monotonicity prognosability trendability] = PPMETRICS(params)
%       evaluates the population of prognostic parameters contained in the
%       cell array params for three metrics of adequacy
%
%           Monotonicity measures the general increasing or decreasing
%           trend of the parameter. Because the assumption is made that
%           systems do not self heal and no corrective action is taken,
%           prognostic parameters are assumed to be monotonic. This
%           assumption may not be valid for some systems such as batteries
%           which do exhibit some self healing during periods of rest, or
%           systems which experience some outside intervention to improve
%           the condition.
%
%           Prognosability is a measure of the variance of the failure
%           values for a population of parameters.
%
%           Trendability characterizes how well a population of parameters
%           can be fit by the same functional form. It measures the
%           similarity of the general trend of the parameter for a
%           population of systems.
%

%   Jamie Coble
%   The University of Tennessee, Knoxville
%   Nuclear Engineering Department
%   Last Update:    10/21/2008
%   Copyright (c)

%% Remove time variable if necessary
for i = 1:length(params)
    if size(params{i},2) == 2
        params{i}(:,1) = [];
    end
end

%% Smooth Prognostic Parameters

sparam = cell(size(params));
for i = 1:length(params)
    if size(params{i},1)>100
        sparam{i} = medfilt1(params{i},10);
        sparam{i}([1:5 end-5:end],:) = []; % remove faulty smoothed values
```

```
due to zero padding %
    else sparam{i} = params{i};
    end
end

%% Monotonicity


m = 30;
mpos = NaN(1,length(sparam));
mneg = NaN(1,length(sparam));
for i = 1:length(sparam)
    if size(sparam{i},1)>2*m
        n = floor(size(sparam{i},1)/m);
    else n = 3; m = (size(sparam{i},1)./3);
    end
    s = zeros(1,n);
    for j = 1:n
        b = regress(sparam{i}(floor((j-1)*m+1):round(j*m)),[floor((j-
1)*m+1):round(j*m);ones(size(floor((j-1)*m+1):round(j*m)))]');
        s(j) = b(1);
    end
    mpos(i) = sum(s>0)/(n);
    mneg(i) = sum(s<0)/(n);
end

monotonicity = nanmean(abs(mpos-mneg));

%% Prognosability


failval = NaN(1,length(params));
startval = NaN(1,length(params));
for i = 1:length(params)
    failval(i) = params{i}(end);
    startval(i) = params{i}(1);
end

prognosability = exp(-std(failval)/mean(abs(startval-failval)));

%% Trendability

% Re-sample data into %life instead of observations %
p = NaN(100,length(sparam));
for i = 1:length(sparam)
    p(:,i) = resample(sparam{i},100,length(sparam{i}));
end

% Trendability is the minimum absolute correlation %
cc = corrcoef(p);
trendability = min(min(abs(cc)));
```

```
function param = paramgen(param_struct,inputs,time)

% PARAMGEN Generate prognostic parameters according to optimized options
%       This function generates a population of prognostic parameters
%       according to the options saved in the structure param_struct
%
%       progparam = PARAMGEN(param_struct,inputs) creates a population of
%       prognostic parameters using the options saved in param_struct and
%       the candidate inputs in the cell array inputs. OPTPARAM() can be
%       used to generate the parameter structure.
%
%       progparam = PARAMGEN(param_struct,inputs,time) creates a population
%       of prognostic parameters which includes the time variable for each
%       case in the first column and the prognostic parameter in the
%       second.  Both inputs and time should be cell arrays where the
%       number of observations (rows) in each cell of inputs should be
%       equal to the corresponding cell of time.
%
%       PARAMGEN calls also produce a plot of the population of prognostic
%       parameters.
%
%   SEE ALSO OPTPARAM

%   Jamie Coble
%   The University of Tennessee, Knoxville
%   Nuclear Engineering Department
%   Last Update:    3/16/2010
%   Copyright (c)

sens = param_struct.inputs;
smooth = param_struct.smoothing;
weights = param_struct.weights;

if exist param_struct.groups
    inputs = inputs(groups);
end

param = cell(size(inputs));
for i = 1:length(inputs)
    param{i} = inputs{i}(:,sens)*weights';
    if smooth
        param{i} = expfilt(param{i});
    end
end

if nargin == 3
    for i = 1:length(param)
        param{i} = [time{i} param{i}];
    end
else time{i} = (1:length(param{1}))';
end

figure
hold on
for i = 1:size(param,2)
    plot(time{i},param{i}(:,end),'b');
    endval(i) = param{i}(end,end);
    endtime(i) = time{i}(end);
end
plot(endtime,endval,'r*')
xlabel('Time (cycles)')
ylabel('Prognostic Parameter')
box
```

C.7 Data Preprocessing

```
function [data map] = MCdata(oc,varargin)

% MCDATA Convert discrete data points into operating classes for MC model
%       This function converts operating condition data into data needed
%       for Markov Chain Models
%
%       [newoperatingconditions map] = MCDATA(operatingconditions) converts
%       the operating condition progressions to failure contained in the
%       cell array operatingconditions into conditions numbered 1-n (no
%       ordinal relation is implied by the new numbers).  The progressions
%       in the cell array operatingconditions may be of size nxm where n is
%       the number of observations in one history and m is the number of
%       variables which fully define the operating condition.  These values
%       need to be discrete (or discretized by some outside method) to
%       apply this function correctly. The output map is a matrix which
%       indicates the relationship between the original operating
%       conditions (of size 1xn) to an operating condition class. The first
%       row in map defines the first operating condition, the second row
%       the second operating condition, and so on.
%
%       [newoperatingconditions map] = MCDATA(operatingconditions, 'flag',
%       value) separates the operating conditions progressions contained in
%       the cell array operatingconditions into MC appropriate conditions.
%       The 'flag'/value pairs may be set to any of the following:
%           'tol'    (0.10)   The noise tolerance for separating operating
%                             conditions
%           'map'    (determined) The map for moving from measured operating
%                             conditions to the MC numbered conditions where
%                             the first row in map defines the first
%                             operating condition, the second row in map the
%                             second operating condition, and so on.
%
%
%   See also INITMC RUNMC

%   Jamie Coble
%   The University of Tennessee, Knoxville
%   Nuclear Engineering Department
%   Last Update:    9/26/2009
%   Copyright (c)

flags = [];
if ~isempty(varargin)
    [flags values] = getfvp(varargin);
end
```

```
% Set defaults and get values for tolerance and map, if provided          %
tol = 0.10;
map = [];

for i = 1:length(flags)
    if strcmpi(flags{i},'tol')
        tol = values{i};
    elseif strcmpi(flags{i}, 'map')
        map = values{i};
    end
end

% Identify the data clusters accounting for noise of tol                  %


cond = [];
for i = 1:length(oc)
    cond = [cond; oc{i}];
end
cond = sort(cond);

unique_cond = [cond(diff(cond)>tol);cond(end)];

% Define a map if one is not supplied %
if isempty(map)
    map = sortrows(unique_cond);
end

data = cell(size(oc));

for i = 1:length(oc)
    opcon = oc{i};
    data{i} = NaN(size(opcon,1),1);
    for j = 1:length(opcon)
        data{i}(j) = findvec(map,opcon(j,:),tol);
    end
end

%%  Find the vector b inside the matrix a
function ind = findvec(a,b,tol)

b = repmat(b,size(a,1),1);
d = sum(abs(b-a),2);
ind = find(d <= size(a,2)*tol);
```

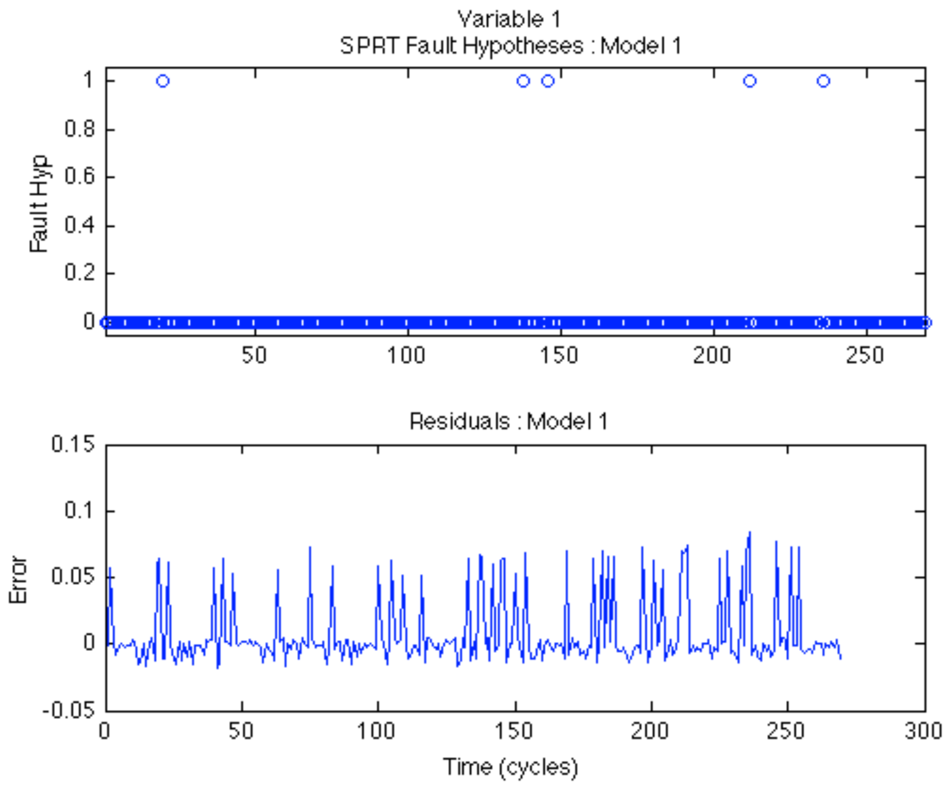
```
function [m s k sk] = timestats2(data,Nobs,varargin)

% TIMESTATS2 Calculate Time-Based Statistics for Multivariate Data
%       This function calculates basic time statistics for multivariate
%       data.
%
%       [Mean StdDev Kurtosis Skewness] = TIMESTATS2(data,N) calculates the
%       appropriate statistics across blocks of time of length N. The
%       requested time statistics are also plotted.
%
%       [Mean StdDev Kurt Skew] = TIMESTATS2(data,N,'flag',value ...)
%       calculates the appropriate time statistics using the user-supplied
%       properties specified in the 'flag'/value pairs. Accepted pairs are:
%               'plot' (True)   Logical indicator to display output plot.
%               'varnames' (numbers) Cell containing variable names in
%                               strings, for plotting purposes.
%

%   Jamie Coble
%   The University of Tennessee, Knoxville
%   Nuclear Engineering Department
%   Last Update:    3/12/2010
%   Copyright (c)
%   Based on timestats() by Michael Sharp

%% set defaults
warning('off','MATLAB:divideByZero')

if isempty(Nobs)
    error('prognostictoolbox:timestats2:missinginput','Missing Block Size');
end
N = floor(size(data,1)/Nobs);

ploton = 1;
varnames = cell(1,size(data,2));
for i = 1:size(data,2)
    varnames{i} = strcat('Var',num2str(i));
end

flags = [];
if ~isempty(varargin)
    [flags values] = getfvp(varargin);
end

for i = 1:length(flags)
    if strcmpi(flags{i},'plot')
        ploton = values{i};
    elseif strcmpi(flags{i},'varnames')
        varnames = values{i};
    else error('prognostictoolbox:timestats2:invalidflag','Invalid Flag')
    end
end

%% Calculate appropriate statistics


m = NaN(N,size(data,2));
s = NaN(N,size(data,2));
k = NaN(N,size(data,2));
sk = NaN(N,size(data,2));

for i = 1:N
    m(i,:) = mean(data(Nobs*(i-1)+1:Nobs*i,:));
    s(i,:) = std(data(Nobs*(i-1)+1:Nobs*i,:));
    k(i,:) = kurtosis(data(Nobs*(i-1)+1:Nobs*i,:));
    sk(i,:) = skewness(data(Nobs*(i-1)+1:Nobs*i,:));
end


%% Plot output


if ploton
    if N<50
        linetype = '-o';
    else linetype = '-';
    end
    if nargout == 0
        nplots = 4;
    else nplots = nargout;
    end
    res = cell(1,4); res{1} = m; res{2} = s; res{3} = k; res{4} = sk;
    ylabs = {'Mean','Std Dev','Kurt','Skew'};
    figure; hold on
    for i = 1:nplots
        subplot(nplots,1,i)
        plot(res{i},linetype)
        ylabel(ylabs{i})
    end
    xlabel(['Time Blocks (',num2str(Nobs),' obs per block)'])
    subplot(nplots,1,1);legend(varnames,'location','bestoutside')
    title('Data Statistics Over Time')
end
```

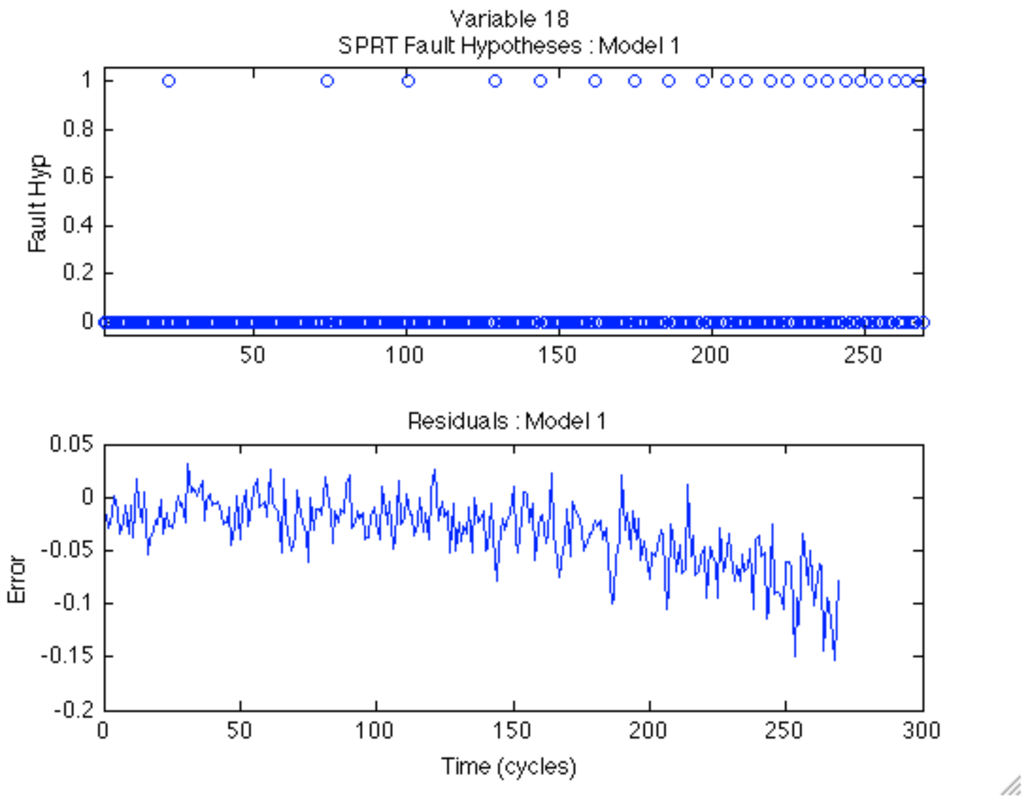
Appendix D: Process and Equipment Prognostics Toolbox Demo

The following code illustrates the use of the PEP toolbox in conjunction with the PEM toolbox for a full suite of monitoring, fault detection and prognostic routines. In this example, the algorithms are applied to the PHM challenge problem data described previously. Appropriate results are included within the code where appropriate to demonstrate the output of the PEP toolbox. Textual results are given in *italics* and plots are inserted following their generating code.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                           %
%                             PHM Software Demo                             %
%                                                                           %
%                              Jamie Coble                                  %
%                             PHM Conference                                %
%                                                                           %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%
clear

% Script to run through model development and optimization, system          %
% monitoring, fault detection, and prognostic model development with the   %
% PEM and PEP Toolboxes                                                     %

% System will be developed for the 2008 PHM Challenge Data, available at   %
% http://ti.arc.nasa.gov/project/prognostic-data-repository                 %

load PHMchalldata

%% Divide Data

% Assume first 15% of each data run is fault free for monitoring system    %
% development and optimization.                                             %

train = [];
for i = 1:260
    train = [train;trn{i}(1:floor(0.15*size(trn{i},1)),:)];
end

% Divide "fault free" data into training, test, and validation data        %
[x1 x2 x3 x4 x5 x6 x7 x8] = vensample(train,8);
training = [x1; x3; x5; x7];
testing = [x2;x6];
validation = [x4;x8];

%% Choose Monitoring Model Inputs

% Model inputs are chosen based on linear correlations between available   %
% variables.  Correlations with magnitude less than 0.3 are considered     %
% unusefulz; those with magnitude between 0.3 and 0.7 may have some        %
% predictive power, and those with magnitude greater than 0.7 are         %
% considered good predictors.                                             %

% The PEM function AAGROUP() divides data into groups for auto-associative%
% model development.  It uses a cut-off of 0.7 to identify appropriate    %
% groups from the data.  This function gives us two groups, as shown.     %
% Notice that some variables are members of both groups.                  %

Groups = aagroup(training)
Groups{1}
Groups{2}
gcplot(training,Groups)

% We can make two models to give the best predictive performance over all %
% variables.  The first model will contain 21 variables, and the second   %
% model will contain 8 variables.                                         %

train_group1 = training(:,Groups{1});
train_group2 = training(:,Groups{2});
test_group1 = testing(:,Groups{1});
test_group2 = testing(:,Groups{2});
val_group1 = validation(:,Groups{1});
val_group2 = validation(:,Groups{2});

% Model 1 %
model1 = initmodel('aakr',train_group1,'nmem',500);
model1 =
setmsa(model1,'plotresults',0,'fdetmethod','sprtn','variablenames',num2cell(G
roups{1}));
model1.architecture.bandwidth
model1 = optmodel(model1,test_group1,'error','bandwidth',[0.5 0.75 1.0 1.5]);
model1.architecture.bandwidth
model1 = modchar(model1,val_group1);
model1.attributes


% Model 2 %
model2 = initmodel('aakr',train_group2,'nmem',500);
model2 =
setmsa(model2,'plotresults',0,'fdetmethod','sprtn','variablenames',num2cell(G
roups{2}));
model2.architecture.bandwidth
model2 = optmodel(model2,test_group2,'error','bandwidth',[0.5 0.75 1.0 1.5]);
model2.architecture.bandwidth
model2 = modchar(model2,val_group2);
model2.attributes

%% Monitoring and Fault Detection
```

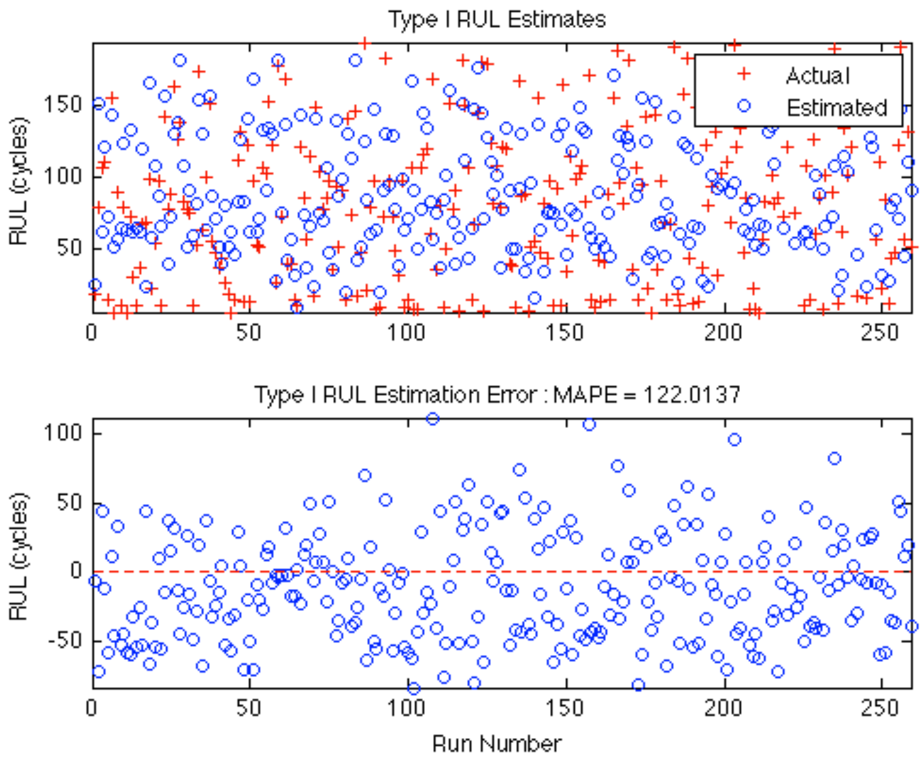
```
% Now we can use our models for system monitoring, residual generation and%
% fault detection.                                                          %

% Extract SPRT attributes for use in fault detection                        %
m1 = model1.attributes.error.mean;
v1 = model1.attributes.error.std.^2;
t1 = model1.attributes.sprttolerance;

m2 = model2.attributes.error.mean;
v2 = model2.attributes.error.std.^2;
t2 = model2.attributes.sprttolerance;

% Calculate model predictions, residuals, and fault hypotheses for each   %
% model                                                                     %

tic
for i = 1:260
  % Model 1 %
    pred1{i} = runmodel(model1,trn{i}(:,Groups{1}));
    res1{i} = pred1{i} - trn{i}(:,Groups{1});
    Fhyp1{i} = sprtn(m1,v1,res1{i},0.01,0.1,t1);


  % Model 2 %
    pred2{i} = runmodel(model2,trn{i}(:,Groups{2}));
    res2{i} = pred2{i} - trn{i}(:,Groups{2});
    Fhyp2{i} = sprtn(m2,v2,res2{i},0.01,0.1,t2);
end

% Let's look at the results for run #2                                      %
for j = 1:21
    figure
    subplot(2,1,1); plot(Fhyp1{2}(:,j),'o');
    ylabel('Fault Hyp');
    title({['Variable ' num2str(model1.attributes.variablenames(j))],.
        'SPRT Fault Hypotheses : Model 1'});
    axis([-inf inf -0.05 1.05])
    subplot(2,1,2);plot(res1{2}(:,j));
    xlabel('Time (cycles)');
    ylabel('Error');
    title('Residuals : Model 1');
end
```


Variable 1
SPRT Fault Hypotheses : Model 1
Fault Hyp
Residuals : Model 1
Error
Time (cycles)



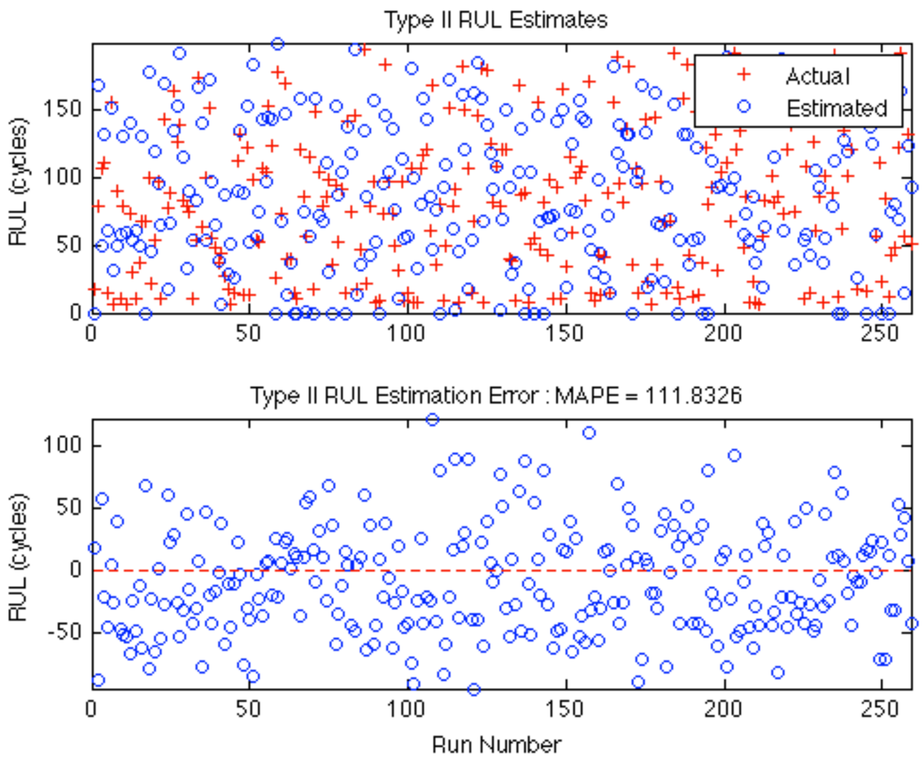
```
%% Prognostic Models - Type I Model

% Type I models are traditional time-to-failure models. The most common   %
% distribution for developing this kind of model is the Weibull           %
% distribution, which can model burn in, random failure, and wear out. The%
% Time of Failure for each failed case is the only data needed to develop %
% this type of model.                                                     %

TOF = zeros(length(trn),1);
for i = 1:length(trn)
    TOF(i) = length(trn{i});
end

typeI = initprog('typeI',TOF)

typeI =

            type: 'TypeI'
    distribution: 'weibull'
      parameters: [1x1 struct]
            data: [1x1 struct]

typeI.parameters
```

```
ans =

     beta: 4.3883
    theta: 225.6644

% The RUL of a new system is estimated based only on the amount of time   %
% that system has been running.                                            %

current_time = zeros(length(tst),1);
for i = 1:length(tst)
    current_time(i) = length(tst{i});
end
RUL_typeI = runprog(typeI,current_time);

MAPE_typeI = mean(abs(RUL - RUL_typeI)./RUL*100);

figure
subplot(2,1,1);hold on; plot(RUL,'r+'); plot(RUL_typeI,'bo');
hold off
legend('Actual','Estimated')
title('Type I RUL Estimates')
ylabel('RUL (cycles)')
axis([0 259 -inf inf])
subplot(2,1,2);hold on; plot(RUL-RUL_typeI,'bo');
plot(1:259,zeros(1,259),'r--'); hold off
xlabel('Run Number')
ylabel('RUL (cycles)')
title(['Type I RUL Estimation Error : MAPE = ' num2str(MAPE_typeI)])
axis([0 259 -inf inf])
```

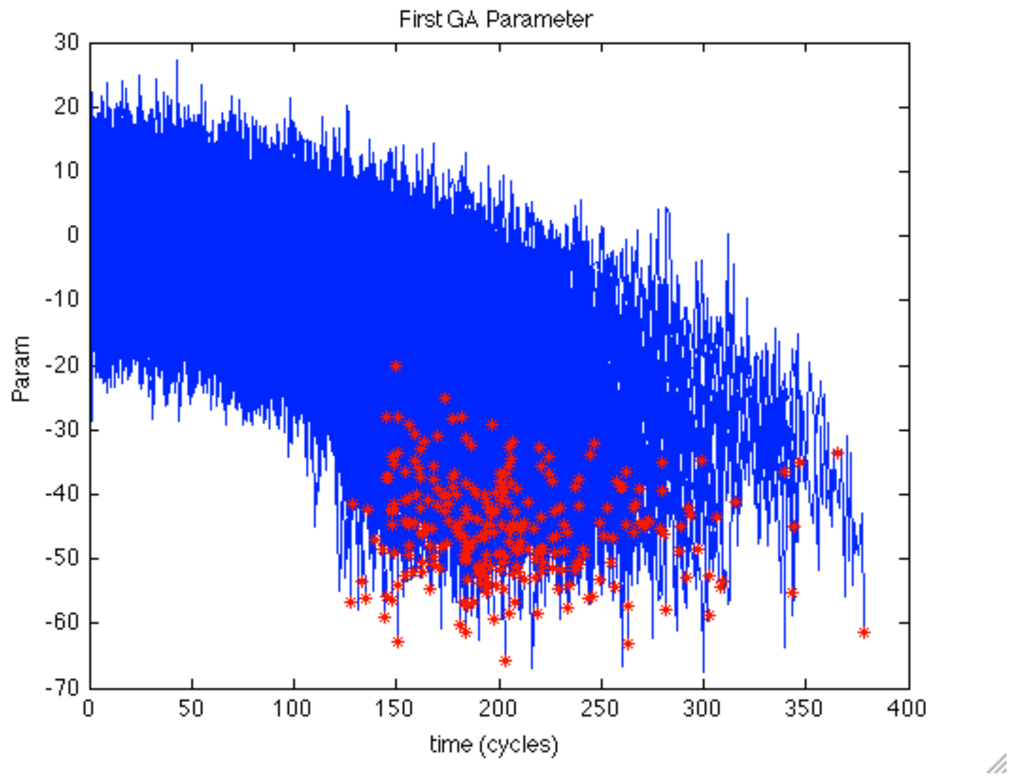


```
%% Prognostic Models - Type II Model

% Type II models consider the past and expected future operating          %
% conditions of a system when making prognostic estimates.  One example of%
% a Type II algorithm is the Markov Chain model. This model uses a        %
% transition probability matrix to simulate possible future operating     %
% conditions and relates these conditions to a degradation measure. The   %
% PHM Challenge Data has six distinct operating condtions, so it may be   %
% well suited to this type of model.                                      %

% Format historic operation condition progressions to be numbered 1 - 6.  %
% These numbers have no ordinal relationship.                             %

old_oc = cell(size(trn));
for i = 1:length(trn)
    old_oc{i} = trn{i}(:,1);
end

[new_oc map] = MCdata(old_oc);

typeII = initprog('MC',new_oc,'RULcon',0.5)

typeII =
          type: 'MC'
             Q: [6x6 double]
             u: [0.1577 0.1346 0.1231 0.1885 0.1385 0.2577]
             f: @(b,t)t*b
             b: [6x1 double]
     threshold: 100
          npop: 1000
        RULcon: 0.5000

typeII.Q

ans =

    0.1474    0.1520    0.1512    0.1521    0.1430    0.2544
    0.1440    0.1508    0.1502    0.1472    0.1524    0.2554
    0.1455    0.1466    0.1517    0.1468    0.1588    0.2506
    0.1543    0.1568    0.1479    0.1507    0.1415    0.2488
    0.1502    0.1551    0.1542    0.1465    0.1511    0.2431
    0.1536    0.1460    0.1518    0.1488    0.1500    0.2498

typeII.b

ans =

    0.6506
    0.3333
    0.4624
    0.5269
    0.7096
    0.2334

% Format test path operating conditions to the MC classes 1 - 6 using the %
% map identified previously.                                              %

test_oc = cell(1,259);
for i = 1:259
    test_oc{i} = tst{i}(:,1);
end
test_oc = MCdata(test_oc,'map',map);

% Estimate the 50% RUL for each test run.                                 %
RUL_typeII = NaN(length(tst),1);
for i = 1:length(tst)
    RUL_typeII(i) = runprog(typeII,test_oc{i});
end

MAPE_typeII = mean(abs(RUL - RUL_typeII)./RUL*100);

figure
subplot(2,1,1);hold on; plot(RUL,'r+'); plot(RUL_typeII,'bo');
hold off
```

```
legend('Actual','Estimated')
title('Type II RUL Estimates')
ylabel('RUL (cycles)')
axis([0 259 -inf inf])
subplot(2,1,2);hold on; plot(RUL-RUL_typeII,'bo');
plot(1:259,zeros(1,259),'r--'); hold off
xlabel('Run Number')
ylabel('RUL (cycles)')
title(['Type II RUL Estimation Error : MAPE = ' num2str(MAPE_typeII)])
axis([0 259 -inf inf])
```

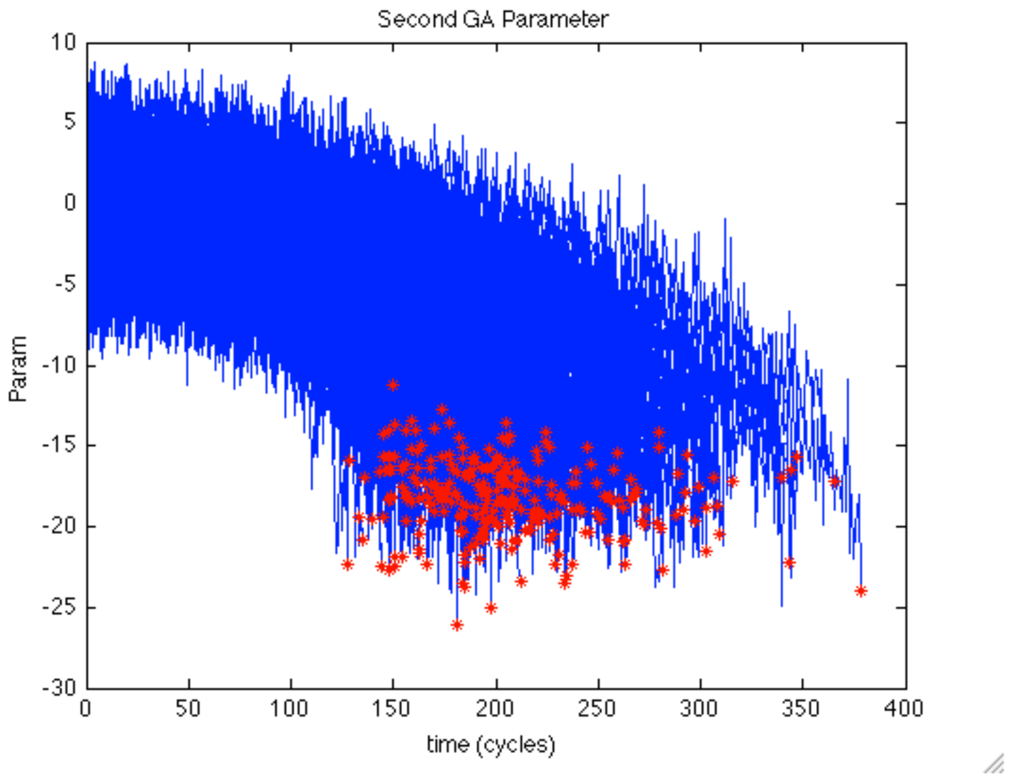


```
% We see that the results of this model are not very good. If we look back%
% at the transition probability matrix and linear fit, we see that we do  %
% not get much extra information from the markov chain formalism. The     %
% probability of transitioning between any two states is nearly equivalent%
% and the total time spent in any one operating condition does not seem to%
% give us much information about the degradation level.                   %

%% Prognostic Models - Type III Model

% Type III models consider the actual condition of the system, either     %
% measured or inferred from other measurements.  These condition          %
% measurements are fit to a parametric model which is then extrapolated to%
% a pre-defined critical failure threshold.                               %

% The first step is to identify an appropriate prognostic parameter.      %
% Monitoring model residuals are a natural choice for prognostic          %
% parameters because they naturally characterize how "far" the system is  %
% operating from normal behavior. For this example, we are looking for the%
% optimal linear combination of the 29 residuals (21 from model1 and 8    %
% from model2) based on a sum of the three suitability metrics -          %
% monotonicity, prognosability, and monotonicity.                         %

inputs = cell(size(trn));
for i = 1:length(trn)
    inputs{i} = [res1{i} res2{i}];
end

% For this demonstration, we will develop two prognostic parameters: the  %
% first will include all available residuals                              %
param1_ga = optparam(inputs,'inputs','all')
par1 = paramgen(param1_ga,inputs);
[m1 p1 t1] = ppmetrics(par1)

m1 =
    0.6415

p1 =
    0.8411

t1 =
    0.8140

figure; hold on
for i = 1:length(par1)
    plot(par1{i},'b')
    endtime(i) = length(par1{i});
    endval(i) = par1{i}(end);
end
plot(endtime,endval,'r*')
xlabel('time (cycles)')
ylabel('Param')
title('First GA Parameter')
```



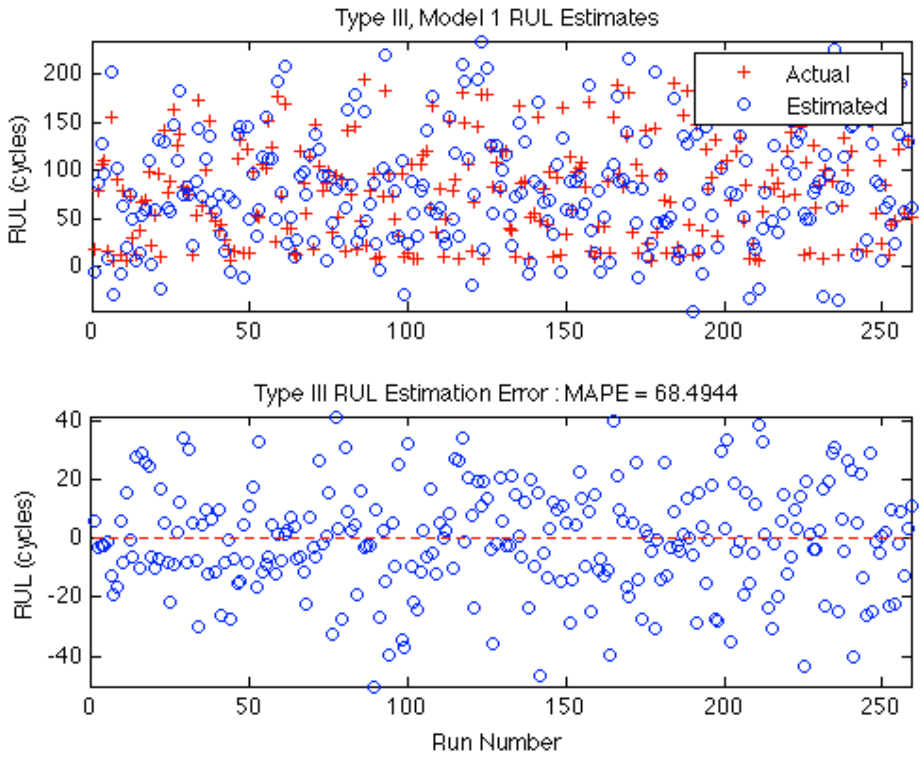
```
% The second parameter includes a subset of the residuals. Residuals are  %
% chosen for inclusion in the GA by calculating the fitness of each       %
% individual residual and including only those with total suitability over%
% 2.0                                                                     %

param2_ga = optparam(inputs,'inputs','subset','cutoff',2.0)
par2 = paramgen(param2_ga,inputs);
[m2 p2 t2] = ppmetrics(par2)

m2 =
    0.7014

p2 =
    0.8789

t2 =
    0.8956

figure; hold on
for i = 1:length(par2)
    plot(par2{i},'b')
    endtime(i) = length(par2{i});
    endval(i) = par2{i}(end);
```

```
end
plot(endtime,endval,'r*')
xlabel('time (cycles)')
ylabel('Param')
title('Second GA Parameter')
```

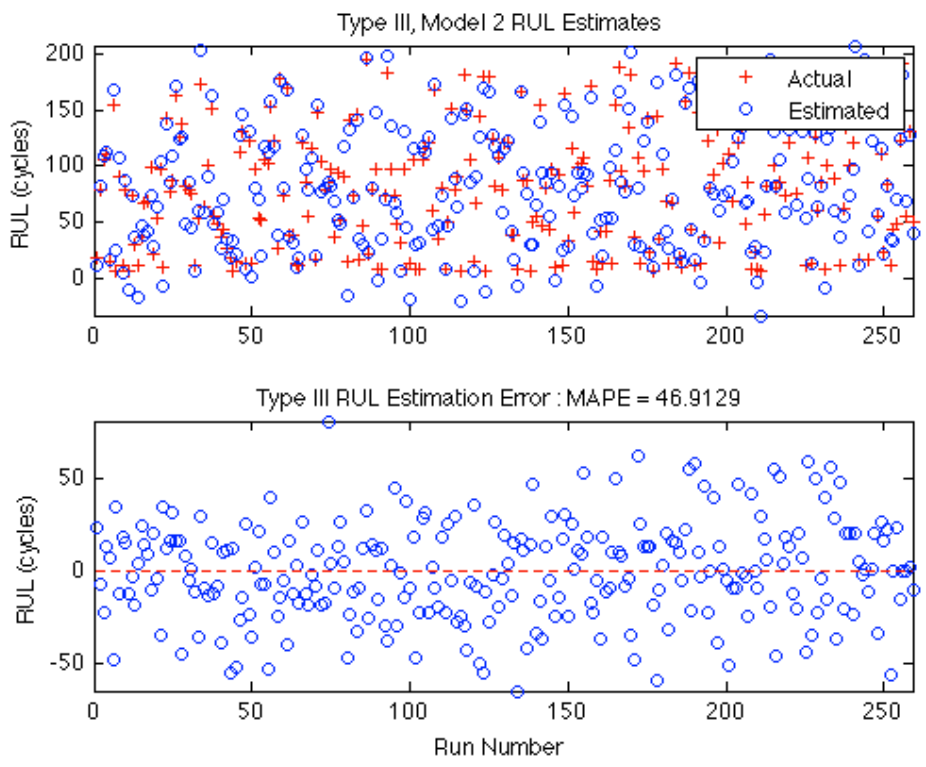


```
% A prognostic model is developed for each of the two parameters          %
typeIII_1 = initprog('gpm',par1,'fit',{@(x)x.^2 @(x)x @(x)1})
typeIII_2 = initprog('gpm',par2,'fit',{@(x)x.^2 @(x)x @(x)1})

% Monitoring system residuals for the test runs are obtained from the     %
% previously developed models                                             %
res1_tst = cell(size(tst));
res2_tst = cell(size(tst));
inputs_tst = cell(size(tst));
for i = 1:length(tst)
    res1_tst{i} = runmodel(model1,tst{i}(:,groups{1}));
    res2_tst{i} = runmodel(model2,tst{i}(:,groups{2}));
    inputs_tst{i} = [res1_tst{i} res2_tst{i}];
end

% Finally, the models are used to estimate the RUL of each test run with  %
% each model                                                              %
```

```
RUL_typeIII_1 = zeros(length(tst),1);
tst_par1 = paramgen(param1_ga,inputs_tst);
RUL_typeIII_2 = zeros(length(tst),1);
tst_par2 = paramgen(param2_ga,inputs_tst);
for i = 1:length(tst)
    RUL_typeIII_1(i) = runprog(typeIII_1,tst{i});
    RUL_typeIII_2(i) = runprog(typeIII_2,tst_par2{i});
end

MAPE_typeIII_1 = mean(abs(RUL - RUL_typeIII_1)./RUL*100);
MAPE_typeIII_2 = mean(abs(RUL - RUL_typeIII_2)./RUL*100);

figure
subplot(2,1,1);hold on; plot(RUL,'r+'); plot(RUL_typeIII_1,'bo');
hold off
legend('Actual','Estimated')
title('Type III, Model 1 RUL Estimates')
ylabel('RUL (cycles)')
axis([0 259 -inf inf])
subplot(2,1,2);hold on; plot(RUL-RUL_typeIII_2,'bo');
plot(1:259,zeros(1,259),'r--'); hold off
xlabel('Run Number')
ylabel('RUL (cycles)')
title(['Type III RUL Estimation Error : MAPE = ' num2str(MAPE_typeIII_1)])
axis([0 259 -inf inf])
```



```
figure
subplot(2,1,1);hold on; plot(RUL,'r+'); plot(RUL_typeIII_2,'bo');
hold off
legend('Actual','Estimated')
title('Type III, Model 2 RUL Estimates')
ylabel('RUL (cycles)')
axis([0 259 -inf inf])
subplot(2,1,2);hold on; plot(RUL-RUL_typeIII_1,'bo');
plot(1:259,zeros(1,259),'r--'); hold off
xlabel('Run Number')
ylabel('RUL (cycles)')
title(['Type III RUL Estimation Error : MAPE = ' num2str(MAPE_typeIII_2)])
axis([0 259 -inf inf])
```



VITA

Jamie Baalis Coble was born on April 4, 1983 in Florence, Alabama. She graduated as valedictorian from Science Hill High School in Johnson City, Tennessee. She then attended the University of Tennessee, Knoxville where she graduated Summa Cum Laude with a Bachelor of Science degree in both Nuclear Engineering and Mathematics and a minor in Engineering Communication and Performance in May, 2005. She also completed the University Honors Program with a Senior Project titled, "Investigating Neutron Spectra Changes in Deep Penetration Shielding Analyses." While an undergraduate, Jamie received several awards, including the Nuclear Engineering Outstanding Student Award in 2003, 2004, and 2005 and a Chancellor's Citation for Academic Performance. She was also honored as a Ned McWherter Scholar, a Bicentennial Scholar, and a Tennessee Valley Authority (TVA) scholar.

As an undergraduate, Jamie worked as a tutor in both the Math Tutorial Center and the Education Advancement program and as a teaching assistant in the Engineering Fundamentals program. In January, 2005, she began work with Dr. Mario Fontana investigating the effects of long-term station black outs on boiling water reactors, focusing particularly on the radionuclide release to the environment. In October, 2005, she began research with Dr. J. Wesley Hines investigating the effects of poor model construction on auto-associative model architectures for on-line monitoring systems. This work was incorporated into the final volume of a NUREG series for the U.S. Nuclear Regulatory Commission (NRC). Jamie received a Master of Science degree in Nuclear Engineering for this work. Her current research is in the area of empirical methods for system monitoring, fault detection and isolation, and prognostics, with a focus on automated methods for identifying appropriate prognostic parameters for use in individual-based prognosis. She was the first ever graduate of the Reliability and Maintainability Engineering MS program in August, 2009. She completed her PhD in Nuclear Engineering May, 2010.

As a graduate student, Jamie received the Hilton A. Smith Graduate Fellowship. She is a member of the engineering honors society Tau Beta Pi, the national honors society Omicron Delta Kappa, the National Society of Collegiate Scholars, the Institute for Electrical and Electronics Engineers Reliability Society, and the American Nuclear Society. In April, 2005 she passed the Fundamentals of Engineering exam to be recognized as an Engineering Intern.

Jamie spent four years working with Dr. Bob Kronick in the full service school initiative. During this time, she coordinated an afterschool program at Sarah Moore Greene Magnet Technology Academy which served 25 – 30 3rd – 5th graders every Monday – Thursday. The program included roughly 150 volunteers from the University of Tennessee.